

Course Code: CSCM77
Topic: Variation Autoencoder and
Generative Adversarial Networks
Student number: 2216686
Student Name: Linda Mafunu

Table of Contents

<i>Introduction.....</i>	<i>2</i>
Generative Adversarial Networks	2
Variational Autoencoders	2
<i>Methodology.....</i>	<i>3</i>
GAN.....	3
VAE.....	3
<i>Results</i>	<i>4</i>
<i>Discussion and Conclusion</i>	<i>4</i>
VAE.....	4
GAN.....	5
<i>References</i>	<i>6</i>
<i>Appendix</i>	<i>7</i>

Introduction

Generative models model the distribution of individual class, whereas discriminative models look for any decision boundary that will split training samples for each class (Turhan et al., 2018). Rashad (2020) indicates that the two most used methodologies for generative modelling are Generative Adversarial Network (GAN) and Variational Autoencoder (VAE). The generative model is typically employed in unsupervised learning, although it can also be used in supervised learning settings. According to Hinners et al. (2018), supervised learning involves learning a mapping from inputs to outputs based on labelled input-output pairings. For unsupervised learning, Brownlee (2019) emphasises that it just requires input variables and not output variables. Its model is formed by extracting or summarising patterns in the input data; there is no model correction because the model does not predict anything. When generative models are unsupervised, the network is trained to learn the distribution of input, which may then be used to generate new material rather than categorise it, as in Variation Autoencoders (Rashad, 2020). In a supervised learning setting, Generative Adversarial Networks may be trained and supervised by conditioning the generator on specific criteria. In this report we will look at the implementation of VAEs and GANs on the MNIST dataset.

Generative Adversarial Networks

Brownlee (2019) describes GAN as an unsupervised machine learning issue that involves automatically finding and learning regularities or patterns in incoming data that the model may generate. He goes on to say that GANs are a brilliant way to train a generative model, framing the task as a supervised learning problem with two sub-models: discriminator and generator. According to Xao et al. (2017), a shortcoming of the GAN loss function is that it causes vanishing gradients in the learning process; therefore, a solution to this conundrum is to use Least Mean Square GAN (LSGAN). They use the least mean square function for the discriminator to produce higher-quality images than traditional GANs. Radford and Chintala (2015) present another alternative, Deep Convolutional GAN, which increases performance by incorporating convolutional networks into both the generator and the discriminator. In this paper, we will look at the differences in performance between the three models.

Variational Autoencoders

Data compression is an important part of network training because it allows the same amount of information to be given in fewer bits. This helps to address the problem of high dimensionality, which is where autoencoders and VAEs come in handy because they enable end-to-end networks for data compression (Anwar, 2021). Autoencoders train efficient embeddings of unlabelled input for a specified network architecture. According to Anwar (2021), the problem with autoencoders is that linear samples between two encodings do not provide generated samples due to discontinuities in the embedding space that prevent smooth transitions. The VAE approach addresses this issue by introducing a probabilistic spin into the encoder and decoder, allowing us to sample from the latent space's learning distribution to generate new images at inference time (Anwar, 2021). Rahman (2020) proposes using Conditional Variational Autoencoders (CVAEs) to address the issue of developing models without control over the data that most generative models have. CVAEs are an extension of VAEs that include conditional information in the model. In this report we will look at the implementation of VAE, CVAE and apply them to MNIST dataset.

Methodology

This section focuses on the implementation of GANS and VAEs. We first set up our developer environment in Google collab and import the necessary python libraires. We download our MSNIST dataset, which has 60000 training and 10000 test pictures, and save it in a folder. We will utilise the data set to train our models and assess its performance with the test data. To determine if the data was successfully downloaded, we choose a batch and visualize the data (see Fig.1).

GAN

Our first network, the discriminator, will determine whether the images are genuine (from the training set) or not. To create a discriminator, we use entirely connected layers that include bias terms (Goodfellow et al.,2014). The generator is the second network, and it employs a neural network to convert random noise to images.

To increase the likelihood of the discriminator selecting the incorrect choice, we alternate the following steps:

1. Update the generator (G) to maximize the possibility that the discriminator will make the wrong choice on generated data:

$$\text{maximize}_G E_{z \sim p(z)} [\log D(G(z))]$$

2. Update the discriminator (D) to maximise the likelihood of making the correct choice on both real and created data:

$$\text{maximize}_D E_{x \sim \text{pdata}} [\log D(x)] + E_{z \sim p(z)} [\log (1-D(G(z)))]$$

In this equation, $z \sim p(z)$ represents random noise samples, $G(z)$ represents images generated by the neural network generator (G), and D represents the discriminator's output, which indicates the likelihood of an input being real.

The discriminator loss is calculated using binary cross-entropy loss, which evaluates the difference between the discriminator's prediction and the actual image labels in a more stable manner. To compute it, we add the binary cross-entropy losses for false and actual data. For generator loss, we obtain the binary cross-entropy loss for fake data. To optimise the loss, we employ a gradient descent optimisation technique like Adam, which is effective in neural networks.


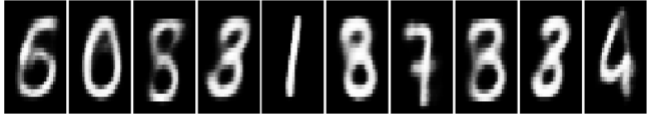


For the LSGAN, we implement the discriminator's least square loss function to compute generator and discriminator loss (Mao et al.,2017). The least square loss function penalises fake samples, motivating the generator to produce real images. The DCGAN design, we employ convolutional networks for both the generator and the discriminator. After defining the previously mentioned, we train using the model the dataset and visualise the outcomes of the three distinct architectures. For the architectures of the models refer to appendix (see Fig 3.1-3.4).



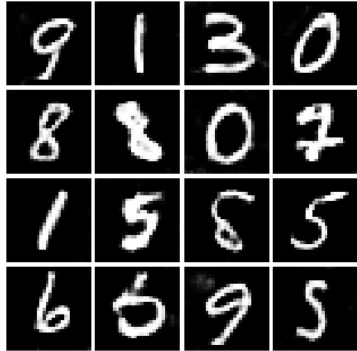
VAE

The VAE consists of an encoder and a decoder. The encoder compresses data from higher to lower dimension. The encoder network converts images to latent vectors. Our initial VAE will consist of entirely linked layers that will flatten the input image's characteristics. Flattening images simplifies the architecture, allows the encoder to function as a feature extractor, and makes the model's output easier to manage and visualise. Following flattening, our encoder will take our images as input and pass them via the three Linear + ReLu layers. We utilise a hidden dimension representation with two independent linear layers to predict posterior mean and posterior log-variance. The decoder, unlike the encoder, takes the latent space representation and reconstructs an image. The loss function is made up of two components: the reconstruction loss, which

evaluates how effectively the decoder can reconstruct the input data, and the KL divergence, which indicates how much the learnt distribution deviates from a conventional Gaussian distribution. Following this, we use the reparameterization method to estimate the posterior z during the forward pass based on the encoder's mean and variance. For the architectures of the discriminator and generator models refer to appendix (see Fig 2.1 and 2.2). CVAEs are VAEs with added conditional information. CVAEs have the same architecture as VAEs, but we add a single-hot label vector to both the flattened picture and the latent space. After specifying the previously mentioned, we train the model using MNIST dataset and display the results.

Results

Training Losses	Reconstructed Image Output	Model
<div>  Train Epoch: 0 Loss: 155.775269 Train Epoch: 1 Loss: 141.299377 Train Epoch: 2 Loss: 125.814720 Train Epoch: 3 Loss: 121.087906 Train Epoch: 4 Loss: 121.011955 Train Epoch: 5 Loss: 117.449791 Train Epoch: 6 Loss: 113.615326 Train Epoch: 7 Loss: 114.995316 Train Epoch: 8 Loss: 112.340248 Train Epoch: 9 Loss: 108.839287 </div>		VAE
<div>  Train Epoch: 0 Loss: 137.802429 Train Epoch: 1 Loss: 131.535339 Train Epoch: 2 Loss: 124.594666 Train Epoch: 3 Loss: 115.604248 Train Epoch: 4 Loss: 118.044617 Train Epoch: 5 Loss: 110.885292 Train Epoch: 6 Loss: 114.613480 Train Epoch: 7 Loss: 108.608154 Train Epoch: 8 Loss: 107.402344 Train Epoch: 9 Loss: 106.561836 </div>		CVAE

		
GAN	LSGAN	DCGAN

Discussion and Conclusion

VAE

The VAE learns the mean (μ) and standard deviation (σ) of a Gaussian distribution. These are then utilised to draw from a parameterized distribution. The encoder learns to anticipate two vectors: the mean and the standard deviation. They are then utilised to parameterize the distribution and

create a sample z , which is decoded using the trained decoder (see Figs. 2.2 and 2.3 in the appendix). The difficulty is that the process of sampling from the distribution parameterized by our model is not differential; to address this, we applied the reparameterization approach to our embedded function (Anwar,2021). The reparameterization approach enables the model to backpropagate throughout the sampling procedure. This approach entails reparametrizing the model to provide the distribution's parameters, which are then used to sample from the distribution. This makes the model differentiable and enables efficient training. To stabilise the training, we employ the logvar layer rather than standard variance (Anwar,2021). The loss function of a VAE is designed to encourage the model to learn an accurate representation of the input data in latent space. The objective is to limit both losses.

When comparing the performance of VAE and CVAE for the training process, CVAE appears to outperform VAE, as seen by the lower loss results in the table above. This is because CVAE constantly leans the posterior distribution. This allows it to learn numerous predictions from a single input, resulting in increasingly accurate and realistic predictions over time. CVAE incorporates conditional information into the model by incorporating the one-hot label in the flattened picture and the latent space, which improves embedded space learning and results in more accurate reconstructions that closely resemble the true data distribution.

GAN

The discriminator's role is to examine each input and determine if it contains true data or not. It returns a probability score between 0 and 1, with 1 indicating that the data is likely to be real and 0 indicating that it is not (Goodfellow et al., 2014). The generator's function is to generate new data instances, such as images that mimic actual data. It takes a random noise vector as input and uses internal layers and learnt patterns to create a new data sample (Brownlee 2019). Training GAN may be difficult since it involves precise hyperparameter adjustment as well as a balance between the generator and discriminator. GAN has vanishing gradients, which make it difficult for the generator to learn successfully since the gradients are too tiny to cause meaningful changes to the generator's weights. The quality of the produced pictures for the GAN fluctuates since GAN training is a complex process due to the instability of GAN learning and training (Mao et al., 2017).

LSGANs increase learning process stability by using the least squares loss function (Mao et al., 2017). The function can pull false samples closer to the decision boundary since it penalises samples that are far from the proper side of the decision boundary. It penalises fraudulent samples and draws them closer to the decision boundary, even if they are correctly categorised (Mao et al., 2017). This pushes the generator to create images that are closer to the original data, resulting in higher quality output images than GAN. DCGAN employs convolutional networks on both the generator and the discriminator (Radford & Chintala, 2015). This design enables DCGAN to train more effectively and capture spatial hierarchies in the data, resulting in more realistic pictures when compared to GAN and LSGAN.

GAN is more efficient in producing high-quality pictures than VAE, although the training process can be unstable. VAE, on the other hand, provides a more consistent and regulated generating process, but the picture generation quality is low. Both GAN and VAE are generative models; the decision between the two is based on task constraints and desired picture quality.

References

- Anwar, A. (2021, November 4). *Difference between autoencoder (Ae) and variational autoencoder (Vae)*. Medium. <https://towardsdatascience.com/difference-between-autoencoder-ae-and-variational-autoencoder-vae-ed7be1c038f2>
- Brownlee, J. (2019, June 16). *A Gentle Introduction to Generative Adversarial Networks (GANs)*. Machine Learning Mastery. <https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/>
- Chen, X., Duan, Y., Houthoof, R., Schulman, J., Sutskever, I., & Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in neural information processing systems*, 29.
- Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014, June 10). *Generative Adversarial Networks*. ArXiv.org. <https://arxiv.org/abs/1406.2661>
- Hinners, T. A., Tat, K., & Thorp, R. (2018). Machine learning techniques for stellar light curve classification. *The Astronomical Journal*, 156(1), 7.
- Mao, X., Li, Q., Xie, H., Lau, R. Y. K., Wang, Z., & Smolley, S. P. (2017). Least Squares Generative Adversarial Networks. *ArXiv:1611.04076 [Cs]*. <https://arxiv.org/abs/1611.04076>
- Radford, A., Metz, L., & Chintala, S. (2015). *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. ArXiv.org. <https://arxiv.org/abs/1511.06434>
- Rahman, M. A. (2020, May 20). *Understanding conditional variational autoencoders*. Medium. <https://towardsdatascience.com/understanding-conditional-variational-autoencoders-cd62b4f57bf8>
- Rashad, F. (2020, September 22). *Generative Modeling with Variational Auto Encoder (VAE)*. ViTrox-Publication. <https://medium.com/vitrox-publication/generative-modeling-with-variational-auto-encoder-vae-fc449be9890e>
- Turhan, C. G., & Bilge, H. S. (2018). Variational autoencoded compositional pattern generative adversarial network for handwritten super resolution image generation. *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, 564–568. <https://doi.org/10.1109/UBMK.2018.8566539>

Appendix

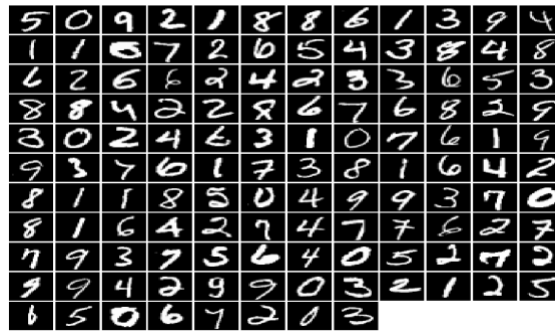


Fig. 1: MNIST dataset

Define an encoder, `hidden_dim (H)`, `mu_layer`, and `logvar_layer` in the initialization of the VAE class in `vae.py`. Use `nn.Sequential` to define the encoder, and separate `Linear` layers for the `mu` and `logvar` layers. In all of these layers, `H` will be a hidden dimension you set and will be the same across all encoder and decoder layers. Architecture for the encoder is described below:

- Flatten (Hint: `nn.Flatten`)
- Fully connected layer with input size 784 (`input_size`) and output size `H`
- ReLU
- Fully connected layer with input size `H` and output size `H`
- ReLU
- Fully connected layer with input size `H` and output size `H`
- ReLU

Fig 2.1: VAE Encoder

- Fully connected layer with input size as the latent size (`Z`) and output size `H`
- ReLU
- Fully connected layer with input size `H` and output size `H`
- ReLU
- Fully connected layer with input size `H` and output size `H`
- ReLU
- Fully connected layer with input size `H` and output size 784 (`input_size`)
- Sigmoid
- Unflatten (`nn.Unflatten`)

Define a decoder in the initialization of the VAE class in `vae.py`. Like the encoding step, use `nn.Sequential`.

Fig 2.1: VAE Decoder

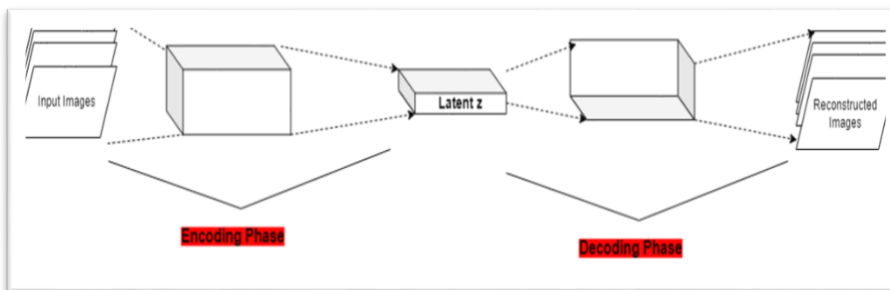


Fig 2.3: VAE

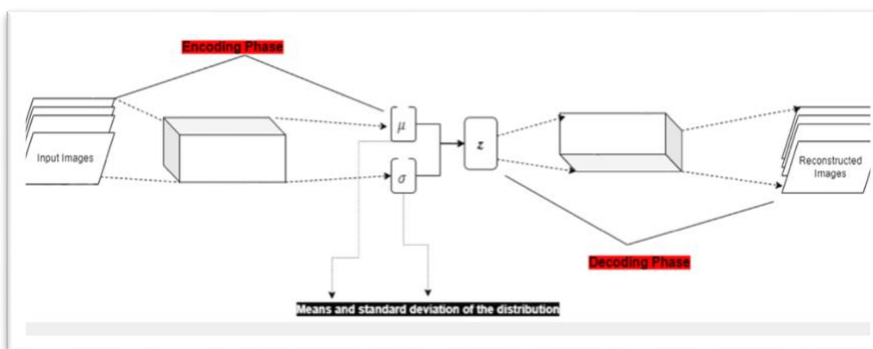


Fig 2.3: VAE ($\sigma + \mu$)

<div> <div>Generator</div> <p>Now to build the generator network:</p> <ul style="list-style-type: none"> Fully connected layer from noise_dim to 1024 ReLU Fully connected layer with size 1024 ReLU Fully connected layer with size 784 Tanh (to clip the image to be in the range of [-1,1]) </div>	<p><i>Fig 3.1: GAN Generator</i></p>
<div> <div>Discriminator</div> <p>Our first step is to build a discriminator. All fully connected layers should include bias terms. The architecture is:</p> <ul style="list-style-type: none"> Fully connected layer with input size 784 and output size 256 LeakyReLU with alpha 0.01 Fully connected layer with input_size 256 and output size 256 LeakyReLU with alpha 0.01 Fully connected layer with input size 256 and output size 1 </div>	<p><i>Fig 3.2: GAN Generator</i></p>
<div> <ul style="list-style-type: none"> Fully connected with output size 1024 ReLU BatchNorm Fully connected with output size 7 x 7 x 128 ReLU BatchNorm Reshape into Image Tensor of shape 7 x 7 x 128 Conv2D* T (Transpose): 64 filters of 4x4, stride 2, 'same' padding (use padding=1) ReLU BatchNorm Conv2D* T (Transpose): 1 filter of 4x4, stride 2, 'same' padding (use padding=1) Tanh Should have a 28 x 28 x 1 image, reshape back into 784 vector </div>	<p><i>Fig 3.3: DCGAN Generator</i></p>
<div> <div>Discriminator</div> <p>We will use a discriminator inspired by the TensorFlow MNIST classification tutorial, which is able to get above 99% accuracy on the MNIST dataset fairly quickly.</p> <ul style="list-style-type: none"> Reshape into image tensor (Use nn.Unflatten) Conv2D: 32 Filters, 5x5, Stride 1 LeakyReLU(alpha=0.01) Max Pool 2x2, Stride 2 Conv2D: 64 Filters, 5x5, Stride 1 LeakyReLU(alpha=0.01) Max Pool 2x2, Stride 2 Flatten Fully Connected with output size 4 x 4 x 64 LeakyReLU(alpha=0.01) Fully Connected with output size 1 </div>	<p><i>Fig 3.4: DCGAN Discriminator</i></p>