Variational Autoencoder without Kullback–Leibler divergence (BSvarautonet)

Sing Kuang Tan
Email: singkuangtan@gmail.com

September 9, 2024

Abstract

In this paper, I am going to propose a new Boolean Structured Variational Autoencoder Deep Learning Network (BSvarautonet) built on top of BSautonet, based on the concept of monotone multi-layer Boolean algebra. Kullback–Leibler (KL) divergence used in traditional Variation Autoencoder has convergence problem and numerical instabilities. Due to the Boolean Structured design of BSautonet, the bottleneck latent space embeddings is naturally distributed in multi-variables Gaussian distribution. By applying a whitening normalization on the latent space, it will transform the latent space to unit Gaussian distribution. Through analysis of the datapoints in latent space and generated MNIST digit images, it has shown that it has all the properties of variational autoencoder. The BS autoencoder is a masked noise denoising model, therefore it can acts like a diffusion model to incrementally generate a digit image from a noisy one through repeated applications of the autoencoder model.

1 Introduction

My BSvarautonet network is the same as BSautonet except that the latent space embeddings are normalized into unit Gaussian distribution.

Variation autoencoer has a lot of advantages over conventional autoencoder. As the generated latent space datapoints are compact with no free space between datapoints, it will ensure the generated image conforms to the actual class of the digits. It can also be used to determine the probability of an occurence of an input image in the dataset. It can be used to generate a series of images transforming a digit image to another digit image.

Please note that class labels are not used when training BSautonet, the latent codes are trained specifically due to the similarities of the appearances of the digit images. No regularization, no weight decay to demostrate the simplicity and prowess of my model. Since no KL divergence is used, it can be trained very fast in 150 epochs.

KL divergence has convergence problem (sometimes unable to learn the dataset) and numerical instabilities. KL divergence can give you sleepless nights, spending hours after hours tuning hyperparameters and retraining the variational autocoder model again and again. Furthermore KL divergence loss function in variational autoencoder has high computational cost. Using my model, default hyperparameters work well and can be train as fast as a normal autoencoder.

2 Hypothesis

Because of the non-negative weights design of our BSvarautonet, the distribution of the latent space embedding is very close to Gaussian distribution

Even if our BSvarautonet is not more accurate than an ordinary network, it is still worth studying as it has well defined and interesting properties an ordinary network does not have. Ordinary network is too complex to analyze as it tries to emulate a high dimensions floating point function, whereas our network tries to emulate a Boolean algebra network. So from our network we can easily derive interesting properties from it.

Our network neuron is a high dimension input and single output monotonic function, which means that the value of the function increase or decrease in one direction. Therefore the datapoints distribution (generated from test set or training set) in the latent space is very "continuous", where there no clear gaps between datapoints. This property is important as it creates a span of "continuous" datapoints like an ordinary variational autoencoder, without the need for KL divergence regularization or any other regularization. Ordinary non-variational autoencoder will have clear gaps inbetween datapoints.

As our network is a Boolean Structured network, the latent space span is a high dimension cubic continuous 0-1 logic space. This space however is skewed (skewed Gaussian distribution with non-identity coveriance matrix). So a whitening operation has to perform on the latent embeddings to make it a unit Gaussian distribution. The output range of a neuron may be in a range bigger than 1, e.g. from 0 to 7, where 0 is logic 0 and 7 is logic 1.

Since the weights are non-negative, during training the output is propagated to the input like a particle filter, each weights will move to the mean propagated values from the input-output pairs. So the output of each neuron will be normal distributed but has different variances at different direction (non-unit covariance matrix). To get a unit normal Gaussian distribution, we simply whiten the distribution of the datapoints (generated from test set or training set) to unit convariance (identity matrix). Look into more closely, since each neuron in the BSvarautonet acts like a Boolean operation, there are 2 operational modes in the output distribution of a neuron. But overall it will looks like a normal distribution.

The weights of by BS series network (BSnet, BSautonet, BSconvnet, BSautoconvnet, BSvarautonet) are always positive. This has an advantage in learning and human interpretation of the weights values. The weights represent how much information is passed from the lower layer outputs (inputs of a neuron) to the next latyer outputs (output of a neuron). If we allow negative weights, then it seems weird as we are passing negative information to the next layer. And for training, weights only move in one direction in the positive region and we do not have to worry whether the weights should be positive or negative.

My BS series networks training process is like solving a Travelling salesman problem using Linear Programming. In TSP, the linear program will cast vote on each edge and the edge with the highest vote will be removed. Then the LP is updated to cast another set of votes. The next highest vote edge will be removed. LP is repeated until a set of edges representing the shortest ring route across the cities. It is the same for my BS networks. The gradient descent will cast votes and one of the weights will be set to zero. It will continue cast votes and set to zero another weight until full learning of the input-output pairs of the dataset.

3 My Model

Our BSvarautoent will be the same as BSautonet[1] except that it has whitening operation on the bottleneck latent space embedding so that the embedding is unit Gaussian distributed (see figure 1). Whitening operation is using an Singular Value Decomposition algorithm to set all the eigenvalues to 1.

The Singular Value Decomposition (SVD) of a matrix A is given by:

$$A = U\Sigma V^T$$

where:

- A is an $m \times n$ matrix.
- U is an $m \times m$ orthogonal matrix (whose columns are the left singular vectors of A).
- Σ is an $m \times n$ diagonal matrix (with non-negative real numbers on the diagonal called singular values of A).
- V is an $n \times n$ orthogonal matrix (whose columns are the right singular vectors of A).
- V^T is the transpose of V.

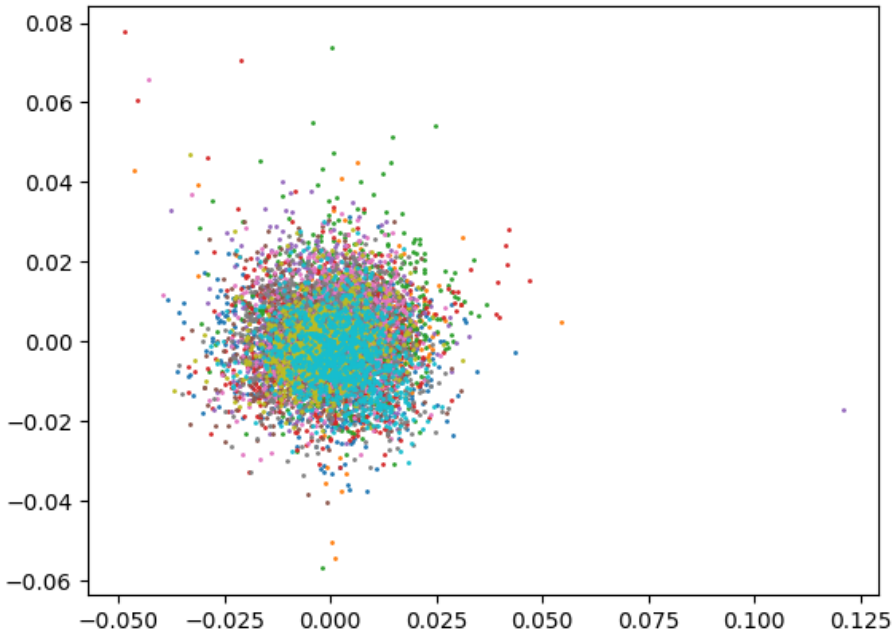

Figure 1: Principle component analysis plot of the latent space embeddings of my BSvarautonet after whitening operation

In comparsion, the latent space plot of a regular autoencoder is shown in figure 2. There are a lot of free space between embeddings. This will lead to garbage image to be generated if you sample any embeddings between in the free space. The probability distribution of the latent space embeddings does not follow any standard mathematical probability distribution, which makes it difficult to generate a sample of the distribution algorithmically. However, for my BSvarautonet, the distribution of the embeddings is Gaussian. So you can sample embeddings distribution simply using the mean and covariance matrix of the Gaussian distribution with some Gaussian random variable generation algorithm.

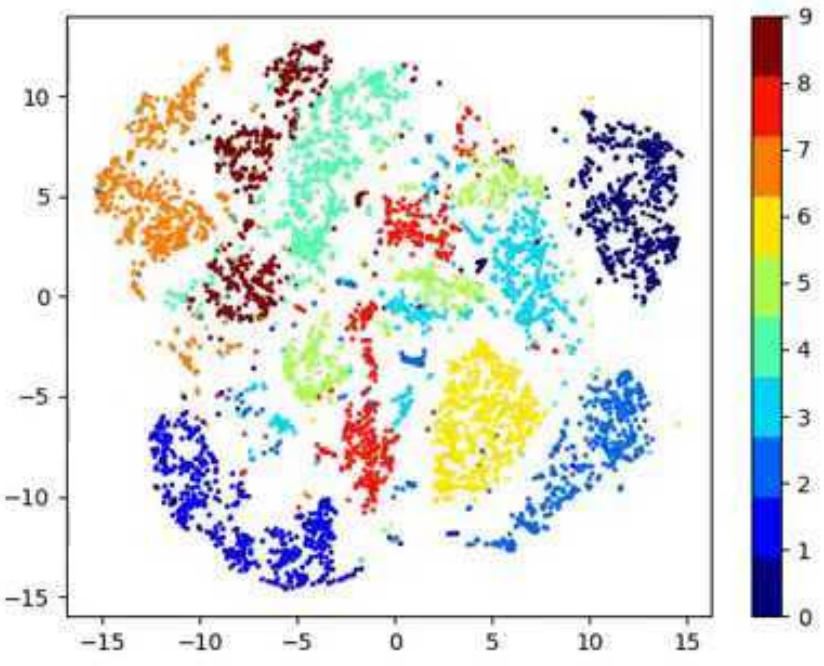

Figure 2: Principle component analysis plot of the latent space embedding of a regular autoencoder. Source: https://mohitjain.me/2018/10/26/variational-autoencoder/

This will lead to garbage image to be generated if you sample any embeddings between in the free space. The probability distribution of the latent space embeddings does not follow any standard mathematical probability distribution, which makes it difficult to generate a sample of the distribution algorithmically. However, for my BSvarautonet, the distribution of the embeddings is Gaussian. So you can sample embeddings distribution simply using the mean and covariance matrix of the Gaussian distribution with some Gaussian random variable generation algorithm.

The T-SNE plot of my BSvarautonet latent space embeddings (Figure 3) shows that it mathematically groups up images of the same digits in the same clusters. This means that my BSvarautonet has achieved the removal of mutual information between digit classes and generate independent cluster component for each digit class.

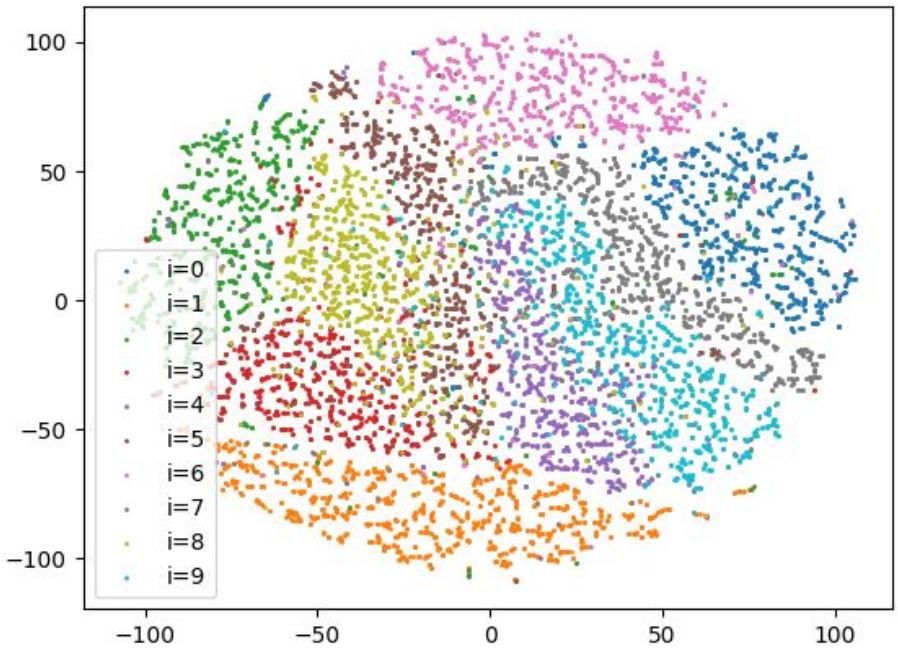

Figure 3: T-SNE plot of the latent space embedding of my BSvarautonet

So transformation in the latent space is equivalent to transformation in the output image. In specific, if you linear transform a digit 6 image embedding to digit 0 embedding, the output image will linear interpolate between the two digits (see figure 4)

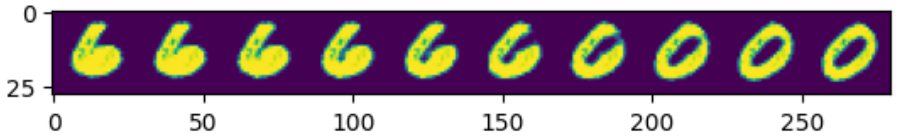

Figure 4: Linearly interpolate between digit 6 and 0 using linear interpolation in the bottleneck latent space embedding

4 Experiment Results

The figure 5 and 6 shows how the output digit images are distributed. If the embedding is close to the origin, the digit images will resemble to digit images found in the dataset. If the embedding is far from the origin, the digit images will be different from what is found in the dataset, which in some cases, may create a weird digit image that does not look like a number.

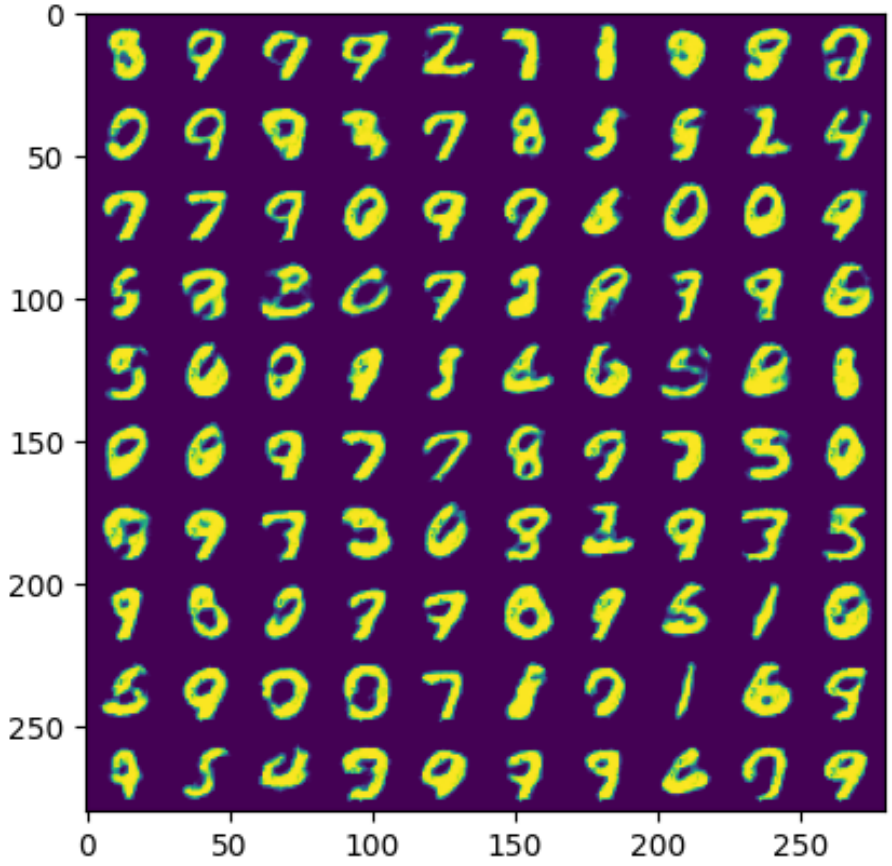

Figure 5: Output digit image distributions generated by sampling within the unit Gaussian distribution of bottleneck latent space embedding

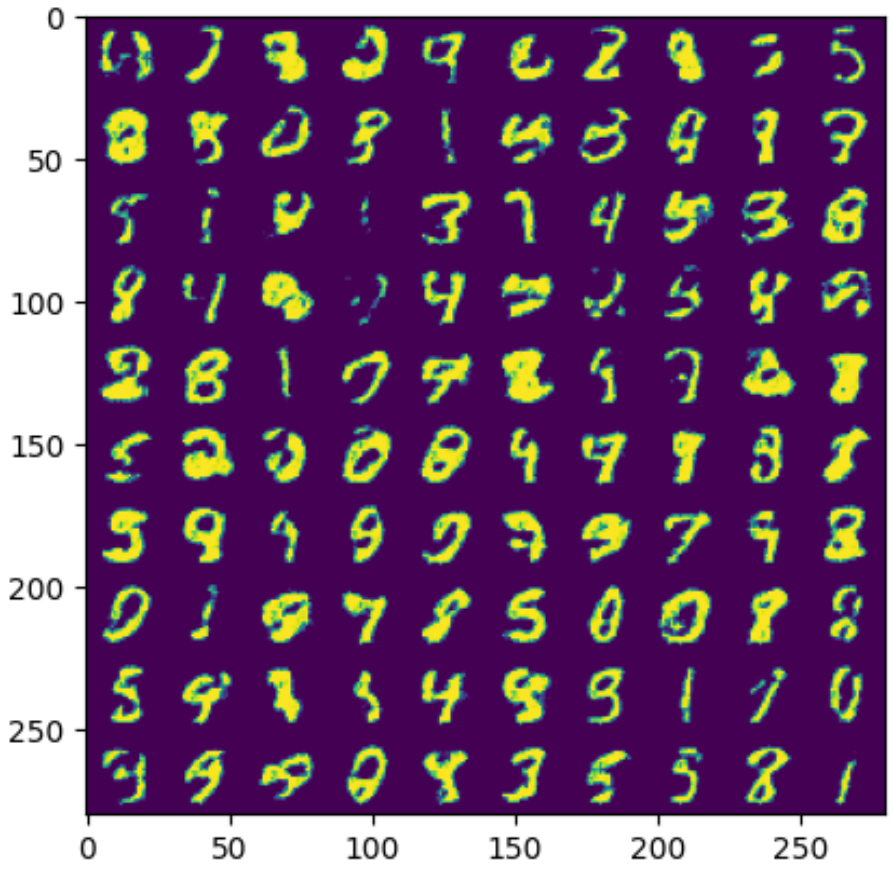

Figure 6: Output digit image distributions generated by sampling out of the unit Gaussian distribution of bottleneck latent space embedding

My BSvarautonet can act like a diffusion network, to generate random digit images or denoise an existing digit image. Figure 7 shows how a random digit image is generated. Start from random noise image, denoise it a few times using BSvarautonet, then add noise to the image. Repeat the denoise and add noise step multiple times. The final output image will be a new digit image generated using the digit image probability distribution of the dataset.

The diffusion behavior of my BSvarautonet is like the mode seeking behavior of mean shift algorithm. If a random generated latent embedding lies between and out of the distributions of all digits, then the generated output digit from my BSvarautonet will be abnormal, looks like something between 2 digits, the handwriting may generate broken line segments. However after diffusion process, the output digit image will be transformed to a digit in one of the digit class distribution.

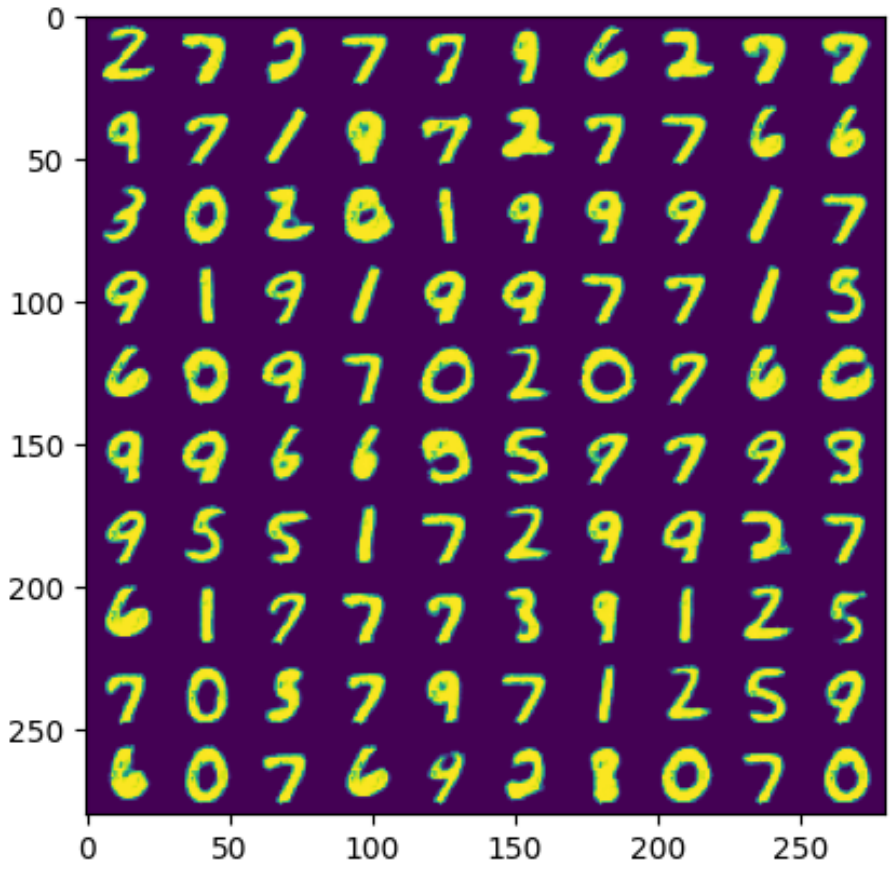

Figure 7: Use BSvarautonet like a diffusion network to generate random digit images

Access my GitHub codes thru this link:
https://github.com/singkuangtan/BSvarautonet

5 Conclusion

I have developed a Boolean Structured Variational Autoencoder Deep Learning Network for general noisy digit image reconstruction problem (MNIST dataset) in machine learning, but it is able behave like a variational autoencoder. Because the variational autoencoder is able to generate continuous, smooth and single blob unit Gaussian distribution, it made random generation and interpolation of digit images close to what is found in the dataset, without generating gabarge images.

It has application to transform one digit image to another, to use like a

diffusion network to generate random digit images or denoise part of an existing digit image.

My ultimate goal is to develop an ultimate deep learning network in its simplest form and canoical form, by making step by step improvements, with one of the step to make each layer output distribution to become unit Gaussian distribution.

References

[1] Sing Kuang Tan. Design autoencoder using bsnet (bsautonet). 2022. Accessed: 2024-03-03.