Implementation of Generative Adversarial Network (GAN) on Anime Faces

ECE 592: Topics in Data Science

Nishanth Vimalesh-[200266367]
Rahul Mishra- [200267922]

Content

1. Motivation:

Our main motivation of doing this project was educational. We are new to this computer vision field and wanted to get hands-on experience of building artificial neural network, tuning the hyper-parameter and use it for classification. Further, we want to explore the field of generative models which has various applications in modern deep learning world such as 1) Generating a model in Reinforcement learning, 2) Training on a semi- supervised dataset, 3) Generation of multi-modal outputs, 3) Generation of realistic data.

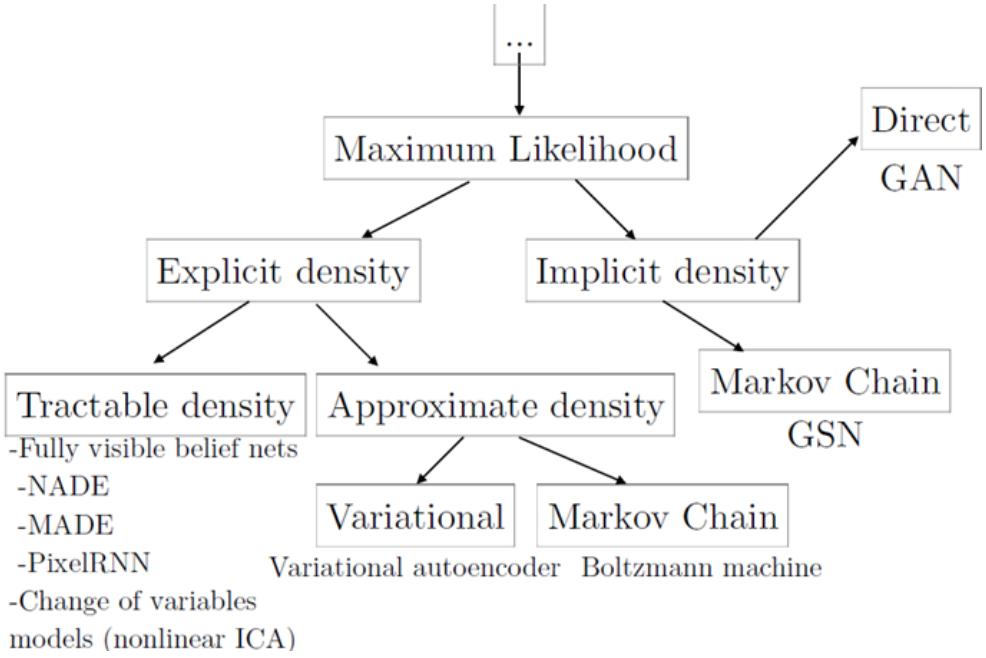
2. Problem Statement:

Artificially generate anime faces by learning the distribution of the dataset using Generative Adversarial Network (GAN).

Dataset: Moe Imouto Faces (14387 Anime Faces)

3. Introduction to Generative Models:

Generative Models refers to any model that takes a training set, consisting of samples drawn from a distribution p_{data}, and learns to represent an estimate of that distribution. The result is a probability distribution p_{model}. Therefore, generative models aim at learning the true data distribution of the training set to generate new data points with some variations. But it is not always possible to learn the exact distribution of our data and so we try to model a distribution which is as similar as possible to the true data distribution.



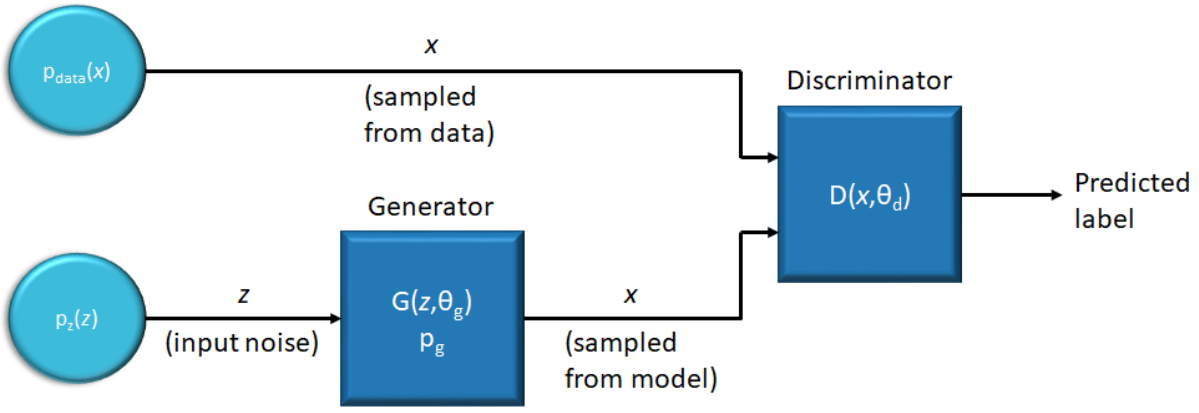
Deep generative models that can learn via the principle of maximum likelihood differ with respect to how they represent or approximate the likelihood. On the left branch of this taxonomic tree, models construct

an explicit density, pmodel(x; θ), and thus an explicit likelihood which can be maximized. Among these explicit density models, the density may be computationally tractable, or it may be intractable, meaning that to maximize the likelihood it is necessary to make either variational approximations or Monte Carlo approximations (or both). On the right branch of the tree, the model does not explicitly represent a probability distribution over the space where the data lies. Instead, the model provides some way of interacting less directly with this probability distribution. Typically, the indirect means of interacting with the probability distribution is the ability to draw samples from it. Some of these implicit models that offer the ability to sample from the distribution do so using a Markov Chain; the model defines a way to stochastically transform an existing sample in order to obtain another sample from the same distribution. Others are able to generate a sample in a single step, starting without any input. While the models used for GANs can sometimes be constructed to define an explicit density, the training algorithm for GANs makes use only of the model's ability to generate samples. GANs are thus trained using the strategy from the rightmost leaf of the tree: using an implicit model that samples directly from the distribution represented by the model.

4. Generative Adversarial Networks (GANs):

In the case of GANs, the generative model is pitted against an adversary: a discriminative model that learns to determine whether a sample is from the model distribution or the data distribution. The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles.

5. Framework of GAN:



The generator's distribution p_g is learnt over data x by defining a prior on input noise variables $p_z(z)$. This is then mapped to the data space as $G(z; \theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters θ_g. Another multilayer perceptron $D(x; \theta_d)$ that outputs a single scalar is defined. D(x) represents the probability that x came from the data rather than p_g. D is trained to maximize the probability of assigning the correct label to both training examples and samples from G. G is simultaneously trained to minimize log(1 −

D(G(z))). Therefore, D and G play a two-player minimax game with the value function V (G, D) given as:

$$\textit{Value Function:} \quad \underbrace{\boldsymbol{min}}_{\boldsymbol{G}} \underbrace{\boldsymbol{max}}_{\boldsymbol{D}} \boldsymbol{V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1 - D(G(z)))]}$$

6. Theory:

D and G play the following two-player minimax game with value function:

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

Training criteria for the discriminator D, given any generator G, is to maximize V(G,D)

$$V(G,D) = \int_x p_{data}(x)\log\big(D(x)\big)\,dx + \int_z p_z(z)\log\big(1 - D(G(z))\big)\,dz$$

$$= \int_x [\, p_{data}(x)\log\big(D(x)\big) + p_g(x)\log\big(1 - D(x)\big)\,]\,dx$$

For any (a,b) $\in$ $R^2 \setminus \{0,0\}$, the function y → a log(y) + b log(1-y) achieves its maximum in [0,1] at $\frac{a}{a+b}$.

The discriminator does not need to be defined outside of Supp(p_{data}) ∪ Supp(p_g).

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

Training objective for D can be interpreted as maximizing the log-likelihood for estimating the conditional probability P(Y=*y*|*x*), where Y indicates whether x comes from p_{data} (with y=1) or p_g (y=0).

Virtual training criteria C(G):

$$\begin{aligned} C(G) = \min_G V(G,D) \quad &= E_{x\sim p_{data}}[\log D_G^*(x)] \\ &+ E_{z\sim p_z}\left[\log\left(1 - D_G^*\big(G(z)\big)\right)\right] \quad = E_{x\sim p_{data}}[\log D_G^*(x)] \\ &+ E_{x\sim p_g}\left[\log\left(1 - D_G^*(x)\right)\right] \quad = E_{x\sim p_{data}}\left[\log\frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\right] \\ &+ E_{z\sim p_z}\left[\log\frac{p_g(x)}{p_{data}(x) + p_g(x)}\right] \\ &= -\log(4) + KL\left(p_{data}\,||\,\tfrac{p_{data}+p_g}{2}\right) + KL\left(p_g\,||\,\tfrac{p_{data}+p_g}{2}\right) \\ &= -\log(4) + 2.JSD(p_{data}||p_g) \end{aligned}$$

Jensen-Shannon Divergence between two distribution is always non-negative, and zero iff they are equal.

C* = -log(4) is the global minima of C(G)

Only solution is $p_g = p_{data}$, i.e., generative model perfectly replicating the data distribution.

7. Algorithm:

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

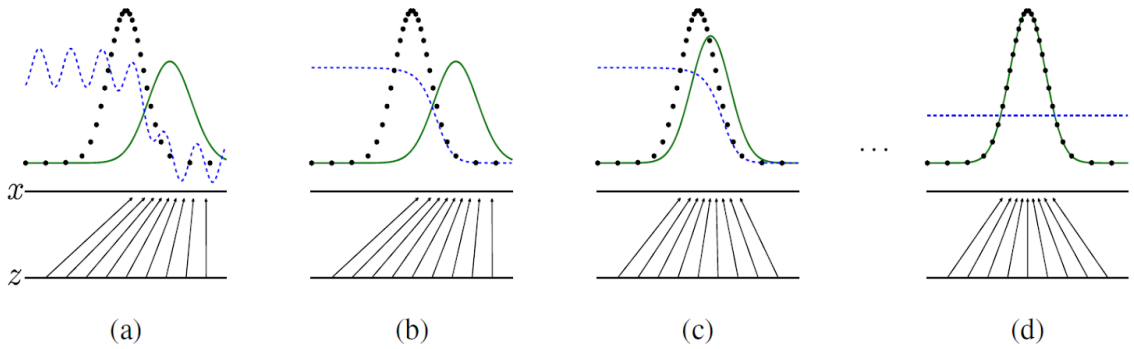
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

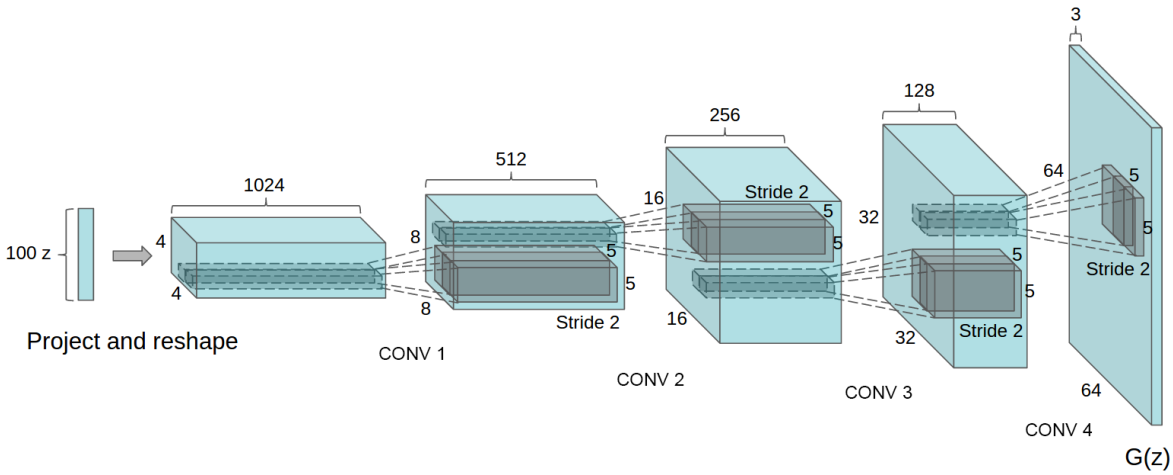
Model representation close to convergence:



(a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D(x) = p_{data}(x)/(p_{data}(x)+p_g(x))$. (c) After an update to G, gradient of D has guided G(z) to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{data}$. The discriminator is unable to differentiate between the two distributions, i.e. D(x) = 1/2.

8. Implementation:

Deep Convolutional Generative Adversarial Networks:



Methods used while building the architecture of DCGAN:

1. Discriminator - Strided Convolution,
2. Generator – Fractional-strided Convolution
3. Batch-Normalization
4. Remove fully connected layers
5. Generator: Rectified Linear Units (ReLU) activation for hidden layer, Tanh activation for output.

6. Discriminator: Leaky ReLU

Steps involved while training the network:

1. Training Data: Resized to shape 64x64; Normalized;
2. Generator Input: 4096 samples from N(0,1) distribution for each image
3. One Epoch: 500 samples from data and 500 samples from model to train Discriminator
4. Add noise to some random image labels.
5. First, we train the discriminator in 1 epoch.
6. Thereafter, we freeze the discriminator weights and only train the generator.
7. Steps from 2 to 6 is repeated for more than 30000 epochs. Once the adversarial loss becomes low and we see no change, we can stop and can conclude that it can be trained no further.

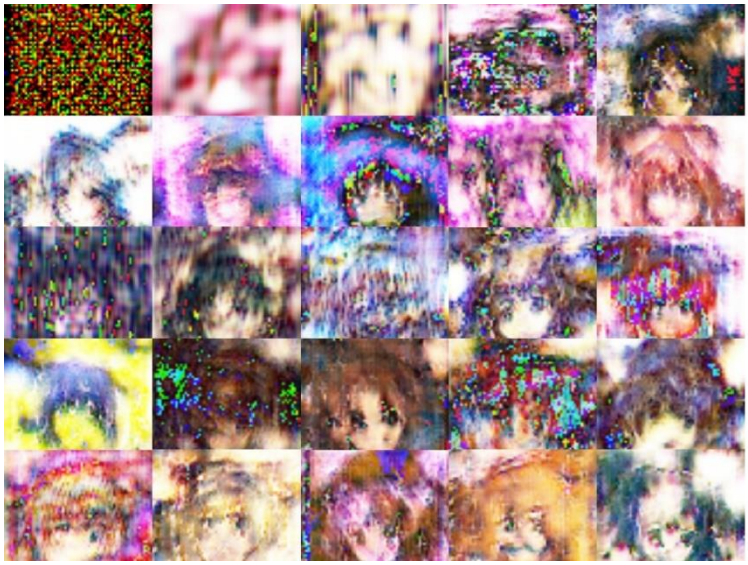
Additional Information about implementation:

1. Library: Keras with Tensorflow-GPU
2. GPU: NVIDIA GeForce GTX1050 Ti
3. Time/ Epoch ≈ 850 mS
4. Epochs: 40000
5. Samples/Epoch: 1000
6. Batch Size = 20

9. Results:

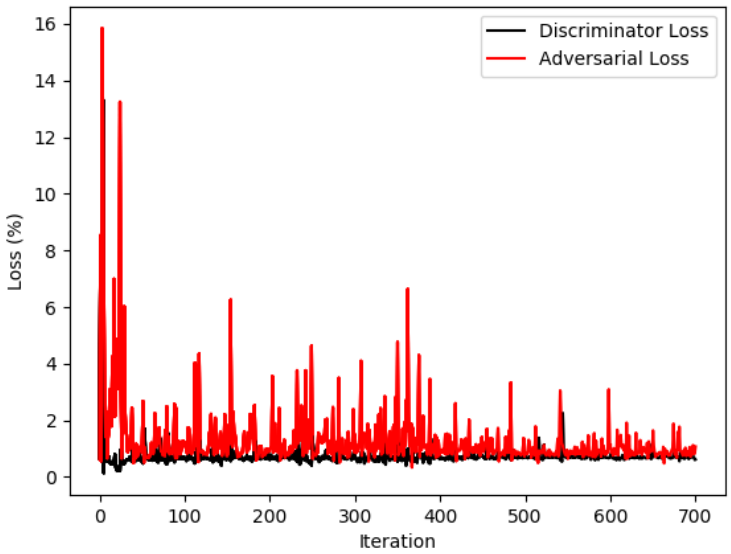
Our dataset of 64x64 images looks as shown below.

For the first few epochs, generator is too naïve and can not produce good results. All the outputs it generates is merely a noise. Examples of some of its initial outputs are shown below.



All outputs are 64x64 pixel images. These images don't make any sense to us. Discriminator can easily identify them from the real data. We keep training the generator to produce more realistic data. After some 40000 epochs this is what we get.

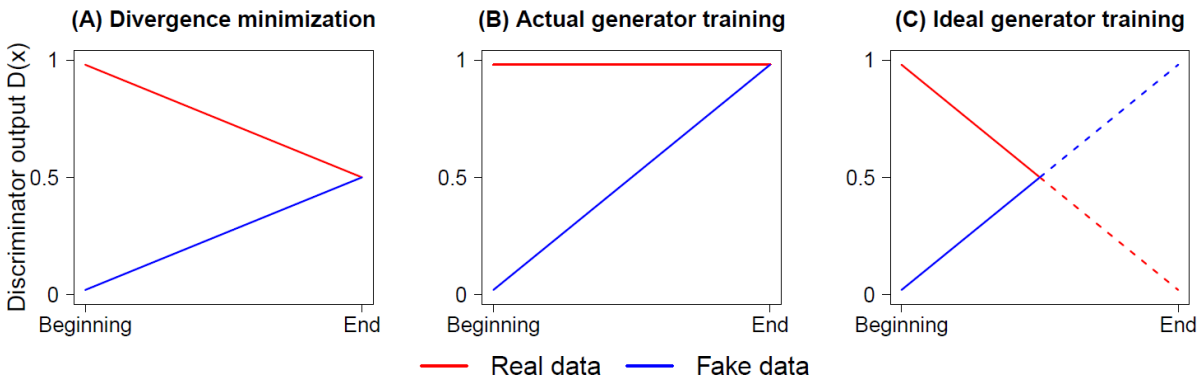
The losses in the initial iterations went down and later kept oscillating due to model being not stable and not being given the enough epochs.



The GAN output has terrific improvement over the initial epochs. Now, the faces it's trying to generate have begun to look similar to the data, though with some artifacts. Out of nowhere, noises appear in the image, or the face, eyes or hair are not complete.

10. Future Works and Improvements:

- Need high resolution images. Better to train on.
- ***Value Function:*** $\underbrace{\boldsymbol{min}}_{\boldsymbol{G}}\ \underbrace{\boldsymbol{max}}_{\boldsymbol{D}}\boldsymbol{V(D,G)} = \boldsymbol{E_{x\sim p_{data}(x)}[logD(x)]} + \boldsymbol{E_{z\sim p_z(z)}[log(1 - D(G(z)))]}$
- Our model is not complete. A paper on Relativistic GAN (RGAN) came out in July 2018 which points out that the above value function is not being modeled properly.
- Whereas we are training G in such a manner that probability of fake data considered as real increases, we are totally ignoring the other half where we have to reduce the probability of real data being considered real. Therefore, we have to modify loss function of G and D to account for the relative loss.



After implementing these improvements, we expect better performance from the generator network.

11. References:

1. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 27, pages 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

2. Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434, 2015.

3. Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems 29, pages 2234–2242. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6125improved-techniques-for-training-gans.pdf.

4. I. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. arXiv preprint arXiv:1701.00160, 2016.

5. A. Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan. arXiv preprint arXiv:1807.00734, 2018.