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

Chapter 1

Introduction

Chapter 2

Literature Survey

2.1   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.

2.2   Text Generation

Text generation is a a type of Language Modelling problem, which in itself, is one of the core natural language processing problems, and it used in a variety of contempoary 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 would 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 increase 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.

2.3   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:

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

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

Traditionally, assigning words to probabilities may conflate syntactically dubious sentences but it remains to be a useful method for representing texts.

In particular, we are more interested in conditional language modelling. This slightly differs from the definition described above - A conditional language model assigns probabilities to sequences of words, $w = (w_1, w_2, ..., w_{i-1})$, given a conditioning variable, x.

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

There exists different types of models, but we shall start with the n-gram, argued by Le (2018) as being the most fundamental. An n-gram is a chunk of n consecutive words. 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.

Modern language models revolve around the use of neural networks, which was introduced by Bengio et al. (2001), with a simple MLP that encoded words. The use of neural networks in language modelling is often called Neural Language Modelling, of NLM for short.

Neural Networks are non-linear statistical models that generate complex 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.

3

2.4 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a class of 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. RNNs can XXXXXXXXXXXXXX.
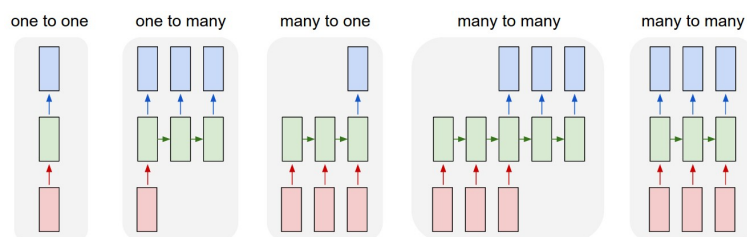

Figure 2.1: Rectangles represent vectors, with red being the input, blue the output, and green representing the state of the RNNs. Arrows represent functions. From left to right: (1) an MLP. (2,3,4,5) are examples of different styles of recurrent neural networks, describing the different types of input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

The architecture of RNNs make it favourable in NLP related problems as words in sentences are typically conditioned on the previous words. Given a sequence of inputs $(x_1, ..., x_M)$, a standard RNN computes a sequence of outputs $(y_1, ..., y_N)$ by performing forward-propagation in a similar to fashion to MLPs, but outputs are chained together as an additional input to other neural networks. This can be visualised best in Figure 2.1. RNNs are trained using backpropagation-through-time.

RNNs showed promise but there existed a multitude of problems. Firstly, 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. Furthermore, they were especially impractical due to the vanishing and exploding gradient problems having a larger effect on training. This is by design, as the architecture of the network does not provide the affordance to represent relationships between arbitary cells in the network.

2.4.1 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) is a type of RNN unit that attempts to retain information based on the previous inputs through the introduction of gated architectures. XXXXXXX

Introduced by Cho et al. (2014), GRUs are used to solve the common issue with LSTMs where the training time was relatively slow due to the number of derivatives necessary to compute. Within the GRU architecture, a feature to retain the previous
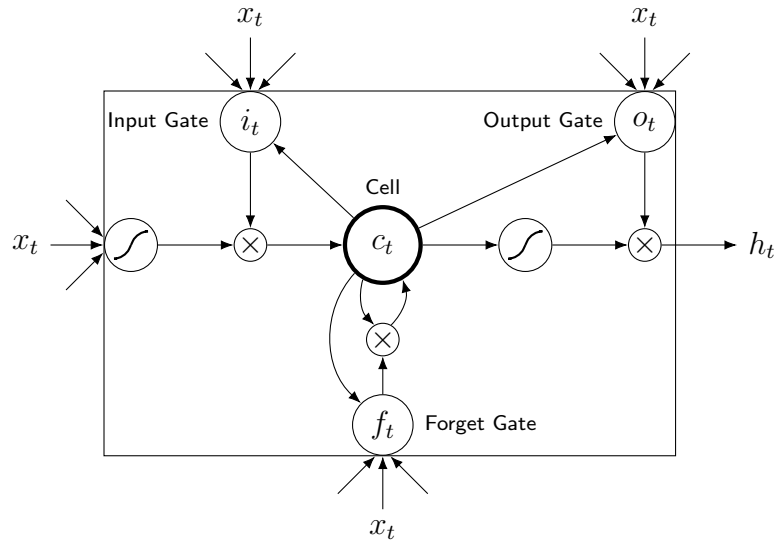
Figure 2.2: A diagram of the LSTM model.

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 between the performance of the two gated units they tested. However, Some research from Chung et al. (2014) suggests that GRUs were found to perform better than LSTMs on smaller datasets.

2.5 Autoencoders

Autoencoders are a specialised form of MLPs where the model attempts to recreate the inputs on the output. Autoencoders typically have a neural network 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.

2.5.1 Measuring Performance

"Bleu measures precision: how much the words (and/or n-grams) in the machine generated summaries appeared in the human reference summaries.
Rouge measures recall: how much the words (and/or n-grams) in the human reference summaries appeared in the machine generated summaries.

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."

2.6   Related Work

2.6.1   Seq2Seq

Seq2Seq, introduced by Sutskever et al. (2014) is a type of neural architecture that models relationships between sequences. In our particular scenario, this could be seen as a modern interpretation of the autoencoder. Seq2Seq comprises of two layers - Encoders and Decoders - which are both are represented with RNNs. This allows the architecture to take in arbitarily sized vectors, and allowing it to output similarly arbitary sized vectors. This flexibility makes it especially adventageous to NLP related problems.
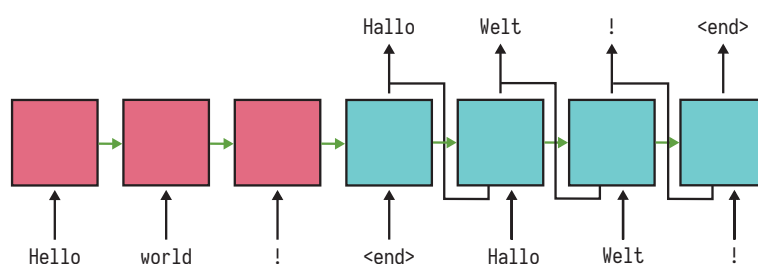

Figure 2.3: An abstracted model of the seq2seq architecture, where the encoder (pink) takes in the input sequence, and the decoder (blue) shows the output sequence.

/ something good about seq2seq.
Traditionally, Seq2seq would be the most common method of tackling this particular problem, but it also presents problems that make it suboptimal. Seq2seq models have shown to be effective in reponse generation, but XXXXX
Furthermore, it was discussed by Jiang and de Rijke (2018) on the lack of diversity in responses. something something.

2.6.2   Seq2Seq with Attention

Further work by Bahdanau et al. (2014) improved the original seq2seq model by providing an attention mechanism. The attention 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
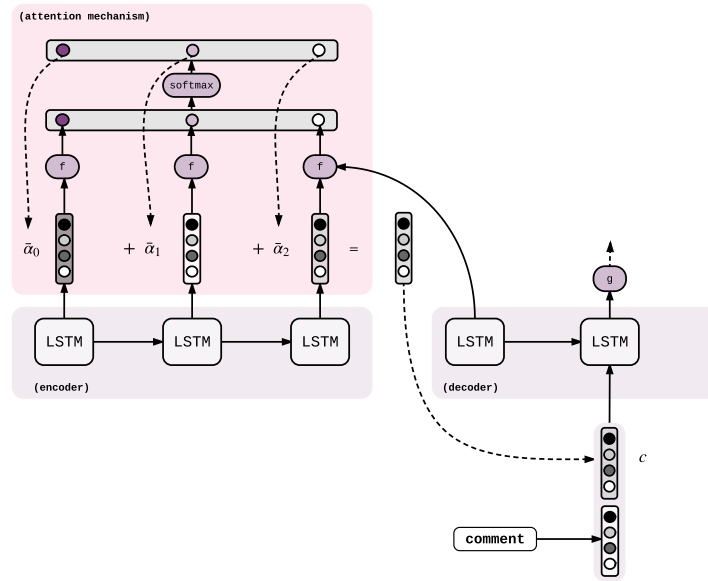
Figure 2.4: An abstracted diagram of the attention mechanism, applied to a seq2seq model.

combination of the two. Bahdanau et al. (2014) have found this to work especially 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.

2.6.3 Transformers

Vaswani et al. (2017) introduced the idea that it is possible to avoid the use of RNNs altogether and focus on leveraging the attention mechanism introduced in seq2seq. The resulting network architecture utilises stacked layers of residual networks for the encoder and the decoder. This adds improvements to computational throughput as sequences can be computed in parallel.

Self Attention

$$Q = X \cdot W^Q$$
$$K = X \cdot W^K \qquad (2.4)$$
$$V = X \cdot W^V$$

Self attention mechanisms involve three components, a Query Q, Key K, and Value V.

$$z = \sigma\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V \qquad (2.5)$$
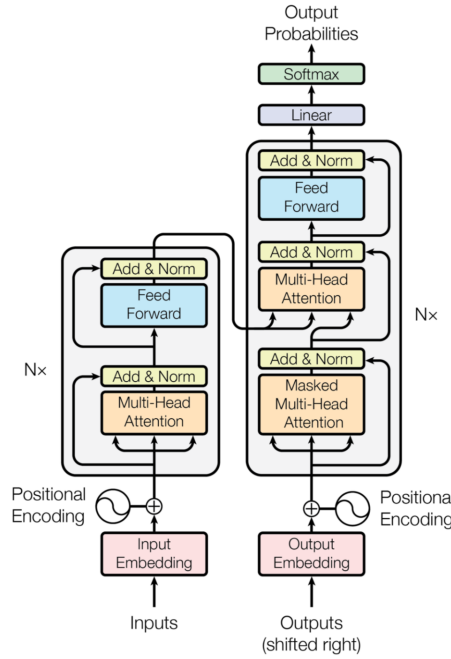
Figure 2.5: The transformer model architecture.Vaswani et al. (2017)

The product of multiple z variables concatenated together, with a weights matrix W^O a multihead attention $\boldsymbol{z}' = [z_1, ..., z_n] \cdot W^O$.

$$\boldsymbol{r} = f_{NN}(\boldsymbol{z}) \tag{2.6}$$

Positionality Hack

Sequential models circumvent the referential ambiguity problems with natural language (such as "the cat is on the mat"), with the inherent properties of time based encodings. This does not necessarily translate well to encoder models as there is no sequentiality. To circumvent this, Transformers encode the positions of words at the embedding level, by adding positional encodings of words.

2.7 Variational Autoencoders

Our proposed solution revolves around the use of Variational Autoencoders. Variational Autoencoders (VAEs) introduce a constraint on the encoding network that forces the model to generate latent vectors that roughly follow a gaussian distribution, as opposed to creating a fixed latent vector. Consequently, VAEs introduce two extra vectors, a mean vector and a standard distribution vector, which is fed from the encoder. A sample is taken from the distribution and that is then fed into the decoder. This consequently allows us to leverage bayesian inference. Model training 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
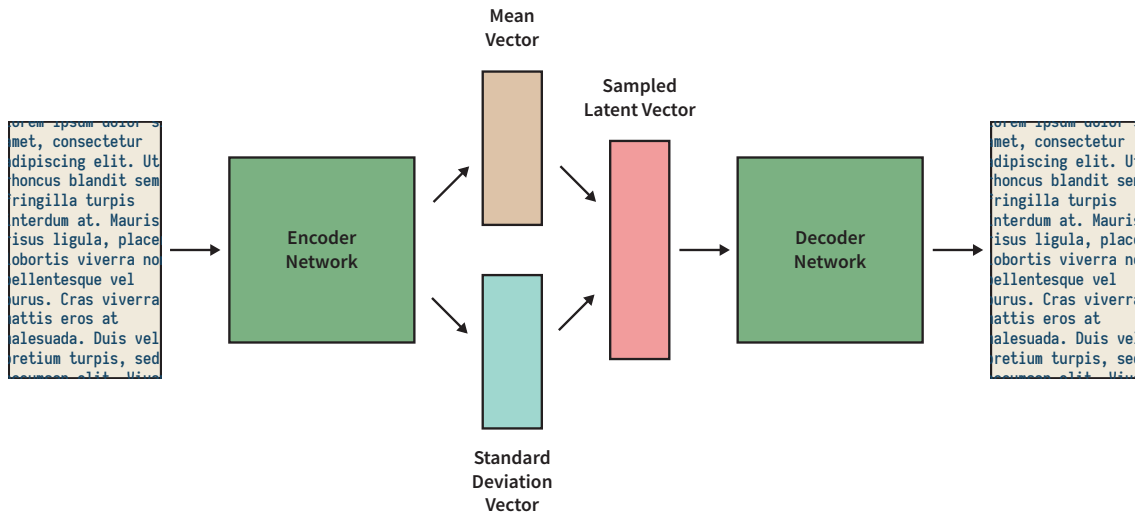
Figure 2.6: A simplified model architecture for a variational autoencoder, which takes as input some text, and it's predicted output being the same text as the input.

should approximate to a standard normal. This stochasticity allows us to model variability in the results.

Due to the stochastic nature of the network, the loss is composed of two components; a reconstruction loss that involves an expection of the output; and a KL divergence, which measures the relative difference of two probability distributions. In this particular case, we will be comparing the distribution of the decoder outputs against a standard gaussian $\mathcal{N}(0, 1)$.

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

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

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

- You'll need to perform a reparameterisation trick (since you can't calculate derivatives of samples.) in order to perform backpropagation. (you can't push gradients through a sampling node.)

$$z = \mu + \sigma \cdot \epsilon$$

where $\epsilon \sim \mathcal{N}(0, 1)$. You want to learn μ, σ.

2.7.1   Conditional Variational Autoencoders

Although VAEs are more robust when compared to their original autoencoder counterparts, 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.
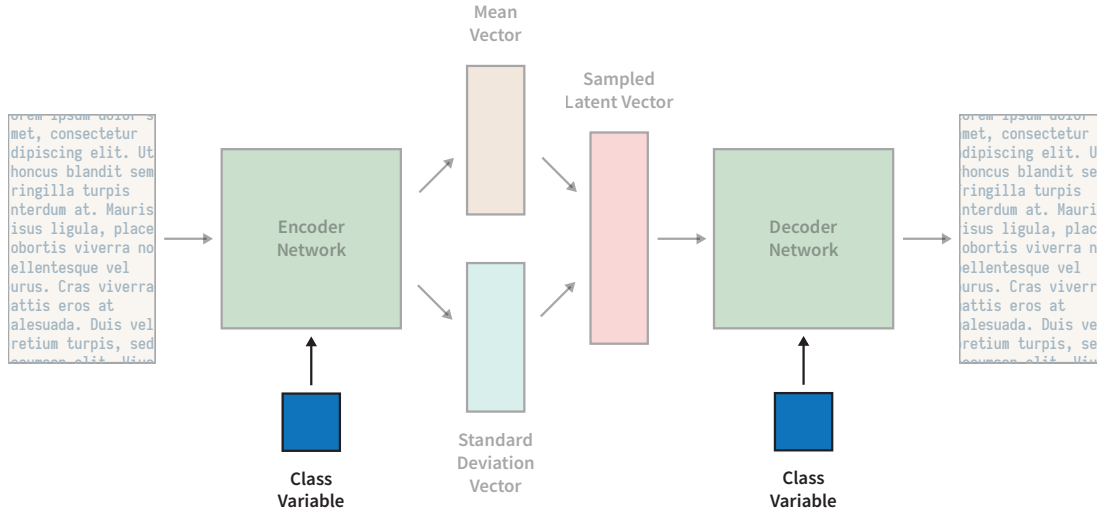

Figure 2.7: A model architecture for a CVAE, which includes the label being fed into the encoder and decoder networks.

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.

2.8   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.

2.8.1 Components

The model is comprised of a variety of components, inspired by the seq2seq network, described earlier.

Encoder

$$
\begin{aligned}
\overrightarrow{h_t^e} &= \overrightarrow{GRU}(x_t, \overrightarrow{h_{t-1}^e}) \\
\overleftarrow{h_t^e} &= \overleftarrow{GRU}(x_t, \overleftarrow{h_{t+1}^e})
\end{aligned}
\tag{2.7}
$$

The encoder works in a similar fashion to the encoder described in the seq2seq network.

Backwards

$$
\overleftarrow{h_t^d} = \overleftarrow{GRU}(y_{t+1}, \overleftarrow{h_{t+1}^d})
\tag{2.8}
$$

Attention

$$
\begin{aligned}
\alpha_{s,t} &= f_{attention}([h_d^e, h_{t-1}^d]) \\
c_t &= \sum_{s=1}^{m} \alpha_{s,t} h_s^e
\end{aligned}
\tag{2.9}
$$

- 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

$$
\begin{aligned}
[\mu^i, \sigma^i] &= f_{infer}([\overrightarrow{h_{t-1}^d}, c_t, \overleftarrow{h_t^d}]) \\
q_\theta(z_t | \boldsymbol{y}, \boldsymbol{x}) &= \mathcal{N}(\mu^i, \sigma^i)
\end{aligned}
\tag{2.10}
$$

Prior Model

$$[\mu^p, \sigma^p] = f_{infer}([\overrightarrow{h^d_{t-1}}, c_t])$$
$$p_\phi(z_t|\boldsymbol{y}_{<t}, \boldsymbol{x}) = \mathcal{N}(\mu^p, \sigma^p) \tag{2.11}$$

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)

$$\overrightarrow{h^d_t} = \overrightarrow{GRU}([y_{t-1}, c_t, z_t], \overrightarrow{h^d_{t-1}}) \tag{2.12}$$

$$p_\phi(y|\boldsymbol{y}_{<t}, \boldsymbol{z}_t, \boldsymbol{x}) = f_{output}([\overrightarrow{h^d_t}, c_t]) \tag{2.13}$$

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^d_t. 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.

Learning Mechanism

VADs use a weighted combination of ELBO and the log likelihood loss of the auxillary objective.

KL Divergence Derivation

As our model involves using multivariate gaussians, we derive the Kullback-Leibler distance such that we can calculate the loss for our distributions. Notice that we retain the means and covariances for our gaussian distributions. 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.

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

$$(2.14)$$

Chapter 3

Contribution

Although these models have been used in NMT related problems, sequence generation is often unexplored within NLP. We propose using models that traditionally perform well in NMT problems and adapt them for use with sequence generation problems.

3.1 Data Augmentation

```
1  {
2    'reviewerID': 'AA8JH8LD2H4P9',
3    'asin': '7214047977',
4    'reviewerName': 'Claudia J. Frier',
5    'helpful': [3, 4],
6    '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!',
7    'overall': 5.0,
8    'summary': 'love it',
9    'unixReviewTime': 1354665600,
10   'reviewTime': '12 5, 2012'
11  }
```

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

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

3.1.1 Embeddings

We use the glove word2vec word embeddings.

3.1.2 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.

3.1.3 Polarity Calculation

This is calculated with:

$$p = \sigma_{tanh}(\frac{h_{pos} - h_{neg}}{h_{pos} + h_{neg}}) \tag{3.1}$$

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

3.1.4 Implementation

Built on PyTorch.

3.2 Testing Method

We'll be using a regressor that determines whether 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.

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 6, 7

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

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 11, 12

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

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 6

Karpathy, A. (2015). The Unreasonable Effectiveness of Recurrent Neural Networks. pages 4

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

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 10

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *arXiv:1409.3215 [cs]*. arXiv: 1409.3215. pages 6

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 7, 8

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 10, 12