IFT 6756 - Lecture 7
Generative Adversarial Networks

February 9, 2021

This version of the notes has not yet been thoroughly checked. Please report any bugs to the scribes or instructor.

Scribes: William Neveu, Olivier Tessier-Larivière, Tianyu Zhang **Instructor:** Gauthier Gidel

1 Summary

In this lecture we introduce generative models, and in particular the use of generative adversarial networks (GANs). In order to paint a big picture of generative models, we will first show how the log-likelihood is used to evaluate generated data but fails in high dimension. This motivates the use of implicit distribution techniques, which GANs are part of. We will go into more details on the GAN objective function, as well as introduce GAN extensions, such as conditional GAN.

2 Supervised and Unsupervised learning

As we already know, supervised learning refers to techniques that learn to predict an output label y, given a new and unlabeled input x. They learn from a set of labeled observations $(x_i, y_i),\ i = \{1, ..., n\}$.

On the other hand, unsupervised learning does not have access to labels y_i. Instead its objective can be :

- Clustering
- Dimensionality reduction
- Feature learning
- Density estimation

3 Generative Modeling

Generative modeling is a form of unsupervised learning. Its goal is to estimate the data distribution. Figure 1 shows an example of density estimation. For lower dimensions, the Maximum Likelihood Estimation is an acceptable approach. However, it suffers from the curse of dimensionality.

Definition 1 (Maximum Likelihood Estimation)**.**

$$\max_{\theta} \prod_{i=1}^{n} p_\theta(x_i)$$

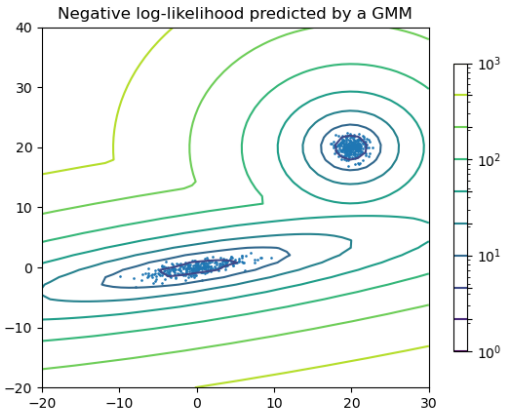

Figure 1: Negative log-likelihood predicted by a Gaussian Mixture Model (GMM). Image from *scikit-learn.org*.

3.1 Log-Likelihood

In practice, we maximize the log-likelihood instead of the likelihood. Since $\log$ is monotonically increasing, maximising the likelihood is equivalent to maximizing the log-likelihood. As we can see in definition 2, the maximum log-likelihood contains a sum instead of a product because of the properties of the log. This simplify the optimization because be can now maximize each term individually.

Definition 2 (Maximum log-likelihood)**.**

$$\arg\max_{\theta} \prod_{i=1}^{n} p_\theta(x_i)$$

$$= \arg\max_{\theta} \sum_{i=1}^{n} \log p_\theta(x_i)$$

Generating realistic samples is surprisingly not necessary to obtain a large log-likelihood [8, 9]. Assume p_{data} corresponds to a density model that is very good (e.g. gets a high likelihood), while q corresponds to a bad model (e.g. noise). At example 3, we define the mixture model p_θ and show that it has a good log-likelihood although it is sampling from the bad model 99% of the time.

Example 3 (Large Log-Likelihood but Poor Samples [8])**.**

$$\begin{aligned} p_\theta(\boldsymbol{x}) &= 0.01 p_{data}(\boldsymbol{x}) + 0.99 q(\boldsymbol{x}) \\ \log p_\theta(\boldsymbol{x}) &= \log\left[0.01 p_{data}(\boldsymbol{x}) + 0.99 q(\boldsymbol{x})\right] \\ &\geq \log\left[0.01 p_{data}(\boldsymbol{x})\right] \\ &\geq \log p_{data}(\boldsymbol{x}) - \log 100 \end{aligned}$$

$\log p_{data}$ is proportional to the dimensionality of **x** while $\log 100$ stays constant. For high-dimensional data such as images, $\log 100$ is negligible [7].

Now, let's show that the log-likelihood scales with d. However, let us first define what it is for a function to be L-Lipschitz continuous with respect to the ℓ_∞ norm.

Definition 4 (L-Lipschitz continuous)**.** *For a function f to be L-Lipschitz continuous over a space $\mathcal{X}$ with respect to the ℓ_∞ norm, f must satisfy :*

$$|f(x) - f(y)| \leqslant L \left|\left|x - y\right|\right|_\infty$$

for all $x, y \in \mathcal{X}$.

Proposition 5 (Log-Likelihood Scales with d)**.** *Under the assumption that $\mathcal{X}$ is $[0, 255]^d$ (for instance images with integer pixel values), and that the density $\log p(x)$ (p can either be p_{data} or p_θ) is L-Lipschitz with respect to the ℓ_∞ norm we have that,*

$$\log(p(x)) \leq L - d\log(2) \tag{1}$$

Proof. The idea is very, simple. Since p is L-Lipschitz continuous we have that for all image x' such that $|x_i - x'_i| \leq 1$, $i \in \{1, \dots, d\}$,

$$\log(p(x)) - L \leq \log(p(x')) \iff p(x)e^{-L} \leq p(x') \tag{2}$$

and thus since $\mathcal{X}$ is discrete we have

$$e^{-L} \sum_{x' \,,\, \|x-x'\|_\infty \leq 1} p(x) \leq \sum_{x' \,,\, \|x-x'\|_\infty \leq 1} p(x') \leq \sum_{x \in \mathcal{X}} p(x) \leq 1 \tag{3}$$

Which leads to (since there at least 2^d element in $B(x, 1) := \{x' \in [0, 255]^d \,,\, \|x - x'\|_\infty \leq 1\}$),[1]

$$e^{-L}2^d p(x) \leq 1 \iff \log p(x) \leq L - d\log(2) \tag{4}$$

□

The L-lipschitz assumption on $x \mapsto \log p(x)$ comes from the intuition we have with adversarial examples: for CIFAR images with pixel values in $[0, 255]$ changing the pixel values of ± 1 of a given image gives a new image that visually looks the same and thus has roughly the same likelihood ('roughly the same' is formalized with the Lipchitz assumption as 'up to a factor' e^L).
As shown in equation 10 of [8], it is also possible to have a poor Log-Likelihood while having great looking samples.

Example 6 (Poor Log-Likelihood and Great Looking Samples [8])**.** *Consider the following mixture of Gaussian.*

$$p_\theta(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{N}(\boldsymbol{x}; \boldsymbol{x}_i^{train}, \delta I_d)$$

Let us assume that it has memorized the training set. As the Gaussians are more and more centered on each example of the training set, $\delta \to 0$ and thus $p_\theta(\boldsymbol{x}) \to 0$. This means that the log-likelihood goes to $-\infty$.

As seen in examples 3 and 6, the log-likelihood is not sufficient to evaluate generative models in high dimensions.

4 Taxonomy of Generative models

As we can see in Fig. 2, generative models are based on maximum likelihood, but they differ when it comes to explicit or implicit density. In the former case, we try to to model and compute the density $P_\theta(x)$. In the latter, we never actually have the value the data distribution of $P_\theta(x)$, but are able to sample from it, as is explained in section 4.2.

4.1 Standard Technique using Explicit Density

$$p_{model}(\boldsymbol{x}) = p_{model}(x_1)\prod_{i=2}^{d} p_{model}(x_i|x_1, ..., x_{i-1})$$

The idea behind the standard technique using explicit density is to generate features one by one. First, generate x_1. Then knowing x_1, generate x_2. Next, knowing x_1 and x_2, generate x_3 and so on. Although the generated data can be good (see PixelRNN [10] and PixelCNN [11] (figure 3)) this technique comes at a cost :

- Algorithms are $O(d)$ (very slow).
- It is not known what is the best order for parameter generation. They may be hand-picked.
- There is no latent space.

[1] Note that most of the time there is 3^d elements in $B(x, 1)$. There is less element only when x has pixel with extremal values: 0 or 255

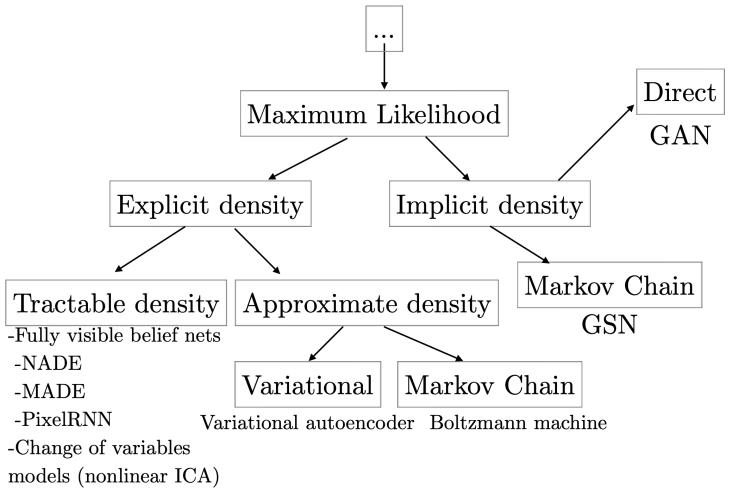

Figure 2: Taxonomy of Generative Models. Image reproduced from [2].

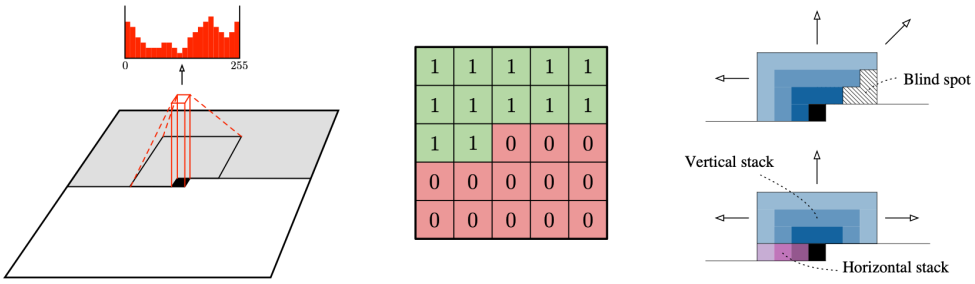

Figure 3: **Left**: A visualization of the PixelCNN that maps a neighborhood of pixels to prediction for the next pixel. To generate pixel x_i the model can only condition on the previously generated pixels $x_1, ...x_{i1}$. **Middle**: an example matrix that is used to mask the 5x5 filters to make sure the model cannot read pixels below (or strictly to the right) of the current pixel to make its predictions. **Right**: Top: PixelCNNs have a blind spot in the receptive field that can not be used to make predictions. Bottom: Two convolutional stacks (blue and purple) allow to capture the whole receptive field. Figure and description reproduced from [11]

4.2 Implicit density

$$x \sim p_\theta \iff \boldsymbol{x} = g_\theta(\boldsymbol{z}), \boldsymbol{z} \sim p_z$$

Instead of having to model the actual distribution of generated data p_θ in order to sample from it, implicit density techniques propose to sample from an easy to sample distributions p_z, such as a Gaussian. This sample, z is then transformed by a generator function g_θ, into an example x belonging to the data distribution p_θ.

5 GAN objective function

In GANs, we have both a generator and a discriminator fighting over the following binary cross entropy classification task. As we can see, the generator wants to minimize it, fooling the discriminator with realistic data. Meanwhile, the discriminator wants to maximize the objective, separating real from generated samples. We can easily see from the mathematical formulation that this task is in fact a zero-sum game between the generator and the discriminator. Fooling the discriminator is a win for the generator and a loss for the discriminator, and vice-versa.

Definition 7 (GAN objective function)**.**

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log(D(x))] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$$

Definition 8 (Payoff function)**.**

$$\varphi(D, G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log(D(x))] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$$

In order to prove that $\varphi(D, G)$ has a Nash Equilibrium, we need to prove the following property:

$$\varphi(D, G^*) \leq \varphi(D^*, G^*) \leq \varphi(D^*, G)$$

Proof.

First, let's show that

$$\varphi(D, G^*) \leq \varphi(D^*, G^*)$$

where $G^* |\ p_g = p_{data}$, and $D^*(\boldsymbol{x}) = \frac{1}{2}\ \forall \boldsymbol{x}$, as the best discriminator can only do a coin-flip when $G = G^*$

$$\begin{aligned}
\varphi(D, G^*) &= \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D(\boldsymbol{x}))] \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log(1 - D(\boldsymbol{x}))] \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log(D(\boldsymbol{x})) + \log(1 - D(\boldsymbol{x}))] \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log(D(\boldsymbol{x})(1 - D(\boldsymbol{x}))] \\
&\leq \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log(D^*(\boldsymbol{x})(1 - D^*(\boldsymbol{x}))]
\end{aligned}$$

Substituting $D^*(\boldsymbol{x}) = \frac{1}{2}$

$$\begin{aligned}
&\leq \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log(\frac{1}{2}(1 - \frac{1}{2}))] \\
&\leq \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log(\frac{1}{4})] \\
&\leq \log(\frac{1}{4}) \\
&\leq \varphi(D^*, G^*)
\end{aligned}$$

Next, let's show that

$$\varphi(D^*, G^*) \leq \varphi(D^*, G)$$

$$\varphi(D^*, G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log(D^*(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D^*(\boldsymbol{x}))]$$

Substituting $D^*(\boldsymbol{x}) = \frac{1}{2}$

$$\begin{aligned}
&= \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log(\frac{1}{2})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(\frac{1}{2})] \\
&= \log(\frac{1}{4})\ \forall G
\end{aligned}$$

Therefore, we have

$$\begin{aligned}
&\varphi(D, G^*) \leq \varphi(D^*, G^*) = \varphi(D^*, G) \\
&\varphi(D, G^*) \leq \varphi(D^*, G^*) \leq \varphi(D^*, G)
\end{aligned}$$

□

6 GAN: a Trivial Game ?

Exactly memorizing the train set seems optimal for the generator because the true data comes from the train sets. So if the fake data also comes from the train sets, the discriminator has no way to distinguish the two. Why this does not happen? It's because the generator never truly touch the training sets. (The target problem of the generator is shown

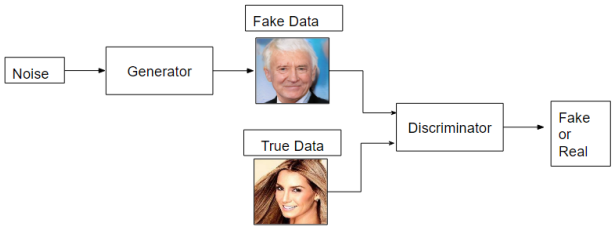

Figure 4: A simple process of the GAN train.

below. It only receives outputs from the discriminator.) It only receives information from the discriminator which only tells *yes* or *no*. It hard even for human to reproduce what the discriminators have in mind. Thus, the generator can generate something close to the training sets but not exact training sets because it is never able to access the training sets.

$$G^* = \arg\min_G \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$$

Overall it does not completely explain why generator usually do not generate pictures from the train set but it give the intuition on why copying the training set is roughly as hard as to generate new images for the generator.

7 Non-Zero Sum Game

Definition 9 (Zero Sum Formulation)**.**

$$\min_G \max_D \mathbb{E}_{x\sim p_{data}}[\log(D(x))] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$$

Definition 10 (Non-zero Sum Formulation)**.**

$$\min_D -\mathbb{E}_{x\sim p_{data}}[\log(D(x))] - \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$$
$$\min_G -\mathbb{E}_{z\sim p_z}[\log(D(G(z))]$$

For the generator in the zero sum formulation, the first term ,$\mathbb{E}_{x\sim p_{\text{data}}}[\log(D(x))]$, actually does not contribute to the gradient of minimizing G. Thus, $G^* = \arg\min_G \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$.

In the case of non-zero sum formulation, we changed the above target to $G^* = \arg\min_G -\mathbb{E}_{z\sim p_z}[\log(D(G(z))]$ because it gives stronger gradient at early learning phases. To explain this specifically, the discriminator always started from showing False to the generator, i.e. $D(G(z))$ should be around 0 at first. However, in the zero sum formulation, the target function is around $\log(1 - D(G(z))), D(G(z)) \approx 0$. In the non-zero sum formulation, on the contrary, the target function is around $-\log(D(G(z))), D(G(z)) \approx 0$ which gives very strong gradient. This intuition is provided by one of the author of the author of the original GAN, Goodfellow. However, people later find that the zero sum formulation also works.

In addition, the process of calculating the equilibrium of the non-zero sum formulation is basically the same as the the zero sum formulation one which we talked about before. The discriminator implicitly developed a metric between the data distribution and the generated distribution (KL divergence). The distance between them is exactly what generator want to minimize.

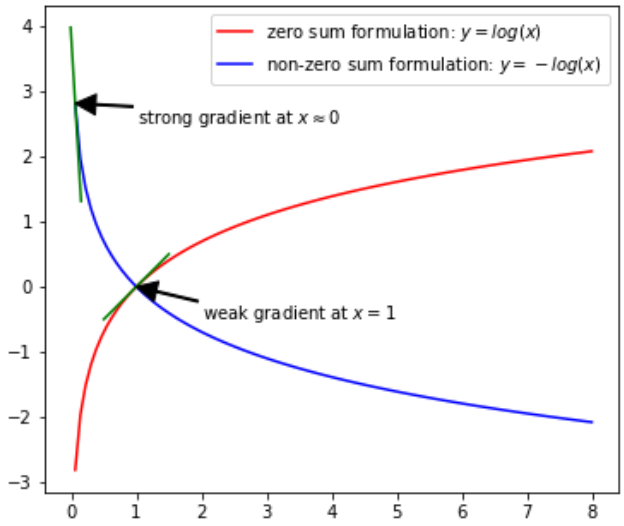

Figure 5: Comparing the gradient in the non-zero sum formulation and zero sum formulation.

Specifically, if we plug in the equilibrium of the discriminator x to the original target function, we will get:

$$\mathbb{E}_{x\sim P_r} \log \frac{P_r(x)}{\frac{1}{2}\left[P_r(x)+P_g(x)\right]} + \mathbb{E}_{x\sim P_g} \log \frac{P_g(x)}{\frac{1}{2}\left[P_r(x)+P_g(x)\right]} - 2\log 2 = 2JS\left(P_r \| P_q\right) - 2\log 2$$

where $JS\left(P_1\|P_2\right) = \frac{1}{2}KL\left(P_1\|\frac{P_1+P_2}{2}\right) + \frac{1}{2}KL\left(P_2\|\frac{P_1+P_2}{2}\right)$
$KL\left(P_1\|P_2\right) = \mathbb{E}_{x\sim P_1} \log \frac{P_1}{P_2}$

8 Conditional GAN

In conditional GANs [6], we want to generate images from a conditional probability distribution. Both the generator and the discriminator are conditioned based on the type of the image that one wants to generate. The difference between this framework and a normal GAN is that an additional condition is added to the input and the output is trained to fit this condition, as shown at figure 6 (top). Figure 6 (bottom) shows a particular conditional GAN called pix2pix [4] which does image to image translation.

9 Further reading and references

9.1 Why generative Modeling?

One motivation for generative modeling is unsupervised learning. More precisely, we can use generative models to learn meaningful latent features. For example, BiGAN [1] (figure 8) introduces an encoder that maps real examples to the GAN latent space. The encoder is trained by the discriminator, which now discriminates not only in data space but jointly in data and latent space.
Another application of deep generative modeling is super-resolution, where the task is to increase the resolution of a picture. Figure 7 shows the results of HiFaceGAN [12], increasing the resolution of a portrait.

9.2 The main Deep Generative Models

One of the main family of deep generative models are GANs [3], which were the subject of lecture 7. However, other popular generative models include auto-regressive models like PixelRNN [10] and PixelCNN [11] as well as variational autoencoders (VAE) [5].

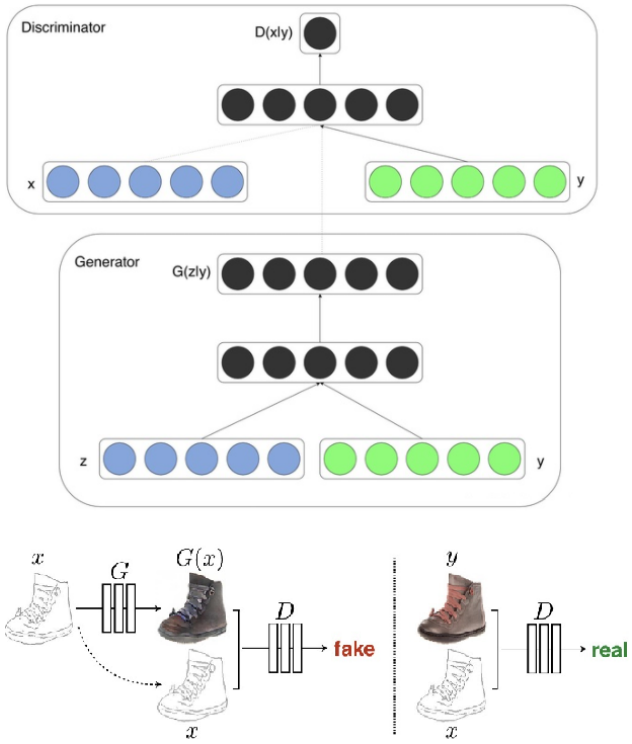

Figure 6: Conditional GAN [6] (top) and pix2pix [4] (bottom)

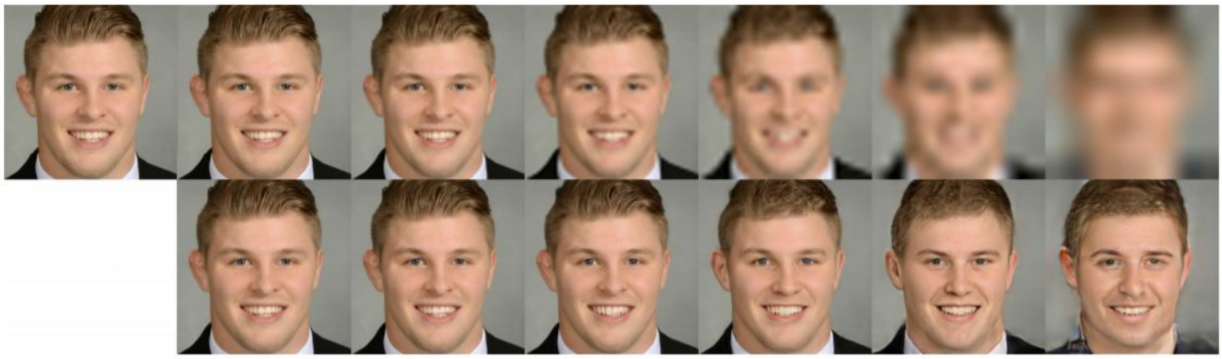

Figure 7: HiFaceGAN generated samples (bottom) [12]

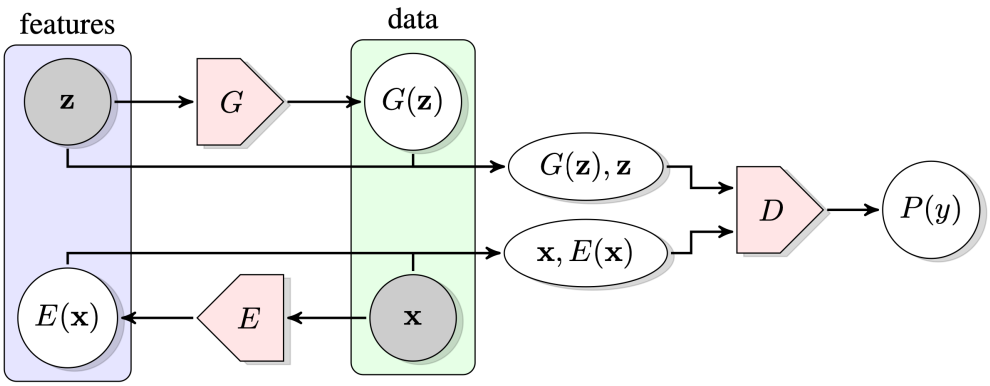

Figure 8: The structure of Bidirectional Generative Adversarial Networks (BiGAN) [1]

References

[1] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial feature learning, 2017.

[2] I. J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017. URL http://arxiv.org/abs/1701.00160.

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[4] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks, 2016. URL http://arxiv.org/abs/1611.07004.

[5] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2014.

[6] M. Mirza and S. Osindero. Conditional generative adversarial nets, 2014. URL http://arxiv.org/abs/1411.1784.

[7] A. Oord and J. Dambre. Locally-connected transformations for deep gmms. In *ICML 2015*, 2015.

[8] L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models, 2016.

[9] A. van den Oord and J. Dambre. Locally-connected transformations for deep gmms. In *International Conference on Machine Learning (ICML) : Deep learning Workshop, Abstracts*, pages 1–8, 2015. URL https://sites.google.com/site/deeplearning2015/20.pdf?attredirects=0.

[10] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks, 2016.

[11] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. Conditional image generation with pixelcnn decoders, 2016.

[12] L. Yang, C. Liu, P. Wang, S. Wang, P. Ren, S. Ma, and W. Gao. Hifacegan: Face renovation via collaborative suppression and replenishment, 2020.