# TEXT GENERATION WITH VARIATIONAL AUTOENCODERS

*Martin Hatting Petersen (s144234)*

Technical University of Denmark
DTU Compute

## ABSTRACT

This paper investigates different architectures for text modelling with the use of a Variational Autoencoder. I will train 5 different models and find out what is the best model for each of the two datasets that I have provided. I have a small toy-dataset to select which models should be trained on the larger twitter dataset. It turns out that the best performing model is the Ladder CNN VAE with 5 latent variables. But the results also shows that the hyperparameters are really important especially the $\beta$ parameter and the number of latent variables.

My implementation can be found at: Github

***Index Terms***— Variational Autoencoders, Hybrid Convolution, Ladder VAE, Text Generation

## 1. INTRODUCTION

In this paper I will introduce five different architectures for the recently proposed variational autoencoder (VAE) model by [1]. The model learns parameters for the latent distribution which is sampled from and thereafter reconstructed to be similar to the original input. The methods presented in this paper is focused on textual data but can be extended to other purposes.

The variational autoencoder tries to find the optimal choice of parameters such that $p(\mathbf{x})$ is maximized. After some calculations we arrive at the following lower bound that we try to maximize (all the derivations are shown in the appendix):

$$\mathcal{L}(\theta, \mathbf{x}) = E\left[\log(p(\mathbf{x}|\mathbf{z})\right] - KL[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})] \leq \log(p(\mathbf{x}))$$

But as discovered in [2] adding a slowly increasing parameter $\beta$ (from 0 to 1) enables the models to learn a better reconstruction and afterwards learning a latent space that ensures this while being close to the priors. This changes the loss to be:

$$\mathcal{L}(\theta, \mathbf{x}) = E\left[\log(p(\mathbf{x}|\mathbf{z})\right] - \beta KL[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})] \quad (1)$$

Thanks to Ole Winther and Valentin Liévin for guidance.

In my experiments I use another range for $\beta$ $(0, \beta_{\text{max}})$ instead of just the usual range $(0, 1)$ to have more control of the $\beta$ parameter.

## 2. MODELS AND COST FUNCTIONS

In this section I will go through the models explored in this paper. The two first models use the same cost function namely the one found in eq. (1) and the others will explained as I reach those models. Another thing to notice is that all models except the Ladder model uses a single latent layer with the posterior set to $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ and the approximate posterior $q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mu, \sigma^2\mathbf{I})$ where $\mu$ and $\sigma$ both are parameters learned from the data. When I arrive at the Ladder model its posterior will be explained further.

### 2.1. RNN model

This model is the classic RNN encoder-decoder architecture as described in [3]. And is illustrated in fig. 1. The encoder is colored in red and the decoder in blue. The black node in the middle illustrates the latent layer where we draw samples from.

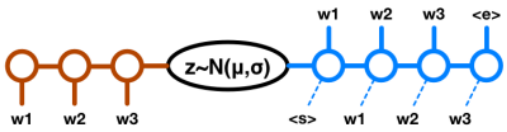


**Fig. 1**. RNN VAE model from [3]. Illustration from [4].

### 2.2. CNN model

Now I introduce the CNN encoder-decoder model from [4] which can be used for text data as well as the RNN model. One major advantage of this model is the computational efficiency on GPU's today with convolutional networks and therefore this model is trained much faster than its RNN counterpart. An illustration of this model is shown in fig. 2.

The only change from before is that now the decoder is green instead of blue.

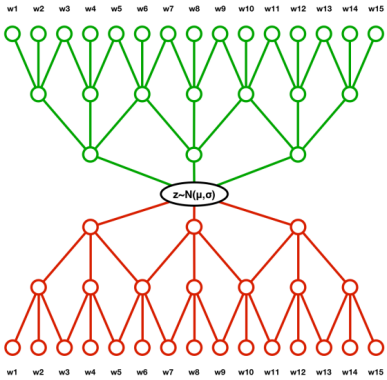


**Fig. 2**. CNN VAE model from [4] as well as illustration.

In this illustration the model consists of 3 convolutional layers in the encoder as well as the decoder and a latent layer in the middle.

### 2.3. Hybrid model

This model is the final one introduced from [4] which uses the ideas from both the CNN model and the RNN model. Because even though the CNN model is much faster to train it still doesn't really capture the spatial relationship between the words in the sentence as well as the RNN model. Therefore in this model we add the RNN decoder to the CNN model as shown in fig. 2. And the added model looks like in the following fig. 3.


**Fig. 3**. Hybrid VAE model from [4] as well as the illustration.

Where the blue nodes are the RNN decoder on top of the CNN model (the green nodes). One other thing they added to this model was a modified loss function that instead of just having eq. (1) we have the modified:

$$\mathcal{L}_{\text{hybrid}}(\theta, \mathbf{x}) = \mathcal{L}_{\text{vae}}(\theta, \mathbf{x}) + \alpha E_{\text{aux}}\left[\log(p(\mathbf{x}|\mathbf{z})\right] \quad (2)$$

Where $E_{\text{aux}}\left[\log(p(\mathbf{x}|\mathbf{z})\right]$ is the loss when compared to the output of the convolution layer just before the RNN part of the decoder. $\alpha$ is just a hyperparameter determining the weight of this loss. This loss enables the model to put more weight on the convolutional part of the decoder and thereby using the latent space even more.

### 2.4. Ladder model

In this final model that differs a lot from the previous model by having multiple stochastic layers instead of just a single one (previous called the latent layer). This model uses another latent prior and approximate prior since this model uses multiple latent variables which are connected. The idea behind the model is that instead of having a single latent variable capture a really complex distribution and then forcing it to be close to a standard Gaussian distribution we have multiple Gaussian distributions and then use another prior for each of them such that they are coupled together. Therefore the priors are now given as:

$$\begin{aligned}
p(\mathbf{z}) &= p(\mathbf{z}_L)\prod_{i=1}^{L-1} p(\mathbf{z}_i|\mathbf{z}_{i-1}) \\
p(\mathbf{z}_L) &= \mathcal{N}(\mathbf{z}_L|0, I) \\
p(\mathbf{z}_i|\mathbf{z}_{i+1}) &= \mathcal{N}(\mathbf{z}_i|\mu_{p,i}(\mathbf{z}_{i+1}), \sigma^2_{p,i}(\mathbf{z}_{i+1})), \quad i = 1, \ldots, L-1 \\
p(\mathbf{x}|\mathbf{z}_1) &= \mathcal{N}(\mathbf{x}|\mu_{p,0}(\mathbf{z}_1), \sigma^2_{p,0}(\mathbf{z}_1))
\end{aligned}$$

This means that we can now find the approximate priors and then use those to optimize the variational lower bound as before with just more latent variables and some changed priors. The approximate priors is given as:

$$\begin{aligned}
q(\mathbf{z}|\mathbf{x}) &= q(\mathbf{z}_1|\mathbf{x})\prod_{i=2}^{L} q(\mathbf{z}_i|\mathbf{z}_{i-1}) \\
q(\mathbf{z}_1|\mathbf{x}) &= \mathcal{N}(\mathbf{z}_1|\mu_{q,1}(\mathbf{x}), \sigma^2_{q,1}(\mathbf{x})) \\
q(\mathbf{z}_i|\mathbf{z}_{i-1}) &= \mathcal{N}(\mathbf{z}_i|\mu_{q,i}(\mathbf{z}_{i-1}), \sigma^2_{q,i}(\mathbf{z}_{i-1})), \quad i = 2, \ldots, L
\end{aligned}$$

And to combine the information from the prior and the approximate Gaussian we use the following equations to form the ladder network:

$$\begin{aligned}
q(\mathbf{z}|\mathbf{x}) &= q(\mathbf{z}_L|\mathbf{x})\prod_{i=1}^{L-1} q(\mathbf{z}_i|\mathbf{z}_{i-1}) \\
\sigma_{q,i} &= \frac{1}{\hat{\sigma}^{-2}_{q,i} + \hat{\sigma}^{-2}_{p,i}} \\
\mu q, i &= \frac{\hat{\mu}_{q,i}\hat{\sigma}^{-2}_{q,i} + \hat{\mu}_{q,i}\hat{\sigma}^{-2}_{p,i}}{\hat{\sigma}^{-2}_{q,i} + \hat{\sigma}^{-2}_{p,i}} \\
q(\mathbf{z}_i|\cdot) &= \mathcal{N}(\mathbf{z}_i|\mu_{q,i}, \sigma^2_{q,i})
\end{aligned}$$

The model described above is from [2] and uses linear layers with a non-linear activation to model the parameters of the latent variables and is visualized in fig. 4.

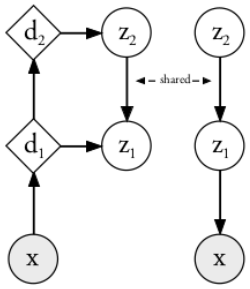


**Fig. 4**. Ladder VAE model from [2] as well as the illustration.

The squared nodes are the deterministic layers determining the latent variable parameters and the circles are the latent variables. In this model you can however use as many latent variables as you like but this just shows the model with 2 latent variables. Please note that I'm actually not using exactly this model as I have added a reconstruction layer just before we get from the last stochastic layer to the reconstructed input. In my experiments I used the RNN decoder from earlier.

## 3. DATASETS

### 3.1. Toydataset

This dataset is a simulated dataset that consist of sine waves to test the models and see which should be tested on the larger twitter dataset. This means that only the best models will be trained/run on the twitter dataset. A small visualization of some samples from this dataset is seen below in fig. 5.

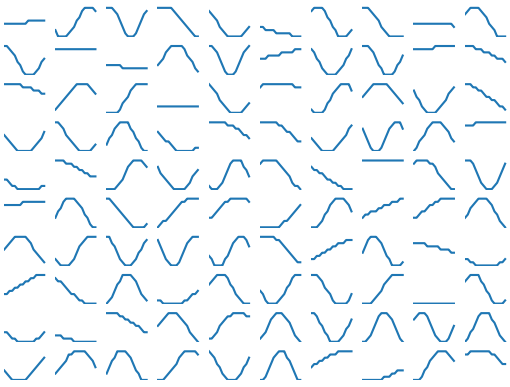

**Fig. 5**. 100 samples from the dataset.

### 3.2. Twitter dataset

This dataset is more complex as it contains more diverse input and also more classes. Some example sentences are given below to illustrate how diverse they might be:

> *without voice, cant even talk.. or sing*

> *To be ill is so boring !!!! Want to go to work tomorrow*

> *Waiting room at the dentist. Theres a man in here with very sad eyes and its kinda breaking my heart*

> *@user sorry, that quot;all entries become property of FOEquot; condition on photos entered in the contest doesnt fly here*

One thing to notice is that the sentences are not of the same length and therefore a padding character is added to the sentences when fed to the model. When I use this dataset for training I will be using the characters and not whole words. This will not perform as well as a word model but will be a lot smaller since a word model would have to keep the whole vocabulary and this only needs to keep all the letters and symbols.

## 4. FURTHER DEVELOPMENTS

In this section I will go through some changes that I have made on the models to further improve their ability to reconstruct from the latent space to realistic samples.

### 4.1. Using a feature map latent space

One of the more advanced things I did was to change the dense latent space into a feature map. This is more easily explained accompanied by the following illustration fig. 6 where I have changed the dense latent space of the RNN VAE into a feature map representation instead.

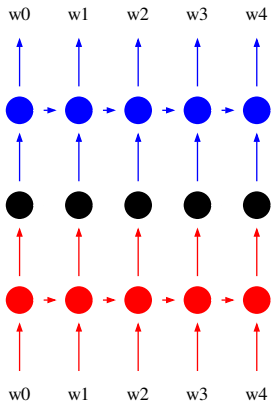


**Fig. 6**. The RNN VAE model but with a feature map latent space.

I also did the same thing to the CNN and Hybrid model such that each black dot is a latent variable that samples from a Gaussian described by the parameters computed by their respective encoders. The idea behind this is to enable the latent space to preserve some spatial relationship between the inputs. When I compute the KL-term I still use the normal prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ and the approximate posterior $q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mu, \sigma^2\mathbf{I})$. But I sum the KL-terms together for each of the latent variables such that I penalize the model for having many latent variables and thereby forces the model to not just copy the input and feeding it through to the output.

### 4.2. Change deterministic layers in Ladder VAE

I also experimented with the Ladder VAE architecture to see if the model would improve by using other layers than linear layers with a non-linear activation. So I tried with a convolutional layer instead to see if that would capture some spatial relationship for each of the stochastic layers. One thing to notice is that in the decoder the convolutional layers is actually transposed convolutions such that we upsample instead of downsample. The remaining part of the model is the same as well as the loss function and such.

## 5. RESULTS

### 5.1. Toydata results

Below in table 1 is the results for the different models tested on the Toy dataset presented earlier. The models that perform the best will be trained and tested on the twitter dataset.

| Model name | ELBO | Reconstruction | KL |
|---|---|---|---|
| RNN (DLS) | -32.1391 | -0.5785 | 31.5606 |
| CNN (DLS) | **-30.3672** | -0.5629 | **29.8043** |
| Hybrid (DLS) | -40.5706 | -0.6674 | 39.9032 |
| RNN (FMP) | -57.5177 | -0.4392 | 57.0785 |
| CNN (FMP) | -38.3562 | -0.6751 | 37.6811 |
| Hybrid (FMP) | -47.1217 | -0.4559 | 46.6658 |
| LVAE (3) | -40.6343 | **-0.0471** | 40.5872 |
| LCNNVAE (3) | -32.4996 | -0.5746 | 31.9250 |

**Table 1**. Results of the model trained on the toy dataset. The numbers marked with bold is the best result in each of the loss parts. DLS: Dense latent space, FMP: Feature map latent space and the numbers in the parenthesis is the number of latent variables.

Please notice that not all the models are of the same size but approximately (the smallest is around 750.000 parameters and largest is 1.000.000).

### 5.2. Twitter results

In the following table 2 the results for the twitter dataset is presented. The best performing model will also be used later to generate some samples and see how realistic they look.

| Model name | ELBO | Reconstruction | KL |
|---|---|---|---|
| Hybrid (DLS) | **-7.0757** | -1.8679 | **5.2078** |
| RNN (FMP) | -54.4468 | -0.5846 | 53.8622 |
| CNN (FMP) | -48.1877 | -0.5796 | 47.6082 |
| Hybrid (FMP) | -58.839 | -0.5022 | 58.9239 |
| LVAE (3) | -72.8252 | -0.5495 | 72.2757 |
| LCNNVAE (3) | -53.4142 | **-0.4337** | 52.9805 |
| LVAE (5) | -69.3656 | -1.5351 | 67.8305 |
| LCNNVAE (5) | **-42.2702** | -0.8711 | **41.3991** |

**Table 2**. Results of the model trained on the twitter dataset. The numbers marked with bold is the best result in each of the loss parts. DLS: Dense latent space, FMP: Feature map latent space and the numbers in the parenthesis is the number of latent variables.

Please notice again that not all the models are of the same size (the smallest is around 1.000.000 parameters and largest is 1.800.000 I noticed this was really large to late and didn't have the time to rerun the model and second largest is only around 1.400.000).

## 6. CONCLUSION

In this paper I have discovered how hard it is to train a variational autoencoder. Both getting a good latent representation and getting a good reconstruction is pretty hard and a nice balance is hard to achieve. It's hard to get the right hyperparameters for the both the encoder and decoder, the right number of layers and both the size of the latent space and the number of latent variables in the case of the ladder network. One particular hyperparameter that is crucial getting right is $\beta$ even though I use $\beta$ annealing I use another variant namely the one in the range $(0, \beta_{\text{max}})$ instead of just the usual range $(0, 1)$.

I have also discovered that the ladder networks is really good and outperforms all other models on complex latent representations but on simpler datasets the network simply becomes intractable and the simpler models outperform this. Another thing that I discovered was that using the convolution layer instead of the fully-connected layer as the deterministic layers in the ladder network improved the model a lot.

My results suggest that using the Hybrid VAE might be best if you only look at the ELBO loss. But since this model also gets a much worse reconstruction loss as apposed to the other mdoels I would consider the best model to be the one with the best weighting between the reconstruction loss and the KL loss. Since this would give a far better model.

I would therefore consider the more complex LCNNVAE model on more complex data to be preferable but finding the correct number of latent variables and the size of the layers is crucial. In my best performing model for the ELBO loss I used 5 latent variable with the following sizes:

- hidden layers: $[512, 256, 128, 64, 32]$
- latent layers: $[128, 64, 32, 16, 8]$
- reconstruction layer: $256$

But further experiments to finding the correct hyperparameters are definitely needed.

Below is some samples generated from the LCNNVAE:

I miss sourtin me to when a se
@usermaea. Toet******* iut ***
I's aygny out Itarttell tore
@user misll u sayie s hare..
I tred**********

**Table 3**. Samples generated by the LCNNVAE model by Gaussian noise

Keep in mind that the data that I trained on is the characters and not whole words which would have made the model learn a lot more realistic looking sentences. But I think its pretty cool that it learns some actual words and the first sentence actually looks like a real sentence.

## 7. FUTURE WORK

This section describes what would have been done if time allowed. It includes further hyperparameter investigation and some changes to the Ladder VAE and other ideas for future work.

### 7.1. Hyperparameter investigation

To have shown even better models and performance I would have liked to do some hyperparameter optimization. This might show that some methods would perform even better. As I would have expected that the Ladder VAE had performed better than the Hybrid VAE.

### 7.2. LadderRNN VAE

One thing I think would really have given a boost in performance would be using a RNN instead of a CNN as the deterministic layers for the Ladder network. The reason behind this is that the RNN explores that the data is sequential and therefore might perform even better than the CNN layers.

### 7.3. GMM VAE

Another really interesting thing to work with would be to replace the prior of the Hybrid/CNN or RNN VAEs with a Gaussian Mixture model instead of the more simple Gaussian. This would enable the model to learn a more complex latent space that might improve the models even more. You just have to derive the right prior and only add one more variable for the encoder to find namely the mixture component and then it's all ready.

## 8. REFERENCES


[1] Diederik P Kingma and Max Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013, https://arxiv.org/abs/1312.6114.

[2] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther, "Ladder variational autoencoders," *arXiv preprint arXiv:1602.02282*, 2016, https://arxiv.org/abs/1602.02282.

[3] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio, "Generating sentences from a continuous space," *arXiv preprint arXiv:1511.06349*, 2015, https://arxiv.org/abs/1511.06349.

[4] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth, "A hybrid convolutional variational autoencoder for text generation," *arXiv preprint arXiv:1702.02390*, 2017, https://arxiv.org/abs/1702.02390.

## 9. APPENDIX

The variational autoencoder as introduced in [1] introduces the following optimization objective (please note that all the expectations is with respect to $q(\mathbf{z}|\mathbf{x})$):

$$p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{z}) dz = \int p(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) dz$$

Now I introduce a term $q(\mathbf{z}|\mathbf{x})$ namely the approximate posterior and add that to the equation.

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) \frac{q(\mathbf{z}|\mathbf{x})}{q(\mathbf{z}|\mathbf{x})} dz = \int q(\mathbf{z}|\mathbf{x}) \frac{p(\mathbf{x}|\mathbf{z}) p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} dz$$

If we now take the log of both sides this yields:

$$\log(p(\mathbf{x})) = \log \left( \int q(\mathbf{z}|\mathbf{x}) \frac{p(\mathbf{x}|\mathbf{z}) p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} dz \right)$$

By using Jensen's inequality we can now find the lower bound:

$$\log(p(\mathbf{x})) \geq \int q(\mathbf{z}|\mathbf{x}) \log \left( \frac{p(\mathbf{x}|\mathbf{z}) p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \right) dz = E \left[ \log \left( \frac{p(\mathbf{x}|\mathbf{z}) p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \right) \right]$$

And this can be rewritten into:

$$E \left[ \log \left( \frac{p(\mathbf{x}|\mathbf{z}) p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \right) \right] = E \left[ \log(p(\mathbf{x}|\mathbf{z}) \right] - E \left[ \log \left( \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})} \right) \right]$$

We now realize that this can be written in terms of the Kullback-Liebler divergence:

$$E \left[ \log(p(\mathbf{x}|\mathbf{z}) \right] - E \left[ \log \left( \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})} \right) \right] = E \left[ \log(p(\mathbf{x}|\mathbf{z}) \right] - KL[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})] \leq \log(p(\mathbf{x}))$$

Where $E \left[ \log(p(\mathbf{x}|\mathbf{z}) \right]$ is the log-likelihood (in my case the cross entropy) and $KL[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})]$ is the Kullback-Liebler divergence between the approximate posterior and the true posterior. Therefore the final loss/cost function is found to be:

$$\mathcal{L}(\theta, \mathbf{x}) = E \left[ \log(p(\mathbf{x}|\mathbf{z}) \right] - KL[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})]$$