Deep Generative Models

Chapter 4: Generative Adversarial Networks

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

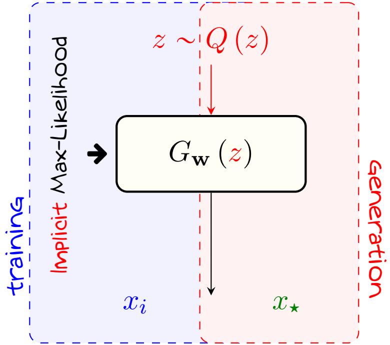
University of Toronto

Summer 2025

Modern Data Generation

As we mentioned at the end of Chapter 2

modern generative models mainly learn how to sample!



Low-dimensional Latent

From our discussions on latent space: *we expect that the latent be of much lower dimension* $m \ll d$

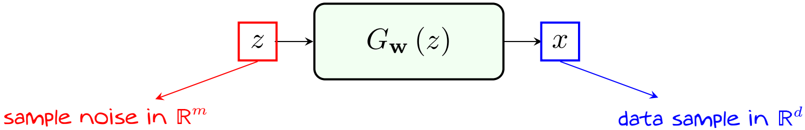
- ! *We could not work with such latent in flow-based models*
  - ↳ *The flow needs to be invertible*
  - ↳ *Invertibility requires the latent and data space to be of the same dimension!*
- *We however intuitively expect that a much smaller latent could do the job*

+ *What if we make a model that maps a low-dimensional latent into the data space?*
- *Well! We can try; however, we get into trouble training it!*
+ *Can't we use simply MLE?!*
- *Not explicitly! Let's take a look*

Generator: *Non-invertible Flow*

Generator Model

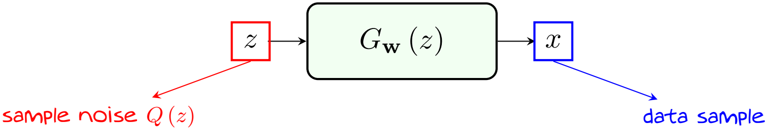
Generator model $G_{\mathbf{w}} : \mathbb{R}^m \mapsto \mathbb{R}^d$ is a mapping that maps a latent sample $z \sim Q(z)$, typically Gaussian noise, into a data sample



+ *Isn't that what we called flow?!*
– Not exactly! *Flow* is *invertible*

Sampling

We can sample from a generator *exactly as in flow-based models*



Sample_Generator():

Sample latent as $z \sim Q(z)$ `#Gaussian noise`
Pass through generator $x \leftarrow G_{\mathbf{w}}(z)$
return x

MLE Learning: *Generator Distribution*

Given $z \sim Q(z)$: *generated sample* $x = G_{\mathbf{w}}(z)$ *is distributed by some* $P_{\mathbf{w}}(x)$

$$\mathrm{CDF}_{\mathbf{w}}(x) = \Pr\{G_{\mathbf{w}}(z) \leqslant x\}$$

The generator distribution $P_{\mathbf{w}}(x)$ *is then given by*

$$P_{\mathbf{w}}(x) = \frac{\partial^d}{\partial x_1 \ldots \partial x_d} \mathrm{CDF}_{\mathbf{w}}(x)$$

Since $G_{\mathbf{w}}$ *is not invertible, we should define*

$$\mathbb{Z}_{\mathbf{w}}(x) = \{z \in \mathbb{R}^m : G_{\mathbf{w}}(z) \leqslant x\}$$

and compute the generator distribution $P_{\mathbf{w}}(x)$ *as*

$$\mathrm{CDF}_{\mathbf{w}}(x) = \int_{\mathbb{Z}_{\mathbf{w}}(x)} P(z) \mathrm{d}z$$

MLE Learning: *Generator Distribution*

The generator distribution is hence given by

$$P_{\mathbf{w}}(x) = \frac{\partial^d}{\partial x_1 \ldots \partial x_d} \int_{\mathbb{Z}_{\mathbf{w}}(x)} P(z) \mathrm{d}z$$

which is a challenging computation!

+ *Isn't this simply computed by chain rule?!*
– *Not that simple! As $G_{\mathbf{w}}$ is not invertible, the set $\mathbb{Z}_{\mathbf{w}}(x)$ cannot be simply characterized! Let's see an example*

Recall

When $G_{\mathbf{w}}$ is invertable, we can readily specify $\mathbb{Z}_{\mathbf{w}}(x)$ using $G_{\mathbf{w}}^{-1}$

$$\mathbb{Z}_{\mathbf{w}}(x) = G_{\mathbf{w}}^{-1}\left(\left\{u \in \mathbb{R}^d : u \leqslant x\right\}\right)$$

Generator Distribution: *Example*

Consider a dummy example: $z \in \mathbb{R}$ *and* $x \in \mathbb{R}^3$ *with generator*

$$G(z) = \left[-z^2, \exp\{-z\}, z\right]$$

Say, want to find the CDF at $x = [x_1, x_2, x_3]$*: we need to find*

$$\begin{aligned}\mathbb{Z}(x) &= \{z \in \mathbb{R} : G(z) \leqslant x\} \\ &= \left\{z \in \mathbb{R} : -z^2 \leqslant x_1 \text{ and } \exp\{-z\} \leqslant x_2 \text{ and } z \leqslant x_3\right\}\end{aligned}$$

---

1. *First constraint requires*

$$-z^2 \leqslant x_1 \rightsquigarrow z^2 \geqslant -x_1 \rightsquigarrow \begin{cases} z \geqslant \sqrt{-x_1} \\ z \leqslant -\sqrt{-x_1} \end{cases}$$

Generator Distribution: *Example*

We need to find

$$\mathbb{Z}(x) = \left\{ z \in \mathbb{R} : -z^2 \leqslant x_1 \ \textit{and} \ \exp\{-z\} \leqslant x_2 \ \textit{and} \ z \leqslant x_3 \right\}$$

---

2. *Second constraint requires*

$$\exp\{-z\} \leqslant x_2 \rightsquigarrow -z \leqslant \log x_2 \rightsquigarrow z \geqslant -\log x_2$$

3. *And, finally the third one restricts z to satisfy*

$$z \leqslant x_3$$

Generator Distribution: *Example*

So, at $x = [x_1, x_2, x_3]$ *we have*

$$\mathbb{Z}(x) = \left(\left[-\infty, -\sqrt{-x_1} \right] \cup \left[\sqrt{-x_1}, +\infty \right] \right) \cap \left[-\log x_2, \infty \right] \cap \left[-\infty, x_3 \right]$$

- *At* $x = [-1, \exp\{2\}, 2]$, *we have*

$$\begin{aligned} \mathbb{Z}(x) &= \left([-\infty, -1] \cup [1, \infty] \right) \cap [-2, \infty] \cap [-\infty, 2] \\ &= [-2, -1] \cup [1, 2] \end{aligned}$$

- *At* $x = [-1, 1, 2]$, *we have*

$$\mathbb{Z}(x) = [1, 2]$$

- *At* $x = [a, 1, 2]$ *with* $a > 0$, *we have*

$$\mathbb{Z}(x) = \emptyset$$

Generator Distribution: *Example*

To find generator distribution, we need $\mathrm{CDF}(x)$ *at all* $x \in \mathbb{R}^3$*; however,*

$$\boxed{\text{computation of } \mathrm{CDF}(x) \text{ changes as } x \text{ changes}}$$

For instance, in our earlier sample points, we have

$$\mathrm{CDF}(x) = \begin{cases} \int_{-2}^{-1} P(z)\mathrm{d}z + \int_{1}^{2} P(z)\mathrm{d}z & x = [-1, \exp\{2\}, 2] \\ \int_{1}^{2} P(z)\mathrm{d}z & x = [-1, 1, 2] \\ 0 & x = [a, 1, 2] \text{ and } a > 0 \\ \vdots & \vdots \end{cases}$$

$$\boxed{\text{Expressing } P(x) \text{ for all } x \in \mathbb{R}^3 \text{ is quite cumbersome!}}$$

MLE Challenge: *Non-Computability of Likelihood*

Computability of Generator Distribution

With a non-invertible generator $G_{\mathbf{w}}$, it is challenging to characterize

$$\mathbb{Z}_{\mathbf{w}}(x) = \{z \in \mathbb{R}^m : G_{\mathbf{w}}(z) \leqslant x\}$$

and thus the output distribution $P_{\mathbf{w}}(x)$ is hard to be computed for all $x \in \mathbb{R}^d$

To train the generator by MLE ⇝ *we need likelihood at all samples x^j*

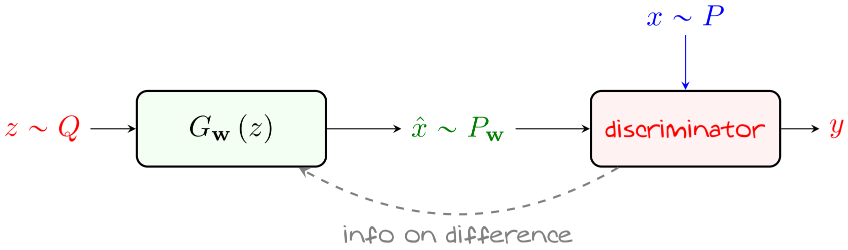
$$\hat{R}(\mathbf{w}) = -\frac{1}{n}\sum_j \boxed{\log P_{\mathbf{w}}(x^j)}$$

?!

Non-Computability of Likelihood

With non-invertible generator, likelihood is not directly computable

Alternative Learning Approach: *Adversarial Architecture*



Discriminator

Discriminator is a binary classifier that classifies sample x as real or fake

To train with discriminator: *we can easily make a labeled dataset*

- *Data samples are labeled as real*
- *Generated samples are labeled as fake*

Dataset for *Adversarial Architecture*

From discriminator viewpoint: *we train a classifier on $2n$ labeled samples*

$$\mathbb{D} = \{\boxed{(x^j, 1)}, \boxed{(\hat{x}^j, 0)} : j = 1, \dots, n\}$$

real data sample

fake generated sample

We can compactly write the dataset as

$$\mathbb{D} = \{(x^j, v^j) : j = 1, \dots, 2n\}$$

where the true label v is defined as

$$v = \begin{cases} 0 & x \text{ if } \textit{fake} \equiv x = G_{\mathbf{w}}(z) \\ 1 & x \text{ if } \textit{real} \equiv x \sim P(x) \end{cases}$$

Training *Computational Discriminator*

We now consider a computational discriminator: *a model* $D_\phi : \mathbb{R}^d \mapsto [0, 1]$ *which classifies sample* x

$$x \longrightarrow \boxed{D_\phi\left(x\right)} \longrightarrow y = P\left(v|x\right)$$

Recall

This is a discriminative model! This explains the appellation ☺

This is a classical classification problem $\rightsquigarrow$ *we use cross-entropy loss*

$$\begin{aligned} \mathrm{CE}\left(y, v\right) &= -v \log y - \left(1 - v\right) \log\left(1 - y\right) \\ &= \begin{cases} -\log y & \text{if } x \text{ is real} \equiv v = 1 \\ -\log\left(1 - y\right) & \text{if } x \text{ is fake} \equiv v = 0 \end{cases} \end{aligned}$$

Empirical Risk of *Adversarial Architecture*

To train on this labeled dataset $\rightsquigarrow$ *we minimize the empirical risk*, *i.e.*,

$$
\begin{aligned}
\hat{R}(\mathbf{w}, \phi) &= \frac{1}{2n} \sum_j \mathrm{CE}\left(y^j, v^j\right) \\
&= \frac{1}{2n} \sum_j \mathrm{CE}\left(D_\phi\left(x^j\right), v^j\right) \\
&= \frac{1}{n} \sum_j \mathrm{CE}\left(D_\phi\left(x^j\right), 1\right) + \frac{1}{n} \sum_j \mathrm{CE}\left(D_\phi\left(G_{\mathbf{w}}\left(z^j\right)\right), 0\right) \\
&= \hat{\mathbb{E}}_{x\sim P}\left\{\mathrm{CE}\left(D_\phi\left(x\right), 1\right)\right\} + \hat{\mathbb{E}}_{z\sim Q}\left\{\mathrm{CE}\left(D_\phi\left(G_{\mathbf{w}}\left(z\right)\right), 0\right)\right\} \\
&= \hat{\mathbb{E}}_{x\sim P}\left\{-\log D_\phi\left(x\right)\right\} + \hat{\mathbb{E}}_{z\sim Q}\left\{-\log\left(1 - D_\phi\left(G_{\mathbf{w}}\left(z\right)\right)\right)\right\} \\
&= -\left[\hat{\mathbb{E}}_{x\sim P}\left\{\log D_\phi\left(x\right)\right\} + \hat{\mathbb{E}}_{z\sim Q}\left\{\log\left(1 - D_\phi\left(G_{\mathbf{w}}\left(z\right)\right)\right)\right\}\right]
\end{aligned}
$$

Empirical risk depends on both generator and discriminator

Training by Min-Max Game: *Discriminator's Role*

To train the adversarial architecture *by the empirical risk*

$$\hat{R}(\mathbf{w}, \phi) = -\left[\hat{\mathbb{E}}_{x\sim P}\left\{\log D_\phi\left(x^j\right)\right\} + \hat{\mathbb{E}}_{z\sim Q}\left\{\log\left(1 - D_\phi\left(G_\mathbf{w}\left(z^j\right)\right)\right)\right\}\right]$$

we let the discriminator and generator play a game

Discriminator tries its best to correctly classify fake sample from true ones, i.e.,

$$\begin{aligned}\min_\phi \hat{R}(\mathbf{w}, \phi) &\equiv \max_\phi \boxed{\hat{\mathbb{E}}_{x\sim P}\left\{\log D_\phi\left(x^j\right)\right\} + \hat{\mathbb{E}}_{z\sim Q}\left\{\log\left(1 - D_\phi\left(G_\mathbf{w}\left(z^j\right)\right)\right)\right\}} \\ &\equiv \max_\phi \boxed{\mathcal{L}(\mathbf{w}, \phi)}\end{aligned}$$

After maximization, discriminator's best try gets the objective[1]

$$\mathcal{L}^\star(\mathbf{w}) = \max_\phi \mathcal{L}(\mathbf{w}, \phi)$$

[1] *Recall that $\mathcal{L}(\mathbf{w}, \phi)$ is indeed the classifier likelihood!*

Training by Min-Max Game: *Generator's Role*

Generator tries its best to *confuse discriminator*

- *This way discriminator cannot distinguish between fake and true samples*
  - ↳ *This means that fake samples look the same as the true ones*
- *The generator can only play with its parameters* $\mathbf{w}$

Generator tries to make discriminator's best try as bad as possible, i.e.,

$$\min_{\mathbf{w}} \mathcal{L}^{\star}(\mathbf{w}) = \min_{\mathbf{w}} \max_{\phi} \mathcal{L}(\mathbf{w}, \phi)$$

Statistical Perspective

Generator minimizes discriminator's maximal likelihood $\rightsquigarrow$ *discriminator remains confused between generated and true samples even if it does its best*

GAN: *Generative Adversarial Network*

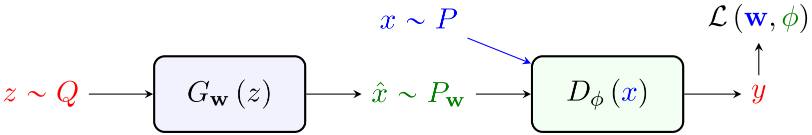
Generative Adversarial Network

GAN consists of a generator $G_{\mathbf{w}}$ and discriminator D_ϕ that are jointly learned through the min-max game

$$\min_{\mathbf{w}} \max_{\phi} \mathcal{L}\left(\mathbf{w}, \phi\right)$$

with objective function

$$\mathcal{L}\left(\mathbf{w}, \phi\right) = \hat{\mathbb{E}}_{x \sim P}\left\{\log D_\phi\left(x\right)\right\} + \hat{\mathbb{E}}_{z \sim Q}\left\{\log\left(1 - D_\phi\left(G_{\mathbf{w}}\left(z\right)\right)\right)\right\}$$



GAN: *Training Loop*

`Train_vanillaGAN(`$\mathbb{D}$`:dataset):`

Initiate the generator $G_{\mathbf{w}}$ *and discriminator* D_ϕ *with some* $\mathbf{w}$ and ϕ
for *multiple epochs* **do**
    Sample a batch of data samples $\{x^j : j = 1, \dots, n\}$ *from* $\mathbb{D}$
    Sample latents $\{z^j : j = 1, \dots, n\}$ *using* Q *and compute* $\hat{x}^j \leftarrow G_{\mathbf{w}}(z^j)$
    for $\xi = 1, \dots, \Xi$ **do**
        for $j = 1, \dots, n$ **do**
            Compute $\mathcal{L}^j = \log D_\phi(x^j) + \log(1 - D_\phi(\hat{x}^j))$
            Backpropagate over discriminator to compute $\nabla_\phi \mathcal{L}^j$
            if $\xi = \Xi$ **then** *Backpropagate over generator to compute* $\nabla_{\mathbf{w}} \mathcal{L}^j$
        end for
        Update ϕ *using* `Opt_avg` $\{\nabla_\phi \mathcal{L}^j\}$
    end for
    Update $\mathbf{w}$ *using* `Opt_avg` $\{\nabla_{\mathbf{w}} \mathcal{L}^j\}$
end for
return *trained generator* $G_{\mathbf{w}}$

GAN Training: *Few Notes*

There are a few tricks to make training work

1. *We update discriminator for Ξ steps before updating generator once*
   - ↳ *This allows the inner maximization to slightly converge*

$$\min_{\mathbf{w}} \max_{\phi} \mathcal{L}(\mathbf{w}, \phi)$$

   - ↳ *Typical choices are $5 \leqslant \Xi \leqslant 10$*

2. *To update generator, we only need to compute derivative of second term*

$$\nabla_{\mathbf{w}} \mathcal{L}^j = \underbrace{\nabla_{\mathbf{w}} \log D_\phi\left(x^j\right)}_{0} + \nabla_{\mathbf{w}} \log\left(1 - D_\phi\left(G_{\mathbf{w}}\left(z^j\right)\right)\right)$$

   - ↳ *This gives very small gradients first and exploding later*
     - ↳ *First $D_\phi\left(G_{\mathbf{w}}\left(z^j\right)\right) \approx 0$ and later when it's fooled $D_\phi\left(G_{\mathbf{w}}\left(z^j\right)\right) \approx 1$*
     - ↳ *This is opposite of what we want!*
   - ↳ *Heuristic remedy is to use $-\nabla_{\mathbf{w}} \log D_\phi\left(G_{\mathbf{w}}\left(z^j\right)\right)$ instead*
     - ↳ *Some people call $\mathcal{L}_G^j = -\log D_\phi\left(G_{\mathbf{w}}\left(z^j\right)\right)$ generator loss*

GAN: *Sampling*

Sampling is exactly as in flow-based models

```
Sample_GAN():
```
1: *Sample latent as* $z \sim Q(z)$ `#Gaussian noise`
2: *Pass through generator* $x \leftarrow G_{\mathbf{w}}(z)$
3: **return** generated sample x

Vanilla GAN

+ *Why you called the training "vanilla"?!*
– Because we used a *basic loss for our adversarial network*
+ *Can we do better?*
– Yes! That's what we do in *Wasserstein GAN!*

Though working with tricks, the vanilla GAN *training is generally unstable*

Wasserstein GAN addresses this issue by a very smart trick

To understand the trick used by Wasserstein GAN, we need to first learn the statistical interpretation of vanilla GAN *training*

Next Stop: Interpreting GAN *training as divergence minimization*