Generative Adversarial Nets
Implicit Generative Model

- **Goal:** a sampler \( g(\cdot) \) to generate images
- A simple generator \( g(z; \theta) \):
  - \( z \sim N(0,I) \)
  - \( x = g(z; \theta) \) deterministic transformation

- Likelihood-free training:
  - Given a dataset from some distribution \( p_{data} \)
  - Goal: \( g(z; \theta) \) defines a distribution, we want this distribution \( \approx p_{data} \)
  - Training: minimize \( D(g(z; \theta), p_{data}) \)
    - \( D \) is some distance metric (not likelihood)
  - Key idea: **Learn a differentiable** \( D \)
GAN (Goodfellow et al., ‘14)

- Parameterize the discriminator $D(\cdot ; \phi)$ with parameter $\phi$

**Goal:** learn $\phi$ such that $D(x; \phi)$ measures how likely $x$ is from $p_{data}$
- $D(x, \phi) = 1$ if $x \sim p_{data}$
- $D(x, \phi) = 0$ if $x! \sim p_{data}$
- a.k.a., a binary classifier

- GAN: use a neural network for $D(\cdot ; \phi)$

**Training:** need both negative and positive samples
- Positive samples: just the training data
- Negative samples: use our sampler $g(z; \theta)$ (can provide infinite samples).

**Overall objectives:**
- Generator: $\theta^* = \max_{\theta} D(g(z; \theta); \phi)$
- Discriminator uses MLE Training:
  $$\phi^* = \max_{\phi} \mathbb{E}_{x \sim p_{data}}[\log D(x; \phi)] + \mathbb{E}_{\hat{x} \sim g(\cdot)}[\log(1 - D(\hat{x}; \phi))]$$
GAN (Goodfellow et al., ‘14)

• Generator $g(z; \theta)$ where $z \sim N(0,I)$
  • Generate realistic data

• Discriminator $D(x; \phi)$
  • Classify whether the data is real (from $p_{data}$) or fake (from $g$)

• Objective function:
  $$L(\theta, \phi) = \min_{\theta} \max_{\phi} \mathbb{E}_{x \sim p_{data}} \left[ \log D(x; \phi) \right] + \mathbb{E}_{\hat{x} \sim g(z, \theta)} \left[ \log(1 - D(\hat{x}; \phi)) \right]$$

• Training procedure:
  • Collect dataset $\{(x, 1) \mid x \sim p_{data}\} \cup \{((\hat{x}, 0) \sim g(z; \theta)\}$
  • Train discriminator
    $$D : L(\phi) = \mathbb{E}_{x \sim p_{data}} \left[ \log D(x; \phi) \right] + \mathbb{E}_{\hat{x} \sim g(z, \theta)} \left[ \log(1 - D(\hat{x}; \phi)) \right]$$
  • Train generator $g : L(\theta) = \mathbb{E}_{z \sim N(0,I)} \left[ \log D(g(z; \theta), \phi) \right]$
  • Repeat
GAN (Goodfellow et al., ‘14)

- Objective function:
  \[ L(\theta, \phi) = \min_{\theta} \max_{\phi} \mathbb{E}_{x \sim p_{data}} \left[ \log D(x; \phi) \right] + \mathbb{E}_{\hat{x} \sim G} \left[ \log (1 - D(\hat{x}; \phi)) \right] \]
Math Behind GAN
Math Behind GAN
KL-Divergence and JS-Divergence
Math Behind GAN
Evaluation of GAN

- No $p(x)$ in GAN.
- Idea: use a trained classifier $f(y \mid x)$:
  - If $x \sim p_{data}, f(y \mid x)$ should have low entropy
    - Otherwise, $f(y \mid x)$ close to uniform.
- Samples from $G$ should be diverse:
  - $p_f(y) = \mathbb{E}_{x \sim G}[f(y \mid x)]$ close to uniform.
Evaluation of GAN

- **Inception Score** (IS, Salimans et al. ’16)
  - Use Inception V3 trained on ImageNet as $f(y | x)$
  - $IS = \exp \left( \mathbb{E}_{x \sim G} \left[ KL(f(y | x) || p_f(y)) \right] \right)$
  - Higher the better
Comments on GAN

• Other evaluation metrics:
  • Fréchet Inception Distance (FID): Wasserstein distance between Gaussians

• Mode collapse:
  • The generator only generate a few type of samples.
  • Or keep oscillating over a few modes.

• Training instability:
  • Discriminator and generator may keep oscillating
  • Example: $-xy$, generator $x$, discriminatory. NE: $x = y = 0$ but GD oscillates.
  • No stopping criteria.
  • Use Wasserstein GAN (Arjovsky et al. ’17):
    $\min G \max f: \text{Lip}(f) \leq 1 \mathbb{E}_{x \sim p_{data}} [f(x)] - \mathbb{E}_{\hat{x} \sim p_G} [f(\hat{x})]$
  • And need many other tricks...
Variational Autoencoder
Architecture

- Auto-encoder: $x \rightarrow z \rightarrow x$
- Encoder: $q(z \mid x; \phi) : x \rightarrow z$
- Decoder: $p(x \mid z; \theta) : z \rightarrow x$

- Isomorphic Gaussian:
  $q(z \mid x; \phi) = N(\mu(x; \phi), \text{diag}(\exp(\sigma(x; \phi))))$
- Gaussian prior: $p(z) = N(0, I)$
- Gaussian likelihood: $p(x \mid z; \theta) \sim N(f(z; \theta), I)$

- Probabilistic model interpretation: latent variable model.
VAE Training

- Training via optimizing ELBO
  \[ L(\phi, \theta; x) = \mathbb{E}_{z \sim q(z|x; \phi)}[\log p(z | x; \theta)] - KL \left( q(z | x; \phi) \left| \left| p(z) \right. \right) \right) \]
  - Likelihood term + KL penalty

- KL penalty for Gaussians has closed form.
- Likelihood term (reconstruction loss):
  - Monte-Carlo estimation
  - Draw samples from \( q(z | x; \phi) \)
  - Compute gradient of \( \theta \):
    - \( x \sim N(f(z; \theta); I) \)
    - \( p(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\|x - f(z; \theta)\|^2\right) \)
VAE Training

- Likelihood term (reconstruction loss):
  - Gradient for $\phi$. Loss: $L(\phi) = \mathbb{E}_{z \sim q(z; \phi)} \left[ \log p(x | z) \right]$ 
  - Reparameterization trick:
    - $z \sim N(\mu, \Sigma) \iff z = \mu + \epsilon, \epsilon \sim N(0, \Sigma)$
    - $L(\phi) \propto \mathbb{E}_{z \sim q(z|\phi)} \left[ \| f(z; \theta) - x \|_2^2 \right]$
    - $\propto \mathbb{E}_{e \sim N(0,I)} \left[ \| f(\mu(x; \phi) + \sigma(x; \phi) \cdot \epsilon; \theta) - x \|_2^2 \right]$
  - Monte-Carlo estimate for $\nabla L(\phi)$

- End-to-end training
**VAE vs. AE**

- **AE**: classical unsupervised representation learning method.
- **VAR**: a probabilistic model of AE
  - AE + Gaussian noise on \( z \)
  - KL penalty: \( L_2 \) constraint on the latent vector \( z \)
Conditioned VAE

- Semi-supervised learning: some labels are also available

conditioned generation
Comments on VAE

• Pros:
  • Flexible architecture
  • Stable training

• Cons:
  • Inaccurate probability evaluation (approximate inference)
Energy-Based Models
Energy-based Models

• Goal of generative models:
  • a probability distribution of data: \( P(x) \)

• Requirements
  • \( P(x) \geq 0 \) (non-negative)
  
  \[
  \int_{x} P(x) dx = 1
  \]

• Energy-based model:
  • Energy function: \( E(x; \theta) \), parameterized by \( \theta \)
  
  \[
  P(x) = \frac{1}{z} \exp(-E(x; \theta)) \text{ (why exp?)}
  \]

  \[
  z = \int_{z} \exp(-E(x; \theta)) dx
  \]
Boltzmann Machine

- Generative model
  - \( E(y) = \frac{1}{2} y^\top W y \)
  - \( P(y) = \frac{1}{Z} \exp(-\frac{E(y)}{T}) \), \( T \): temperature hyper-parameter
  - \( W \): parameter to learn
  - When \( y_i \) is binary, patterns are affecting each other through \( W \)

\[
Z_i = \frac{1}{T} \sum_j w_{ji} s_j
\]

\[
P(s_i = 1|s_j \neq i) = \frac{1}{1 + e^{-Z_i}}
\]
Boltzmann Machine: Training

- Objective: maximum likelihood learning (assume $T=1$):
  - Probability of one sample:
    \[ P(y) = \frac{\exp(\frac{1}{2}y^\top Wy)}{\sum_{y'} \exp(y'^\top Wy')} \]
  - Maximum log-likelihood:
    \[ L(W) = \frac{1}{N} \sum_{y \in D} \frac{1}{2} y^\top Wy - \log \sum_{y'} \exp(\frac{1}{2} y'^\top W y') \]
Boltzmann Machine: Training
Boltzmann Machine: Sampling
Restricted Boltzmann Machine

- A structured Boltzmann Machine
  - Hidden neurons are only connected to visible neurons
  - No intra-layer connections
  - Invented by Paul Smolensky in ’89
  - Became more practical after Hinton invested fast learning algorithms in mid 2000
Restricted Bolzmann Machine

- Computation Rules
  - Iterative sampling

  Hidden neurons \( h_i \): 
  \[
  z_i = \sum_j w_{ij} v_j, \quad P(h_i | v) = \frac{1}{1 + \exp(-z_i)}
  \]

  Visible neurons \( v_j \): 
  \[
  z_j = \sum_i w_{ij} h_i, \quad P(v_j | h) = \frac{1}{1 + \exp(-z_j)}
  \]
Restricted Bolzmann Machine

- Sampling:
  - Randomly initialize visible neurons $v_0$
  - Iterative sampling between hidden neurons and visible neurons
  - Get final sample $(v_\infty, h_\infty)$

- Training:
  - MLE
  - Sampling to approximate gradient
Restricted Boltzmann Machine

- Maximum likelihood estimated:
  \[ \nabla_{wij} L(W) = \frac{1}{NPK} \sum_{v \in P} v_0h_{0j} - \frac{1}{M} \sum v_\infty h_{\infty j} \]

- No need to lift up the entire energy landscape!
  - Raising the neighborhood of desired patterns is sufficient
Deep Boltzmann Machine

- Can we have a **deep** version of RBM?
  - Deep Belief Net (’06)
  - Deep Boltzmann Machine (’09)

- Sampling?
  - Forward pass: bottom-up
  - Backward pass: top-down

- Deep Boltzmann Machine
  - The very first deep generative model
  - Salakhudinov & Hinton
Deep Boltzmann Machine

Deep Boltzmann Machine

4000 units

4000 units

4000 units

Preprocessed transformation

Stereo pair

Gaussian visible units (raw pixel data)

Training Samples

Generated Samples
Summary

• Pros: powerful and flexible
  • An arbitrarily complex density function $p(x) = \frac{1}{z} \exp(-E(x))$

• Cons: hard to sample / train
  • Hard to sample:
    • MCMC sampling
  • Partition function
    • No closed-form calculation for likelihood
    • Cannot optimize MLE loss exactly
    • MCMC sampling