Interpretable dimensionality reduction for high-dimensional and unstructured data

Kyle Nickerson

Project Report
Completed as part of a Mitacs Accelerate Internship at Verafin

Project Supervisors:
Terrence Tricco (Verafin)
Ting Hu (Queens University)
Yuanzhu Chen (Memorial University)

1. Introduction

Both in industry and academic research, modern data analysis often involves working with complex data to make predictions or gain insights. While there can be many factors which make data complex, in this project we examine techniques for better understanding two types of complex data; *high dimensional* data and *unstructured sequence* data. High dimensional data refers to data where each observation is composed of a large number of features. A typical example of high dimensional data are images, which are represented as arrays of pixels. In our work on high dimensional data, we use the MNIST dataset (LeCun, Cortes, & Burges, 2010), which contains 28x28 pixel images of hand-written digits (see section 2.1 of the appendix for examples). Unstructured data generally refers to data that does not have a fixed dimension, such as text-based datasets, where observations are sentences or paragraphs which may vary in length.

In this project, we implement a version of the recently proposed *interpretable lens variable* model (ILVM) (Adel, Ghahramani, & Weller, 2018), which is an interpretability framework based on probabilistic modeling. Further, we propose a modified version of the ILVM, the *fixed interpretable lens variable* (FILVM), and demonstrate this model can learn to disentangle unique factors of variation in the learned representation better than the original ILVM.

To test the viability of our methods on unstructured data, we use a toy dataset containing variable length sequences of linearly increasing numbers (for example 7, 10, 13, 16, 19, 22). Due to time constraints, we were not able to validate our methodology on any text-based datasets, however this may be explored in future work. We also highlight issues with the ILVM, and FILVM, which may limit its general usefulness.

2. Methods

Consider the general scenario where we have a dataset X composed of N independent observations $\{x_1, x_2, \dots, x_N\}$ of a random variable x, sampled from some complexed distribution $p(x)$. Here x may be high dimensional or unstructured.

When working with complex data, we often assume the complex observations we have are the result of some unobserved process operating on unobserved factors. For example, in the MNIST dataset, the unobserved factors may be the intended digit and the thickness of the written implement; whereas with a text dataset, they may be the thought the author is trying to communicate and the target audience. A *latent variable model* is a probabilistic model which assumes that the observed data (x) are generated by a stochastic process operating on unobserved factors (z). A central task is probabilistic machine learning is the inference of unobserved latent factors by combining prior assumptions (expressed as probabilistic models) with observed data. Generally speaking, a probabilistic model is a description of a set of random variables and their interactions. A common and convenient way to specify a probabilistic model is as a description of a generative process which produces observed data. This is the approach adopted by all the following methods.

2.1. Variational Autoencoders (VAEs)

Variational autoencoders are a powerful framework for jointly learning a generative model $p(x, z; \theta)$ relating observed samples (x) to unobserved variables (z) as well as a recognition model used to infer an approximate posterior distribution $\tilde{p}(z|x; \phi)$ over unobserved variables.

[Note: Variables occurring after a semi colon represent parameters to the distribution. The pipe (|) sign indicates a conditional probability distribution. For example, $\tilde{p}(z|x; \phi)$ indicates a probability distribution over the random variable z given x, with parameters ϕ]

The generative model is specified in two parts, the prior distribution over latent variables $p_\theta(z)$ and the conditional distribution $p(x|z; \theta)$, which specifies the process mapping latent variables to observations. A common choice is to use a multivariate normal distribution, with **0** mean vector and **I** covariance for $p(z; \theta)$, and to represent $p(x|z; \theta)$ using either a multivariate normal (for real valued x) or Bernoulli (for binary x) distribution, whose parameters are computed by a neural network with parameters θ (Kingma & Welling, 2014). The full generative model is then

$$p(x, z; \theta) = p(z; \theta)p(x|z; \theta)$$

Often, we are interested in inferring the unobserved factors z underlying our observed data. This can be useful for a variety of tasks including clustering, segmentation and prediction. Bayes rule tells us how we can use our generative model to get a distribution over latent variables as $p(z|x; \theta) = \frac{p(x,z;\theta)}{p(x)} = \frac{p(x,z;\theta)}{\int_z p(x,z;\theta)}$, however in practice we cannot compute this because of the denominator is intractable. Because of this, we must rely on an approximate posterior $\tilde{p}(z|x; \phi)$ to infer these factors. As with the generative model, there is some flexibility in how we specify this distribution. In our VAE models we follow (Kingma & Welling, 2014) and use a multivariate normal distribution with a diagonal covariance structure to approximate $\tilde{p}(z|x; \phi)$, where the parameters of the normal distribution are computed by the recognition model with parameters ϕ for each sample x_i. This covariance structure corresponds to the assumption that each dimension of z is independent. There are two main benefits to this covariance structure. The first is our recognition network only needs to learn to approximate d parameters for the diagonal covariance, where as d^2 parameters are needed to specify the covariance matrix. The second is that we would like to learn representations where each factor is independent, as these are both more interpretable, and easier to work with in downstream tasks.

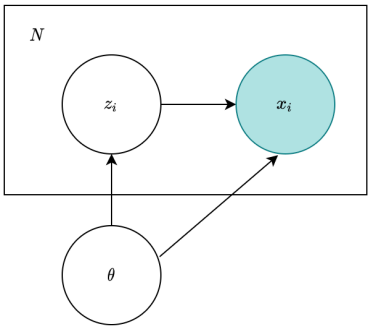

Figure 1: Graphical model representing the generative model used in a VAE. This model encodes the fact that we N occurrences of observed variable x_i and unobserved variable z_i, as well as a global unobserved parameter θ, which effects the generation of each z_i and x_i. This graphical representation encodes the assumption that each z_i depends only on θ, and each x_i depends only on θ and z_i

When working with VAEs, a typical choice for the prior distribution over the latent variable $p(z;\theta)$ is a multivariate normal distribution with mean **0** and covariance **I** (where **0** is a vector of zeros and **I** is the identity matrix, and the dimensions are the same as z). However, this choice is often the result of properties of the normal distribution which make it convenient to work with, rather than a strong belief that the latent variables truly follow this distribution. The motivation behind normalizing flows is to allow us to work with complex distributions which better describe our data, while retaining the coinvent properties of normal distributions. There are actually many models in machine learning and statistics which similarly make use of multivariate normal distributions for their convenience, and thus many ways in which normalizing flows can be used to improve data modeling.

2.2.Normalizing Flows

Normalizing flows are a relatively new technique for defining expressive probability distributions, which have been developed over the past decade (for a comprehensive review of the current state of the art see (Kobyzev, Prince, & Brubaker, 2020) and (Papamakarios, Nalisnick, Rezende, Mohamed, & Lakshminarayanan, 2019)). The basic idea behind normalizing flows is simple — we can define a complex probability distribution over a random vector x, by defining an invertible differentiable transformation $T(\cdot)$, and a simple base distribution $p(u)$, such that $x = T(u)$, where u is a random vector sampled from the distribution $p(u)$. We can then define the density of x using a change of variables as

$$p(x) = p(u)\,|\det \mathbf{J}_T(u)\,|^{-1}$$

where $\det \mathbf{J}_T(u)$ is the determinate of the Jacobian of the transformation T evaluated at the point u.

This simple idea can be extended by composing a series of transformations as $T(\cdot) = T_K \circ T_{k-1} \circ \ldots \circ T_1(\cdot)$, and parameterizing each transformation T_i with a set of parameters η_i, which can be inferred from data. In this project, we consider a particular type of normalizing flow, called planar flows (Rezende & Mohamed, 2015), in which each transformation has the form $T_i(z;\eta_i) = T_i(z;w_i,u_i,b_i) = z + u_i \tanh(w_i^T z + b_i)$, where u_i, w_i are vectors with the same dimension as z and b_i is a scaler. The main benefits of planar flows are they are easy to implement and allow us to compute $\det \mathbf{J}_T(u)$ in linear time (linear in dimension of z). In practice we use a modified version of these transformations in order to ensure invertibility, suggested in the appendix of (Rezende & Mohamed, 2015). In under this modification we compute flows of the form $f(z;\ w,u,b) = z + \hat{u}(u,w)\tanh(w^T z\ + b)$, where $\hat{u}(u,w) = u + [m(w^T u) - w^T u\ ]\frac{w}{||\ w||^2}$ and $m(x) = -1 + \log(1+e^x)$.

There are two main use cases of normalizing flows, both related to problem of modeling complex probability distribution $p^*(x)$ over a multidimensional continuous random variable x. The first scenario is when we have samples from $p^*(x)$ but are unable to evaluate $p^*(x)$ to compute the probability density of samples. For an example of this, consider the MNIST dataset, where $p^*(x)$ represents the naturally occurring distribution over 28x28 pixel images of handwritten digits. In this scenario we have samples from $p^*(x)$, but no way to evaluate the probability of images under this distribution, which could tell us if a sample is typical or an outlier. In this scenario we can create a flow-based model $\hat{p}(x;\eta)$ to approximate $p^*(x)$. To fit the flow-based model to the data, we use a stochastic optimization method (such as stochastic gradient descent) to find the value of η which minimizes the forward KL divergence (Papamakarios, Nalisnick, Rezende, Mohamed, & Lakshminarayanan, 2019).

$$Loss(\eta) = D_{KL}\big(p^*(x) \parallel \hat{p}(x;\eta)\big)$$

We can estimate this loss using a set of samples $\{x_1, x_2, \ldots, x_n\}$ from $p^*(x)$, as

$$Loss(\eta) \approx -\frac{1}{N}\sum_{n=1}^{N} \log p_U(T^{-1}(x_n;\eta)) + \log|\det \mathbf{J}_{T^{-1}}(x_n;\eta)| + constant$$

In practice, we estimate this loss using mini batches of samples, and iteratively update the parameters η using a gradient based method until convergence. See page 6 of Papamakarios, Nalisnick, Rezende, et al (2019) for a derivation of this loss estimator.

The other case where normalizing flows are useful is when we have a function $f(x) = Cp^*(x)$, which we can evaluate, but we cannot compute C. This scenario comes up in variational inference, when we have a probabilistic model $p(x,z)$ which describes relationships between observed and unobserved variables, and want to model the posterior distribution over hidden variables $p(z|x) = \frac{p(x,z)}{p(x)} = \frac{p(x,z)}{\int_z p(x,z)}$. Unless we restrict our self to only using very simple models $p(x,z)$, this denominator will not have a closed form.

When fitting the model in this case, the only thing that changes from the first scenario is we now minimize the reverse KL divergence.

$$Loss(\eta) = D_{KL}\big(\hat{p}(x;\eta) \parallel p^*(x)\big)$$

Which can be estimated as

$$Loss(\eta) = \mathbb{E}_{p_U(u)}\big[\log p_U(u) - \log|\det \mathbf{J}_T(u;\eta)| - \log f\big(T(u;\eta)\big)\big] + constant$$

To compute this, we sample from the base distribution $u \sim p(u)$, and apply the transformation to u with parameters η to obtain $T(u;\eta)$. As with the forward KL, we minimize this quantity by iteratively updating η using a gradient based optimizer. See page 6 of Papamakarios, Nalisnick, Rezende, et al (2019) for a derivation of this loss estimator.

Once we have fit a flow-based model, we can use it to generate new samples approximately following $p^*(x)$ by sampling the base distribution and applying the transformation T. We can also estimate the density of new samples, by applying the inverse transformation T^{-1} to map the samples to the base distribution, which we can then compute the density of.

2.3. Interpretable Lens Variable Model (ILVM)

The ILVM extends the framework of VAEs to a model with multiple types of hidden and observed variables and is designed to learn interpretable representations. To achieve this, the ILVM incorporates two types of observations; the samples we are interested in modeling (x) and additional side information (s) about some samples. The side information refers to a small number of feature(s) which a human can meaningfully relate to the samples (as an example, if the samples are images of hand drawn digits, the meaningful information could be the thickness and label of the digit). The motivation for including this side information is that it allows the ILVM to learn a secondary latent space z^* in which certain dimensions correspond to the side information.

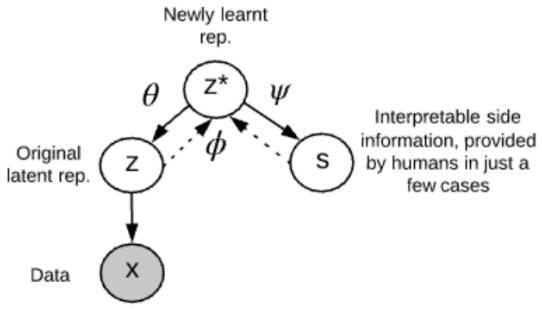

Figure 2: Diagram of the ILVM from (Adel, Ghahramani, & Weller, 2018). In this diagram, solid arrows indicate the generative model, and dashed arrows indicate the recognition model. The generative model shows the assumption the z and s only depend on z^* and parameters (θ for z, ψ for s), and x depends only on z (and the VAE parameters, which are not indicated here). The recognition model shows that the inference of z^* depends on both z and s.

The ILVM framework assumes that a VAE has already learned a generative model which relates unobserved factors z to observations x. Further, it assumes there are other unobserved factors z^*, which underly both the human interpretable factors s, as well as the latent representation z. In order to encourage interpretability, the model from z^* to s is restricted to be a linear, whereas the model from z^* to z may be arbitrarily complex. We implement this model as a deep neural network, essentially the same as the decoder network in a VAE. We assume this is how it was implemented in (Adel, Ghahramani, & Weller, 2018), however they do not explicitly state this. The remaining piece is the recognition model, which approximates the posterior distribution $q(z^*|z, s; \phi)$. Again, (Adel, Ghahramani, & Weller, 2018) does not give details about how they implement this, so we implement this also with a deep neural network, similar to the recognition model in a VAE. In our implementation we also have a secondary recognition network, which is designed to approximate $q(s|z)$ for the samples where s is not observed.

When fitting this model to data, the motivation behind the objective function is to find the model parameters which assign the highest probability to the observed data using a stochastic gradient based optimizer. This is known as maximum likelihood estimation and is a common objective for fitting probabilistic models. Because the VAE which maps x to z has been fit prior to the ILVM, we treat the encoding z as an observed random variable, as we encode x as z prior to fitting the ILVM. In practice we do not directly maximize the likelihood, instead we maximize a lower bound on the logarithm of the probability of the observed data, known as the evidence lower bound (ELBO). Working with the logarithms of probability values improves numerical stability, by converting the multiplication of probability values to addition of log probabilities. The parameters which maximize the original probability and log probability will be the same. In the ILVM, the log probability of a single data point is $\log p(z, s; \theta)$ if side information is observed and $\log p(z; \theta)$ otherwise. The total log probability of the dataset is the sum of these quantities over all data points in the dataset. Next we will see how to take this ideal but intractable objective and derive a tractable lower bound.

In the case where we have observed side information, we have:
$\log p(z, s; \theta, \psi) = \log \int_{z^*} p(z^*)p(z|z^*; \theta)p(s|z^*; \psi)\, dz^*$ **(1)**

We obtain the R.H.S. of this expression by using the general fact that $p(a) = \int_b p(a, b)db$ (for any random variable a and continuous random variable b), and the ILVM assumption that z and s are conditionally independent (see figure 2).

We can then introduce the approximation of the posterior by multiplying by one as

$\log p(z, s; \theta, \psi) = \log \int_{z^*} p(z^*)p(z|z^*; \theta)p(s|z^*; \psi) \frac{q(z^*|z, s; \phi)}{q(z^*|z, s; \phi)} dz^*$ **(2)**
This expression can be written as an expectation as

$\log p(z, s; \theta, \psi) = \log \mathbb{E}_{q(z^*|z, s; \phi)}[p(z^*)p(z|z^*; \theta)p(s|z^*; \psi)q(z^*|z, s; \phi)^{-1}]$ **(3)**

Using Jensen's inequality, this can be lower bounded as

$$\log p(z, s; \theta, \psi) \geq \mathbb{E}_{q(z^*|z,s;\phi)}[\log p(z^*)p(z|z^*;\theta)p(s|z^*;\psi)q(z^*|z,s;\phi)^{-1}] \quad \textbf{(4)}$$

The R.H.S of this expression is the ELBO. Using properties of logs, we have

$$ELBO = \mathbb{E}_{q(z^*|z,s;\phi)}[\log p(z^*) + \log p(z|z^*;\theta) + \log p(s|z^*;\psi) - \log q(z^*|z,s;\phi)] \quad \textbf{(5)}$$

Because $q(z^*|z,s;\phi)$ is represented with normalizing flows, with $q(z^*|z,s;\phi) = q(z_K|z,s)$, this becomes

$$\mathbb{E}_{q_0(z_0)}[\log p(z_K) + \log p(z|z_K;\theta) + \log p(s|z_K;\psi) - \log q_0(z_0) + \sum_{k=0}^{K-1} \log \det\left|J_{T_k}(z_k)\right|] \quad \textbf{(6)}$$

To understand how to compute this, I will first describe a forward pass through the model. The first step is the recognition network with parameters ϕ takes both z and s as inputs, and outputs a distribution over z^*, by outputting the parameters to a complex probability distribution represented with normalizing flows. Specifically, the network outputs mean and variance parameters for the base distribution $q_0(z_0)$, which is a gaussian, and flow parameters w_k, u_k, b_k, $k = 1..K$, for each of the K flow layers. Once this step is completed, we can sample from the approximate posterior $q(z^*|z,s;\phi)$ by sampling $z_0 \sim q_0$ and then deterministically mapping z_0 to $z_K = z^*$ with the flow parameters. We then use the sampled z_k as inputs to the two generative models and get distributions $p(z|z_K;\theta)$ and $p(s|z_K;\psi)$ as outputs. In the case where the predicted variable is continuous, the distribution is gaussian with a fixed variance, and the model outputs the mean. When the predicted variable is categorical, the model outputs a categorical distribution. Once we have finished this forward pass, we can compute each of the terms in **(6)** as follows:

1. $\mathbb{E}_{q_0(z_0)}[\log p(z_K)]$: The logarithm of the probability of z_k, under the prior distribution $p(z_k)$. We assumed this prior to be a unit gaussian, following (Tomczak & Welling, 2016), as it was not clear what was used in (Adel, Ghahramani, & Weller, 2018).

2. $\mathbb{E}_{q_0(z_0)}[\log p(z|z_K;\theta)]$: Because z is continuous, $p(z|z_K;\theta)$ is a gaussian with a mean predicted by the generative model, and constant variance. In this case, we can use the negative mean squared error (MSE) between the true and predicted z as $\log p(z|z_K;\theta)$, as they are equivalent to within a constant factor.

3. $\mathbb{E}_{q_0(z_0)}[\log p(s|z_K;\psi)]$: If s is continuous, such as the thickness score, then this is computed with the negative MSE as above. If s is discrete, such as the digit label, then this is computed as the negative cross entropy between the predicted categorical distribution and the true label.

4. $\mathbb{E}_{q_0(z_0)}[\log q_0(z_0)]$: This is the log probability of the sample z_0, under the base distribution $q_0(z_0)$, which is a gaussian distribution with parameters output by the recognition model.

5. $\mathbb{E}_{q_0(z_0)}[\sum_{k=0}^{K-1} \log \det \left| J_{T_k}(z_k) \right|]$: This is the sum of the log determinants over each step in the flow.

To understand what this objective does, it can be helpful to rewrite **(5)** as

$$-\mathbb{E}_{q(z^*|z,s;\phi)}[\log q(z^*|z,s;\phi) - \log p(z^*)] + \mathbb{E}_{q(z^*|z,s;\phi)}[\log p(z|z^*;\theta) + \log p(s|z^*;\psi)] \quad \textbf{(7)}$$

In (7), the first expectation is the negative KL divergence between the approximate posterior $q(z^*|z,s;\phi)$ and the prior $p(z^*)$, and the second term measures the quality of the generative models reconstructions of z and s. In our experiments, we multiply the KL portion of the objective by a constant β, which allows us to put more or less emphasis on the KL term. With this modification to **(6)**, the final objective becomes

$$\mathbb{E}_{q_0(z_0)}[\beta \log p(z_K) + \log p(z|z_K;\theta) + \log p(s|z_K;\psi) - \beta \log q_0(z_0) + \beta \sum_{k=0}^{K-1} \log \det \left| J_{T_k}(z_k) \right|] \quad \textbf{(8)}$$

In the case where there is no observed side information s, the above procedure only changes slightly, by adding an adding an additional first step to the forward pass. In this additional step, a secondary recognition network is used to give an approximate distribution over the unobserved s, and this approximate distribution is used in term 3. of **(8)** instead of the true s.

2.4. Fixed Interpretable Lens Variable Model (FILVM)

In the ILVM, interpretability of z^* results from the model assumption is that z^* can be used to predict s with a linear model. As will be covered in the discussion section, we found that while the standard ILVM did produce a z^* which was more interpretable than the original latent space z, the dimensions of z^* did not directly align with the side information attributes. This led us to propose a modified model, the fixed interpretable lens variable model (FILVM), which is better able to align z^* to with the side information.

Whereas the ILVM assumes the side information can be generated from z^*, in the FILVM, we instead assume that the j^{th} side information attribute can be generated from the j^{th} dimension of z^* with a linear model. The paper by (Adel, Ghahramani, & Weller, 2018) does not go into detail about the implementation of the linear model used, aside from stating that "for a k-dimensional z^*, $p_\psi(s|z^*)$ depends on $\sum_{j=1}^{k} \psi_j z_j^* + \psi_0$". Using this type of linear model however does not appear to achieve this goal, neither in theory nor in practice. The dimensionality of the ψ terms in this expression are not explicitly mentioned in the paper, but they must be vectors with the same dimensionality as s. To see why this type of model will not result in each attribute aligning with a single dimension, consider the case where s is 1-dimensional. If $p_\psi(s|z^*)$ depends on $\sum_{j=1}^{k} \psi_j z_j^* + \psi_0$, this mean the models' prediction of s is based on a linear combination of all dimensions of z^*, plus a bias term (in the case of a d-dimensional s, each dimension s_i is also predicted based on a linear combination of the dimensions of z^*). Empirically we found that

using this type of objective lead to latent spaces where attributes of s were significant, but not well aligned with the axis's (see figure 7).

3. Datasets and Side Information

In this we detail the three datasets used in our experiments, and the attributes which were used as side information in each dataset.

3.1. MNIST

The MNIST dataset contains 70,000 28x28 pixel images of digits, divided into a training set of 60,000 images and a test set of 10,000 images (LeCun, Cortes, & Burges, 2010).

For this dataset, the two side information attributes used are the digits label, and the stroke thickness. The label information is included in the original data. To compute the stroke thickness, we followed the pipeline suggested in (Coelho de Castro, Tan, Kainz, Konukoglu, & Glocker, 2018) (see my notebook digit_thickness_dataset.ipynb for an example). See figure B1 in appendix B for examples of MNIST digits of various thicknesses.

3.2. MNIST Rotated (MNIST-R)

This was a dataset we created by applying rotations to the original MNIST dataset. Each image in the original dataset was used to generate 8 images in the rotated dataset, by selecting 8 random rotations from the range $[-90°, 90°]$ and applying them to the original image. See figure B2 in appendix B for examples of MNIST-R digits of various rotations.

3.3. Toy sequence data

We also performed tests using a toy dataset of sequences of numbers. In this dataset, a sample is a variable length sequence of linearly increasing numbers. These sequences can be completely determined by any three of the following four numbers: the starting number, the step size, the sequence length, and the final number. To create each sequence in this dataset, we randomly select a starting number, step size and sequence length from a range of possible values (see table 1 for the ranges used for each attribute), and compute the final number based on these. See table B1 in appendix B for examples of sequences in this dataset.

	Min	Max
Start	1	1000
Step	1	50
Length	3	30

Table 1: Range of values for each of the three attributes used to generate sequences. These values imply the final numbers in the sequence will range from 3 to 2500.

4. Experiments

In this section I will describe the experiments we performed to validate the models we have implemented. The first two sections, on VAEs and normalizing flows, contain basic experiments which verify these models are working correctly. In the subsequent sections we discuss the interpretability experiments using the lens models.

4.1. Variational Autoencoders (VAEs)

To test the fidelity of our VAE architectures, we performed qualitative experiments to examine the ability of the model to compress and regenerate held-out samples from the datasets of interest. We also evaluate these experiments quantitatively, by measuring the mean squared error between the original samples and reconstructions. We perform these tests on VAEs trained on 3 datasets; MNIST (LeCun, Cortes, & Burges, 2010), a custom version of MNIST where the digits have been rotated (MNIST-R), and a toy dataset of sequences of linearly increasing numbers of various lengths. These 3 datasets are also used in evaluating the lens models. For more information on these datasets see section 2 of the appendix.

4.2. Normalizing flows

While we primarily use normalizing flows as a tool for variational inference in this project, we also performed basic density estimation experiments following to test our implementation of normalizing flows. In this experiment we are given a 2-dimensional unnormalized probability distribution where computing the normalizing constant would be intractable, and we aim to approximate the true distribution using normalizing flows. To accomplish this, we tested 2-dimensional normalizing flow models with l layers for $l = [2, 8, 16, 32]$, on a density estimation task. In this task we wish to model a probability distribution $p(z) \propto e^{-U(z)}$, where

$$U(z) = \frac{1}{2}\left(\frac{\|z\| - 2}{0.4}\right)^2 - \ln\left(e^{\frac{1}{2}\left[\frac{z_1 - 2}{0.6}\right]^2} + e^{\frac{1}{2}\left[\frac{z_1 + 2}{0.6}\right]^2}\right)$$

4.3. Lens Models

The experiments with lens models are primarily done with the FILVM model, as this model was showing better results, and we did not have sufficient time to preform a comprehensive comparison of the ILVM and FILVM. We do include a simple experiment using the ILVM to demonstrate its issue and how FILVM fixes this. We performed two types of qualitative experiments to evaluate how well aligned the learned latent space z^* was with the side information attributes. The first of these is a simple 2D visualization of held out samples (samples not used for training the model) in the 2 most relevant dimensions of z^*. To create these visualizations, we use the lens model to map the samples to z^*, and then create a scatter plot of these samples, using color to indicate the side information attribute of interest (see figures 7-9).

The second type of experiments are latent variable manipulation experiments. In these, we demonstrate that with the FILVM model we can manipulate specific dimensions in z^* to cause changes in x which correspond to the side information. We were only able to perform these using the FILVM and not the ILVM, as we could not get alignment between z^* and s using the ILVM. This point will be expanded on in the discussion section.

5. Results

5.1. Variational Autoencoder (VAE) Reconstructions

5.1.1. Image Data (MNIST & MNIST-R)

We found qualitatively that encoding an image and then decoding the encoding produced images which are quite similar to the original images, particularly when the images are not anomalous. This result held for both the MNIST and MNIST-R datasets (see figures 3 & 4). Quantitatively, we measured the mean squared error (MSE) between original and reconstructed images over all samples in the validation dataset, and found an MSE of 0.01904 on the original MNIST dataset, and 0.01879 on MNIST-R. For comparison, (Kormaris & Titsias, 2020) reported an MSE of 0.02033 on this task with the original MNIST dataset.

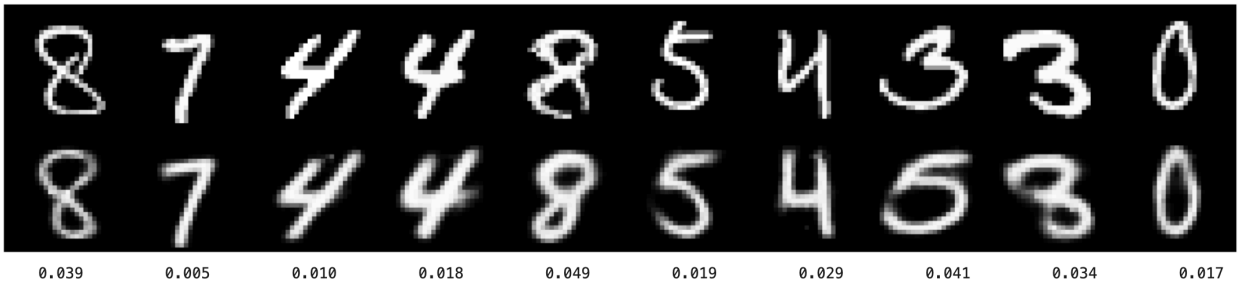

Figure 3: Example reconstructions of digits by a VAE trained on the original MNIST dataset. The top row contains the original images. The images in the second row are created by encoding and then decoding the original images with the trained VAE. Below the images are the mean squared errors between the original and reconstructed images. The pixel intensities are represented as numbers in the range [0.0, 1.0].

Figure 4: Example reconstructions of digits by a VAE trained on the MNIST-R dataset. The top row contains the original images. The images in the second row are created by encoding and then decoding the original images with the trained VAE. Below the images are the mean squared errors between the original and reconstructed images. The pixel intensities are represented as numbers in the range [0.0, 1.0].

5.1.2. Sequence Data

Our results here are similar to our results from the image domain. The VAE was able to learn to approximately reconstruct the original sequences. In figure 5 we present examples of the reconstructions achieved by our model.

```
956 ----> 941
985 ----> 969
1014 ----> 996
1043 ----> 1024
1072 ----> 1053
1101 ----> 1082
1130 ----> 1111
1159 ----> 1140
1188 ----> 1170
1217 ----> 1199
1246 ----> 1229
1275 ----> 1258
1304 ----> 1288
1333 ----> 1317
1362 ----> 1347
1391 ----> 1377
1420 ----> 1406
1449 ----> 1436
1478 ----> 1465
1507 ----> 1494

true step: 29.0
mean pred step 29.11
median pred step 29.00
True length: 20.0
Predicted length: 20.0
```

```
13 ----> 46
31 ----> 48
49 ----> 62
67 ----> 80
85 ----> 101
103 ----> 124
121 ----> 146
139 ----> 163
157 ----> 178
175 ----> 190
193 ----> 205
211 ----> 220
229 ----> 237
247 ----> 254
265 ----> 271
283 ----> 289
301 ----> 307
319 ----> 324
337 ----> 342
355 ----> 360
373 ----> 377
391 ----> 395
409 ----> 412
427 ----> 429

true step: 18.0
mean pred step 16.65
median pred step 17.00
True length: 24.0
Predicted length: 24.0
```

```
783 ----> 769
797 ----> 789
811 ----> 804
825 ----> 819
839 ----> 833
853 ----> 847
867 ----> 861
    ----> 874

true step: 14.0
mean pred step 15.00
median pred step 14.00
True length: 7.0
Predicted length: 8.0
```

```
99 ----> 73
105 ----> 85
111 ----> 99
117 ----> 114
123 ----> 124
129 ----> 131
135 ----> 137
141 ----> 142
147 ----> 147
153 ----> 152
159 ----> 158
165 ----> 163
171 ----> 168
    ----> 174

true step: 6.0
mean pred step 7.77
median pred step 6.00
True length: 13.0
Predicted length: 14.0
```

Figure 5: Example reconstructions of sequences by a VAE trained on the toy sequence dataset. In each example, the true sequence appears on the left and the predicted sequence on the right. This figure demonstrates the VAE has learned to approximate the true sequences, however it is not completely accurate. We found the VAE predictions tend to become closer to the true values the longer the sequence goes.

5.2. Normalizing Flows

In our density estimation experiments with normalizing flows, we fit flow models of various lengths to the unnormalized target distribution. As can be seen in figure 6, the shape of the high density regions more closely approximates the true density as the flow length increases. We found that increasing the flow length also lead to more instability in fitting the model. Figure 6 was created by taking the best model of each flow length out of 20 attempts, judged by the value of the objective function. In the case of length 32 flows, only one of the 20 attempts sucessfully terminated without producing nan values in the loss function. These nan values appear whenever the determinateof the jacobian of the transformation is 0.

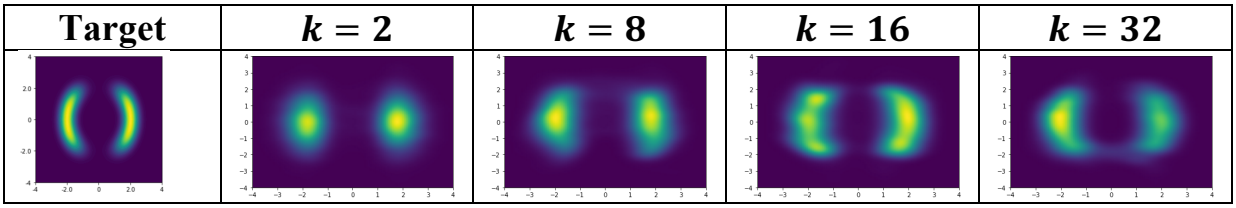

Figure 6: Density estimation with planar flows. The left most plot shows the target density function. Each of the subsequent plots shows the density of a flow model consisting of k planar flows, which has been fit to the target density using the reverse KL divergence.

5.3. Lens Models

[Note: I broke these down by type of experiment, instead of image vs sequence. I can do it the other way if that would be better. Or I could further break down these sections into image vs seq]

5.3.1. Latent Space Visualizations

The primary goal of these experiments is to visually verify that dimensions of interest in z^* align with the side information attributes. Of the results on working with image data (MNIST & MNIST-R), we found significantly better alignment using the original MNIST dataset with thickness as the secondary side attribute, than for the MNIST-R dataset with rotation as the second attribute (the digit label was the first attribute in both cases) [see figures 7 & 8]. We also see in figure 7 that the FILVM learns a more aligned z^* than the ILVM.

When working with the toy sequence dataset, we have four factors which can potentially be used as side information; the starting number, the final number, the step size and the sequence length. Notice that any 3 of numbers are sufficient to be able to recreate the entire sequence. In creating the dataset, we randomly independently selected the starting number, step size and sequence length, and computed the final number from the other three values (see section 2 of appendix for more information on the datasets). The FILVM model was fit using the starting number and step size as the two pieces of side information, however results from visualizing the latent space z^*

learned by the model show that all four factors are significant in the first two dimensions of z^* (see figure 9).

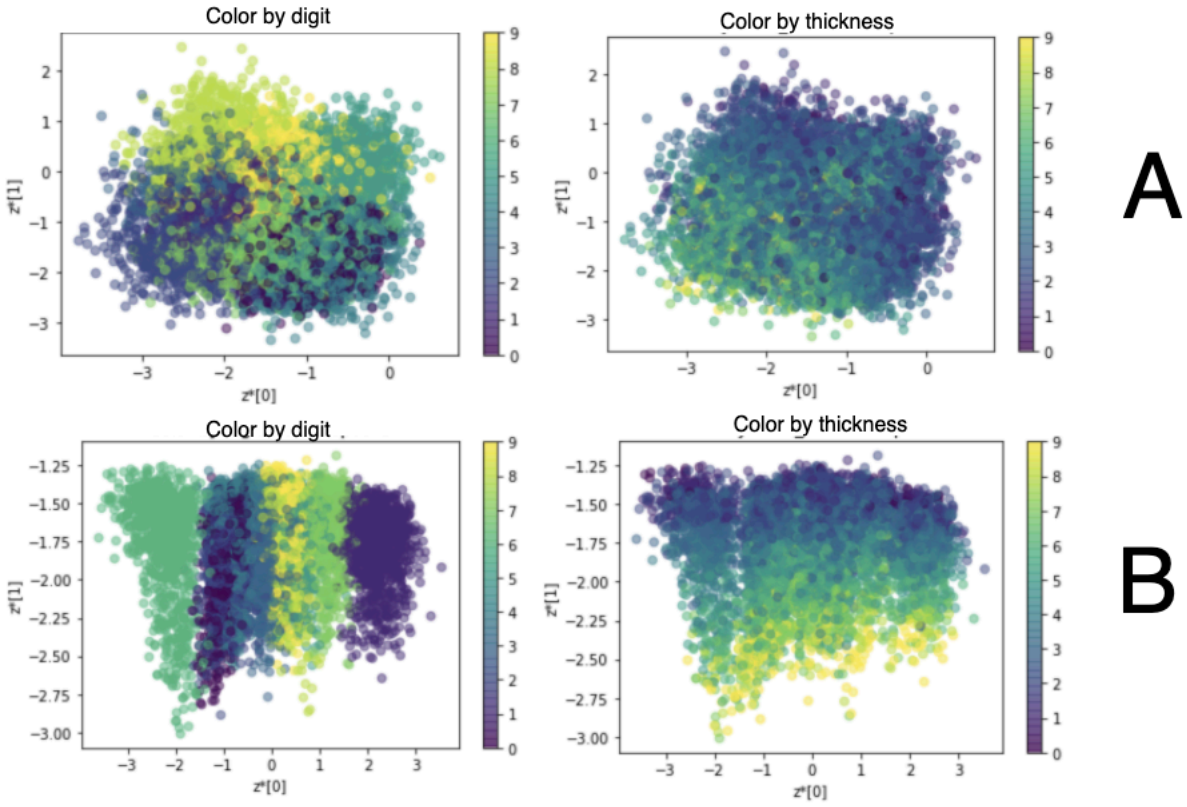

Figure 7: Latent space comparison of ILVM and FILVM. Each point represents the x, y coordinates of the encoding an image from the test set of MNIST inferred by the lens model. In (A) we see the ILVM has learned a latent space z^* which is organizes samples based on the attributes of digit label and thickness, however these attributes do no directly align with dimensions in z^*. In (B) we see the latent space learned by the FILVM has aligned the digit label with the dimension represented on the x-axis, and the thickness with the dimension on the y-axis.

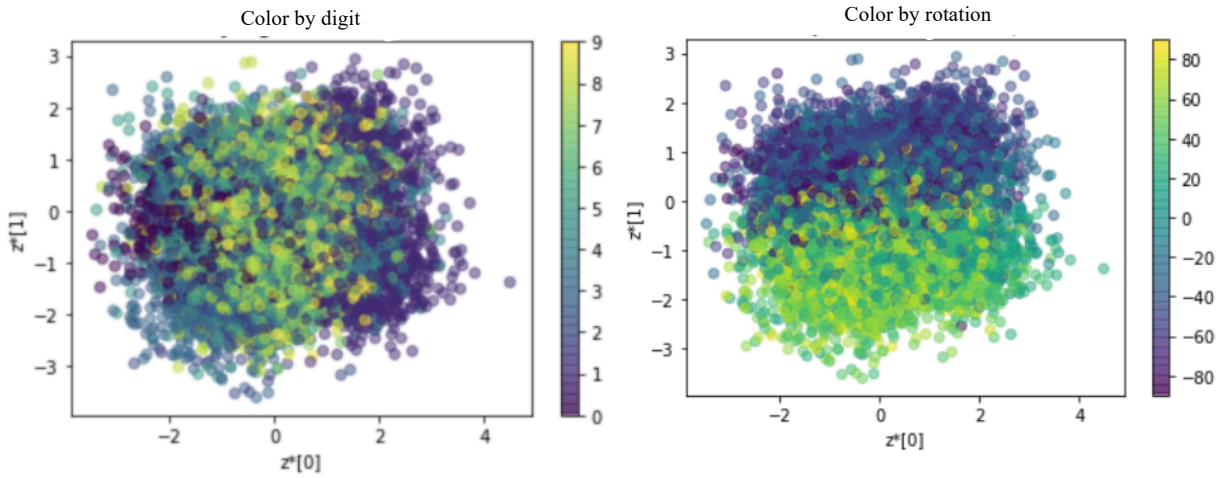

Figure 8: Latent space visualization of FILVM trained on the MNIST-R dataset. The left plot shows points from the test set of MNIST-R, plotted in the first 2 dimensions of z^*, and colored by digit label. There is some organization of samples along the x-axis ($z^*[0]$), however it is not as clear as the model using the basic MNIST. The right plot shows a more pronounced relationship between the y-axis ($z^*[1]$) and the rotation, with positively rotated images being encoded with negative $z^*[1]$ values, and vice versa. However, if we only consider positive samples with positive $z^*[1]$ (or negative), then the relationship between $z^*[1]$ and rotation appears less significant.

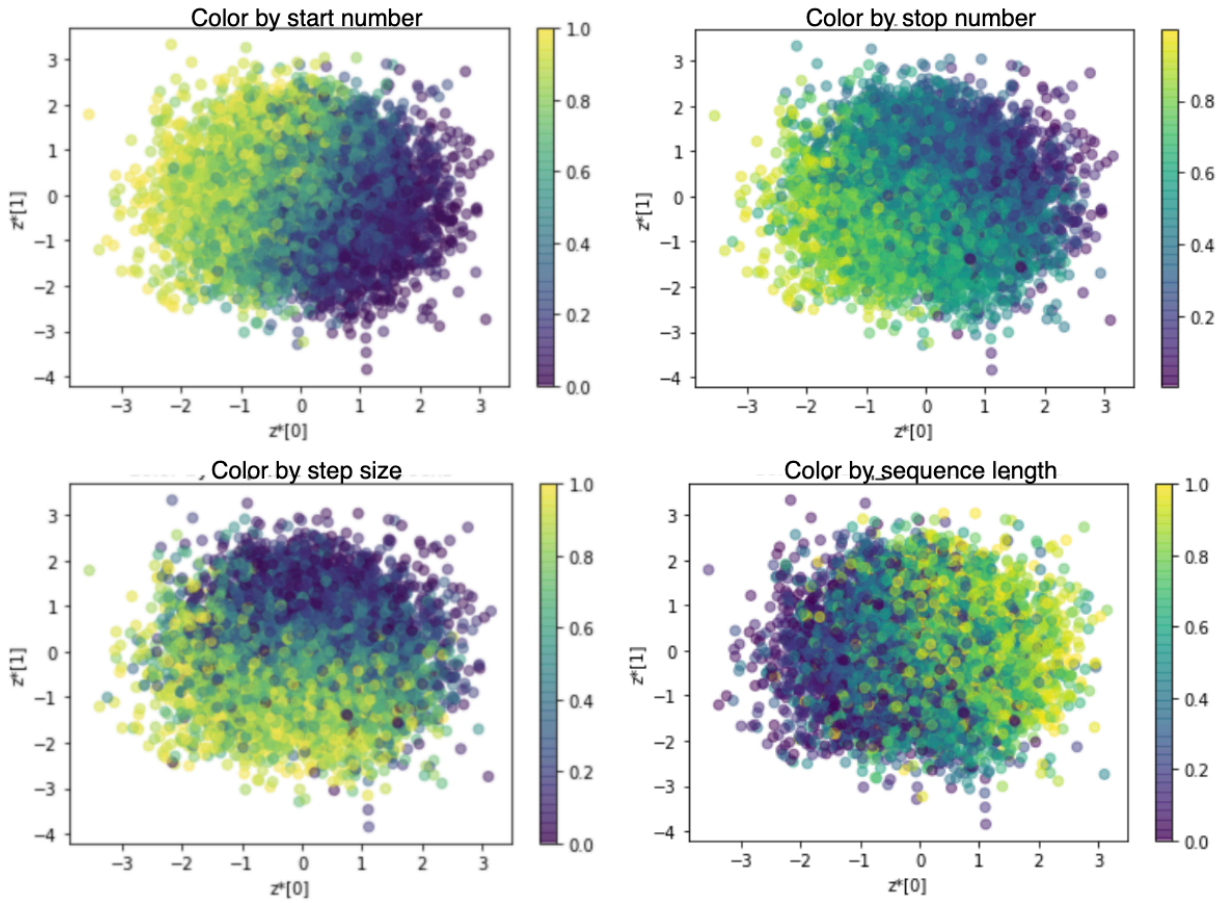

Figure 9: Latent space visualization of FILVM trained on the toy sequence dataset. The FILVM was fit using the sequence start and step size as side information. The plots on the left-side show that the FILVM learned a latent space where the first two dimensions of z^* are aligned with the side information used when fitting the model. The plots on the right-side show that the same dimensions of z^*also align with two other possible side information attributes, the sequence end value (stop) and length, despite the fact these were not used in training the model.

5.3.2. Latent Space Manipulations

In the latent space manipulation experiments, we demonstrate the effects of manipulating specific dimensions of z^* while keeping the others constant. We found the most success with these experiments using the MNIST dataset with digit label and thickness provided as side information. Figure 10 demonstrates that we are able to generate images of each digit, by taking a point in z^*, altering the value of the dimension related to digit label, and then transforming the new point into an image, by applying the generative model from z^* to z, and then the decoder to z to recover an image. We also see from figure 10 that by altering the dimension of z^* related to thickness, that we can generate a sequence of images varying in thickness, however our model was limited in the range of thicknesses it would generate, and did not generate extremely thin digits.

We also attempted this experiment with the MNIST-R dataset, using digit label and rotation as side information. As can mentioned in the latent space visualization experiments, the FILVM model trained on MNIST with thickness as side information does better at aligning its latent space with the side information. The latent space manipulation experiments with the MNIST-R dataset show there is some alignment between z^* and the side information, however it is not possible for our model to generate images of all digits with similar rotations by altering only one dimension of z^* (see figure 11).

Finally, we attempted latent variable manipulation experiments with the FILVM on the toy sequence data. As mentioned in latent space visualizations, we see the first two dimensions of z^* appear related to all four potential side information attributes. When attempting latent space manipulation experiments, we found that changes in either of the first two dimensions of z^* lead to sequences which were altered in all attributes (see figure 12). Interestingly, attempting to manipulate the remaining dimensions of z^* produced only minimal changes in all attributes. This suggests that most of the information about the sequences has been squeezed into the first two dimensions of z^*.

Figure 10: Visual results of manipulating interpretable features in the latent space z^* learned by the FILVM model on the standard MNIST dataset. The images in each row are generated by manipulating only the dimension of z^* most informative about the digit label, while keeping the others constant. Similarly, the images in each column are generated by only manipulating the dimension most informative about the thickness. Note: The order of images in the rows has been changed for visualization. A fuller picture of this latent space is available in the figure B3 in appendix B.

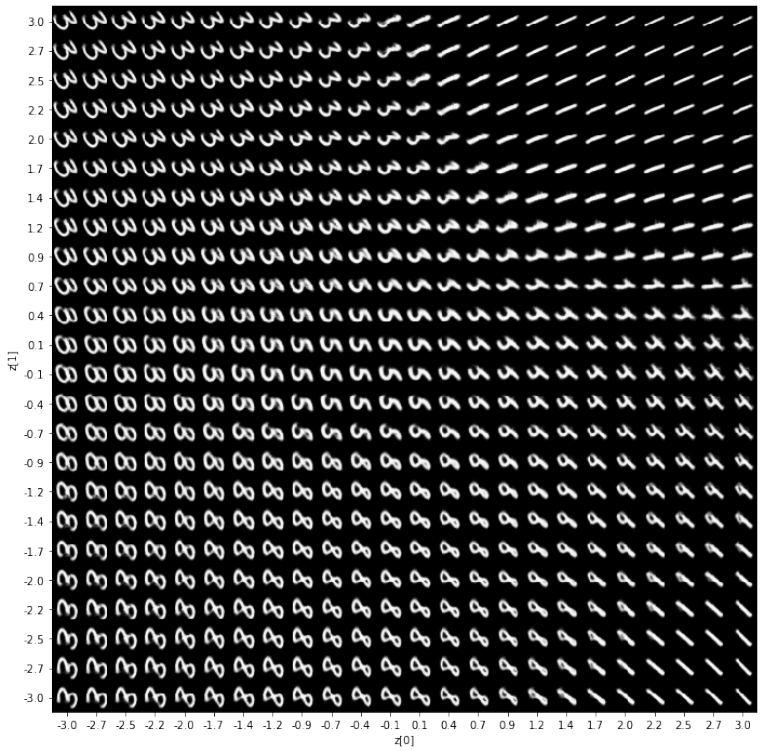

Figure 11: Visualization of z^* from FILVM trained on MNIST-R. The images in each row are generated by manipulating only the dimension of z^* most informative about the digit label, while keeping the others constant. Similarly, the images in each column are generated by only manipulating the dimension most informative about the rotation. In the column on the left side, we see the images almost all appear to be the digit 3, and that digits near the top and bottom of this column are similar but have opposite rotations. Throughout the plot we see images in the top half are nearly all rotated clockwise, and those in the bottom half rotated counterclockwise, however we see very few images with little to no rotation near the center, as we hoped to find. We can also see from this figure that as we move across the plot, the digit does change, but we do not see a full variety of digits in any row.

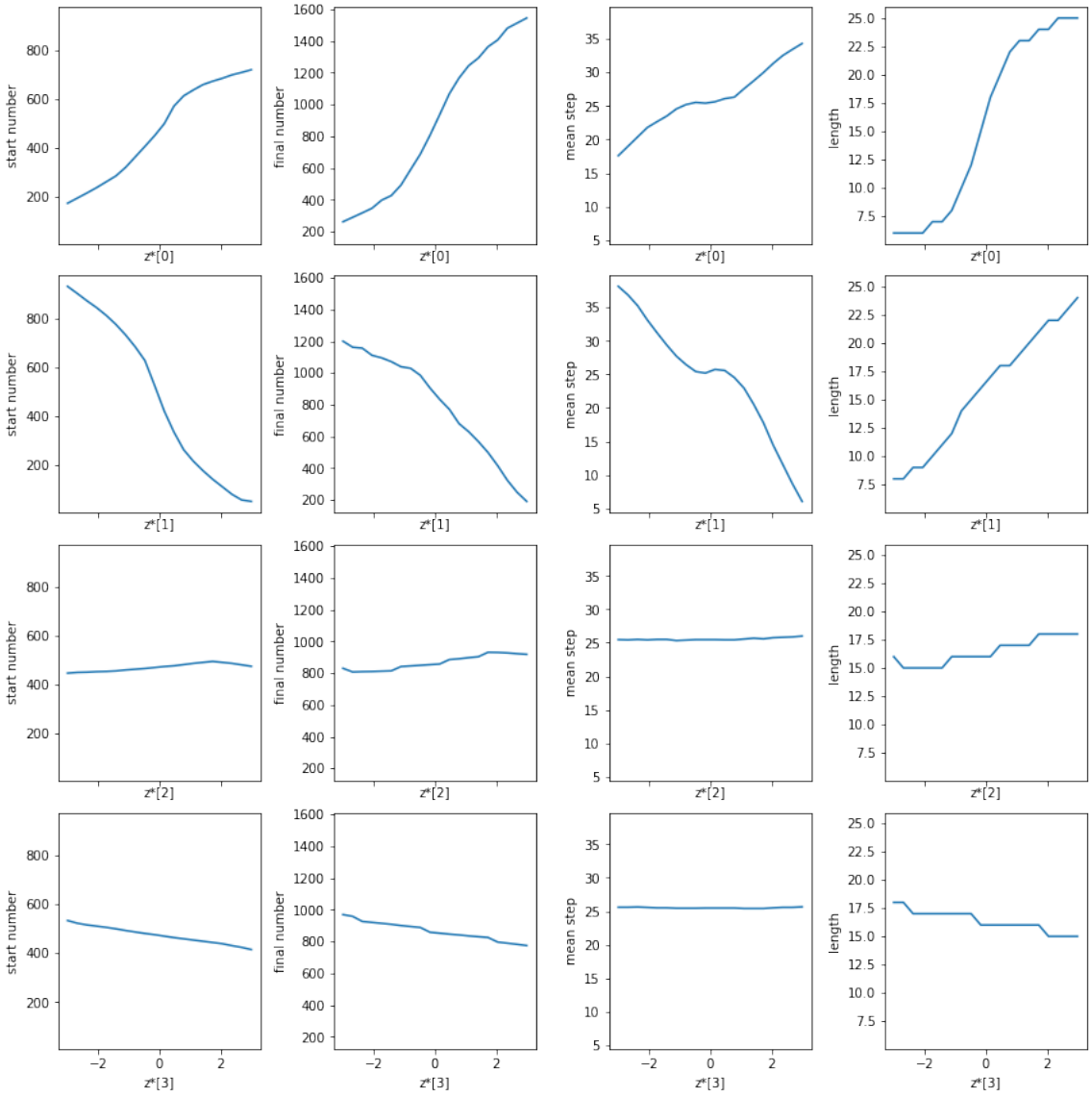

Figure 12: Plots of changes to side information attributes induced by manipulating latent dimensions of z^*. Each row represents the values of a single dimension of z^* which is being manipulated, while all other dimensions are held constant at 0.0. Each column represents one of the four potential side information attributes. Looking at the rows, we can see that manipulations in the first 2 dimensions have a greater impact on all four attributes than do manipulations of the remaining dimensions. Looking at the columns, we see that we can produce sequences spanning a wide range of attribute values for all attributes, by only manipulating either the first or second dimension. It's interesting to note that the first two plots in the left most column show that manipulating $z^*[1]$ effects the sequence start number more that $z^*[0]$, despite the model being trained to align this attribute with $z^*[0]$. The model was training to align $z^*[1]$ with the step size (3rd column), and we can see from the figure that manipulating $z^*[1]$ has a greater impact on the mean step size of the generated sequence than $z^*[0]$.

6. Discussion

The main focus of this project was to use the ILVM model to learn interpretable representations for complex data, including high dimensional image data, and variable length sequence data. More specifically, we aimed to learn latent representations where the dimensions of the representations correspond to interpretable aspects of the data. The paper which proposed ILVM (Adel, Ghahramani, & Weller, 2018) highlighted certain aspects of the model which made it appear desirable for this purpose, however after working through this project, the ILVM appears less useful than it did at the beginning.

One of the main purported benefits of ILVM is the fact that the transformation from z to z^* is invertible (Adel, Ghahramani, & Weller, 2018). Our initial assumption was that this meant it would be possible to map the encoding of an image (or sequence) in z to an encoding in z^*, manipulate the encoding in z^* in a meaningful way (such as changing its thickness) and then simply map the manipulated encoding back to z (where it can be decoded), by inverting the transformation used to get from z to z^*. To see why this cannot be done, consider the encoding of a sample z_i. To map this sample to the interpretable space, we first use the recognition model to compute the normalizing flows parameters η_i needed to map z_i to z_i^*. Once we have computed these parameters, we can map the encoding z_i from z to a corresponding encoding z_i^* in z^*, by applying the transformation $z_i^* = T(z_i; \eta_i)$, where T is a series of normalizing flows with parameters η_i. However, if we then manipulate the encoding z_i^* to get a new encoding $z_i^{*\prime}$, the parameters η_i will no longer be valid, as they are dependent on z_i, and $z_i^{*\prime}$ is not and encoding of z_i. To see this, if we use compute $z_i{}' = T(z_i^{*\prime}; \eta_i)$, and then use the recognition model to compute parameters $\eta_i{}'$ from $z_i{}'$, we will find that $\eta_i \neq \eta_i{}'$. In order to map $z_i^{*\prime}$ back to the corresponding $z_i{}'$, we need flow parameters $\eta_i{}'$ which we can only compute if we have $z_i{}'$. In our experiments on manipulating latent variables, we use the generative model $p_\theta(z|z^*)$ to recover an approximation $\tilde{z}_i$, which can then be decoded into an image by decoder. Using this approach, we do get a generative model which can map latent factors in the interpretable space z^* to the space of observations x, by combining the ILVMs generative model with the VAEs generative model. This was the approach taken for the FILVM as well.

Of the three datasets we tested the FILVM on, we found the experiments using the basic MNIST dataset with digit label and thickness produced the best results. One significant reason why this is likely to be the case is that due to time constraints, we spent much more time tuning the model to this problem that the other two. When tuning the FILVM, we experimented with a variety of neural network architectures for implementing the generative and recognition models, a variety of number of layers to use in the normalizing flows and optimization algorithms, as well as adjusting the constant β from **(8)** and the scaling of side information attributes. In addition to these factors, we also experimented with the size of the latent dimensions in the image VAEs [for the rest of the VAE architecture we followed the design (Adel, Ghahramani, & Weller, 2018) used on MNIST, however they do not mention the size of the latent dimension]. For the sequence data, we also had to design a sequence based VAE which was capable of modeling of the dataset, in addition to tuning the FILVM parameters. Because of time limitations, we only tuned the β parameter and side information scaling when using the FILVM with the MNIST-R

and toy sequence datasets, and used architectures and optimization setting which had work best with the original MNIST dataset.

We actually had more difficulty fitting a VAE to the toy sequence dataset then we had anticipated. The dataset was designed to contain sequences which are easy to model and can be encoded in a small number of features, independent of their length. We believe a primary source if this difficulty comes from the fact that our toy sequences are deterministic functions of the latent factors (starting number, step size and length), whereas the assumption in VAEs is that the observed data is a stochastic function of the latent factors. Because there is randomness in the sampling step of the VAE, it unlikely that the VAE will be able to exactly compress and then reconstruct sequences. This should not be too surprising, as even in the image domain the reconstructions are not identical to the originals. We can see this is in figures 3-5, which show the reconstruction quality of VAEs trained on the three datasets.

It was interesting to find that with the toy dataset, the FILVM appeared to do well at aligning the first two dimensions of z^* with the sequence start and step size in the latent space experiments (figure 9), however when we performed the latent variable manipulation experiments (figure 12), we saw that each of those dimensions impact both the sequence start and step size, in addition to the sequence end and length. This was surprising, as the sequence start, step size and sequence length were each selected independently, and thus should not be correlated. The sequence end was computed from the sequence start, step size and sequence length, so it may be correlated with these factors. A possible way to encourage the FILVM to better separate these factors would be too include a third side information attribute, the sequence length, when fitting the model. We considered making this change but did not have sufficient time.

7. Conclusion

The fact that we were not easily able to adapt the model which worked well on the basic MNIST dataset to either of the other datasets, make us question the usefulness of this as a general framework for learning interpretable representations. However, completing this project was a useful exercise in many respects. It has increased our understanding of import models in the field of probabilistic machine learning, including normalizing flows and variational autoencoders. We also gained experience in taking a theoretical model and developing it into a concreate implementation in TensorFlow. Overall this project has be a great opportunity to explore some of what is possible with modern methods in probabilistic machine learning, and we hope to continue this work, focusing on using generative models for producing synthetic data.

Appendix

A. Network architectures

A.1. VAE

A.1.1. MNIST & Rotated MNIST

We used nearly identical VAE architectures on both the MNIST and MNIST-R datasets, with the sole difference being the size of the latent dimension. The size of the latent dimension for the VAE used on MNIST was 8 and it was 12 in the version used on MNIST-R. The size of the latent dimension was selected using cross-validation to determine the size which produced the lowest loss. All other aspects of the architecture are following the design (Adel, Ghahramani, & Weller, 2018) reported using on the MNIST dataset.

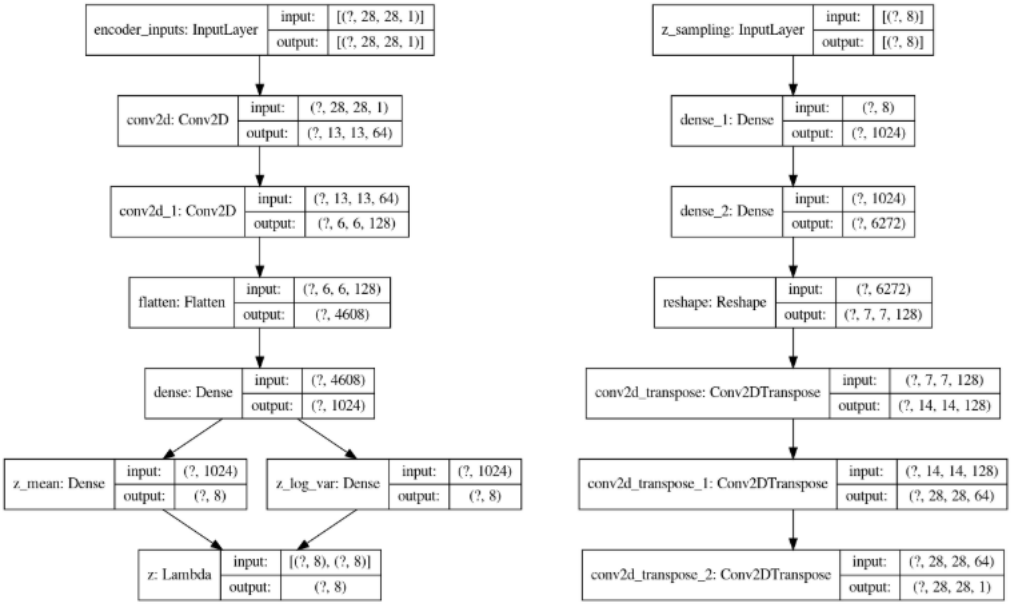

Figure A1: Architecture of VAEs used on MNIST dataset. The left side shows the encoder, and the right side is the decoder. The first number in the input and output shapes indicates the batch size, with a ? meaning the batch size may vary. The encoder maps batches of images with the shape (28, 28, 1) to batches of length 8 vectors. The input dimension is due to the fact MNIST digits are 28x28 and contain only one channel (as opposed to RGB images which have 3 channels). The output shape is determined by the latent dimension, which in this model is 8. The decoder attempts to reverse engineer the process of the encoder, and maps batches of length 8 vectors to images with shape (28,28,1). The VAE used for MNIST-R is identical to this, except the 8's which signify the latent dimension are 12's in the MNIST-R version.

A.1.2. Toy Data

The VAE architecture used in the toy data experiments, is designed to encode variable length sequences into fixed length output vectors. To accomplish this both the encoder and decoder networks contain Gated Recurrent Unit (GRU) layers, which are a type of recurrent neural network layers. In the encoder network (figure A2), the GRU takes sequences as input, and outputs its final activation after processing the sequence as well as its internal state. This information is then feed through densely connected layers to produce mean and variance vectors encoding the sequence. A sample is then drawn from a normal distribution with the given mean and variance, and the sampled vector is then feed to decoder network. The length of this vector is the size of the latent dimension, which was 4 in our experiments.

The decoder network (figure A3) is designed to take the sample drawn from the distribution output by the encoder and recover the original sequence. Whereas we only call the encoder network once to map our input sequences to fixed length vectors, we must call the decoder network once for each element in the output sequence. To determine how many steps to run the decoder for, we employ a secondary length decoder network (figure A4), which maps the encoded sequence to the length of the decoded sequence. At each timestep, the decoder network takes three pieces of information as inputs, the output vector from the encoder, the its previous output and its previous hidden state. During the first timestep, the previous input value is set to -1, which does not occur in the sequences, as a marker, and the hidden state is initialized to all zeros.

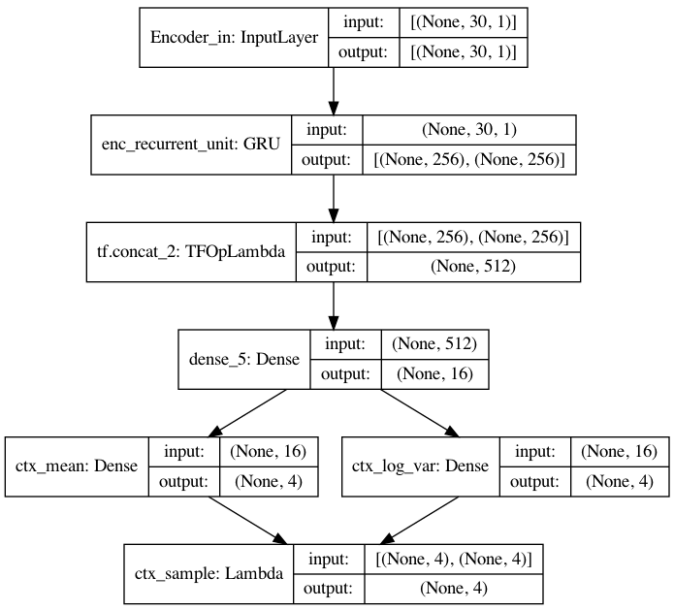

Figure A2: Architecture of the Encoder portion of VAE used on the toy dataset. Here the input shape represents (batch size, time steps, # of features), where None means the batch size may vary, 30 indicates the maximum sequence length and 1 indicates that there is only 1 feature, the number. Input sequences of lengths less than 30 were padded with zeros so the input shape was consistent across all samples. The two outputs from the GRU are its output and final state after processing the entire input sequence. The final output vectors are length 4, which is the latent dimension of the model.

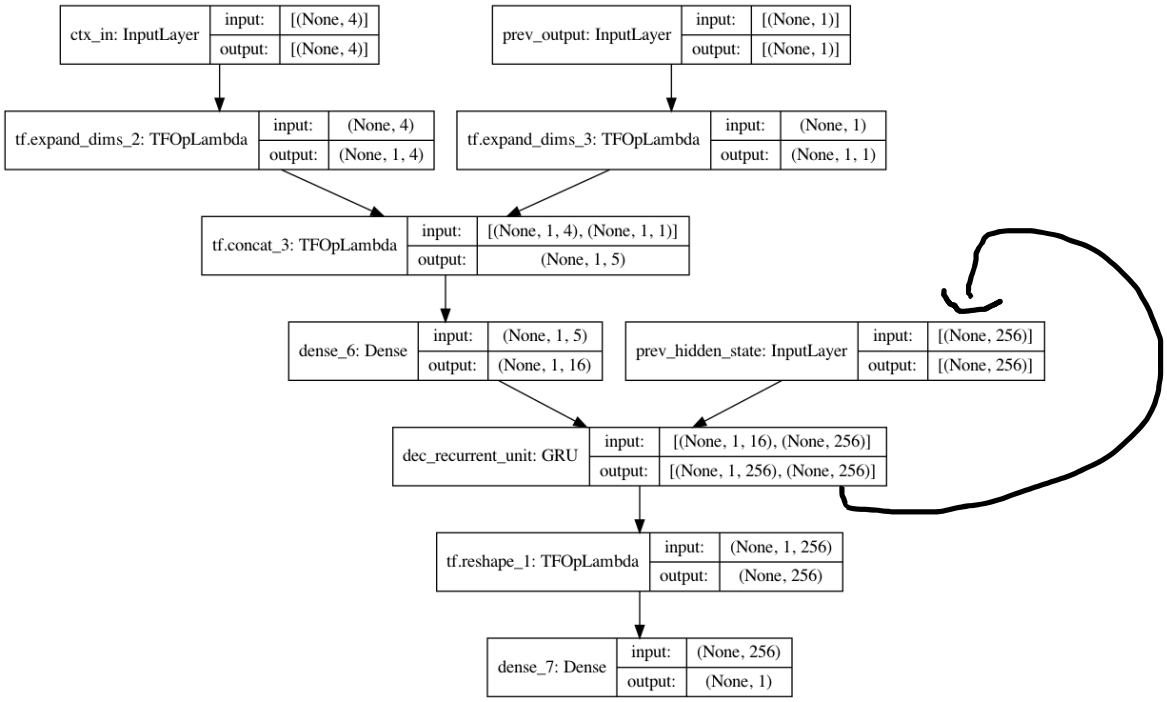

Figure A3: Architecture of the Decoder portion of VAE used on the toy dataset. In the decoder, the output from the encoder (ctx_in) is combined with the decoders last output and this combined information is fed to the GRU layer, along with the GRU hidden state from the previous time step. The output from the GRU is then processed by a final dense layer to produce a prediction about the next number in the sequence. The output shape is (None, 1) because for each sequence in the batch, the decoder predicts the single next number in the sequence.

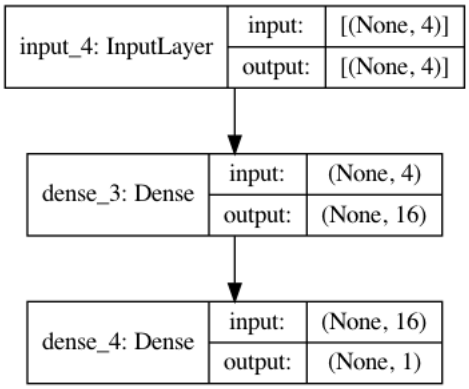

Figure A4: Architecture of the length decoder network. This network takes the information output by the encoder and determines the length of the sequence produced by the decoder (figure A3).

A.2. FILVM

In this section we detail the FILVM model architecture used in our experiments. In the FILVM, if we have samples (z) with side information (s), this information is combined and feed directly into the recognition model as the first step (figure A5). If we do not have side information, we first use the secondary recognition network to approximate the side information from the samples (figure A6), and then use this approximated side information along with the samples, as input to the recognition model. The output from the recognition network contains a sample z_0, along with parameters to a series of planar flows. The series of flows is then applied to z_0 to obtain z_t, which is a representation of the sample z in the space z^*. Finally, z_t is used as input to the generative model (figure A7), which attempts to generate z and s from z_t.

The diagrams below show the model used with the MNIST-R dataset but are nearly identical to the models used with the other two datasets. The only differences are in the size of the latent dimensions is different for each dataset, and for the toy dataset the dimension of the first side information is 1, whereas it is 10 in the image datasets. This is because the digit label is a discrete attribute, so we represent it using a one-hot encoding, whereas all the other side information attributes are continuous, and can easily be represented by one number.

A.2.1. Recognition Model

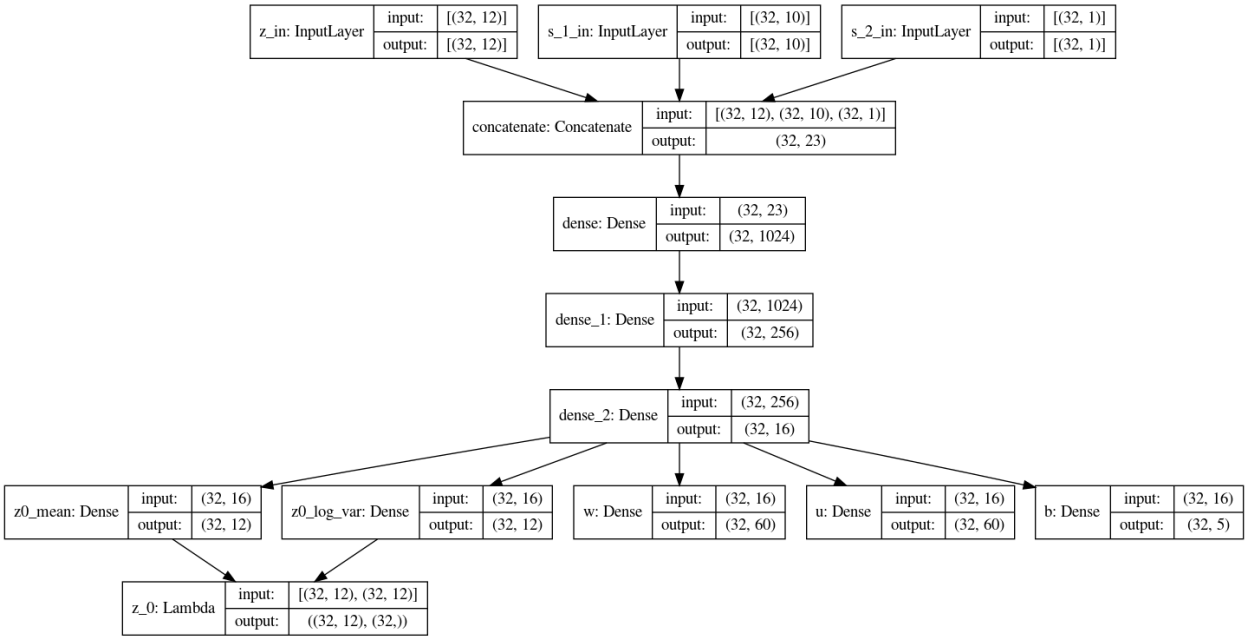

Figure A5: Architecture of the Recognition Model in FILVM. In this model, input information about the samples encoding and side information is combined, and feed through a deep densely connected neural network, which outputs parameters to a series of planar normalizing flows. These parameters include the mean and variance of the base distribution, as well as the parameters for each planar flow in the series. The output shapes of w, u are 60, because there are 5 planar flows used in this model, and each w_i, u_i have the same dimension as the latent dimension, which in this model is 12. The output shape of b is 5, because each b_i is a scalar. The output z_0 contains two parts, the first is the actual sample z_0 which has the same dimension as the latent dimension, and the second is the probability density of the sample, which is used for computing the loss. For technical reasons, the batch size here is fixed as 32, although this can be set to any sized batch in the code.

A.2.2. Secondary Recognition Model

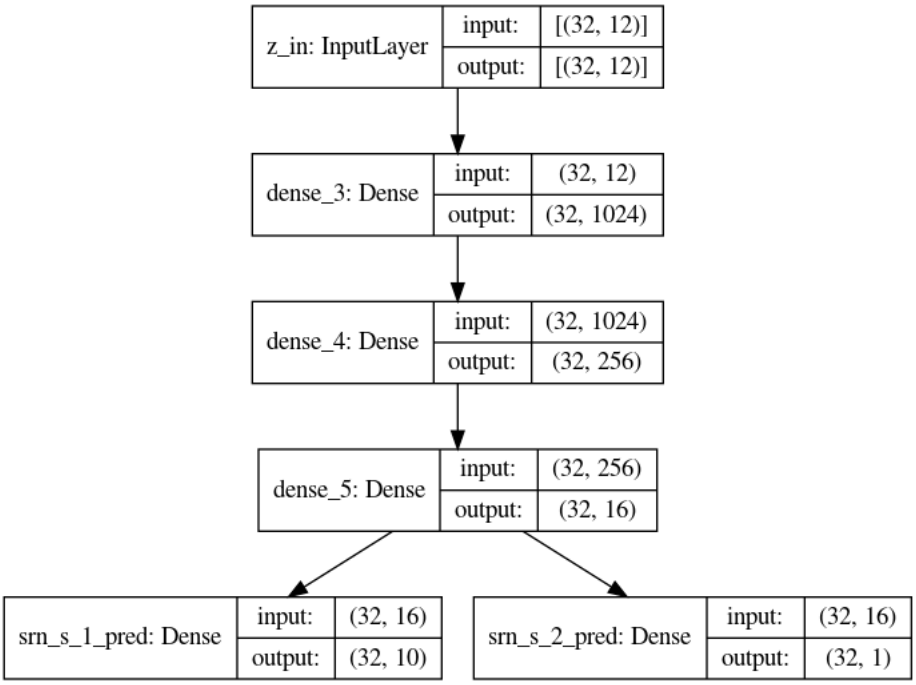

Figure A6: Architecture of the Secondary Recognition Model in FILVM. This is a simple deep, densely connected neural network which is designed to approximate the side information when it is not available. The input shape reflects the latent dimension, and the output shapes the dimensions of the side information. For technical reasons, the batch size here is fixed as 32, although this can be set to any sized batch in the code.

A.2.3. Generative Model

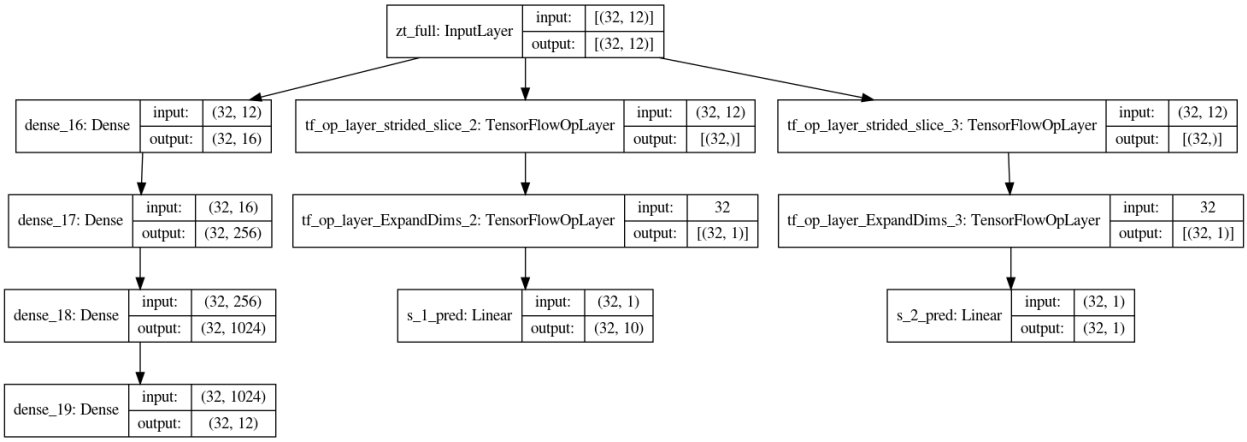

Figure A7: Architecture of the Generative Model in FILVM. The piece on the left is a simple deep, densely connected neural network which reconstructs z from z_t. The other branches are used to reconstruct the side information, each of these branches takes a single dimension of z_t, and tries to reconstruct a single piece of side information with a linear model. For technical reasons, the batch size here is fixed as 32, although this can be set to any sized batch in the code.

A.3. ILVM

The architecture of the ILVM is identical to the FILVM, with the exception of the generative model. In the ILVM, the full vector z_t is used to predict each piece of side information; whereas in the FILVM each piece of side information is predicted from only one dimension of z_t.

A.3.1. Generative Model

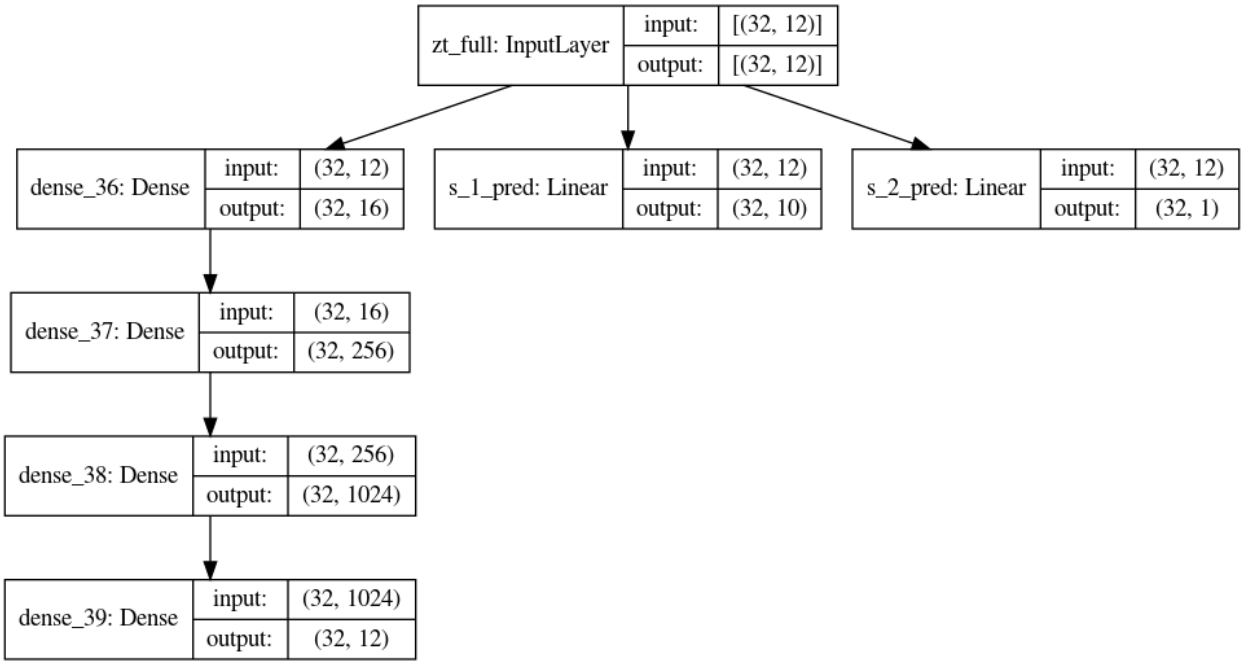

Figure A8: Architecture of the Generative Model in ILVM. As with the generative model of the FILVM (figure A7), this generative model aims to reconstruct z and the side information using z_t. The difference between this model and the model in figure A9, is here the linear models which predict the side information have access to the full vector z_t.

B. Supplementary Figures

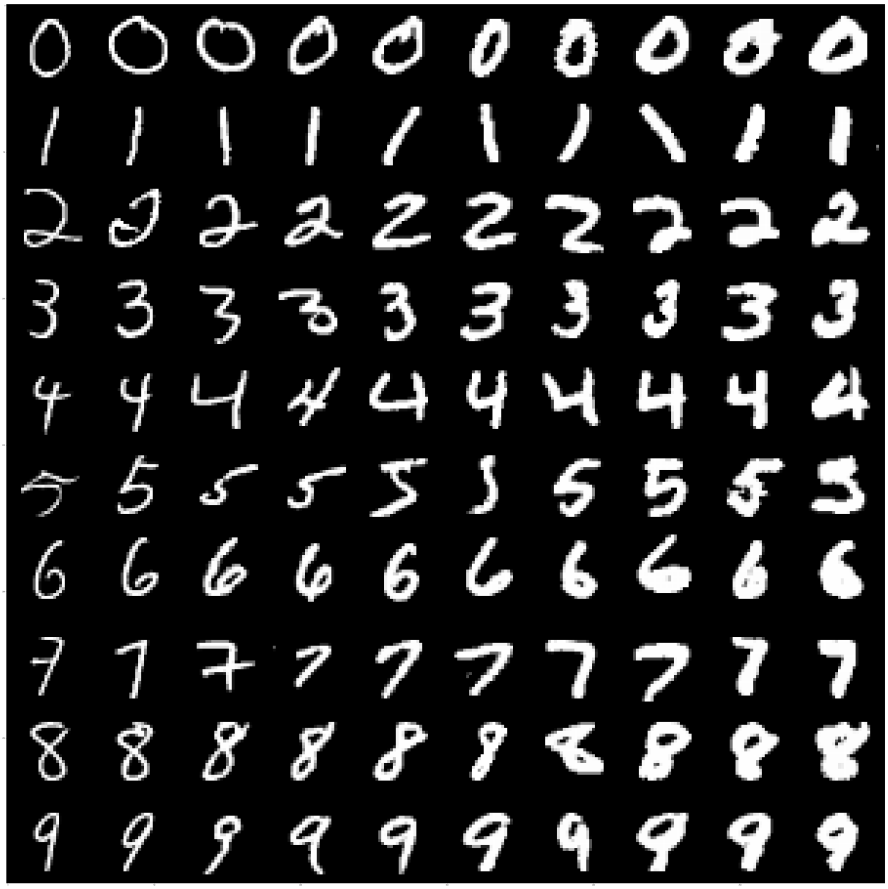

Figure B1: A random selection of digits from MNIST, sorted into columns by their computed thickness score. This shows that the computed thickness score is useful for determining the digit thickness and shows the variety of images containing in the MNIST dataset.

Figure B2: A random selection of digits from MNIST-R, sorted into rows by their rotation. This shows the variety of images containing in the MNIST-R dataset.

Start	Step	Length	Final	Sequence
4	10	7	64	4, 14, 24, 34, 44, 54, 64
123	25	9	323	123, 148, 173, 198, 223, 248, 273, 298, 323
1209	179	5	1925	1209, 1388, 1567, 1746, 1925

Table B1: Example of random samples from the toy dataset.

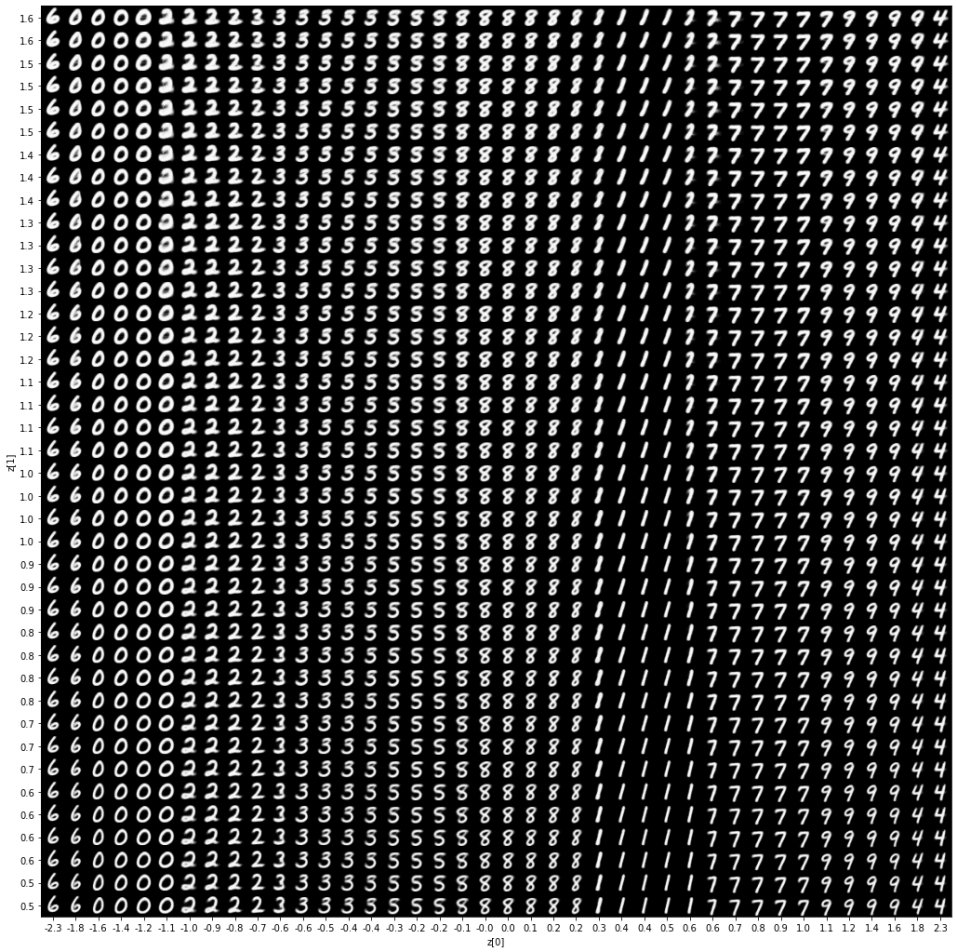

Figure B3: Full latent space of FILVM used to create Figure 10 in the main body of the report. Each digit is created by taking a vector z^* with its first two dimensions set to the x,y coordinates in this plot and the remaining elements set to 0, and then processing this z^* through the generative model to map it to an encoding z and using the decoder of a trained VAE to map the encoding z to an image.

```
Sequences generated by changing z*[0] from −3 to 3
_____________________________________________
174 198 214 230 246 262
218 244 263 282 301 321
268 296 319 340 362 383 404
333 363 389 414 438 461 485 508 531
428 458 488 516 541 566 591 615 640 664 688 712 736 759
543 571 603 631 657 683 709 735 760 786 811 837 862 887 912 936 961 986 1010
631 659 690 718 744 770 797 824 852 879 906 934 961 988 1014 1041 1068 1094 1121 1147 1174 1200
672 702 735 765 794 823 852 882 912 941 971 1002 1032 1061 1091 1121 1150 1179 1209 1238 1267 1296 1326 1355
699 731 767 800 831 863 895 928 960 993 1025 1058 1090 1123 1156 1188 1220 1253 1285 1317 1349 1381 1413 1445 1478
721 756 793 828 861 895 929 964 998 1032 1067 1101 1135 1170 1204 1238 1273 1307 1341 1375 1409 1443 1477 1511 1545


Sequences generated by changing z*[1] from −3 to 3
_____________________________________________
933 959 993 1034 1077 1119 1160 1200
871 897 930 967 1005 1043 1079 1115 1151
807 832 863 895 928 959 991 1022 1053 1084
720 747 776 805 832 859 886 913 940 967 993 1020 1047
595 623 652 679 704 728 753 777 802 826 850 874 899 923 946
375 404 435 464 491 517 542 568 593 618 643 667 692 716 740 764 788 812
230 252 278 302 326 350 374 398 422 446 470 494 517 540 563 586 609 632 654
151 163 181 199 218 237 256 275 294 313 332 352 371 390 409 428 446 465 483 502 520
85 92 105 116 126 137 148 160 172 185 198 211 223 236 249 262 274 286 298 310 322 333
52 43 41 44 48 55 61 68 75 83 91 100 109 120 133 144 153 160 167 172 177 182 187 191


Sequences generated by changing z*[2] from −3 to 3
_____________________________________________
448 477 508 536 562 587 612 637 662 686 710 734 758 782 806 830
453 482 513 541 566 591 616 641 665 690 714 738 762 785 809
456 485 516 544 569 594 619 644 668 693 717 741 765 788 812
462 491 522 550 576 601 626 650 675 699 724 748 772 795 819 842
469 498 529 557 583 608 633 658 683 707 731 756 780 803 827 851
477 506 537 565 591 616 641 666 691 715 740 764 788 812 836 859 883
486 515 547 575 601 626 651 676 701 726 750 775 799 823 847 871 895
495 524 556 584 610 636 661 686 711 736 761 785 810 834 858 882 906 930
489 518 550 578 605 630 656 681 707 732 757 782 806 831 855 879 903 928
476 505 538 566 593 619 645 670 696 721 746 771 796 821 845 870 894 918


Sequences generated by changing z*[3] from −3 to 3
_____________________________________________
534 563 594 622 648 674 699 724 749 774 799 823 848 872 897 921 945 969
516 544 576 604 630 655 680 705 730 755 780 804 829 853 877 901 925
505 534 565 593 619 644 669 694 719 744 768 793 817 841 865 889 913
491 520 552 579 605 630 655 680 705 730 754 778 803 827 850 874 898
479 508 539 567 593 618 643 668 693 717 742 766 790 814 838 861 885
467 496 527 555 581 606 631 656 681 705 730 754 778 802 825 849
455 485 516 544 570 595 620 645 669 694 718 742 766 790 814 837
445 474 505 533 559 584 609 634 659 683 707 731 755 779 803 826
432 461 492 520 546 572 596 621 646 670 694 718 742 766 790
415 445 476 504 530 555 580 605 629 654 678 702 726 750 774
```

Figure B4: Example sequences generated by altering dimensions of z^* in FILVM on toy sequence data. Each group of sequences is generated by taking a vector consisting of all zero's from z^*, manipulating a single dimension to values linearly spaced in the range -3 to 3. This figure shows that manipulating the first two dimensions of z^* has a much greater impact on all sequence attributes than manipulating the remaining two dimensions.

References

Adel, T., Ghahramani, Z., & Weller, A. (2018). Discovering Interpretable Representations for Both Deep Generative and Discriminative Models. *Proceedings of the 35th International Conference on Machine Learning*, 50-59.

Coelho de Castro, D., Tan, J., Kainz, B., Konukoglu, E., & Glocker, B. (2018). Morpho-MNIST: Quantitative Assessment and Diagnostics for Representation Learning. *arvix: availible at https://arxiv.org/abs/1809.10780*.

Kingma, D. P., & Welling, M. (2014). Auto-Encoding Variational Bayes. *arxiv*, http://arxiv.org/abs/1312.6114.

Kobyzev, I., Prince, S., & Brubaker, M. (2020). Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Kormaris, C., & Titsias, M. (2020). Variational Autoencoders & Applications. *Postgraduate Dissertation- availible https://www.researchgate.net/publication/337000568_Postgraduate_Thesis_-_Variational_Autoencoders*.

LeCun, Y., Cortes, C., & Burges, C. (2010). MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*.

Papamakarios, G., Nalisnick, E. T., Rezende, D. J., Mohamed, S., & Lakshminarayanan, B. (2019). Normalizing Flows for Probabilistic Modeling and Inference. *arxiv*.

Rezende, D., & Mohamed, S. (2015). Variational Inference with Normalizing Flows. *Proceedings of Machine Learning Research*.

Tomczak, J., & Welling, M. (2016). Improving Variational Auto-Encoders using Householder Flow. *arvix: availible https://arxiv.org/abs/1611.09630*.