AIML 425 Assignment 3

James Thompson
Victoria University of Wellington
300680096

I. Introduction

Autoencoders are a type of neural network uses for unsupervised learning of efficient latent space representations of data. An auto encoder consists of two main parts the encoder which compresses the input data into a lower dimensional space and the decoder which reconstructions the original data. Variational Autoencoder are improved Autoencoders that change the output of the encoder to be a distribution over the latent space rather than a fixed vector and uses a ELBO loss function. I will explore both the generative capabilities of regular Autoencoders and Variational Autoencoders as well as the meaning of the latent space and how much information is retained in the latent space. I will train both models on a generated dataset of shapes of different size and locations on a 28x28 canvas. Both models achieve good reconstruction of the input data, however only the VAE is able to generate reasonable new images by sampling the latent space.

II. Theory

Autoencoder and their variations are a type of neural networks arhcitecture that are used for unsupervised learning of an efficient representation of the input data. Basic autoencoders use a encoder-decoder architecture to compress and then reconstruct the input data. Variational Autoencoders (VAEs) extend this architecture by introducing a probabilistic approach to the encoding process, allowing for the latent space to be a known distribution. This section provides a description of both model types as well as a discussion of the latent space.

A. Autoencoders

TODO: add mathematical/probablistic description of model

An Auto encoder has three important parts the encoder f_{enc}, the decoder f_{dec} and the latent layer z. The latent layer is typically of a much lower dimension (2D-32D) than the input data. The encoder and decoder are typically implemented as neural networks with learnable parameters ϕ and θ. Furthermore the encoder and decoder are generally the inverse of each other.

$$z = f_{enc}(x; \phi) \tag{1}$$

$$\hat{x} = f_{dec}(z; \theta) \tag{2}$$

The input data x goes through f_{enc} to z then through f_{dec} to produce the reconstruction $\hat{x}$. This model is then trained based on the reconstruction loss between the input and output data.

$$\mathcal{L}_{AE} = \|x - \hat{x}\|^2 \tag{3}$$

$$\tag{4}$$

Training with this loss function means that the model will learn a compressed representation of the data. However there is no reason that the distribution of the latent space will be smooth or continuous (i.e that all squares will be near each other in the latent space). This means that the generative capabilities of the model will be limited. Only certain part of the latent space will correspond to valid images, without access to the encoder and training data we have no way of knowing which parts of the latent space are valid. This means that sampling the latent space to generate new images is likely to produce random noise rather than valid images.

B. Variational Autoencoders

Variational Autoencoders (VAEs) are a type of generative model that builds upon the architecture of traditional autoencoders by introducing a probabilistic approach to the encoding process. In a VAE, the encoder does not produce a single point in the latent space but rather a distribution over the latent space, typically modeled as a Gaussian distribution. The goal is to learn the parameters of the generative model $p_\theta(x|z)$ which would allow us to generate new data points by sampling from the latent space.

$$q(z|x) = \mathcal{N}(z; \mu(x), \text{diag}(\sigma^2(x))) \tag{5}$$

$$\mu(x), \sigma(x) = f_{enc}(x; \phi) \text{ where } \mu \text{ is mean and } \sigma \text{ is variance}$$

$$z \sim q(z|x) \tag{6}$$

$$\hat{x} = f_{dec}(z; \theta) \tag{7}$$

This changes the latent space so that it comes from a normal distribution. Therefore the latent space is continuous, smooth and has a known distribution. The latent space will now be valid everywhere in the space, meaning that we can sample from the latent space to generate new images.

Furthermore, VAEs are trained using a loss function that combines the reconstruction loss (similar to traditional autoencoders) with a regularization term that encourages the learned latent distributions to be close to a prior distribution (typically a standard normal distribution). This regularization is achieved using the Kullback-Leibler (KL) divergence, which measures the difference between the learned distribution and the prior distribution. The overall loss function for a VAE can be expressed as:

$$\mathcal{L}_{VAE} = \mathcal{L}_{reconstruction} + \beta\mathcal{L}_{KL} \tag{8}$$

Using MSE for reconstruction loss and Gaussian prior:

$$= \mathbb{E}_{x\sim p_{data}(x)}\left[\log p_\theta(x|z)\right] - \beta D(q(z|x)||p(z))$$
$$= \|x-\hat{x}\|^2 + \beta\frac{1}{2}\sum_{i=1}^{d}\left(\mu_i^2+\sigma_i^2-\log(\sigma_i^2)-1\right) \tag{9}$$

This loss function is derived using the Jensen's inequality to maximize the evidence lower bound (ELBO) of the data likelihood, which will also maximize the likelihood of θ. The reconstruction loss ensures that the decoded outputs are similar to the inputs, while the KL divergence term regularizes the latent space to follow the prior distribution. The hyperparameter β controls the trade-off between these two objectives.

For back propagation through the stochastic sampling process, VAEs employ the reparameterization trick. This involves expressing the latent variable z as a deterministic function of the encoder outputs and a random noise variable ϵ sampled from a standard normal distribution:

$$z = \mu + \sigma\cdot\epsilon, \quad \epsilon\sim\mathcal{N}(0,I) \tag{10}$$

Where ϵ is sampled for the minibatch and assumed to be constant for the back-propagation step. This allows gradients to be computed with respect to the encoder parameters, enabling effective training of the VAE using standard back-propagation techniques.

C. Latent space information

1) Information rate: When we compress data, we inevitably lose some information. The amount of information retained in the compressed representation can be quantified using the concept of information rate. Assuming that we add noise to the latent space, $z = y + \epsilon$ where $\epsilon\sim\mathcal{N}(0,\sigma^2)$, the information rate can be calculated as:

$$R = \sum_i \frac{1}{2}\log_2\left(1+\frac{\sigma_{i,Z}^2}{\sigma_{i,\epsilon}^2}\right) \tag{11}$$

Where $\sigma_{i,Z}^2$ is the variance of the i-th dimension of the latent space and $\sigma_{i,\epsilon}^2$ is the variance of the noise added to that dimension.

This means that for a given noise level, we can estimate the amount of information that can be retained in the latent space by sampling the encoder outputs and measuring the variance of each dimension.

2) Information meaning: The information that the latent space retain can be interpreted in multiple ways. One way is that each dimension of the latent space could represent a specific feature of the input data. For example in my data generation process I generate each shape with only 4 numbers (x position, y position, size, shape). Therefore one could imagine a 4D latent space that perfectly encodes these 4 features. In practice this is unlikely to happen, but we can still interpret the latent space as a compressed representation of the input data.

III. Experiments

A. Dataset

The dataset used for training both models is a generated dataset of simple shapes (circles, squares, triangles) on a 28x28 canvas. Each shape is generated with random size and position. The shape may go up to the bounds but will never go out it and be clipped. There are 8 sizes ($[7,14]$) which means that there are a total 8002 unique images. The bounds of sizes were determined by how small the circles still look circle (lower bound) and no taking up more than half of the canvas (upper bound). The dataset used for training is 90% of the total possible images, and the test set is the remaining 10%. Holding a validation allows me to test the models ability to reconstruct unseen images, and so not just memorize the training data.

The size of the square determines it width/height, the size of the circle determines its diameter and the size of the triangle determines the heigh (but the size is modified to be a bit shorter to make it fit with the other image sizes). The shapes are drawn in white on a black background. The triangle is an upright isosceles triangle.

B. Variational autoencoder

1) setup: The VAE model is two fully connected neural networks. The encoder outputs 2 values (mean and variance) while the decoder outputs a single scalar. The loss function used was the ELBO loss function with a β of 1 (using BCE for reconstruction error and KL divergence for regularization). A complete list of hyperparameters is shown in table I. The fine tuning of parameters was done through trial and error using the loss curve and generative performance metrics as a guide.

2) generative performance: There are different ways of understanding the generative performance of the model. This situation we are involves a finite set of valid images (8002) in a very large space of images (2^{784}). Using this valid image set we can calculate three different metrics. Therefore we can use this valid image set to evaluate; how many of the valid images can the model generate (coverage), how close are the generated images to the nearest valid image (nearest neighbor performance) and how similar is the distribution of generated images to the distribution of valid images (KL divergence).

$$\text{coverage} = \frac{|\{x\in\text{images}\mid \min_{\hat{x}\in\text{generated}} d(x,\hat{x}) < \text{thres}\}|}{|\text{images}|}$$
$$\text{nearest neighbor} = \frac{1}{|\text{images}|}\sum_{x_i\in\text{images}}\min_{\hat{x}_j\in\text{generated}} d(x_i,\hat{x}_j)$$
$$\text{KLD} = D_{KL}(p_{data}||p_{model})$$
$$\approx \sum_{x_i\in\text{images}} p_{data}(x_i)\log\frac{p_{data}(x_i)}{p_{model}(x_i)}$$
$$= \sum_{x_i\in\text{images}}\frac{1}{|images|}\log\frac{1}{q_{counts}(x_i)}$$

Where $d(x_i,\hat{x}_j)$ is the distance between two images, and the thres is a hyperparameter that determines how close the generated image needs to be to a valid image to count as ”an attempt at generating it”, and $q_{counts}(x_i)$ is the number of times the model has generated an image close enough to x_i (within the threshold). I choose euclidean as the distance metric and the threshold of 3 by looking at sample images and aligning the cutoff and distance metric with my intuition.

The results of the generative peformance can be found in **??**

C. Autoencoder

1) setup:

2) Inforamtion rate:

3) Information meaning:

D. Autoencoder vs Variational autoencoder

IV. Conclusion

TABLE I
HYPERPARAMETERS USED FOR TRAINING THE VAE AND AE MODELS.

Hyperparameter	**VAE**	**AE**
Latent dimension	32	10
Encoder architecture	[2000, 2000, 2000]	[2000, 2000, 2000]
Decoder architecture	[2000, 2000, 2000]	[2000, 2000, 2000]
Optimizer	Adam(lr=0.001)	Adam(lr=0.001)
Minibatch size	256	512
Number of epochs	200	200

Statement

TODO: add colab link The code and report of the Assignment was solely completed by myself (Thompson James). The complete source code can be found here https://gitea.james-server.duckdns.org/james/AIML425_assignment_3, with a link to a colab notebook found here: TODO:addcolablink. A complete run through of the notebook takes about an hour to complete on a university lab machine.

I completed my work using the following tools:

- **Jupyter Notebook [1] and JupyterText [2]:** For interactive development and hosting.
- **LaTeX**: For writing the report.
- **VSCode [3]:** As IDE.
- **JAX [4] and Flax [5]:** For implementing the neural network and training logic.
- **Matplotlib [6] and Pandas [7]:** For data visualization and management.

References

[1] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, “Jupyter Notebooks – a publishing format for reproducible computational workflows,” in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87–90.

[2] M. Wouts, “Mwouts/jupytext,” Aug. 2025.

[3] “Microsoft/vscode,” Microsoft, Jul. 2025.

[4] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, “JAX: Composable transformations of Python+NumPy programs,” 2018.

[5] J. Heek, A. Levskaya, A. Oliver, M. Ritter, B. Rondepierre, A. Steiner, and M. van Zee, “Flax: A neural network library and ecosystem for JAX,” 2024.

[6] J. D. Hunter, “Matplotlib: A 2D graphics environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[7] The pandas development team, “Pandas-dev/pandas: Pandas.”

Appendix