2 GAN

I've been working on a small project involving Generative Adversarial Networks (GANs) [1]. A GAN is a machine learning framework designed to generate new data that resembles a given dataset. It consists of two neural networks: a Generator G and a Discriminator D, which are trained in competition with one another.

The Generator takes a random noise vector $z \sim p_z(z)$ as input and produces a 64×64 image, $G(z)$. The Discriminator receives an image (either real from the dataset or generated by G) and outputs a probability $D(x) \in [0, 1]$ indicating whether the image is real (from the dataset) or fake (produced by the Generator).

Training alternates between improving the Discriminator and the Generator. The Discriminator is trained to maximize the probability of assigning the correct label to both real and fake images. Its loss function is [1]:

$$\mathcal{L}_D = -\left[\log D(x_{\text{real}}) + \log(1 - D(G(z)))\right]$$

Here, x_{real} is a real image from the dataset, and $G(z)$ is a fake image produced from noise.

The Generator, on the other hand, tries to fool the Discriminator by producing images that it classifies as real. Its loss function is:

$$\mathcal{L}_G = -\log D(G(z))$$

Training proceeds iteratively. First, I sample a random noise vector, generate a fake image, and evaluate it with the Discriminator to compute the `loss_fake`. Then, I pass a real image through the Discriminator to compute the `loss_real`. The total Discriminator loss is the sum of these two. After updating D, I sample new noise, generate another fake image, and compute the Generator loss by treating the fake image as if it were real, encouraging the Generator to improve.

Over time, this adversarial process improves both models. The Discriminator becomes better at distinguishing real from fake, and the Generator becomes better at producing realistic images. Eventually, the Generator can produce samples that are nearly indistinguishable

[1]Note that for a lot of these equations and mathematical jargon, I am simply including it for my own reference in case I decide to reproduce this later. Feel free to skim over such details, as it and will not jeopardize your understanding

from the real dataset, and we use it to generate new images.

Now, let's see what this is like in practice. I have a dataset of 200k celebrity faces [2], and I have a deep convolutional model for both the Generator and Discriminator. This is therefore called a DCGAN (Deep Convolutional GAN) [3]. Here are the architectures:

DCGAN Generator Architecture

Layer	**In Ch.**	**Out Ch.**	**Kernel / Stride / Pad**	**Activation**
ConvTranspose2d	100	512	4 / 1 / 0	ReLU
BatchNorm2d	–	512	–	–
ConvTranspose2d	512	256	4 / 2 / 1	ReLU
BatchNorm2d	–	256	–	–
ConvTranspose2d	256	128	4 / 2 / 1	ReLU
BatchNorm2d	–	128	–	–
ConvTranspose2d	128	64	4 / 2 / 1	ReLU
BatchNorm2d	–	64	–	–
ConvTranspose2d	64	3	4 / 2 / 1	Tanh

DCGAN Discriminator Architecture

Layer	**In Ch.**	**Out Ch.**	**Kernel / Stride / Pad**	**Activation**
Conv2d	3	64	4 / 2 / 1	LeakyReLU (0.2)
Conv2d	64	128	4 / 2 / 1	LeakyReLU (0.2)
BatchNorm2d	–	128	–	–
Conv2d	128	256	4 / 2 / 1	LeakyReLU (0.2)
BatchNorm2d	–	256	–	–
Conv2d	256	512	4 / 2 / 1	LeakyReLU (0.2)
BatchNorm2d	–	512	–	–
Conv2d	512	1	4 / 1 / 0	Sigmoid

As mentioned before, I used the Binary Cross-Entropy (BCE) loss function to train both the generator and discriminator. This is defined as:

$$\mathcal{L}_{\text{BCE}}(p, y) = -\left[y \log(p) + (1 - y) \log(1 - p)\right]$$

where $p \in [0, 1]$ is the predicted probability and $y \in \{0, 1\}$ is the ground truth label.

I used the Adam optimizer for both the generator and the discriminator, with learning rate $\alpha = 0.0002$ and momentum parameters $\beta_1 = 0.5$, $\beta_2 = 0.999$. These are commonly used hyperparameters in GAN training.

I ended up training the models with a batch size of 128 across 20 epochs. It took quite a while, but after 15 epochs, here are the results:

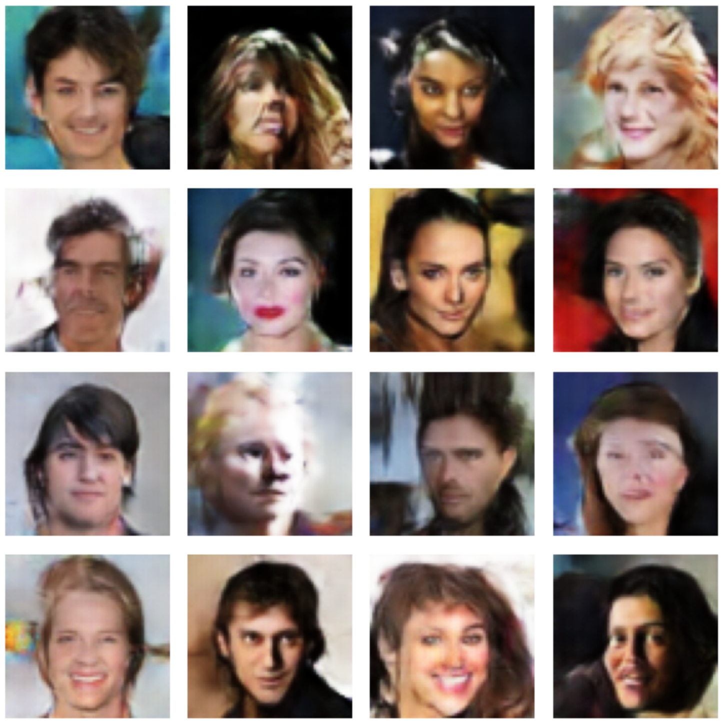

Figure 1: Generator outputs using 16 different noisy inputs after 15 epochs

The generated faces are starting to form. The model is slowly learning how to create features like eyes, hair, facial structure, jaw, etc. I think at this point, I had a Discriminator loss of 0.3211 and a Generator loss of 3.1851, which is pretty standard. Ideally, these losses should keep fluctuating. One model cannot win (because that spoils the adversarial part of the training). Interestingly, however, after the full 20 epochs, we get the following:

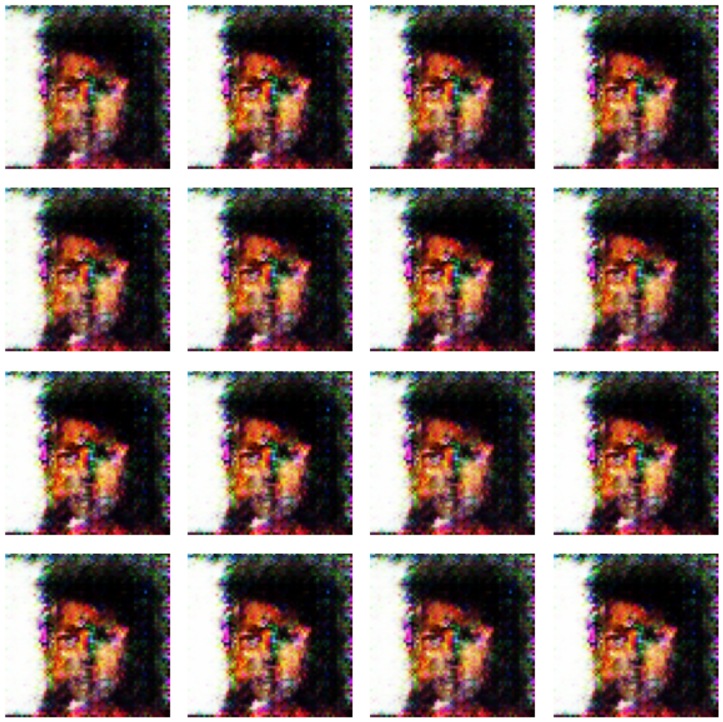

Figure 2: Generator outputs using 16 different noisy inputs after 20 epochs

Upon inspecting the losses, we have a Discriminator loss of 0.000 and a Generator loss of 64.0911. It seems like the Generator decided to output this one weird image all the time, and the Discriminator was not fooled. This is known as *mode collapse*. This happens when the Generator finds one output that consistently fools the Discriminator (until it doesn't anymore) [4]. I will try to figure out how to fix this later.

References

[1] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: `1406.2661 [stat.ML]`. URL: `https://arxiv.org/abs/1406.2661`.

[2] Ziwei Liu et al. "Deep Learning Face Attributes in the Wild". In: *Proceedings of International Conference on Computer Vision (ICCV)*. Dec. 2015.

[3] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: `1511.06434 [cs.LG]`. URL: `https://arxiv.org/abs/1511.06434`.

[4] Tim Salimans et al. *Improved Techniques for Training GANs*. 2016. arXiv: `1606.03498 [cs.LG]`. URL: `https://arxiv.org/abs/1606.03498`.