Project Report
Variational Recurrent Neural Networks

Frederik Harder
frederik.harder@student.uva.nl

1 Introduction and related work

Over last few years, several deep learning applications have been developed with the goal of generating highly complex samples and advances in the state of the art have come to a point, where these results are drawing attention beyond academic circles. Work by Gatys et al. [3] which started out as texture generation has given rise to a wave of applications in neural style transfer [4]. Generative adversarial models (GAN) by Goodfellow et al [6] have become a highly active research area in image generation. Google has set a new state of the art in text to speech modelling with the WaveNet architecture by van den Oord et al. [14], working on raw audio data, and has launched Magenta[1], a project working on music generation and other topics involving creativity and artificial intelligence. Generative Modelling is becoming an increasingly relevant part of the machine learning landscape.

Variational Autoencoders (VAE) by Kingma and Welling [10] continue to be a popular approach to generative modeling, with new variants and improvements still in active development [11] [15]. What makes them attractive compared to other models like GANs [6] or Pixel-RNNs [13] is their formulation as a graphical model with explicit latent distribution. This latent distribution has proven valuable in disentangling and compressing representations and learning interpretable latent dimensions, while the model also allows for a clean introduction of priors and is easy to train.

This project takes a look at a particular extension to the VAE for sequential data. The Variational Recurrent Neural Network (VRNN) model by Chung et al. [1] couples the VAE with an LSTM network [8] which is then trained to inform the prior of the latent distribution at each time step. The authors have used this model to generate speech using raw audio data and handwriting as described below. On the practical side, the project encompasses an implementation of both the VAE and VRNN models, in order to experiment with sequential MNIST and attempt to reproduce results by Chung et al. on the task of handwriting generation. In addition, it takes methods for using Gaussian mixtures as latent distributions in VAEs and develops the theoretical analog for VRNNs. The report begins by laying out the definitions of both models in section (2), followed by a discussion of the data-sets in (3) and experiments in (4). Section (5) develops two ways of using latent Gaussian mixtures in VRNNs and discusses their viability and (6) closes with a discussion of the project as a whole.

2 Models

2.1 Variational Autoencoder

The Variational Autoencoder [10] is a generative latent variable model, which learns a data distribution $p(x)$ by maximizing its variational lower bound, shown in equation 1, under the assumption of a latent variable z. Approximate posterior $q(z|x)$ and prior $p(z)$ are generally chosen as an uncorellated and a unit Gaussian distribution respectively[2] The likelihood distribution $p(x|z)$ can be chosen more freely but is also typically a Gaussian or mixture of Gaussians. During training, the model shown in figure 1 consists of an encoder and a decoder network. Given a data sample $x^{(i)}$, the encoder network

[1] Magenta website: `https://magenta.tensorflow.org`

[2] For a discussion, on what latent distributions are feasible, see [10], section 2.4

outputs distribution parameters of $p(z|x)$. The decoder takes a sample $z^{(i)}$ from this distribution, and outputs parameters for $p(x|z)$, thus sampling the expected log likelihood present in the lower bound. Optimizing the lower bound minimizes the KL divergence between $q(z|x)$ and $p(z)$ and maximizes the expected log likelihood. Once the model has been trained and $q(z|x) \approx p(z)$, one can draw noise from the prior and the decoder then yields samples from the learned data distribution, as is shown in figure 2.

$$\log p(x) \geq \mathbb{E}_{q(z|x)}[\log p(x|z)] - KL(q(z|x)||p(z)) \tag{1}$$

An important feature of this model is the re-parametrization trick. As sampling from the latent distribution directly would not allow for the computation of gradients with respect to the encoder, sampling being non-differentiable, the parameters are instead combined with unit Gaussian noise ε as $z = \mu + \Sigma \cdot \varepsilon$ for the same result. This, and the fact that the KL divergence of two Gaussians has an analytic solution, make the Gaussian a particularly good choice for the latent distribution.

2.2 Variational Recurrent Neural Network

The VRNN model by Chung et al. [1] expands upon the VAE architecture in order to handle sequential data as opposed to single samples. This is achieved by incorporating an LSTM, which maintains a hidden state h_t based on z_t, x_t and h_{t-1} and informs prior, approximate posterior and likelihood in timestep $t + 1$, leading to the factorization shown in equation 2.

$$p(x_{\leq T}, z_{\leq T}) = \sum_{t=1}^{T} p(x_t|x_{<t}, z_{\leq t})p(z_t|z_{<t}, x_{<t}) \tag{2}$$

Further, the model gains two networks φ_x and φ_z for learning useful representations of samples from both data and latent samples respectively for added flexibility. Besides that, each timestep functions like a regular VAE. During generation, values for latent variable z_t at each timestep are sampled from the prior $p(z_t|x_{<t}, z_{<t})$. A sample x_t is taken from the output distribution and its representation $\varphi_x(x_t)$ is fed into the LSTM denoted as f_θ. In this way, the full sequence can be generated in one go. The model is shown in figures 3 and 4. Note that, as in the VAE, φ_{enc} and φ_{prior} utilize re-parametrization, but this is omitted to make the model more readable.

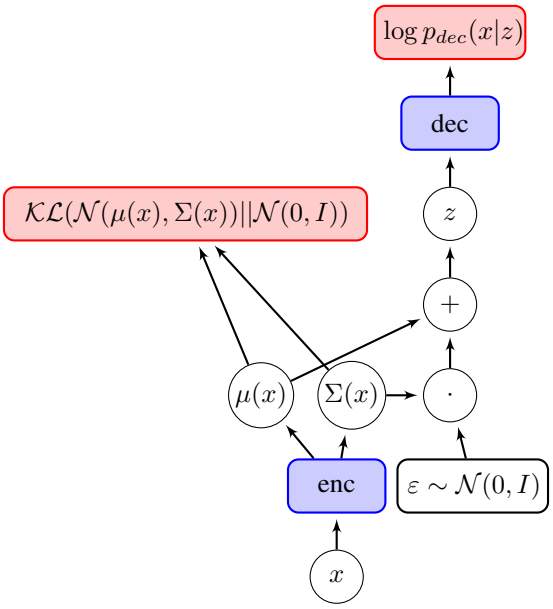

Figure 1: VAE model - training

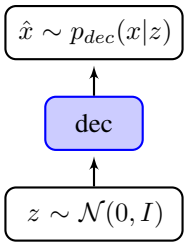

Figure 2: VAE model - generation

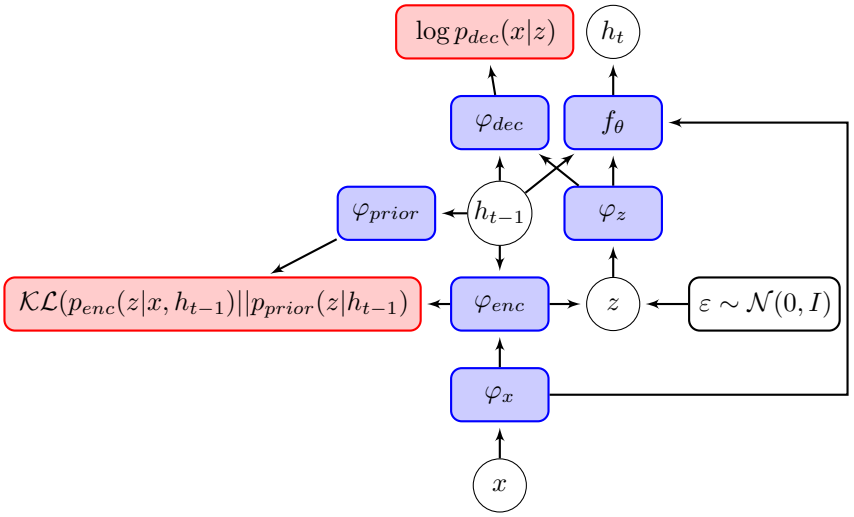

Figure 3: VRNN model - training

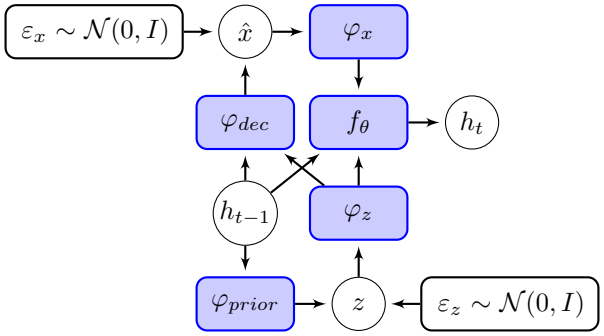

Figure 4: VRNN model - generation

3 Data-Sets

3.1 MNIST

This widely used data set of handwritten digits[3] contains 70000 greyscale images at a resolution of 28 by 28 pixels. Each image contains a digit between 0 and 9 which has been size normalized and centered and thus requires no further preprocessing. The set is usually split into train, validation and test sets of 50k, 10k and 10k images respectively. During initial phases of the project, it was used for debugging the variational auto-encoder. For the later tests using the VRNN, which are discussed

[3]MNIST data set available here: `http://yann.lecun.com/exdb/mnist`

below, each image was split into rows and given to the model top to bottom as a sequence with 28 time steps.

3.2 On-Line Handwriting Data Set

The IAM-OnDB[4] has gathered handwriting data on a 'smart whiteboard' from 221 individual writers. Sequences consist of (x, y) coordinates of the pen over time along with an additional bit b indicating, where the pen is lifted off the board, usually at the end of a character. The set consists of 13,049 written lines containing a total of 86,272 words. A VRNN has already been trained on this data set in [1], where the authors were able to generate outputs resembling handwriting, as they claim, of varying styles, which stay consistent across a sequence. individual characters are also recognizable, while sequences of characters tend to be nonsensical. This data set is chosen for the the task of reproduction primarily for its simplicity, which allows for comparatively short training times, as other applications like raw audio data are significantly more complex. The data set has previously been used in a paper by Graves [5] and the pre-processing follows the procedures described there. Absolute pen positions for each sequence are turned into relative steps, starting at $(0, 0)$. Steps in each dimension are reduced to the interval of $[-300, 300]$ and the data set is then normalized to zero mean and unit variance. All sequences are cut to the same maximum length, with shorter ones padded by a masking constant of 500, which is ignored by the model during training. The final results were produced by training on sequences of length 500.

4 Experiments

The main goal of experiments has been the reproduction of results on handwriting generation published by Chung et al. After failing to produce results of comparable quality, despite prolonged effort, and identifying potential problems inherent in the data set, the model was additionally tested on sequential MNIST data. The models in both experiments use ReLUs throughout the networks and were trained with the ADAM optimizer with mild gradient clipping to a size ± 1000. They were run in Tensorflow using an Nvidia Titan X GPU. The code is available on GitHub[5].

4.1 Handwriting generation

On the task of handwriting generation, the initial setup was adopted from [1], with each MLP set to a single ReLU layer of size 200 and an LSTM of size 1200. The layers were soon reduced to to sizes 50 and 600, in order to speed up training, with no discernible difference in training error. Models were trained with a learning rate of 3×10^{-5}.

An investigation of the code provided by Chung et al.[6] revealed, that the binary output of the pen lift-off is not modelled as part of the Gaussian but separately as a sigmoid with binary cross-entropy loss. Modelling the binary output separately turns out to be crucial both for performance and numerical stability, as variance in this dimension will rapidly go to 0 otherwise. Another detail, which poses a bit of a problem for this data set, is the choice of latent dimension. The authors do not mention their solution in the paper, but three or more dimensions allow for trivial identity mappings with no need to compress the data in any meaningful way, in which case the latent covariance goes to zero and performance is decided solely, by how well the LSTM can model the latent distribution. On the other hand, limiting the latent distribution to 1 or 2 dimensions restricts the of the data too much, as it clearly has 3 independent components.

Figure 5 shows results generated with a single latent dimension and represents the highest quality of generated samples I was able to attain. It can be seen, that a basic semblance with handwriting is learned, as the drawn lines are locally smooth, but the quality is very inconsistent over the course of each sequence. The model does not capture the structure of letters with sufficient consistency to be interpretable. The binary lift-off information is lost, potentially owed to a class imbalance, as only 4% of data points are "lift-off" points, and the style, as far as this term is applicable here, is not particularly varied. While explorations of 1, 2 and 10 latent dimensions did not yield significant

[4]Handwriting data available here: `http://www.fki.inf.unibe.ch/databases/iam-on-line-handwriting-database`

[5]Project code available here: `https://github.com/Fr-d-rik/vrnn`

[6]Chung et al's VRNN code available here: `https://github.com/jych/nips2015_vrnn/`

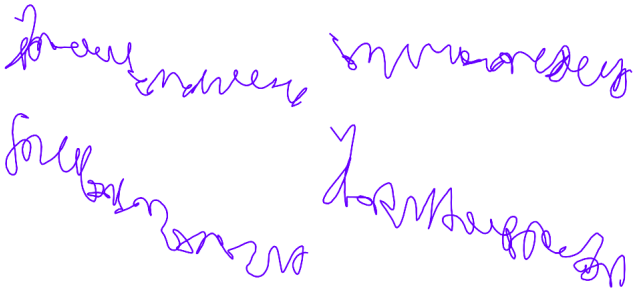

Figure 5: Sequences from single latent dimension, after 6500 iterations

differences, this may be owed to the overall poor quality of the results. As in [1], both a Gaussian and mixture of Gaussians were explored as output distributions. However, analysis of the generating process showed in the cases when mixtures were used, one Gaussian would grow to accumulate almost all of the probability mass, at which point the models were equivalent. After several weeks of experimenting with this data set without significant improvements, the task was put on halt in favour of experiments on MNIST.

One insight with regards to training which may be worth mentioning here after all, is that the training process seems to proceed in typical stages. Initially, when accuracy is low, both latent and output variance increase significantly, and can remain at high values for a while, before predictions become more accurate and variance decreases. When measuring only the lower bound, this can look like early convergence with little to no improvement. Tracking partial loss terms and distribution parameters has proven vital, as the lower bound tends to be uninformative.

4.2 Sequential MNIST

The comparatively short sequences of 28 MNIST rows make training with larger models more viable and as such, experiments were conducted with a 2 layer LSTM of 1500 units per layer and every other network set to 300 hidden and output units each, which totals around 4.2 million parameters. Considering the handwriting results, the output was chosen as a single Gaussian. The final model is trained for 60k iterations with a batch size of 100, using decreasing learning rates every 20k iterations, starting at 1×10^{-4}, then 3×10^{-5} and finally 3×10^{-6}, at which point the bound converges. The final score on the validation set settles at a mean of 21.3 per sample.

Figures 6, 7 and 8 show samples generated by the VRNN, the latter two being primed with the first 7 and 14 rows of validation set samples. While results are for the most part legible and improve when given a prime sequence, they lack the polish presented in other publications such as [10]. Potentially, some of this is owed to the sequential presentation of the data, which may decrease global consistency of the samples, as errors add up over timesteps, but ultimately this should not pose a problem for the model. On the chance that the model may over-regularize, as observed in [2] among others, a separate training run was conducted, discounting the KL divergence term by a factor of 0.1. The results displayed in figure 9 show no improvement, suggesting, that this is not the problem here.

For comparison, a simple LSTM model is trained, which consists of two LSTM layers with 2000 units each, surrounded by two MLPs at input and output, with a single 500 unit hidden layer each. The second MLP outputs parameters for a Gaussian distribution, which is trained to optimize the log likelihood of the following timestep. This architecture has a total of around 6 million parameters and after training to convergence after 40k iterations, achieves a mean log likelihood score of 18.4 on the validation set.

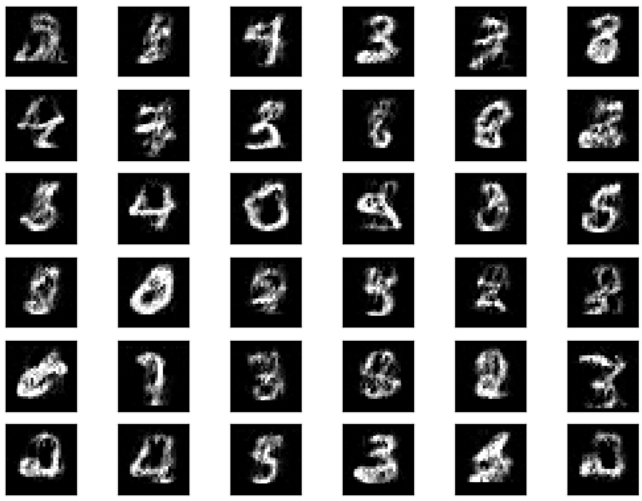

Figure 6: VRNN generated MNIST samples without prime

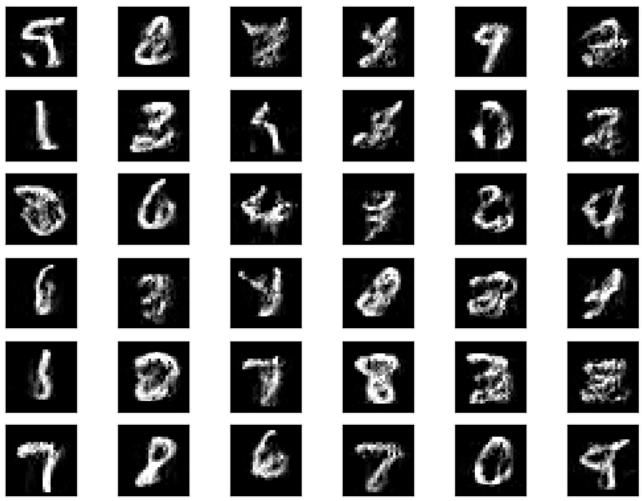

Figure 7: VRNN generated MNIST samples with 25% prime

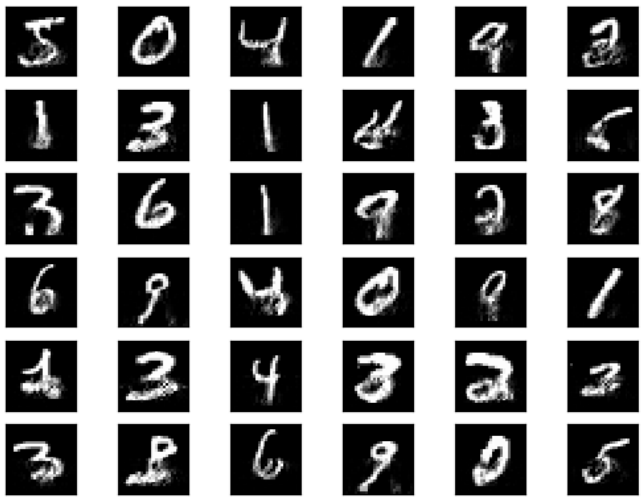

Figure 8: VRNN generated MNIST samples with 50% prime

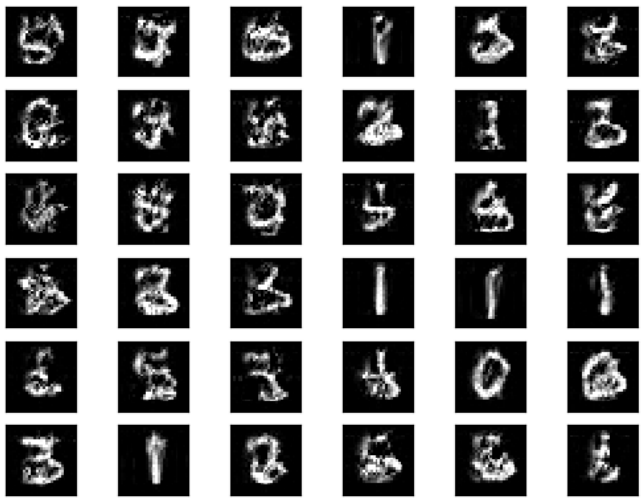

Figure 9: VRNN generated MNIST samples with KL divergence loss discounted by a factor of 0.1

5 Latent Gaussian Mixtures for Variational RNN

While an uncorrelated latent Gaussian distribution can have the desirable effect of generating interpretable features along its independent dimensions, more complex latent distributions promise superior model flexibility. Gaussian Mixtures (GM) are a natural candidate for such an effort, but present two challenges. The KL divergence between two GMs has no analytic solution and sampling from a GM has no straightforward re-parametrization trick. Below, I first present a naïve approach, which seeks to solve both issues with approximations and then take work from Dilokthanakul et al. [2], who have developed a VAE with latent GMs, and integrate this into the VRNN architecture.

5.1 Approximation via Gumbel-Softmax trick

The KL-divergence for GMs has no analytic solution, but [7] provides a survey of approximations and bounds. Given Gaussian mixtures $f = \sum_a \pi_a f_a$ and $g = \sum_a \omega_a g_a$, the matched bound approximation shown in equation 3 provides an upper bound to the KL divergence. Its major drawback is, that the orders of Gaussians f_a and g_a are arbitrary, but as both distributions are learned in the VRNN, matching orders can be learned as well.

$$D(f||g) \leq \sum_a \pi_a (\log \pi_a / \omega_a + D(f_a||g_a)) \tag{3}$$

A re-parametrisation trick for GMs has two components: Each Gaussian can utilize the same trick as before. Which Gaussian is sampled from, can be decided by sampling from a multinomial distribution parameterized by π. However, this in turn requires a reparametrization trick for multinomials. The Gumbel-Softmax trick explored in several recent publications ([9], [12], [15]) offers a continuous approximation of multinomials, which can be used for this purpose. Following [9], the multinomial is approximated with a softmax distribution using noise from the Gumbel g_i distribution and a temperature $\tau > 0$. With μ_i, σ_i and π_i as given as outputs of φ^{enc} and φ^{prior}, both latent GMs are then defined as follows:

$$g_i \sim \text{Gumbel}(0,1) = \exp(-g_i - \exp(-g_i)) \tag{4}$$

$$v_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=0}^{K} \exp((\log(\pi_j) + g_j)/\tau)} \tag{5}$$

$$\varepsilon_i \sim \mathcal{N}(\varepsilon_i|0, I) \tag{6}$$

$$w_i = \mu_i + \sigma_i \varepsilon_i \tag{7}$$

$$z = \sum_i^K v_i w_i \tag{8}$$

The approximation becomes exact as τ goes to 0, with the trade-off of gradients going to 0 almost everywhere. This can be addressed by annealing τ over the course of training. In [9], Jang et al. try a number of annealing schedules of the form $\tau = \max(0.5, \exp(-rt))$ for timestep t and parameter r. Whether the approximations of both KL-divergence and multinomial sampling are good enough to make the model as a whole feasible has to be determined empirically.

5.2 Variational approximation

In [2] Dilokthanakul et al. propose their model of a Gaussian mixture VAE (GMVAE), which defines the latent distribution using two additional random variables, v and w.[7] Here v is a multinomial variable deciding, which Gaussian of the mixture is sampled from and w is a Gaussian variable, which serves as input to a function such that $\varphi_{GM}^{(k)}(w)$ is the kth component of the mixture. The above-mentioned problems are avoided by choosing a different factorization for the variational approximation $q(v, w, z|x)$ and choosing $q(z|x)$ as a Gaussian, while the prior $p(z|v, w)$ is a GM. This model can readily be integrated into a VRNN, as specified in the following:

[7] In order to keep the notation consistent with the VRNN model, the names from [2] have been changed: x, y, z are referred t as z, x and v respectively

5.2.1 Generation

$$w_t \sim \mathcal{N}(\mu_{0,t}, diag(\sigma^2_{0,t})), \quad \text{where } [\mu_{0,t}, \sigma_{0,t}] = \varphi_{prior_w}(h_{t-1}) \tag{9}$$

$$v_t \sim \text{Mult}(\pi_{0,t}), \quad \text{where } \pi_{0,t} = \varphi_{prior_v}(h_{t-1}) \tag{10}$$

$$z_t|v_t, w_t \sim \prod_{k=1}^{K} \mathcal{N}(\mu^{(k)}_{w,t}, diag(\sigma^{2^{(k)}}_{w,t}))^{v_k} \quad \text{where } [\mu^{(k)}_{w,t}, \sigma^{(k)}_{w,t}] = \varphi^{(k)}_{GM}(w_t) \tag{11}$$

$$x_t|z_t \sim \mathcal{N}(\mu_{x,t}, diag(\sigma^2_{x,t})) \quad \text{where } [\mu_{x,t}, \sigma_{x,t}] = \varphi_{dec}(\varphi_z(z_t), h_{t-1}) \tag{12}$$

5.2.2 Inference

$$w_t|x_t \sim \mathcal{N}(\mu_{w,t}, diag(\sigma^2_{w,t})), \quad \text{where}[\mu_{w,t}, \sigma_{w,t}] = \varphi_{enc_w}(\varphi_x(x_t), h_{t-1}) \tag{13}$$

$$z_t|x_t \sim \mathcal{N}(\mu_{z,t}, diag(\sigma^2_{z,t})), \quad \text{where}[\mu_{z,t}, \sigma_{z,t}] = \varphi_{enc_z}(\varphi_x(x_t), h_{t-1}) \tag{14}$$

$$v_t|w_t, z_t \sim \text{Mult}(\tau), \quad \text{where } \tau_i = \frac{\pi_i \mathcal{N}(z_t|\mu_i(w_t;\beta), \sigma_i(w_t;\beta))}{\sum_{k=1}^{K} \pi_k \mathcal{N}(z_t|\mu_k(w_t;\beta), \sigma_k(w_t;\beta))} \tag{15}$$

The objective function is the timestep-wise variational lower bound, as in a standard VRNN, but now with additional variables v and w. With $h_t = f_\theta(x_t, z_t, h_{t-1})$ as the information bottleneck between time steps, conditioning on h_{t-1} can replace conditioning on $(x_{<t}, z_{<t}, w_{<t}, v_{<t})$.

$$\mathcal{L}_{ELBO} = \sum_t \mathbb{E}_q \left[\log \frac{p(x_t, z_t, w_t, v_t|x_{<t}, z_{<t}, w_{<t}, v_{<t})}{q(z_t, w_t, v_t|x_{\leq t}, z_{<t}, w_{<t}, v_{<t})} \right] \tag{16}$$

$$= \sum_t \mathbb{E}_q \left[\log \frac{p(x_t, z_t, w_t, v_t|h_{t-1})}{q(z_t, w_t, v_t|x_t, h_{t-1})} \right] \tag{17}$$

$$\begin{aligned} = \sum_t \Big[& \mathbb{E}_{q(z_t|x_t,h_{t-1})}[\log p_\theta(x_t|z_t, h_{t-1})] \\ & - \mathbb{E}_{q(w_t|x_t,h_{t-1})p(v_t|z_t,w_t,h_{t-1})}[KL(q(z_t|x_t, h_{t-1})||p(z_t|w_t, v_t, h_{t-1}))] \\ & - KL(q(w_t|x_t, h_{t-1})||p(w_t|h_{t-1})) \\ & - \mathbb{E}_{q(z_t|x_t,h_{t-1})q(w_t|x_t,h_{t-1})}[KL(p(v_t|z_t, w_t h_{t-1})||p(v_t|h_{t-1}))] \Big] \end{aligned} \tag{18}$$

Following [2], each timestep decomposes into 4 terms: reconstruction term, conditional prior term, w-prior term and v-prior term. Three out of the four must be sampled, while the w-prior can be solved analytically. Note that after rewriting the prior term, all KL-divergence terms in the approximation contain pairs of either Gaussian or multinomial distributions of matching dimensionality and thus have analytic solutions, given in the appendix.

$$\begin{aligned} \mathcal{L}_{ELBO} \approx \sum_t \Big[& \frac{1}{M} \sum_{j=1}^{M} [\log p_\theta(x_t|z_t^{(j)}, h_{t-1})] \\ & - \frac{1}{M} \sum_{j=1}^{M} \sum_{k=1}^{K} p(v_t = k|z_t^{(j)}, w_t^{(j)}, h_{t-1}) KL(q_{\phi_z}(z_t|x_t, h_{t-1})||p_\beta(z_t|w_t, v_t = k, h_{t-1}))] \\ & - KL(q_{\phi_w}(w_t|x_t, h_{t-1})||p(w_t|h_{t-1})) \\ & - \frac{1}{M} \sum_{j=1}^{M} [KL(p_\beta(v_t|z_t^{(j)}, w_t^{(j)} h_{t-1})||p(v_t|h_{t-1}))] \Big] \end{aligned} \tag{19}$$

The GMVAE model adds several networks to the VRNN architecture. The prior network for z is replaced with priors for v and w, an encoder function for w is added, and a separate network computes

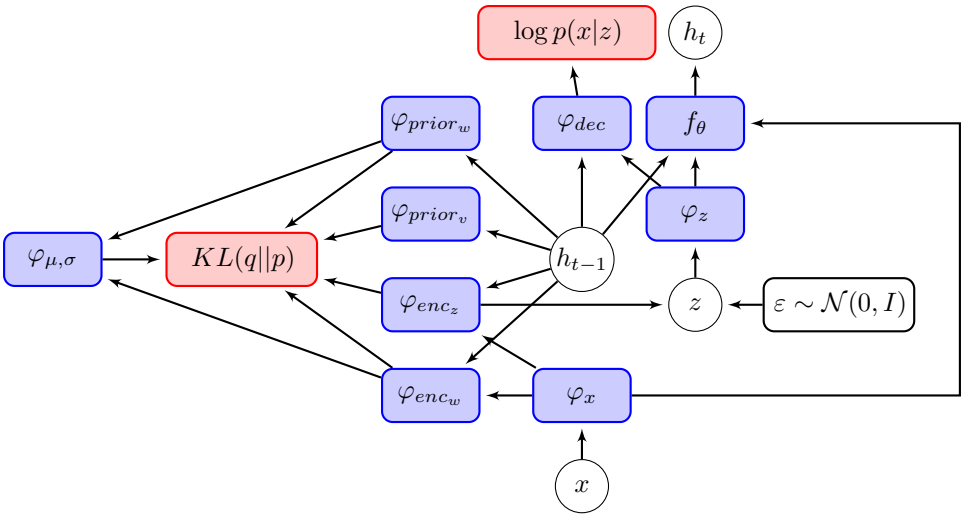

Figure 10: GMVRNN model - training

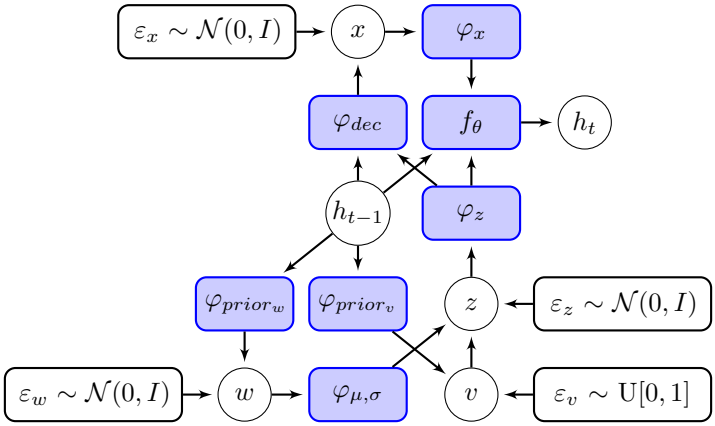

Figure 11: GMVRNN model - generation

the parameters of the Gaussian mixture $\mu(w)$ and $\sigma(w)$. The figures below show the model during training (10) and generation (11).

6 Discussion

Working on this project had two main motivations. The first was to attain a thorough understanding of the VRNN architecture and its relation the other extensions of the variational autoencoder, and secondly, I hoped that this complex generative model could be used to produce high quality results, as demonstrated by the original authors, where previous attempts using simple LSTMs had shown their limitations. The latent Gaussian mixture augmentations discussed in section 5 are a way of putting the acquired theoretical understanding to use, by spelling out, how the VRNN model may be modified further using existing methods. What has been less successful is the practical application of the model. The experiments presented here have shown the implementation to be working and, in the case of MNIST, outperforming a naïve LSTM model, but fall behind what has already been presented by Chung et al. Containing a total of 6 neural nets, this model comes with a considerable number of parameters which need to be chosen. Training times of 6 and more hours on MNIST have slowed down iteration considerably and looking back, exploring more efficient means of parameter tuning should have been a priority. Where experienced researchers may have developed intuitions they can fall back on, my attempts at optimizing the architecture have had marginal impact at best. And so any future projects using the VRNN architecture will require a plan for dealing with the model's complexity, either by scaling up the difficulty of the task along with the model, or by finding more informative ways of monitoring performance during training.

References

[1] Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., & Bengio, Y. (2015). A recurrent latent variable model for sequential data. In Advances in neural information processing systems (pp. 2980-2988). Chicago

[2] Dilokthanakul, N., Mediano, P. A., Garnelo, M., Lee, M. C., Salimbeni, H., Arulkumaran, K., & Shanahan, M. (2016). Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders. arXiv preprint arXiv:1611.02648.

[3] Gatys, L., Ecker, A. S., & Bethge, M. (2015). Texture synthesis using convolutional neural networks. In Advances in Neural Information Processing Systems (pp. 262-270).

[4] Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576.

[5] Graves, A. (2013). Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850.

[6] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).

[7] Hershey, J. R., & Olsen, P. A. (2007, April). Approximating the Kullback Leibler divergence between Gaussian mixture models. In Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on (Vol. 4, pp. IV-317). IEEE.

[8] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[9] Jang, E., Gu, S., & Poole, B. (2016). Categorical Reparameterization with Gumbel-Softmax. arXiv preprint arXiv:1611.01144.

[10] Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.

[11] Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., & Welling, M. (2016). Improved variational inference with inverse autoregressive flow. In Advances in Neural Information Processing Systems (pp. 4743-4751).

[12] Maddison, C. J., Mnih, A., & Teh, Y. W. (2016). The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. arXiv preprint arXiv:1611.00712.

[13] Oord, A. V. D., Kalchbrenner, N., & Kavukcuoglu, K. (2016). Pixel recurrent neural networks. arXiv preprint arXiv:1601.06759.

[14] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... & Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. CoRR abs/1609.03499.

[15] Rolfe, J. T. (2016). Discrete Variational Autoencoders. arXiv preprint arXiv:1609.02200.

A Closed form KL divergence

A.1 between two Gaussians

$$KL(\mathcal{N}(x|\mu_1,\Sigma_1)||\mathcal{N}(x|\mu_2,\Sigma_2)) = \frac{1}{2}\left(\log\frac{|\Sigma_2|}{|\Sigma_1|} - d + \text{tr}(\Sigma_2^{-1}\Sigma_1) + (\mu_2-\mu_1)^T\Sigma_2^{-1}(\mu_2-\mu_1)\right) \tag{20}$$

A.2 between two multinomials

$$KL(\text{Mult}(x|\pi)||\text{Mult}(x|\tau)) = \sum_k^K \text{Mult}(x=k|\pi)\log\frac{\text{Mult}(x=k|\tau)}{\text{Mult}(x=k|\pi)} = \sum_k^K \pi_k \log\frac{\tau_k}{\pi_k} \tag{21}$$