Variational Auto-Encoders

Jwalant Bhatt
Department of Electrical Engineering
North Carolina State University
Raleigh, United States
jdbhatt@ncsu.edu

Shahryar Rashid
Department of Electrical Engineering
North Carolina State University
Raleigh, United States
srashid3@ncsu.edu

Eshaan V. Kirpal
Department of Electrical Engineering
North Carolina State University
Raleigh, United States
evkirpal@ncsu.edu

***Abstract*—This report summarizes the results of Project 6 for ECE 542 Fall 2018. Students were tasked with implementing a variational autoencoder using a deep learning framework. The network was then used to reproduce results from recent publications. Both the latent space and manifold produced by the network was explored.**

I. Introduction

Traditional autoencoders consist of an encoder and a decoder. The encoder is responsible for compressing the input into a smaller representation while the decoder aims to reconstruct the input. The network is trained with respect to the reconstruction loss, which quantifies how well the output matches the original input. This structure allows for the autoencoder to learn a latent representation that well characterizes the data. Autoencoders have been useful in applications such as removing noise from images, but have been found to be limited as generative models. Due to gaps between the clusters in latent space, the model struggles to produce alterations of the input. Variational autoencoders (VAEs) attempt to address this by having the encoder output a vector of means (μ) and a vector of standard deviations (σ). These vectors are then used to sample values from a random distribution that are fed to the decoder. The resulting latent space is continuous which allows for better interpolation.

The goal of this study was to implement a basic VAE architecture and explore different characteristics of the network. The VAE was implemented in Python with the Keras deep learning framework. The first part focuses on the latent space for a VAE fitted to a 2-D Gaussian. The second part focuses on the resulting manifold when using higher dimensions for the latent code. In both experiments, the model was trained on handwritten digits from the MNIST dataset.

II. Equations

VAEs build on a directed latent model in which x can be explained by a set of hidden variables z. This relationship is described in Eq. 1 where $p(z)$ is assumed to be $\mathcal{N}(0, I)$. Intuitively, z explains the additional features of x not seen during training. The model aims to learn the mapping from z to the complex distribution of x.

$$p(x, z) = p(x|z)p(z) \tag{1}$$

This model may be evaluated in terms of inference. The goal is to infer optimal values of the latent variables given observed data, or to calculate the posterior $p(z|x)$. Applying Bayes theorem we can derive

$$p(z|x) = p(x|z)p(z)/p(x) \tag{2}$$

where $p(x) = \int p(x|z)p(z)dz$. Unfortunately this is intractable so we must find an approximation for the posterior distribution.

Variational inference approximates the posterior with a family of distributions $q_\lambda(z|x)$. The variational parameter λ indexes the family of distributions. For example, if q were Gaussian, it would be the mean and variance of the latent variables for each datapoint $\lambda_{xi} = (\mu_{xi}, \sigma_{xi})^2$.

The Kullback-Leibler divergence measures the information lost when using q to approximate p. By minimizing KL-divergence it is ensured that the approximation is close to the real distribution. Eq. 2 shows the KL-divergence between $q(z|x)$ and $p(z|x)$, where $q(z|x)$ is the approximated posterior of z.

$$D_{KL}[q(z|x)||p(z|x)] = E_z[log(q(z|x)) - log(p(z|x))] \tag{3}$$

When expanded, you will find that $log(p(x))$ is independent of z and can be factored out. After moving this term over, the right side of the equation can be written with the following form.

$$E_z[log(p(x|z))] - D_{KL}[q(z|x)||p(z)] \tag{4}$$

Note that in Eq. 3 E_z represents the reconstruction likelihood while D_{KL} ensures that our learned distribution q is similar to the true prior distribution p. These values are tractable can be solved analytically or with the reparameterization trick. Fig. 1 provides a high level overview of a VAE. The encoder model learns a mapping from x to z and the decoder model learns a mapping from z back to x. The structure is driven by the equations above.

III. Network Architecture

A. Dataset

The MNIST dataset was used for training the network. The dataset contains handwritten digits where each sample is a grayscale image with 28 by 28 dimensions. 60,000 images were used in the training set and 10,000 images were used in the test set.

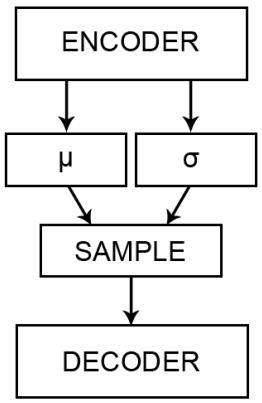

Fig. 1. Variational Autoencoder

B. Encoder Network

For the encoder, a simple convolutional network was used to map the input image x to two vectors representing the mean and standard deviation. The raw pixels of a 28 by 28 grayscale image was used as the input.

The input is fed through 4 convolutional layers, each using a ReLu activation function to introduce non-linearity. The output from these layers is then flattened into an array with a shape of (12544, 1). This array is then used to generate the parameters of the the statistical distribution (i.e. vectors of mean and standard deviation for a normal distribution).

The parameters of the statistical distribution are used to generate a latent space point z using the below mentioned formula. Note that the Lambda layer in Fig. 2 is a custom layer that performs this operation.

$$z^{(i,l)} = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)}, \ where \ \epsilon^{(l)} \sim \mathcal{N}(0, I) \qquad (5)$$

C. Decoder Network

For the decoder, the vector z generated by the Lambda layer is first reshaped to the dimensions of an image. Next, a few convolution layers are used to obtain an output image that has the same dimensions as the original input image.

The Conv2DTranspose layer shown in Fig. 3, also known as a deconvolutional layer, performs a reverse operation to the initial stack of convolutional layers. This results in transforming the shape of the image from a lower 14x14x64 dimension image to a 28x28x32 image while maintaining a compatible connectivity pattern. Another convolution layer is then added to generate a feature map the same size as the input image.

D. Training

The model was trained using 50 epochs, a batch size of 16, and the RMSprop optimizer. For the loss term, two separate losses were summed up. The reconstruction loss, which is a binary cross entropy that measures how accurately the network reconstructed the images, and a latent loss, which is a KL-divergence that measures how closely the latent variables match the true prior distribution $p(z)$. It was assumed that

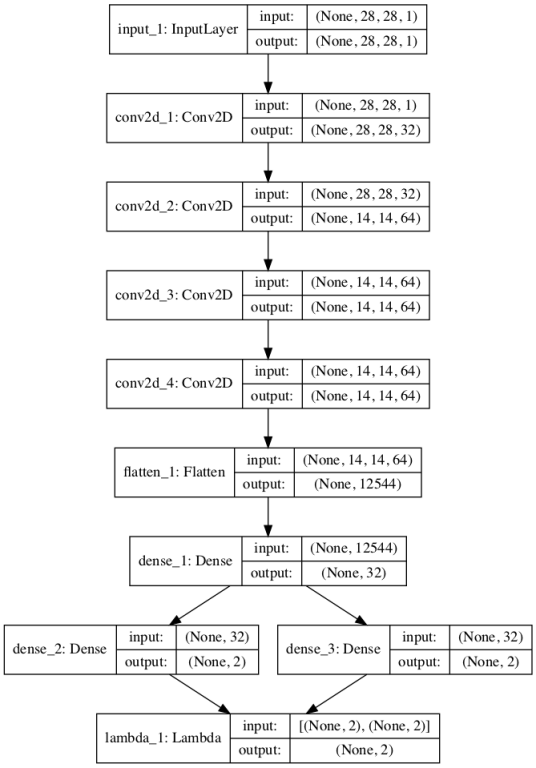

Fig. 2. Architecture for the Encoder Network

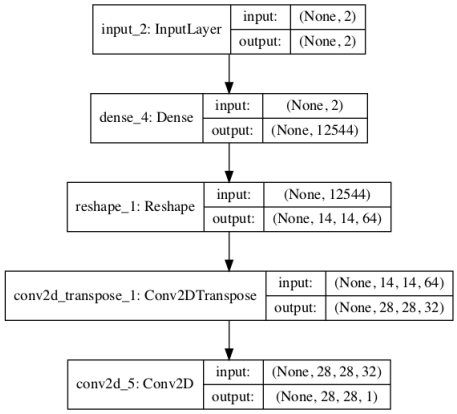

Fig. 3. Architecture for the Decoder Network

the prior distribution follows a unit Gaussian distribution for each dimension j of the latent space. The reconstruction loss penalizes the network for creating outputs different from the input.

$$L(x, \hat{x}) + \sum_j D_{KL}(q_j(z|x)||\mathcal{N}(0, I)) \qquad (6)$$

Optimizing the two together results in the generation of a latent space which maintains the similarity of nearby encodings on the local scale via clustering, yet globally, is very densely packed near the latent space origin as can be seen in Fig. 4.

IV. Results

A. Part 1

In this section, the results from Fig. 2C and Fig. 2E of the original paper [2] are reproduced. First, the VAE model is trained on the MNIST dataset with two latent dimensions. The continuous latent space can be visualized using a scatter plot. Each of the colored clusters in Fig. 4 is a different digit. Close clusters are digits that are structurally similar (i.e. digits that share information in the latent space) and can smoothly transform from one digit to another.

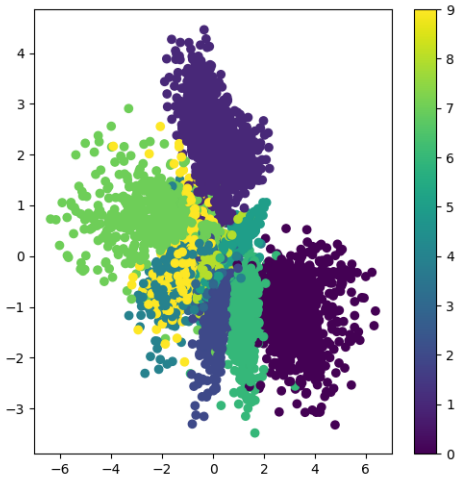

Fig. 4. Latent Space Representation for a 2-D Variational Autoencoder

Next by sampling from the continuous latent space, the decoder network is used as a generative model capable of creating new data similar to what was observed during training. Specifically, a latent point is sampled from the prior distribution $p(z)$ which was assumed to follow a unit Gaussian distribution.

Fig. 5 visualizes the data generated by the decoder network trained with 2 latent dimensions. Here the latent points are sampled at regular intervals, generating the corresponding digit for each of these points. This gives us a visualization of the latent manifold that "generates" the MNIST digits.

The grid of sampled digits shows a completely continuous distribution of the different digit classes, with one digit morphing into another as you follow a path through latent space. Specific directions in this space have a meaning (e.g. there is a direction for "four-ness", "one-ness", and so on). This smooth transformation can be quite useful when one would like to interpolate between two observations.

Fig. 6 shows the resulting loss for the training and the validation sets during training. The loss appears to have converged over 50 epochs without any signs of over fitting.

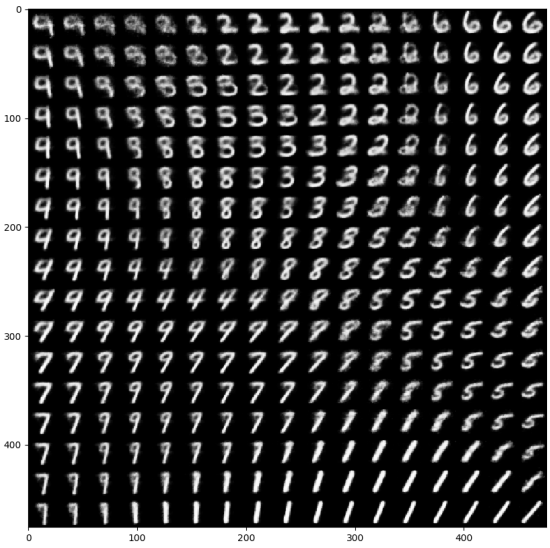

Fig. 5. Visualisations of learned data manifold for generative models with 2-dimensional latent space. Since the prior of the latent space is Gaussian, linearly spaced coordinates on the unit square were transformed through the inverse CDF of the Gaussian to produce values of the latent variables z. For each of these values z, we plotted the corresponding generative $p_\theta(x|z)$ with the learned parameters θ.

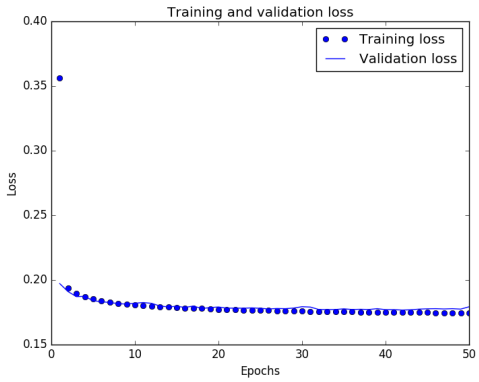

Fig. 6. Loss vs. Epochs

B. Part 2

In Fig. 7, random samples are generated from the learned 2 dimensional VAE model. Next, the model is trained using 10 and 20 latent space dimensions. Latent points are then sampled from the two learned generative models and the corresponding digits for each point is generated. The results for these models are shown in Fig. 8 and 9.

It is evident that increasing the number of dimensions in latent space gives us sharper images that closely resemble the input images. This is because the dimensions of the latent space indicates the number of features selected when the image is being reconstructed. This is why certain numbers generated

from the 2-D latent produce more error compared to the 10-D and 20-D latent space output.

It should be noted that there is not much of a difference between the 10-D and 20-D output because the MNIST image only has a certain number of major features. The 10-D output appears to cover most of the major features.

Fig. 7. Random samples from learned generative models of MNIST for 2-dimensional latent space.

Fig. 8. Random samples from learned generative models of MNIST for 10-dimensional latent space.

V. Conclusion

The learned continuous latent space representation of a variational autoencoder models allows for easy random sampling and interpolation, thus this model is a powerful data generative tool. The smooth transformation can be quite useful when one would like to interpolate between two observations. In this report the latent space for 2-dimensional variational autoencoders was visualized, where the smooth transitions between the digits was observed. At the same time, a comparative analysis of the generated random samples from three trained models with different latent dimensions was performed. For a more thorough analysis of the affect of increasing latent dimensions, it would be interesting to use a different dataset with many target labels instead of our given dataset.

Fig. 9. Random samples from learned generative models of MNIST for 20-dimensional latent space.

References

[1] Kingma, Diederik P., and Max Welling. "Auto-Encoding Variational Bayes." arXiv preprint arXiv:1312.6114 (2013).

[2] Makhzani, Alireza, et al. "Adversarial Autoencoders." arXiv preprint arXiv:1511.05644 (2015).