Variational Autoencoders

Artificial Neural Network Architecture

Gupta Ananya Vikas (A0226576W) - *Data Science and Analytics*
Bharath Shankar (A0219924X) - *Data Science and Analytics*

Abstract

Variational autoencoders provides a base to describe an observation in latent space. In this project, we provide an introduction to variational autoencoders and the logic behind their framework using the mathematical concepts involved in defining them. The basic architecture of an autoencoder and the application of Principle Component Analysis for proving that maximising variance, minimises error, followed by the development into variational autoencoders and their latent space visualisation via a coding component.

Motivation and Context

We can consider the following problem: Suppose there exists a process f that operates on a vector parameter $\mathbf{z} \in \mathbb{R}^d$, generating an output $\mathbf{x} \in \mathbb{R}^n$. (We assume that $d < n$). We refer to $\mathbf{z}$ as being a latent vector.

We now observe $\mathbf{x}$, and we want to figure out a mapping from the n dimensional space to the d dimensional space, to get an estimate of the underlying parameters $\hat{\mathbf{z}}$.

Why do we want to find z?

The latent space of a distribution can be thought of as, a space of vectors $\mathbf{z}$ where if $\mathbf{x}_1$ and $\mathbf{x}_2$ correspond to similar observations, the corresponding vectors $\mathbf{z}_1$ and $\mathbf{z}_2$ are close to each other in that space. Ideally, we would like each dimension of the latent space to correspond to an easily-understood parameter to a human, but that may not be a possibility when dealing with data such as text and images.

Nonetheless, reducing the dimensions of $\mathbf{x}$ has its utility, particularly in the context of downstream tasks, such as classification and regression. High dimensional spaces cause issues for models that depend on distance metrics for their outputs, such as K-Nearest Neighbours, Linear and Logistic Regression. This problem is known as the curse of dimensionality.

Reducing the number of dimensions fed to any downstream models lets us use the fact that the dimensions in the latent space are meaningful, as well as avoiding problems with using distance in high-dimensional spaces, improving the performance of downstream models.

Generative modelling

A subdomain of machine learning is generative modeling, which aims to solve the more general problem of learning the distribution over variables and simulates behaviour of data generation in the real world.

We looked at different real-life examples of such modelling, like

- Meteorologists model the weather using partial differential equations showcase the physics of the weather
- Astronomer models the formation of galaxies using physical laws for the celestial and stellar bodies.

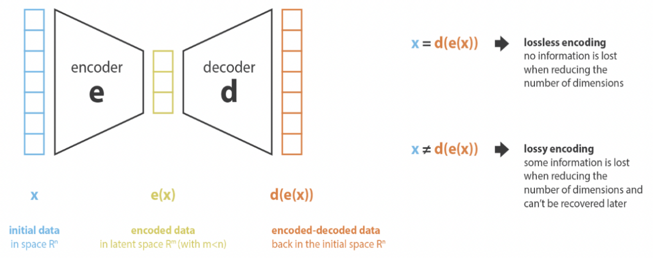
Hence, we were determined to dig deeper into this concept for the project to understand the concepts taught better through this example widely used by various industries.

Index

Principal Component Analysis

Dimensionality Reduction is the process of transforming high dimensional space into low-dimensional space, retaining the important original patterns. It works by reducing the number of features that describe the data.

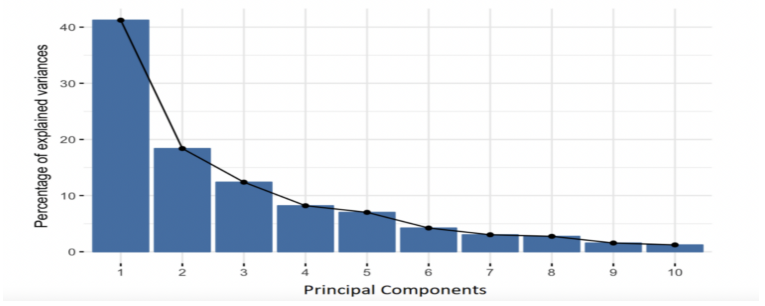
The main purpose of a dimensionality reduction method is to find the best encoder/decoder from the possible encoders and decoders, that retains maximum information when encoding, hence resulting in minimum reconstruction error while decoding.


encoder
e
decoder
d
x = d(e(x))
lossless encoding
no information is lost when reducing the number of dimensions
x ≠ d(e(x))
lossy encoding
some information is lost when reducing the number of dimensions and can't be recovered later
x
e(x)
d(e(x))
initial data
in space Rⁿ
encoded data
in latent space Rᵐ (with m<n)
encoded-decoded data
back in the initial space Rⁿ

A main linear method of dimensionality reduction is Principal Component Analysis (PCA). To give a quick explanation of PCA, it involves trying to find components that cover most of the variance in the input data. In other words, PCA tries to find the best linear subspace of the initial space such that the error of approximating the data by their projections, is minimised.

How PCA works to reduce dimensionality:

1. Firstly, it standardises the initial variables $\hat{X} = X - \bar{X}$
2. Then, the covariance matrix is calculated using the covariance between all possible combination pairs of input variables which is S$=\frac{1}{N}\hat{X}^T\hat{X}$
3. From the covariance matrix, the eigen vectors and eigen values are calculated which attribute into the calculation of the principal components of the data. The principal components can be understood as linear combinations of the initial variables so the new variables are uncorrelated.
4. Following this framework, maximum information from initial variables is tried to be fitted in the first component, then the maximum of the remaining information in the second component and so on.
5. As such, the first principle component will represent the largest possible variance from the dataset. The second principal component similarly calculated will represent the next highest variance when it is perpendicular to or not correlated to the first principal component. This is repeated till p principal components have been calculated for p variables.



As seen above in the graph, the 10-dimensional data produces 10 principle components and the maximum percentage of information that is explained variances is fitted in the first component. After which, the information in each component keeps reducing.

6. When information is organised like seen above in principal components, it allows dimensionality reduction without the loss of important information, while discarding the components with low information.

7. Just like how the eigenvectors helped us find the principal components, it also helps choose the components with useful information (high eigenvalue) from the ones with less important information (low eigenvalue). The vectors that represent important information forms a matrix of vectors that we call 'Feature vector'.

   The reason for doing so is that only p components out of n selected so mapping this will give us p dimensions in final output.

8. Finally, the data is reoriented to the axes represented by the principal components using the feature vector as seen in the equation below:

   $$OutputDataset = (Featurevector)^T * (Standardisedinputdataset)^T$$

The objective of the process above is maximising the variance in order to minimise the reconstruction loss.

Maximising Variance:
The goal is to maximise the variance of the projected data so we look for the orthogonal projection of the data into a lower dimensional linear space.
Lets see for dimensions=1: $\|w_1\| = 1$

$$\sigma^2(\hat{x}) = \frac{1}{N}\sum_{n=1}^{N}(\hat{x_n} - \hat{\bar{x}})^2 = \frac{1}{N}\sum_{n=1}^{N}(w_1^T x_n - w_1^T \bar{x})^2 = w_1^T S w_1$$

where S is the covariance matrix

Taking Lagrange multiplier to ensure the unit norm of w for the maximum variance optimisation problem

$$J(w_1) = w_1^T S w_1 + \lambda_1\ (1 - w_1^T w_1)$$

The derivative set to zero shows that $Sw_1 = \lambda_1 w_1$

Hence, on substitution, the maximum variance in lower dimensional space is equal to the eigen value $w_1^T S w_1 = \lambda_1$

Overall, for lower dimensional space with d dimensions, the principal components are the eigenvectors corresponding to the largest eigenvalues as seen above for $d = 1$

<u>Minimising Reconstruction Error:</u>
The goal is to minimise the mean squared error between the data points and their linear projection.

$J(x, \tilde{x}) = \frac{1}{N} \sum_{n=1}^{N} (\|x_n - \tilde{x_n}\|)^2$

where the reconstruction from the the lower dimensional latent variable is $\tilde{x}$

Based on properties of orthonormal basis $w_i^T w_j$, the completeness of the basis so any linear combination of basis vectors can be used.

$\tilde{x_n} = \sum_{i=1}^{d} (x_n^T w_i) w_i + \sum_{i=d+1}^{D} b_i w_i$

Computing $x_n - \tilde{x_n}$ $=\sum_{i=d+1}^{D} (x_n^T w_i) w_i - b_i w_i$

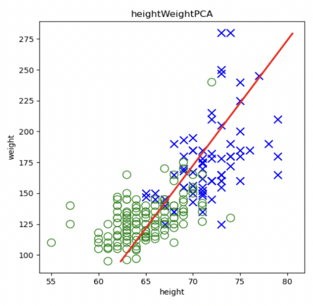
On substitution in J and its derivative set to zero, $J(w) = \sum_{i=d+1}^{D} w_i^T S w_i$

To overcome the trivial solution at w=0, we normalise to $(\|w\|)^2 = 1$

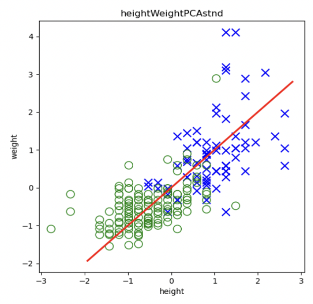
The Minimum Error for reconstruction J works by choosing the eigenvectors w_i from the covariance matrix when $Sw_i = \lambda_i w_i$

so minimum reconstruction error $J = \sum_{i=d+1}^{D} \lambda_i$

This can be reflected in the maximising problem above for variance too so proves that minimising error maximises variance.



(a)



(b)

Autoencoders

A brief aside into Neural Nets

When discussing Kernels, we discussed the idea of feature extraction. In other words, we use the features $\phi(\mathbf{x})$ rather than the input features $\mathbf{x}$. Then, a downstream model can then be of the form

$$f(\mathbf{x};\theta) = \mathbf{W}\phi(\mathbf{x}) + \mathbf{b}$$

The above model is linear in in the features $\theta = (\mathbf{W}, \mathbf{b})$. However, the transformation $\mathbf{x} \rightarrow \phi(\mathbf{x})$ itself may be nonlinear. As observed in the lecture, identifying the optimal transformation may be difficult in many cases. However, we can try to parameterise the transformation, and then try to learn the new features from the data. Let the new learnable parameter be θ_2. Then, we have:

$$f(\mathbf{x};\theta) = \mathbf{W}\phi(\mathbf{x};\theta_2) + \mathbf{b}$$

Effectively, we have composed 2 functions together. We can then extend this to compose even more functions together. The strength of neural nets lies in this composition of functions.

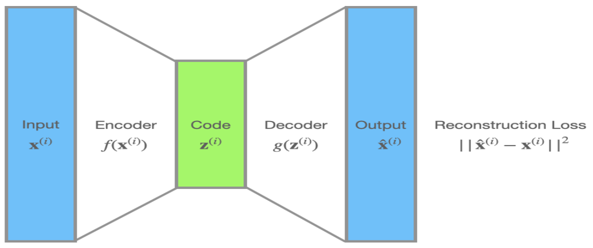
The Universal Approximation Theorem tells us that irrespective of what the function is, or the number of inputs and outputs, it will approximate and give a reasonable result.
While encoding data, it is an important property to achieve universality for optimal results however, given certain constraints it can be difficult to achieve universality.
Considering weak learners act as building blocks and if we stack many such building blocks and add all of them up, it helps you approximate any function 'g' can be written as a composite combination of the individual functions.
For any neural network architecture, finding any mathematical function $y = f(x)$ that can map attributes(x) to output(y), this results in allowing us to approximate any complex true relationship between input and output.

Architecture of an Autoencoder

An Autoencoder consists of 2 sub-networks - an Encoder and Decoder network. The Encoder approximates the transformation $f : \mathbb{R}^n \rightarrow \mathbb{R}^d$. The d-dimensional co-domain of the function f is the latent space of the data.

The decoder is, at its core, a reversed version of the encoder. It approximates the transformation $g : \mathbb{R}^d \rightarrow \mathbb{R}^n$. The decoder maps from the latent space to the reconstruction of our original input x.

Usually, $d \ll n$. This leads to an under-complete representation of the data, forcing generalisation. It is possible to take $d \gg n$, but that case requires some form of regularisation to prevent overfitting.



Training an Autoencoder

Autoencoders are a form of unsupervised learning. This means that the labels of the data are not utilized for the training of the network. Like most other neural network architectures, autoencoders try to optimize an objective function. The usual objective function for autoencoder is:

$$\underset{\theta}{min}\|\hat{\mathbf{x}} - \mathbf{x}\|^2$$

Where θ represents all the parameters of the neural network, $\mathbf{x}$ is the input into the neural net, and represents the reconstructed $\mathbf{x}$. This loss function, however, looks familiar. It is, in fact, the same objective we saw earlier in PCA - minimizing the reconstruction loss!

Linear Autoencoders as PCA

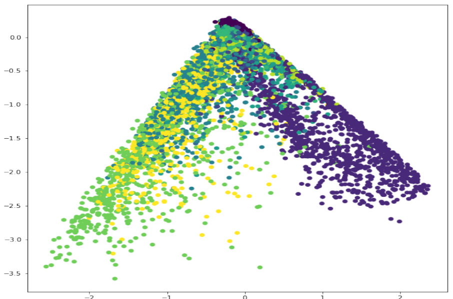
Consider the case of a linear autoencoder, i.e. a one-layer encoder and a one-layer decoder, with no activation function (it needn't be one-layer, but no activation function is what makes it linear. We take only 1 layer for the sake of simplicity). Additionally, we consider the case with no bias. We then can represent out autoencoder by $\hat{\mathbf{x}} = g(f(\mathbf{x}))$, where $f(\mathbf{x}) = \mathbf{W}\mathbf{x}$ and $g(\mathbf{z}) = \mathbf{V}\mathbf{z}$, where $\mathbf{W} \in \mathbb{R}^{d \times n}$, and $\mathbf{V} \in \mathbb{R}^{n \times d}$. The autoencoder objective can then be represented as:

$$\underset{\mathbf{W},\mathbf{V}}{min}\|\mathbf{V}\mathbf{W}\mathbf{x} - \mathbf{x}\|^2$$

Which is then equivalent to the aforementioned objective for PCA. We can then see that adding nonlinearities via adding activation functions can be viewed as a non-linear extension to PCA.

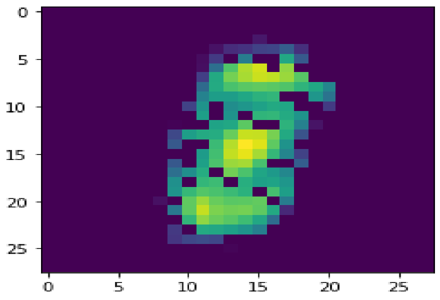
Latent Space Visualization - Autoencoder

Since similar points should lie closer to each other in the latent space, and dissimilar points lie far away from each other, we should see natural clustering emerge. Below, we show the latent space obtained by passing the MNIST test dataset to an autoencoder trained on the MNIST train dataset.



Oddly, we do not see an efficient utilisation of the latent space! We see that the points are all clustered along 2 axes, with most of the "weight" being on one axis or the other. This can limit the capacity of the model to interpolate, leading to the decoder having a very poor capacity for generating valid samples from the latent space.

We can see this effect in the generated interpolations from the latent space of the autoencoder, most of which are of poor quality.



To improve the quality of our latent space, for more efficient utilization and better quality generation outcomes, we need to make a modification to the autoencoder. This comes in the form of Variational Autoencoders .

Variational Autoencoders

In conventional bottleneck autoencoders, we were stymied in generation by the fact that randomly sampling from a point in the latent space completely changed the type of reconstruction we received. In other words, we could not guarantee that for a point $\mathbf{z} \in \mathbb{R}^d$, and decoder $g : \mathbb{R}^d \to \mathbb{R}^n$, where $g(\mathbf{z}) = \hat{\mathbf{x}}$, a point $\mathbf{z'} = \mathbf{z} + \epsilon$ (where $\epsilon \in \mathbb{R}^d$ is a small perturbation) does not generate a value $\hat{\mathbf{x'}} = g(\mathbf{z'})$ close to $\hat{\mathbf{x}}$ (A property called continuity). To put it simply, a random point close to an encoded point need not encode useful information.

This comes from the design of the bottleneck autoencoder. The normal autoencoder transforms the data input to another vector in the latent space.

This does not encourage 2 properties we want from our latent space: continuity, as well as completeness (Every point in the latent space corresponds to a meaningful reconstruction).

How do we encourage continuity and completeness?

Effectively, we need some way to teach our network that close points in the latent space should look similar once decoded, to ensure continuity. Also, we want to ensure that the model learns to spread out its encoded outputs throughout the latent space, to ensure completeness.

We do this by no longer mapping each input to a vector. We now attempt to map the inputs to a probability distribution in the latent space. In other words, we now consider each attribute of our latent space as being a probability distribution that we sample from for generation.

Statistical Motivation

Let us consider the following process: Suppose there exists a process that operates on a random variable Z, that generates an outcome x. We can only observe the realizations x of the process, and we wish to infer the properties of Z. To this end, we would want to compute $p(z|x)$.

By Bayes' Theorem, we have:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

However, the main problem arises in the term in the denominator, $p(x)$. By the law of total probability, we can see that:

$$p(x) = \int p(x|z)p(z)dz$$

However, the above integral is intractable (i.e., there is no closed-form solution). This leads to us needing to find another way to calculate the probability $p(z|x)$.

Let us try to approximate $p(z|x)$ by another distribution $q(z|x)$. We consider a form for $q(z|x)$ such that the integral becomes tractable. However, how do we figure out the best values for the parameters of $q(z|x)$?

The above is a common tactic in variational inference, which utilizes optimization to figure out the best parameters for $q(z|x)$. While euclidean distance is used in the case of vectors, how do we quantify the difference between 2 distributions?

For this purpose, we use the Kullback-Liebler Divergence (KL Divergence). The KL divergence between 2 distributions can be seen as a measure of how different the 2 distributions are. We can then see that our optimization objective can be represented as:

$$minKL(q(z|x)||p(z|x))$$

Deriving the VAE objective

We can express the KL divergence as:

$$KL(q(z|x)||p(z|x)) = -\sum q(z|x)log(\frac{p(z|x)}{q(z|x)})$$

Recall that:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

Therefore:

$$-\sum q(z|x)log(\frac{p(z|x)}{q(z|x)}) = -\sum q(z|x)log(\frac{p(x,z)}{p(x)q(z|x)})$$

$$= -\sum q(z|x)(log(\frac{p(x,z)}{q(z|x)}) + log(\frac{1}{p(x)}))$$

$$= -\sum q(z|x)(log(\frac{p(x,z)}{q(z|x)}) - log(p(x)))$$

$$= -\sum q(z|x)log(\frac{p(x,z)}{q(z|x)}) + \sum q(z|x)log(p(x))$$

Observe that we try to sum over all values of z, since x is fixed. Thus, we can see that the second summation term is a constant! Thus, minimizing the KL divergence can be seen as minimizing the following term:

$$= -\sum q(z|x)log(\frac{p(x,z)}{q(z|x)})$$

The negative of this term is referred to as the variational lower bound.

$$= -\sum q(z|x)log(\frac{p(z|x)p(z)}{q(z|x)})$$

$$= -\sum q(z|x)(log(p(x|z)) + log(\frac{p(z)}{q(z|x)}))$$

$$= -\sum q(z|x)log(p(x|z)) - \sum q(z|x)log(\frac{p(z)}{q(z|x)})$$

We now see that our second term is the KL divergence between $q(z|x)$ and $p(z)$! The second term can be expressed as the negative of expectation of $log(p(x|z))$ with respect to $q(z|x)$.

If we assume a Gaussian $q(z|x)$, we can then show that the first term reduces down to: $\|\hat{x} - x\|^2$, which is reconstruction error! In other words, we can re-express our objective as minimizing the sum of the reconstruction error and KL divergence!

The KL-divergence in this case acts as a regularization term for the variational autoencoder.

Trade-off between KL divergence and Reconstruction error

VAEs encode their inputs as a distribution rather than as vectors and the distributions of the VAE are regularized. With this regularisation term, the model does not encode data far apart in the latent space. This increases the amount of overlap within the latent space. However, this regularisation term results in a higher reconstruction error on the training data. So, the two have contrasting effects: The reconstruction loss is minimised to improve the quality of the reconstruction,but the shape of the latent space is neglected. The KL-divergence normalizes the latent space, but results in some additional "overlapping" between latent variables. Hence, the trade-off between the reconstruction error and the KL divergence needs to be adjusted.

Minimizing the KL distance between $q(z|x)$ and the prior distribution $p(z)$.

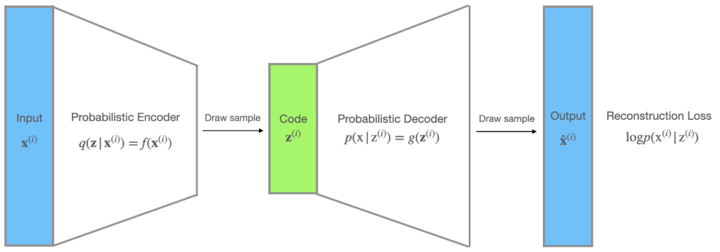
1. Log-likelihood: $E_{z\ Q(z|x)} log(p(x|z))$

2. KL Divergence: $KL((q|z)||p(z))$

$$log(p(x)) - KL(q(z|x)||p(z|x)) = E_{z\ Q(z|x)} log(p(x|z)) - KL((q|z)||p(z))$$

We can observe the trade-off between the two terms — maximisation of the expected log-likelihood and minimisation of the KL divergence.

Components of VAE



- An image is fed as input x
- The probabilistic encoder compresses the input 'x' based on the distribution using the mean and standard deviation as a sampled latent vector
- The probabilistic decoder then reconstructs and expands the compressed version of the input based on the probability

- The output x' is the reconstructed image of the input
- Further comparisons are made between the input and output image and the loss function is represented by the reconstruction loss and the regularizer term.

As a summary, a variational autoencoder works similar to an autoencoder but with refined features and better representation and reconstruction of the input.

Real-life Applications of VAEs

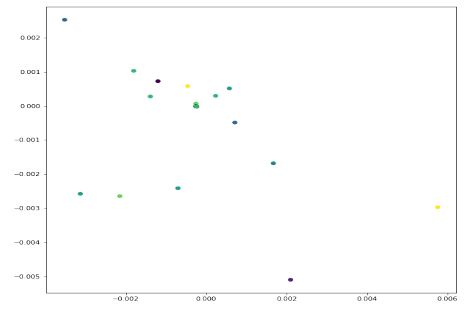
1. Image Re-synthesis:
   On optimising a VAE, a generative model can be designed over images, which can synthesize images and change features in them like colors, shape, etc can be modified and re-synthesized.
2. Compound Generation:
   VAEs ca be used in different forms of drug discovery and the most common one is to generate new chemical/molecule structures using the patterns and

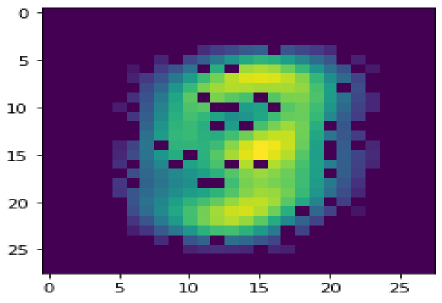
Experiments

We attempted to visualize the latent space of the VAE, while weighting the KL Diveregence. We show below the results of training a network at 3 values of weight for the KL divergence term.

KL Weight = 8e-4

In this case we see that the weight on the KL term is far too high. We can observe that the latent space is over-regularised, with only a few allowed points spread across the space.

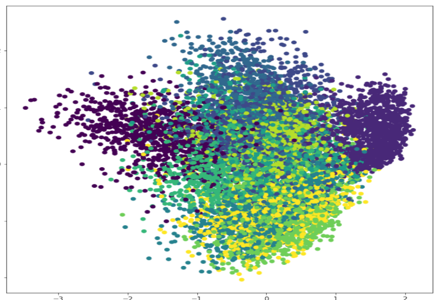
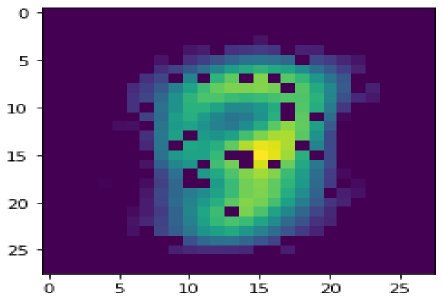




The quality of the generation is also not too good:

KL Weight = 5e-5

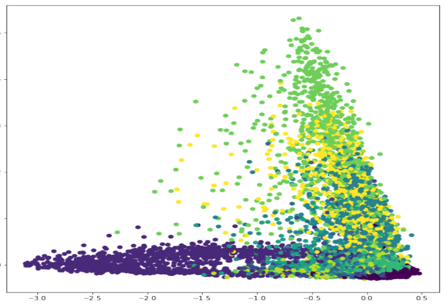
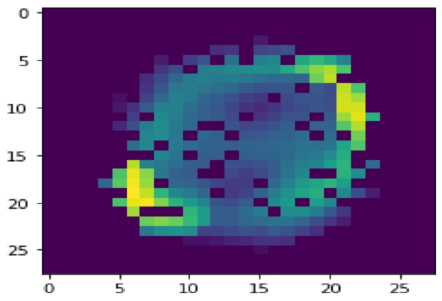
In this case we see that the weight on the KL term seems appropriate. The points are well spread out, with efficient utilisation of the latent space.

The quality of the generation is also better:

KL Weight = 1e-8

In this case we see that the weight on the KL term is far too low. The VAE now resembles an autoencoder.

The quality of the generation is not great, either:

Appendix

Reflection

Overall, this project has drawn us towards to the practical and theoretical aspect of Variational Autoencoders. It further helped us question every step of our approach in terms of the proof and reasoning behind every assumption and claim made. Within our group, each of us was involved in reading and researching deeply about the topic of auto-encoders and the mathematical motivation behind variational autoencoders. We spent time discussing the applications of

VAEs and how each component is developed with a mathematical purpose for generative modelling, which drew us towards deep self-learning over the process.

Contributions

Bharath Shankar: Code + Motivation and Context + Autoencoders (Except UAT) + VAE (Except Tradeoff and Components)
Ananya Gupta: Abstract + PCA + UAT + Preliminary Statistics + Tradeoff + Components of VAE + Reflection

Coding Component

Python Code using Pytorch

References

1. Jordan, J. (2018, July 16). Variational autoencoders. Jeremy Jordan. https://www.jeremyjordan.me/variational-autoencoders/

2. Huang, Y. (2022). Exploring Factor Structures Using Variational Autoencoder in Personality Research. Frontiers.

3. Hinton, G. E., Salakhutdinov, R. R. (2016, April 26). Reducing the Dimensionality of Data with Neural Networks. Www.Sciencemag.Org.

4. Smith, L. I. (2002). A tutorial on Principal Components Analysis. Elementary Linear Algebra 5e. http://axon.cs.byu.edu/Dan/478/Reading/PCA.pdf

5. Pictures from: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

6. Python Code written using: https://www.pytorchlightning.ai/