CS274b Project Report

Understanding Variational Auto-Encoder

Peiyun Hu
Computer Vision Group
University of California, Irvine
peiyunh@ics.uci.edu

Abstract

Auto-Encoder (AE) is widely used for representation learning for many applications. However, how to interpret it remains questionable. Variational Auto-Encoder (VAE), as a directed latent variable graphical model, comes with a probabilistic interpretation. VAE has shown impressive performance on generative image modeling in the past few years. In this paper, we will discuss the formulation of VAE, the implementation of VAE, and the visualization of VAE's latent space.

1 Introduction

Auto-Encoder (AE) [2] is widely used for learning latent representation from data. Variants of AEs were proposed such as Sparse Auto-Encoder[4] and Denoise Auto-Encoder[6]. However, how to interpret these models remain questionable.

Variational Auto-Encoder (VAE) [4], as a directed graphical model, provides such a probabilistic way of interpreting what the model learns. It assumes data is generated with a conditional distribution given some latent variable. Therefore, we can interpret the latent space by first sampling the latent variable and then draw samples of data following the conditional distribution.

In this paper, we first introduce the formulation of VAE Sec. 2. Then, in Sec. 3, we talk about implementation and experiments. Finally, we visualize the latent space learned by VAE from various perspectives.

2 Variational Auto-Encoder

Given a dataset $\mathbf{X} = \{\mathbf{x}^{(i)}\}$, where $i = 1, \ldots, N$, we assume data $\mathbf{x}^{(i)}$ is generated by a two-step process: (1) $\mathbf{z}^{(i)} \sim p_\theta(\mathbf{z})$; (2) $\mathbf{x}^{(i)} \sim p_\theta(\mathbf{x}|\mathbf{z})$. For learning, we have to estimate the posterior density $p_\theta(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})/p_\theta(\mathbf{x})$, which is often intractable. To address the intractability, we introduce a proposal distribution $q_\phi(\mathbf{z}|\mathbf{x})$ to approximate the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$. We illustrate the process as a directed graphical model in Fig 1. Below, we will introduce how to learn ϕ and θ jointly.

The marginal likelihood over $\mathbf{X}$ can be decomposed into individual data points. For each data point, we can write

$$\log p_\theta(\mathbf{x}^{(i)}) = D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z}|\mathbf{x}^{(i)})) + \mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) \tag{1}$$

where,

$$\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}[-\log q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) + \log p_\theta(\mathbf{x}^{(i)}, \mathbf{z})] \tag{2}$$

Since KL divergence is non-negative, the second RHS term can be viewed as a lower bound on the marginal likelihood of data point i

$$\log p_\theta(\mathbf{x}^{(i)}) \geq \mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) \tag{3}$$

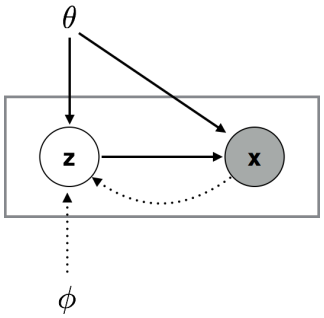


Figure 1: Directed graphical model under consideration

Instead of maximizing marginal likelihood $\log p_\theta(\mathbf{x}^{(i)})$, now we maximize its variational lower bound $\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)})$ w.r.t. ϕ and θ. We can further break down $\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)})$ as

$$\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}[-\log q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) + \log p_\theta(\mathbf{z})] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})] \tag{4}$$

$$= -D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})] \tag{5}$$

The first RHS term in Eq. (5) can often be analytically integrated with proper choice of prior $p_\theta(\mathbf{z})$. To do so, we first reparameterize z by introducing a differentiable transformation $g_\theta(\epsilon, \mathbf{x})$ with a noise random variable ϵ so that

$$z = g_\phi(\epsilon, \mathbf{x}^{(i)}), \epsilon \sim p(\epsilon) \tag{6}$$

Specifically, [4] uses

$$z^{(i,l)} = g_\phi(\epsilon, \mathbf{x}^{(i)}) = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)}, \epsilon^{(l)} \sim \mathcal{N}(0, 1) \tag{7}$$

where l means l-th sample of noise ϵ, i-th means i-th data point, and $\odot$ denotes element-wise product. $\mu^{(i)}$ and $\sigma^{(i)}$ are outputs of non-linear mapping (*encoder*) from $x^{(i)}$. Non-linear mapping is implemented with MLP and discussed in detail in Sec. 3.

According to Eq. (7), we have

$$q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) = \mathcal{N}(\mathbf{z}; \mu^{(i)}, \sigma^{2(i)}\mathbf{I}) \tag{8}$$

Meanwhile, we assume the prior over latent variables z are centered isotropic multi-variate Gaussian $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, \mathbf{I})$. Therefore, we can derive the analytic form of the KL divergence term in Eq. 5 as

$$-D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})) = \frac{1}{2}\sum_{j=1}^{J}(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2) \tag{9}$$

The second RHS term in Eq. (5) can be estimated by sampling. We implement $p_\theta(\mathbf{x}^{(i)}|\mathbf{z})$ with another MLP (*decoder*). Implementation details are discussed in Sec. 3.

Finally, we have a lower bound that can be easily maximized using stochastic gradient descent, which was referred to as Auto-Encoder Variational Bayes (AEVB) in [4]:

$$\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) = \frac{1}{2}\sum_{j=1}^{J}(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2) + \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)}) \tag{10}$$

3 Experiment

As mentioned in Sec. 2, VAE mainly contains two parts: encoder and decoder. Both encoder and decoder are implemented via MLP. Depending on the type of data, decoder has different types of outputs. We focus on binary data in this paper, which means decoder outputs a multi-variate Bernoulli distribution. We illustrate the implementation in Fig. 2.

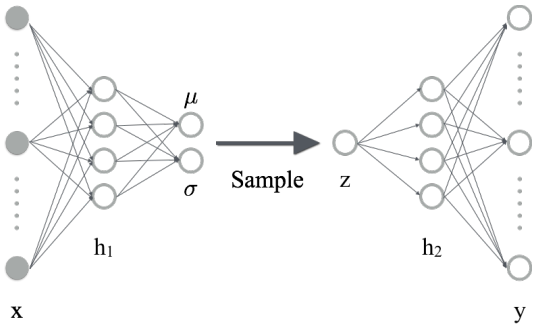
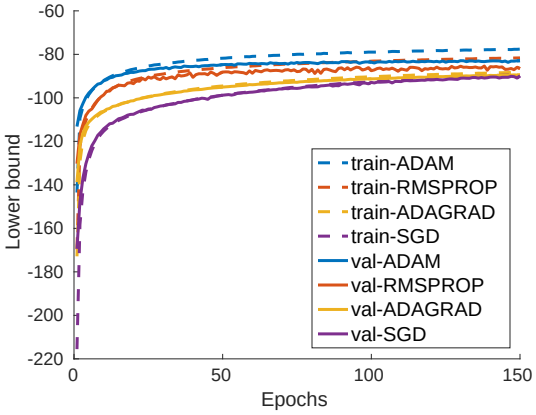
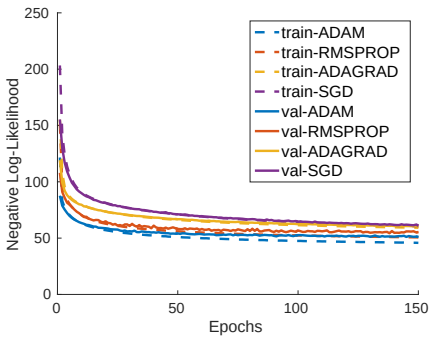


Figure 2: Illustration of implementing VAE with MLP. On the **left**, encoder maps input $\mathbf{x}$ to μ and σ. In the **middle**, we sample $\mathbf{z}$ from $q_\phi(\mathbf{z}|\mathbf{x})$. On the **right**, decoder maps $\mathbf{z}$ to y, which denotes $p_\theta(\mathbf{x}|\mathbf{z})$.

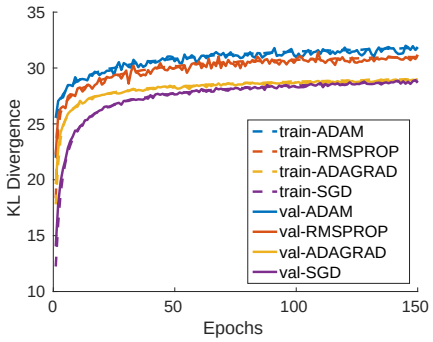
We train and test VAE on MNIST dataset. The input size is 28x28=784. We use 500 hidden units for both encoder and decoder. We experiment with code sizes of 2 and 20. We also experiment with four stochastic gradient optimization algorithms: ADAM[3], RMSPROP[5], ADAGRAD[1], and standard SGD with momentum. We compare between different optimization algorithms in terms of maximizing the lower bound along with Negative Log-Likelihood and KL Divergence in Fig. 3a, 3b, and 3c. The code is available online[1]. More implementation details can be found in the Appendix.



(a) Lower Bound



(b) Negative Log-Likelihood



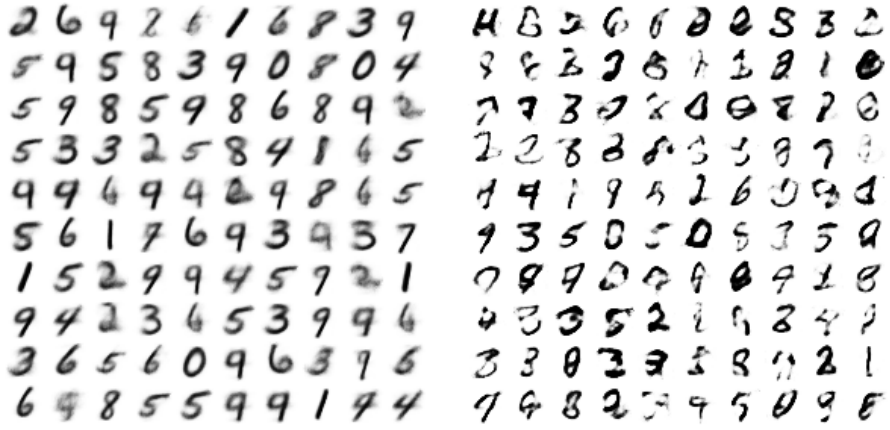
(c) KL Divergence

Figure 3: Comparison between different stochastic gradient optimization algorithms in terms of variational Lower Bound, KL Divergence and Negative Log-Likelihood. ADAM achieved highest lower bound best among four algorithms.

[1]https://github.com/peiyunh/mat-vae

4 Visualization

Random sampling Since we assume the prior over latent variables $\mathbf{z}$ to be centered isotropic multivariate Gaussian $\mathcal{N}(\mathbf{z}; 0, \mathbf{I})$, we can directly sample $\mathbf{z}$ from latent space and decode them $\mathbf{z}$ to visualize in the observed space. We visualize such random samples in Fig. 4.



(a) **2-d** latent space (b) **20-d** latent space

Figure 4: Random sampling in latent space

Manifold Similar as above, but instead of random sampling, we sample a linearly spaced grid of coordinates from a 2-d latent space, and visualize them in the observed space. We visualize a manifold of digits in Fig. 5.

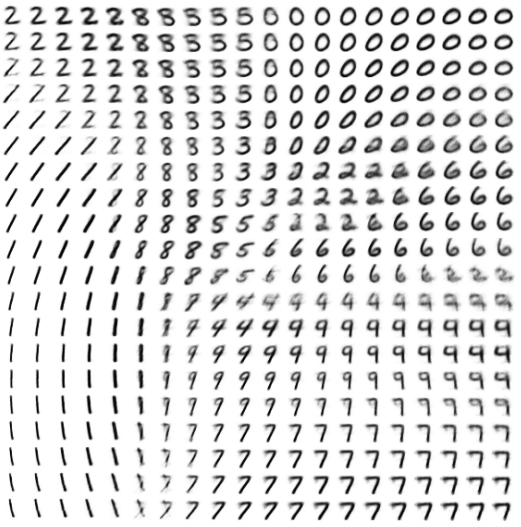
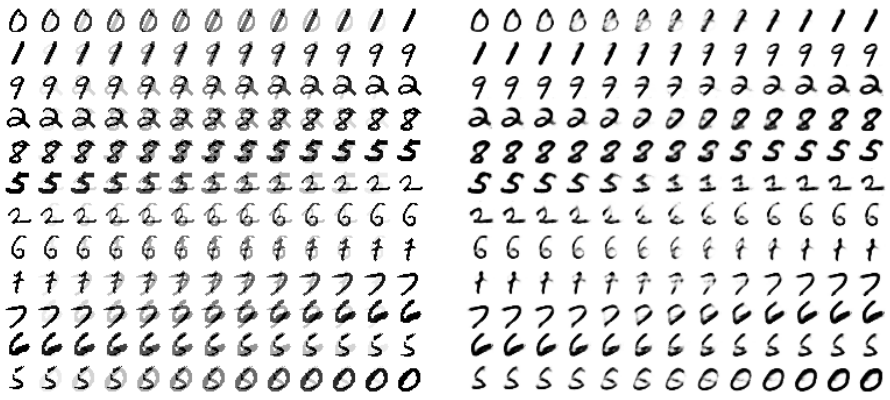


Figure 5: Digit-manifold mapped from a 2-d latent-manifold

Latent morphing Given a pair of images $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$, we can morph from one to another. The simplest way is take linear steps in image space. Another interesting way is morphing in latent space and reflect it in the observed image space. We visualize such latent morphing in Fig. 6.

5 Conclusion

We introduced how VAE works from a theoretical perspective. We re-implement the AEVB algorithm in MATLAB and re-produced the experiment results on MNIST. As extra, we explored different



(a) Morphing in **image** space. (b) Morphing in **latent** space.

Figure 6: Comparison between morphing in image space v.s. latent space. Each row represents one such morphing. At each row, we always start with where last row ends. Morphing in latent space is semantically smooth; however, no semantic transition is shown when morphing in image space.

stochastic gradient optimization algorithms and found ADAM works best with AEVB. With ADAM, we obtained a noticeable higher variational lower bound on MNIST than ADAGRAD, which was used in [4]. At the end, we show exploration in latent space with intuitive visualization.

References

[1] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[2] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[3] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[5] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop. *COURSERA: Neural networks for machine learning*, 2012.

[6] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

Appendix

External libraries We implement MLP with MatConvNet.

Numerical stability To avoid overflow or underflow, we always use an ϵ of 10^{-12}.