Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Analysis of Neural Language Models for Artificial Data Generation

Author:
Thien P. Nguyen

Supervisor:
Julia Ive

Submitted in partial fulfillment of the requirements for the MSc degree in Computing (Machine Learning) of Imperial College London

May 2019

Abstract

Variational autoencoders (VAEs) have shown a lot of promise in other machine learning applications but are relatively unexplored in comparison to their potential as a language model. By applying them to Recurrent Neural Networks, we explore VAEs for the purposes of data represention on various text based datasets, representing a variety of sequence based problems. In this paper, we cover issues from both the sequentiality of recurrent models and also variational autoencoders, and discuss approaches that contemporary models have used to circumvent them.

To narrow the objective, we focus mostly on the Variational Autoregressive Decoder model (VADs; Du et al. (2018)) and compare it to historical predecessors. Quantitative and qualitative experiments conducted on the Amazon Reviews, OpenSubtitles, and Penn-Tree Bank datasets show how the models encompass information within the latent parameters, and to understand whether the models in their current state are capable of tackling the problem domain. We conclude that VADs are not only sensible for this approach, but find many more research directions that could lead the model to become even more capable for the purpose at hand.

Contents

Chapter 1

Introduction

Generative models have advanced greatly in the field of machine learning, most notably in the visual domain, but for NLP purposes it is often left unexplored.
Synthetic text-based data is highly sought after due to the lack of high-quality labelled datasets available, mainly as a result of the resources required to make such a dataset. Sequence-to-sequence (Sutskever et al. (2014)) models have shown a lot of promise in recent years, lending itself to domains such as translation, query answering, and summarisation. However, it lacks strong generative properties, limiting their capabilities for the purposes of generating artificial data. Recently, variational models have shown potential from both theoretical and practical perspectives (Kingma and Welling (2013)). We explore the capabilities of one of the recent advances in neural sequence modelling by Du et al. (2018), called the Variational Autoregressive Decoder (VAD). The nature of this paper is exploratory, and not necessarily to introduce novel contributions to the field. In this paper, we review historical implementations, the theoretical and emperical properties of the VAD, and evaluate their applicability for sequence generation of synthetic data using datasets by He and McAuley (2016), Lison and Tiedemann (2016), and Lison and Tiedemann (2016). We conclude with future steps towards reaching a solution to the problem domain.

Chapter 2

Literature Survey

To provide context into the literature survey, we assume a scenario where a client requests the use of our dataset. Granting the client access to the data could be inappropiate for a variety of reasons, ranging from the exposure of the privacy of people within the dataset, to the impracticalities of transferring large files to the client. The objective is to provide some alternative mechanism such that value can be deduced from the data, but the privacy of the users in our original dataset is maintained.
This can be accomplished by creating a language model that would encompass the lexical properties of our original dataset. This language model would train on our original dataset, and produce data that is semantically and lexically similar to our original data, but diverges enough such that it could potentially be seen as an entirely new and independent dataset. This model can be exposed to the client such that they could use it to generate responses to produce a somewhat similar dataset, without exposing the intricacies defining the original properties of the dataset.
The premise of this literature survey is to explore the core components necessary to construct our solution, and to also discuss approaches to the problem.

2.1 Text Generation

Text generation is a type of language modeling problem, which in itself is one of the core natural language processing problems. It is typically considered challenging as samples are discrete and results are non-differentiable (Kovalenko (2017a); Kovalenko (2017b)), as opposed to other types of data mediums, such as images and audio.
Traditionally, data itself could be generated via the use of data augmentation techniques. This is primarily the case for images; for instance, images can be indiscriminately flipped, cropped and transformed with no immediate consequence. These operations cannot trivially be extended to sequences of text as words are temporally and contexturally bound, where the omission of certain words in a sequence could dramatically change the semantic definition of the sequence. Even if these methods could be successfully applied, the resulting data would be relatively rudimentary in terms of their throughput, lexical diversity, and coherence.
With the advent of neural language models, text generation became more viable, and can now be seen in a variety of applications, ranging from machine translation (Sutskever

et al. (2014)), to email response generation (Kannan et al. (2016)), to document summarisation (Nallapati et al. (2016)).

2.2 Language Modelling

Dyer (2017) describes an unconditional language model as assigning a probability to a sequence of words, $w = (w_1, w_2, ..., w_{i-1})$. This probability can be decomposed using the chain rule:

$$p(w) = p(w_1) \times p(w_2|w_1) \times p(w_3|w_1, w_2) \times ... \times p(w_i|w_1, w_2, ..., w_{i-1}) \tag{2.1}$$

$$p(w) = \prod_{t=1}^{|w|} p(w_t|w_1, ..., w_{t-1}) \tag{2.2}$$

Traditionally, assigning words to probabilities may conflate syntactically spurious sentences[1] but it remains to be a useful method for representing texts in a sequence. For the purposes of this paper, we concentrate on conditional language modelling. A conditional language model assigns probabilities to sequences of words, $w = (w_1, w_2, ..., w_{i-1})$, given some conditioning variable, x.

$$p(w|x) = \prod_{t=1}^{|w|} p(w_t|x, w_1, ..., w_{t-1}) \tag{2.3}$$

There exists a variety of language models; we start with the n-gram, argued as being the most fundamental (Le (2018)). An n-gram is a chunk of items, where each item consists of n consecutive words in a sequential order. For instance, given the sentence "the quick brown fox . . . ", the respective n-grams are:

- unigrams: "the", "quick", "brown", "fox"
- bigrams: "the quick", "quick brown", "brown fox"
- trigrams: "the quick brown", "quick brown fox"
- 4-grams: "the quick brown fox"

The intuition of n-grams within the contexts of NLP is that relationships between words are encompassed in the n-gram item. For instance, by plotting a histogram of n-gram items on some corpus, you would be able to see how frequent certain words would be in proximity to other words.
Statistical inference can be applied on the frequency and distribution of the n-gram items, which could be used to predict the next word, using a soft probablistic classifier such as Naive Bayes, and thus making a language model.

[1] A case for this could be that predicting two verbs together is improbable but still technically possible - This is not naturally observed in the english language.

That being said, one of the caveats of n-grams is that it fails to capture sparsity; it is not viable to capture the relationship between words that are somewhat remotely related to each other. For example, increasing the sparse relationships between words can be captured by increasing the n-gram size, but this increases the odds of capturing an n-gram of zero probability. There exists mechanisms to bypass this issue (such as Add-One Smoothing, Backoff and Interpolation (Jurafsky and Martin (2019))), but using n-grams still remain infeasible as using such mechanisms does not scale to larger datasets.
Later on, modern language models began to revolve around the use of neural networks (Bengio et al. (2001)), which started with a multi-layer perceptron (MLP) that formed the premise of word prediction. The model would take as input a series of one hot values, representing words, and would predict another word in a similar one-hot fashion. The use of neural networks in language modelling is often called Neural Language Modelling, of NLM for short.
Neural Networks are non-linear computational metaphors of the brain that model arbitrary relationships between input and output vectors. A textbook architecture of a neural network revolves around the multi-layer perceptron. Note that the input and output vectors are of a fixed dimension. Neural Networks evaluate an input using forward propagation, to produce an output. Traditionally, neural networks are trained to produce optimal outputs via the use of backpropagation.

2.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a class of neural networks where the outputs are not necessarily restricted or discrete in dimension (as opposed to the MLP). RNNs operate over some variable-length vector, and produces an output of some other arbitarily sized variable-length vector. This circumvents a problem with using an MLP, where sentences are not typically fixed in length.
The architecture of RNNs make them favourable in NLP related problems as words in sentences are typically conditioned on previous words. A standard RNN language model (also known as a recurrent language model) predicts each word of a sentence conditioned on the previous word and an evolving hidden state.

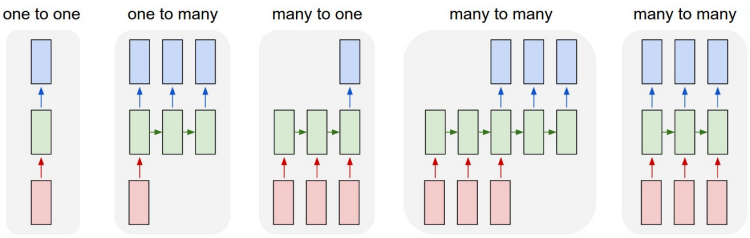

Figure 2.1: Rectangles represent vectors, with red being the input, blue the output, and green representing the state of the RNNs. Arrows represent functions. From left to right: (1) an MLP. (2,3,4,5) examples of different styles of recurrent neural networks, describing the different types of input and output combinations. (Diagram from Karpathy (2015)).

At each time step t, a simple RNN produces a hidden vector h_t, derived from the input

vector x_t and the previous state h_{t-1} in the function $\overrightarrow{h_t} = \overrightarrow{f_w}(h_{t-1}, x_t)$.[2]
The hidden vector is usually obtained by some affine transformation offset by a bias vector, described in more detail in equation 2.4. The same function and the same parameters are used at each time step t. RNNs, by design, are not computationally parallelisable. This hidden vector h_t is continously updated as it traverses through the RNN, learning the temporal relationships between the inputs.

$$\begin{aligned} h_t &= tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t &= W_{hy}h_t \end{aligned} \tag{2.4}$$

Equations for the RNN cell.

RNN cells can be stacked, making a deeper recurrent network, and can also be bi-directional. Bi-directionality explores the temporal relationships of the sequence in both directions (left to right, and vice versa.) This is made possible by stacking two recurrent cells on top of each other, and having one forward propagate in one direction, and another in the opposing direction.
RNNs are typically trained with backpropagation through time (see equation 2.5) The gradient flow $w \leftarrow w - \alpha(\delta L/\delta w)$ is computed at every timestep, meaning that every path from W to the loss $\mathcal{L}$ needs to be considered.

$$\frac{\delta L}{\delta w} = \sum_{j=0}^{T}\sum_{k=1}^{j} \frac{\delta L_j}{\delta y_j}\frac{\delta y_j}{\delta h_j}\Big(\prod_{t=k+1}^{j}\frac{\delta h_t}{\delta h_{t-1}}\Big)\frac{\delta h_k}{\delta w} \tag{2.5}$$

Typically, there are two caveats with training RNNs. Firstl, the product (in brown) within the loss function in Equation 2.5 causes both vanishing and exploding gradients, either of which make RNNs especially difficult to train. The magnitude of this issue increases as the sequences increase in length. This is compounded by the fact that the limited expressibility of the gradients means that relationships between timesteps earlier in the recurrent computation are harder to learn than timesteps further in the sequence. This is described as the long range dependency problem.
Additionally, it is also difficult for RNNs to establish relationships between potentially relevant non-contiguous inputs and outputs, especially in the case of longer sequences. There is not necessarily a clear indicator in the architecture facilitating this feature, which becomes a problem for translation (e.g. languages with a differing lexical order, such as English and German.)
That being said, RNNs remain appropiate for language modelling primarily for their ability to encapsulate temporally contextual relationships within sequences. The idea of using RNNs for language models can be realised in Section 2.5.1. This paper revolves around the use of recurrent language models in a generative manner.

[2]In terms of notation, an arrow over the equations describe the directionality of the model. It is assumed that in the absence of an arrow, sequences go from left to right.

2.3.1 LSTMs and GRUs

LSTMs and GRUs are types of recurrent cells, introduced to circumvent the issue of long range dependency problems, and gradient sentivities that RNN cells suffered.

$$
\begin{aligned}
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \qquad & u_t &= \sigma(W_u \cdot [h_{t-1}, x_t] + b_u) \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \qquad & r_t &= \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \\
g_t &= \sigma(W_g \cdot [h_{t-1}, x_t] + b_g) \qquad & c_t &= tanh(W w_{t-1} + U(r_t \odot h_{t-1})) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \qquad & h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot c_t \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
\tag{2.6}
$$

Figure 2.3: Equations for LSTM cells (left) and GRU cells (right).

LSTM (Long Short Term Memory; Hochreiter and Schmidhuber (1997)) are a type of recurrent cell that retains information based on the previous inputs through the introduction of gated architectures. These gated architectures regulate the flow of information coming to and from the cell, helping to mitigate the issue of gradient sentivity and improving long range dependencies. Gates of the cell are parameterised and therefore learnable. LSTM cells can be swapped in place with the RNN cells. It has been shown to have a considerably stronger performance for a variety of tasks compared to RNN cells.
GRUs (Gated Recurrent Units; Cho et al. (2014)) are functionally similar to LSTMs, but uses less parameters. Within the GRU architecture, a feature to retain the previous weights remain, but there exists a direct path to the input from the output, allowing a reduction in training time. Chung et al. (2014) suggests that GRUs were found to perform at least equivalently to LSTMs but show slight improvements on smaller datasets. As there are objectively less parameters for the GRU, it theoretically converges faster than the LSTM.

2.4 Autoencoders

Autoencoders (E. Rumelhart et al. (1986)) are a specialised form of neural networks where the model attempts to faithfully recreate the inputs at the output - forcing the model to learn the properties of the input data. Autoencoders typically have a layer in the model where its dimension is smaller than the input space, therefore representing a dimensionality reduction in the data (Autoencoders can be considered to be a non-linear representation of PCA). Autoencoders within the domain of NLP are popularised through their use in machine translation, word embeddings, and document clustering.
Autoencoders are composed of two different components, an encoder network α and a decoder network β, such that $\alpha : X \to Z$ and $\beta : Z \to X$. Note that Z is typically smaller in dimension than X, otherwise the model would be learning some form of the identity vector, and would trivially pass through the input. Quantifying the performance of the model is deduced by a reconstruction loss formula. This reconstruction loss compares the output of the decoder against the input of the encoder.

Although the model learns the distribution of the data, unlike PCA one cannot immediately interpret the principal components of the data. In fact, it is difficult to understand the structure of the latent variable, as is the case for understanding the layers of neural networks in general. Additionally, as there is no sampling mechanism within this model, effectively prohibiting any generative capabilities.

2.4.1 Variational Autoencoders

Variational Autoencoders (VAEs, Kingma and Welling (2013)) are an adaptation of regular autoencoders. As opposed to learning a fixed latent vector, VAEs force the encoder to learn parameters of a gaussian distribution - a mean μ and a covariance Σ, essentially replacing the encoder from the standard autoencoder with a learned posterior *recognition model*, $q_\phi(z|x)$. This component of the autoencoder parameterises an approximate posterior distribution over z, using a neural network conditioned on x. This consequently allows us to take advantage of recent advances in variational inference. During the decoder phase, a sample z is generated from a gaussian using the learned parameters $\mathcal{N}(\mu, \Sigma)$, which is then fed into the decoder network, generating our reconstructed prior $p_\theta(x|z)$. This sampling mechanism allows VAEs to have generative capabilities. With this, one can entirely avoid the encoder network, and augment the mean and variance vectors to sample outputs from the decoder.

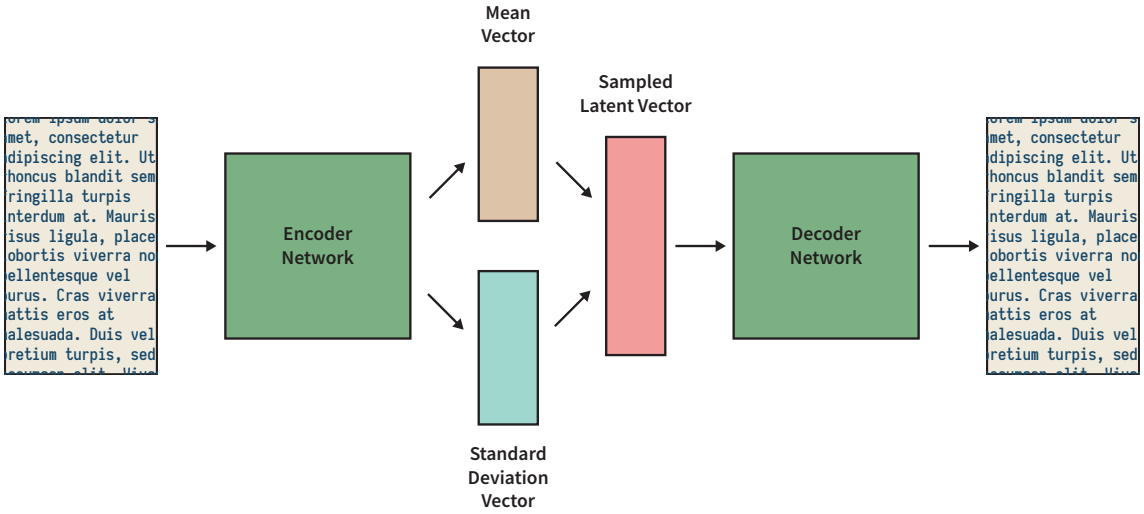

Figure 2.4: An abstracted model architecture for a variational autoencoder, which takes as input some text, and it's predicted output being the same text as the input.

For this to be possible, the latent space would learn to represent a non-standard gaussian, which encodes the properties of the inputs.
If VAEs were trained with a standard autoencoder's reconstruction objective, then it would learn to encode its inputs determinsitically by making variances in the *recognition model*, $q(\overrightarrow{z}|x)$ vanishingly small (Raiko et al. (2014)), such that when measuring the influence of the gaussian parameters against a standard gaussian, the difference would be obsolete. This is known as a vanishing KL[3] (Fu* et al. (2019)).

[3]KL stands for Kullback-Leibler, the authors behind the KL Divergence (Equation 2.7.)

$$D_{KL}(P||Q) = \sum_{x \subset X} P(x) \cdot log(\frac{P(x)}{Q(x)}) \tag{2.7}$$

Instead, the loss function for VAEs is composed of two components; a reconstruction loss that involves an expectation of the output; and a KL Divergence (see Equation 2.7), which helps encourage the learned posterior $q_\phi(z|x)$ to be similar to the true posterior distribution $p(z|x)$. The goal of this loss function would be to find the variational parameters that minimise the divergence, but This would typically be mathematically intractable, as $p(x)$ appears in the evidence. However, the equations can be rewritten, seperating the intractable components from the tractable.

$$\mathcal{L}(\theta, \phi, x, z) = \mathbb{E}_{q\phi(z|x)}[log\, p_\theta(x|z)] - D_{KL}(q_\phi(z|x)\,||\,p(z)) \leq log(p(x)) \tag{2.8}$$

KL divergence is always greater than or equal to zero, by definition of Jensen's inequality. This means that minimising the KL Divergence is equivalent to maximising the ELBO (Evidence Lower Bound). This loss mechanism (Equation 2.8) us to approximate the posterior inference. This objective provides a tractable lowerbound on the true likelihood of the data, and forces the model to encode some information on the data into the gaussian parameters.
Note that the ELBO is not necessarily a loss function. To observe how well the model has learned to model properties into the gaussian, we observe the KL divergence. Typically, the larger the better.

Reparameterisation Trick

$$z = \mathcal{N}(\mu, \Sigma) \equiv \mu + \Sigma \cdot \epsilon \tag{2.9}$$

If z was generated in exactly the method described above, calculating gradients through backpropagation would be impossible as gradients would have to eventually push through a randomly sampled latent variable. The gaussian reparameterisation trick (Kingma and Welling (2013)) subverts this issue through the rearrangement of the gaussian parameters. Instead of sampling z through a gaussian $\mathcal{N}(\mu, \Sigma)$, z can be redefined with a summation of μ and some gaussian noise $\epsilon \sim \mathcal{N}(0, 1)$ applied to Σ (See Equation 2.9). This rearrangement allows for derivatives to be calculated with respect to the parameters that are needed for learning as derivatives with respect to the relevant components would cancel out ϵ (thus making learning possible).

2.4.2 Conditional Variational Autoencoders

In a VAE, the decoder cannot typically produce outputs of a particular condition on demand - the description for the VAE matches the definition of semi-supervised learning. CVAEs add to the VAE to include conditioning on another description of the data, a conditioning descriptor y.
During training time, a condition (represented by some arbitrary vector) is fed at the same time to the encoder and decoder. After training, to generate an output that depends on y we feed that datum to the decoder along with a random point in the latent space sampled from a standard normal distribution.

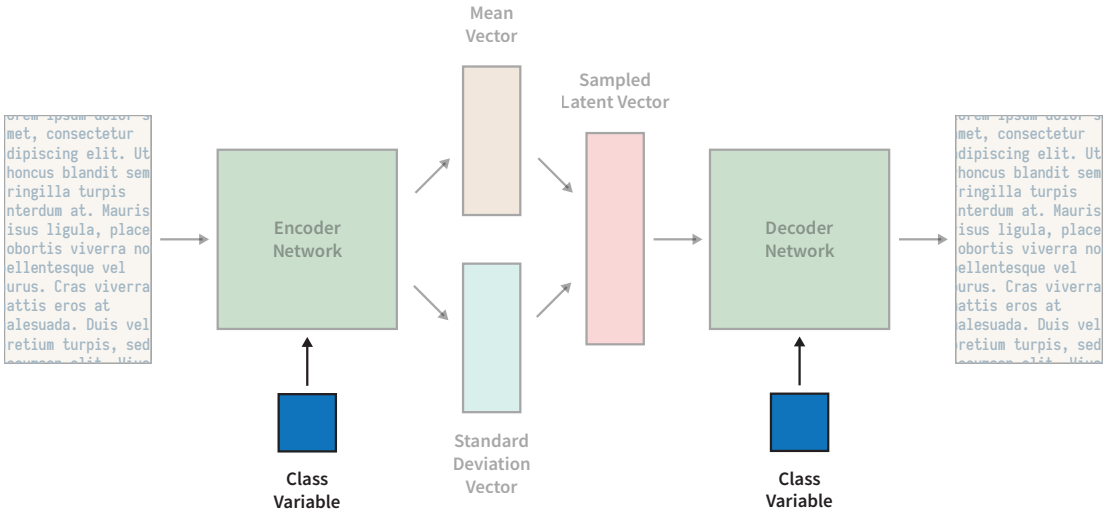

Figure 2.5: A model architecture for a CVAE, which includes the label being fed into the encoder and decoder networks.

Samples can be generated from the conditional distribution $p(x|y)$. By changing the value of y, we can get corresponding samples $x \sim p(x|y)$. The system no longer relies on the latent space to encode what output is necessary; instead the latent space encodes other information that can distinguish itself based on the differing y values.
In terms of implementation, it happens to be more feasible to concatenate the conditioning variable to the input prior to feeding them into the encoder. In the event of inference (where samples are generated independently of the encoder), one can concatenate y with some arbitrary conditional vector z into the decoder to generate outputs. This keeps the model from needing to adjust its architectural characteristics in order to accomadate the conditioning variables.

2.5 Related Work

2.5.1 Sequence To Sequence

Sequence to Sequence, (Seq2Seq, Sutskever et al. (2014)) is a type of neural network model that models relationships between sequences. Seq2Seq comprises of two components, encoders and decoders (in a similar fashion to autoencoders), both of which are represented with RNN models (although as explained earlier, can be replaced with LSTM or GRU cells). Defining Seq2Seq as a language model, it takes a pair of sequential data, which within the contexts of text generation, would be query and response sequences. The model would take in the query sequence, and attempt to recreate the response. The query sequence would go through the encoder model. The last hidden state value of the encoder is used as the context variable, which encodes the characteristics of the input sequence. This is used as the first hidden vector value for the decoder, which would then predict the response sequence. This sequence can be visualised in Figure 2.6. For the purposes of this paper, we describe this as the standard recurrent language model.
Seq2Seq models have shown to be effective in reponse generation, but is not without its

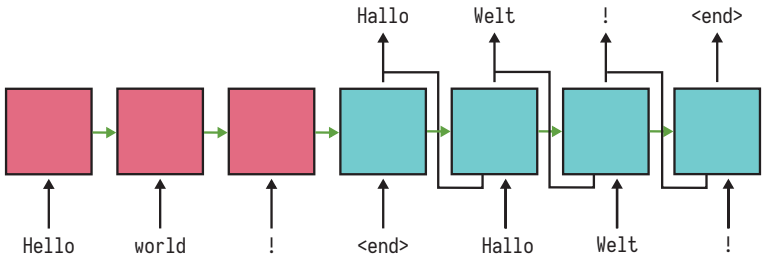

Figure 2.6: An abstracted model of the Seq2Seq architecture, where the encoder (pink) takes in the input sequence, and the decoder (blue) shows the output sequence. The encoder outputs are effectively ignored.

own set of problems. For instance, the context variable itself is considered to be the bottleneck as it has to encompass the properties of the entire input sequence in a single vector. As Seq2Seq is based on RNNs, they are also subject to vanishing and exploding gradients (Section 2.3), and that it becomes increasingly difficult to retain information about earlier parts of the sequence as opposed to recent parts of the sequence in the encoder stages as a consequence. A popular approach to improving long term dependencies exists however, with the introduction of attention mechanisms, discussed in Section 2.7.
Furthermore, the responses produced from a Seq2Seq model naturally lacks lexical diversity and richness (Serban et al. (2016), Zhao et al. (2017), Jiang and de Rijke (2018)). There tends to be many causes, but one reason (which we explore in the paper) is due to the lack of statistical inferencing. The model itself lacks the possibilty of presenting a variety of lexical responses in the decoder, in a manner that a VAE could through the sampling of a latent vector.

Attention Mechanism

In addition to the context vector, the decoder can also retrieve non-contiguous relationships on the encoder values using attention (Bahdanau et al. (2014)). This mechanism, when applied to Seq2Seq, looks at all of the hidden states from the encoder outputs, and generates relationships between these outputs against the decoder output. This allows the decoder network to "attend" to different parts of the source at each step of the output generation, learning relevant non-contiguous relationships between the query a nd response sequences. Attention mechanisms would work in tandem with the context vector provided from the encoder, helping the model with long-range dependency problems.
Attention mechanisms were found to perform particularly well in machine translation problems where languages which are arbitarily aligned, and has eventually led to the creation of transformer models (Vaswani et al. (2017)), avoiding the use of recurrent networks altogether.

2.5.2 Variational Sequence To Sequence

Instead of passing through the last hidden state drom the encoder to the decoder, it is possible to encode the context variable produced from the encoder in the form of gaussian parameters in a similar fashion to VAEs. This is known as a variational sequence to

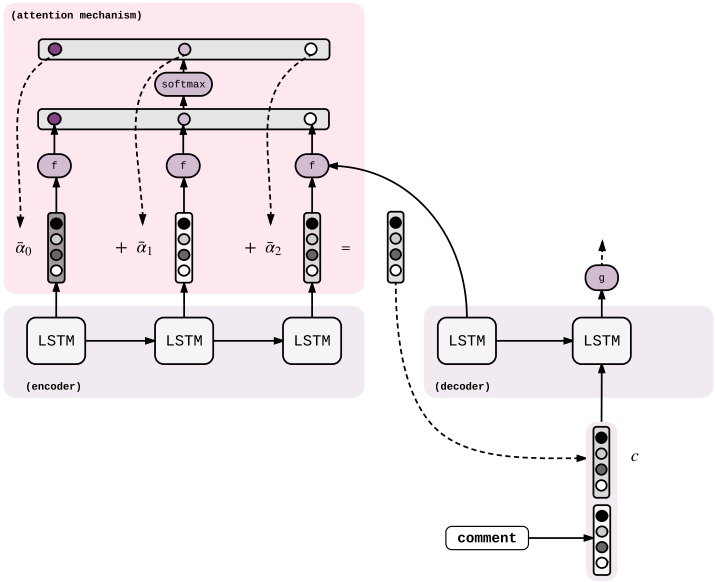

Figure 2.7: An abstracted diagram of the attention mechanism, applied to a Seq2Seq model. (Diagram from Genthial (2017))

sequence network (Bowman et al. (2015)), but for the remainder of this paper we refer to them as Bowman models. The intention of this model is to learn the high level global syntactic features of the input data into the latent vector. This is such that when inference occurs, the model could generate unconditioned sequences which looks like the input data. This is typically described as representation learning.
The encoder RNN takes the input sequence, and produces a hidden vector. This is used to encode the parameters of the gaussian, which define the latent vector. During the decoder phase, a sample latent vector is produced (in a similar manner to VAEs), which is then fed as the hidden vector for the decoder RNN, which also takes the input variables and attempts to recreate the input in an autoencoder fashion.

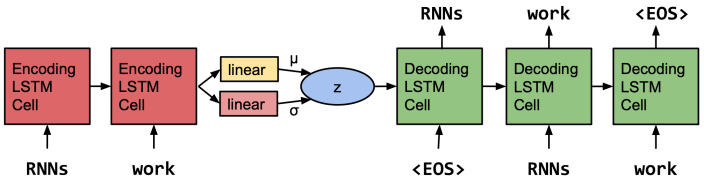

Figure 2.8: The core structure of the VAE language model - words are represented as embedded vectors (Diagram from Bowman et al. (2015)).

The model is trained in a similar fashion to VAEs where the overall loss is composed of a reconstruction loss and a KL divergence (see Equation 2.8). However, it also utilises additional mechanisms to force the language model to encode features into the gaussian parameters. We explain these features in further detail in Section 3.2.
As described above, it is not necessarily designed for conditioned discourse generation. Even if it were modified for that specific purpose, the responses would be generated from the same latent variable, restricting its variability in responses. (Zhao et al. (2017); Du

et al. (2018)) With that being said, it remains important for discussion as it handles optimisation challenges in a manner which we could leverage later on for our model of choice.

2.5.3 Conditional Variational Sequence to Sequence

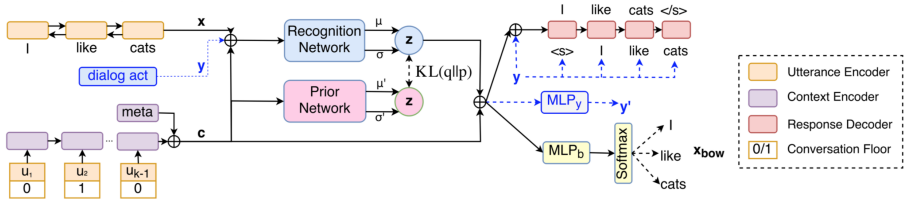

Figure 2.9: The structure of the CVAE language model. (Diagram from Zhao et al. (2017)).

Conditional Variational Sequence to Sequence (CVAE Seq2Seq; Zhao et al. (2017)) is an extension of Bowman's model for the purposes of discourse generation. It includes a plethora of additional components, including attention mechanisms to make discourse generation more viable. The model takes as input some query sequence as the condition, and attempts to generate a response sequence.
Leveraging Bayesian inference, the CVAE defines the conditional distribution $p(x, z|c) = p(x|z, c) \cdot p(z|c)$, where z represents a gaussian latent variable, and c corresponds to the context vector, produced from the attention mechanism. The model's goal was to leverage neural networks (parameterised by θ) to approximate $p(x|z, c)$, which in a bayesian setting is described as the likelihood and $p(z|c)$ being the prior.
In a Bayesian setting, the prior represents beliefs on some quantity before some additional information (in the form of evidence) is taken into account. CVAE uses prior knowledge in the greedy decoder to help generate diversity in responses. The generative process for x, seen in Figure 2.9 is as follows:

- Sample a latent variable z from the prior network $p_\theta(z|c)$.
- Generate x through the response deocder $p_\theta(x|z, c)$.

The model is trained to maximise the conditional likelihood of x, given c. This involves the intractable marginalisation over z, but this has been discussed to be trainable using the ELBO loss with Formula 2.8.
Once a latent variable is generated (notice that it is the case that the latent z is generated once during the decoder phase), it is fed to the decoder model to predict words in x sequentially. This diverges from Bowman's model, where the words are pushed in one go. One of the paper's novel contribution is an auxiliary function described as a bag of words loss. Within the NLP domain, a bag of word refers to a set of unique words. The model's loss function revolves around using the latent variable z within its model to predict the set of output words (in no particular order) as a multi-class one-hot value. The model we discuss in the next chapter will go into the bag of words loss in more detail.

Chapter 3

Model

In this chapter we focus on an approach called the Variational Autoregressive Decoder (VAD). The model itself is a combination of the components and models described in the literature review (Chapter 2). To better understand the capabilities of the VAD, we subject it to a series of datasets to understand how it works and its performance against them. The VAD has been shown to better encapsulate a distribution of the data within the latent parameters than the CVAE Seq2Seq, and retains a higher KL ratio respective to its overall loss function. This model is used as the candidate mechanism for encapsulating the dataset (for our problem domain), which could be sent to clients.

3.1 Variational Autoregressive Decoders

Variational Autoregressive Decoders (VADs; Du et al. (2018)) extends CVAE Seq2Seq by introducing multiple latent variables into the autoregressive Decoder at different time-steps. This allows the decoder to produce a multimodal distribution of text sequences, allowing a greater variety of responses to be produced than what is possible with the CVAE Seq2Seq.

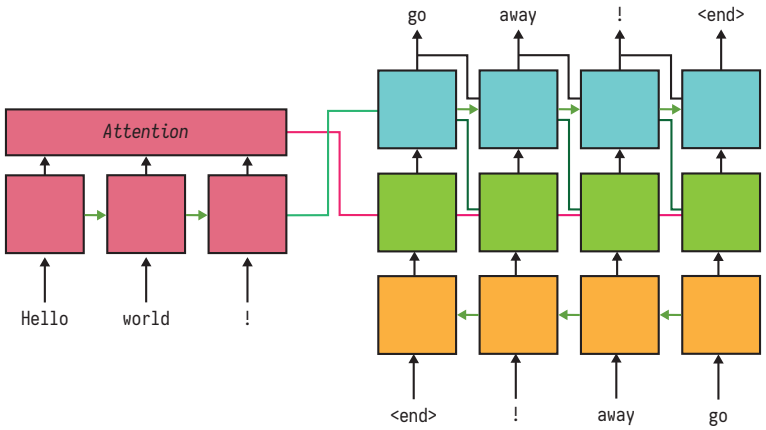

Figure 3.1: An abstracted diagram of the VAD model, where the encoder (pink) takes in the input sequence, orange represents the backwards model, green represents the latent context generation, and the decoder (blue) shows the output sequence.

VADs use the vanilla Seq2Seq architecture as the base (Section 2.5.1) taking as input, sequences $x = \{x_1, x_2, ..., x_n\}$, and $y = \{y_1, y_2, ..., y_m\}$ representing the query and response sequences respectively. Both the encoder and decoder use GRUs, with the encoder utilising a bidirectional GRU, and the decoder being unidirectional. For each timestep t, each GRU in the decoder network is encoded with hidden state h_t^d. The components of the model are broken down into their subsections below.

3.1.1 Architecture

Encoder

$$\begin{aligned} \overrightarrow{h_t^e} &= \overrightarrow{GRU}(x_t, \overrightarrow{h_{t-1}^e}) \\ \overleftarrow{h_t^e} &= \overleftarrow{GRU}(x_t, \overleftarrow{h_{t+1}^e}) \end{aligned} \tag{3.1}$$

The encoder works in a similar fashion to the encoder described in the Seq2Seq network. The only difference being that the encoder described in the implementation leverages bi-directionality. Similarly to the Seq2Seq and other recurrent language models, the last hidden vector is passed through as the initial hidden vector for the decoder model.

Backwards and Attention

$$\overleftarrow{h_t^d} = \overleftarrow{GRU}(y_{t+1}, \overleftarrow{h_{t+1}^d}) \qquad \begin{aligned} \alpha_{s,t} &= f_{attention}([h_d^e, h_{t-1}^d]) \\ c_t &= \sum_{s=1}^{m} \alpha_{s,t} h_s^e \end{aligned} \tag{3.2}$$

Figure 3.2: Equations for Backwards (left) and Attention (right).

The backwards RNN (only activated during training) is used to feed additional contextual information to the inference model. During training, the backwards RNN takes as input the training outputs y and outputs hidden vectors in a reverse, sequential manner. The VAD leverages the attention mechanism in a similar fashion to Seq2Seq; it takes in the input sequence, and is contextualised by the decoder's previous hidden output at each time step.

Decoder

The decoder component of the model comprises multiple sub-components, each of which are described in detail below. The sampling mechanism is within this decoder, which differs from other variational approaches - Earlier approaches sample a latent variable outside the decoder phase.

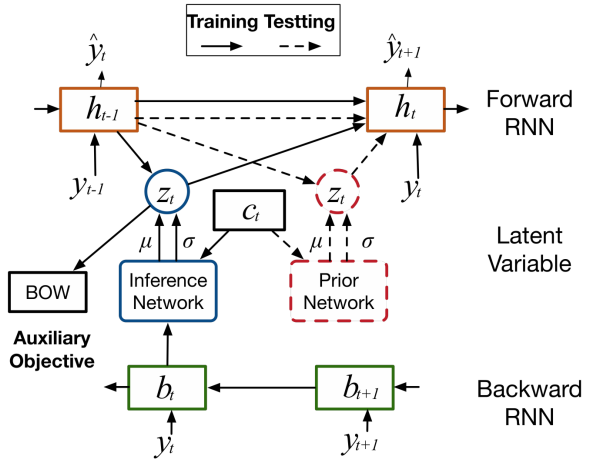

Figure 3.3: A High level diagram of the decoding component of the VAD. (Diagram from Du et al. (2018))

$$
\begin{aligned}
[\mu^i, \sigma^i] &= f_{infer}([\overrightarrow{h^d_{t-1}}, c_t, \overleftarrow{h^d_t}]) & [\mu^p, \sigma^p] &= f_{prior}([\overrightarrow{h^d_{t-1}}, c_t]) \\
q_\theta(z_t|\boldsymbol{y}, \boldsymbol{x}) &= \mathcal{N}(\mu^i, \sigma^i) & p_\phi(z_t|\boldsymbol{y}_{<t}, \boldsymbol{x}) &= \mathcal{N}(\mu^p, \sigma^p)
\end{aligned}
\tag{3.3}
$$

Figure 3.4: Equations for Inference (left) and Prior (right) models.

Inference and Prior Models

The inference model attempts to learn a conditional posterior distribution z_t using the backward output at that timestep $\overleftarrow{h^d_{t-1}}$, the attention context c_t, and the decoder outputs $\overrightarrow{h_{t-1}}$. z_t is created using the gaussian reparameterisation trick described in Section 2.4.1. This component stems away from Zhao et al. (2017), as $\boldsymbol{z}$ is now decomposed into sequential variables $\boldsymbol{z} = \{z_1, ..., z_t\}$, which are generated at each time step of the decoder phase. Du et al. (2018) suggests that this conditioning allows the latent variables to be guided for long-term generation.
The prior network is restricted to using observable variables during testing of the model to generate z_t using only observable information c_t and h^d_{t-1}; i.e. the inference model is not used to generate z_t during testing. It is designed in a similar fashion to the decoder network where the input variables are concatenated together, which can be seen in Equation 3.4.

Forward RNN

$$
\begin{aligned}
\overrightarrow{h^d_t} &= \overrightarrow{GRU}([y_{t-1}, c_t, z_t], \overrightarrow{h^d_{t-1}}) \\
p_\phi(y|\boldsymbol{y}_{<t}, \boldsymbol{z}_t, \boldsymbol{x}) &= f_{output}([\overrightarrow{h^d_t}, c_t])
\end{aligned}
\tag{3.4}
$$

The decoder then takes in z_t, alongside the previous output y_{t-1} and the context vector produced from the attention mechanism c_t to produce the response y_t.

3.1.2 Auxillary Objective

One of the novel contributions of the model is the use of a temporally contextual auxiliary objective that uses the sampled latent vector z_t to predict the sequential bag of words (SBOW) of the response sequences $\boldsymbol{y}_{bow}(t+1, T)$. Let $f = MLP_b(z_t) \in \mathcal{R}^V$ where V be the vocabulary size, then the bag of words can be deduced using the following equation:

$$log[p_\xi(y_{bow(t+1,T)}|z_{t:T})] = log\frac{e^{f_{z_t}}}{\sum_j^V e^{f_j}} \tag{3.5}$$

The intuition is that this auxiliary objective helps to improve the KL divergence, as the latent variables would capture the information from a different perspective (Du et al. (2018)).

3.1.3 Learning Mechanism

VADs uses the ELBO (Equation 2.8)[1] and a weighted negative log likelihood loss of the auxillary objective, controlled by α. This loss is computed at each timestep, and is then summed at the end of the sequence generation during the decoder stage. During training and evaluation, we sample both the inference and prior models to facilitate measuring the loss.

$$\begin{aligned}\mathcal{L} &= \sum_t [\mathcal{L}_{ELBO}(t) + \alpha\mathcal{L}_{AUX}(t)] \\ \mathcal{L} &= \sum_t [(\mathcal{L}_{LL}(t) - \mathcal{L}_{KL}(t)) + \alpha\mathcal{L}_{AUX}(t)]\end{aligned} \tag{3.6}$$

During implementation, it was found that the auxiliary loss weight α typically initially increases the KL divergence, but eventually produces a diminishing effect to both the auxiliary loss and the KL divergence. Attempts to contact the original authors of the VAD paper were to no avail, and is left for future work. Presently, we set $\alpha = 1$.

3.2 Optimisation Challenges

In addition to the fact that recurrent networks themselves are difficult to train (see Section 2.3), it may occasionally be the case that with variational models, the sampled latent variable z is often ignored during training. Ideally, a model that encodes useful information in z will have a non-zero KL divergence term in addition to a relatively small negative log likelihood loss.

Due to the sensitivity of the latent component, early implementations of the VAD would often ignore z and go after the low hanging fruit during the learning process of the decoder. z would encorporate a lack of relevant information and is essentially ignored. That

[1] As the KL divergence no longer compares the latent variable against a standard gaussian, we derive the equation to allow gradient propagation between two arbitrary distribution. This can be seen in the Appendix (Section 8.1).

is not to say that the model does not learn - it would rather allocate more learning weight to the information from other inputs such as the attention mechanism and the encoder's hidden value.
The absence of learning in z results in a KL collapse (or Vanishing KL), such that the KL loss reaches zero, indicating that the posterior distribution of z has "collapsed" under the prior. This effectively renders z to be ignored, and thus the model is then effectively equivalent to an RNN language model. (Bowman et al. (2015)) Although we explain one of the mechanisms used to prevent KL collapse with the auxiliary objective (see Section 3.1.2), we will also discuss some additional approaches to overcome learning difficulties, primarily around z, but also for the model overall.

3.2.1 KL Annealing

KL Annealing is a simple approach by Bowman et al. (2015), it involves adding a variable weight to the KL term in the cost function during training. Initially, the weight is zero, forcing the model to encode as much information into the latent variable as it can. The weight is then gradually increased, eventually reaching 1. At this stage the weighted cost function is equivalent to the ELBO loss. This could be thought of as annealing from a regular autoencoder towards a variational one.
That being said, Bowman et al. exposes a variety of additional hyperparameters including the step size, the total number of steps, the step rate (for instance, a linear progression or a logistic). For the sake of simplicity, we explore a linear approach with a fixed linear progression.

3.2.2 Word Dropout

Also proposed by Bowman et al. (2015), Word Dropout involves weakening the decoder by removing some conditioning information during training. This is done by randomly replacing a fraction of the conditioned-on word tokens with a generic unknown token UNK, which forces the model to depend on the latent variable z for prediction. This works in a similar fashion to the standard dropout (Srivastava et al. (2014)) where connections are dropped between layers of a network, but is instead applied to the input data of a recurrent cell. We do not explore this technique further for the VAD as a case by Bowman et al. (2015) suggests that it could distrupt the performance of the model when the drop rate is suffiently high.

Chapter 4

Experimental Setup

To explore the performance of the VAD, we benchmark its performance against two other models: a baseline model, and the Bowman model (see Section 2.5.2). The Seq2Seq model and the Bowman models were chosen for comparision not necesssarily to compare state-of-the-art performance but to verify the correctness of the implementations of our VAD model. We do not implement the CVAE Seq2Seq model as it is not the focus of this ISO.

The baseline model represents a vanilla encoder-decoder Seq2Seq network (Sutskever et al. (2014); Section 2.5.1) that uses the same architectural base as the VAD but only minimises the standard negative log likelihood loss (i.e. the KL divergence computations are not computed). We use this approach since KL collapse on variational models is effectively equivalent to a recurrent language model.

As stated earlier, the Bowman model is primarily designed for language modeling, whereas the VAD is designed for response generation. With this prior knowledge it was expected that the VAE Seq2seq model would perform worse than the VAD model for all tasks, but it remains important to understand from a variational standpoint, what a KL baseline should be.

Weight Initialisation

Typically, we intend to have the weights asymmetricly generated; i.e. have random starting positions for gradient descent, which would enable us to have a better chance of finding the minimum as opposed to having starting points that would somewhat correlate with each other.

Glorot and Bengio (2010) proposed to initialise the weights based on a gaussian $Var(w) = \mathcal{N}(0, \frac{2}{|n_{in}|+|n_{out}|})$ where w represents a layer in a network, and $|n_{in}|$ represents the number of neurons feeding into it, and similarly so for $|n_{out}|$. Note this descript uses a gaussian distribution, but this could also be uniform. This method has shown a lot of success for sigmoidal and tanh based activation functions. We initialise the weights for all models with this approach.

Teacher Forcing

Teacher Forcing (Williams and Zipser (1989)) is a concept of using real target outputs for the next input in the decoder sequence of RNNs as opposed to using the decoder's prediction. This causes the model to converge faster, but may exhibit instability when the trained network is exploited. Typically, this can be controlled with a probability p such that the decoder sequence has a p chance of using teacher forcing during training.
With teacher forced decoders, outputs that read with coherent grammer can be observed, but the generated responses can wander far from the correct response; i.e. with some $p < 1$, some outputs from the decoder at some timestep would be correct, but the others would have a chance to completely change the semantics of the output. As the purposes of this paper is to explore artificial texts, we set the VAD and Seq2Seq teaching probabilities $p = 1$, but not the Bowman model as it is not applicable[1].

Word Embeddings

To feed inputs into the model, we leverage the use of word embeddings. Within the NLP domain, word embeddings refer to vector based representations of words (or tokens) such that a semantic analysis can be performed between words. For the purposes of our models, they all leverage the GloVe (Pennington et al. (2014)) word embedding set with a dimension size of 50.

4.1 Datasets

We subject the models to three different datasets: Amazon Reviews (He and McAuley (2016)), Penn TreeBank (Marcus et al. (2002)), and OpenSubtitles (Lison and Tiedemann (2016)). Each dataset serves a different purpose, which are described in their respective sections.

4.1.1 Amazon Reviews

The Amazon products reviews (He and McAuley (2016)) comprises a set of customer submitted reviews of products on the popular consumer e-commerce Amazon.com, spanning May 1996 - July 2014. For the purposes of this ISO, we select the electronic products dataset, spanning 1.6m reviews, but due to computational constraints, we work on a subset of this dataset as reviews would also need to be broken down into sequences of sentences. This dataset is used as the primary indicator of the performance of the models in their capability for text generation, and the capabilities of encompassing lexical variety, as described in the original objective of the ISO. The objective of the models were to produce the next sentence from the previous sentence. Pre-processing includes:

1. Sentenisation and tokenisation using SpaCy (Honnibal and Montani (2017)).

[1] As stated earlier, the responses are generated in its entirety in one operation, and not necessarily seperated with recurrent steps.

```
{
  'reviewerID': 'AA8JH8LD2H4P9',
  'asin': '7214047977',
  'reviewerName': 'Claudia J. Frier',
  'helpful': [3, 4],
  'reviewText': 'This fits my 7 inch kindle fire hd perfectly! I love it.
   It even has a slot for a stylus. The kindle is velcroed in so it\'s
   nice and secure. Very glad I bought this!',
  'overall': 5.0,
  'summary': 'love it',
  'unixReviewTime': 1354665600,
  'reviewTime': '12 5, 2012'
}
```

```
IDENTITY: ['7', '2', '1', '4', '0', '4', '7', '9', '7', '7',
   'rating_5.0', 'polarity_-0.1']

['<sos>', 'love', 'it', '<sor>']
['this', 'fits', 'my', '7', 'inch', 'kindle', 'fire', 'hd', 'perfectly',
   '!', 'i', 'love', 'it', '.']
['it', 'even', 'has', 'a', 'slot', 'for', 'a', 'stylus', '.']
['the', 'kindle', 'is', 'velcroed', 'in', 'so', 'it', "'", 's', 'nice',
   'and', 'secure', '.']
['very', 'glad', 'i', 'bought', 'this', '!', '<eos>']
```

Figure 4.1: A sample review and the augmented data. Note that for each sequence from line 3 onwards has the identity sequence contatenated before it.

2. Concatenation of sequence conditioning on the input sequences.

3. Filtering tokens based on whether they exist in GloVe.

4. Keeping the top 10K frequent tokens of the vocabulary.

5. Removing sequences where the "<unk>" tags in the sequences represent a ratio of greater than 0.1.

6. Removing sequences that are greater than a length of 60.

Each input sentence is conditioned by the Item ID, review rating (of a score from 1-5), and a polarity value (calculated from Amazon's "helpful" values) consisting of -1 to $+1$ with an increment of 0.1. GloVe Word embeddings are used to represent tokens in the dataset. The polarity value p is computed with a two decimal place rounded ratio $p = round(helpful_+/helpful_-, 2)$. The polarity word vector is a sum of the polarity value p and the word vector corresponding to the word "polarity". This works in a similar fashion for the rating word embedding (which does not require calculating a ratio). Item IDs (recogised by Amazon as ASINs) are split character wise.

```
1 "<sos> ya ! <eos>" -> "<sos> write to your <unk> and hurry that up .
    <eos>"
2 "<sos> your paycheck ? <eos>" -> "<sos> back off tucker , you don ' t
    sketch regulations . <eos>"
```

Figure 4.2: A sample review and the augmented data. Note that for each sequence from line 3 onwards has the identity sequence contatenated before it.

4.1.2 OpenSubtitles

The OpenSubtitles dataset (Lison and Tiedemann (2016)) is used as a measurement in a similar fashion to the Amazon dataset, but does not involve any item descriptive conditioning concatenated to the input sequence. They are configured in a similar fashion to the Amazon dataset, described above. GloVe is also used. We use this to measure the effectiveness of the models on discourse generation without the use of additional contextual information. The goal for the models was to predict the next subtitle line, given a previous subtitle line. Pre-processing works in a similar fashion to the amazon dataset. It includes:

1. Sentenisation and tokenisation using SpaCy (Honnibal and Montani (2017)).
2. Concatenation of sequence conditioning on the input sequences.
3. Filtering tokens based on whether they exist in GloVe.
4. Keeping the top 10K frequent tokens of the vocabulary.
5. Removing sequences where the “<unk>” tags in the sequences represent a ratio of greater than 0.1.
6. Removing sequences that are greater than a length of 60.

4.1.3 Penn TreeBank

To verify implementation validation, we compute initial results with the Penn Tree-Bank Dataset. (Marcus et al. (2002)) This dataset is typically used for verification of model performance for generative models; the dataset consists of sequences which are the same for the input and output. (See 4.3) For the purposes of this ISO, we used the exemplar dataset which consists of 43K sequences. Pre-processing includes (1) tokenisation using NLTK; (2) trimming sequences that are greater than a length of 60; (3) replacing tokens with <unk>where the token does not exist in the vocabulary.

1. Sentenisation and tokenisation using SpaCy (Honnibal and Montani (2017)).
2. Tokenisation with NLTK (Bird et al.).
3. Trimming sequences that are greater than a length of 60.
4. Collecting the most frequent 10K words in the dataset to create the vocabulary.

```
1 [['<sos>', 'the', 'n', 'stock', 'specialist', 'firms', 'on', 'the',
    'big', 'board', 'floor', 'the', 'buyers', 'and', 'sellers', 'of',
    'last', 'resort', 'who', 'were', 'criticized', 'after', 'the', 'n',
    'crash', 'once', 'again', 'could', "n't", 'handle', 'the', 'selling',
    'pressure'], ['the', 'n', 'stock', 'specialist', 'firms', 'on',
    'the', 'big', 'board', 'floor', 'the', 'buyers', 'and', 'sellers',
    'of', 'last', 'resort', 'who', 'were', 'criticized', 'after', 'the',
    'n', 'crash', 'once', 'again', 'could', "n't", 'handle', 'the',
    'selling', 'pressure', '<eos>']]
2 [['<sos>', 'big', 'investment', 'banks', 'refused', 'to', 'step', 'up',
    'to', 'the', 'plate', 'to', 'support', 'the', 'beleaguered', 'floor',
    'traders', 'by', 'buying', 'big', 'blocks', 'of', 'stock', 'traders',
    'say'], ['big', 'investment', 'banks', 'refused', 'to', 'step', 'up',
    'to', 'the', 'plate', 'to', 'support', 'the', 'beleaguered', 'floor',
    'traders', 'by', 'buying', 'big', 'blocks', 'of', 'stock', 'traders',
    'say', '<eos>']]
```

Figure 4.3: A sample review and the augmented data. Note that for each sequence from line 3 onwards has the identity sequence contatenated before it.

5. Replacing tokens with "<unk>" where the token does not exist in the vocabulary.

For this particular dataset, we do not use any pre-trained word-embeddings; as the purposes of this dataset does not necessarily benefit from its addition. The purpose of this dataset is to measure the models reconstructive capabilities as a regular autoencoder. Models would use this dataset to repeat the input as the output sequence.

4.2 Training Setup

For the sake of comparisons, all models leverage a single bidirectional encoder. For variational models we utilise a linear KL annealing with a step size of $2.5 \cdot 10^{-5}$ with a threshold of 2000 steps. Dimension sizes for the hidden layer is 512, with the latent vector size being 400. The negative log likelihood loss (NLL), KL loss, and BOW losses are normalised by the sequence length.

All models use the Adam optimiser (Kingma and Ba (2014)) with a base learning rate of 0.001; subject to hyperparameterisation. As we measure the performance of three fundamentally different neural language architectures, we performed hyperparameter tuning for each of the models independently. All models are trained for 10 epochs to reduce the chances of overfitting.

Datasets were split in a 80:15:5 ratio, for training, validation, and test data respectively. Models were subject to roughly 15 hyperparameter tuning rounds (dependent on the confidence of the optimisation model) using Bayesian Optimisation (GPyOpt authors (2016)) where the objective was to minimise the ELBO (or NLL for the case of Seq2Seq). Early stopping is initialised such that the training will end when the mean loss of the validation batch reaches below an overall loss of 1.0.

4.3 Evaluation Measurements

To quantify how useful the models could be within the problem domain, we subject them to a variety of measurements, ranging from its learning rate, to the ability to recreate the appropiate response, and the ability to generate diverse responses.

4.3.1 KL Ratio

To quantify the performance of the models, we measure KL loss during training. (The KL distance would naturally be higher during inference and sampling, as this would be the primary cause of variance in the outputs). As the ELBO loss is comprised of both the reconstruction loss and the KL divergence, we use the KL loss to compute the KL Ratio (How much of the ELBO loss is comprised of the KL divergence). The KL ratio can help to understand the amount of information contained in the latent variable, and thus retaining more characteristics of the variational autoencoder as opposed to a regular autoencoder. Typically, the higher the divergence, the more information is encapsulated in the latent variable. We compare the performance of only the variational models (The VAD and the Bowman model).

4.3.2 BLEU and ROUGE

It is often the case to use subject human guided, hand-picked results for emperical measurements but due to a variety of constraints for the ISO this would not be feasible. Alternatively, we subject the models generated responses to shallow and automatic linguistic metrics such as the BLEU (Papineni et al. (2001)) and ROUGE (Lin (2004)) emperical scores. We measure the BLEU and ROUGE at each epoch to quantify the quality of our responses as it learns. These measurements are used alongside the Loss, and human analysis of the results. These metrics compare the generated responses against the labelled results; such that a higher score encompasses a greater similarity to the original dataset. BLEU (Papineni et al. (2001)) is originally defined to measure n-gram precision p_n of two text based sequences, by summing the n-gram matches for every hypothesis sentence S in the test corpus C. ROUGE is used in a similar fashion for recall. Both BLEU and ROUGE have the option to compute the n-gram match for several sizes of n-grams, but for simplicity we reduce it to uni-grams.

$$p_n = \frac{\sum_{S \in C} \sum_{ngram \in S} Count_{clip}(ngram)}{\sum_{S \in C} \sum_{ngram \in S} Count(ngram)} \qquad p_n = \frac{\sum_{S \in C} \sum_{ngram \in S} Count_{matched}(ngram)}{\sum_{S \in C} \sum_{ngram \in S} Count(ngram)} \tag{4.1}$$

Figure 4.4: Equations for BLEU (left) and ROUGE (right).

Although both equations look very similar, BLEU introduces a brevity penalty term where it would penalise results that are shorter than the general length of the test corpus.

$$f1_n = 2\frac{BLEU_n \cdot ROUGE_n}{BLEU_n + ROUGE_n} \tag{4.2}$$

Figure 4.5: The harmonic mean of BLEU and ROUGE scores.

To understand an overall picture of the model performance, we also measure the harmonic mean of BLEU and ROUGE, by measuring the F1 performance (Equation 4.5). The larger the F1 performance, the more faithful the model is in replicating the sequences of the output data given its conditioning. It is expected that the models should not necessarily reach perfect F1 scores, as this would indicate overfitting and a limited range in outputs.

4.3.3 Semantic Similarity

1: **procedure** SEMANTIC VARIANCE
2: *query* $\leftarrow$ input *sequence*
3: *resp* $\leftarrow$ []
4: **for** $i = 1$ to n **do**
5: $r_i \leftarrow$ model(*query*)
6: $r_i \leftarrow$ [embedding($token$) for $token$ *in* r_i]
7: *resp*[*i*] $\leftarrow$ *mean*(r_i)
8: *m* $\leftarrow$ *mean*(*resp*)
9: **return** *max*(*euclidean*($m, r_{i=1 \text{ to } n}$))

Additionally, to measure the variation in responses, we calculate the semantic variance between the generated outputs. For the purposes of this ISO, we simplify the semantic similarity measurements by leveraging pre-trained word embeddings, and calculating euclidean distances between vectors. The algorithm is described above.
Typically, the larger the distance, the more varied the responses. a semantic variance score of zero would suggest no diversity between the responses. In ideal conditions, large variances are observed. As we use the GloVe embeddings, each dimension in the embedding has a value range of $[-1, 1]$, leading to a theoretical maximum semantic similarity distance of 2.

Chapter 5

Experimental Results

5.1 Model Optimisation

5.1.1 NLL Loss

As we compare loss values for variational and non-variational models, we split the ELBO Loss (See Equation 2.8) into its constituent parts and analyse them independently to provide a better understanding of the overall performance of the models. The measurements below are taken from the average loss over the training batches for each epoch.

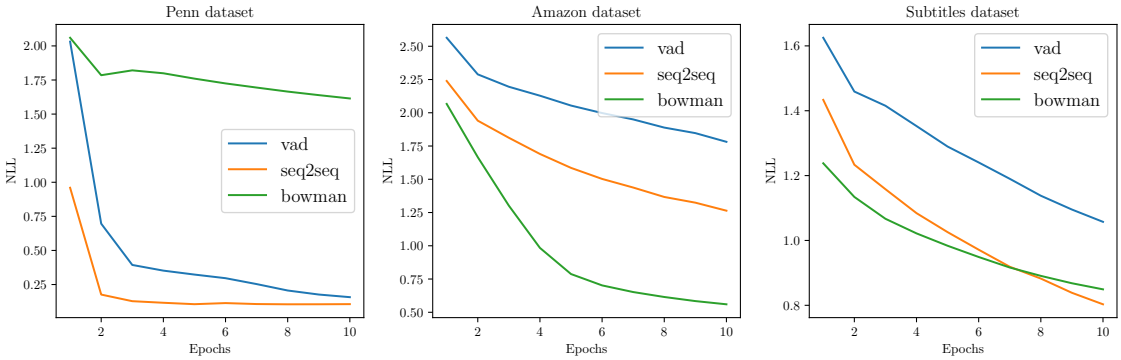

Figure 5.1: Reconstruction losses of the three models across the three datasets; lower loss is better.

With the Penn TreeBank dataset, The results shown are not particularly surprising. As there are less weights to learn, it was expected that Seq2Seq model would converge faster than the other models. Additionally, varational models are expected to perform worse than discrete models as the sampling mechanism will produce some variation in the responses. Interestingly the Bowman model performed the worst on the Penn dataset.

For the Amazon and Subtitles dataset, we observe that the Bowman model converges somewhat dubiously quickly, but the other statistics show that the bowman model has understood little about the dataset. The VAD and Seq2Seq has performed within expectations in terms of relative learning.

5.1.2 KL Ratio

Interestingly, it was easier for the VAD to initially encode more information into the latent variable for the Penn dataset. It is suggested that it is probably the case that when comparing the overall ELBO loss, the VAD model has most likely overfit on the dataset; leading it to destroy any progress during learning.

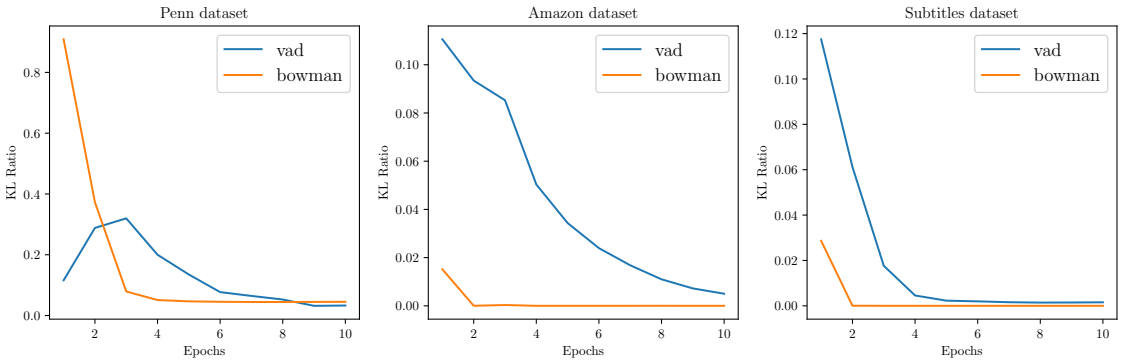

Figure 5.2: KL ratios of the three models across the three datasets; higher is better.

It however manages to encode some non-zero KL ratios on the other datasets, albeit it encompasses a small fraction of the ELBO. This could potentially be caused by the fact that the discourses in these datasets are one-to-many responses - The model may be confused by the various outcomes, forcing it to understand more on the reconstructive side as opposed to the variational sides.

5.2 Linguistic Analysis

Note that from this section onwards, we will be looking at the performance against the test splits of the datasets.

5.2.1 BLEU and ROUGE

Surprisingly, results for the Penn Dataset (Figure 5.3, left), the VAD is stronger than the other models by a considerable margin for both BLEU and ROUGE suggesting that it performs very strongly as an autoencoder. The VAD also performs considerably better on the Subtitles dataset (Figure 5.3, right) against Seq2Seq, indicating its applicability for conditional response generation.
The VAD is somewhat comparable with Seq2Seq on the Amazon dataset (Figure 5.3, center). This is somewhat contradictory to its performance on the Subtitles dataset, which may indicate some issue with how the Amazon dataset is initialised.
Unexpectedly the Bowman model performs considerably poorly against the Penn Dataset. Under further investigation with the Sampling examples, we see that the Bowman model does indeed capture the high level features, but the outputs do not reflect that of the

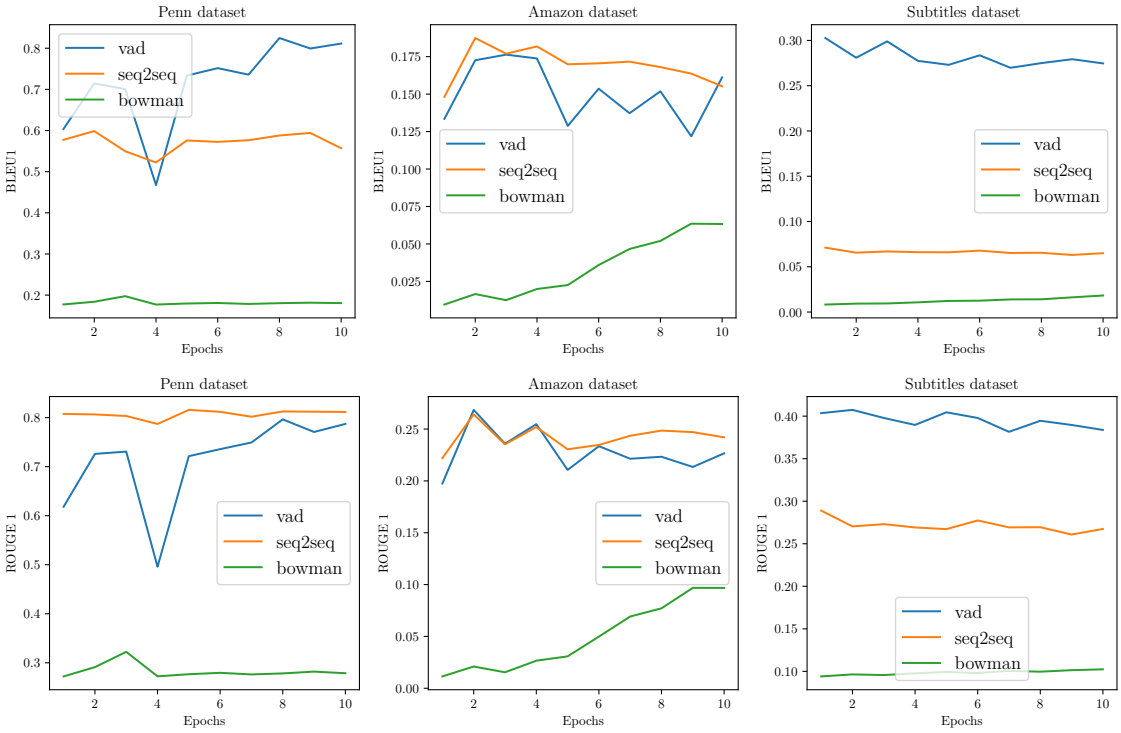

Figure 5.3: $BLEU_1$ (top) and $ROUGE_1$ (bottom) scores across the three models; higher is better.

input. Attempts to verify the authenticity of the results also confirm this. [1] For the other datasets, it was expected that this model would perform poorly by design, for reasons stated earlier (see Section 2.5.2).

5.2.2 F1

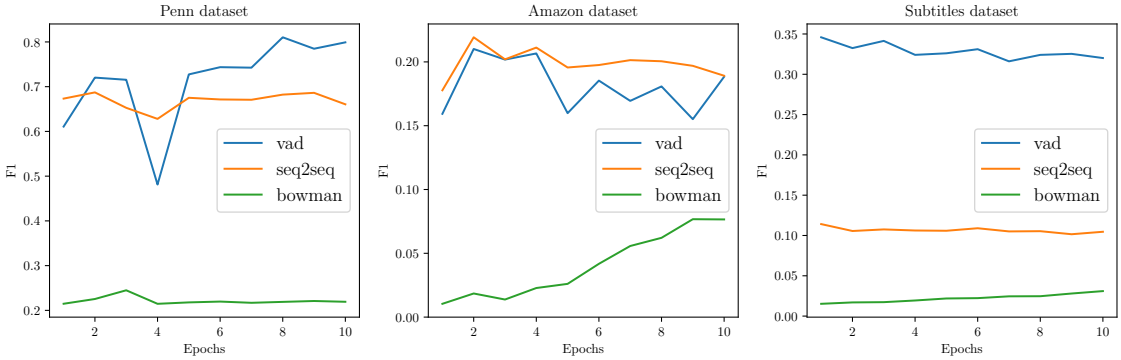

Figure 5.4: $F1_{1\text{-gram}}$ scores across the three models; higher is better.

The VAD performs the best with the Penn and Subtitles dataset, and is tied for the Amazon

[1] The Bowman Model was tested on a popular GitHub implementation, which can be found on https://github.com/timbmg/Sentence-VAE.

dataset. This strikes as particularly interesting, as it was originally hypothesized that additional conditional information could have helped to guide the responses.
Interestingly, the conditional datasets (Amazon and Subtitles) seem somewhat punishing for the models, with objectively low F1 scores. When compared to the example responses in Section , we notice that the responses are somewhat related and coherent for the VAD and Seq2Seq. One possible reason for is due to the open-endedness of responses in both of these datasets, where some fixed dialogue may typically lead to multiple responses[2].

5.2.3 Output Variance

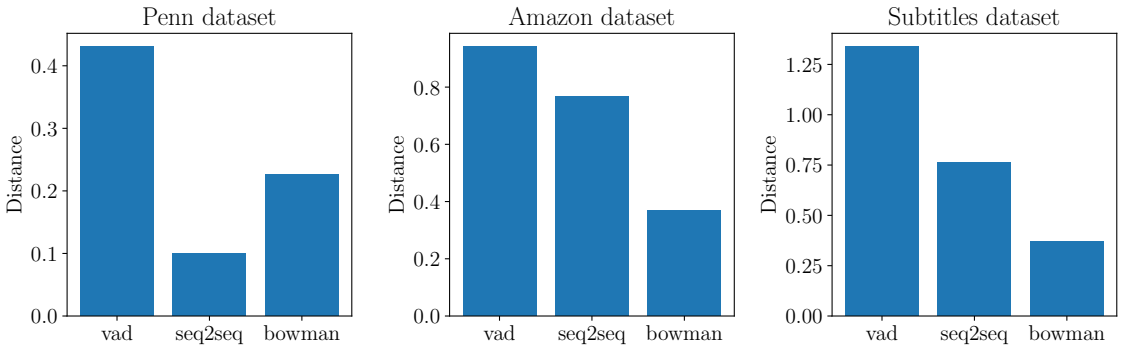

Figure 5.5: Semantic similarity scores across the datasets and models, higher is better.

We can clearly see the disparity of the results between the models, with the VAD performing the best across the datasets. High variance for the Seq2Seq model on the Amazon and Subtitles dataset puts more confidence on the suggestion made earlier about one-to-many responses.
The Penn dataset represents a case where the latent sampling comes into play and shows a scenario it does increase the variance in the results, for the Bowman model. Otherwise, for the other datasets, the Bowman model naturally underperforms and leaves much to be desired.

5.2.4 Sampling Examples

Table 5.1 shows some of the outputs generated from the three models, against the three datasets. As some of our models are based on bayesian inference, we sample ten outputs and hand pick the most interesting responses from the datasets from each of the relevant models.
For the Subtitles dataset, it was observed that the VAD generated linguistically superior and more plausible results compared to the Seq2Seq network. These results compliment the quantitiative measurements observed earlier.
On the contrary, all models performed less admirably on the Amazon dataset. The Seq2Seq and VAD model produces somewhat coherent responses but generates relatively generic responses, or does not leverage the contexts provided by the query enough.

[2]e.g. With the Subtitles dataset, a query with ”who is it?” could elicit a variety of responses.

Dataset	Query	Bowman	Seq2Seq	VAD
Penn	these stocks eventually reopened	the <unk> $ cents <unk> the <unk> open	these stocks eventually reopened	these stocks eventually reopened
	as a result the link between the futures and stock markets , apart	they is n't elaborate whether the <unk> is a we is s going to be a a years	as a result the link between the futures and stock markets , apart	as a result the link between the futures between and futures markets apart
Open Subtitles	i give you ride .	N/A	you ' re going to take a delivery hour	i ' m gon na take a walk .
	not exactly .	?	he ' s hideous <unk>, he ' s a musician	i ' m gon na be able to finish the door .
Amazon	b 0 0 d q d c 1 y 6 rating_4.0 polarity_0.8 i also have a background in it .	i the : the the the , you 's the i , is the ,	i am going to get it to work well .	the email has a nice connection .
	b 0 0 9 l l 9 v d g rating_5.0 polarity_-0.8 you will need the desktop to run larger things off of , and to use a printer remotely .	N/A	i have a fit of the canon .	i a little to in distracting for the price .

Table 5.1: Responses generated from the models across the datasets.

The Bowman model failed to generate remotely coherent responses for the Open Subtitles and Amazon datasets, and this is most likely related to the fact that it was designed for representation learning.

Chapter 6

Evaluation and Future Work

Although some of the datasets were different and varying in purpose, scores are weaker for all models compared to their portrayed potential in their respective papers. This could primarily be attributed to two causes: (1) The dataset sizes used for the models were too small (as they were reduced to accomodate feasible training times), or (2) hyperparameter optimisation was too short in duration. In hindsight, increasing both should provide a clearer potential performance across the models, but the timescale and computational resources for the project was limited and could not afford the ability to scale up.
Further work should be taken to understand the influence of the BOW loss. Attempts to contact the the authors of the VAD paper have been unsuccessful, and it remains unclear what the influence of the α is in regards to the weight of the BOW loss. Initial experiments have shown a positive correlation with the KL divergence when α is increased, but details remain uncertain. An assorted set of additional comments are below:

- Generated responses may not necessarily be indicative of the data it is modelled against.
- Although there are mechanisms to control the vanishing KL problem, it is yet to be seen how universally applicable it would be.
- The amount of hyperparameter tuning for the VAD is considerably larger than other neural models due to the inclusion of additional mechanisms for controlling the vanishing KL.
- Further testing of the additional components introduced to the VAE should be measured independently to measure their influence to the KL loss.
- BLEU/ROUGE calculations should have been taken on the best output instead of an average of the whole sequence, but required extensive memory usage and would take a considerably longer running time due to technical implementation. In hindsight, this should have been considered as an additional measurement to improve the robustness of the results.
- The Amazon reviews sequence data could have more experimentation. For instance, the ISIN/Item ID could be represented as a single word token as opposed to broken down by character level, and the embedding value could be the mean of the

embedding values of the character level tokens. Alternatively, the item identities could be inserted into the model through another input to the decoder, as opposed to concatenating it onto the input sequence.

- There is a potential for improving the KL performance by masking the SBOW matrix to the predicted output, such that the reconstruction loss would be a directly influenced by the auxiliary function.
- An additional measurement could be used to determine whether the language models could pass off as a replacement to the original dataset. A classification model could be used to segregate sentences between the dataset and model generated samples.

Chapter 7

Conclusion

In conclusion, we explored a neural language model in detail that could potentially be used for the purposes of encompassing the lexical variety of the original dataset. We have discussed the limitations of the model and used them to augment the components in a manner that allow them to work in unison. By subjecting them to quantitiative and qualitative experimental results, we discover that the VAD can be sensibly considered for the task at hand. In future works, the model could be extended such that it would produce stronger and more lexically coherent discourses in a similar fashion as the dataset. One particularly interesting avenue for exploration could be related to adversarial approaches, which could be used in unison with the model to improve the diversity of artficial text generation.

Bibliography

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*. arXiv: 1409.0473. pages 11

Bengio, Y., Ducharme, R., and Vincent, P. (2001). A Neural Probabilistic Language Model. In Leen, T. K., Dietterich, T. G., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13*, pages 932–938. MIT Press. pages 5

Bird, S., Klein, E., and Loper, E. *Natural Language Processing with Python*. pages 22

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. (2015). Generating Sentences from a Continuous Space. *arXiv:1511.06349 [cs]*. arXiv: 1511.06349. pages 12, 18

Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *arXiv:1409.1259 [cs, stat]*. arXiv: 1409.1259. pages 7

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv:1412.3555 [cs]*. arXiv: 1412.3555. pages 7

Du, J., Li, W., He, Y., Xu, R., Bing, L., and Wang, X. (2018). Variational Autoregressive Decoder for Neural Response Generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3154–3163, Brussels, Belgium. Association for Computational Linguistics. pages i, 2, 12, 14, 16, 17

Dyer, C. (2017). Conditional Language Modelling. original-date: 2017-02-06T11:32:46Z. pages 4

E. Rumelhart, D., H. Hinton, G., and RJ, W. (1986). Learning Internal Representations by Error Propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1. pages 7

Fu*, H., Li*, C., Liu, X., Gao, J., Celikyilmaz, A., and Carin, L. (2019). Cyclical Annealing Schedule: A Simple Approach to Mitigating KL Vanishing. pages 8

Genthial, G. (2017). Seq2seq with Attention and Beam Search. pages 12

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. pages 19

GPyOpt authors (2016). GPyOpt: A Bayesian Optimization framework in python. pages 23

He, R. and McAuley, J. (2016). Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In *Proceedings of the 25th International Conference on World Wide Web - WWW '16*, pages 507–517, Montréal, Québec, Canada. ACM Press. pages 2, 20

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780. pages 7

Honnibal, M. and Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. pages 20, 22

Jiang, S. and de Rijke, M. (2018). Why are Sequence-to-Sequence Models So Dull? Understanding the Low-Diversity Problem of Chatbots. In *Proceedings of the 2018 EMNLP Workshop SCAI: The 2nd International Workshop on Search-Oriented Conversational AI*, pages 81–86, Brussels, Belgium. Association for Computational Linguistics. pages 11

Jurafsky, D. and Martin, J. (2019). *Speech and Language Processing*. 3 edition. pages 5

Kannan, A., Young, P., Ramavajjala, V., Kurach, K., Ravi, S., Kaufmann, T., Tomkins, A., Miklos, B., Corrado, G., Lukacs, L., and Ganea, M. (2016). Smart Reply: Automated Response Suggestion for Email. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, pages 955–964, San Francisco, California, USA. ACM Press. pages 4

Karpathy, A. (2015). The Unreasonable Effectiveness of Recurrent Neural Networks. pages 5

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. pages 23

Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*. arXiv: 1312.6114. pages 2, 8, 9

Kovalenko, B. (2017a). *Controllable text generation*. PhD thesis, Stanford University. pages 3

Kovalenko, B. (2017b). Controllable text generation. pages 3

Le, J. (2018). Recurrent Neural Networks: The Powerhouse of Language Modeling. pages 4

Lin, C.-Y. (2004). ROUGE: A Package for Automatic Evaluation of Summaries. pages 74–81. pages 24

Lison, P. and Tiedemann, J. (2016). OpenSubtitles2016: Extracting Large Parallel Corpora from Movie and TV Subtitles. page 7. pages 2, 20, 22

Marcus, M., Ann Marcinkiewicz, M., and Santorini, B. (2002). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19:313–330. pages 20, 22

Nallapati, R., Zhai, F., and Zhou, B. (2016). SummaRuNNer - A Recurrent Neural Network based Sequence Model for Extractive Summarization of Documents. pages 4

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2001). BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, page 311, Philadelphia, Pennsylvania. Association for Computational Linguistics. pages 24

Pennington, J., Socher, R., and Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. pages 20

Raiko, T., Berglund, M., Alain, G., and Dinh, L. (2014). Techniques for Learning Binary Stochastic Feedforward Neural Networks. *arXiv:1406.2989 [cs, stat]*. arXiv: 1406.2989. pages 8

Serban, I. V., Sordoni, A., Lowe, R., Charlin, L., Pineau, J., Courville, A., and Bengio, Y. (2016). A Hierarchical Latent Variable Encoder-Decoder Model for Generating Dialogues. *arXiv:1605.06069 [cs]*. arXiv: 1605.06069. pages 11

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958. pages 18

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *arXiv:1409.3215 [cs]*. arXiv: 1409.3215. pages 2, 3, 10, 19

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention Is All You Need. *arXiv:1706.03762 [cs]*. arXiv: 1706.03762. pages 11

Williams, R. J. and Zipser, D. (1989). A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, 1(2):270–280. pages 20

Zhao, T., Zhao, R., and Eskenazi, M. (2017). Learning Discourse-level Diversity for Neural Dialog Models using Conditional Variational Autoencoders. *arXiv:1703.10960 [cs]*. arXiv: 1703.10960. pages 11, 12, 13, 16

Chapter 8

Appendix

8.1 KL Divergence Derivation

Let $\mu_1, \sigma_1 \rightarrow \mathcal{N}(\mu_1, \sigma_1)$ be our first distribution, and $\mu_2, \sigma_2 \rightarrow \mathcal{N}(\mu_2, \sigma_2)$ be our second distribution. By deriving this, we would be able to calculate a derivative friendly loss function for our models.

$$
\begin{aligned}
KL &= \int \left[\log(p(x)) - \log(q(x))\right]\ p(x)\ dx \\
&= \int \left[\frac{1}{2}\log\frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) + \frac{1}{2}(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2)\right] \times p(x)dx \\
&= \frac{1}{2}\log\frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2}\text{tr}\ \left\{E[(x-\mu_1)(x-\mu_1)^T]\ \Sigma_1^{-1}\right\} + \frac{1}{2}E[(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2)] \\
&= \frac{1}{2}\log\frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2}\text{tr}\ \{I_d\} + \frac{1}{2}(\mu_1-\mu_2)^T\Sigma_2^{-1}(\mu_1-\mu_2) + \frac{1}{2}\text{tr}\{\Sigma_2^{-1}\Sigma_1\} \\
&= \frac{1}{2}\left[\log\frac{|\Sigma_2|}{|\Sigma_1|} - d + \text{tr}\{\Sigma_2^{-1}\Sigma_1\} + (\mu_2-\mu_1)^T\Sigma_2^{-1}(\mu_2-\mu_1)\right].
\end{aligned}
\tag{8.1}
$$