Coursework on Generative Adversarial Networks

Ashish Sundar
CID: 01058844
as13215@ic.ac.uk

Harshil Sumaria
CID: 01053573
hs2715@ic.ac.uk

Abstract

In this coursework we use GANs (DCGAN and CGAN) to generate synthetic handwritten digit images using the MNIST dataset. The synthetic images are compared to the real images and used to train a classifier and evaluated. Python and tensorflow was used.

1. DCGAN

1.1. Architecture

There are two networks competing against each other, the generator and the discriminator. The generator aims to create images which can fool the discriminator into thinking they are real images, whereas the discriminator indicates whether the generated image is real or fake. The generator architecture consists of 'Conv2DTranspose' layers which upsample the generated noise with convolutional filters twice in order to reach the MNIST image size of 28x28x1. Each layers has a Leaky ReLU activation with alpha value 0.2, except for the output which has a tanh layer.

The discriminator architecture is a convolutional neural network (CNN) based classifier. It has 2 'Conv2D' layers and outputs a positive value if it classifies the image as real, and negative if it classifies it as fake.

Virtual batch normalisation (VBN) and early stopping are used as regularisation methods. Batch normalisation is where only a small batch of data is provided at a time instead of providing the whole dataset, which has a regularising effect on the model. Futhermore, early stopping is also used to check when mode collapse occurs and the model hasn't recovered, or if both the generator and the discriminator losses straighten out after a lot of epochs.

The optimisers used for the generator and discriminator were both Adam. The loss function was 'Binary Cross Entropy'.

1.2. Varying Architectures

A trial was carried out where one of the convolution layers were removed and the model was run. It was surprisingly found that the model was able to generate recognisable numbers at a lower number of epochs, and after 50 epochs the model with one layer produced better images. This may be that a weaker generator was less likely to lead to mode collapse in which the generator wins the minmax game that GANs are based on.

1.3. Hyper-Parameters

As part of testing for the DCGAN, hyper-parameters such as batch-size, epochs and learning rate were varied.

Batch-Size

The values tried for batch-size were 32, 64, 128, 256, 512 and 1024. We found that a lower batch-size took a longer time to train however the images took fewer epochs to become recognisable as numbers. The lower batch-size images were of a far superior quality. This is due to the fact that a lower batch-size allows for more regularisation which in turn improves performance.

Epochs

As the number of epochs increased the quality of the produced images improved initially as the generator tried to beat the discriminator. However, once mode collapse occurred the number of epochs no longer affected the image quality and they merely fluctuated slightly. After around 50 epochs the changes were minimal.

Learning Rate

The Discriminator Learning Rate (DLR) and Generator Learning Rate (GLR) were both varied and images were produced. It was found that the best performance was attained when the discriminator was stronger than the generator, but if this was made too strong it would collapse as detailed in the Mode Collapse section.

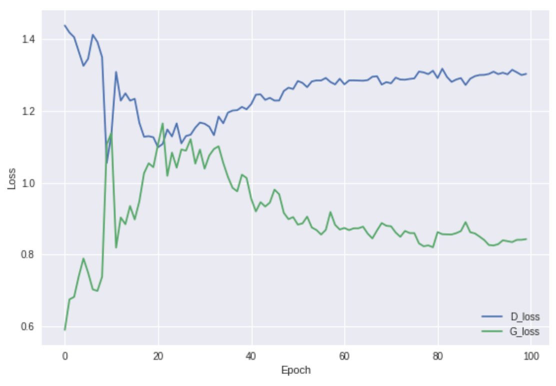

Figure 1. An example loss function for the DCGAN

1.4. Challenges and how they are addressed

Quality of Images

Mode Collapse

Most GANs suffer from a common issue known as mode collapse, where instead of both the networks being equally good and thus improving each others' performance, one network ends up winning the minmax function and the other network starts producing noise. This is because the losing network essentially gives up - no matter what it produces the winning network will always get it right. It usually happens when the discriminator gets too good - since it's much easier to tell whether an input is fake or real than creating it.

We saw the mode collapse occur from the graphs of the loss functions as well as the outputs produced. The loss functions for the discriminator and generator would initially interact, and then diverge and stagnate. This is a problem, as the loss functions should be constantly interacting to show the generator is still improving the images and producing a wide variety of samples. In order to mitigate this differing learning rates were tested which did lead to some improvements but eventually led to instability as shown by the loss diagram in figure 1. This was obtained with a generate learning rate of 0.001 and a discriminator learning rate of 0.0001.

2. CGAN

A conditional-GAN is similar to a normal DCGAN, except that it is also passed labels, meaning that it can represent it's understanding of the data generating process conditionally - instead of just telling whether an image is fake or not, the labels of the images (stating what number it is) are provided to the CGAN and thus the discriminator, which means that the discriminator will give out a a boolean value which states whether the image given is real and the number that has been provided in the label.

This means that when the network is trained, the generator can be asked to draw a specific number instead of having no control over the number like in a DCGAN. In this way, a CGAN provides a window for the model to express it's learned representation in terms of the data intrinsic to the model.

2.1. Architecture

The generator architecture consists of dense feedforward layers which process the generated noise concatenated with a one hot encoding of which number it is twice, once with 128 nodes and the next time with 784 nodes to draw the number in a 28 by 28 by 1 format. Each layers has a Leaky ReLU activation with alpha value 0.2, except for the output which has a tanh layer.

The discriminator architecture is a dense feedforward (CNN) based classifier. It has 2 such layers, which take as input a flattened 784 bit array concatenated with a y label that tells it what number it is meant to be. It will then process it at first with 128 nodes and then with 1 node. This outputs a positive value if it classifies the image as real and as the number given in as the label, and negative if it classifies it as fake or not the number given in as the label.

VBN and early stopping were again used as the regularisers, and the optimisers used for the generator and discriminator were both Adam optimisers. The loss function was sigmoid cross entropy.

2.2. Varying Architectures

A variety of architectures were tried, which will be detailed below.

Gen Arch	Disc Arch	Inception Score
128->784	128->1	76.2%
128->784	128->8->1	58.0%
128->512>784	128->1	52.0%
128->512>784	128->8->1	40.5%
128->256>784	128->8->1	57.2%
128->256>784	128->64->1	66.8%
128->256>784	128->32->1	72.9%

Table 1. Results for section 3 using different CGAN architectures to generate images. This was all done on a 100 epochs with both learning rates set to 0.002

Table 1 shows that adding layers seem to decrease performance. This is thought to be because adding layers makes the model more complex and since the batch size wasn't decreased to increase the regularisation, the model starts over-fitting and thus gives out a lower performance. When the discriminator layers were increased, the generator starts having worse performance and this is thought to be because of mode collapse.

However, when both are increased and the discriminator's layers are decreased, performance starts increasing again. At this point, with higher virtual batch normalisation added onto more layers will probably result in the best performance.

A convolutional version was also tried but convergence was even harder to achieve on that due to mode collapse and the gradient vanishing problem being much more prevalent on complex architectures, as well as the epoch time taking 10 to 15 times longer than the dense feedforward version. It was ultimately decided that the additional computational power and time required was not worth the gain in performance achieved.

2.3. Hyper-Parameters

Learning Rate

The learning rate was also varied to see the extent of its effect on the model. When the generator learning rate was weaker than the discriminator, the two loss functions would take a long time to interact and would then be unstable. Having trialled combinations of learning rates for both the discriminator and generator from the list [0.002, 0.0002, 0.00002] we found the best images were produced when the learning rate for both was 0.002 or the generator was slightly stronger with 0.002 when the discriminator rate was 0.0002.

The learning rate was also varied individually by choosing to the update either the generator or the discriminator model twice compared to the other model. The learning rate in this case was varied by doing two updates each of the generator and the discriminator instead of one at a time.

Updating the generator twice led to mode collapse multiple times, where the generator losses would get too high and the discriminator losses would get too low, and a variety of learning rates were also tried. When it did work, it led to marginally better performance but the model was prone to instability, so overall the more stable build was preferred.

Updating the discriminator twice, however, gave a much more stable model with better results (67.4% inception score with 200 epochs and a batch size of 100 using the same architecture as the last line on table 1). However, this result is still lower than the ideal performance that can be expected from the network when changing learning rates, as mentioned below in section 2.3.

This leads us to think that this effect is caused by the generator being much more prone to over-fitting rather than the discriminator, and when the discriminator catches on to a mistake after the generator has over-fit, mode collapse occurs and takes the entire model down.

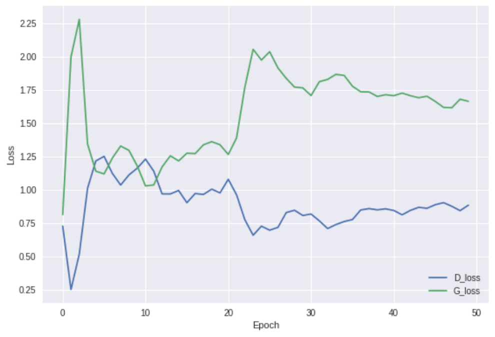

Figure 2. A sample loss function from the hyper-parameter optimisation of CGAN

Epochs

As was the case with the DCGAN, a higher number of epochs corresponds to higher performance but only as long as the GAN doesn't reach mode collapse, if mode collapse is reached then it will start undoing all the work it had previously done and start learning noise. Therefore, a careful balance has to be struck with epoch size to ensure the model has enough time to train but not enough time to reach mode collapse.

Batch-Size

As described in the DCGAN section, the lower batch-size allowed for more regularisation and clearer images as proven by the inception score for the batch-size of 25 of 82.6%, and the inception score decreasing as the batch-size increased.

Leaky ReLU Alpha

The leaky RELU allows for some negative values instead of zeroing out all negative values, and an alpha value of 0.2 was used after considerable research. Furthermore, LRELUs are used as the activation function because they seem to alleviate the gradient vanishing problem somewhat - with sigmoidal functions, if the loss gets too high then the gradient starts to level off and vanishes. To solve this, RELUs have an identity function for the positive side, which means the gradient isn't taken away. However, since the negative gradient can also be of use, a leaky RELU is used, which allows 0.2 of the negative gradient to come through, to further avoid the vanishing gradient.

2.4. Challenges and how they are addressed

Mode collapse was a common issue with many models and addressing it proved to be the most common challenge while doing this coursework. Mode collapse was addressed in this case by either choosing to train the generator or the

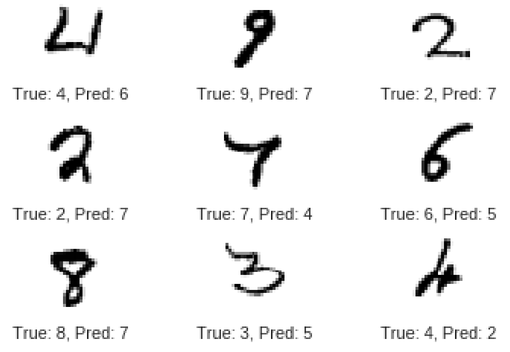

Figure 3. Incorrectly classified images from the real MNIST test set.

discriminator twice or in a different ratio. This is because the learning rates are far too sensitive to change, as found out from training the DCGAN. Increasing it too far would result in the model becoming unstable and lead to mode collapse in the other way, and making the learning rate too small leads to the gradient vanishing problem.

This is when the model can't make any changes because the learning rate is too low and thus the loss becomes straighter, and as the deep networks do gradient based learning, there won't be a gradient for it to learn from and thus the model will stop learning.

To combat this, the losses were monitored while the model was training and the model would be reset with differently tuned parameters to avoid mode collapse - this is often done via increasing the learning rate of the losing network and decreasing the learning rate of the winning network, or updating the losing network twice instead of just once per training iteration.

3. Inception Score

This is carried out with LeNet as provided.

When the classifier was tested on the MNIST test set, the classification accuracy was 94.2%. 10,000 images were generated by the optimised CGAN as described in the previous section, this gave an accuracy of 71.9% when used as the test set. This indicates the classifier is able to generalise well and that the generated images are of a quality comparable to the MNIST images.

Everytime the CGAN was optimised as described above, images were generated and tested with a LeNet classifier trained on the MNIST dataset was used to classify the generated images. The results are shown in table 1. These were all carried out multiple times and the results were averaged.

As expected, the lowest batch-size had the best performance as did the highest epochs. Too low a learning rate led to poor performance as the network would then require a higher number of epochs to train well. Figures 3 and 4 show examples of incorrectly classified images. Figure 3 is using the real MNIST test set, demonstrating the model struggles with extravagant digits. Figure 4 is using the test set generated with a batch-size of 25, and demonstrates that some of the produced images were poor quality.

Hyper-Parameter Changed	Inception Score
Batch-Size = 25	82.6%
Batch-Size = 50	69.6%
Batch-Size = 100	75.3%
Batch-Size = 200	45.7%
Batch-Size = 400	17.0%
Epochs = 10	14.1%
Epochs = 25	22.8%
Epochs = 50	59.1%
Epochs = 75	66.2%
Epochs = 100	77.8%
DLR = 0.00002, GLR = 0.00002	11.1%
DLR = 0.00002, GLR = 0.0002	13.6%
DLR = 0.00002, GLR = 0.002	0.00%
DLR = 0.0002, GLR = 0.00002	10.5%
DLR = 0.0002, GLR = 0.0002	65.1%
DLR = 0.0002, GLR = 0.002	74.0%
DLR = 0.002, GLR = 0.00002	9.6%
DLR = 0.002, GLR = 0.0002	70.4%
DLR = 0.002, GLR = 0.002	85.9%

Table 2. Results for section 3 using different CGAN hyper-parameters to generate images

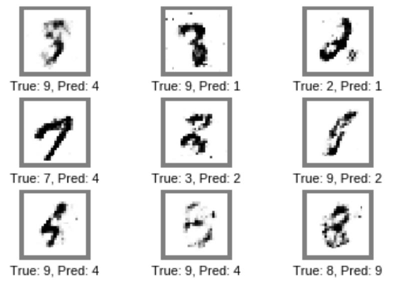

Figure 4. Incorrectly classified images from the generated images from the CGAN.

4. Retraining the handwritten digit classifier

The unoptimised CGAN was used to generate 55,000 images. These were mixed with the MNIST training images to form a training set composed of real and generated images as detailed in table 3. In fact, using a mix of 75% real data and 25% generated data produced a recognition accuracy of 92.3% on the MNIST test set of similar strength to the pure MNIST training set. When the classifier was retrained with images generated by an optimsed CGAN with a batch-size of 25, learning rates of 0.002, and an architecture of a generator with 3 layers going from 128 to 256 to 784 and a discriminator going from 128 to 32 to 1, which ran for 300 epochs, mixed with real data in a 25:75 synthetic to real ratio, the results were **97.7%**. This is significant as it exceeds the pure MNIST set accuracy. The full results for the optimised CGAN are detailed in table 4. Figure 5

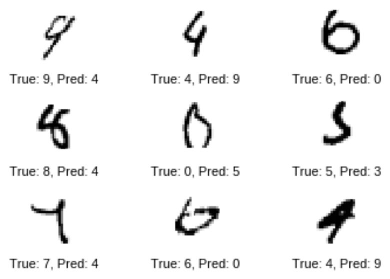

Figure 5. Incorrectly classified images from the generated images from the optimised CGAN.

% Real Images	% Generated Images	Inception Score
100	0	94.2%
75	25	92.3%
50	50	87.6%
25	75	65.8%
0	100	9.8%

Table 3. Results for section 4 using different splits of training data with the optimised CGAN

% Real Images	% Generated Images	Inception Score
100	0	94.2%
75	25	97.7%
50	50	97.1%
25	75	96.1%
0	100	9.8%

Table 4. Results for section 4 using different splits of training data with the unoptimised CGAN

In each case, the mix of synthetic and real data was used to initialise the network parameters, and parameters were fine-tuned with the real validation set. This meant the model was able to incorporate diversity into the training but was validated on images of similar form to the test set.

As table 4 shows, when training on the synthetic images from the optimsised CGAN, the performance of the classifier exceeds that of the pure MNIST set, but it still requires the MNIST set. We can therefore conclude that the images generated by the CGAN can be used to complement the real training set in order to create more training samples. This increase in training data will lead to better models. We can be confident this is not due to over-fitting as the generated images are not perfect, and therefore they add diversity to the training set.

5. Bonus Section

How do you compare GAN with PCA?

GANs are used to learn the generating process behind a set of given data, and as they are a true generative model they can also produce more of the same data after learning the process. Meanwhile, PCA learns a simplified version of the generating process, where only the principal components are concerned. GANs can in theory learn a process well enough that they can minimise losses while generating new data or to compress data, whereas PCA will always lose more information.

However, PCA is a much more stable and quicker model, and isn't a deep model so the computational resources and time required are much smaller than GANs need. GANs are usually deep networks and are very unstable without optimisation. This is because they have two networks that are playing a minmax game - a zero sum game where neither wins but each keeps improving the other. If one network were to win, then it would reach a state of stable equilibrium where nothing changes. GANs only work while they are in unstable equilibrium, and as such balancing them there can prove very difficult.

Furthermore, when the generator in a GAN is restricted to be linear (using linear activation functions on the neurons) and the discriminator is restricted to quadratic loss functions, an interesting result is observed. The outputs are very similar to that of PCA, which is a linear transformation. This leads us to the fact that a very basic GAN is just PCA but done on a deep network, and when non-linearities are added to it to make the GAN more complex, it can do so much more by representing a much wider range of functions. This is naturally where the differences (and similarities) between the two generative models stem from. [1] [2] [3] [4] [5]

References

[1] S. Feizi, F. Farnia, T. Ginart, and D. Tse. Understanding gans: the lqg setting. /10/30 2017.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[3] A. Odena. Semi-supervised learning with generative adversarial networks, Jun 5, 2016.

[4] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans, Jun 10, 2016.

[5] J. T. Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *ICLR 2016*, /11/19 2015.