# LSTM-VAE FOR SLANG GENERATION

*Martin Skovsbøl Friis s153684, Mathias Niemann Tygesen s153583, Søren Møller Christensen s153571*

Code on https://github.com/MrFriis/NL-Formality-Translation-VAE


## ABSTRACT

Formality used in both written and spoken language often depend on the social context. It is however not always straight forward what level of formality is appropriate to use in a given social situation. Machine learning assitance tools can therefore prove useful in some situations. For example for writing emails, resumés or similar.
In this paper we explore the subject of language formality modelling, specifically for the English language. Our modelling approach is based on a LSTM-VAE generative language model proposed in [1]. We can then get continuous latent space representations of formal/informal sentence pairs using the model. The dataset used in the project has been obtained from Grammarly inc and consists of approximately 120k formal/informal sentence pairs. Using the LSTM-VAE model it is possible to generate new sentences by sampling between the formal/informal sentence pairs. The model produces meaningful sentences between these. We also try adding formality in the latent space with limited success.


## 1. INTRODUCTION

Writing the perfect email is an art form. What is the correct wording? How formal should one be? Questions like these have inspired the company Grammarly to use machine learning in order to help people write emails and résumés with appropriate formality [2]. What Grammerly does can be seen as a transformation/translation model that takes informal english and translates it into formal english. This model only makes the distinction between formal and informal sentences and does not take account for degrees of formality. This raises the question "what lies between informality and formality?".
Variational autoencoders (VAEs) have been used for many natural language modelling purposes. In the 2015 paper "Generating Sentences from a Continuous Space" [1], Bowman et al. showed that a version of the VAE called the LSTM-VAE could be used to construct a continuous latent space from which semantically and syntactical correct sentences could be sampled. In order to be able to sample with varying degrees of formality, we use a LSTM-VAE model to get latent space representations of sentences. The latent space should capture the underlying semantics in each sentence. The goal of the following project is to be able to sample sentences given a sentence prompt with varying degrees of formality, but retain all other original information such as sentence topic and syntax. Therefore latent space should capture the underlying semantics in each sentence.

Our contribution is to introduce the LSTM-VAE presented in the paper to a data set of formal/informal sentence pairs and evaluate how well the model can capture sentence formality and how well it can generate new sentences with varying formality. Furthermore the latent space was explored to examine if a separation between formality and informality existed.

## 2. THEORY

### 2.1. Language Models [3]

The goal of a Language Model (LM) is to assign probabilities to a sequence of words, $p(\mathbf{x})$, $\mathbf{x} = \{x_i\}_{i=1}^N$. To do this we condition the probability of a word on all preceding words:

$$p(\mathbf{x}) = \prod_{n=1}^{N} p(x_n|x_{<n}) \tag{1}$$

A common approach to do this is a long short-term memory (LSTM) model. Through the use of the input gate, forget gate and the output gate it maintains a hidden state of long and short term memory. That way when the model predicts the probability of the current word it is conditioned on all of the previous words in the sentence.
Note also that a language model can predict the next word in a sentence or generate full sentences from a partial sentence seed by selecting the next word with the highest probability $p(x_{n+1}|x_{\leq n})$.

### 2.2. Variational Autoencoder

Variational autoencoders (VAEs) [4] is a class of models that does unsupervised approximate inference on directed probabilistic models. A simple such model could be that we assume that our observed data $X = \{x^{(n)}\}_{n=1}^N$ is generated from some unobserved latent variables $z$. The process is that the latent variable $z$ is generated from some prior distribution

We thank Valentine Lievin for his guidance.

$p_{\theta*}(z)$ and the observation, $x$, is then generated from some conditional distribution $p_{\theta*}(x|z)$. We assume that both distributions come from parametric families of distributions and we wish to infer approximations approximate estimations, $\theta$, of the parameters. Unfortunately exact inference is often intractable.
In order to remedy this we introduce the approximate posterior (or recognition model) $q_\phi(z|x)$. Then through Jensen's inequality we can formulate a lower bound on the marginal likelihood

$$\log p_\theta(x) \geq \mathcal{L}(\theta, \phi; x). \tag{2}$$

where

$$\mathcal{L}(\theta, \phi; x) = -D_{KL}(q_\phi(z|x)||p_\theta(z)) + \mathbb{E}_{q_\phi}[\log p_\theta(x|z)] \tag{3}$$

$\mathcal{L}$ is called the evidence lower bound or ELBO. The first term, the KL divergence, acts as a reguralizer forcing the approximate posterior $q_\phi(x|z)$ closer to the prior $p_\theta(x)$ and the second term is the reconstruction loss. By sampling the reconstruction loss can be approximated and with the reparameterization trick the ELBO and its derivative can be used to update the parameters in $\theta$ and $\phi$ using gradient descent algorithms like SGD or ADAM.

### 2.3. LSTM-VAE [1]

In order to use VAEs in a language model (VAE-LM) setting we modify the VAE slightly. Specifically we change the encoder and decoder to LSTMs. We feed the input sentence directly to the encoder LSTM and propogate the last hidden state through a linear layer to generate a mean and covariance as parameters for the assumed multivariate Gaussian distribution of the approximated posterior $q_\phi(z|x)$. Note here that the used multivariate distributions for $q_\phi(z|x)$ and $p_\theta(z)$ are simply assumptions, see section 7.
For the decoder LSTM we sample the latent variable $z$ and generate the first hidden state through a linear layer. We also add the transformation of $z$ to each time step of the LSTM.
Using these modification the VAE becomes a LSTM-VAE that encodes sentences into a latent space representation and decodes them again using the generative model:

$$p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{n=1}^{N} p(x_n|x_{<n}, \mathbf{z}) \tag{4}$$

Note that in the case where the latent space representation z of the sentence contains no information, the model devolves into the standard LSTM language model (eq. 1) described in section 2.1.

### 2.4. Avoiding Posterior Collapse

As mentioned above when the latent space representations contains no information the model devolves into a LSTM model. This is called posterior collapse. As intuitively explained in [5] this is a prevalent problem for VAE-LMs. The root cause for the problem is that the decoder, being a LSTM model in its own, is too "strong" a model. Hence the decoder can gain decent performance by ignoring the latent space representations. As the optimization is based on the output of the decoder, all improvements from training will be primarily from the decoder. Hence in order to get better latent space representations the model would have to sacrifice performance and base decoding more on the latent space representations. Unluckily this is not how optimization algorithms work so we have to deal with the problem in other ways.
The posterior collapse fixes focus on two things. First they limit the decoder such that it performs worse and learns slower. This way the encoder can keep up and the model is less likely to get stuck in local minima ignoring the encoder. Second they make the encoder cheaper, slacking the cost of the KL divergence between approximate posterior and prior.

#### *2.4.1. Word dropout*

Word dropout is a technique used in LMs to avoid overfitting to the training sentences. Each word in the input sentence has a preset probability to be dropped out and replaced with the <UNK >token:

$$x_n = \begin{cases} x_n, & c > p \\ \text{<UNK>}, & c \leq p \end{cases} \qquad c \sim \mathcal{U}(0, 1) \tag{5}$$

Here $x_n$ is the n'th word in an input sentence. By dropping words in the input sentence, information is hidden from the model. Thus the model is forced to rely on information from the sampled latent variable [1]. This can can be seen by the almost linear increase in KL Divergence as the probability for dropout is increased. This also intuitively aligns with equation 4.

#### *2.4.2. Annealing*

In the case of VAEs, annealing is the practice of reducing the penalty in the ELBO from the KL divergence by adding a weight factor $\beta$ to the ELBO [6]. The ELBO then becomes

$$\mathcal{L}(\theta, \phi; x) = -\beta D_{KL}(q_\phi(z|x)||p_\theta(z)) + \mathbb{E}_{q_\phi}[\log p_\theta(x|z)]. \tag{6}$$

$\beta$ starts as 0 so the ELBO function is purely the marginal log-likelihood as in a regular autoencoder. This allows the model to map sentences into the latent space without a penalty. As the loss function then gradually goes towards the full ELBO (allowing the KL divergence to penalize the model), the sentences will go from discrete values in the latent space to filling it out continuously.

**Linear annealing** is scheduled as

$$\beta = \begin{cases} 1, & e \geq n \\ \frac{e}{n}, & e<n \end{cases} \tag{7}$$

Here $e$ is the current epoch, and $n$ is a predetermined number of epochs the annealing is running.

**Cyclical annealing** [7] is scheduled as

$$\beta = \begin{cases} f(\tau), & \tau \leq R \\ 1, & \tau > R \end{cases} \tag{8}$$

where $\tau = \frac{mod(e-1,[E/M])}{E/M}$ is the epoch number, E is the total number of epochs, M is the number of cycles, R is the proportion of epochs that it takes to go from 0 to 1 within a cycle, and f is a monotonically increasing function. The restarts of the annealing allows the model to put in more information into the latent space for 'free', and then smoothing out the added information gradually.

### *2.4.3. Free bits*

Free bits is a method for putting extra penalization on the ELBO loss function when the prior and posterior are too similar ie. KLD is very small. The penalization is applied by giving the KL term a threshold, $\lambda$, that pulls the KLD up for 'free'. Free bits is given as [6]

$$KL_{fb} = \sum_i max[\lambda, D_{KL}(q(z_i|x)||p(z_i)] \tag{9}$$

Here $\lambda$ is a threshold that is set as a hyper parameter and $z_i$ is the i'th latent variable. Intuitively this also puts a limit on how low the KL term can drop, and therefore forces the model to use the latent space.

### *2.4.4. Active Units*

It has previously been shown that the technique 'active units' is a good metric for how much information that is present in the latent space [8]. It is given as

$$A_{z_i} = Cov_x(\mathbb{E}_{z_i \sim q(z_i|x)}[z_i]) > \delta \tag{10}$$

Where the `'Cov'` operator is the covariance of a latent variable given all test data and $\delta$ is a preset variable that determines when a latent variable is active. We are using the value 0.02 based on Burda et al. [8]

## 3. MODEL

A model with a structure very similar to [1] was used. For an overview of this structure see figure 1. In the preprocessing step (see section 4) we tokenize the sentences using a sentece piece model trained with vocabulary size 16000 or 32000 (see 4) on the entire data set. We cut sequences that contain over 10 tokens. The <bos> and <eos> tokens are added to all sentences. Sentences that contain less than 12 tokens are then padded with <pad> after <eos>. The sequence is reversed before being used as input to the encoder. The encoder first embeds the sentence and uses the embeddings as input to a LSTM layer. The output of the last encoder LSTM cell is used as input to two linear layers, that are used to determine $\mu$ and $\Sigma$ of the approximate posterior distribution, $q_\phi(z|x)$ assumed to be Gaussian distributed. We then sample from the latent dimension as described in 2.2. The sample is projected with a linear layer and set as the first hidden state of the decoders LSTM layer, and z is concatenated to the embedded sentence before being used as input to the LSTM layer.

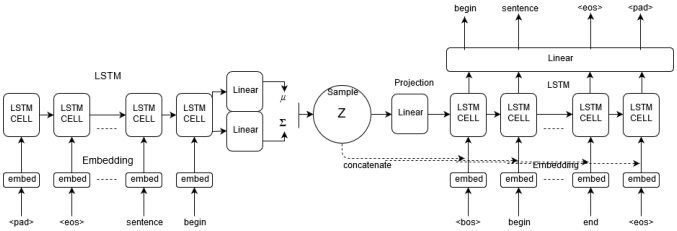


**Fig. 1**. The model that we used for our VAE language model. The model consists of an almost symmetric structure with the output of an embedding layer used as input to a LSTM layer. The output of the encoder is an approximation of $\mu$ and $\Sigma$ used to sample from the latent space, $Z$, as described in section 2.2. The output of the decoder is a sentence generated from the sample.

During training the entire sentence is used as input to the decoders embedding layer with word dropout 25% to be replaced by the <UNK >token. The embedded sentences concatenated with the projected latent variables is then used as input to the decoders LSTM layer. The hidden state of each cell is used as input to a linear layer, that projects the output back to sentence space providing us with the logits for each token in the vocabulary per output of the linear layer. The word with the highest probability is then the chosen one per output.

The way that sentences are generated differ slightly from the way that the VAE-LM is trained. Instead of taking the whole sentence as input in the decoder, the model only generates one word at a time, starting with the token <bos >and the sample from the latent dimension, z . Predicted words are concatenated with the input sentence and used to predict the next word - see section 2.1 for details.

## 4. DATA AND PRE-PROCESSING

### 4.1. Data

In order to train the embeddings and detect the latent direction of slang we use Grammarly's YahooAnswers Formality Corpus (GYAFC) [2].

The data set is made by Grammarly and is based on the Yahoo Answers L6 Corpus [1] which consists of questions and answers from the website *Yahoo Answers*. Since *Yahoo Answers* is a website the questions and answers in the L6 Corpus are all quite informal and filled with slang. Grammarly have then randomly selected 53K informal sentences in the genres *Entertainment & Music* and *Family & Relationships* and added a formal translation of the sentences to form the training data. GYAFC training data consists of $\sim 110K$ labelled formal-informal sentence pairs. An example of such a pair is:

Informal: *if u know what i mean*
Formal: *If you understand me.*

The test set includes an additional 4849 formal-informal sentence pairs.

### 4.2. Pre-processing

In order to use the GYAFC we need to pre-process it. We use *SentencePiece* [2] to tokenize our sentences on a subword level. We hope that using a subword tokenization will improve the models performance on contractions like *don't* and *isn't* as they will be tokenized into multiple tokens (e.g. *do* and *n't*). After tokenization we filter the data to pairs where both the formal and informal sentece contains less than 10 tokens. This is for purely practical reasons as we did not have access to enough compute power to handle all the data[3]. We create 2 data sets: a small and a large. The small data set consists of the tokenized filtered sentences from the *EM* dataset. This set consists of 1284 sentence pairs in the training set and 582 sentence pairs in the test set with a vocabulary size of 16000. The large data is based on both *EM* and the *FR* sets and have 55280 sentence pairs in the training data and 2650 sentence pairs in the test set with a vocabulary size of 32000. Note that Bowman et al. [1] uses a data set of approximately 80 million sentences.

## 5. RESULTS

### 5.1. Comparisons of models

As mentioned in Section 2.4 VAE language models often suffer from posterior collapse. As an information rich latent space is the end goal for our model we wish to combat this. We have therefore implemented 5 different LSTM-VAE models with increasing amount of "tricks" implemented. We have also implemented a LSTM model to compare against. The results for the trained models can be seen in Appendix A. All models have been trained for 600 epochs and suffer from over fitting. See section 7 for more discussion on this. Note that when comparing results all of the VAE models use ELBO as loss function while the LSTM models use cross entropy. As we are interested in the best possible VAE we start with comparing them. We observe that *VAE*, *WD-VAE* and *WD-LA-VAE* all suffer from posterior collapse. This can be seen from the close to zero KL divergence and zero active units. Hence these models did not succeed in adding information into the latent space. If we look at *WD-CA-VAE* and *FB-WD-CA-VAE* we can see that both have non-zero KL divergence and active units. So they both have information in the latent space. As the models have comparable loss and reconstruction but *FB-WD-CA-VAE* have the most active units we decide to go with this model[4]. If we compare *FB-WD-CA-VAE* to *LSTM* we can see that its performance is worse. Since the ELBO is only a lower bound this could potentially be even worse. However looking at the models reconstructions they are comparable. Furthermore both models are only trained for 600 epochs and since the big VAE model is much more complex it is clear that the VAE can obtain the same performance as the the LSTM.

### 5.2. Sentence generation and latent space exploration

#### *5.2.1. Separation of formality in latent space*

One of the goals of the LSTM-VAE is to be able to add slang to a sentence in the latent space. Hence it is interesting to see if formality separates sentences in the latent space. In Figure 2 we can see the latent space representations of 800 sentences generated using the model *WD-CA-FB*.

In the figure we can see that there is an area on the right hand side that have almost only informal representations. We suspect this is all of the overly informal sentences as there is also informal representations in all of the areas with formal representations. This is due to some of the informal sentences in the data set not being that informal. Hence it does not seem like there is any clear separation of sentences based on formality in the latent space. We note that some separation might still be present in the latent space but formality is not responsible for the most variance in the representations as otherwise the T-SNE would have showed that.

In Figure 2 it is not depicted which sentences matches together in pairs. So even though there is no clear separation of formality it could still be that formality had a clear direction. In Figure 3 we plot the directions between formal-informal pairs. Unfortunately it does not seem like formality have a

[1] Available at https://webscope.sandbox.yahoo.com/catalog.php?datatype=l&guccounter=1

[2] The package can be found at https://github.com/google/sentencepiece

[3] On this subset of data training already took $\sim$ 2 hours using a Tesla P100 GPU on Google Colab and we wanted multiple models to compare.

[4] The rest of the analysis for *WD-CA-VAE* can be found in notebooks, it's comparable to *FB-WD-CA-VAE*

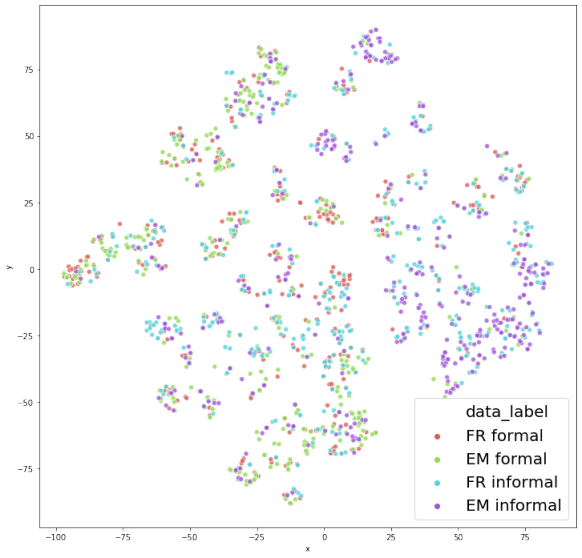


**Fig. 2**. Latent space representations of 800 (200 informal and 200 formal from both the *FR* and *EM* datasets) sentences using the model *WD-CA-FB*. Dimension have been reduced from 64 to 2 using T-SNE.

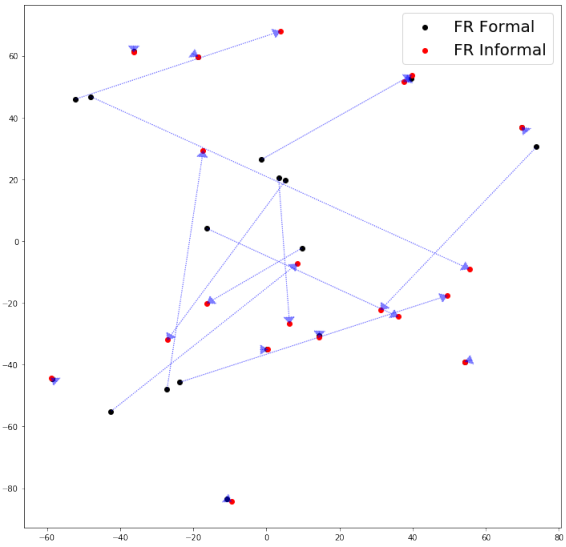


**Fig. 3**. The direction between formal-informal sentence pairs latent space representations using the *WD-CA-FB* model. The sentences come from the *FR* data set. Dimensionality have been reduced using T-SNE

specific direction in the latent space. We also note that we can see from the plot that some formal-informal sentence pairs are placed very closely together while others are far apart in completely different directions. Hence it is clear that formality does not correspond to most of the variance or structure in the latent space. We suspect that there are other features about the sentences that corresponds more to the structure but we will return to that later.

### *5.2.2. Adding slang in latent space*

Even though Section 5.2.1 showed that formality have no clear direction in the latent space we will still try to add informality/slang to a sentence in the latent space.
First we calculate the mean slang direction $\mathbf{v}_{slang}$ by

$$\mathbf{v}^{slang} = \frac{\sum_{(x^{if},x^f)\in D}\left(q(x^{if}) - q(x^f)\right)}{N} \tag{11}$$

where $D$ is the data set, $N = |D|$, $(x^f, x^{if})$ is a formal-informal sentence pair and q is the embedding function given by

$$q(x) = \underset{z}{\operatorname{argmax}}\ q_\phi(z|x)$$

. We can then add slang to a sentence in the latent space by

$$\tilde{z}^{if} = z^f + \mathbf{v}_{slang} \tag{12}$$

where $z^f = q(x^f)$ and then reconstructing the sentence from it's latent representation by the generative function p is given by

$$p(z) = \underset{x}{\operatorname{argmax}}\, p_\theta(x|z)$$

such that $\tilde{x}^{if} = p(\tilde{z}^{if})$

If we calculate $\mathbf{v}^{slang}$ using the *WD-CA-FB* model we get that the mean change over all direction is $0.02$ with a variance of $0.46$. So again, like in Figure 3, it seems like there is no clear formality direction in the latent space. We test this by trying to embed a sentence, add $\mathbf{v}^{slang}$ and reconstruct the sentence. Note we choose a sentence that reconstructs correctly before we move it.

Sentence: *are you hesitant to talk to them.*
Unmoved reconstruction: *are you hesitant to talk to them.*
Moved reconstruction: *are you hesitant to talk to them.*

Here we can see that the unmoved and moved reconstruction is the same. We see a similar behaviour with different sentences:

Sentence: *am i going to believe in him?*
Unmoved reconstruction: *am i going to be rich?*
Moved reconstruction: *am i going to read them?*

Here the unmoved reconstructions is wrong but again we see that the unmoved and moved sentence are not completely the same but very similar. In fact we can see that the starts of the sentences are identical. This shows that representations in the latent space is structured around the starts of sentences[5].

[5]See notebook `Latent_space_exploration_fb_ca_wd.ipynb` for more examples

#### 5.2.3. Generating from continuous space

The project aim is to continuously generate from the latent space between the formal and informal sentence pair. This is done by getting latent space representations for the formal/informal sentence pairs and traversing between them by calculating the intermediate steps by

$$\tilde{z} = (1 - \alpha)z^f + \alpha z_{if}, \;\; \alpha \in [0, 1] \tag{13}$$

where $z_f$ and $z^{if}$ are the representations of the formal and informal sentences in the latent space.

Using the *WD-CA-FB* model we get that most sentence pairs generate coherent sentences in the entire intermediate space. However most of the time the sentences does not match the given end sentences. For example:

| Sentence 1: *the majority of the websites are starting with tamil.* |
|---|
| *the majority of my friends are men as well.* |
| *the majority of my friends are mens.* |
| *the newer questions on yahoo get more activity.* |
| *the most expensive thing was the preacher...* |
| *most of the websites starting with tamil* |
| *most of the websites starting with tamil* |
| Sentence 2: *most of the websites starting with tamil* |

This seems to be because the sentences generated from latent space does not match the embedded sentences. If the end sentences are picked such that they can be reconstructed it works:

| Sentence 1: *are you hesitant to talk to them.* |
|---|
| *are you hesitant to talk to them.* |
| *are you hesitant to talk to them.* |
| *talk to her about what she likes.* |
| *go to mapquest and it should be there.* |
| *call him and talk with him about it.* |
| *call him and talk to him about it.* |
| Sentence 2: *call him & talk to him about it.* |

Here we can see that the model successfully generated meaningful sentences between the target sentences. Furthermore we can see a bit of how the latent space is structured. The first word in the formal sentence is *the* and the first word in the informal sentence is *most*. In the traversal we can see that the generated sentences go from starting with *the* to *a lot* to *most*. Hence it seems like that the latent space is structured around the first word in the sentences. Furthermore we can see that word with similar meaning like *a lot* and *most* a places close to each other.

## 6. CONCLUSION

We have trained multiple LSTM-VAE models on the GYAFC data set. By using cyclic annealing, word dropout and free bits we have successfully avoided posterior collapse and managed to train a LSTM-VAE without posterior collapse. Even though the model suffers from over fitting it can still be used to examine the latent space. Doing this we can see that it seems like some informal sentences are separated from the rest of the sentences but a general separation was not visible.

Using the LSTM-VAE model we have succeeded in sampling valid English sentences from a continuous latent space between two latent space representations of sentences. However in most cases the sampled sentences do not match up completely with the target sentences. We have also tried to calculate a formality direction in the latent space however there were no general direction. We note that the problem of training language models VAEs is hard and that similar models often is trained on much more data and consider this a proof of concept.

## 7. FUTURE WORK

The next step in development of the model would be looking into the over fitting problem. A first step could be to not filter the GYAFC data set on short sentences. This would effectively double the amount of training data. However the full data set is still only 100k which is nowhere near the 80m in [1] so it is probably not enough in itself. Another measure to try would be to add dropout to the network like in [9], however we note that we tried it on a standard LSTM on the data without much impact. During the research phase we also stumbled upon other interesting work that inspired us but that we did not have sufficient time to experiment with. It could be interesting to explore the prior assumption of a Gaussian distribution. Students t-distribution is something that has previously been explored with VAEs [10], and is something that would be worth looking into, even though the reparameterization trick is slightly more complicated. [10]

## 8. REFERENCES


[1] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio, "Generating sentences from a continuous space," 2015.

[2] Sudha Rao and Joel Tetreault, "Dear sir or madam, may i introduce the gyafc dataset: Corpus, benchmarks and metrics for formality style transfer," 2018.

[3] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney, "Lstm neural networks for language modeling," in *INTERSPEECH*, 2012.

[4] Diederik P Kingma and Max Welling, "Auto-encoding variational bayes," 2013.

[5] Toward Data Science, "The variational autoencoder as a two-player game," https://towardsdatascience.com/the-

variational-autoencoder-as-a-two-player-game-part-i-4c3737f0987b, Accessed: 03-01-20.

[6] Bohan Li, Junxian He, Graham Neubig, Taylor Berg-Kirkpatrick, and Yiming Yang, "A surprisingly effective fix for deep latent variable modeling of text," 2019.

[7] Xiaodong Liu Jianfeng Gao Asli Celikyilmaz Lawrence Carin Hao Fu, Chunyuan Li, "Cyclical annealing schedule: A simple approach to mitigating kl vanishing," in *NAACL*, 2019.

[8] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov, "Importance weighted autoencoders," 2015.

[9] Theodore Bluche, Christopher Kermorvant, and Jérôme Louradour, "Where to apply dropout in recurrent neural networks for handwriting recognition?," 08 2015, pp. 681–685.

[10] Najmeh Abiri and Mattias Ohlsson, "Variational auto-encoders with student's t-prior," in *ESANN 2019 - Proceedings : The 27th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2019.

## A. MODEL RESULTS TABLE

| | LOSS | | RL | | KLD | | AU |
|---|---|---|---|---|---|---|---|
| | training | test | training | test | training | test | X |
| LSTM S | 5.76 e-3 | 47.2 e-3 | X | X | X | X | X |
| LSTM B | 11.3 e-3 | 86.4 e-3 | X | X | X | X | X |
| VAE S | 0.0057 | 0.047 | 5.72 e-3 | 47.2 e-3 | 1.11 e-5 | 1.13 e-5 | 0 |
| WD-VAE S | 11.1 e-3 | 39.6 e-3 | 11.3 e-3 | 39.6 e-3 | 0.64 e-3 | 0.63 e-3 | 0 |
| WD-VAE B | 22.0 e-3 | 61.5 e-3 | 22.0 e-3 | 61.5 e-3 | 1.16 e-3 | 1.07 e-3 | 0 |
| WD-LA-VAE S | 10.9 e-3 | 39.9 e-4 | 10.9 e-3 | 39.9 e-3 | 0.21 e-3 | 0.20 e-3 | 0 |
| WD-LA-VAE B | 24.0 e-3 | 56.0 e-3 | 24.0 e-3 | 56.0 e-3 | 0.321 e-3 | 0.317 e-3 | 0 |
| WD-CA-VAE S | 8.68 e-3 | 36.9 e-3 | 8.68 e-3 | 36.9 e-3 | 3.66 | 3.69 | 3 |
| WD-CA-VAE B | 14.7 e-3 | 55.4 e-3 | 14.7 e-3 | 55.4 e-3 | 6.33 | 6.27 | 6 |
| WD-CA-FB-VAE S | 8.78 e-3 | 36.6 e-3 | 8.78 e-3 | 36.6 e-3 | 8.92 | 8.89 | 64 |
| WD-CA-FB-VAE B | 14.5 e-3 | 53.9 e-3 | 14.5 e-3 | 53.9 e-3 | 11.1 | 10.9 | 64 |

**Table 1**. Loss, Reconstruction loss and KL-divergence for different models using word dropout (WD), linear annealing (LA), cyclic annealing (CA) and free bits (FB). S denote the model has been trained on the small data set and B denotes the model has been trained on the large data set (see Section 4). All models have been trained for 600 epochs. Note that all models suffer from overfitting.