“Discovering Latent Dependency Structure in VAEs”

CS726 Final Project Report

Team BLYN

Bhavesh (150100007), Lalit (150070036), Yash (160050002), Nihal (150040015)

Goal

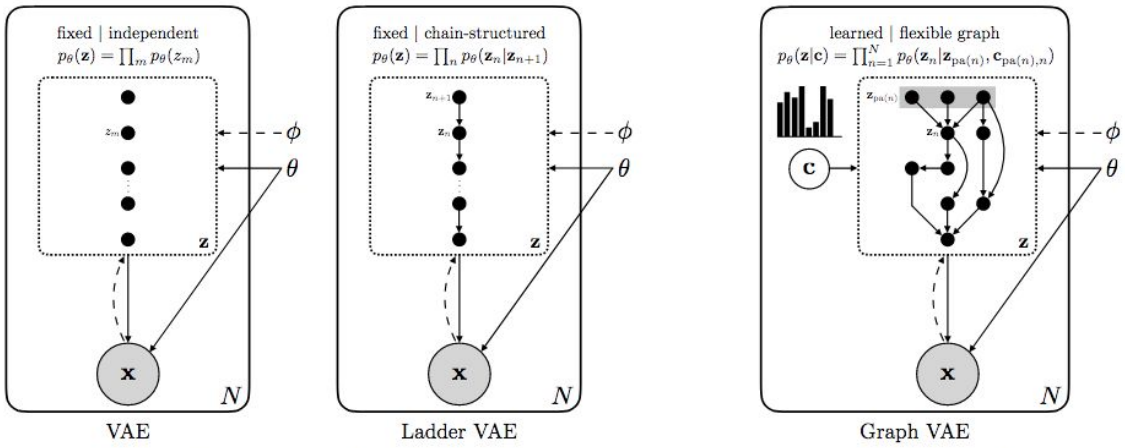
Variational Auto-Encoders typically make use of latent variables with the assumption that they are independent of each other. Ignoring the dependencies between latent variables limits the flexibility of these models, negatively impacting the model’s ability to fit the data. Leveraging the dependency structure would help the model to better learn the distribution of the latent variables and perform better on the underlying generation task. It would also help us interpret their effect and inter-relationship in a much better manner.

“Variational Autoencoders with Jointly Optimized Latent Dependency Structure” is a recent paper that appeared in ICLR 2019, which assumes a Bayesian Network as the dependency structure between the bottleneck latent variables of the VAE. They propose a model architecture that has the ability to jointly learn the dependency structure in the form of a Bayesian Network along with learning the parameters for the encoder and the decoder. In doing so, the proposed architecture combines the strength of deep generative models and probabilistic graphical models.

Our goals with the project are as follows:

1. **Implementation:** Since the code for the aforementioned paper has not been made available, we wrote our own implementation in Python.
2. **Dependency Structure using LSTMs:** The paper proposes a top down architecture to model the dependency structure for each latent variable to its parent latent variables. We experiment by replacing these top down inference modules by a LSTM network to model the dependency between latent variables.
3. **Extension to Sequential Data:** We propose an RNN extension of the idea to deal with sequential data using a VAE. We also propose the corresponding ELBO for optimization.

Related Literature



(a) Traditional Latent Variable Models (b) Proposed Model

1. **VAE**

   Variational autoencoders (VAEs) are a deep learning technique for learning latent representations. The traditional VAE, as we know it, came around in 2014 with the Auto Encoding Variational Bayes (AEVB) algorithm. Variational autoencoders are powerful generative models for unsupervised learning.

   AEVB is based on ideas from variational inference. In variational inference, the objective is to maximize the evidence lower bound (ELBO)

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

   over the space of all q_ϕ. The ELBO satisfies the equation

$$\log p_\theta(x) = KL(q_\phi(z|x)||p(z|x)) + \mathcal{L}(p_\theta, q_\phi).$$

   Since x is fixed, *q(z|x)* is defined to be conditioned on x. This means that for every *x*, a different *q(z)* is chosen, which will produce a better posterior approximation than always choosing the same *q(z)*. Optimizing our objective requires a good estimate of the gradient. The main technical contribution of the VAE paper is a low-variance gradient estimator based on the reparameterization trick. Gaussian variables provide the simplest example of the reparameterization trick. Instead of writing

$$z \sim q_{\mu,\sigma}(z) = \mathcal{N}(\mu, \sigma)$$

   we may write

$$z = g_{\mu,\sigma}(\epsilon) = \mu + \epsilon \cdot \sigma,$$

where ϵ ~ N(0,1)
It is easy to check that the two ways of expressing the random variable *z* lead to the same distribution. The biggest advantage of this approach is that we many now write the gradient of an expectation with respect to *q(z)* (for any f) as -

$$\nabla_\phi \mathbb{E}_{z\sim q(z|x)}\left[f(x,z)\right] = \nabla_\phi \mathbb{E}_{\epsilon\sim p(\epsilon)}\left[f(x,g(\epsilon,x))\right] = \mathbb{E}_{\epsilon\sim p(\epsilon)}\left[\nabla_\phi f(x,g(\epsilon,x))\right]$$

The gradient is now inside the expectation and we may use MCMC sampling to get an estimate of the right-hand term. **Even though this approach works well, it suffers from the limitation that it treats latent variables as independent.**

Paper: https://arxiv.org/abs/1312.6114
Implementation: https://github.com/pytorch/examples/tree/master/vae
Key points: Treats latent variables independently

2. Ladder VAE

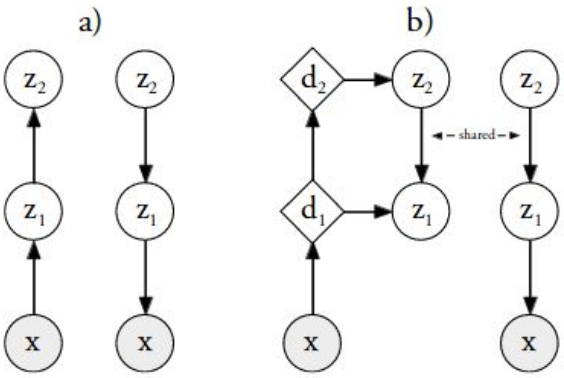


Figure 1: Inference (or encoder/recognition) and generative (or decoder) models for a) VAE and b) LVAE. Circles are stochastic variables and diamonds are deterministic variables.

This paper proposes a new inference model, the Ladder Variational Autoencoder, that recursively corrects the generative distribution by a data dependent approximate likelihood in a process resembling the *Ladder Network*. First, a deterministic upward pass computes the Gaussian likelihood like contributions:

$$\mathbf{d}_0 = \mathbf{x}.$$

$$\mathbf{d}_n = \mathrm{MLP}(\mathbf{d}_{n-1})$$

$$\hat{\mu}_{q,i} = \texttt{Linear}(\mathbf{d}_i), i = 1 \dots L$$

$$\hat{\sigma}^2_{q,i} = \texttt{Softplus}(\texttt{Linear}(\mathbf{d}_i)), i = 1 \dots L$$

This is followed by a stochastic downward pass recursively computing both the approximate posterior and generative distributions.

$$q_\phi(\mathbf{z}|\mathbf{x}) = q_\phi(\mathbf{z}_L|\mathbf{x}) \prod_{i=1}^{L-1} q_\phi(\mathbf{z}_i|\mathbf{z}_{i+1}, \mathbf{x})$$

$$\sigma_{q,i} = \frac{1}{\hat{\sigma}_{q,i}^{-2} + \sigma_{p,i}^{-2}}$$

$$\mu_{q,i} = \frac{\hat{\mu}_{q,i}\hat{\sigma}_{q,i}^{-2} + \mu_{p,i}\sigma_{p,i}^{-2}}{\hat{\sigma}_{q,i}^{-2} + \sigma_{p,i}^{-2}}$$

$$q_\phi(\mathbf{z}_i|\cdot) = \mathcal{N}\left(\mathbf{z}_i|\mu_{q,i}, \sigma^2_{q,i}\right),$$

Paper: https://papers.nips.cc/paper/6275-ladder-variational-autoencoders.pdf
Implementation: https://github.com/geosada/LVAE
Key points: Assumes chain dependency structure for latent variables.

3. **FC VAE**
   Paper: https://arxiv.org/pdf/1606.04934.pdf
   Implementation: https://github.com/openai/iaf
   Key points: posterior is iteratively refined, models a fully connected graph over latent variables

Approach

We propose two **main modifications** (apart from implementing the paper itself) to the method already described in the paper (for details regarding other experiments, see the *Experimental results* section) -

(i) Replacing top-down inference module with an LSTM network

We try to model the dependency structure in the top-down structure. The set of parent candidates for Z_i is the entire set Z_N, Z_{N-1} … Z_{i+1} and we try to model this in our top-down network. After discussing with the TA, we thought this was similar to using an LSTM instead of the top down network, in which the hidden state from Z_{i+1} would provide context from all Z_N, Z_{N-1} … Z_{i+1} to Z_i . Our intuition turned out to be right, and replacing the top-down version with LSTM actually gave us slightly better results. The only drawback was the lack in interpretability, since we don't get an explicit graph structure in this case.

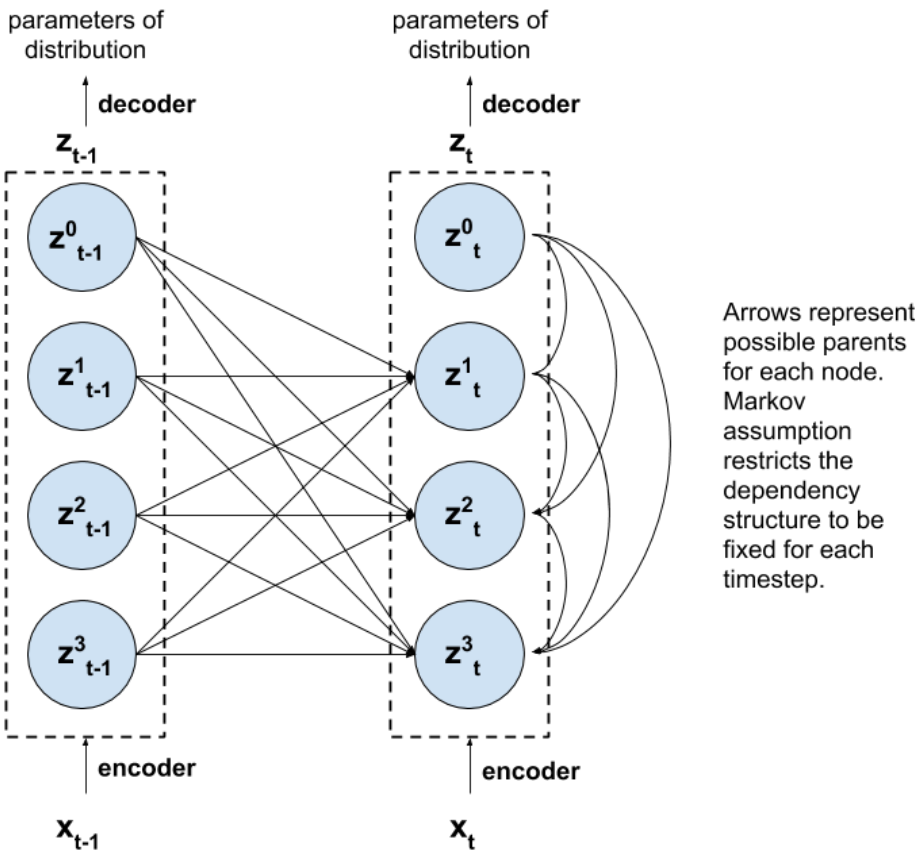
(ii) Extending GraphVAE for sequential data

We propose a simple extension of the method described in the paper for handling sequential data. We allow the parent set of Z_i^t to be the union of $\{Z_N^t, Z_{N-1}^t \dots Z_{i+1}^t\}$ as well as $\{Z_N^{t-1}, Z_{N-1}^{t-1} \dots Z_1^{t-1}\}$ i.e. latent variables of both timesteps *t* and *t-1*. We further make the Markov assumption, which forces the dependency structure between Z_i's to remain fixed across all timesteps. This further enables the approximate posterior to be decomposed into a product of conditional distributions of the latent variables over all timesteps, thereby allowing the formulation of a new ELBO term -

$$q_\phi(\mathbf{z}|\mathbf{x},\mathbf{c}) = \prod_{n=1}^{N} q_\phi(\mathbf{z}_n|\mathbf{x},\mathbf{z}_{\mathrm{pa}(n)},\mathbf{c}_{\mathrm{pa}(n)})$$

$$\mathbb{E}_{p(\mathbf{c})}\left[\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z},\mathbf{c})\right] - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{c}))\right] = \mathbb{E}_{p(\mathbf{c})}\left[\mathcal{L}_\mathbf{c}\right]$$

Here $Z_{Pa(n)}$ includes latent variables of both timesteps *t* and *t-1*. The following diagram gives an overview of our proposition:



Implementation details

We've used PyTorch to code up the architecture. There are roughly about a total of 3000 lines of code in the repository. There are 3 main branches, *master* (for GraphVAE), *LSTM* (for GraphLSTMVAE) and *vrnn* (for RecurrentGraphVAE). You can find the code here. We coded up the entire architecture ourselves, using the PyTorch template as our starting point.

Platform

We ran our experiments on Nvidia 1080 Tis.
The timing studies are as follows. For GraphVAEs one epoch takes about 2s, and we ran it for about 1200 epochs.
For the LSTM variant, one epoch takes around 6s, and we ran it for about 1000 epochs
For our RecurrentGraphVAE, one epoch took about 1500s per epoch.

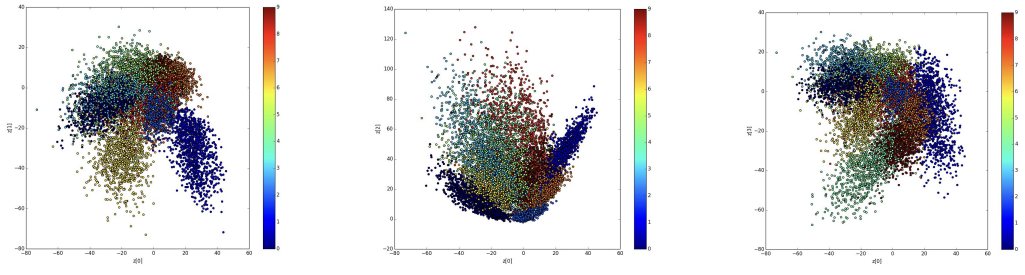
Experimental results

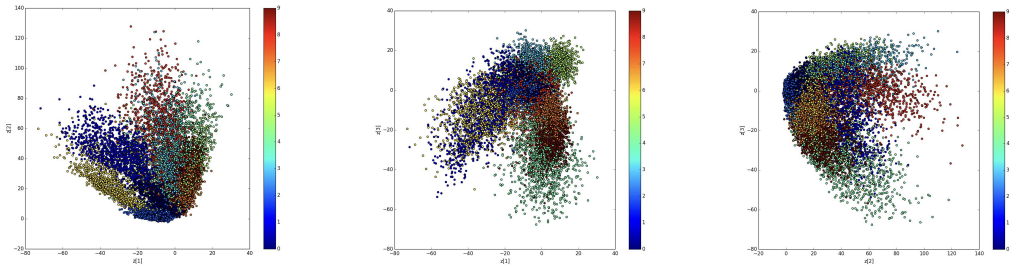
Datasets: Preliminary checks were performed on the standard MNIST dataset. Original GraphVAE model was evaluated on the binarized MNIST dataset, where each pixel value was first normalized to [0,1] range followed by discretization to a binary value by thresholding at 0.5. Each pixel was then modelled by a bernoulli random variable whose parameter μ was predicted by the VAE. The recurrent variant of GraphVAE model was evaluated on the preprocessed IAM Online Handwriting dataset. It consists of sequences of the 3-tuple *(x, y, e)* where *(x, y)* denote coordinates of the pen along the curve and *e* denotes whether it touches the surface or not. We modelled *(x, y)* using a bivariate gaussian with diagonal covariance matrix, and *e* using a bernoulli distribution.

We perform experiments for the following three tasks. Results obtained in each task are also provided.

(i) Ensuring that there indeed is some correlation amongst latent variables
One of the preliminary investigations we did was to deduce the correlation between the latent variables of a standard VAE. For each training sample we plotted the parameters of each latent variable against those of another. If latent variables were independent of one another, we should get a uniformly scattered plot. However, the plots (as well as the correlation matrix) we got clearly indicated that there was some correlation amongst latent variables. This served as a green flag for going ahead and actually finding the dependency structure amongst latent variables for this dataset.

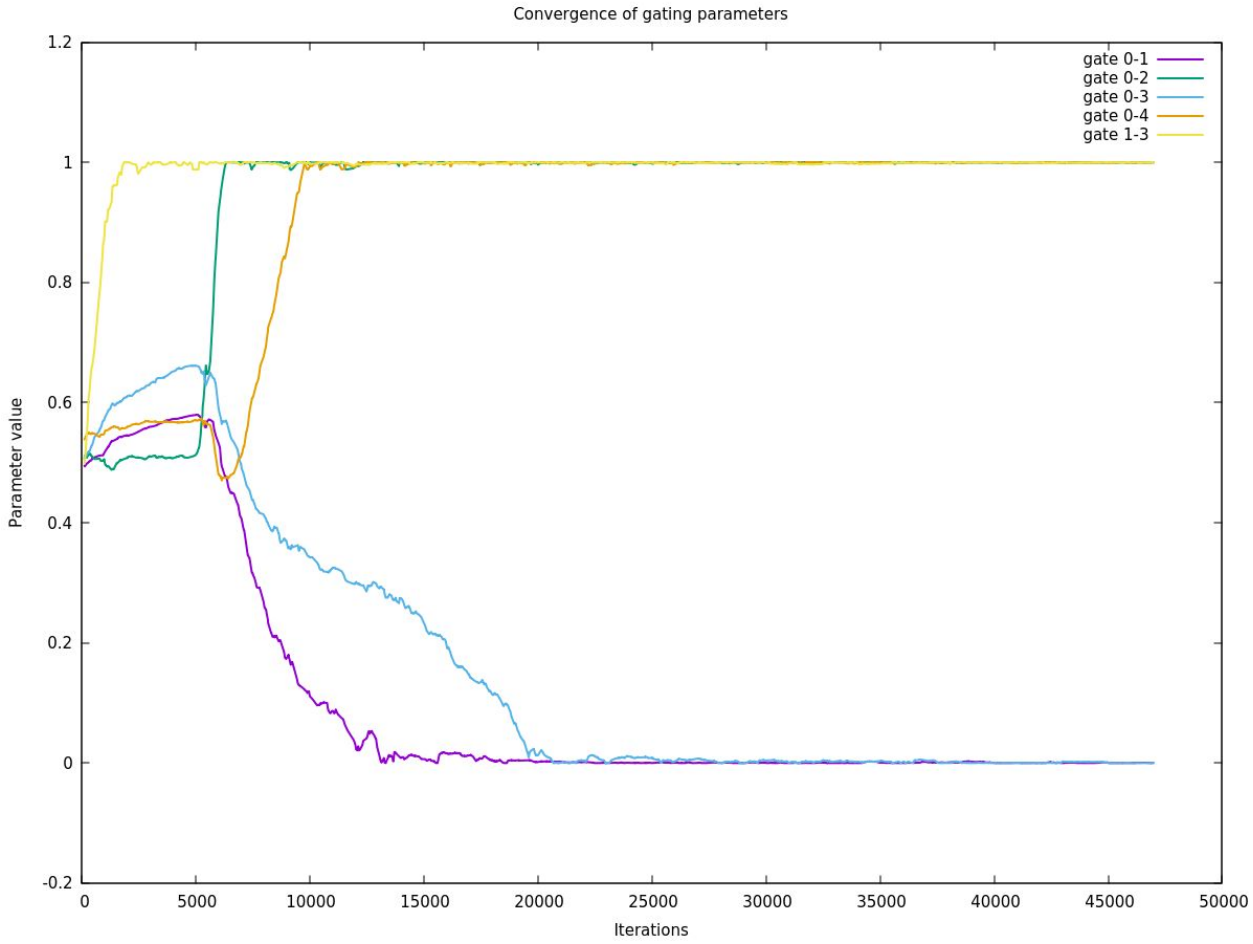
The image on the left is the correlation matrix between the predicted parameters of the distributions of latent variables. Had there been no correlation, this matrix would be the identity matrix. Non-zero values at non-diagonal positions indicate some correlation between latent variables (note that only the leading and trailing 4x4 submatrices are meaningful). Following pictures indicate the presence of some dependency structure in the distributions of latent variables.





(ii) Latent dependency structure for binarized MNIST dataset

This task involved implementing the GraphVAE model described in the paper, and using it to learn the structure of a bayesian network over the latent variables. Due to computational and time constraints, we had to make several simplifying assumptions; the authors used 5 hidden variables with 16 dimensions per variable, while we used single-dimensional 5 hidden variables. After 1000 epochs of training, only 5 out of the 10 gates had converged to 0/1 - the remaining gating parameters had either saturated to an intermediate value, or were changing very slowly. The plot below shows the variation of $\mu_{i,j}$ parameters for gates that converged (gate i-j is for the edge directed from z_j to z_i).

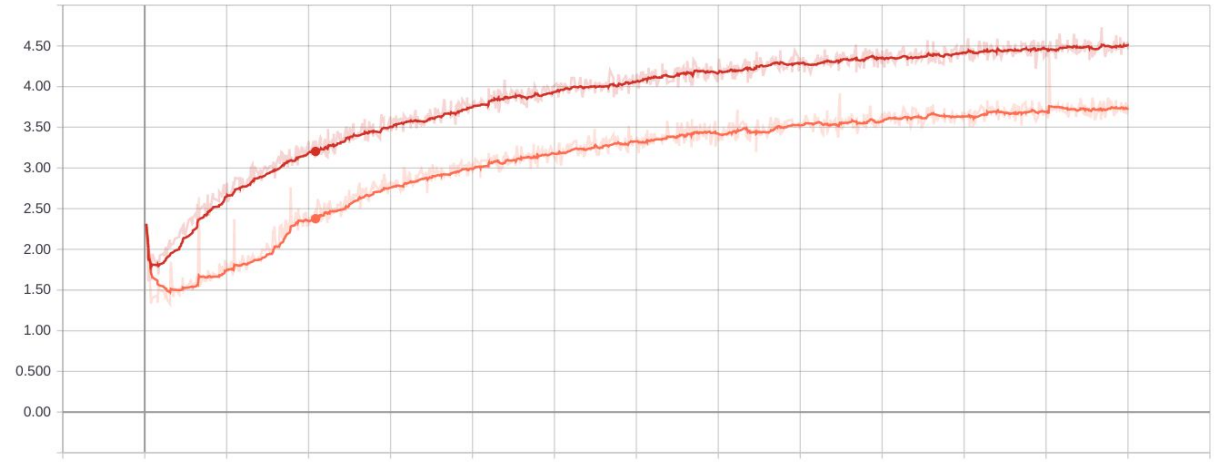


The orange plots below show the variation in $\mu_{i,j}$ values for all gates till 1000 epochs. We observed that after this, other gate values were changing quite slowly. In order to boost the process, we freezed the values of already converged gates to 0/1 accordingly and resumed training the model. The blue curve in the plot below depicts this change.

(iii) Replacing the top-down inference module with an LSTM network

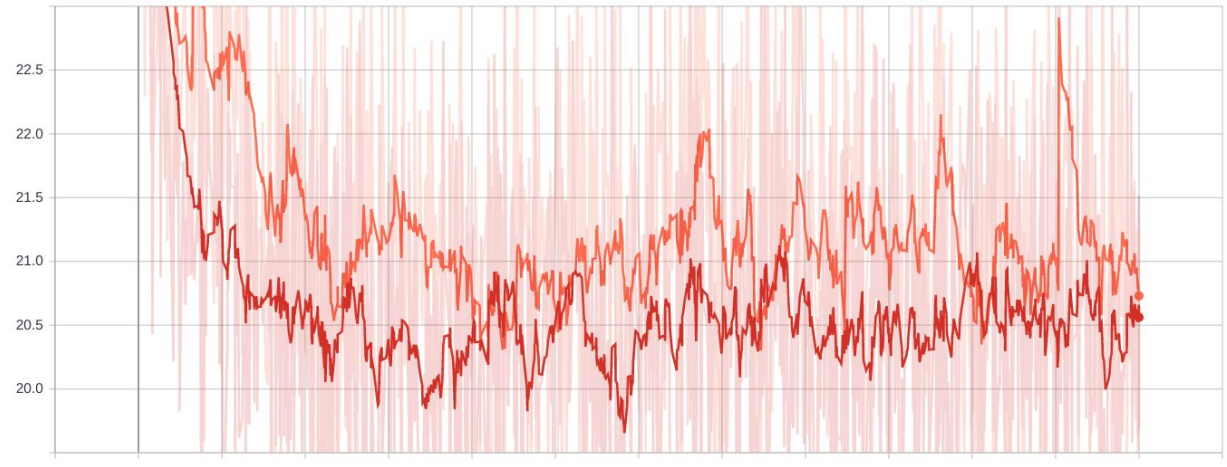
The top-down sampling process could be replaced by an LSTM network without changing the semantics. The original GraphVAE used separate top-down inference modules for each z_i, while the LSTM variant (we call it GraphLSTMVAE in our experiments) shares those parameters for all z_i. Thus, if one isn't concerned about explicitly finding the dependency structure (since this information is lost within the recurrent connections of the LSTM), GraphLSTMVAE provides a less expensive method to accomplish the same goal. Our experiments support this hypothesis, and indeed show that GraphLSTMVAE performs better than GraphVAE when trained for the same number of epochs (red - GraphLSTMVAE, orange - GraphVAE)

valid/kl



valid/nll

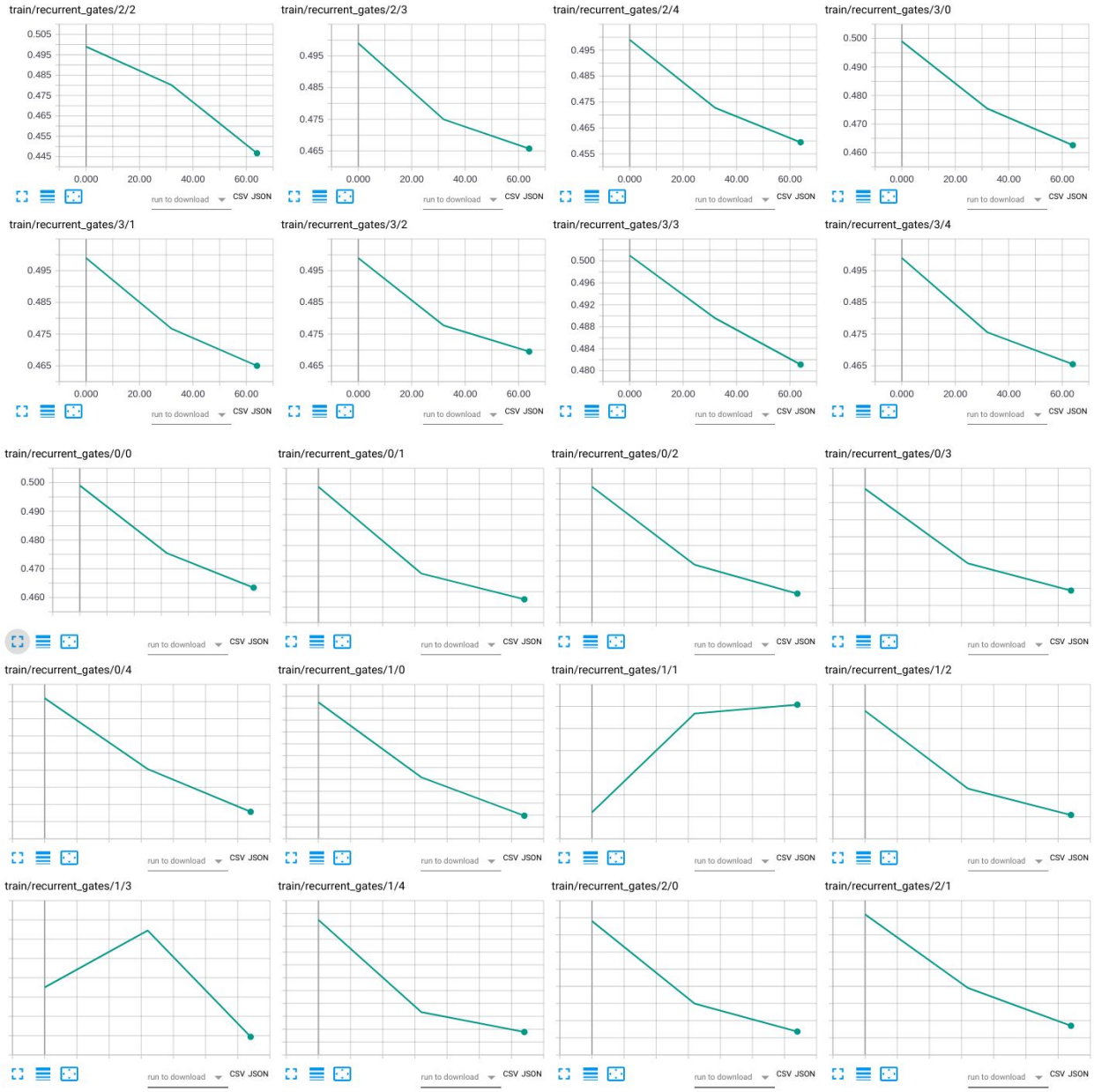
valid/loss



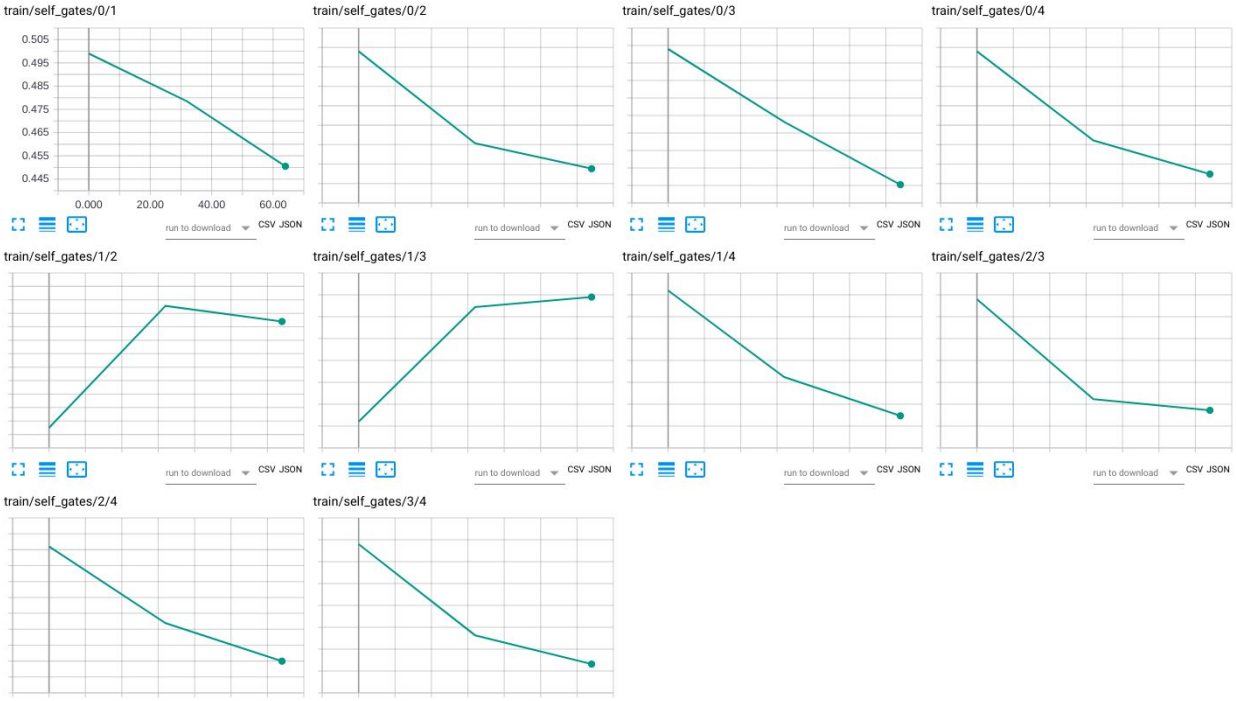
(iv) Extending GraphVAE for sequential data

We implemented RecurrentGraphVAE as described earlier, but due to computational constraints were unable to train the model on IAM online handwriting dataset quite well. The drastic increase in time per epoch was attributed to two nested for loops, one over timesteps and the other over a topological ordering of latent variables for each timestep. We include partially obtained results below for completeness.

Gating parameters for recurrent connections

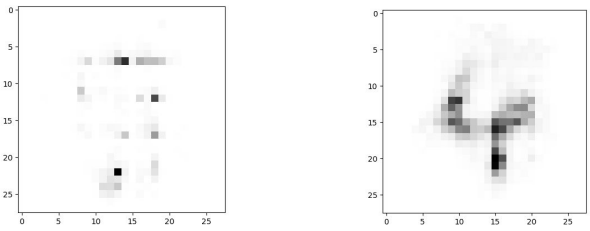


Gating parameters for self connections

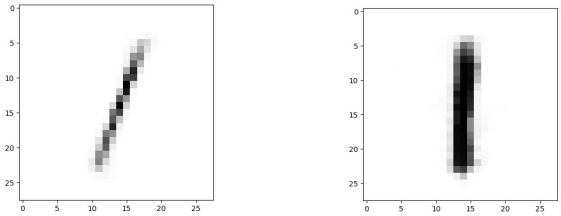


Generation

We tried using the decoder to generate some images by sampling from a unit multivariate Gaussian and passing it through the decoder. Here are some of the sample images that we got (5 and 4) -



Here are some of the sample images that were generated during training/reconstruction (1 and 1) -



Effort

Time Distribution:

Literature Survey:	20%
Coding:	40%
Experiments:	30%
Documentation:	10%

Most Challenging Part:

Resolving NaN mysteries
PyTorch variable freezing

Work Distribution:

Paper implementation (GraphVAE) and RecurrentGraphVAE - Yash
GraphLSTMVAE - Bhavesh and Nihal
Preliminary investigations - Lalit
Documentation - Yash, Bhavesh and Nihal