Generating Images with a Generative Adversarial Network

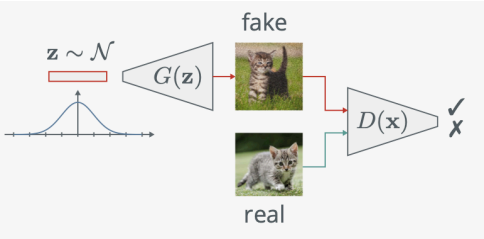
Anonymous author

Abstract

This paper proposes using a Generative Adversarial Network (GAN) to generate images. More specifically, a Deep Convolutional Generative Adversarial Network (DCGAN) is used, in conjunction with a few techniques and adjustments aimed at improving training stability and ultimately producing better images. The methodology describes the underpinning theory of Generative Adversarial Networks, as well as the relevant theory in relation to the use of convolutional layers and other techniques used within this method. Example generated images and interpolations are displayed, and some limitations of this model are also outlined.

1 Methodology

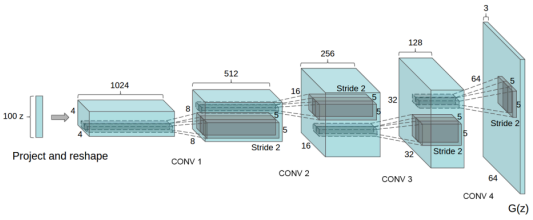
The method is to train a Generative Adversarial Network [1]. This is a non-cooperative zero-sum game where two networks, the generator, G, and the discriminator, D, compete against each other [8]:



The aim of G is to predict the distribution of the input data (denoted p_{data}) in order to generate new images from that distribution. Let z represent a latent space vector sampled from a Gaussian distribution. The generator network $G(z)$ therefore aims to map the latent space vector z to data-space. Let x represent data representing an image. The discriminator function $D(x)$ represents the probability that x is an image originally from the input data as opposed to having been generated by G. Thus, D attempts to maximise the probability, $logD(x)$, it correctly guesses which images are real and which are fake, whilst G attempts to minimise the probability $log(1 - D(G(z)))$ that D will correctly guess its generated images are indeed sourced from G. D and G therefore play this minimax game, which hypothetically converges to an equilibrium whereby G is generating perfect images such that D is incapable of distinguishing whether the image was originally from the input data or generated by G. The value function, $V(G, D)$, is as follows [1]:

$$\min_G \max_D V(D,G) = E_{x \sim p_{data}(x)}[logD(x)] + E_{z \sim p_z(z)}[log(1 - D(G(z)))]$$

Extending from this, the method implements a DCGAN, first introduced by Radford et. al. [6]. Convolutional layers are used in D, and convolutional-transpose layers in G, in order to achieve higher stability in training, and allow for training higher resolution and deeper generative models. Furthermore, having analysed the respective experimental findings described in the paper, the method adopts the rest of the family of architectural definitions too; namely, the use of batchnorm, the use of ReLU activation in G for all layers except for the output (which uses Tanh), and the use of LeakyReLU activation in D for all layers. These ensure congruent gradient flow which is vital for the learning processes of G and D. This decision was further backed up by the fact that the method initially had sparse gradients and this was negatively impacting the GAN stability.



The above diagram shows the architecture of how G maps a latent space vector z to data-space. It is initially projected to a small spatial extent convolutional representation with many feature maps. It is then converted into an image of the desired size (in this context, 64x64) via a series of fractionally-strided convolutions. D, on the other hand, takes a full size image as input, and processes it through a series of strided convolution layers, batch norm layers, and LeakyReLU activations, ultimately leading to a scalar probability determining whether or not the image is fake as output. The method uses a Binary Cross Entropy loss function to underpin D and G's learning, which includes the calculations for both log functions in the main value function mentioned previously. In the end, the initialised G had 3576704 parameters whereas D had 2765568 parameters. The method could have perhaps been tweaked further such that the number of parameters in each network are as close to being equal as possible. This would be a better design given the nature of the minimax game that the networks are trying to play.

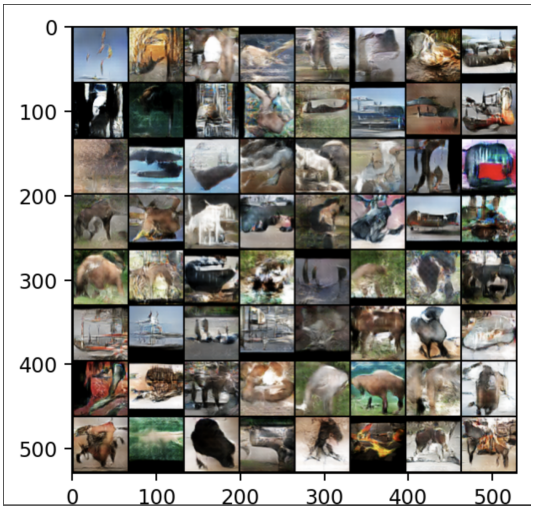
Any further best practices, hyper-parameter values or other associated initialisations presented by Radford et. al. were implemented into the method accordingly where sufficient justification juxtaposed with some independent research and/or experimentation deemed it appropriate. In particular, all model weights are randomly initialised from a standard normal distribution with standard deviation of 0.02. The selected optimiser was Adam, which appeared to be an obvious choice given its heavy establishment throughout research literature as the optimal choice in contexts similar to this [3]. Having experimented with a few different learning rates, it was decided to use 0.0002. When using anything larger than this, the loss function was experiencing diverging behaviour. The initial decay rate beta1 parameter for Adam was set to 0.5; outcomes similar to those experienced by Radford et. al. occurred when using a larger value for this whereby the training became unstable and somewhat oscillatory in nature.

The method includes a few more techniques to improve the model further, the first of which concerns the training of D, which receives separate batches of real and fake images. The method first forward passes a batch of purely real images through D, then calculates the gradients for D in a backward pass. Then, a batch of purely images generated by G is forward passed through D, followed by a backward pass, which this time calculates the accumulated gradients from each of the aforementioned batches. Only then is a step of D's optimiser called. D's loss is calculated as a sum of its loss on each of the two batches. The rationale behind this is to simplify training and ensure maximum effectiveness

in batch normalisation. Batch normalisation's purpose is to reduce internal covariance shift in activation maps, but especially near the start of training, real and fake images tend to have very different distributions. Therefore by separating the two into independent batches, the need for the neural network having to adapt to changes in the distributions between the two is avoided.

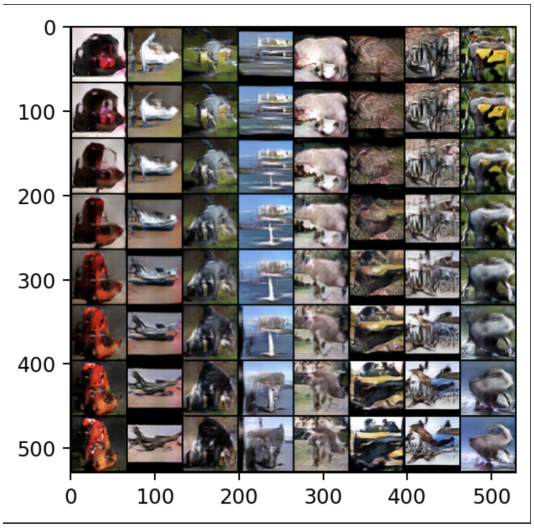
2 Results

Once the method had been running for enough epochs such that any further convergence was no longer particularly noticeable, a random batch of non-cherry picked generated samples looked like this:



The images are very clear and not blurry. Some of the generated images are strong, but several do lack realism. The model seems to have performed better in some image categories than others - for instance, the generated images of horses and ships were generally quite realistic, but the generated images of monkeys less so. By eye, the generated images appear to be diverse in nature from their nearest neighbours in the training data. However, it would be better if there was a more robust, numerical method of determining this, such as the use of a perceptual similarity metric, which could calculate the distance between the generated images and their respective nearest neighbours from the data set [10]. This was not implemented because it wasn't clear how each image's nearest neighbours in the STL-10 data set could be established in a computational manner in order to subsequently carry out the comparisons.

The next results show images generated by interpolating between points in the latent space. The interpolation method used was spherical linear interpolation. This method was selected because research undertaken by Tom White has shown it prevents divergence from a model's prior distribution and produces sharper samples [7]:



Finally, here are some cherry-picked samples that show some particularly strong outputs that the model has generated:

3 Limitations

It is hoped that G will produce a wide variety of strong outputs, but it could overfit to learn to produce only a single strong output that fools D. If G starts repeatedly producing the same output in this way, it is within D's best interests to learn to always reject this output. But if the following D iteration gets stuck in a local minimum and fails to find the optimal strategy, it will then become too easy for the next G iteration to produce the strongest output against the current D. Thus, each G iteration could over-optimise for a particular D, and D would never manage to learn its way out of this trap. This would ultimately lead to the Gs rotating through a small set of outputs - an issue referred to as mode collapse [2]. Judging by some of the generated samples across multiple epochs, there is evidence to suggest that the implemented method is suffering from this issue to some extent. Given mode collapse is an inherent issue with GAN models by definition, this is not surprising. To address this, the method could have benefited from the use of spectral normalisation [5]. This would need to replace the current batchnorm normalisation layers in D that sit between the convolutional layers and normalise the weights for every layer such that the Lipschitz constant for every layer is 1 [9]. This would ultimately have the effect of constraining D to moderate it and ensure it can capture all of the modes of the data set. The method could have also made use of the categorical class labels that are associated with the STL-10 data set, perhaps by passing in the label as an additional parameter to G - which would make it a conditional GAN [4]. This could have helped produce better images for categories such as monkeys where the generated images from the implemented method were much worse, because the GAN would be conditioned on tailored learning for each individual class.

Bonuses

This submission has a total bonus of -2 marks (a penalty) because: -4 marks (as its a GAN), then 1+1 = +2 marks for STL-10 at 64x64 resolution.

References

[1] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* (2014), pp. 2672–2680.

[2] Bhagyashree, Vandana Kushwaha, and G. C. Nandi. "Study of Prevention of Mode Collapse in Generative Adversarial Network (GAN)". In: *10.1109/CICT51604.2020.9312049* (2020).

[3] Sebastian Bock and Martin Weiß. "A Proof of Local Convergence for the Adam Optimizer". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8852239.

[4] Mehdi Mirza and Simon Osindero. "Conditional Generative Adversarial Nets". In: *arXiv e-prints*, arXiv:1411.1784 (Nov. 2014), arXiv:1411.1784. arXiv: 1411.1784 [cs.LG].

[5] Takeru Miyato et al. "Spectral Normalization for Generative Adversarial Networks". In: *arXiv:1802.05957* (2018).

[6] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: *arXiv preprint arXiv:1511.06434v2* (2016).

[7] Tom White. "Sampling Generative Networks". In: *arXiv:1609.04468* (2016).

[8] Chris G Willcocks. *Adversarial Models*. 2019. URL: https://cwkx.github.io/data/teaching/deep-learning/dl-lecture4.pdf.

[9] G. R. Wood. "Estimation of the Lipschitz constant of a function". In: *J Glob Optim 8, 91–103* (1996).

[10] Richard Zhang et al. "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric". In: *CVPR*. 2018.