Imperial College
London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Chameleon Text: Exploring ways to increase variety in artificial data

Author:
Thien P. Nguyen

Supervisor:
Julia Ive, Lucia Specia

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

May 2019

Contents

Chapter 1

Literature Survey

1.1 Artificial Data

The premise of this literature survey is to describe the relevant components necessary to construct our solution, and to also discuss alternative approaches to the problem.

1.2 Text Generation

Text generation is a a type of Language Modeling problem, which in itself, is one of the core natural language processing problems. It is used in a variety of contemporary applications, ranging from machine translation, to email response generation, to document summarisation.
In our particular case we imagine a scenario where a client requests the use of our dataset. We could permit them access to said data, but it would reveal the identities of the users in the data. The objective is to have some alternative dataset to ours 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 variety of our original dataset. This language model would train on our original dataset, and would 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 new dataset can be assigned to other clients.

1.3 Language Modelling

Language modelling is the task of predicting a word w_i in a text w given some sequence of previous words $(w_1, w_2, ..., w_{i-1})$. More formally, 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:

$$p(w) = p(w_1) \times p(w_2|w_1) \times p(w_3|w_1, w_2) \times ... \times p(w_i|w_1, w_2, ..., w_{i-1}) \tag{1.1}$$

$$p(w) = \prod_{t=1}^{|w|} p(w_t|w_1, ..., w_{t-1}) \tag{1.2}$$

Traditionally, assigning words to probabilities may conflate syntactically dubious sentences but it remains to be a useful method for representing texts.
This paper concentrates on conditional language modelling. A conditional language model assigns probabilities to sequences of words, $w = (w_1, w_2, ..., w_{i-1})$, given a conditioning variable, x.

$$p(w|x) = \prod_{t=1}^{|w|} p(w_t|x, w_1, ..., w_{t-1}) \tag{1.3}$$

There exists different types of language models; we start with the n-gram, argued as being the most fundamental (Le (2018)). An n-gram is a chunk 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 was that statistical inference can be applied on the frequency and distribution of of the n-grams, which could be used to predict the next word. However, sparsity is not captured.
Later on, modern language models revolve around the use of neural networks (Bengio et al. (2001)), which itself started off with a relatively modest MLP that formed the premise of word prediction. The use of neural networks in language modelling is often called Neural Language Modelling, of NLM for short.
Neural Networks are non-linear and computational metaphors of the brain that model arbitary relationships between input and output vectors. (This type of neural network architecture is commonly described as the multi-layer perceptron). Note that the input and output vectors are of a fixed dimension, which becomes a problem for our task at hand. 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.

1.4 Dataset

For our models we use the Amazon Reviews dataset byHe and McAuley (2016). This particular dataset is relatively unforgiving as opposed to using language translation.

1.5 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a class of feed forward neural networks such that the outputs are not necessarily restricted and discrete (as opposed to the MLP). RNNs operate over a sequence of variable-length vectors, and produces an output of similarly variable-length vectors. This circumvents a problem introduced with using an MLP, where sentences are not typically fixed length.
The architecture of RNNs make it favourable in NLP related problems as words in sentences are typically conditioned on the previous words. When treated as language models, a standard RNN language model predicts each word of a sentence conditioned on the previous word and an evolving hidden state.
At each time step t, a simple RNN takes produces a hidden vector h_t, derived from the input vector x_t and the previous state h_{t-1} in the function $h_t = f_w(h_{t-1}, x_t)$. The hidden vector is usually obtained by some affine transformation offset by a bias vector, described in more detail in equation 1.4. The same function and the same parameters are used at each time step t. RNNs typically are not parallelised.

$$\begin{aligned} h_t &= tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t &= W_{hy}h_t \end{aligned} \tag{1.4}$$

RNNs are typically trained with backpropagation through time. 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.

$$\frac{\delta L}{\delta w} = \sum_{j=0}^{T}\sum_{k=1}^{j}\frac{\delta L_j}{\delta y_j}\frac{\delta y_j}{\delta h_j}\Big(\prod_{t=k+1}^{j}\frac{\delta h_t}{\delta h_{t-1}}\Big)\frac{\delta h_k}{\delta w} \tag{1.5}$$

Typically, there are two caveats with training RNNs. One of which involves the sensitivity of the gradients: The product symbol within the loss function in 1.5 is the cause of vanishing and exploding gradients. The sensitivity of the gradients make it especially difficult to train RNNs. Additionally, it became aparrent that it was very difficult for RNNs to leverage relationships between potentially relevant inputs and outputs - there isn't necessarily a clear indicator in the architecture that would facilitate this feature. This is described as a long range dependency problem.

1.5.1 LSTMs and GRUs

Both gates were introduced to circumvent the issue of long range dependency problems by providing multiple avenues, with each one having a different approach.
LSTM (Long Short Term Memory; Hochreiter and Schmidhuber (1997)) are a type of RNN cell that attempts to retain information based on the previous inputs through the introduction of gated architectures.
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 an direct path to the input data from the output, allowing a reduction in training time. They are unable to clearly distinguish

$$
\begin{aligned}
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \qquad & u_t &= \sigma(W_u \cdot [h_{t-1}, x_t] + b_u) \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \qquad & r_t &= \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \\
g_t &= \sigma(W_g \cdot [h_{t-1}, x_t] + b_g) \qquad & c_t &= tanh(W w_{t-1} + U(r_t \odot h_{t-1})) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \qquad & h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot c_t \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
\tag{1.6}
$$

Figure 1.1: Equations for LSTM cells (left) and GRU cells (right).

between the performance of the two gated units they tested. Chung et al. (2014) suggests that GRUs were found to perform at least equivalent to LSTMs but show slight improvements on smaller datasets.

1.6 Autoencoders

Autoencoders are a specialised form of neural networks where the model attempts to recreate the inputs on the output. Autoencoders typically have layer in the model where its dimension is smaller than the input space, therefore representing a dimensionality reduction in the data.
Autoencoders are composed of two different principal components, an encoder network α and a decoder network β, such that $\alpha : X \to F$ and $\beta : F \to X$. Measuring the success of the reconstruction is deduced by a reconstruction loss formula. This reconstruction loss comapres the output of the decoder and compares it against the input of the encoder. The two networks are trained together in a manner that allows them to preserve the input as much as possible.
Autoencoders are popularised through their use in Machine Translation, Word Embeddings, and document clustering.

1.6.1 Variational Autoencoders

Variational Autoencoders (VAEs, Kingma and Welling (2013)) are based on a regularised version of the standard auteoncoder. VAEs forces the encoder to learn parameters of a gaussian distribution - a mean and a covariance - as opposed to creating a fixed latent vector. In more discrete terms, it replaces the encoder from the standard autoencoder with a learned posterior *recognition model*, $q(\overrightarrow{z}|x)$. This component of the autoencoder parameterises an approximate posterior distribution over $\overrightarrow{z}$, using a neural network conditioned on x. This consequently allows us to take advantage of recent advances in variational inference. A sample is taken from a gaussian using the learned parameters, which is then fed into the decoder. Training this particular model involves learning the parameters involved for our latent distribution.
Note that the decoder receives samples from a non-standard normal distribution produced by the encoder. The average of the samples of the different distributions

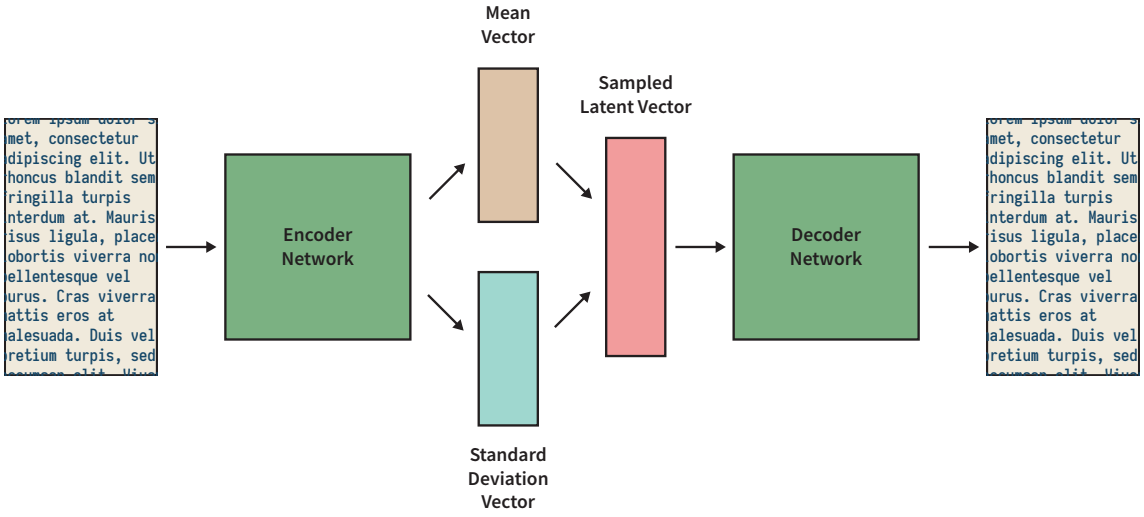

Figure 1.2: An abstracted model architecture for a variational autoencoder, which takes as input some text, and it's predicted output being the same text as the input.

should approximate to a standard normal. This stochasticity allows us to model variability in the results.
If VAEs were trained with a standard autoencoder's reconstruction objective, then it would learn to encode it's inputs determinsitically by making variances in *recognition model*, $q(\overrightarrow{z}|x)$ vanishingly small. (Raiko et al. (2014)) Instead, the loss function for VAEs is composed of two components; a reconstruction loss that involves an expection of the output; and a KL divergence (see Equation 1.8), which measures the distribution of the posterior distributions against a standard gaussian $\mathcal{N}(0,1)$). This objective provides a valid lowerbound on the true likelihood of the data, making VAEs a generative model.

$$\mathcal{L}(\theta,\phi,x,z) = \mathbb{E}_{q\phi(z|x)}[log\, p_\theta(x|z)] - D_{KL}(q_\phi(z|x)\,||\,p(z)) \leq log(p(x)) \tag{1.7}$$

$$D_{KL}(P||Q) = \sum_{x \subset X} P(x) \cdot log(\frac{P(x)}{Q(x)}) \tag{1.8}$$

In other words, it is the expectation of the logarithmic difference between the probabilities P and Q, where the expectation of P is already known.

Reparameterisation Trick

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

The Gaussian reparameterisation trick (Kingma and Welling (2013)) subverts the issue of performing backpropagation against sampled latent variables through the rearrangement of the gaussian parameters. Instead of sampling z through a gaussian $\mathcal{N}(\mu, \Sigma)$, z can be created by a summation of μ and some gaussian noise $\epsilon \sim \mathcal{N}(0,1)$ applied to Σ. This rearrangement allows for derivatives to be calculated with respect to the parameters that are needed for learning.

1.6.2 Conditional Variational Autoencoders

In a VAE, the decoder class cannot produce outputs of a particular class on demand. CVAEs are an improved model of the original VAE architecture by conditioning on another description of the data, a class descriptor y.

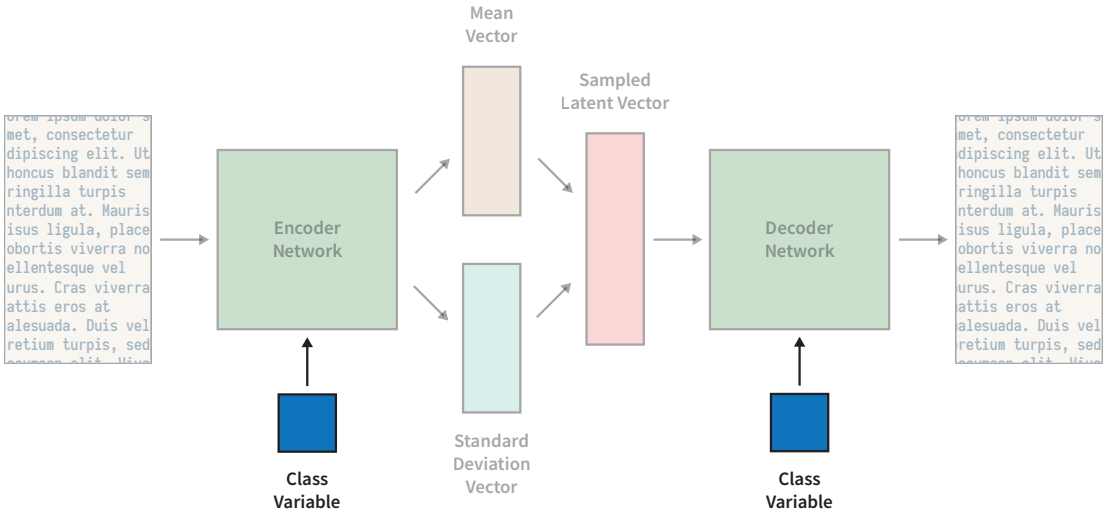

Figure 1.3: A model architecture for a CVAE, which includes the label being fed into the encoder and decoder networks.

During training time, a class (represented by some arbitary vector) is fed at the same time to the encoder and decoder. To generate an output that depends on y we feed that number to the decoder along with a random point in the latent space sampled from a standard normal distribution.
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 concate the class variable to the dataset prior to feeding. This allows the model to retain its characteristics without needing to adjust in order to accomadate the conditioning variables.

1.7 Related Work

1.7.1 Sequence To Sequence

Sequence to Sequence, (seq2seq, Sutskever et al. (2014)) is a type of neural language model that models relationships between sequences. In our particular scenario, this could be seen as a modern interpretation of the autoencoder. Seq2Seq comprises of two components, encoders and decoders, both of which are represented with RNN models (although can be replaced with LSTM or GRU cells). The last hidden state value of the encoder is recognised as the context variable, which encodes

the characteristics of the input sequence. This is used as the first hidden vector value for the decoder.

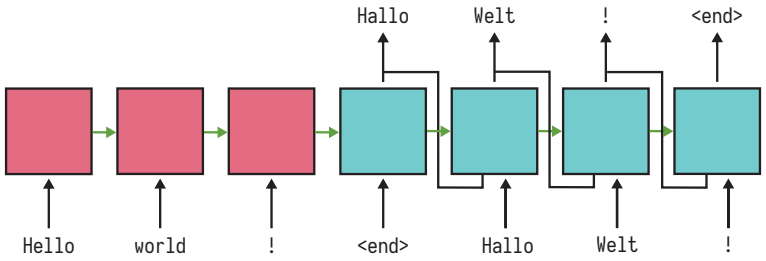

Figure 1.4: An abstracted model of the seq2seq architecture, where the encoder (pink) takes in the input sequence, and the decoder (blue) shows the output sequence.

Seq2seq models have shown to be effective in reponse generation, but is no longer considered to be state of the art. For instance, the context variable itself is considered to be bottlenecked as it has to encompass the entire input sequence in a single vector. As we've discussed earlier in Section 1.5, the training mechanism is subject to vanishing and exploding gradients; 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. We explain later about approaches to mitigate the issue.
Furthermore, the responses produced from a vanilla seq2seq model tends to lack 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 tackle 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.

Variational Context

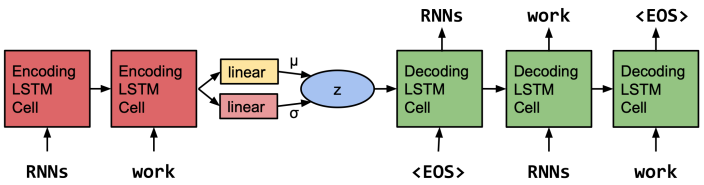

Figure 1.5: The core structure of the VAE language model - words are represented as embedded vectors (Bowman et al. (2015)).

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 how VAEs are designed (Bowman et al. (2015)).
The model is trained in a similar fashion to VAEs in general (see Equation 1.7) but utilises additional mechanisms to force the language model to encode features into the gaussian parameters. We explain these features in further detail in Section 2.3.

Attention Mechanisms

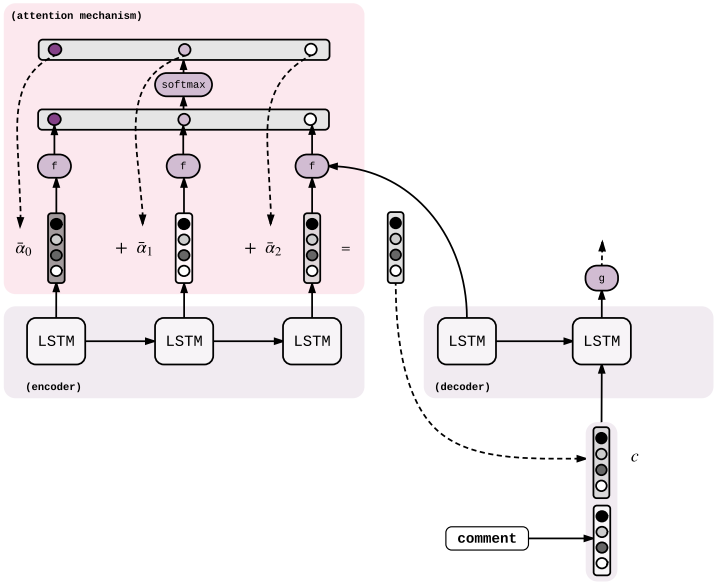

Figure 1.6: An abstracted diagram of the attention mechanism, applied to a seq2seq model.

In addition to the context vector, the decoder can also retrieve information on the encoder values using attention (Bahdanau et al. (2014)). This mechanism looks at all of the inputs from the hidden states of the encoders so far. This allows the decoder network to ”attend” to different parts of the source input at each step of the output generation. This circumvents the need to encode the full source input into a fixed-length vector, helping it deal with long-range dependency problems.
Furthermore, this attention mechanism allows the model to learn what to attend to based on the input sequence and what it has so far, represented through a weighted combination of the two.
Attention mechanisms were found to perform particularly well in machine translation problems where languages which are relatively well aligned (such as English and German). The decoder is most likely able to choose to attend to the response generation sequentially, such that the first output from the decoder can be generated based on the properties of the first input of the encoder and so on.

Chapter 2

Model

In this chapter we discuss the current implementation in detail. It builds on the knowledge in the first chapter, and the model itself is a combination of features described above.

2.1 Variational Autoregressive Decoders

To increase variability of outcomes produced by the original seq2seq model, Serban et al. (2016); Zhao et al. (2017) proposed the use of variational autoencoders to seq2seq models. At generation time, the latent variable z can be used as a conditional signal of the decoder in the seq2seq model. Although this introduces variability in the outputs, the variability is not controlled; i.e it is produced from the randomness of z. The underlying seq2seq model remains suboptimal.
Introduced by Du et al. (2018), Variational Autoregressive Decoders (VADs) attempt to circumvent the sampling problem introduced from CVAEs 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 variety of responses to be produced.
We will be using this model to help solve the variability of outcomes. *WHY?*
VADs use the seq2seq architecture as the base with variable-length queries $x = \{x_1, x_2, ..., x_n\}$, and $y = \{y_1, y_2, ..., y_n\}$ representing the input and output responses respectively. The encoder network is a Bidirectional RNN with GRUs. The decoder network is an unidirectional RNN with GRUs. For each timestep t, each GRU in the decoder network is encoded with hidden state h_t^d.

Encoder

$$\begin{aligned} \overrightarrow{h_t^e} &= \overrightarrow{GRU}(x_t, \overrightarrow{h_{t-1}^e}) \\ \overleftarrow{h_t^e} &= \overleftarrow{GRU}(x_t, \overleftarrow{h_{t+1}^e}) \end{aligned} \tag{2.1}$$

The encoder works in a similar fashion to the encoder described in the seq2seq network. The only changes here we see is that the encoder described in the implementation leverages bi-directionality. This leverages the sequentiality 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.

Backwards

$$\overleftarrow{h_t^d} = \overleftarrow{GRU}(y_{t+1}, \overleftarrow{h_{t+1}^d}) \tag{2.2}$$

The backwards RNN is one of the components used to condition the latent variable z in the decoder. During training, the backwards RNN takes as input the training outputs y and outputs hidden vectors in a sequential manner.

Attention

$$\begin{aligned} \alpha_{s,t} &= f_{attention}([h_d^e, h_{t-1}^d]) \\ c_t &= \sum_{s=1}^{m} \alpha_{s,t} h_s^e \end{aligned} \tag{2.3}$$

- Uses Luong's Attention mechanism that involves a concatenation. - uses a backwards RNN that represents the actual response to better create a posterior distribution.

Inference Model

$$\begin{aligned} [\mu^i, \sigma^i] &= f_{infer}([\overrightarrow{h_{t-1}^d}, c_t, \overleftarrow{h_t^d}]) \\ q_\theta(z_t|\boldsymbol{y}, \boldsymbol{x}) &= \mathcal{N}(\mu^i, \sigma^i) \end{aligned} \tag{2.4}$$

The inference model attempts to create a posterior distribution z using the conditioned output, the attention weights, and the decoder outputs.

Prior Model

$$\begin{aligned} [\mu^p, \sigma^p] &= f_{infer}([\overrightarrow{h_{t-1}^d}, c_t]) \\ p_\phi(z_t|\boldsymbol{y}_{<t}, \boldsymbol{x}) &= \mathcal{N}(\mu^p, \sigma^p) \end{aligned} \tag{2.5}$$

The prior network is restricted to using observable variables in the testing phase in order to generate z_t. It is designed in a similar fashion to the decoder network where the input variables are concatenated together.

Decoder (Variational Autoregressive)

$$\overrightarrow{h_t^d} = \overrightarrow{GRU}([y_{t-1}, c_t, z_t], \overrightarrow{h_{t-1}^d}) \tag{2.6}$$

$$p_\phi(y|\boldsymbol{y}_{<t}, \boldsymbol{z}_t, \boldsymbol{x}) = f_{output}([\overrightarrow{h_t^d}, c_t]) \tag{2.7}$$

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. z_t is conditioned by the backwards hidden vector h_t^d. Du et al. (2018) suggests that this conditioning allows the latent variables to be guided for long-term generation.

Auxillary Objective

An auxillary objective that attempts to predict the bag of words based on the latent variable. Helps to improve the output options. Let $f = MLP_b(z_t) \in \mathcal{R}^V$ where V be the vocabulary size:

$$log[p_\xi(y_{bow(t+1,T)}|z_{t:T})] = log\frac{e^{f_{z_t}}}{\sum_j^V e^{f_j}} \tag{2.8}$$

Learning Mechanism

VADs use a weighted combination of ELBO (Equation 1.7) the log likelihood loss of the auxillary objective.

$$\mathcal{L}'(\theta, \phi, x, c) = \mathcal{L}'(\theta, \phi, x, c) + E_{q_\phi(z|c,x,y)}[log(p(x_{bow}|z, c))] \tag{2.9}$$

KL Divergence Derivation

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

$$\begin{aligned}
KL &= \int [\log(p(x)) - \log(q(x))]\ p(x)\ dx \\
&= \int \left[\frac{1}{2}\log\frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) + \frac{1}{2}(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2)\right] \times p(x)dx \\
&= \frac{1}{2}\log\frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2}\text{tr}\ \left\{E[(x-\mu_1)(x-\mu_1)^T]\ \Sigma_1^{-1}\right\} + \frac{1}{2}E[(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2)] \\
&= \frac{1}{2}\log\frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2}\text{tr}\ \{I_d\} + \frac{1}{2}(\mu_1-\mu_2)^T\Sigma_2^{-1}(\mu_1-\mu_2) + \frac{1}{2}\text{tr}\{\Sigma_2^{-1}\Sigma_1\} \\
&= \frac{1}{2}\left[\log\frac{|\Sigma_2|}{|\Sigma_1|} - d + \text{tr}\{\Sigma_2^{-1}\Sigma_1\} + (\mu_2-\mu_1)^T\Sigma_2^{-1}(\mu_2-\mu_1)\right].
\end{aligned} \tag{2.10}$$

We derive the KL Divergence as we no longer measure the distance of our posterior distribution against a standard gaussian. We instead train the

2.2 Data Augmentation

Spacy is used. The dataset produced has to be augmented in order for the model to process them.

2.2.1 Embeddings

We use the glove word2vec word embeddings.

```
{
  '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'
}
```

```
1 IDENTITY: ['7', '2', '1', '4', '0', '4', '7', '9', '7', '7',
    'rating_5.0', 'polarity_-0.1']
2
3 ['<sos>', 'love', 'it', '<sor>']
4 ['this', 'fits', 'my', '7', 'inch', 'kindle', 'fire', 'hd',
    'perfectly', '!', 'i', 'love', 'it', '.']
5 ['it', 'even', 'has', 'a', 'slot', 'for', 'a', 'stylus', '.']
6 ['the', 'kindle', 'is', 'velcroed', 'in', 'so', 'it', "'", 's',
    'nice', 'and', 'secure', '.']
7 ['very', 'glad', 'i', 'bought', 'this', '!', '<eos>']
```

Figure 2.1: A sample review and the augmented data. Note that for each sequence from line 3 onwards has the identity sequence contatenated before it.

2.2.2 Review Prepropressing

The model would be forced to produce the next sentence from the previous sentence. Sentences are split by periods. Tokens are represented as words or individual punctuation marks. Additional tags are introduced. Lowercase.
ASIN represents item ID. this is split by characters.

2.2.3 Polarity Calculation

This is calculated with:

$$p = \sigma_{tanh}(\frac{h_{pos} - h_{neg}}{h_{pos} + h_{neg}}) \tag{2.11}$$

The polarity word vector is a sum of the polarity value p and the word vector corresponding to the word ”polarity”.

2.2.4 Implementation

Built on PyTorch. Models take a considerable amount of time to train, which does not necessarily align with rapid evolution of the implementation. Components are typically built with states in mind such that when running a model, it keeps a state

of the source codes at the time of running in its own results container. This allows us to repeat the results using the code produced at the time.

Weight Initialisation

Typically, we start with wanting to have the weights being asymmetric (such that when we perform our gradient descent we would have random starting positions, which would enable us to have a better chance to find the minimum as opposed to having a fixed starting point). Fortunately, pulling from a gaussian distribution is sufficent. The problem here is that we're fixing some parameter for the standard deviation.
Xavier 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 that I have used a gaussian distribution, but it can also be uniform. This method has shown a lot of success for sigmoid and tanh based activation functions.
For ReLU based networks, suggests that the weights should be initialised using $Var(w) = \mathcal{N}(0, \frac{2}{|n_{in}|})$.

2.3 Optimisation Challenges

In addition to the fact that RNNs itself are difficult to train (see Section 1.5), it is commonly the case that the latent variable is often ignored during training.
Ideally, a model that encodes useful information in the latent variable $\overrightarrow{z}$ will have a non-zero KL divergence term and a relatively small cross-entropy term. Straightforward implementations of our model failed to learn this behaviour.
Due to the sensitivity of our model, which is compounded by the sensitivity of the GRU cells in our model, early implementations of our model would initially ignore $\overrightarrow{z}$ and go after the low hanging fruit during the learning process of the decoder. Once this occurs, the decoder ignores the encoder and gradients between the two components become non-existent, preventing learning.
is it often likely that in a degenerative setting, the latent vector in the model would encorporate a lack of relevant information and is essentially ignored. This results in a KL collapse s.t the KL loss reaches zero, and thus the model is then effectively equivalent to an RNN language model.

2.3.1 Teacher Forcing

Teacher Forcing (Williams and Zipser (1989)) is a concept of using real target outputs as each next input in the decoder sequence as opposed to using the decoder's guess. This causes the model to converge faster, but may exhibit instability when the trained network is exploited.
Outputs can be observed when using teacher forced decoders that read with coherent grammar but can wander far from the correct response. Typically, this can be

controlled with a probability p such that the decoder sequence has a p chance of using teacher forcing during training.

2.3.2 KL Annealing

A simple approach by Bowman et al. (2015), it involves adding a variable weight to the KL term in the cost function at training time. Initially, the weight is zero, forcing the model to encode as much information into the latent variable as much 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.

2.3.3 Word Dropout

Also proposed by Bowman et al. (2015), it 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 applied to the input data of a recurrent cell.

2.4 Testing Method

We benchmark the performance of our model against a regular seq2seq network.
her the data produced corresponds to a fake model or from the dataset. This is not used as a loss function (this would consequently create a generative model).
We compare the performance of the VAD against two baselines, a seq2seq model described above and the Transformer.
Models are computed on an 4.4GHz intel 3770k with a GTX 1080 with 8GB VRAM.

2.4.1 Measuring Performance

To understand the quality of our models, we subject them to the BLEU (Papineni et al. (2001)) and ROUGE (Lin (2004)) emperical scores at each epoch to quantify the quality of our responses. These measurements are used alongside the Loss, and human analysis of the results.
BLEU (Papineni et al. (2001)) is originally defined with n-gram precision p_n by summing the n-gram matches for every hypothesis sentence S in the test corpus C. In our particular case, we use unigrams.
Although both equations look very similar, BLEU introduces a brevity penalty term, and also compute the n-gram match for several sizes of n-grams (unlike the ROUGE-n, where there is only one chosen n-gram size).

$$p_n = \frac{\sum_{S \in C} \sum_{ngram \in S} Count_{clip}(ngram)}{\sum_{S \in C} \sum_{ngram \in S} Count(ngram)} \qquad p_n = \frac{\sum_{S \in C} \sum_{ngram \in S} Count_{matched}(ngram)}{\sum_{S \in C} \sum_{ngram \in S} Count(ngram)} \tag{2.12}$$

Figure 2.2: Equations for BLEU (left) and ROUGE (right).

Additionally, we use BLEU to measure the precision performance, i.e. how much the words (and/or n-grams) in the model responses appeared in the responses from the dataset. ROUGE is used in a similar fashion for recall.
Naturally - these results are complementing, as is often the case in precision vs recall. If you have many words from the system results appearing in the human references you will have high Bleu, and if you have many words from the human references appearing in the system results you will have high Rouge."

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 9

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 3

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 8, 15

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 4

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 5

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 10, 11

Dyer, C. (2017). Conditional Language Modelling. original-date: 2017-02-06T11:32:46Z. pages 2

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 3

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780. pages 4

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 8

Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*. arXiv: 1312.6114. pages 5, 6

Le, J. (2018). Recurrent Neural Networks: The Powerhouse of Language Modeling. pages 3

Lin, C.-Y. (2004). ROUGE: A Package for Automatic Evaluation of Summaries. pages 74–81. pages 15

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 15

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 6

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 8, 10

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 15

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *arXiv:1409.3215 [cs]*. arXiv: 1409.3215. pages 7

Williams, R. J. and Zipser, D. (1989). A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, 1(2):270–280. pages 14

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 8, 10, 11