# Review for "A Recurrent Latent Variable Model for Sequential Data", Chung et al. 2015

Sampreet Arthi

August 8, 2019

## 1 Introduction

This paper aims to incorporate the principle of VAEs into RNNs to increase the predicting power of the RNNs. The authors believe that natural sequential data has a lot of variability involved. For example, a single change in a word in a sentence can drastically change its meaning. What this means is that we need to be able to map a small variation in our input X to large changes in our output function. This cannot be modelled by RNNs because they have limited variability. To understand all this, let's start by understanding what these latent variables really are. Following are a few concepts we will need to understand first.

1. Latent Variables:
   These are actually variables that cannot be observed directly. Latent random variables are latent variables which follow some probability distribution.

2. Recurrent Neural Networks:
   Henceforth referred to as RNNs, they are a kind of Neural Network which are useful for sequential data, especially when the input size and/or the output size are not fixed. We'll delve a bit into them in Section 2.

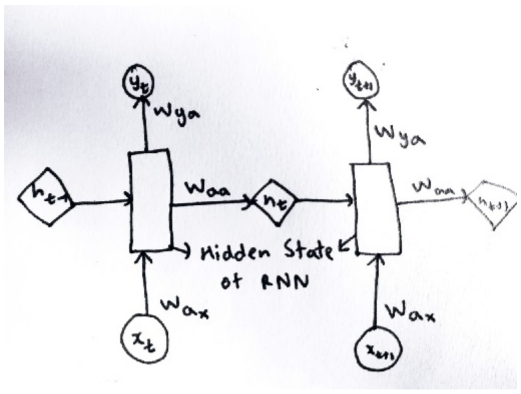
3. Autoencoders:
   They are a sort of Neural Networks. They have an encoder and a decoder. The encoder converts the input into a denser representation by moving irrelevant data and is stored in a layer with lesser neurons. The decoder then rebuilds the input to give an output as close as possible to the original image. More on this in Section 3. We will also discuss Variational Autoencoders over there (VAEs)

## 2 Recurrent Neural Networks

Recurrent Neural Networks have a simple yet powerful idea behind them. They are the same as Feed Forward Networks (FNNs) except they have a small difference. The information of the Hidden Layer is passed on to the next layer of the RNN. For example, if we pass a sentence to am FNN, the output would be of fixed length and so would the input. The output would be computed wordwise from the input and one word would not affect the output of the other. But this does not make any sense. What RNNs do is provide the ability to pass on knowledge from one word to the other. A brief illustration of an RNN follows:
As you can see, the hidden state passes on its information to the next. The governing function behind this update is

$$h_t = f_\theta(x_t, h_{t-1}) \tag{1}$$

where $t$ is the timestep, $x_t \in \mathbb{R}^d$ are the input values and $h_t \in \mathbb{R}^p$ is the hidden state. Here $f$ is a deterministic non-linear transition function and $\theta$ is



the parameter set for the function. $f$ is often implemented using LSTMs or GRUs which are known as gated activation functions. These functions are not entirely relevant for the understanding of the paper so we won't cover it here. What we do need to note is the next two equations which are the only source of randomness for the RNN. We map the hidden state $h_{t-1}$ to the output values.

$$p(x_t|x_{<t}) = g_\tau(h_{t-1}) \tag{2}$$

$$p(x_1, x_2, x_3 ... x_T) = \prod_{t=1}^{T} p(x_t|x_{<t}) \tag{3}$$

Every $p(x_t) = p(x_t|x_{t-1})$ as a result of which Eq. (3) is true. Here, $g$ is a probability distribution for the output function with parameters $\tau$. Because $f$ is entirely deterministic, $g$ is the only function that can cause variability. Here we often use Gaussian Mixture Models (GMMs) which are basically summations of multiple simple Gaussians. These are used to support multimodality and add complexity to the output function. It is much more useful to use GMMs than use a simple Gaussian which is very limited. This function $g$ is not as functional to use so we split it into two parts.
$\phi_t = \varphi_\tau(h_{t-1})$ represents the parameter set of our output function corresponding to timestep $t$ or corresponding to hidden state $h_{t-1}$
$p_{\phi_t}(x_t|x_{t-1})$ represents the actual probability distribution of the output.

Now, even though we use a GMM to represent our output function, it is not good enough to represent the high variability in data like speech or handwriting. These data have what is called a high signal-to-noise ratio meaning that almost the whole data is relevant and needed and not noise. Sentences only make sense when they're read whole. This means that for small changes in $x_t$ there needs to be a large change in $h_t$. The need for highly structured output has been recognised and several solutions have been given. What this specific paper is trying to achieve is to incorporate the VAE principle into every timestep of the RNN.

## 3 Variational Auto Encoders(VAEs)

### 3.1 Basic Autoencoders

Auto Encoders are a kind of Neural Networks which compress and decompress data. The following image shows the basic model of an autoencoder

There is an encoder which is often a CNN which maps the large input into a hidden layer with lesser neurons therefore attempting to store only the relevant information. This is known as downsampling. We represent the same data in a more compact form. That helps in applications like dimensionality reduction. This hidden layer is then read by the decoder which tries to reproduce the exact input with the same dimensions as the input. This is known as upsampling. We try to increase the resolution of data. This can be used to create higher resolution images from lesser quality images. Then the network is trained by applying backpropogation using what is known as the reconstruction loss.

There are several uses of Autoencoders which include image compression, image classification, anomaly detection and denoising. One pretty interesting usage that Google is currently developing is for cloud storage. They are trying to apply this to encode images and save them on the cloud in a low resolution. The local device which will later be used to view that image will have a decoder which will reproduce the image with the natural resolution. However, here we discuss only one specific kind of autoencoder, the variational autoencoder.

### 3.2 How a VAE works

Variational Auto Encoders are more powerful than regular autoencoders because they incorporate a probability distribution in their hidden state. It is not easy to produce variability using a small number of neurons so what it does is use a new set of values, say $z$ in the form of latent random variables to *capture the variations* in the input $x$. We can represent a joint distribution as such:

$$p(x, z) = p(x|z)p(z) \tag{4}$$

Now here, we use a simple gaussian to represent the probability distribution of z. $p(z)$ is known as the prior over latent random variable $z$. The conditional likelihood $p(x|z)$ is what we want to find in the case of a VAE as opposed to $p(x)$. This function is modelled or parametrised by a function approximator like a neural network. The VAE is unique in that it uses a neural network to model Eq. (4). Now what this does differently from a normal autoencoder is that the hidden state is replaced with a probability distribution with the mean $\mu$ and the variance $\sigma^2$ of the latent variable $z$.

Here, we since we use a probability distribution to forward propogate, it is slightly different from regular neural networks. This also means $p(x|z)$ is not differentiable so you can't backward propogate directly. Following is the mathematical explanation:

We use the log likelihood function to represent the reconstruction loss which measures how close the output is to the input.

$$Loss = \sum log\ p_\theta(x) = \sum log\ p(x|z)p(z)) = \mathbb{E}_{z \sim p(z)}(p(x|z)) \tag{5}$$

This shows that the loss is equal to the expectation value of our conditional likelihood $p(x|z)$ given that $z$ follows the distribution $p(z)$. Now since $p(x|z)$ is not an actual function, we can't integrate it to find its expectation instead we sample it and approximate the value by sampling $z$ and computing it via the neural network. Now this means that if we want to perform backward propogation on this operation, we need to partially differentiate it with respect to the parameters involved in $p(z)$. But this is not possible as the operation is sampling.

The backward of $p(x|z)$ is essentially the posterior $p(z|x)$ which is hard to understand or rather compute. So we approximate it with a simple gaussian which we'll call $q(z|x)$. Now this will have two parameters, the gaussian's mean and variance. As the following image shows, we'll have to use our neural network to simultaneously train both $p(x|z)$ which is our generative model and train $q(z|x)$ which is our inference model. In other words, we can saw that $q(z|x)$ is our encoder as given a value of z, we can find out what values of x would be needed and we can say that $p(x|z)$ is our decoder as we can find out the values of x needed to reproduce the input given values of z[2]. Now this means that our loss function will be different as we are approximating $p(x|z)$. We will use something called Kullback-Leibler Divergence to represent this approximation.

## 3.3 Kullback-Leibler Divergence(KL Divergence)

KL Divergence is also known as relative entropy. It is used to measure how much a specific probability distribution differs from another probability distribution. Often the latter is a reference probability distribution. In the case of a VAE, we use a standard normal distribution as the reference distribution. It is represented by the following equation:

$$D_{KL}(Q(x)||P(x)) = -\sum_{x \in \chi} Q(x) log\ \frac{Q(x)}{P(x)} \tag{6}$$

This can be interpreted as the information lost when the function $P(x)$ is approximated by $Q(x)$. This term is added to the loss function that we saw previously to give us the new loss function for the VAE. The new loss function is will be the equality case of the following.

$$log\ p(x) \geq -D_{KL}(q(z|x)||p(z) + \mathbb{E}_{q(z|x)}[log\ p(x|z)] \tag{7}$$

### 3.4 Reparametrisation

Now that we have a loss function, we can differentiate it. The KL-divergence term can be integrated analytically and then partially differentiated normally. However differentiating $\mathbb{E}_{q(z|x)}[log\ p(x|z)]$ is a trickier task. The expectation can only be found by sampling. And differentiating the expectation is not directly possible. However, we can find a way around this.

We know that our latent variable $z$ follows the distribution $q(z|x)$ in the encoder. And $q(z|x) = \mathbb{N}(\mu = 0, \sigma^2 = 1)$. In this case, we have to differentiate some function of $z$ (which is actually in this case the expectation of $log\ p(x|z)$) which we'll represent with $h(z)$ for now.

$$\mathbb{E}_{q(z|x)}[h(z)] = \mathbb{E}_{\mathbb{N}(\mu,\sigma^2)}[h(z)]$$

One small change we can make in the way we consider the latent variable $z$ is to write it as $\mu + \sigma\epsilon$ where $\epsilon = \mathbb{N}(0, 1)$. Now what is actually happening is

$$\mathbb{E}_{\mathbb{N}(\mu,\sigma^2)}[h(z)] = \mathbb{E}_{\mathbb{N}(0,1}[h(\mu + \sigma\epsilon)] \sim \frac{1}{T}\sum_{t\in T} h(\mu + \sigma\epsilon_t)$$

This means that we can sample $T$ different values from $\epsilon$ and partially differentiate w.r.t. $\mu$ and $\sigma$ without any problem.

## 4 Variational Recurrent Neural Network

Now the main concern of the paper is to incorporate VAEs at each timestep of an RNN. What they are trying to achieve is make sure that information is passed on over the data sequence. Unlike regular VAEs, they're making sure that the prior is not independant of timestep which is a unique idea.

### 4.1 Mathematics behind the working of the VRNN

The way they propose to do it is by replacing the latent variable $z$ as shown in section 2 with a function which depends on hidden state $h_{t-1}$.

$$z_t \sim \mathbb{N}(\mu_{0,t}, \sigma_{0,t}^2)$$

where the $\mu$ and $\sigma^2$ represent the parameters of the simple gaussian of the prior of $z$ for timestep $t$.

$$[\mu_{0,t}, \sigma_{0,t}^2] = \varphi_\tau^{prior}(h_{t-1}) \tag{8}$$

Now because our input $x$ and output $z$ will be a little too complex to analyse, we can simplify them using feature extractors $\varphi^x$ and $\varphi^z$ using the same neural network. For the generation, the likelihood distribution will have the parameters $\mu_{x,t}$ and $\sigma_{x,t}^2$ which can be computed as follows.

$$[\mu_{x,t}, \sigma_{x,t}^2] = \varphi_\tau^{decoder}(\varphi^z(z_t), h_{t-1}) \tag{9}$$

Here, $\tau$ represent the parameters of our neural network which is used to compute them. Once we have our generation, we can pass on information from the hidden layer to the next timestep using an equation similar to Eq. (1). In the following, we see that because the likelihood we're computing is $p(x|z)$, our hidden state is dependant on both $x$ and $z$.

$$h_t = f_\theta(\varphi^x(x_t), \varphi^z(z_t), h_{t-1}) \tag{10}$$

Eq. (9) and Eq. (12) represent $p(z_t|(x_{<t}, z_{<t}))$ and $p(x_t|(z_{<t}, x_{<t}))$ respectively. Our final output of forward propogation will then be $p(x_{\leq T}, z_{\leq T})$.

$$p(x_{\leq T}, z_{\leq T}) = \sum_{t=1}^{T}(p(z_t|x_{<t}, z_{<t})p(x_t|z_{\leq t}, x_{<t})) \tag{11}$$

It is like Eq. (4), an application of the Bayes Theorem. Next, we move onto the inference model which is also in a way the backward of the generative model. The inference model or the posterior distribution will have parameters $\mu_{z,t}$ and $\sigma^2_{z,t}$

$$[\mu_{z,t}, \sigma^2_{z,t}] = \varphi_\tau^{encoder}(\varphi^x(x_t), h_{t-1}) \tag{12}$$

As the Eq. (11) is the parametrisation of the generative model, the following is the parametrisation of the inference model.

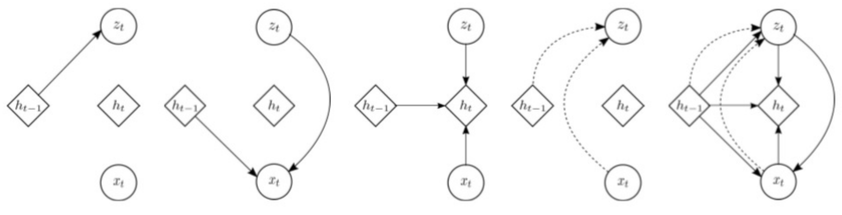
$$q(x_{\leq T}, z_{\leq T}) = \sum_{t=1}^{T} q(z_t|x_{<t}, z_{<t}) \tag{13}$$

Note that our generative model is $p(x|z)$ which is the decoder and our inference model is the approximation of the posterior is $q(z|x)$ which is the encoder. The decoder needs $z$ as conditional input and the encoder needs $x$ as conditional input which explains the subscripts of the different values of $\mu$ and $\sigma$.

Now what remains is calculating the loss function using Eq. (11) and Eq. (13).

$$\mathbb{E}_{x_{\leq T}, z_{\leq T}}\left[\sum_{t=1}^{T}(-\mathbb{D}_{KL}(q(z_t|x_{<t}, z_{<t}||p(z_t|x_{<t}, z_{<t})) + log\ p(x_t|z_{\leq t}, x_{<t})\right] \tag{14}$$

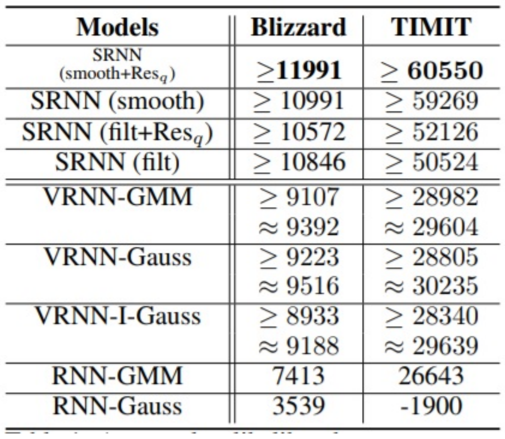
This is the lower bound of the log likelihood loss and it needs to be maximised for better results. Below is a summarisation of all the computations our neural network needs to perform.



(a) Prior (b) Generation (c) Recurrence (d) Inference (e) Overall

## 5 Conclusion

The paper introduces a concept that is very useful for modelling sequential data. It is unique in that they were able to execute their logic in a neat and efficient manner on large datasets. Previous attempts were far less complex and did not experiment beyond toy data. Their results clearly show that they are better than traditional RNNs. This shows that there is more scope for delving into more complexity in this area. This paper was not particularly something pathbreaking but it shows a new way to deal with things. One of the major problems with this is the same as with simple RNNs. Every output is affected only by the inputs before and not after it. It is uni-directional and like RNNs, they do not retain memory for a long while. If somehow the VAE principle could be efficiently modelled into more complex RNNs or LSTMs or GRUs, it might show significantly greater results. This has been achieved using SRNNs.[5]



| **Models** | **Blizzard** | **TIMIT** |
|---|---|---|
| SRNN (smooth+Res$_q$) | $\geq$**11991** | $\geq$ **60550** |
| SRNN (smooth) | $\geq$ 10991 | $\geq$ 59269 |
| SRNN (filt+Res$_q$) | $\geq$ 10572 | $\geq$ 52126 |
| SRNN (filt) | $\geq$ 10846 | $\geq$ 50524 |
| VRNN-GMM | $\geq$ 9107 | $\geq$ 28982 |
| | $\approx$ 9392 | $\approx$ 29604 |
| VRNN-Gauss | $\geq$ 9223 | $\geq$ 28805 |
| | $\approx$ 9516 | $\approx$ 30235 |
| VRNN-I-Gauss | $\geq$ 8933 | $\geq$ 28340 |
| | $\approx$ 9188 | $\approx$ 29639 |
| RNN-GMM | 7413 | 26643 |
| RNN-Gauss | 3539 | -1900 |

## 6 Bibliography

1. Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron Courville and Yoshua Bengio. *A Latent Random Variable Model for Sequential Data.* NIPS 2015

2. D. P. Kingma and M. Welling. *Auto-encoding variational bayes.* In Proceedings of the International Conference on Learning Representations (ICLR), 2014.

3. Stephen G. Odaibo. *Tutorial: Deriving the Standard Variational Autoencoder (VAE) Loss Function.* arxiv preprint arxiv:1907.08956, 2019

4. Otto Fabius & Joost R. van Amersfoort. *Variational Recurrent Auto Encoders.* Workshop contribution to ICLR 2015

5. Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. *Sequential Neural Models with Stochastic Layers.* NIPS 2016

6. Iulian V. Serban, Alexander G. Ororbia II, Joelle Pineau, Aaron Courville. *Multi-modal Variational Encoder-Decoders.* ICLR conference submission, 2017

7. Sequence Models by deeplearning.ai. https://www.coursera.org/lecture/nlp-sequence-models/

8. Deep Learning Wizard.
https://www.deeplearningwizard.com/deep_learning/practical_pytorch/

9. Bayesian Methods for Machine Learning.
https://www.coursera.org/lecture/bayesian-methods-in-machine-learning

10. Miscelleanous articles and videos from coursera.com, youtube.com, towardsdatascience.com, analyticsvidhya.com, stackexchange.com, github.com