CS6730 Probabilistic Graphical Models
Course Project
Variational Auto-Encoder

CS17S008 Nitesh Methani
CS17S013 Pritha Ganguly

April 27, 2018

Contents

List of Figures

1 Abstract

Probabilistic graphical models provide many tools to build structured representations for data, but often make rigid assumptions and may require significant feature engineering. Alternatively, deep learning methods allow flexible data representations to be learned automatically but may not directly encode interpretable or tractable probabilistic structure. Our aim for this project is to learn, understand and implement the general modeling and inference framework that combines these complementary strengths. Variational autoencoders are the perfect models for that. VAEs couple nonlinear likelihoods with structured latent variable representations. Thus VAEs enables the combination of flexible deep learning feature models with structured Bayesian priors.

2 Roadmap

We will start with the problem of fitting the distribution over a set of data-points. In Section 2 we will first try to motivate why we would want to fit a distribution by giving some practical examples. Then in Section 3 we try to list out few novel techniques and give the reasons why they won't work. After getting enough motivation for using VAEs, in Section 4 we describe the problem scenario in which VAEs can be implemented. In Section 5 we briefly describe several techniques we learned in the course to find which technique is suitable to address the challenges described in Section 4. With this background, we present two perspectives for VAEs, namely, Graphical Model Perspective (Section 6) and Deep Learning Perspective (Section 7). Section 8 describes how these two perspectives unite to form a single framework. To concretize the theory we did some experiments and reported the observations in Section 9. In Appendix, we compared FASHION MNIST dataset with MNIST dataset.

3 Why to model $\mathbb{P}(x)$?

1. **Generate new data**
   We might want to fit our natural image dataset into a probabilistic distribution to generate new data. In this example, fake celebrity faces are generated by using a generative model.[5].

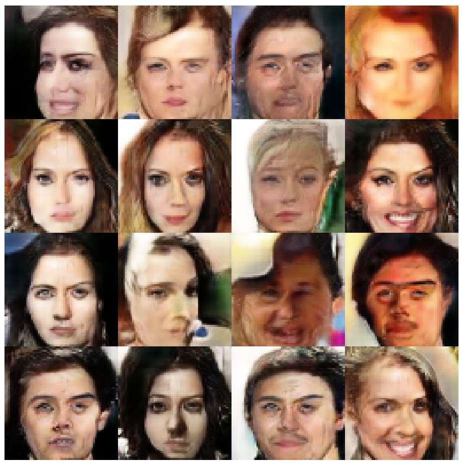


Figure 1: Fake celebrity faces generated using Energy-based GANs [5]

   In reference to the above example, if we have a model, $\mathbb{P}(x)$, we can do some kind of photoshop over the images. With a few brush strokes, we can change few pixels in our image and the program will try to recolor everything else, so it will change the color of the hair and etc. Therefore, there can be many interesting applications if we have a probability distribution of our training data.

2. **Detect anomalies and outliers**
   We can also try to detect anomalies if we have the distribution $\mathbb{P}(x)$. For example, given a sequence of bank transactions, for a new transaction one can predict how probable this transaction is according to the underlying model. If the transaction is not very probable

then it might get classified as a suspicious one and hence might require human intervention. Another example could be, if one has a security camera and if something suspicious happens then one can detect by seeing that some images from the camera have a low probability $\mathbb{P}$ of the image according to the model. So in this way, one can detect anomalies and suspicious behaviour.

3. **Work with missing data** Another reason to learn the distribution over the data is to handle missing data. For example, say there are some images with obscured parts, and we want to do predictions. In this case, if we know $\mathbb{P}(\mathbb{X})$ (probability of your data), it will help us greatly to deal with it.

4. **Represent data in a nice way**
   Another reason could be to represent highly structured data in low dimensional embeddings. The model will give a latent code to any object it sees, and then we can use this latent code to explore the space of our objects. For example, people sometimes build these kind of latent codes for molecules and then try to discover new drugs by exploring this space of molecules in this latent space.

4 How to model $\mathbb{P}(x)$?

1. $\mathbf{log\hat{P}(x) = CNN(x)}$
   The most natural approach is to use Convolutional Neural Networks (CNN) because it's something that works very well with images. Lets say CNN will take our image as input and return the the probability of this image. It's the simplest possible parametric model that returns the probability of any image. For numerical stability, we let CNN return the log probability.

$$log\hat{p}(x) = CNN(x)$$
$$p(x) = \frac{exp(CNN(x))}{\mathbf{Z}}$$

   The problem with this approach is that we have to normalize the distribution. We want the distribution to sum up to one, the sum with respect to all possible images in that space, and there are billions of them. So, this approach is infeasible because we can't compute normalization constant Z as it is very expensive to compute.

2. **Chain Rule**
   From probability and statistics, we know that any joint probability distribution can be written as the product of the conditional distributions.

$$p(x_1, x_2, \dots, x_d) = p(x_1)p(x_2|x_1)\dots p(x_d|x_{d-1}, \dots, x_1)$$

   We can also apply this idea to natural images. Say we have an image, in this case a 3×3 pixel image (ofcourse in reality we can have very high resolution images). We can enumerate the pixels of this image in some fashion, say row fashion.

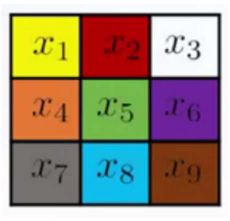


Figure 2: Sample Image

   We can now say that the distribution of this whole image is the joint distribution of these pixels and then can write it as the product of the conditional distributions as follows:

$$p(x_1, x_2, \dots, x_9) = p(x_1)p(x_2|x_1)\dots p(x_9|x_8, \dots, x_1)$$

Thus we can model these conditional probability distributions to model the overall joint distribution. And if the conditional model is flexible enough, we won't lose any information because they can represent, this way, any distribution.
First question that arises is how do we model these conditional distributions ? One natural choice is to use Pixel Recurrent Neural Networks.[2] It reads the image pixel by pixel and outputs the probability of the next pixel.
Now this approach is very easy to model because the normalization constant has to think about one dimension only. Its downside is that if we have to generate new images we have to do it pixel by pixel and it will be extremely slow if we want to generate high resolution natural images.

3. **Assume independence**
Another way is to assume that the distribution over pixels is independent i.e.

$$p(x_1, x_2, \dots, x_d) = p(x_1)p(x_2)\dots p(x_d)$$

But this is a very restrictive assumption to be made about the natural images. If we are provided the upper half of the image then we can easily guess the other half which implies that the pixels are not independent. For example, consider the hand-written MNIST dataset. If we train our model according to this assumption, we observe that the samples generated are noisy.

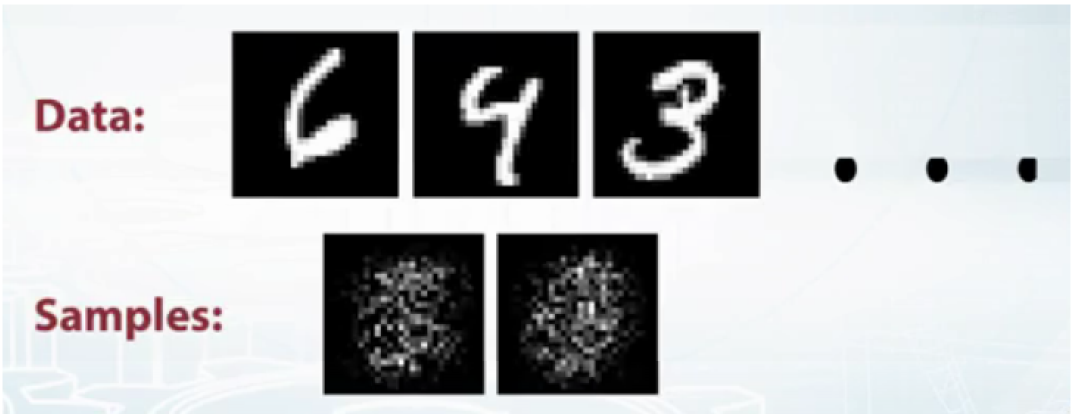


Figure 3: Samples generated from the model by making independence assumption [3]

This suggests that the independence assumption does not hold on true data.

4. **Gaussian Mixture model**
GMMs are really flexible and can model any distribution but we will be needing several thousands of gaussians. In this case, overall method fails to capture the distribution because it will be too hard to train.
We can modify this method by using **mixture of infinitely many gaussians**. The idea is that we will assume for each data-point x, there is a latent variable t associated with it and the data-point x is caused by this t. So we can marginalize out t as :

$$p(x) = \int p(x|t)p(x)$$

and we also assume that the conditional distribution $p(x|t)$ is gaussian. So we have to find mixture of infinitely many gaussians. For each value of t, we will have one gaussian and we mix them with weights.
However note that even though the gaussians are factorized (so they have independent components for each dimension), the mixture is not.

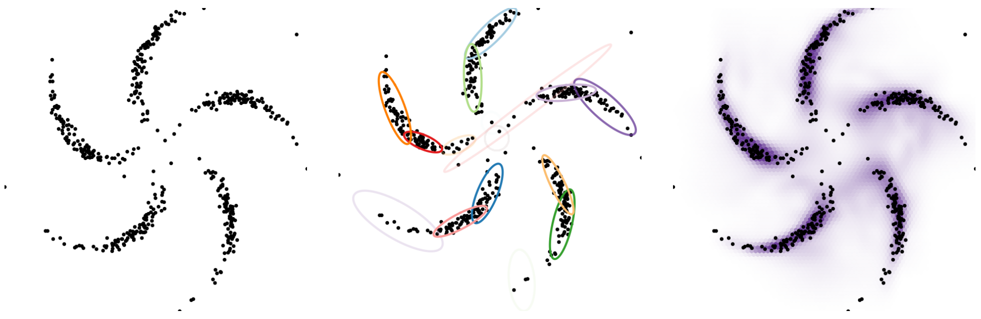

(a) Data (b) GMM (c) VAE

Figure 4: Comparison of generative models on spiral data [4]

Consider the problem of modeling the data $y = \{y_n\}_{n=1}^{N}$ shown in Fig. 3a. A standard approach to finding the clusters in data is to fit a Gaussian Mixture Model (GMM) with a conjugate pair. However, the fit GMM does not represent the natural clustering (Fig. 3b). Its inflexible Gaussian observation model limits its ability to fit the data and their natural semantics.
Instead of using GMM, a more flexible alternative would be a neural network density model (VAE). This model fits the data very well as we can see in Fig. 3c.
Given this motivation, we will now explore VAEs in detail.

5 Problem Scenario

Let us consider some dataset $\mathbb{X} = \{x_i\}_{i=1}^{N}$ consisting of N i.i.d samples of some discrete or continuous variable x. In this project, we have used MNIST images as our dataset. It is assumed that the data is generated by some random process, involving an unobserved hence latent continuous random variable z, such that latemt variables are associated per datapoint. In the problem definition, no common simplifying assumptions about the marginal or posterior probabilities is made. Therefore, the goal is to give a general algorithm that even works efficiently in the case of :

- Intractability : where the true posterior density $p_\theta(z|x)$ is intractable.
- A large dataset : where the amount of data is so much that batch optimization becomes costly. Therefore, the objective is to make parameter updates using small mini-batches or even individual data points.

Given the above scenario, the three related problems which can be solved using variational autoencoder are :

- Efficient approximation (using ML or MAP estimation) for the parameters θ, which will help to mimic the hidden random process and generate artificial data that resembles the real data
- Efficient approximation of posterior inference of the latent variable z given an observed value x for yhe choice of parameteres θ, which is useful for data representation tasks.
- Efficient approximation of marginal inference of the variable x to perform all kinds of inference tasks where a prior of x is required.

6 PGM Techniques used

We have learned several techniques in the course that could be used to solve our three tasks. They are :

1. The EM algorithm can be used to learn latent-variable models. However, performing the Expectation(E) step requires computing the approximate posterior $p(z|x)$, which we will see is intractable. In the Maximization(M) step, we try to learn the parameter θ by looking at the entire dataset, which would take up too much memory.

2. To perform approximate inference, mean field methods may be used. However, one step of mean field requires to compute an expectation whose time complexity scales exponentially with the size of the Markov blanket of the target variable, which will make the mean-field intractable.

3. Another approach would be to use sampling-based methods. In the paper, the authors compare against these kinds of algorithms, but they find that they dont scale well to large datasets. In addition, techniques such as Metropolis-Hastings require a hand-crafted proposal distribution, which might be difficult to choose.

7 Graphical Model Perspective

In this framework, a variational autoencoder contains a specific probability model of data x and latent variables z [8]. The joint probability of the model can be written as $p(x, z) = p(x|z)p(z)$. The generative process can be written for each datapoint i,

- A latent variable $z_i \sim p_{\theta^*}(z)$ is drawn.
- A value $x_i \sim p_{\theta^*}(x|z)$ is drawn, i.e x_i is drawn from some conditional distribution.

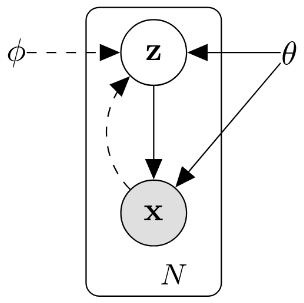


Figure 5: Graphical Model

It is further assumed that the prior $p_{\theta^*}(z)$ and the likelihood $p_{\theta^*}(x|z)$ are obtained from parametric families of distribution $p_\theta(z)$ and $p_\theta(x|z)$. The true parameters θ^* as well as the values of the latent variables z_i are unknown to us.
The goal of inference in this model is to infer good values of the latent variables given observed data, or to calculate the posterior $p(z|x) = \frac{p(x|z)p(z)}{p(x)}$. On close examination, it is found that the denominator $p(x)$ requires exponential time to compute as it needs to be evaluated over all configurations of latent variables. Therefore, we need to approximate this posterior distribution.

7.1 Auto-encoding variational Bayes(AEVB algorithm)

This algorithm can efficiently solve the three inference and learning tasks; the variational auto-encoder will be one instantiation of this algorithm. AEVB is based on ideas from variational inference, where we are interested in maximizing the evidence lower bound (ELBO):

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

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

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

To optimize over $q(z|x)$, this algorithm maximizes ELBO using gradient descent over ϕ. Furthermore, instead of doing only inference, it will perform learning via gradient descent on both ϕ and θ jointly. The gradient of the ELBO can be given as :

$$\nabla_{\theta,\phi}\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x, z) - \log q_\phi(z)]$$

Here, taking the gradient with respect to q is more difficult. Notice that we cannot swap the gradient and the expectation, since the expectation is being taken with respect to the distribution that we are trying to differentiate! One way to estimate the gradient, is to reformulate the above equation as :

$$\nabla_\phi \mathbb{E}_{q_\phi(z)}[\log p_\theta(x,z) - \log q_\phi(z)] = \mathbb{E}_{q_\phi(z)}[(\log p_\theta(x,z) - \log q_\phi(z))\nabla_\phi \log q_\phi(z)]$$

This identity places the gradient inside the expectation, which we may now evaluate using Monte Carlo but it suffers due to high variance i.e one has to take a very large number of samples to figure out that the true expectation is actually one.
The key contribution of the VAE paper is to propose an alternative estimator that is much better behaved. This is done in two steps: we first reformulate the ELBO so that parts of it can be computed in closed form (without Monte Carlo), and then we use an alternative gradient estimator, based on the so-called reparametrization trick.

7.2 The SGVB estimator

The reformulation of the ELBO is as follows :

$$\log p(x) \geq \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x,z) - KL(q_\phi(z|x)||p(z))]$$

Here x can be an observed data point. The LHS consists of two terms; both involve taking a sample $z \sim q(z|x)$, which can be interpreted as a code describing x. In the first term, $\log p(x|z)$ is the log-likelihood of the observed x given the code z that we have sampled. This term will be maximized when $p(x|z)$ assigns a high probability to the original x. It is trying to reconstruct x given the code z. The second term is the divergence between $q(z|x)$ and the prior $p(z)$, which will be fixed to be a unit Normal. It encourages the codes z to look Gaussian. It prevents $q(z|x)$ from simply encoding an identity mapping, and instead forces it to learn some more interesting representation. Thus, the optimization objective is trying to fit a $q(z|x)$ that will map x into a useful latent space z from which we are able to reconstruct x via $p(x|z)$.

7.3 Reparametrization trick

Optimizing the objective requires a good estimate of the gradient which is based on the reparametrization trick. Under certain mild conditions, we may express the distribution $q_\phi(z|x)$ as the following two-step generative process.

1. A noise variable ϵ is sampled from a simple distribution $p(\epsilon) \sim \mathcal{N}(0,1)$
2. A deterministic transformation $g_\phi(\epsilon, x)$ that maps the random noise into a more complex distribution $z = g_\phi(\epsilon, x)$

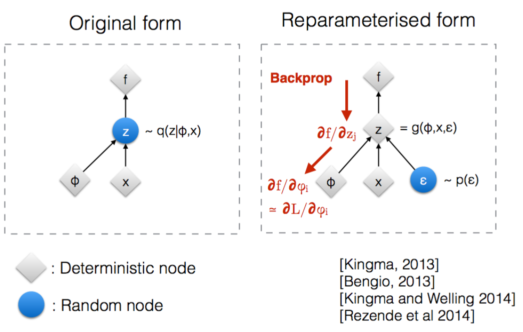


Figure 6: Reparametrization trick

For interesting classes of q_ϕ, it is possible to choose a $g_\phi(\epsilon, x)$ such that $z = g_\phi(\epsilon, x)$ will be distributed according to $q_\phi(z|x)$. Gaussian variables provide the simplest example of the reparametrization trick. Instead of writing $z = q_{\mu,\sigma}(z) = \mathcal{N}(\mu, \sigma$, we may write :

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

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:

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

The gradient is now inside the expectation and Monte Carlo may be used to get an estimate of the right-hand term. This approach will have a much lower variance.

7.4 Choosing p and q

The need arises to specify the exact form of the distributions q and p. The best $q(z|x)$ should be able to approximate the true posterior $p(z|x)$. Similarly, p(x) should be flexible enough to represent the richness of the data. For these reasons, p and q are parametrized by neural networks. These are extremely expressive function approximators that can be efficiently optimized over large datasets. This choice also draws a fascinating bridge between classical machine learning methods (approximate Bayesian inference in this case) and modern deep learning.

8 Deep Learning Perspective

In neural network language, a variational autoencoder consists of an encoder, a decoder, and a loss function [7].

8.1 Encoder

The encoder is a neural network. Its input is a datapoint x, its output is a hidden representation code z, and it has weights and biases ϕ. To be concrete, lets say x is a 28 by 28-pixel image of a handwritten digit. The encoder encodes the data which is 784 - dimensional into a latent (hidden) representation space z, which is much less than 784 dimensions. This is typically referred to as a bottleneck because the encoder must learn an efficient compression of the data into this lower-dimensional space. The encoder is denoted by $q_\phi(z|x)$. We note that the lower-dimensional space is stochastic: the encoder outputs parameters to $q_\phi(z|x)$ which is a Gaussian probability density. We can sample from this distribution to get noisy values of the representations z.

8.2 Decoder

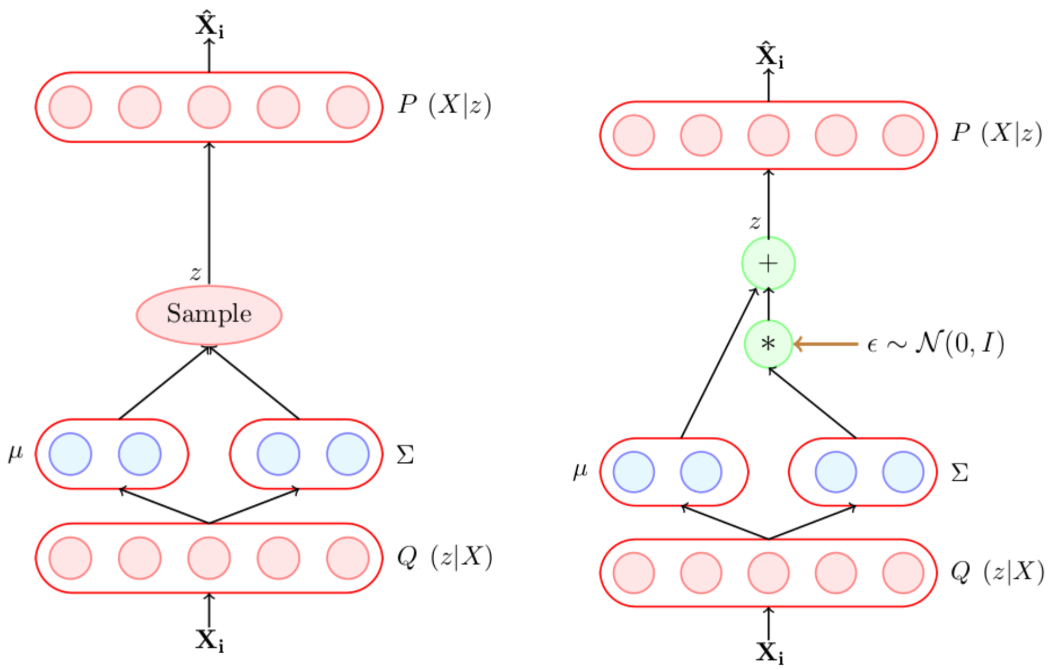
The decoder is another neural net. Its input is the representation z, it outputs the parameters to the probability distribution of the data, and has weights and biases θ. The decoder is denoted by $p_\theta(x|z)$. Running with the handwritten digit example, lets say the photos are black and white and represent each pixel as 0 or 1. The probability distribution of a single pixel can be then represented using a Bernoulli distribution. The decoder gets as input the latent representation of a digit z and output 784 Bernoulli parameters, one for each of the 784 pixels in the image. The decoder decodes the real-valued numbers in z into 784 real-valued numbers between 0 and 1. Information is lost because it goes from a smaller to a larger dimensionality. The measure of information loss is done using the reconstruction log-likelihood $\log p_\theta(x|z)$. This measure tells us how effectively the decoder has learned to reconstruct an input image x given its latent representation z.

8.3 Loss function

The loss function of the variational autoencoder is the negative log-likelihood with a regularizer. Because there are no global representations that are shared by all datapoints, the loss function can be decomposed into only terms that depend on a single datapoint l_i The total loss is then $\sum_{i=1}^{N} l_i$ for N total datapoints. The loss function l_i4 for datapoint x_i is :

$$l_i(\theta, \phi) = -E_{z\sim q_\phi(z|x_i)}[\log p_\theta(x_i|z)] + KL(q_\phi(z|x_i)||p(z))$$

The first term is the reconstruction loss, or expected negative log-likelihood of the i^{th} datapoint. The expectation is taken with respect to the encoders distribution over the representations. This term encourages the decoder to learn to reconstruct the data. If the decoders output does not reconstruct the data well, it will incur a large cost in this loss function. The second term is a regularizer that is throw in. This is the Kullback-Leibler divergence between the encoders distribution $q_\phi(z|x)$ and p(z).



(a) Without reparametrization (b) With reparametrization

Figure 7: Deep Learning Model [9]

8.4 Training

In the variational autoencoder, p is specified as a standard Normal distribution with mean zero and variance one, or $p(z) = \mathcal{N}(0, 1)$. If the encoder outputs representations z that are different than those from a standard normal distribution, it will receive a penalty in the loss. This regularizer term means keep the representations z of each digit sufficiently diverse. The variational autoencoder is trained using gradient descent to optimize the loss with respect to the parameters of the encoder and decoder ϕ and θ. For stochastic gradient descent with step size ρ, the encoder parameters are updated using $\phi \leftarrow \phi - \rho\frac{\partial l}{\partial \phi}$ and the decoder is updated similarly.

9 Marriage between Graphical Model and Deep Learning

In general, Data Modeling has two goals:

- To learn a flexible representation of complex high-dimensional data,such as images or speech recordings.
- To find structure that is interpretable and generalizes to new tasks.

Variational Auto-Encoders provide a single comprehensive framework to solve the above tasks by marrying the following two models [6]:

- Generative models where the data generation process is modelled
- Discriminative models, such as deep learning, where measurements are directly mapped to class labels.

Deep learning is particularly successful in learning powerful features from raw, unstructured data. Deep neural networks can effectively turn raw data streams into new representations that represent abstract, disentangled and semantically meaningful concepts. The downside of these models are to learn them one needs very large quantities of annotated data. Though they are flexible input-output mappings but they do not incorporate a very sophisticated inductive bias about the world. For example, given an image of a scene, deep learning models are able to efficiently segment out objects and classify them but they fail to infer a story about which events caused other events. This causal story is also a powerful tool to predict how the events may unfold into the future.

Therefore, to understand and reason about the world we need to find its causal atoms and their relationships. This is precisely what Bayesian networks were intended to do. Each random variable connects to other random variables and their directed relations model their causal relationships. Generative models can also be used for classification by inverting the relationship they model from class label to input features.

The variational auto-encoder (VAE) naturally combines generative models with discriminative models where the generative model can be a Bayesian network or a simulator and the discriminative model a deep neural network. The discriminative model performs inference of the unobserved (latent) variables necessary to perform the (variational EM) learning updates. In this view the discriminative model approximately inverts the generative model.

10 Observations

10.1 Visualisation of high-dimensional data

If we choose a low-dimensional latent space (e.g. 2D), we can use the learned encoders to project high-dimensional data to a low-dimensional manifold. We experimented with different dimensions for the latent space and the resulting images generated are plotted below.

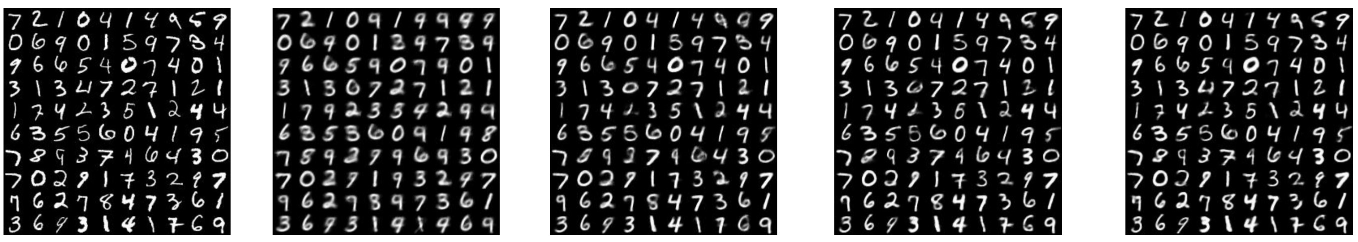

(a) Input Image (b) 2-D Latent Space (c) 5-D Latent Space (d) 10-D Latent Space (e) 20-D Latent Space

Figure 8: Random samples from learned generative models of MNIST for different dimensionalities of latent space.

We can see from the Figure 7 that as we increase the dimensions of the latent space, the images generated are more accurate. Fig 7 corresponds to the Fig 5 of original paper [1].

10.2 De-noising

When training, salt and pepper noise is added to input image, so that VAE can reduce noise and restore original input image.

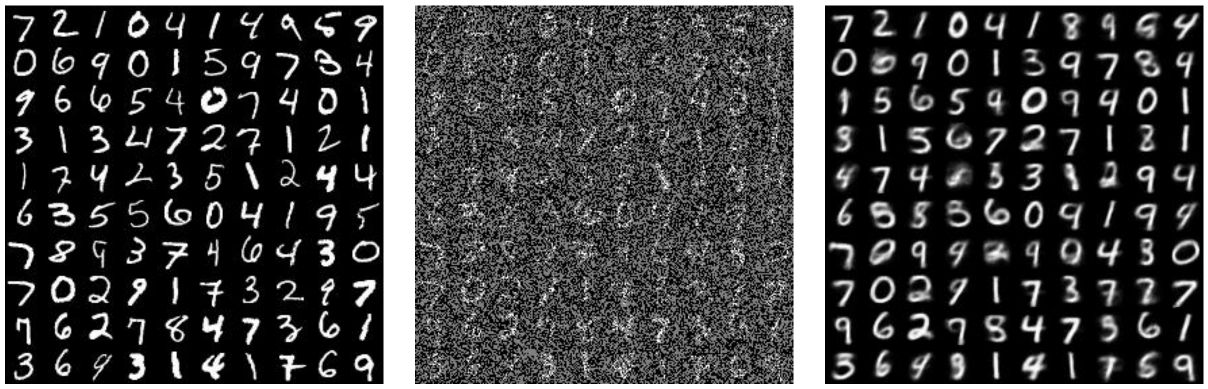
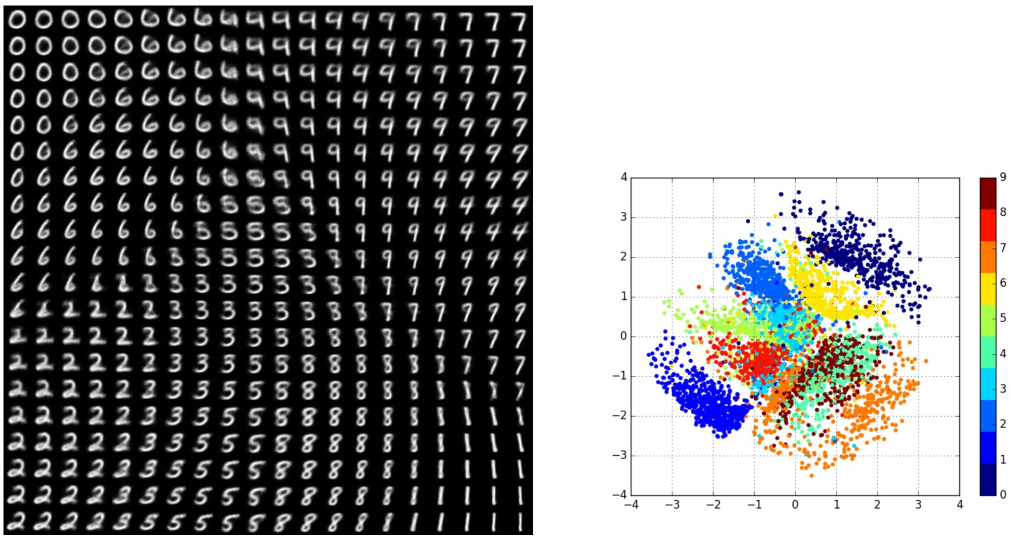

(a) Original Input Image (b) Noisy Input Image (c) Restored Image

Figure 9: VAE as denoising autoencoder

Even though the image input to the model is noisy, the reconstructed images has very low misclassification. Thus we can say that the model learns a very good abstract representation of the input and this abstract representation can be used to reconstruct the distorted images.

10.3 Learned MNIST manifold



(a) Learned MNIST manifold (b) Distribution of labelled data

Figure 10: Visualizations of learned data manifold for generative models with two-dimensional latent space, learned with AEVB.

In Figure 9, since the prior of the latent space is Gaussian, linearly spaced coordinates on the unit square were transformed through the inverse CDF of the Gaussian to produce values of the latent variables z. For each of these values z, we plotted the corresponding generative $p_\theta(x|z)$ with the learned parameters θ. Fig 9 corresponds to the Fig 4 of original paper [1].

11 Future Work

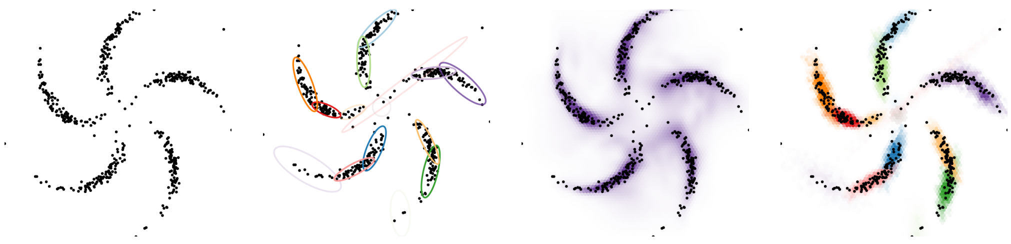

(a) Data (b) GMM (c) VAE (d) SVAE

Figure 11: Comparison of generative models on spiral data [4]

This model (VAEs) fits the data density well (Fig. 10(c)) but does not explicitly represent discrete mixture components, which might provide insights into the data or natural units for generalization. By composing a latent GMM with nonlinear observations, we can combine the modeling strengths of both the models, learning both discrete clusters along with non-Gaussian cluster shapes. This combination of flexibility and structure is shown in Fig. 1d.
Structured variational autoencoders provide a general framework that combines some of the strengths of probabilistic graphical models and deep learning methods. In particular, they use graphical models both to give models rich latent representations and to enable fast variational inference with CRF-like structured approximating distributions. To complement these structured representations,

SVAEs use neural networks to produce not only flexible nonlinear observation models but also fast recognition networks that map observations to conjugate graphical model potentials. [4]

12 Conclusion

In this project, we tried to use various ideas that we have learned in the class in order to present a very influential recent probabilistic model called the variational autoencoder. Variational autoencoders (VAEs) are a deep learning technique for learning latent representations. They have also been used to draw images, achieve decent results in semi-supervised learning.

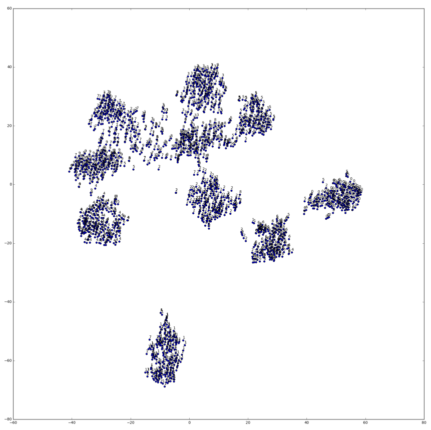
13 Appendix

13.1 Why not MNIST ?

A few reasons can be enumerated for choosing Fashion MNIST over MNIST [4].

- **MNIST is too easy** : Convolutional nets can achieve 99.7% on MNIST easily, and similarly, even classic ML algorithms can achieve 97%. **??**
- **MNIST is overused** : Almost everyone who has experience with deep learning has come across MNIST at least once.
- **MNIST cannot represent modern CV tasks** : This statement was noted by Franois Chollet, the author of the Keras library.

The following plot shows the t-SNE representation of MNIST and Fashion-MNIST dataset after 400 epochs.


(a) MNIST

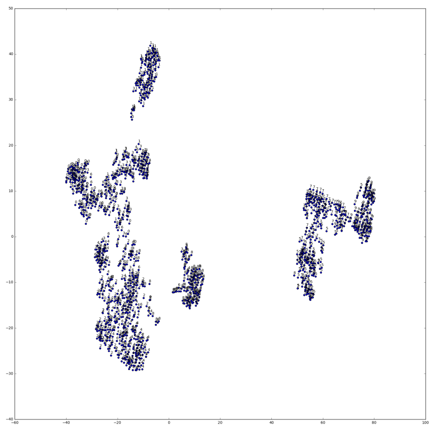

(b) Fashion-MNIST

Figure 12: Comparison between t-SNE representations

We can see that MNIST data is well clustered in just 400 epochs unlike Fashion-MNIST dataset. This observation justifies our claim. Therefore we experimented the Variational Auto-Encoder model with a more complex Fashion MNIST dataset.

13.2 Fashion MNIST

13.2.1 Visualisation of high-dimensional data

We experimented with different dimensions for the latent space and the resulting images generated are plotted below.

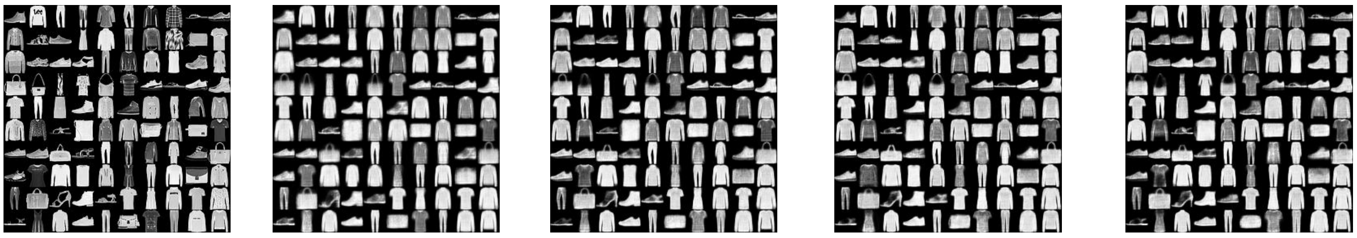

(a) Input Image (b) 2-D Latent Space (c) 5-D Latent Space (d) 10-D Latent Space (e) 20-D Latent Space

Figure 13: Random samples from learned generative models of Fashion MNIST for different dimensionalities of latent space.

13.2.2 De-noising

When training, salt and pepper noise is added to input image, so that VAE can reduce noise and restore original input image.

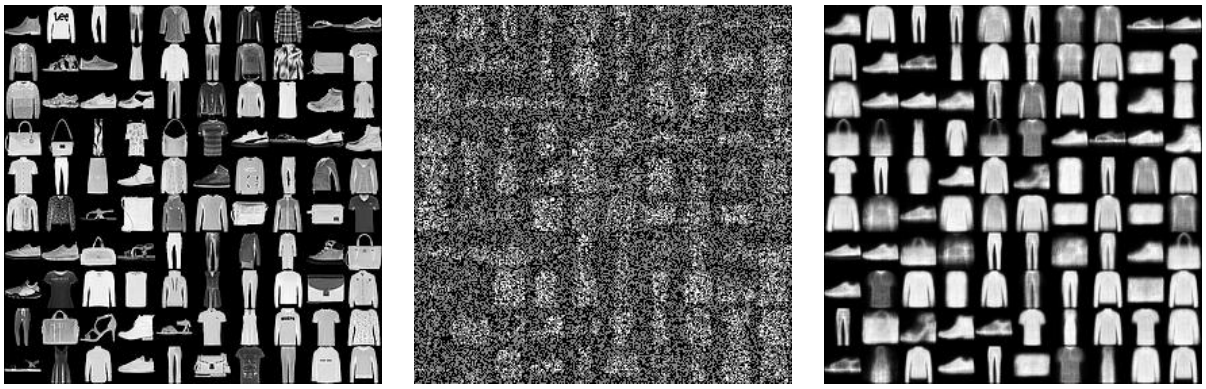
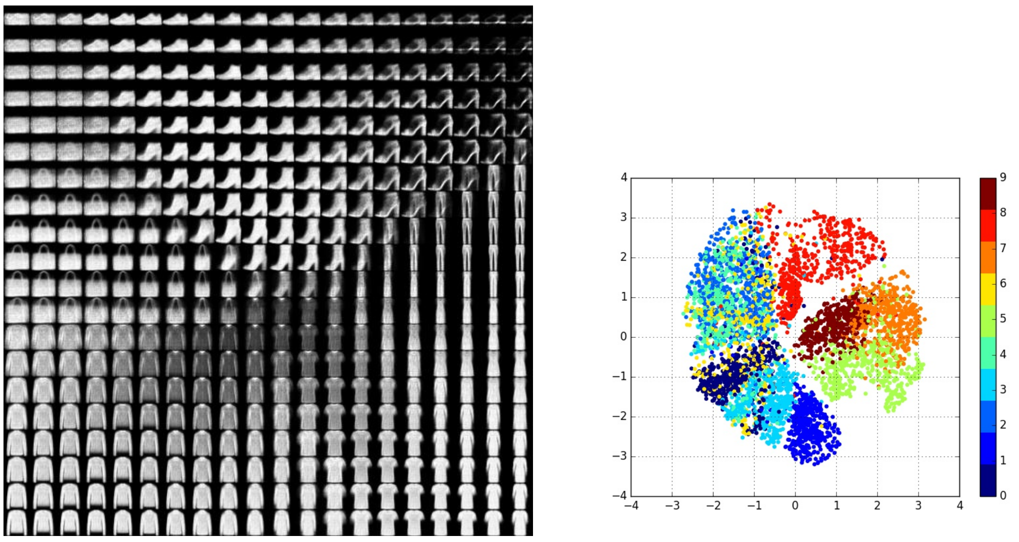

(a) Original Input Image (b) Noisy Input Image (c) Restored Image

Figure 14: VAE as denoising autoencoder on FashionMNIST

13.2.3 Learned Fashion MNIST manifold



(a) Learned Fashion MNIST manifold (b) Distribution of labelled data

Figure 15: Visualizations of learned data manifold for generative models with two-dimensional latent space, learned with AEVB.

References

[1] Auto-Encoding Variational Bayes
https://arxiv.org/pdf/1312.6114.pdf

[2] Pixel RNN
https://arxiv.org/abs/1601.06759

[3] Bayesian Statistics
https://www.coursera.org/learn/bayesian-methods-in-machine-learning/lecture/KaGAF/modeling-a-distribution-of-images

[4] SVAEs
https://arxiv.org/pdf/1603.06277.pdf

[5] Energy-based Generative Adversarial Network
https://arxiv.org/pdf/1609.03126.pdf

[6] Marriage between PGM and DL
https://ercim-news.ercim.eu/en107/special/marrying-graphical-models-with-deep-learning

[7] VAE tutorial
https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

[8] PGM notes *https://ermongroup.github.io/cs228-notes/extras/vae/*

[9] CS7015 DL *https://www.cse.iitm.ac.in/course_details.php?arg=MTE1*