CW2: Generative Adversarial Network

Kevin Yu - kky16@ic.ac.uk - 01185891 Joe Gardner - jog116@ic.ac.uk - 01232601

1. Introduction

Using the MNIST dataset, the coursework aims to generate synthetic images of handwritten numerical digits with the aid of a generative adversarial network (GAN). The MNIST dataset comes loaded with 60,000 samples for training (`x_train`) and 10,000 samples for testing (`x_test`), each being monochrome 28x28 images.

1.1. Dataset

The images are first preprocessed such that the range of values are normalised from [0,255] to [-1,1]. GANs are notoriously difficult to train; hence, discriminator normalisation (and consequently generator normalisation) is implemented to make training easier in the form of more efficient gradient flow and stable optimisation [1].

1.2. GAN

GANs are a type of generative model as the model attempts to learn the underlying distribution of the data (p_{data}) to produce an estimate (p_{model}) commonly through maximum likelihood estimation (MLE). The parameters of the model are fitted to capture either the joint probability $P(X, Y)$ or simply $P(X)$ distribution.

The structure of GANs differ as the network consists of both a discriminator D and generator G part. The generator's objective is to create samples from random noise which appear 'real' while the task of the discriminator is to determine the 'real' samples from the 'fake'. In this sense, the networks are in contest with each other often in a zero-sum game as any gain in the discriminator's ability to detect 'real' images $D(x)$ leads to loss in generator's ability to fool the discriminator $D(G(z))$ and vice versa. The minimax definition can be set where D aims to maximize while G aims to minimize the value V as:

$$\begin{aligned}\min_G \max_D V(D,G) = E_{\mathbf{x}\sim p_{data}}[log(D(\mathbf{x}))] \\ + E_{\mathbf{z}\sim p_{\mathbf{z}}}[log(1-D(G(\mathbf{z})))]\end{aligned} \tag{1}$$

The solution is also the Nash equilibrium of the problem $D(x) = \frac{1}{2}$ where the discriminator can no longer distinguish between 'real' and 'fake' inputs and $p_{model} = p_{data}$.

Each turn in the game is represented by one iteration (one batch) of the training process of either the discriminator or generator. The generator is trained by passing its output of 'fake' images into the discriminator i.e. $D(G(z))$ and then the gradient of the loss is back propagated directly to and through the generator as:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} log(D(G(\mathbf{z}^{(i)})) \tag{2}$$

where m is the size of each batch.

On the other hand, the discriminator is trained by passing 'real' images or training set (`x_train`) i.e. $D(x)$ and then the cross entropy loss is calculated and back propagated exclusively over the discriminator network, while locking the generator network, to update its weights.

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [log(D(\mathbf{x}^{(i)})) + log(1-D(G(\mathbf{z}^{(i)}))] \tag{3}$$

1.3. Evaluation

The accuracy of the models tested are determined both qualitatively and quantitatively through visual inspection and the measurement of inception scores. These scores are determined by training a separate classifier network based on LeNet [2]. This classifier consists of two convolutional layers and a single dense layer. It has been pre-trained to a recognition accuracy of 99.17% and a testing accuracy of 98.64% using the same MNIST dataset used for training the GANs. The accuracy of this classifier when provided with a set of generated images from a GAN (with corresponding labels, see section 3) is calculated, providing a more objective measure of how well each network performs.

2. Deep Convolutional GAN

A deep convolutional generative adversarial network (DCGAN) follows the GAN model as described before, except it achieves this by using convolutional and deconvolutional layers.

2.1. Architecture

The generator and discriminator are both convolutional neural networks (CNNs). The discriminator takes in a randomised batch of 28x28 pixel MNIST images (or generated

ones) and consists of initially several convolutional layers in order to extract features and reduce the dimensionality of the images. Each convolutional layer halves the side dimension of each image (i.e. 28x28 to 14x14) using a 5x5 kernel and is followed by batch normalisation (BN), a leaky ReLU activation function and a dropout function. After 3 of these layers, the output tensor is flattened and fed into a dense layer of size 4 and once again passed through BN and leaky ReLU. It is finally passed through a size 1 dense layer with sigmoid activation, giving with a single output between 0 and 1, representing the probability of the input image being real.
The generator takes as input a 100x1 vector of normally distributed random variables ($\mu = 0$, $\sigma = 1$). The first layer is a dense one, scaling it up to 1024x1, followed by BN and leaky ReLU activation. It is then scaled up again to (7*7*256)x1 with the same method before being reshaped into a 7x7x256 tensor, ready for being passed into a transposed convolutional layer. There are 3 transposed convolutional layers. The first reduces the tensor to 7x7x128 and is followed by BN and a leaky ReLU. The second scales to 14x14x64 (with the same BN and activation) and the third finally scales it up to a single image of 28x28x1, this being the generated output.

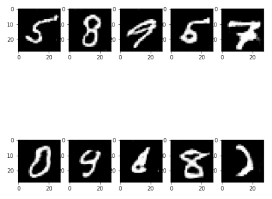


Figure 1: DCGAN output images using the best model

2.1.1 Depth

The depth of the GAN represents the number of convolutional layers the discriminator and generator have (keeping the ratio between them the same). Adding more layers extract more features up to a certain extent before overfitting.

Increasing the depth significantly increase the time per epoch (91 s) without major improvements to the images. Hence, they were not explored any further.

2.1.2 Balance

The balance of a GAN is needed to ensure one network does not overpower the other. If the discriminator is strongly biased the generator will converge to only generate noise (vanishing gradient) and conversely if the generator is strongly biased the discriminator will recognise all images as real. This can occur if one network is more suspectable to training resulting in varied rates in gradient descent. Usually, the discriminator overpowers the generator as it is a larger and deeper network. Hence, the depth of the discriminator was varied to judge the effects and the uneven training ratios between the discriminator and generator were tested as another method of balancing the system.

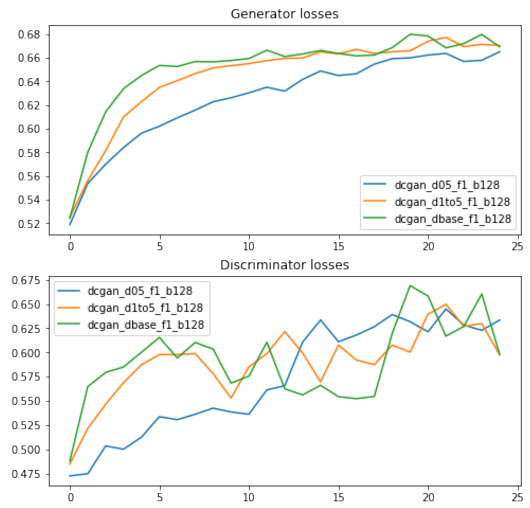


Figure 2: DCGAN output images using the best model

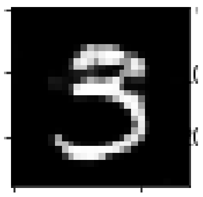
The losses of the different DCGANs can be seen to be roughly similar with the best remaining as the standard architecture showing that the model was originally balanced. However, advantages were found by altering the ratio of training 1:5 discriminator:generator greatly reducing execution time (t = 24 s/epoch) in comparison with increasing the depth to adjust for balance (t = 58 s/epoch) while producing similar losses.

2.1.3 Batch Normalisation

BN simply computes the normal distribution with the mean and variance of the current minibatch. Placing a BN after each Dense and convolution layer avoids overfitting by providing regularization and allows for smooth flow through the network.

The method of BN creates dependency between samples and hence produce similar aspects in the images generated. Virtual batch normalisation (VBN) avoids this by sampling a reference batch before training, otherwise known as reference batch normalisation (RBN). Additionally, to mitigate the effects of overfitting from a single batch VBN combines RBN with the current batch to compute normalisation parameters.

Overall, the VBN showed no improvement over standard architecture due to it already implementing a form of normalisation. For more complicated image datasets, the dependency avoided by VBN is visually evident by generated images of the same batch showing mutual colour tint.


(a) Image without VBN

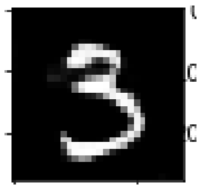
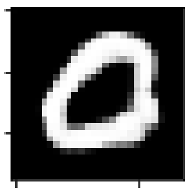

(b) Image with VBN

Figure 3: Varying virtual batch normalisation

However, simpler lower resolution datasets like MNIST this problem clearly does clearly exist.

2.1.4 Filter

The filter size of the CNN determines the number of channels in each layer. Thus, it alters the capacity of the network. As the filter size increases, it is able to capture a greater number of features and hence enable the generator to produce more accurate images. The filter size is changed by adding filter multipliers (`f`) throughout the network.


(a) No extra filters

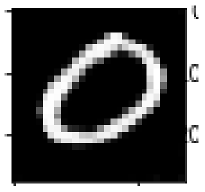

(b) Double the number of filters

Figure 4: Varying the number of filters

Analysing the different filter multiples, it can be seen that upping the filter size does infact produce visually better images. As noticed by the difference from `f = 1` to `f = 2`, the strokes become more precise as the DCGAN is more confident in determining the correct corresponding features.

2.2. Hyperparameter

2.2.1 Batch Size

The batch size (`B`) controls the accuracy of the estimation of the loss of the model. Therefore, smaller batch sizes tend to induce a more volatile training process while larger values of `B` cause slower but stable convergence.

B = 128, t = 26 s/epoch B = 64, t = 33 s/epoch B = 32, t = 40 s/epoch

A smaller number of batch sizes took longer time to train but can also prevent mode collapse by encouraging training diversity and generalisation as the model is forced to extract features with less amount of information.

2.2.2 Optimiser

The optimizers that allowed for convergence showed no significant variance in performance these include in ascending order of time as Adam (t = 28 s/epoch), SGD (t = 30 s/epoch), Adagrad (t = 34 s/epoch), RMSProp (t = 40 s/epoch).

2.2.3 Learning Rate

The learning rate was experimented for the most ideal optimiser Adam. The best results visually were produced using a larger rate lr = 0.001 compiling the model faster compared to the default lr = 0.0001 and for smaller values no convergence was achieved and only noise was produced.

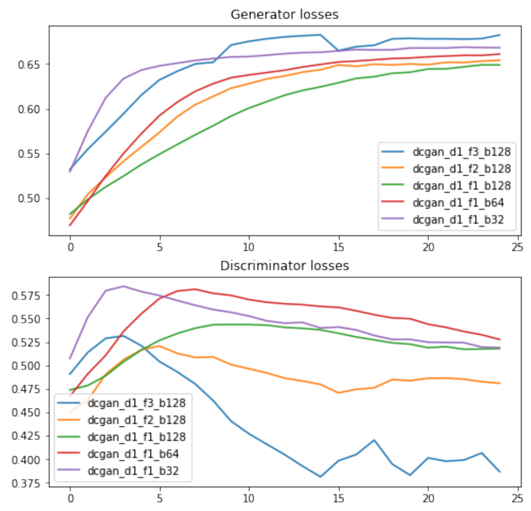


Figure 5: Losses of various DCGANs tested

3. Conditional GAN

A conditional generative adversarial network (CGAN) utilizes the labels (`y_train`) in addition to `x_train`. The labels are passed through both the discriminator and generator networks in conjunction with the training data, concatenated together and then proceed to follow the same architecture as a DCGAN. As seen in the previous section, GANs can suffer badly compared to other deep neural networks, so introducing labels to the system is another method for improving the performance of GANs and help training. The labels are expressed in the form of one-hot encoding to introduce the same dimensionality as their associated images, e.g. 3 = [0 0 0 1 0 0 0 0 0 0]

The minimax game can then be formulated as:

$$\min_G \max_D V(D,G) = E_{\mathbf{x},l\sim p_{data}}[log(D(\mathbf{x},l))] + E_{\mathbf{z}\sim p_{\mathbf{z}},l\sim p_l}[log(1-D(G(\mathbf{z},l),l))] \tag{4}$$

3.1. Architecture

The discriminator of the CGAN initially consists of two separate branches to handle the image data and labels separately. The image data is passed through a convolutional layer with a 5x5 kernel to produce a tensor of size 28x28x64 (retaining image dimensions). The 10x1 label data is passed first into a dense layer of size 28*28*64 then reshaped into a 28x28x64 tensor before being passed through a similar convolutional layer, retaining the size. The image and label data are now of the same dimensions so are concatenated together to produce a 28x28x128 tensor. This tensor is then passed through the same convolutional layers as in the DCGAN discriminator, to reduce the data to a single output value.

The generator passes both the noise vector and the labels through identical branches before being concatenated and sent through the transposed convolutional DCGAN layers used previously. These branches consist of first a dense layer of 7*7*256 with BN and leaky ReLU activation function, before being reshaped into a 7x7x256 tensor and passed through a transposed convolutional layer. This layer uses a 3x3 kernel and retains the tensor dimensions. Finally the two branches are concatenated into a 7x7x512 tensor, ready for the transposed convolutional layers of the DCGAN.

3.1.1 Filter size

Varying the filter size improves the quality of the images as mentioned previously, as expected due to a larger number of features being extracted. Inception scores of 95.8% and 97.8% were achieved for no added filters and double the number of filters respectively. Visually the effects of this are show in figure 4.

3.2. Hyperparameters

3.2.1 Batch Size

Varying the batch size, as with the DCGAN, slowed the training time significantly as it was increased. However, inception scores of 95.8%, 91.7% and 97.1% were achieved for batch sizes of 128, 64, and 32 respectively. This indicates a possible increase in accuracy with batch size, which can be explained through smaller batches containing less information and therefore allowing the model to become more generalised. It is suggested that the lower value for a batch size of 64 is an anomaly due to the random nature of training these networks.

3.2.2 Optimiser and Learning Rate

Similarly to DCGAN, the optimisers showed no significant in performance, this time qualitively giving IS = 95.8%, 95.2%, 95.1%, 94.8% respective to Adam, SGD, Adagrad, RMSProp.

The learning rate was previously determined and showed no difference when alternating to CGAN.

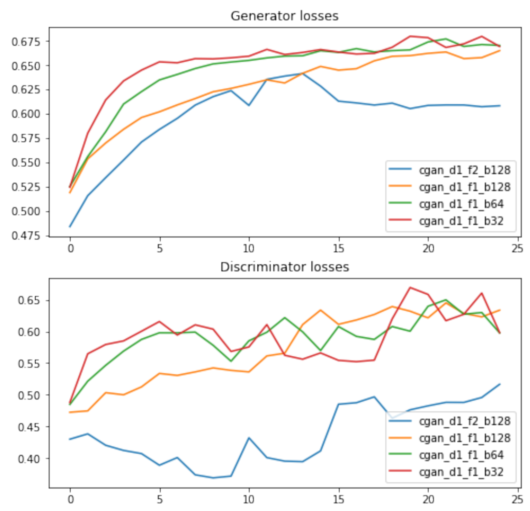


Figure 6: Losses of various CGANs tested

3.2.3 Conclusion

In conclusion, the best performing GAN that was achieved was a CGAN with a batch size of 32, no extra filters and no extra depth. This GAN performed very well, with an inception score of 97.1%, with examples images shown below. An exhaustive search of hyperparameters could not be carried out due to time constraints.

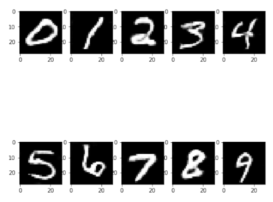


Figure 7: CGAN output images using the best model

References

[1] K. Kurach, M. Lucic, X. Zhai, M. Michalski, and S. Gelly. A large-scale study on regularization and normalization in gans, 2018.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.