My Notes for XCS236

Fangfei Li

December 15, 2025

Contents

1 Introduction & Background

1.1 Introduction

Definition: Generative model (or statistical generative model) is,

- Probability distribution: $p(\boldsymbol{x})$ with $\boldsymbol{x} \in \mathcal{X}$
- Or in some cases, conditional probability distribution: $p(\boldsymbol{x} \mid \boldsymbol{c})$ with $\boldsymbol{x} \in \mathcal{X}$ and $\boldsymbol{c} \in \mathcal{C}$

Type: Generative model includes:

- Probabilistic graphical models (PGM):
  - Directed Probabilistic graphical models (DPGM): Bayesian Networks (BN)
  - Undirected Probabilistic graphical models (UPGM): Markov random field (MRF)
- Deep generative models (DGM):
  - Density-based generative models: Explicit $p(\boldsymbol{x})$, trained by maximum likelihood.
    - Autoregressive models (ARM)
    - Variational autoencoders (VAE)
    - Normalizing flows
  - Likelihood-free generative models: Implicit $p(\boldsymbol{x})$, generate samples from noise. through a generator network)
    - Energy based models (EBM): learn from energy differences
    - Diffusion models: learn from denoising objectives
    - Generative adversarial networks (GAN): learn from sample comparisons via a discriminator

Applications: Generative model can be used for,

1. Generating data:
   - Generate new data samples (e.g., image, text, audio, etc).
   - Create synthetic data for training discriminative models.
   - Usually use a conditional generative model $p(\boldsymbol{x} \mid \boldsymbol{c})$ to control what is generated.
     (a) text-to-image model: $\boldsymbol{c}$ = text prompt, $\boldsymbol{x}$ = image
     (b) Image-to-text model: $\boldsymbol{c}$ = image, $\boldsymbol{x}$ = text (image captioning)
     (c) image-to-image model: $\boldsymbol{c}$ = image, $\boldsymbol{x}$ = image (image colorization, inpainting, uncropping, JPEG artefact restoration)
     (d) speech-to-text model: $\boldsymbol{c}$ = sequence of sounds, $\boldsymbol{x}$ = sequence of words (automatic speech recognition (ASR))
     (e) sequence-to-sequence
       - $\boldsymbol{c}$ = sequence of English words, $\boldsymbol{x}$ = sequence of French words (machine translation)
       - $\boldsymbol{c}$ = initial prompt, $\boldsymbol{x}$ = continuation of the text (text generation)
   - Difference between discriminative model and conditional generative model
     (a) Discriminative (predictive) model: A function $f_\theta : \boldsymbol{x} \to \boldsymbol{y}$, where there is a single correct output $\boldsymbol{y}$ for each input $\boldsymbol{x}$.

   (b) Conditional generative model: A probability distribution $p(\boldsymbol{x} \mid \boldsymbol{c})$, where multiple outputs $\boldsymbol{x}$ may correspond to the same input $\boldsymbol{c}$. This makes generative models more harder to evaluate.

2. Density estimation:
   - Evaluate the probability of an observed data vector, i.e., compute $p(\boldsymbol{x})$.
   - Used in outlier detection, data compression, generative classifiers, model comparison, etc.
   - Approaches for outlier detection:
     (a) kernel density estimation (KDE): a simple non-parametric generative model only for low-dimensional data. For higher-dimensional data, it suffers from the curse of dimensionality, making density estimation inaccurate and computationally expensive. Need parametric generative models (e.g., Gaussian mixtures, Hidden Markov Models).
     (b) deep generative model: can learn complex, high-dimensional data distributions.
3. Imputation:
   - "Filling in" missing values of a data vector or data matrix
   - Used in in-painting occluded pixels in an image
   - Approaches for imputation
     (a) mean value imputation
     (b) multiple imputation
4. Structure discovery:
   - For generative models with latent variables, the latent vector $\boldsymbol{z}$ is assumed to represent the underlying "causes" that generate the observed data $\boldsymbol{x}$. By Bayes' rule, the posterior distribution is obatained as
   $$p(\boldsymbol{z} \mid \boldsymbol{x}) = \frac{p(\boldsymbol{z})p(\boldsymbol{x} \mid \boldsymbol{z})}{p(\boldsymbol{x})} \propto p(\boldsymbol{z})p(\boldsymbol{x} \mid \boldsymbol{z})$$
   which is useful for discovering latent, low-dimensional patterns in the data.
5. Latent space interpolation:
   - Certain latent variable models can generate new samples with desired semantic properties by interpolating between existing datapoints in the latent space.
     (a) Given datapoints $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, obtain their latent representation using the encoder $e(\cdot)$, $\boldsymbol{z}_1 = e(\boldsymbol{x}_1)$ and $\boldsymbol{z}_2 = e(\boldsymbol{x}_2)$.
     (b) Construct an interpolated latent point, $\boldsymbol{z} = \lambda \boldsymbol{z}_1 + (1 - \lambda)\boldsymbol{z}_2$.
     (c) Decoding $\boldsymbol{z}$ using the decoder $d(\cdot)$, $\boldsymbol{x} = d(\boldsymbol{z})$. $\boldsymbol{x}$ typically combines semantic features from both $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$.
     (d) Linear interpolation is commonly used because the learned latent manifold often exhibits approximately zero curvature in practice. However, in some cases nonlinear interpolation (e.g., spherical interpolation, geodesic interpolation) may better respect the geometry of the latent space.
6. Latent space arithmetic:
   - Increase or decrease the amount of a desired "semantic factor of variation".
7. Generative design:
   - Generate candidate objects (e.g., molecules) which have desired properties.
   - Fit a VAE to unlabeled samples, then to perform Bayesian optimization.
8. Model-based reinforcement learning:

9. Representation learning:
   - Learn latent features $\boldsymbol{z}$ from the observed data $\boldsymbol{x}$, that can be used for downstream supervised tasks.
10. Data compression:

Slide Note

- Make sense of complex, high-dimensional data (e.g., Image, speech signal, a sequence of tokens/characters, etc.) by converting them into useful representations that reveal their true meaning.
- Quote:
  - Richard Feynman: "What I cannot create, I do not understand."
  - Philosophy Behind Generative modeling: "What I understand, I can create."
- Model structure:
  - Generation (graphic): High level description → Raw sensory output
  - Inference (inverse graphic): Raw sensory output → High level description

  Note: Many of models in this course will have similar structure: Generation + Inference
- A statistical generative model is a probability distribution $p(\boldsymbol{x})$ that is learned from,
  - Data (samples) - more weight
  - Prior Knowledge - less weight
- Data → Data simulator (= Statistical model = Generative model = Probability distribution $p(\boldsymbol{x})$)
  - Generation (Sampling): Called generative because sampling from $p(\boldsymbol{x})$ generates new sample.

    Control signal (e.g., texts/captions, images, medical signals, etc.) → Data simulator → New data
  - Density estimation: Used for tasks such as outlier or anomaly detection.

    New data → Data simulator → Density

Model Type: Models can be categorized along two dimensions:

- What the model learns
  1. Generative models
     - Learn the underlying data distribution and can generate new samples.
     - E.g., Generative Pretrained Transformer (GPT)
  2. Discriminative models
     - Learn to predict labels or distinguish classes, modeling only $p(y \mid \boldsymbol{x})$.
     - E.g., Bidirectional Encoder Representations from Transformers (BERT)
- How the model is trained
  1. Supervised learning
     - Trained with labeled input–output pairs.
     - E.g., classifiers
  2. Self-supervised learning
     - Trained using labels derived automatically from the data itself.
     - E.g., GPT (predicts next token), BERT (predicts masked tokens)
  3. Unsupervised learning

- Trained on unlabeled data to discover structure or generate data.
- E.g., Variational Autoencoder (VAE)

1.2 Background

Examples of Data Distributions

- Bernoulli distribution
  - $X \sim \text{Bern}(p)$:

    $$P(X = 1) = p, \quad P(X = 0) = 1 - p, \quad \text{Val}(X) = \{0, 1\}$$

  - A black–white image with n pixels:
    1. Each pixel X_i is Bernoulli with $\text{Val}(X_i) = \{0, 1\}$.
    2. The joint distribution of the full image is $p(x_1, x_2, \ldots, x_n)$.
    3. The number of possible images (states) is 2^n.
    4. The number of parameters required to specify the joint distribution is $2^n - 1$.
- Categorical distribution
  - $Y \sim \text{Categorical}(p_1, p_2, \ldots, p_m)$:

    $$P(Y = i) = p_i, \quad \sum_{i=1}^{m} p_i = 1, \quad \text{Val}(Y) = \{1, 2, \ldots, m\}$$

  - Color of a single RGB pixel:
    1. Each channel (R, G, B) is categorical with $\{0, \ldots, 255\}$.
    2. The full pixel has a joint distribution $p(r, g, b)$.
    3. The total number of possible pixel values is 256^3.
    4. The number of parameters required to specify the joint distribution is $256^3 - 1$.

Independence vs. Conditional Independence

1. Independence assumption is too strong.
2. Conditional independence assumption is better.

Chain Rule

$$p(x_1, \ldots, x_n) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_n \mid x_1, \ldots, x_{n-1})$$

Bayes' Rule

$$p(x \mid y) = \frac{p(y \mid x)\, p(x)}{p(y)}$$

$$p(y) = \sum_{x} p(y \mid x)\, p(x)$$

Bayesian Network (BN):

- A BN is defined by a directed acyclic graph (DAG).
- The DAG structure encodes conditional independence relationships among variables.

- The joint distribution factorizes into local conditional probability distributions (CPDs),

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid \boldsymbol{x}_{\text{Parent}(i)})$$

- This factorization defines a valid probability distribution (i.e., it sums to 1) because the DAG provides an ordering consistent with the conditional dependencies.

Neural Models vs. Bayesian Networks:

- Logistic regression is generally more powerful than naive Bayes for predicting Y because it does not rely on the conditional independence assumption $X_i \perp \boldsymbol{X}_{-i} \mid Y$. A sufficiently deep neural network has greater flexibility, as it can approximate arbitrary functions.
- Naive Bayes, however, is more versatile than logistic regression in certain settings. It can be used for tasks beyond predicting Y. Even when the goal is prediction, naive Bayes can still operate when some features in $\boldsymbol{X}$ are missing, while logistic regression typically requires all features to be observed.

Generative vs. Discriminative Models

- A joint distribution can be factorized in two equivalent ways:
  - Generative form: $p(y, \boldsymbol{x}) = p(\boldsymbol{x} \mid y)p(y)$
  - Discriminative form: $p(y, \boldsymbol{x}) = p(y \mid \boldsymbol{x})p(\boldsymbol{x})$
- Examples of generative models:
  - Naive Bayes (Naive BN): Under the conditional independence assumption $X_i \perp \boldsymbol{X}_{-i} \mid Y$, the joint distribution factorizes as $p(y, x_1, \ldots, x_n) = p(y) \prod_{i=1}^{n} p(x_i \mid y)$. The posterior becomes

$$p(Y = 1 \mid x_1, \ldots, x_n) = \frac{p(Y = 1) \prod_{i=1}^{n} p(x_i \mid Y = 1)}{p(Y = 0) \prod_{i=1}^{n} p(x_i \mid Y = 0) + p(Y = 1) \prod_{i=1}^{n} p(x_i \mid Y = 1)}$$

- Example of discriminative models
  - Logistic regression (linear dependence): Let sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ (used for generating probability with values in $[0, 1]$). Given parameters $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_n)$,

$$p_{\text{logit}}(Y = 1 \mid \boldsymbol{x}; \boldsymbol{\alpha}) = \sigma(\alpha_0 + \sum_{i=1}^{n} \alpha_i x_i)$$

  - Neural network model (non-linear dependence): Let $\boldsymbol{h} = f(A\boldsymbol{x} + \boldsymbol{b})$ be a nonlinear hidden representation. Then

$$p_{\text{Neural}}(Y = 1 \mid \boldsymbol{x}; \boldsymbol{\alpha}, A, \boldsymbol{b}) = \sigma(\alpha_0 + \sum_{i=1}^{n} \alpha_i h_i)$$

Continuous Variable

Continuous variables

- If X is a continuous random variable, we can usually represent it using its **probability density function** $p_X : \mathbb{R} \to \mathbb{R}^+$. However, we cannot represent this function as a table anymore. Typically consider parameterized densities:
  - Gaussian: $X \sim \mathcal{N}(\mu, \sigma)$ if $p_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$
  - Uniform: $X \sim \mathcal{U}(a, b)$ if $p_X(x) = \frac{1}{b-a} 1[a \leq x \leq b]$
  - Etc.
- If $\boldsymbol{X}$ is a continuous random vector, we can usually represent it using its **joint probability density function**:
  - Gaussian: if $p_X(x) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(x-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(x-\boldsymbol{\mu})\right)$
- Chain rule, Bayes rule, etc all still apply. For example,

$$p_{X,Y,Z}(x, y, z) = p_X(x) p_{Y|X}(y \mid x) p_{Z|\{X,Y\}}(z \mid x, y)$$

Continuous variables

- This means we can still use Bayesian networks with continuous (and discrete) variables. Examples:
- **Mixture of 2 Gaussians**: Bayes net $Z \to X$ with factorization $p_{Z,X}(z, x) = p_Z(z) p_{X|Z}(x \mid z)$ and
  - $Z \sim \textit{Bernoulli}(p)$
  - $X \mid (Z = 0) \sim \mathcal{N}(\mu_0, \sigma_0)$, $X \mid (Z = 1) \sim \mathcal{N}(\mu_1, \sigma_1)$
  - The parameters are $p, \mu_0, \sigma_0, \mu_1, \sigma_1$
- Bayes net $Z \to X$ with factorization $p_{Z,X}(z, x) = p_Z(z) p_{X|Z}(x \mid z)$
  - $Z \sim \mathcal{U}(a, b)$
  - $X \mid (Z = z) \sim \mathcal{N}(z, \sigma)$
  - The parameters are a, b, σ
- **Variational autoencoder**: Bayes net $Z \to X$ with factorization $p_{Z,X}(z, x) = p_Z(z) p_{X|Z}(x \mid z)$ and
  - $Z \sim \mathcal{N}(0, 1)$
  - $X \mid (Z = z) \sim \mathcal{N}(\mu_\theta(z), e^{\sigma_\phi(z)})$ where $\mu_\theta : \mathbb{R} \to \mathbb{R}$ and σ_ϕ are neural networks with parameters (weights) θ, ϕ respectively
  - **Note**: Even if μ_θ, σ_ϕ are very deep (flexible), functional form is still Gaussian

2 Autoregressive Models

2.1 Introduction

Two-Step Process

1. Model representation: Parameterize a model family $\{p_\theta(x), \theta \in \Theta\}$ to represent $p(x)$
2. Model Learning: Estimate (or optimize) the parameters θ using the training data $\mathcal{D}$

Definiton: Autoregressive (AR) models use parameterized functions to predict each data element (e.g., a pixel) based on all previously observed elements. The joint distribution is defined as

$$p(x_1, \ldots, x_d) = \prod_{i=1}^{d} p(x_i \mid x_1, \ldots, x_{i-1}) = \prod_{i=1}^{d} p(x_i \mid \boldsymbol{x}_{<i})$$

It's self-supervised learning, since the model learns from the data itself without manual labels.

2.2 Examples of AR Models

Example 1 – Recurrent Neural Networks (RNNs): RNNs are autoregressive (AR) models that process a sequence of input vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T$, with each $\boldsymbol{x}_t \in \mathbb{R}^I$. At each time step, the network maintains a hidden state $\boldsymbol{h}_t \in \mathbb{R}^H$.

- Dimensions:
  - I: input dimension
  - H: hidden state dimension (hyperparameter)
  - V: output dimension
- Parameters:
  - $W_x \in \mathbb{R}^{H \times I}$: input-to-hidden weight matrix
  - $W_h \in \mathbb{R}^{H \times H}$: recurrent (hidden-to-hidden) weight matrix
  - $\boldsymbol{b}_h \in \mathbb{R}^H$: bias for the hidden state
  - $W_o \in \mathbb{R}^{V \times H}$: hidden-to-output weight matrix
  - $\boldsymbol{b}_o \in \mathbb{R}^V$: output bias
- Hidden state update using a nonlinear activation $f(\cdot)$ (e.g., tanh, ReLU)

  $$\boldsymbol{h}_t = f(W_h \boldsymbol{h}_{t-1} + W_x \boldsymbol{x}_t + \boldsymbol{b}_h)$$

- Output distribution:

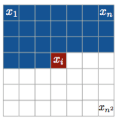
  $$p(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_{\leq t}) = g(W_o \boldsymbol{h}_t + \boldsymbol{b}_o)$$

  where $g(\cdot)$ is typically a softmax.
- $W_o \boldsymbol{h}_t + \boldsymbol{b}_o$: logits, which are the raw, unnormalized scores before applying softmax to convert them into probabilities.

Example 2 – Long Short-Term Memory Networks (LSTMs): LSTMs are AR models, which is an improved form of RNNs that capture long-range dependencies through gating mechanisms.

Example 3 – PixelRNN: PixelRNNs are AR models for images that generate each pixel conditioned on all previously generated pixels.

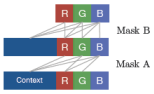
Pixel RNN (Oord et al., 2016)



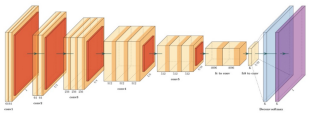
1. Model images pixel by pixel using raster scan order
2. Each pixel conditional $p(x_t \mid x_{1:t-1})$ needs to specify 3 colors

   $$p(x_t \mid x_{1:t-1}) = p(x_t^{red} \mid x_{1:t-1})p(x_t^{green} \mid x_{1:t-1}, x_t^{red})p(x_t^{blue} \mid x_{1:t-1}, x_t^{red}, x_t^{green})$$

   and each conditional is a categorical random variable with 256 possible values
3. Conditionals modeled using RNN variants. LSTMs + masking (like MADE)

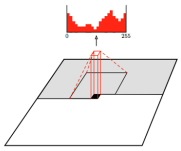


Convolutional Architectures

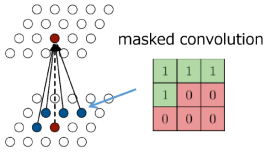


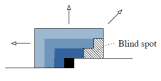
Convolutions are natural for image data and easy to parallelize on modern hardware.

PixelCNN (Oord et al., 2016)



Idea: Use convolutional architecture to predict next pixel given context (a neighborhood of pixels).
Challenge: Has to be autoregressive. Masked convolutions preserve raster scan order. Additional masking for colors order.



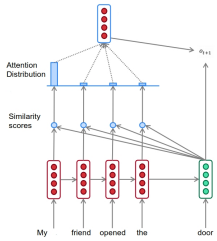


Example 4 – Generative Pre-trained Transformer (GPT): GPT models are transformer-based autoregressive

models that predict the next token given all preceding tokens.

Attention Based Models: A broad class of neural models that use attention mechanisms to selectively weight input information.

Attention based models



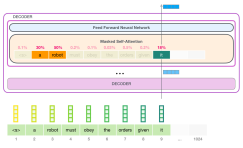
Attention mechanism to compare a *query* vector to a set of *key* vectors

1. Compare current hidden state (query) to all past hidden states (keys), e.g., by taking a dot product
2. Construct attention distribution to figure out what parts of the history are relevant, e.g., via a softmax
3. Construct a summary of the history, e.g., by weighted sum
4. Use summary and current hidden state to predict next token/word

Generative Transformers:

- Generative transformers = Transformers used for generative modeling
- Examples: GPT

Generative Transformers



Current state of the art (GPTs): replace RNN with Transformer

- Attention mechanisms to adaptively focus only on relevant context
- Avoid recursive computation. Use only self-attention to enable parallelization
- Needs **masked** self-attention to preserve autoregressive structure
- Demo: `https://transformer.huggingface.co/doc/gpt2-large`
- Demo: `https://huggingface.co/spaces/huggingface-projects/llama-2-13b-chat`

2.3 Fully Visible Sigmoid Belief Network (FVSBN)

Definition: In particular, an AR model is called a Fully Visible Sigmoid Belief Network when

1. Each variable is binary (Bernoulli).
2. Each conditional probability is modeled as a sigmoid function of a weighted sum of all previous variables.

A FVSBN is a probabilistic graphical model where all variables are observed (fully visible) and connected in a DAG according to an autoregressive ordering.

Example: For a black-white image with 28×28 pixels (so $d = 784$), the joint distribution can be written autoregressively as

$$p(x_1, \ldots, x_{784}) = p_{\text{CPT}}(x_1; \alpha^1) p_{\text{logit}}(x_2 \mid x_1, \boldsymbol{\alpha}^2) \cdots p_{\text{logit}}(x_d \mid x_1, \ldots, x_{d-1}, \boldsymbol{\alpha}^d),$$

Here, each conditional variables $X_i \mid x_1, \ldots, x_{i-1}$ is Bernoulli with parameter

$$\begin{aligned}
\hat{x}_i &= p(X_i = 1 \mid x_1, \ldots, x_{i-1}; \boldsymbol{\alpha}^i) \\
&= p(X_i = 1 \mid \boldsymbol{x}_{<i}; \boldsymbol{\alpha}^i) \\
&= \sigma(\alpha_0^i + \sum_{j=1}^{i-1} \alpha_j^i x_j)
\end{aligned}$$

and $\boldsymbol{\alpha}^i = (\alpha_0^i, \alpha_1^i, \ldots, \alpha_{i-1}^i)$.

Thus the conditional distributions are simply

$$\begin{aligned}
&p_{\text{CPT}}(X_1 = 1; \alpha^1) = \alpha^1 \\
&p_{\text{CPT}}(X_1 = 0; \alpha^1) = 1 - \alpha^1 \\
&p_{\text{logit}}(X_2 = 1 \mid x_1; \boldsymbol{\alpha}^2) = \sigma(\alpha_0^2 + \alpha_1^2 x_1) \\
&p_{\text{logit}}(X_2 = 0 \mid x_1; \boldsymbol{\alpha}^2) = 1 - \sigma(\alpha_0^2 + \alpha_1^2 x_1) \\
&\ldots \\
&p_{\text{logit}}(X_d = 1 \mid x_1, \ldots, x_{d-1}; \boldsymbol{\alpha}^d) = \sigma(\alpha_0^d + \sum_{j=1}^{d-1} \alpha_j^d x_j) \\
&p_{\text{logit}}(X_d = 0 \mid x_1, \ldots, x_{d-1}; \boldsymbol{\alpha}^d) = 1 - \sigma(\alpha_0^d + \sum_{j=1}^{d-1} \alpha_j^d x_j)
\end{aligned}$$

The total number of parameters is $1 + 2 + \cdots + d \approx \frac{d^2}{2}$.

2.4 Neural Autoregressive Density Estimation (NADE)

Neural Autoregressive Density Estimation extends the FVSBN model by replacing the linear component of the logistic regression with a single-layer neural network, while still maintaining a sigmoid output layer. Specifically,

$$\begin{aligned}\hat{x}_i &= p(X_i = 1 \mid x_1, \ldots, x_{i-1}; A_i, \boldsymbol{c}_i, \boldsymbol{\alpha}_i, b_i) \\ &= p(X_i = 1 \mid \boldsymbol{x}_{<i}; A_i, \boldsymbol{c}_i, \boldsymbol{\alpha}_i, b_i) \\ &= \sigma(\boldsymbol{\alpha}_i \boldsymbol{h}_i + b_i)\end{aligned}$$

Here,

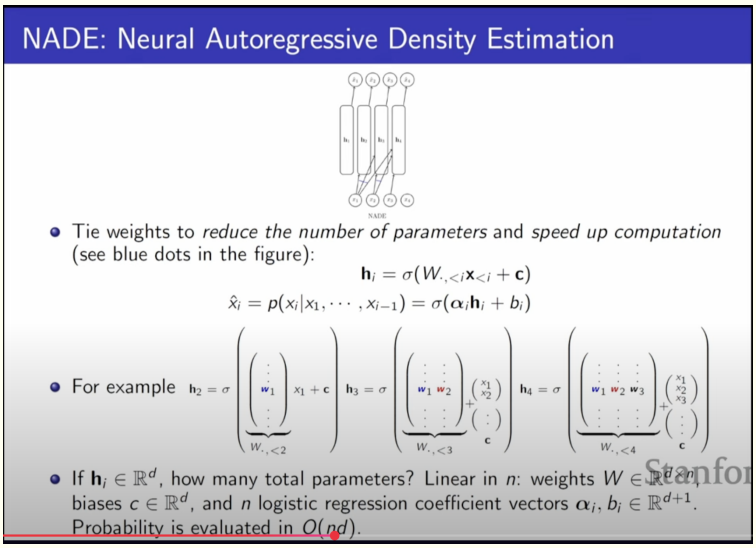
1. $\boldsymbol{x}_{<i} \in \mathbb{R}^{(i-1)\times 1}$ is vector of previous variables $x_1, \ldots, x_{i-1}$.
2. $A_i \in \mathbb{R}^{H\times(i-1)}$ is weight matrix connecting inputs $\boldsymbol{x}_{<i}$ to hidden units.
3. $\boldsymbol{c}_i \in \mathbb{R}^{H\times 1}$ is bias vector for hidden layer.
4. $\boldsymbol{h}_i \in \mathbb{R}^{H\times 1}$ is hidden layer activations, i.e.,

$$\boldsymbol{h}_i = \sigma(A_i \boldsymbol{x}_{<i} + \boldsymbol{c}_i) = \sigma\left(A_i \begin{pmatrix} x_1 \\ \vdots \\ x_{i-1} \end{pmatrix} + \boldsymbol{c}_i\right)$$

5. $\boldsymbol{\alpha}_i \in \mathbb{R}^{1\times H}$ is output weight vector.
6. $b_i \in \mathbb{R}$ is output bias.

Instead of having a separate weight matrix A_i for each i, one can use a shared weight matrix $W \in \mathbb{R}^{H\times d}$. For each i, only the first $i-1$ columns are used. And also shared $\boldsymbol{c} \in \mathbb{R}^{H\times 1}$ is bias vector for hidden layer.

$$\boldsymbol{h}_i = \sigma(W_{\cdot,<i} \boldsymbol{x}_{<i} + \boldsymbol{c})$$



2.5 General Discrete Distributions

General discrete distributions

How to model non-binary discrete random variables $X_i \in \{1, \cdots, K\}$? E.g., pixel intensities varying from 0 to 255

One solution: Let $\hat{\boldsymbol{x}}_i$ parameterize a categorical distribution

$$\mathbf{h}_i = \sigma(W_{\cdot,<i}\mathbf{x}_{<i} + \mathbf{c})$$
$$p(x_i|x_1, \cdots, x_{i-1}) = Cat(p_i^1, \cdots, p_i^K)$$
$$\hat{\boldsymbol{x}}_i = (p_i^1, \cdots, p_i^K) = softmax(A_i\mathbf{h}_i + \mathbf{b}_i)$$

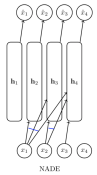
Softmax generalizes the sigmoid/logistic function $\sigma(\cdot)$ and transforms a vector of K numbers into a vector of K *probabilities* (non-negative, sum to 1).

$$softmax(\mathbf{a}) = softmax(a^1, \cdots, a^K) = \left(\frac{\exp(a^1)}{\sum_i \exp(a^i)}, \cdots, \frac{\exp(a^K)}{\sum_i \exp(a^i)}\right)$$

In numpy: `np.exp(a)/np.sum(np.exp(a))`

2.6 Real-valued Neural Autoregressive Density Estimator (RNADE)

RNADE



How to model continuous random variables $X_i \in \mathbb{R}$? E.g., speech signals
Solution: let $\hat{\boldsymbol{x}}_i$ parameterize a continuous distribution
E.g., In a mixture of K Gaussians,

$$p(x_i|x_1, \cdots, x_{i-1}) = \sum_{j=1}^{K} \frac{1}{K} \mathcal{N}(x_i; \mu_i^j, \sigma_i^j)$$

$$\mathbf{h}_i = \sigma(W_{\cdot,<i}\mathbf{x}_{<i} + \mathbf{c})$$

$$\hat{\boldsymbol{x}}_i = (\mu_i^1, \cdots, \mu_i^K, \sigma_i^1, \cdots, \sigma_i^K) = f(\mathbf{h}_i)$$

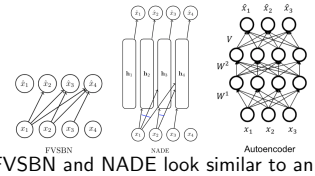
$\hat{\boldsymbol{x}}_i$ defines the mean and standard deviation of each of the K Gaussians (μ_i^j, σ_i^j).
Can use exponential $\exp(\cdot)$ to ensure non-negativity

Stefano Ermon (AI Lab) Deep Generative Models Lecture 3 14 / 36

2.7 Other

Autoregressive Models vs. Autoencoders

Autoregressive models vs. autoencoders



- On the surface, FVSBN and NADE look similar to an **autoencoder**:
- an **encoder** $e(\cdot)$. E.g., $e(x) = \sigma(W^2(W^1 x + b^1) + b^2)$
- a **decoder** such that $d(e(x)) \approx x$. E.g., $d(h) = \sigma(Vh + c)$.
- Loss function for dataset $\mathcal{D}$

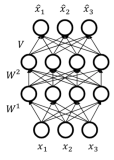
$$\text{Binary r.v.:} \quad \min_{W^1, W^2, b^1, b^2, V, c} \sum_{x \in \mathcal{D}} \sum_i -x_i \log \hat{x}_i - (1 - x_i) \log(1 - \hat{x}_i)$$

$$\text{Continuous r.v.:} \quad \min_{W^1, W^2, b^1, b^2, V, c} \sum_{x \in \mathcal{D}} \sum_i (x_i - \hat{x}_i)^2$$

- e and d are constrained so that we don't learn identity mappings. Hope that $e(x)$ is a meaningful, compressed representation of x (feature learning)
- A vanilla autoencoder is *not* a generative model: it does not define a distribution over x we can sample from to generate new data points.

Stefano Ermon (AI Lab) Deep Generative Models Lecture 3 15 / 36

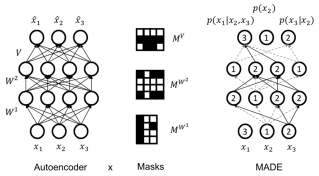
Autoregressive autoencoders



- On the surface, FVSBN and NADE look similar to an **autoencoder**. Can we get a generative model from an autoencoder?
- We need to make sure it corresponds to a valid Bayesian Network (DAG structure), i.e., we need an *ordering* for chain rule. If ordering is $1, 2, 3$, then:
  - $\hat{x}_1$ cannot depend on any input $x = (x_1, x_2, x_3)$. Then at generation time we don't need any input to get started
  - $\hat{x}_2$ can only depend on x_1
  - $\cdots$
- **Bonus**: we can use a single neural network (with n inputs and outputs) to produce all the parameters $\hat{x}$ in a single pass. In contrast, NADE requires n passes. Much more efficient on modern hardware.

Stefano Ermon (AI Lab) Deep Generative Models Lecture 3 16 / 36

MADE: Masked Autoencoder for Distribution Estimation

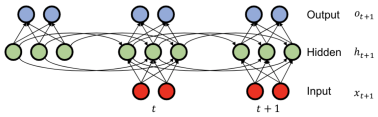


1. **Challenge**: An autoencoder that is autoregressive (DAG structure)
2. **Solution**: use masks to disallow certain paths (Germain et al., 2015). Suppose ordering is x_2, x_3, x_1, so $p(x_1, x_2, x_3) = p(x_2)p(x_3 \mid x_2)p(x_1 \mid x_2, x_3)$.
   1. The unit producing the parameters for $\hat{x}_2 = p(x_2)$ is not allowed to depend on any input. Unit for $p(x_3|x_2)$ only on x_2. And so on...
   2. For each unit in a hidden layer, pick a random integer i in $[1, n-1]$. That unit is allowed to depend only on the first i inputs (according to the chosen ordering).
   3. Add mask to preserve this invariant: connect to all units in previous layer with smaller or equal assigned number (strictly $<$ in final layer)

Autoregressive Models vs. RNNs

RNN: Recurrent Neural Nets

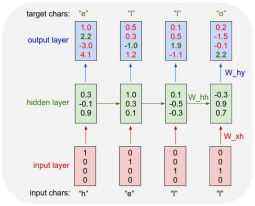
Challenge: model $p(x_t|x_{1:t-1}; \alpha^t)$. "History" $x_{1:t-1}$ keeps getting longer.
Idea: keep a summary and recursively update it



$$\begin{aligned} \text{Summary update rule: } h_{t+1} &= tanh(W_{hh}h_t + W_{xh}x_{t+1}) \\ \text{Prediction: } o_{t+1} &= W_{hy}h_{t+1} \\ \text{Summary initalization: } h_0 &= \boldsymbol{b}_0 \end{aligned}$$

1. Hidden layer h_t is a summary of the inputs seen till time t
2. Output layer o_{t-1} specifies parameters for conditional $p(x_t \mid x_{1:t-1})$
3. Parameterized by $\boldsymbol{b}_0$ (initialization), and matrices W_{hh}, W_{xh}, W_{hy}. Constant number of parameters w.r.t n!

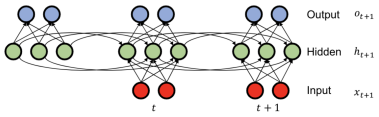
Example: Character RNN (from Andrej Karpathy)



1. Suppose $x_i \in \{h, e, l, o\}$. Use one-hot encoding:
   - h encoded as $[1, 0, 0, 0]$, e encoded as $[0, 1, 0, 0]$, etc.
2. **Autoregressive**: $p(x = hello) = p(x_1 = h)p(x_2 = e|x_1 = h)p(x_3 = l|x_1 = h, x_2 = e) \cdots p(x_5 = o|x_1 = h, x_2 = e, x_3 = l, x_4 = l)$
3. For example,

$$\begin{aligned} p(x_2 = e|x_1 = h) &= softmax(o_1) = \frac{\exp(2.2)}{\exp(1.0) + \cdots + \exp(4.1)} \\ o_1 &= W_{hy}h_1 \\ h_1 &= tanh(W_{hh}h_0 + W_{xh}x_1) \end{aligned}$$

RNN: Recurrent Neural Nets



Pros:

1. Can be applied to sequences of arbitrary length.
2. Very general: For every computable function, there exists a finite RNN that can compute it

Cons:

1. Still requires an ordering
2. Sequential likelihood evaluation (very slow for training)
3. Sequential generation (unavoidable in an autoregressive model)

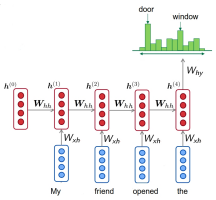
Example: Character RNN (from Andrej Karpathy)

Train 3-layer RNN with 512 hidden nodes on all the works of Shakespeare. Then sample from the model:

> KING LEAR: O, if you were a feeble sight, the courtesy of your law,
> Your sight and several breath, will wear the gods
> With his heads, and my hands are wonder'd at the deeds,
> So drop upon your lordship's head, and your opinion
> Shall be against your honour.

Note: generation happens **character by character**. Needs to learn valid words, grammar, punctuation, etc.

Issues with RNNs



Issues with RNN models

- A single hidden vector needs to summarize all the (growing) history. For example, $h^{(4)}$ needs to summarize the meaning of "My friend opened the".
- Sequential evaluation, cannot be parallelized
- Exploding/vanishing gradients when accessing information from many steps back

RNNs can be autoregressive. E.g., when generating text, RNNs use the previous token to predict the next. That

makes them autoregressive in behavior. Transformers (like GPT) are also autoregressive models. They generate text one token at a time, using previous tokens as context.

Comparison of AR, AE, and RNN

Model	**AR (Autoregressive)**	**AE (Autoencoder)**	**RNN (Recurrent Neural Network)**
Goal	Model the joint distribution by predicting each element from past elements	Learn a compressed latent representation and reconstruct the input	Model sequential dependencies using a recurrent hidden state
Input	Past values or past tokens	Entire input vector at once	Input fed one step at a time (sequence)
Memory	Yes (explicit dependence on past tokens)	No explicit memory mechanism	Yes (hidden state stores past information)
Output	Next value/token in the sequence	Reconstructed input	Sequence outputs (one per time step or final state)
Example Use	Language modeling, time series forecasting	Denoising, dimensionality reduction, anomaly detection	Machine translation, speech recognition, sequence classification
Training	Self-supervised (predict next token from context)	Self-supervised / unsupervised (reconstruction loss)	Typically supervised, but can be self-supervised (e.g., language modeling)

Summary of Autoregressive Models

- Sampling: slow (sequential, cannot be parallelized)
  - Sample $\bar{x}_0 \sim p(x_0)$
  - Sample $\bar{x}_1 \sim p(x_1 \mid x_0 = \bar{x}_0)$
  - Sample $\bar{x}_2 \sim p(x_1 \mid x_0 = \bar{x}_0, x_1 = \bar{x}_1)$
  - $\cdots$
- Density estimation (for training): fast (can be parallelized)
  - Compute $p(x_0 = \bar{x}_0)$
  - Compute $p(x_1 = \bar{x}_1 \mid x_0 = \bar{x}_0)$
  - Compute $p(x_2 = \bar{x}_2 \mid x_0 = \bar{x}_0, x_1 = \bar{x}_1)$
  - $\cdots$
  - Multiply all terms together (or sum their logarithms), i.e.,

$$p(\boldsymbol{x} = \bar{\boldsymbol{x}}) = p(x_0 = \bar{x}_0)p(x_1 = \bar{x}_1 \mid x_0 = \bar{x}_0)p(x_2 = \bar{x}_2 \mid x_0 = \bar{x}_0, x_1 = \bar{x}_1)\cdots$$

$$\Leftrightarrow$$

$$\log p(\boldsymbol{x} = \bar{\boldsymbol{x}}) = \log p(x_0 = \bar{x}_0) + \log p(x_1 = \bar{x}_1 \mid x_0 = \bar{x}_0) + \log p(x_2 = \bar{x}_2 \mid x_0 = \bar{x}_0, x_1 = \bar{x}_1) + \cdots$$

- Easy to extend to continuous variables. E.g.,
  1. Gaussian conditional

$$p(x_t \mid \boldsymbol{x}_{<t}) = \mathcal{N}(\mu_\omega(\boldsymbol{x}_{<t}), \sigma_\omega(\boldsymbol{x}_{<t}))$$

2. Mixture of logistics

3 Maximum Likelihood Learning (MLE)

3.1 Introduction

How to Understand Generative Model

- Learning a generative model = Density estimation
- Once the full probability distribution is learned, the mocel can answer any probabilistic inference query.

Given training examples $\{\boldsymbol{x}^{(i)}\}$ drawn from an unknown true data distribution $p_{\text{data}}(\boldsymbol{x})$, the goal of a generative model is to learn a parameterized distribution $p_\theta(\boldsymbol{x})$ that approximates $p_{\text{data}}(\boldsymbol{x})$.

To achieve this, a discrepancy measure $d(p_{\text{data}}(\boldsymbol{x}), p_\theta(\boldsymbol{x}))$ is minimized. This measures how different the model distribution $p_\theta(\boldsymbol{x})$ is from the true distribution $p_{\text{data}}(\boldsymbol{x})$.

Common choices for the discrepancy $d(\cdot)$ include:

- Kullback-Leibler (KL) divergence: likelihood-based generative models
- JS divergence: GANs
- f-divergence: f-GANs

However, because $p_{\text{data}}(\boldsymbol{x})$ is unknown, the discrepancy $d(p_{\text{data}}(\boldsymbol{x}), p_\theta(\boldsymbol{x}))$ cannot be computed exactly and therefore cannot be used directly as a loss function.

Instead, a practical loss function is defined that satisfies two key properties:

1. It can be computed from available data examples $\{\boldsymbol{x}^{(i)}\}$.
2. Minimizing the loss encourages the model distribution $p_\theta(\boldsymbol{x})$ to become closer to the true data distribution $p_{\text{data}}(\boldsymbol{x})$, thereby reducing the underlying discrepancy $d(p_{\text{data}}(\boldsymbol{x}), p_\theta(\boldsymbol{x}))$.

Relationship Between Loss and KL Divergence

This part explains how the loss function in likelihood-based generative models relates to the KL divergence, thereby justifying its form by showing that the loss satisfies the two desired properties.

For likelihood-based generative models (e.g., autoregressive model, VAEs, normalizing flows), the loss function is defined as the negative log-likelihood of the training data,

$$\mathcal{L}(\theta) = -\sum_{i=1}^{n} \log p_\theta(\boldsymbol{x}^{(i)})$$

- Minimizing this loss encourages the model distribution p_θ to assign high probability to samples drawn from the true data distribution p_{data}.
- Samples $\boldsymbol{x}$ for which $p_\theta(\boldsymbol{x}) \approx 0$ incur very large penalties because the term $-\log p_\theta(\boldsymbol{x}) \to \infty$ as $p_\theta(\boldsymbol{x}) \to 0$.

By calculation,

$$\begin{aligned} D_{\mathrm{KL}}(p_{\text{data}}(\boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{x})) &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log\left(\frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x})}\right)\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log\left(p_{\text{data}}(\boldsymbol{x})\right) - \log\left(p_\theta(\boldsymbol{x})\right)\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log\left(p_{\text{data}}(\boldsymbol{x})\right)\right] - \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log\left(p_\theta(\boldsymbol{x})\right)\right] \\ &\approx \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log\left(p_{\text{data}}(\boldsymbol{x})\right)\right] + \mathcal{L}(\theta) \\ &= -\mathbb{H}(P_{\text{data}}) + \mathcal{L}(\theta) \end{aligned}$$

Here,

$$\mathbb{H}(P_{\text{data}}) = -\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log\left(p_{\text{data}}(\boldsymbol{x})\right)\right]$$

is the entropy term of the true data distribution p_{data}.

Since the entropy term does not depend on θ,

$$\begin{aligned} \arg\min_\theta \mathcal{L}(\theta) &\approx \arg\min_\theta \left(D_{\mathrm{KL}}(p_{\text{data}}(\boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{x})) - \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log\left(p_{\text{data}}(\boldsymbol{x})\right)\right]\right) \\ &= \arg\min_\theta D_{\mathrm{KL}}(p_{\text{data}}(\boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{x})) \end{aligned}$$

However, we cannot directly measure how close we are to the true optimum, since the optimization ignores the entropy term $\mathbb{H}(P_{\text{data}})$. Because this term is unknown and cannot be computed, the absolute value of the KL divergence and therefore the actual distance to the optimum remains unobservable.

Minimizing loss function (i.e., Maximizing Log-likelihood) = Minimizing KL divergence

$$\arg\min_\theta \mathcal{L}(\theta) = \arg\max_\theta \sum_{i=1}^{n} \log p_\theta(\boldsymbol{x}^{(i)}) = \arg\min_\theta D_{\mathrm{KL}}(p_{\text{data}}(\boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{x}))$$

KL Divergence

The KL divergence is defined as,

$$D_{\mathrm{KL}}(p_{\text{data}}(\boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{x})) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log\left(\frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x})}\right)\right]$$

The properties of KL divergence:

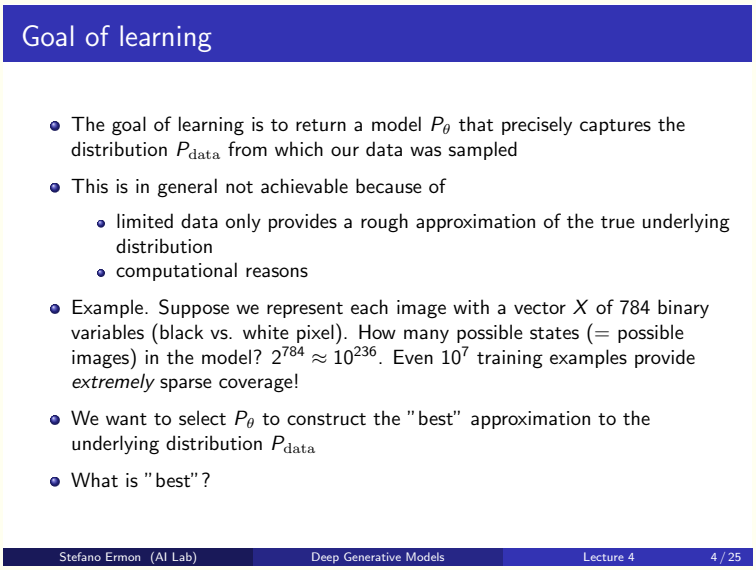
- Non-negative:

$$\begin{aligned}
D_{\mathrm{KL}}(p_{\mathrm{data}}(\boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{x})) &= \mathbb{E}_{\boldsymbol{x}\sim p_{\mathbf{data}}}\left[\log\left(\frac{p_{\mathrm{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x})}\right)\right] \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{\mathbf{data}}}\left[-\log\left(\frac{p_\theta(\boldsymbol{x})}{p_{\mathrm{data}}(\boldsymbol{x})}\right)\right] \\
&\geq \mathbb{E}_{\boldsymbol{x}\sim p_{\mathbf{data}}}\left[1-\frac{p_\theta(\boldsymbol{x})}{p_{\mathrm{data}}(\boldsymbol{x})}\right] \\
&= 1-\mathbb{E}_{\boldsymbol{x}\sim p_{\mathbf{data}}}\left[\frac{p_\theta(\boldsymbol{x})}{p_{\mathrm{data}}(\boldsymbol{x})}\right] \\
&= 1-\int \frac{p_\theta(\boldsymbol{x})}{p_{\mathrm{data}}(\boldsymbol{x})}\cdot p_{\mathrm{data}}(\boldsymbol{x})\,d\boldsymbol{x} \\
&= 1-\int p_\theta(\boldsymbol{x})\,d\boldsymbol{x} \\
&= 1-1 \\
&= 0
\end{aligned}$$

  Note: $\ln x \leq x-1$.

- Non-symmetric: $D_{\mathrm{KL}}(P_{\mathrm{data}} \,\|\, P_\theta) \neq D_{\mathrm{KL}}(P_\theta \,\|\, P_{\mathrm{data}})$
- $D_{\mathrm{KL}}(P_{\mathrm{data}} \,\|\, P_\theta) = 0$ iff the two distributions are the same.
- It measures the compression loss (in bits) of using P_θ to encode data that actually comes from P_{data}.

Goal of Learning for Generative Model



What is "best"?

This depends on what we want to do

1. Density estimation: we are interested in the full distribution (so later we can compute whatever conditional probabilities we want)
2. Specific prediction tasks: we are using the distribution to make a prediction
   - Is this email spam or not?
   - **Structured prediction:** Predict next frame in a video, or caption given an image
3. Structure or knowledge discovery: we are interested in the model itself
   - How do some genes interact with each other?
   - What causes cancer?
   - Take CS 228

Properties of the Monte Carlo Estimate

Properties of the Monte Carlo Estimate

- **Unbiased:**

$$E_P[\hat{g}] = E_P[g(x)]$$

- **Convergence:** By law of large numbers

$$\hat{g} = \frac{1}{T}\sum_{t=1}^{T} g(x^t) \rightarrow E_P[g(x)] \text{ for } T \rightarrow \infty$$

- **Variance:**

$$V_P[\hat{g}] = V_P\left[\frac{1}{T}\sum_{t=1}^{T} g(x^t)\right] = \frac{V_P[g(x)]}{T}$$

Thus, variance of the estimator can be reduced by increasing the number of samples.

3.2 MLE for Autoregressive Models

Define the probability function for an autoregressive model with n variables,

$$p_\theta(\boldsymbol{x}) = \prod_{i=1}^{n} p_{\text{neural}}(x_i \mid \boldsymbol{x}_{<i}; \theta_i),$$

with $\theta = (\theta_1, \ldots, \theta_n)$ are parameters of all the conditionals.

Let $\mathcal{D} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$. Define the likelihood function,

$$L(\theta, \mathcal{D}) = \prod_{j=1}^{m} P_\theta(\boldsymbol{x}^{(j)}) = \prod_{j=1}^{m} \prod_{i=1}^{n} P_{\text{neutral}}(x_i^{(j)} \mid \boldsymbol{x}_{<i}^{(j)}; \theta_i)$$

Define

$$\ell(\theta) = \log L(\theta, \mathcal{D})$$

The optimization turns from $\max_\theta L(\theta, \mathcal{D})$ to $\max_\theta \ell(\theta)$, that is,

$$\begin{aligned} \arg\max_\theta \ell(\theta) &= \arg\max_\theta \log L(\theta, \mathcal{D}) \\ &= \arg\max_\theta \sum_{j=1}^{m} \sum_{i=1}^{n} \log P_{\text{neutral}}(x_i^{(j)} \mid \boldsymbol{x}_{<i}^{(j)}; \theta_i) \end{aligned}$$

1. Gradient descent
   (a) Initialize $\theta^0 = (\theta_1, \ldots, \theta_n)$
   (b) Compute $\nabla_\theta \ell(\theta)$ (by back propagation)
   (c) $\theta^{t+1} = \theta^t + \alpha_t \nabla_\theta \ell(\theta)$
2. Stochastic gradient descent

MLE Learning: Stochastic Gradient Descent

$$\ell(\theta) = \log L(\theta, \mathcal{D}) = \sum_{j=1}^{m} \sum_{i=1}^{n} \log p_{\text{neural}}(x_i^{(j)} | \mathbf{x}_{<i}^{(j)}; \theta_i)$$

1. Initialize θ^0 at random
2. Compute $\nabla_\theta \ell(\theta)$ (by back propagation)
3. $\theta^{t+1} = \theta^t + \alpha_t \nabla_\theta \ell(\theta)$

What is the gradient with respect to θ_i?

$$\nabla_{\theta_i} \ell(\theta) = \sum_{j=1}^{m} \nabla_{\theta_i} \sum_{i=1}^{n} \log p_{\text{neural}}(x_i^{(j)} | \mathbf{x}_{<i}^{(j)}; \theta_i) = \sum_{j=1}^{m} \nabla_{\theta_i} \log p_{\text{neural}}(x_i^{(j)} | \mathbf{x}_{<i}^{(j)}; \theta_i)$$

Each conditional $p_{\text{neural}}(x_i | \mathbf{x}_{<i}; \theta_i)$ can be optimized separately if there is no parameter sharing. In practice, parameters θ_i are shared (e.g., NADE, PixelRNN, PixelCNN, etc.)

Stefano Ermon (AI Lab) Deep Generative Models Lecture 4 18 / 25

MLE Learning: Stochastic Gradient Descent

$$\ell(\theta) = \log L(\theta, \mathcal{D}) = \sum_{j=1}^{m} \sum_{i=1}^{n} \log p_{\text{neural}}(x_i^{(j)} | \mathbf{x}_{<i}^{(j)}; \theta_i)$$

1. Initialize θ^0 at random
2. Compute $\nabla_\theta \ell(\theta)$ (by back propagation)
3. $\theta^{t+1} = \theta^t + \alpha_t \nabla_\theta \ell(\theta)$

$$\nabla_\theta \ell(\theta) = \sum_{j=1}^{m} \sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)} | \mathbf{x}_{<i}^{(j)}; \theta_i)$$

What if $m = |\mathcal{D}|$ is huge?

$$\begin{aligned} \nabla_\theta \ell(\theta) &= m \sum_{j=1}^{m} \frac{1}{m} \sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)} | \mathbf{x}_{<i}^{(j)}; \theta_i) \\ &= m E_{x^{(j)} \sim \mathcal{D}} \left[\sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)} | \mathbf{x}_{<i}^{(j)}; \theta_i) \right] \end{aligned}$$

Monte Carlo: Sample $x^{(j)} \sim \mathcal{D}$; $\nabla_\theta \ell(\theta) \approx m \sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)} | \mathbf{x}_{<i}^{(j)}; \theta_i)$

3.3 Bias and Variance

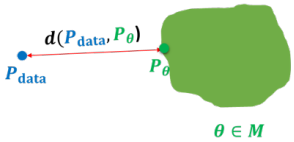
Bias-Variance trade off

- If the hypothesis space is very limited, it might not be able to represent P_{data}, even with unlimited data
  - This type of limitation is called **bias**, as the learning is limited on how close it can approximate the target distribution
- If we select a highly expressive hypothesis class, we might represent better the data
  - When we have small amount of data, multiple models can fit well, or even better than the true model. Moreover, small perturbations on $\mathcal{D}$ will result in very different estimates
  - This limitation is call the **variance**.

How to avoid overfitting?

- Hard constraints, e.g. by selecting a less expressive model family:
  - Smaller neural networks with less parameters
  - Weight sharing

$\mathbf{x}_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$



Model family

- Soft preference for "simpler" models: **Occam Razor**.
- Augment the objective function with **regularization**:

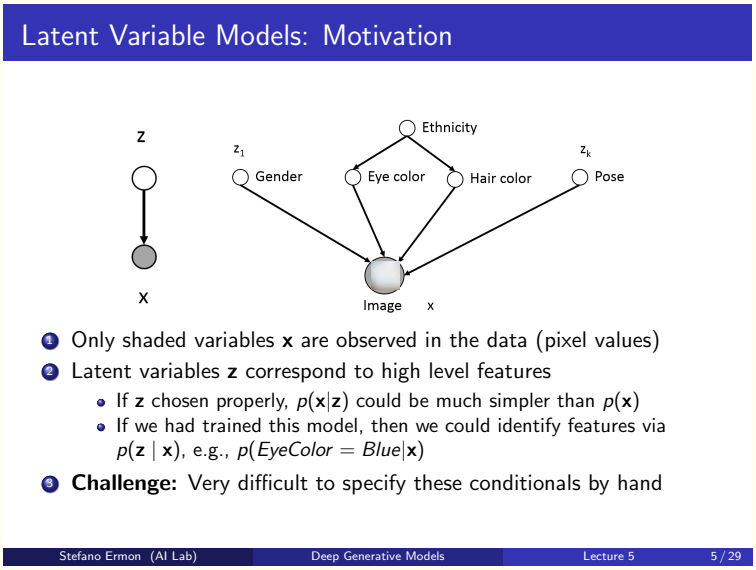
$$objective(\mathbf{x}, \mathcal{M}) = loss(\mathbf{x}, \mathcal{M}) + R(\mathcal{M})$$

- Evaluate generalization performance on a held-out validation set

4 Latent Variable Models (VAEs)

4.1 Latent Variable Models

Motivation



Definition: A latent variable model is a probabilistic model, where

1. The data observed $\boldsymbol{x}$ is assumed to be generated from hidden (latent) variables $\boldsymbol{z}$.
2. These latent variables capture underlying structure or factors in the data.

Basic structure:

$$p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{z})p(\boldsymbol{x} \mid \boldsymbol{z})$$

Here, $p(\boldsymbol{z})$ is prior over latent variables, and $p(\boldsymbol{x} \mid \boldsymbol{z})$ is likelihood (how data $\boldsymbol{x}$ is generated from $\boldsymbol{z}$)).

1. Allow us to define complex models $p(\boldsymbol{x})$ in terms of simpler building blocks $p(\boldsymbol{x} \mid \boldsymbol{z})$.
2. Natural for unsupervised learning tasks (clustering, unsupervised representation learning, etc.)
3. No free lunch: much more difficult to learn compared to fully observed, autoregressive models

Example:

1. A Mixture of Gaussians (or Gaussian Mixture Model (GMM)) is a shallow (non-deep, only one level of latent variables) latent-variable model in which a discrete latent variable selects one of K Gaussian components:

   $$\boldsymbol{z} \sim \text{Categorical}(\pi_1, \ldots, \pi_K)$$

   $$\boldsymbol{x} \mid \boldsymbol{z} = k \sim \mathcal{N}(\mu_k, \Sigma_k)$$

   Here, Σ_k is covariance matrix.

The marginal distribution is obtained by summing over the latent variable:

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x}, \boldsymbol{z}) = \sum_{k=1}^{K} p(\boldsymbol{z} = k) p(\boldsymbol{x} \mid \boldsymbol{z} = k) = \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(\boldsymbol{x}; \mu_k, \Sigma_k)$$

2. The Variational Autoencoder (VAE) is a type of latent variable model.

4.2 Notations of VAEs

Notation	Explanation
$\boldsymbol{x} = (x_1, \ldots, x_D) \in \mathbb{R}^D$	Observed data (e.g., images, texts, etc.)
$\boldsymbol{z} = (z_1, \ldots, z_L) \in \mathbb{R}^L$	Latent (unobserved) variable representing hidden structure or features
θ	Parameters of the decoder (generative model) • It is used in the likelihood $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ and sometimes the prior $p_\theta(\boldsymbol{z})$ (e.g., when $p_\theta(\boldsymbol{z}) \approx \mathcal{N}(\boldsymbol{z} \mid 0, I)$).
ϕ	Parameters of the encoder (inference model) • It is used in the approximate posterior $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$.
$p_\theta(\boldsymbol{x})$ or $p(\boldsymbol{x})$	Marginal likelihood (or true likelihood) of the observed data • It's called marginal because it is obtained by integrating out (marginalizing over) the latent variable $\boldsymbol{z}$: $$p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}, \boldsymbol{z})\, d\boldsymbol{z} = \int p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) p_\theta(\boldsymbol{z})\, d\boldsymbol{z} = \mathbb{E}_{p_\theta(\boldsymbol{z})}[p_\theta(\boldsymbol{x} \mid \boldsymbol{z})]$$ Here, $p_\theta(\boldsymbol{x}, \boldsymbol{z})$ is the joint distribution of observed and latent variables, which is also parameterized by θ since $$p_\theta(\boldsymbol{x}, \boldsymbol{z}) = p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) p_\theta(\boldsymbol{z})$$ • It is intractable to compute exactly, because $\boldsymbol{z}$ is high-dimensional.
$\log p_\theta(\boldsymbol{x})$	Marginal log-likelihood (or true log-likelihood) of the observed data • It is approximated using variational lower bounds (e.g., ELBO, IWAE bound, etc), since it is intractable to compute exactly.
$p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$	Likelihood (or decoder), parameterized by θ • It models how the observed data $\boldsymbol{x}$ is generated given the latent variable $\boldsymbol{z}$. • It typically has a simpler form than the marginal likelihood $p_\theta(\boldsymbol{x})$. • Parameters are obtained from the decoder neural network with parameters θ.
$p_\theta(\boldsymbol{z})$	Prior distribution over the latent variable, sometimes parameterized by θ
$p_\theta(\boldsymbol{z} \mid \boldsymbol{x})$	True posterior distribution of the latent variable $\boldsymbol{z}$ given the observation $\boldsymbol{x}$ • It is defined as $$p_\theta(\boldsymbol{z} \mid \boldsymbol{x}) = \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{p_\theta(\boldsymbol{x})} = \frac{p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) p_\theta(\boldsymbol{z})}{\int p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) p_\theta(\boldsymbol{z})\, d\boldsymbol{z}}$$ • This posterior characterizes how likely each latent variable $\boldsymbol{z}$ is, given the observation $\boldsymbol{x}$. • Although it is the quantity we aim to infer, it is generally intractable and thus approximated by a variational distribution $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ (as in VAE).
$q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$	Approximate posterior (or encoder), parameterized by ϕ • It approximates the true posterior $p_\theta(\boldsymbol{z} \mid \boldsymbol{x})$. • Parameters (e.g., mean and log-variance) are obtained from the encoder neural network with parameters ϕ.
$\mathcal{L}_m(\boldsymbol{x}; \theta, \phi)$	IWAE bound, a tighter lower bound on $\log p_\theta(\boldsymbol{x})$ that uses m importance-weighted samples
m (sometimes iw)	Number of Monte Carlo samples $\boldsymbol{z}^{(i)} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ used per data point to estimate the IWAE bound
$w_i = \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z}^{(i)})}{q_\phi(\boldsymbol{z}^{(i)} \mid \boldsymbol{x})}$, $i = 1, \ldots, m$	Importance weights used to reweight samples in the IWAE objective

4.3 Evidence Lower Bound (ELBO)

Likelihood Intractability

The marginal likelihood is

$$p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) p_\theta(\boldsymbol{z})\, d\boldsymbol{z}$$

However, this integral is generally intractable, because the latent $\boldsymbol{z}$ is high-dimensional and the decoder $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ is nonlinear (e.g., a neural network).

If the true posterior $p_\theta(\boldsymbol{z} \mid \boldsymbol{x})$ were known, the marginal likelihood can be rewritten as an expectation over the posterior,

$$\begin{aligned} p_\theta(\boldsymbol{x}) &= \int \frac{p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) p_\theta(\boldsymbol{z})}{p_\theta(\boldsymbol{z} \mid \boldsymbol{x})} \cdot p_\theta(\boldsymbol{z} \mid \boldsymbol{x})\, d\boldsymbol{z} \\ &= \mathbb{E}_{\boldsymbol{z} \sim p_\theta(\boldsymbol{z} \mid \boldsymbol{x})} \left[\frac{p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) p_\theta(\boldsymbol{z})}{p_\theta(\boldsymbol{z} \mid \boldsymbol{x})} \right] \end{aligned}$$

In principle, this expectation could be approximated via Monte Carlo sampling if able to sample from the true posterior $p_\theta(\boldsymbol{z} \mid \boldsymbol{x})$. However, sampling from the true posterior is itself intractable, because depends on unknown $p_\theta(\boldsymbol{x})$.

Variational Inference

To address this, a tractable approximate posterior (encoder) $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ is introduced. Introducing a tractable approximate posterior is known as variational inference (VI).

By calculation,

$$\begin{aligned} \log p_\theta(\boldsymbol{x}) &= \log p_\theta(\boldsymbol{x}) - \mathbb{E}_{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})} \left[\log \left(\frac{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}{p_\theta(\boldsymbol{z} \mid \boldsymbol{x})} \right) \right] + \mathbb{E}_{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})} \left[\log \left(\frac{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}{p_\theta(\boldsymbol{z} \mid \boldsymbol{x})} \right) \right] \\ &= \mathbb{E}_{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})} \left[\log p_\theta(\boldsymbol{x}) - \log q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) + \log p_\theta(\boldsymbol{z} \mid \boldsymbol{x}) \right] + \mathbb{E}_{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})} \left[\log \left(\frac{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}{p_\theta(\boldsymbol{z} \mid \boldsymbol{x})} \right) \right] \\ &= \mathbb{E}_{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})} \left[\log \left(\frac{p_\theta(\boldsymbol{x}) \cdot p_\theta(\boldsymbol{z} \mid \boldsymbol{x})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})} \right) \right] + \mathbb{E}_{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})} \left[\log \left(\frac{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}{p_\theta(\boldsymbol{z} \mid \boldsymbol{x})} \right) \right] \\ &= \mathbb{E}_{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})} \left[\log \left(\frac{p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) \cdot p_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})} \right) \right] + D_{\mathrm{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{z} \mid \boldsymbol{x})) \end{aligned}$$

The second term is a KL divergence between the approximate posterior and the true posterior, which measures how well $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ approximates $p_\theta(\boldsymbol{z} \mid \boldsymbol{x})$.

Definition & Property

The Evidence Lower Bound (ELBO) is defined as,

$$\mathcal{L}_{\mathrm{ELBO}}(\boldsymbol{x}; \theta, \phi) = \mathbb{E}_{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})} \left[\log \left(\frac{p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) \cdot p_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})} \right) \right]$$

ELBO forms a lower bound on the true log-likelihood,

$$\log p_\theta(\boldsymbol{x}) \geq \mathcal{L}_{\text{ELBO}}(\boldsymbol{x}; \theta, \phi)$$

In practice, the ELBO is maximized rather than the true log-likelihood, because the true posterior is intractable. The ELBO serves as a tractable, differentiable surrogate objective for learning parameters θ and ϕ. The closer the approximate posterior $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ is to the true posterior $p_\theta(\boldsymbol{z} \mid \boldsymbol{x})$, the smaller the KL divergence $D_{\text{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{z}))$ becomes, and the tighter the ELBO is as an lower bound to the true log-likelihood.

Decomposition & Calculation

ELBO can be rewritten as

$$\begin{aligned}
\mathcal{L}_{\text{ELBO}}(\boldsymbol{x}; \theta, \phi) &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) p_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}\right)\right] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) - \log\left(\frac{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}{p_\theta(\boldsymbol{z})}\right)\right] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\right] - \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}{p_\theta(\boldsymbol{z})}\right)\right] \\
&= \underbrace{\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\right]}_{\text{Reconstruction term}} - \underbrace{D_{\text{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{z}))}_{\text{KL divergence (regularization) term}} \\
&\approx \underbrace{\frac{1}{S}\sum_{s=1}^{S} \log p_\theta(\boldsymbol{x} \mid \boldsymbol{z}^{(s)})}_{\text{Monte Carlo approximation}} - D_{\text{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{z})) \\
&\approx \log p_\theta(\boldsymbol{x} \mid \boldsymbol{z}^{(1)}) - D_{\text{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{z}))
\end{aligned}$$

Here, $\boldsymbol{z}^{(s)} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ for $s = 1, \ldots, S$.

When $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ and $p_\theta(\boldsymbol{z})$ are both Gaussian distributions,

$$q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) = \mathcal{N}(\mu_\phi(\boldsymbol{x}), \text{diag}(\sigma^2_\phi(\boldsymbol{x})))$$
$$p_\theta(\boldsymbol{z}) = \mathcal{N}(0, I)$$

the KL divergence term becomes

$$D_{\text{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{z})) = \frac{1}{2}\sum_{i=1}^{L}\left(\mu_i^2 + \sigma_i^2 - 1 - \log \sigma_i^2\right)$$

which is analytically computable (i.e., with a closed-form expression).

Training Objective

By calculation,

$$-\mathcal{L}_{\text{ELBO}}(\boldsymbol{x}; \theta, \phi) = \underbrace{-\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\right]}_{\text{Reconstruction loss}} + \underbrace{D_{\text{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{z}))}_{\text{KL divergence term}}$$

For a dataset $\mathcal{D}$, the learning objective becomes

$$\begin{aligned}\arg\max_{\theta} \sum_{\boldsymbol{x}^{(i)} \in \mathcal{D}} \log p_\theta(\boldsymbol{x}^{(i)}) &\geq \arg \max_{\theta, \boldsymbol{\phi}^{(1)}, \ldots, \boldsymbol{\phi}^{(n)}} \sum_{i=1}^{n} \mathcal{L}_{\text{ELBO}}(\boldsymbol{x}^{(i)}; \theta, \boldsymbol{\phi}^{(i)}) \\ &= \arg \min_{\theta, \boldsymbol{\phi}^{(1)}, \ldots, \boldsymbol{\phi}^{(n)}} \sum_{i=1}^{n} -\mathcal{L}_{\text{ELBO}}(\boldsymbol{x}^{(i)}; \theta, \boldsymbol{\phi}^{(i)})\end{aligned}$$

Note that use different set of variational parameters $\boldsymbol{\phi}^{(i)}$ for every data point $\boldsymbol{x}^{(i)}$.

4.4 β-VAE

Definition: β-VAE is a variation of the normal VAE, which is defined as

$$\mathcal{L}_{\beta\text{-ELBO}}(\boldsymbol{x};\theta,\phi) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\right] - \beta D_{\mathrm{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{z}))$$

Property

In β-VAE, the parameter β controls how much the model focuses on making good reconstructions versus keeping the latent space simple and organized.

1. When $\beta = 1$, β-VAE becomes the normal VAE. The model tries to do two things at once:
   (a) Reconstruct the input data $\boldsymbol{z}$ as accurately as possible, i.e., make the reconstruction term as large as possible,
   $$\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\right]$$
   (b) Make sure the latent variables $\boldsymbol{z}$ follow the prior distribution $p_\theta(\boldsymbol{z})$, i.e., make the KL term as small as possible,
   $$D_{\mathrm{KL}}(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{z}))$$
2. When $\beta > 1$, the term KL term is multiplied by a larger number. The stronger KL penalty forces the encoder distribution $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ to stay close to the prior $p_\theta(\boldsymbol{z})$, which assumes that all latent dimensions are independent and have simple Gaussian structure. As a result, the latent variables become simpler and more independent, meaning that different dimensions of $\boldsymbol{z}$ capture different aspects of the data with less overlap between them. This often leads the model to learn more interpretable and disentangled latent features, i.e., each latent variable tends to represent one clear factor of variation in the data (e.g., color, shape, or position).
   Meanwhile, because the model focuses more on keeping $\boldsymbol{z}$ better structured (simpler, smoother, more disentangled), it cannot devote as much capacity to maximizing the reconstruction term. As a result, the reconstructed data may become less accurate or blurrier.

To summarize, increasing β produces a latent space that is simpler and more disentangled, but this comes at the cost of reduced reconstruction quality.

4.5 Importance Weighted Autoencoder (IWAE) Bound

Definition: Let $\mathcal{L}_m(\boldsymbol{x}; \theta, \phi)$ denote the Importance Weighted Autoencoder (IWAE) bound, which is defined as

$$\begin{aligned}
\mathcal{L}_m(\boldsymbol{x}; \theta, \phi) &= \mathbb{E}_{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)} \overset{\text{i.i.d.}}{\sim} q_\phi(\boldsymbol{z}|\boldsymbol{x})} \left[\log \left(\frac{1}{m} \sum_{i=1}^{m} w_i \right) \right] \\
&= \mathbb{E}_{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)} \overset{\text{i.i.d.}}{\sim} q_\phi(\boldsymbol{z}|\boldsymbol{x})} \left[\log \left(\frac{1}{m} \sum_{i=1}^{m} \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z}^{(i)})}{q_\phi(\boldsymbol{z}^{(i)} \mid \boldsymbol{x})} \right) \right] \\
&\approx \underbrace{\frac{1}{S} \sum_{s=1}^{S} \log \left(\frac{1}{m} \sum_{i=1}^{m} \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z}^{(i,s)})}{q_\phi(\boldsymbol{z}^{(i,s)} \mid \boldsymbol{x})} \right)}_{\text{Monte Carlo approximation}} \\
&\approx \log \left(\frac{1}{m} \sum_{i=1}^{m} \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z}^{(i)})}{q_\phi(\boldsymbol{z}^{(i)} \mid \boldsymbol{x})} \right) \\
&= \log \left(\frac{1}{m} \sum_{i=1}^{m} \frac{p_\theta(\boldsymbol{x} \mid \boldsymbol{z}^{(i)}) p_\theta(\boldsymbol{z}^{(i)})}{q_\phi(\boldsymbol{z}^{(i)} \mid \boldsymbol{x})} \right)
\end{aligned}$$

Here, $z^{(i,s)} \sim q_\phi(\boldsymbol{z} \mid x)$ for $i = 1, \ldots, m$ and $s = 1, \ldots, S$. m is called the number of importance-weighted samples. w_i (for $i = 1, \ldots, m$) denotes the importance weights

$$w_i = \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z}^{(i)})}{q_\phi(\boldsymbol{z}^{(i)} \mid \boldsymbol{x})}$$

Property

1. For any $m \geq 1$,

   $$\log p_\theta(\boldsymbol{x}) \geq \mathcal{L}_m(\boldsymbol{x}; \theta, \phi) \geq \mathcal{L}_1(\boldsymbol{x}; \theta, \phi) = \mathcal{L}_{\text{ELBO}}(\boldsymbol{x}; \theta, \phi)$$

   Thus, the IWAE bound is always a valid lower bound on the true log-likelihood, and using multiple importance samples ($m > 1$) yields a tighter bound than the ELBO.

   - IWAE bound: tighter lower bound obtained with multiple samples.
   - ELBO: lower bound corresponding to a single sample.

2. As the number of importance samples grows,

   $$\lim_{m \to \infty} \mathcal{L}_m(\boldsymbol{x}; \theta, \phi) = \log p_\theta(\boldsymbol{x})$$

   so the IWAE bound converges to the true marginal log-likelihood.

Proof for Property 1

1. Prove that IWAE bound is a lower bound of the true log-likelihood $\log p_\theta(\boldsymbol{x})$, i.e.,

   $$\log p_\theta(\boldsymbol{x}) \geq \mathcal{L}_m(\boldsymbol{x}; \theta, \phi)$$

By calculation,

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(m)} \overset{\text{i.i.d.}}{\sim} q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{1}{m}\sum_{i=1}^{m} w_i\right] &= \mathbb{E}_{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(m)} \overset{\text{i.i.d.}}{\sim} q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{1}{m}\sum_{i=1}^{m} \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z}^{(i)})}{q_\phi(\boldsymbol{z}^{(i)} \mid \boldsymbol{x})}\right] \\
&= \frac{1}{m}\sum_{i=1}^{m} \mathbb{E}_{\boldsymbol{z}^{(i)} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{p_\theta(\boldsymbol{x}, \boldsymbol{z}^{(i)})}{q_\phi(\boldsymbol{z}^{(i)} \mid \boldsymbol{x})}\right] \\
&= \frac{1}{m} \cdot m \cdot \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}\right] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}\right] \\
&= \int \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})} q_\phi(\boldsymbol{z} \mid \boldsymbol{x})\, d\boldsymbol{z} \\
&= \int p_\theta(\boldsymbol{x}, \boldsymbol{z})\, d\boldsymbol{z} \\
&= p_\theta(\boldsymbol{x})
\end{aligned}$$

That is, $\frac{1}{m}\sum_{i=1}^{m} w_i$ is an unbiased estimator of the marginal likelihood $p_\theta(\boldsymbol{x})$.

Furthermore, according to Jensen's inequality,

$$\begin{aligned}
\log p_\theta(\boldsymbol{x}) &= \log\left(\mathbb{E}_{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(m)} \overset{\text{i.i.d.}}{\sim} q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{1}{m}\sum_{i=1}^{m} w_i\right]\right) \\
&\geq \mathbb{E}_{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(m)} \overset{\text{i.i.d.}}{\sim} q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{1}{m}\sum_{i=1}^{m} w_i\right)\right] \\
&= \mathcal{L}_m(\boldsymbol{x}; \theta, \phi)
\end{aligned}$$

2. Prove that

$$\mathcal{L}_m(\boldsymbol{x}; \theta, \phi) \geq \mathcal{L}_1(\boldsymbol{x}; \theta, \phi)$$

Let W be a discrete random variable with values in the set $\{w_1, \ldots, w_m\}$, each with equal probability $\frac{1}{m}$, i.e., $P(W = w_i) = \frac{1}{m}$, for $i = 1, \ldots, m$. By Jensen's inequality,

$$\log\left(\frac{1}{m}\sum_{i=1}^{m} w_i\right) = \log \mathbb{E}[W] \geq \mathbb{E}[\log W] = \frac{1}{m}\sum_{i=1}^{m} \log w_i$$

Therefore, by calculation,

$$\begin{aligned}
\mathcal{L}_m(\boldsymbol{x}; \theta, \phi) &= \mathbb{E}_{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(m)} \overset{\text{i.i.d.}}{\sim} q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{1}{m}\sum_{i=1}^{m} w_i\right)\right] \\
&\geq \mathbb{E}_{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(m)} \overset{\text{i.i.d.}}{\sim} q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{1}{m}\sum_{i=1}^{m} \log w_i\right] \\
&= \frac{1}{m}\sum_{i=1}^{m} \mathbb{E}_{\boldsymbol{z}^{(i)} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log w_i\right] \\
&= \frac{1}{m}\sum_{i=1}^{m} \mathbb{E}_{\boldsymbol{z}^{(1)} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log w_1\right] \\
&= \frac{1}{m} \cdot m \cdot \mathbb{E}_{\boldsymbol{z}^{(1)} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log w_1\right] \\
&= \mathbb{E}_{\boldsymbol{z}^{(1)} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log w_1\right] \\
&= \mathcal{L}_1(\boldsymbol{x}; \theta, \phi)
\end{aligned}$$

3. Prove that

$$\mathcal{L}_1(\boldsymbol{x};\theta,\phi) = \mathcal{L}_{\text{ELBO}}(\boldsymbol{x};\theta,\phi)$$

Let $\mathcal{L}_{\text{ELBO}}$ denote the ELBO, by the definition,

$$\begin{aligned}
\mathcal{L}_{\text{ELBO}}(\boldsymbol{x};\theta,\phi) &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}\mid\boldsymbol{x})}\right)\right] \\
&= \mathbb{E}_{\boldsymbol{z}^{(1)}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{p_\theta(\boldsymbol{x},\boldsymbol{z}^{(1)})}{q_\phi(\boldsymbol{z}^{(1)}\mid\boldsymbol{x})}\right)\right] \\
&= \mathbb{E}_{\boldsymbol{z}^{(1)}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log w_1\right] \\
&= \mathcal{L}_1(\boldsymbol{x};\theta,\phi)
\end{aligned}$$

Proof for Property 2

$$\begin{aligned}
\lim_{m\to\infty}\mathcal{L}_m(\boldsymbol{x};\theta,\phi) &= \mathbb{E}_{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(m)}\overset{\text{i.i.d.}}{\sim}q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{1}{m}\sum_{i=1}^{m}\frac{p_\theta(\boldsymbol{x},\boldsymbol{z}^{(i)})}{q_\phi(\boldsymbol{z}^{(i)}\mid\boldsymbol{x})}\right)\right] \\
&= \mathbb{E}_{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(m)}\overset{\text{i.i.d.}}{\sim}q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}\mid\boldsymbol{x})}\right]\right)\right] \\
&= \mathbb{E}_{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(m)}\overset{\text{i.i.d.}}{\sim}q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\int\frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}\mid\boldsymbol{x})}q_\phi(\boldsymbol{z}\mid\boldsymbol{x})\,d\boldsymbol{z}\right)\right] \\
&= \mathbb{E}_{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(m)}\overset{\text{i.i.d.}}{\sim}q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\int p_\theta(\boldsymbol{x},\boldsymbol{z})\,d\boldsymbol{z}\right)\right] \\
&= \mathbb{E}_{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(m)}\overset{\text{i.i.d.}}{\sim}q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(p_\theta(\boldsymbol{x})\right)\right] \\
&= \log p_\theta(\boldsymbol{x})
\end{aligned}$$

4.6 Assumptions of VAEs

	Variational Auto Encoder (VAE)	**Mixture of Gaussians VAE (GMVAE)**	**Importance Weight Auto Encoder (IWAE)**
Optimization	ELBO	ELBO	IWAE bound (IW-ELBO)
Prior	$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z} \mid 0, I)$	$p_\theta(\boldsymbol{z}) = \sum_{i=1}^{k} \frac{1}{k} \mathcal{N}(\boldsymbol{z} \mid \mu_i, \text{diag}(\sigma_i^2))$	$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z} \mid 0, I)$
Likelihood (decoder)	1. Binary data ($x_i \in \{0, 1\}$): Bernoulli distribution $p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) = \text{Bern}(\boldsymbol{x} \mid f_\theta(\boldsymbol{z}))$ 2. Continuous data ($x_i \in \mathbb{R}$): Gaussian distribution $p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) = \mathcal{N}(\boldsymbol{x} \mid \mu_\theta(\boldsymbol{z}), \Sigma_\theta(\boldsymbol{z})) = \mathcal{N}(\boldsymbol{x} \mid \mu_\theta(\boldsymbol{z}), I)$ Here $\mu_\theta(\boldsymbol{z}) = \sigma(A\boldsymbol{z} + c) \in \mathbb{R}^L$ $\Sigma_\theta(\boldsymbol{z}) = \text{diag}\left(e^{\sigma(B\boldsymbol{z}+d)}\right) \in \mathbb{R}^L$ $\theta = (A, B, c, d)$ 3. Categorical data ($x_i \in \{1, 2, \ldots, K\}$): Categorical distribution $p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) = \text{Categorical}(\boldsymbol{x} \mid f_\theta(\boldsymbol{z}))$		
Approximate posterior (encoder)	$q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{z} \mid \mu_\phi(\boldsymbol{x}), \text{diag}(\sigma_\phi^2(\boldsymbol{x})))$		

4.7 Workflow of VAEs

1. Feed the input, an observation $\boldsymbol{x} \in \mathbb{R}^D$, into the encoder neural network, which outputs two vectors for each input
   (a) The mean vector of the encoder $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$: $\mu_\phi(\boldsymbol{x}) \in \mathbb{R}^L$
   (b) The log-variance vector of the encoder $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$: $\log \sigma_\phi^2(\boldsymbol{x}) \in \mathbb{R}^L$
2. Compute the encoder $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$, i.e.,

$$q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{z} \mid \mu_\phi(\boldsymbol{x}), \text{diag}(\sigma_\phi^2(\boldsymbol{x})))$$

3. Sample $\boldsymbol{z} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$, by reparameterization trick.
   It is impossible to directly backpropagate through the sampling process

$$\boldsymbol{z} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{z} \mid \mu_\phi(\boldsymbol{x}), \text{diag}(\sigma_\phi^2(\boldsymbol{x})))$$

   since this stochastic operation is non-differentiable with respect to the parameters ϕ.
   To make gradient-based optimization possible, the random variable $\boldsymbol{z}$ is rewritten as a deterministic function of a parameter-free random variable ϵ and differentiable parameters $\mu_\phi(\boldsymbol{x})$ and $\sigma_\phi(\boldsymbol{x})$

$$\boldsymbol{z} = \mu_\phi(\boldsymbol{x}) + \sigma_\phi(\boldsymbol{x}) \odot \epsilon$$
$$\epsilon \sim \mathcal{N}(0, I)$$

   Here, $\odot$ represents element-wise product. The randomness is entirely captured by ϵ, which does not depend on ϕ, enabling backpropagation through $\boldsymbol{z}$ during training.
   (a) VAE: One sample $\boldsymbol{z} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$
   (b) IWAE: m samples $\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$
4. Feed $\boldsymbol{z} \in \mathbb{R}^L$ into the decoder neural network $f_\theta(\cdot)$, which output
   (a) Continuous, $x_i \in \mathbb{R}$: The mean vector of the Gaussian likelihood

$$\mu_\theta(\boldsymbol{z}) = (\mu_1, \ldots, \mu_D) = f_\theta(\boldsymbol{z}) \in \mathbb{R}^D$$

   (b) Binary (discrete), $x_i \in \{0, 1\}$: The vector of D logits (representing the raw, unnormalized activations), i.e., $(\alpha_1, \ldots, \alpha_D)$

$$(\alpha_1, \ldots, \alpha_D) = f_\theta(\boldsymbol{z}) \in \mathbb{R}^D$$

   (c) Categorical (discrete), $x_i \in \{1, 2, \ldots, K\}$: The matrix of $D \times K$ logits, i.e.,

$$\boldsymbol{\ell} = f_\theta(\boldsymbol{z}) \in \mathbb{R}^{D \times K}$$

5. Compute the decoder $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$. Specifically,
   (a) Continuous, $x_i \in \mathbb{R}$:

$$p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) = \prod_{i=1}^{D} \mathcal{N}(x_i \mid \mu_i, 1)$$

   (b) Binary, $x_i \in \{0, 1\}$:

$$p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) = \prod_{i=1}^{D} \text{Bern}(x_i \mid p_i)$$

Here, p_i is obtained by applying the sigmoid to logits,

$$p_i = \sigma(\alpha_i) = \frac{1}{1 + e^{-\alpha_i}}$$

(c) Categorical, $x_i \in \{1, 2, \ldots, K\}$:

$$p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) = \prod_{i=1}^{D} \text{Categorical}(x_i \mid \boldsymbol{p}_i) = \prod_{i=1}^{D} \text{Categorical}(x_i \mid (p_{i,1}, \ldots, p_{i,K}))$$

Here, $\boldsymbol{p}_i = (p_{i,1}, \ldots, p_{i,K})$ is obtained by applying the softmax to logits,

$$\boldsymbol{p}_i = (p_{i,1}, \ldots, p_{i,K}) = \left(\frac{e^{\boldsymbol{\ell}_{i,1}}}{\sum_{j=1}^{K} e^{\boldsymbol{\ell}_{i,j}}}, \ldots, \frac{e^{\boldsymbol{\ell}_{i,K}}}{\sum_{j=1}^{K} e^{\boldsymbol{\ell}_{i,j}}} \right)$$

6. Apply $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ to

(a) Approximate (using Mont Carlo) ELBO reconstruction term

$$\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log \log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})] \approx \log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$$

(b) Approximate (using Mont Carlo) IWAE bound

$$\mathcal{L}_m(\boldsymbol{x}; \theta, \phi) \approx \log \left(\frac{1}{m} \sum_{i=1}^{m} \frac{p_\theta(\boldsymbol{x} \mid \boldsymbol{z}^{(i)}) p_\theta(\boldsymbol{z}^{(i)})}{q_\phi(\boldsymbol{z}^{(i)} \mid \boldsymbol{x})} \right)$$

7. Repeat the above process for n observations $\boldsymbol{x}$,

(a) ELBO

$$(\theta^*, \phi^*) = \arg\max_{\theta,\phi} \sum_{i=1}^{n} \mathcal{L}_{\text{ELBO}}(\boldsymbol{x}^{(i)}; \theta, \phi)$$

(b) IWAE bound

$$(\theta^*, \phi^*) = \arg\max_{\theta,\phi} \sum_{i=1}^{n} \mathcal{L}_m(\boldsymbol{x}^{(i)}; \theta, \phi)$$

8. Sampling after model training

(a) Sample a latent from the prior, i.e., $\boldsymbol{z} \sim p_\theta(\boldsymbol{z})$.

(b) Compute the parameters (mean and variance) of the decoder $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ from $\boldsymbol{z}$.

(c) Sample an observation from the decoder. i.e., $\boldsymbol{x} \sim p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$.

5 Normalizing Flows

5.1 Introduction

Change of Variables formula

1. 1-dimensional continuous case
   (a) Let $X, Z \in \mathbb{R}$ be two random variables with probability densities $p_X(x)$ and $p_Z(z)$. Suppose $X = f(Z)$, and $f(\cdot)$ is monotone with inverse $Z = h(X) = f^{-1}(X)$. Then
   $$p_X(x) = p_Z(h(x))\,|h'(x)| = p_Z(z)\frac{1}{|f'(x)|}$$
2. n-dimensional continuous case, linear transformation
   (a) Let $X, Z \in \mathbb{R}^n$ be two random variables with probability densities $p_X(\boldsymbol{x})$ and $p_Z(\boldsymbol{z})$. Suppose $X = AZ$, where A is an invertible square matrix with inverse $W = A^{-1}$. Then
   $$p_X(\boldsymbol{x}) = p_Z(W\boldsymbol{x})\,|\det(W)| = p_Z(\boldsymbol{z})\frac{1}{|\det(A)|}$$
3. n-dimensional continuous case, general transformation
   (a) Let $X, Z \in \mathbb{R}^n$ be two random variables with probability densities $p_X(\boldsymbol{x})$ and $p_Z(\boldsymbol{z})$. Suppose $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$ is invertible such that $X = \boldsymbol{f}(Z)$ and $Z = \boldsymbol{f}^{-1}(X)$. Then
   $$p_X(\boldsymbol{x}) = p_Z(\boldsymbol{f}^{-1}(\boldsymbol{x}))\left|\det\left(\frac{\partial \boldsymbol{f}^{-1}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right| = p_Z(\boldsymbol{z})\left|\det\left(\frac{\partial \boldsymbol{f}(\boldsymbol{z})}{\partial \boldsymbol{z}}\right)\right|^{-1}$$

5.2 Comparison of VAE & NF

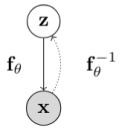
Both Variational Autoencoders (VAE) and Normalizing Flows (NF) consist of an encoder and a decoder. The key distinction is whether the decoder is invertible, which fundamentally impacts the design of the encoder and the computation of the data likelihood.

	VAE	**NF**
Decoder Definition	The decoder is defined as a conditional probability distribution $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$. Typically, a neural network $f_\theta(\boldsymbol{z})$ outputs the parameters (e.g., mean and variance) of this distribution.	The decoder is defined as a deterministic and invertible transformation, $$\boldsymbol{f}_\theta : \boldsymbol{z} \to \boldsymbol{x}, \quad \boldsymbol{x} = \boldsymbol{f}_\theta(\boldsymbol{z})$$
Decoder Feature	The decoder $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ is stochastic and therefore not invertible.	The decoder $\boldsymbol{f}_\theta$ is deterministic and invertible.
Decoder Feature	Given a latent variable $\boldsymbol{z}$, the output $\boldsymbol{x}$ is stochastic, i.e., multiple possible $\boldsymbol{x}$ samples can be drawn from $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$.	Given a latent variable $\boldsymbol{z}$, the output $\boldsymbol{x}$ is deterministic, i.e., one $\boldsymbol{z}$ corresponds to exactly one $\boldsymbol{x}$.
Encoder Definition	Given an observation $\boldsymbol{x}$, there may be multiple possible $\boldsymbol{z}$ values. Therefore, the encoder is defined probabilistically, $$\boldsymbol{z} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$$	Given an observation $\boldsymbol{x}$, there is exactly one corresponding $\boldsymbol{z}$. The encoder is defined deterministically, $$\boldsymbol{z} = \boldsymbol{f}_\theta^{-1}(\boldsymbol{x})$$
Mapping	There is no one-to-one mapping between $\boldsymbol{x}$ and $\boldsymbol{z}$.	There is a one-to-one mapping between $\boldsymbol{x}$ and $\boldsymbol{z}$.
Type of $\boldsymbol{x}$	Can be continuous or discrete (binary, categorical)	Must be continuous
Type of $\boldsymbol{z}$	Usually continuous	Must be continuous
Latent Dimension	Since the decoder is not invertible, the latent dimension can differ from the input dimension, allowing a lower-dimensional representation.	Invertibility mathematically requires equal dimensionality. The latent dimension must equal the input dimension, limiting the possibility of a lower-dimensional representation.
Likelihood Computation	The lack of invertibility prevents direct computation of the likelihood $p_\theta(\boldsymbol{x})$. Instead, a variational lower bound (e.g., ELBO, IWAE bound, etc) is optimized.	Invertibility allows direct computation of the likelihood $p_\theta(\boldsymbol{x})$, using the change-of-variables formula.
Sampling (Post-training)	1. Sample from the prior $\boldsymbol{z} \sim p_\theta(\boldsymbol{z})$. 2. Compute the parameters (mean and variance) of the decoder $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$, using $\boldsymbol{z}$. 3. Sample from the decoder. i.e., $\boldsymbol{x} \sim p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$.	1. Sample from the prior $\boldsymbol{z} \sim p_\theta(\boldsymbol{z})$. 2. Compute $\boldsymbol{x} = \boldsymbol{f}_\theta(\boldsymbol{z})$.
Workflow	$\boldsymbol{x} \xrightarrow[\text{NN:}\mu_\phi(x),\sigma_\phi(x)]{\text{Encoder}} q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \xrightarrow[\boldsymbol{z}\sim q_\phi(\boldsymbol{z}\mid\boldsymbol{x})]{\text{Sampling}} \boldsymbol{z} \xrightarrow[\text{NN}f_\theta(z)]{\text{Decoder}}$ $p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) \xrightarrow{\text{Compute}}$ Lower bound of $p_\theta(x)$	$p_\theta(\boldsymbol{z}) \xrightarrow[\boldsymbol{z}\sim p_\theta(\boldsymbol{z})]{\text{Sampling}} \boldsymbol{z} \xrightarrow[\boldsymbol{x}=\boldsymbol{f}_\theta(\boldsymbol{z})]{\text{Decoder}} \boldsymbol{x} \xrightarrow[\text{Change-of-variable}]{\text{Compute}}$ $p_\theta(x)$

5.3 Normalizing Flows

Normalizing flow models

- Consider a directed, latent-variable model over observed variables X and latent variables Z
- In a **normalizing flow model**, the mapping between Z and X, given by $\mathbf{f}_\theta : \mathbb{R}^n \mapsto \mathbb{R}^n$, is deterministic and invertible such that $X = \mathbf{f}_\theta(Z)$ and $Z = \mathbf{f}_\theta^{-1}(X)$



- Using change of variables, the marginal likelihood $p(\mathbf{x})$ is given by

$$p_X(\mathbf{x};\theta) = p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right)\left|\det\left(\frac{\partial \mathbf{f}_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)\right|$$

- Note: $\mathbf{x}, \mathbf{z}$ need to be continuous and have the same dimension.

A Flow of Transformations

Normalizing: Change of variables gives a normalized density after applying an invertible transformation
Flow: Invertible transformations can be composed with each other

$$\mathbf{z}_m = \mathbf{f}_\theta^m \circ \cdots \circ \mathbf{f}_\theta^1(\mathbf{z}_0) = \mathbf{f}_\theta^m(\mathbf{f}_\theta^{m-1}(\cdots(\mathbf{f}_\theta^1(\mathbf{z}_0)))) \triangleq \mathbf{f}_\theta(\mathbf{z}_0)$$

- Start with a simple distribution for $\mathbf{z}_0$ (e.g., Gaussian)
- Apply a sequence of M invertible transformations to finally obtain $\mathbf{x} = \mathbf{z}_M$
- By change of variables

$$p_X(\mathbf{x};\theta) = p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right)\prod_{m=1}^{M}\left|\det\left(\frac{\partial (\mathbf{f}_\theta^m)^{-1}(\mathbf{z}_m)}{\partial \mathbf{z}_m}\right)\right|$$

(Note: determininant of product equals product of determinants)

Learning and Inference

- Learning via **maximum likelihood** over the dataset $\mathcal{D}$

$$\max_\theta \log p_X(\mathcal{D};\theta) = \sum_{\mathbf{x}\in\mathcal{D}} \log p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right) + \log\left|\det\left(\frac{\partial \mathbf{f}_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)\right|$$

- **Exact likelihood evaluation** via inverse tranformation $\mathbf{x} \mapsto \mathbf{z}$ and change of variables formula
- **Sampling** via forward transformation $\mathbf{z} \mapsto \mathbf{x}$

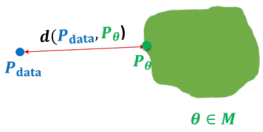
$$\mathbf{z} \sim p_Z(\mathbf{z}) \quad \mathbf{x} = \mathbf{f}_\theta(\mathbf{z})$$

- **Latent representations** inferred via inverse transformation (no inference network required!)

$$\mathbf{z} = \mathbf{f}_\theta^{-1}(\mathbf{x})$$

Recap

$\mathbf{x}_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$



Model family

- Model families
  - Autoregressive Models: $p_\theta(\mathbf{x}) = \prod_{i=1}^{n} p_\theta(x_i|\mathbf{x}_{<i})$
  - Variational Autoencoders: $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z}$
  - Normalizing Flow Models: $p_X(\mathbf{x}; \theta) = p_Z\left(\mathbf{f}_\theta^{-1}(\mathbf{x})\right) \left|\det\left(\frac{\partial \mathbf{f}_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)\right|$
- All the above families are trained by minimizing KL divergence $d_{KL}(p_{\text{data}} \| p_\theta)$, or equivalently maximizing likelihoods (or approximations)
- Today: alternative choices for $d(p_{\text{data}} \| p_\theta)$

6 Generative Adversarial Networks (GANs)

6.1 Motivation of Likelihood-Free Learning

Types of learning

1. Density estimation (maximize likelihood)
   - Classical generative models (e.g., AR, VAE, NF) are typically trained by maximizing the log-likelihood $\max_{p_\theta} \sum_{i=1}^{n} \log P_\theta(\boldsymbol{x}^{(i)})$, which is equivalent to minimizing the KL divergence between the data and model distributions $\min_{p_\theta} D_{\mathrm{KL}}(P_{\mathrm{data}} \,\|\, P_\theta)$.
2. Likelihood-free learning (train a binary classifier, which is called discriminator in GAN)
   - However, a high log-likelihood (or small KL divergence) does not necessarily imply good sample quality, and vice versa. Therefore, it is important to disentangle likelihood from sample quality, motivating the development of likelihood-free learning approaches.

6.2 Notation for GAN

Notation	Explanation
$D_\phi : \mathbb{R}^D \to [0,1]$	Discriminator: • It tries to tell apart real samples $\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})$ from fake samples $\boldsymbol{x} = G_\theta(\boldsymbol{z})$ with $\boldsymbol{z} \sim p(\boldsymbol{z})$. • Input: data sample $\boldsymbol{x}$ (real or fake) • Output: probability being real • Typically implemented as a neural network with parameters ϕ, e.g., $D_\phi(\boldsymbol{x}) = \sigma(h_\phi(\boldsymbol{x}))$, – Logits $h_\phi(\boldsymbol{x})$: $\boldsymbol{x} \to$ Flatten $\to$ Linear $\to$ LeakyReLU $\to$ Linear – Sigmoid activation $\sigma(\cdot)$: convert logits $h_\phi(\boldsymbol{x})$ to probability $D_\phi(\boldsymbol{x})$
$G_\theta(\boldsymbol{z}) : \mathbb{R}^L \to \mathbb{R}^D$	Generator : • It tries to fool the discriminator by generating fake samples that appear real, i.e., $D_\phi(G_\theta(\boldsymbol{z})) \to 1$. • Input: latent vector $\boldsymbol{z} \sim p(\boldsymbol{z})$ • Output: fake sample $\boldsymbol{x}$ • Typically implemented as a neural network with parameters θ.

6.3 GAN

- A Generative Adversarial Network (GAN) consists of two players:
  1. Discriminator $D_\phi(\boldsymbol{x})$
  2. Generator $G_\theta(\boldsymbol{z})$
- Each player minimizes its own loss while maximizing against the other.

6.4 Discriminator Loss

- Intuitively, $D_\phi(\boldsymbol{x})$ should assign
  1. high probability to real samples, $D_\phi(\boldsymbol{x}) \to 1$
  2. low probability to fake samples, $D_\phi(G_\theta(\boldsymbol{z})) \to 0$
- The loss function is defined as,

$$\begin{aligned}\mathcal{L}_D(\phi;\theta) &= -\mathbb{E}_{\boldsymbol{x}\sim p_{\textbf{data}}(\boldsymbol{x})}[\log D_\phi(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[\log(1 - D_\phi(\boldsymbol{x}))] \\ &= -\mathbb{E}_{\boldsymbol{x}\sim p_{\textbf{data}}(\boldsymbol{x})}[\log D_\phi(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D_\phi(G_\theta(\boldsymbol{z})))]\end{aligned}$$

  Here, $p_\theta(\boldsymbol{x})$ denotes the distribution of fake samples generated by $G_\theta(\boldsymbol{z})$ with $\boldsymbol{z} \sim p(\boldsymbol{z})$ and $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z} \mid 0, I)$.
- The goal of the discriminator is to minimize the loss, which is equivalent to maximizing the negative loss,

$$\max_\phi -\mathcal{L}_D(\phi;\theta) = \max_\phi \left(\mathbb{E}_{\boldsymbol{x}\sim p_{\textbf{data}}(\boldsymbol{x})}[\log D_\phi(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D_\phi(G_\theta(\boldsymbol{z})))]\right)$$

- $\mathcal{L}_D(\phi;\theta)$ is minimized when $D_\phi(\boldsymbol{x}) = D^*_\phi(\boldsymbol{x})$, with

$$D^*_\phi(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}$$

  – According to definition of $\mathcal{L}_D(\phi;\theta)$,

$$\begin{aligned}\mathcal{L}_D(\phi;\theta) &= -\mathbb{E}_{\boldsymbol{x}\sim p_{\textbf{data}}(\boldsymbol{x})}[\log D_\phi(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[\log(1 - D_\phi(\boldsymbol{x}))] \\ &= -\int p_{\text{data}}(\boldsymbol{x}) \log D_\phi(\boldsymbol{x})\, d\boldsymbol{x} - \int p_\theta(\boldsymbol{x}) \log(1 - D_\phi(\boldsymbol{x}))\, d\boldsymbol{x} \\ &= \int \left(-p_{\text{data}}(\boldsymbol{x}) \log D_\phi(\boldsymbol{x}) - p_\theta(\boldsymbol{x}) \log(1 - D_\phi(\boldsymbol{x}))\right)\, d\boldsymbol{x}\end{aligned}$$

  Thus, by calculation,

$$\begin{aligned}\mathcal{L}'_D(\phi;\theta) &= \frac{d}{dD_\phi(\boldsymbol{x})} \int \left(-p_{\text{data}}(\boldsymbol{x}) \log D_\phi(\boldsymbol{x}) - p_\theta(\boldsymbol{x}) \log(1 - D_\phi(\boldsymbol{x}))\right)\, d\boldsymbol{x} \\ &= \int \frac{d}{dD_\phi(\boldsymbol{x})} \left(-p_{\text{data}}(\boldsymbol{x}) \log D_\phi(\boldsymbol{x}) - p_\theta(\boldsymbol{x}) \log(1 - D_\phi(\boldsymbol{x}))\right)\, d\boldsymbol{x} \\ &= \int \frac{d}{dt} \left(-p_{\text{data}}(\boldsymbol{x}) \log t - p_\theta(\boldsymbol{x}) \log(1 - t)\right)\Big|_{t=D_\phi(\boldsymbol{x})}\, d\boldsymbol{x} \\ &= \int \left(\frac{p_\theta(\boldsymbol{x})}{1 - t} - \frac{p_{\text{data}}(\boldsymbol{x})}{t}\right)\Big|_{t=D_\phi(\boldsymbol{x})}\, d\boldsymbol{x} \\ &= \int \left(\frac{p_\theta(\boldsymbol{x})}{1 - D_\phi(\boldsymbol{x})} - \frac{p_{\text{data}}(\boldsymbol{x})}{D_\phi(\boldsymbol{x})}\right)\, d\boldsymbol{x}\end{aligned}$$

and

$$\begin{aligned}
\mathcal{L}_D''(\phi;\theta) &= \frac{d}{dD_\phi(\boldsymbol{x})} \int \left(\frac{p_\theta(\boldsymbol{x})}{1-D_\phi(\boldsymbol{x})} - \frac{p_{\text{data}}(\boldsymbol{x})}{D_\phi(\boldsymbol{x})} \right) d\boldsymbol{x} \\
&= \int \frac{d}{dD_\phi(\boldsymbol{x})} \left(\frac{p_\theta(\boldsymbol{x})}{1-D_\phi(\boldsymbol{x})} - \frac{p_{\text{data}}(\boldsymbol{x})}{D_\phi(\boldsymbol{x})} \right) d\boldsymbol{x} \\
&= \int \frac{d}{dt} \left(\frac{p_\theta(\boldsymbol{x})}{1-t} - \frac{p_{\text{data}}(\boldsymbol{x})}{t} \right) \Bigg|_{t=D_\phi(\boldsymbol{x})} d\boldsymbol{x} \\
&= \int \left(\frac{p_\theta(\boldsymbol{x})}{(1-t)^2} + \frac{p_{\text{data}}(\boldsymbol{x})}{t^2} \right) \Bigg|_{t=D_\phi(\boldsymbol{x})} d\boldsymbol{x} \\
&= \int \left(\frac{p_\theta(\boldsymbol{x})}{(1-D_\phi(\boldsymbol{x}))^2} + \frac{p_{\text{data}}(\boldsymbol{x})}{D_\phi^2(\boldsymbol{x})} \right) d\boldsymbol{x} \\
&\geq 0
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathcal{L}_D'(\phi;\theta) = 0 &\Rightarrow \int \left(\frac{p_\theta(\boldsymbol{x})}{1-D_\phi(\boldsymbol{x})} - \frac{p_{\text{data}}(\boldsymbol{x})}{D_\phi(\boldsymbol{x})} \right) d\boldsymbol{x} = 0 \\
&\Rightarrow \frac{p_\theta(\boldsymbol{x})}{1-D_\phi(\boldsymbol{x})} - \frac{p_{\text{data}}(\boldsymbol{x})}{D_\phi(\boldsymbol{x})} = 0 \\
&\Rightarrow D_\phi(\boldsymbol{x}) \cdot p_\theta(\boldsymbol{x}) - (1-D_\phi(\boldsymbol{x})) \cdot p_{\text{data}}(\boldsymbol{x}) = 0 \\
&\Rightarrow D_\phi(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}
\end{aligned}$$

- If $D_\phi(\boldsymbol{x}) = D_\phi^*(\boldsymbol{x})$, then

$$h_\phi(\boldsymbol{x}) = \log \left(\frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x})} \right)$$

  - Since

$$\begin{aligned}
D_\phi^*(\boldsymbol{x}) &= \frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})} \\
D_\phi(\boldsymbol{x}) &= \sigma(h_\phi(\boldsymbol{x})) = \frac{1}{1+e^{-h_\phi(\boldsymbol{x})}}
\end{aligned}$$

    Thus, when $D_\phi(\boldsymbol{x}) = D_\phi^*(\boldsymbol{x})$,

$$\begin{aligned}
\frac{1}{1+e^{-h_\phi(\boldsymbol{x})}} = \frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})} &\Rightarrow 1 + e^{-h_\phi(\boldsymbol{x})} = \frac{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})} \\
&\Rightarrow e^{-h_\phi(\boldsymbol{x})} = \frac{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})} - 1 \\
&\Rightarrow e^{-h_\phi(\boldsymbol{x})} = \frac{p_\theta(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})} \\
&\Rightarrow -h_\phi(\boldsymbol{x}) = \log \left(\frac{p_\theta(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})} \right) \\
&\Rightarrow h_\phi(\boldsymbol{x}) = \log \left(\frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x})} \right)
\end{aligned}$$

6.5 Generator Minimax Loss

- The generator aims to fool the discriminator, i.e., minimizes the same value function that the discriminator tries to maximize,

$$\begin{aligned}
&\arg\max_{\theta} \left(-\mathbb{E}_{\boldsymbol{x}\sim p_{\textbf{data}}(\boldsymbol{x})}[\log D_{\phi}(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D_{\phi}(G_{\theta}(\boldsymbol{z})))]\right) \\
&= \arg\min_{\theta} \left(\mathbb{E}_{\boldsymbol{x}\sim p_{\textbf{data}}(\boldsymbol{x})}[\log D_{\phi}(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D_{\phi}(G_{\theta}(\boldsymbol{z})))]\right) \\
&= \arg\min_{\theta} \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D_{\phi}(G_{\theta}(\boldsymbol{z})))]
\end{aligned}$$

- The generator minimax loss is defined as,

$$\begin{aligned}
\mathcal{L}_G^{\text{minimax}}(\theta;\phi) &= \mathbb{E}_{\boldsymbol{x}\sim p_{\theta}(\boldsymbol{x})}[\log(1 - D_{\phi}(\boldsymbol{x}))] \\
&= \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D_{\phi}(G_{\theta}(\boldsymbol{z})))]
\end{aligned}$$

- The goal of the generator is to minimize this loss,

$$\min_{\theta} \mathcal{L}_G^{\text{minimax}}(\theta;\phi) = \min_{\theta} \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D_{\phi}(G_{\theta}(\boldsymbol{z})))]$$

6.6 Generator Non-Saturating Loss

- The generator non-saturating loss is defined,

$$\begin{aligned}\mathcal{L}_G^{\text{non-sat}}(\theta;\phi) &= -\mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[\log D_\phi(\boldsymbol{x})] \\ &= -\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log D_\phi(G_\theta(\boldsymbol{z}))]\end{aligned}$$

- The goal of the generator is to minimize this loss, which is equivalent to maximize its negative,

$$\max_\theta -\mathcal{L}_G^{\text{non-sat}}(\theta;\phi) = \max_\theta \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log D_\phi(G_\theta(\boldsymbol{z}))]$$

6.7 Generator KL-based Loss

- The generator KL-based loss is defined as the sum of the minimax loss and the non-saturating loss,

$$\begin{aligned}\mathcal{L}_G(\theta;\phi) &= \mathcal{L}_G^{\text{minimax}}(\theta;\phi) + \mathcal{L}_G^{\text{non-sat}}(\theta;\phi) \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[\log(1 - D_\phi(\boldsymbol{x}))] - \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[\log D_\phi(\boldsymbol{x})] \\ &= \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D_\phi(G_\theta(\boldsymbol{z})))] - \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log D_\phi(G_\theta(\boldsymbol{z}))]\end{aligned}$$

- If $D_\phi(\boldsymbol{x}) = D_\phi^*(\boldsymbol{x})$, then

$$\mathcal{L}_G(\theta;\phi) = D_{\text{KL}}(p_\theta(\boldsymbol{x}) \,\|\, p_{\text{data}}(\boldsymbol{x}))$$

  - Since

$$D_\phi^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}$$

    Thus, by calculation,

$$\begin{aligned}\mathcal{L}_G(\theta;\phi) &= \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[\log(1 - D_\phi(\boldsymbol{x}))] - \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[\log D_\phi(\boldsymbol{x})] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}\left[\log\left(\frac{1 - D_\phi(\boldsymbol{x})}{D_\phi(\boldsymbol{x})}\right)\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}\left[\log\left(\frac{1 - D_\phi^*(\boldsymbol{x})}{D_\phi^*(\boldsymbol{x})}\right)\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}\left[\log\left(\frac{1 - \frac{p_{\textbf{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x}) + p_{\textbf{data}}(\boldsymbol{x})}}{\frac{p_{\textbf{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x}) + p_{\textbf{data}}(\boldsymbol{x})}}\right)\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}\left[\log\left(\frac{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x}) - p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})}\right)\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}\left[\log\left(\frac{p_\theta(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})}\right)\right] \\ &= D_{\text{KL}}(p_\theta(\boldsymbol{x}) \,\|\, p_{\text{data}}(\boldsymbol{x}))\end{aligned}$$

6.8 WGAN Critic Loss

- The WGAN critic loss is defined as,

$$\begin{aligned}\mathcal{L}_D(\phi;\theta) &= -\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[D_\phi(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[D_\phi(\boldsymbol{x})] \\ &= -\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[D_\phi(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[D_\phi(G_\theta(\boldsymbol{z}))]\end{aligned}$$

6.9 WGAN-GP Critic Loss

- The WGAN-GP critic loss is defined as,

$$\begin{aligned}\mathcal{L}_D(\phi;\theta) &= -\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[D_\phi(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[D_\phi(\boldsymbol{x})] + \lambda\mathbb{E}_{\boldsymbol{x}\sim r_\theta(\boldsymbol{x})}\left[(\|\nabla D_\phi(\boldsymbol{x})\|_2 - 1)^2\right] \\ &= -\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[D_\phi(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[D_\phi(G_\theta(\boldsymbol{z}))] + \lambda\mathbb{E}_{\boldsymbol{x}\sim r_\theta(\boldsymbol{x})}\left[(\|\nabla D_\phi(\boldsymbol{x})\|_2 - 1)^2\right]\end{aligned}$$

6.10 WGAN Generator Loss

- The WGAN generator loss is defined as,

$$\begin{aligned}\mathcal{L}_G(\theta;\phi) &= -\mathbb{E}_{\boldsymbol{z}\sim p_\theta(\boldsymbol{x})}[D_\phi(\boldsymbol{x})] \\ &= -\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[D_\phi(G_\theta(\boldsymbol{z}))]\end{aligned}$$

6.11 Standard GAN

- Definition:
  1. Discriminator loss
     - Optimal discriminator,

$$D_\phi^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}$$

  2. Generator minimax loss
- Note: Discriminator loss and generator minimax loss can be combined as,

$$\begin{aligned} V(G, D) &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D_\phi(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[\log(1 - D_\phi(\boldsymbol{x}))] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D_\phi(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D_\phi(G_\theta(\boldsymbol{z})))] \end{aligned}$$

  with the training objective

$$\min_G \max_D V(G, D) = \min_G V(G, D^*)$$

  By calculation,

$$\begin{aligned} V(G, D^*) &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D_\phi^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[\log(1 - D_\phi^*(\boldsymbol{x}))] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\log\left(\frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}\right)\right] + \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}\left[\log\left(1 - \frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}\right)\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\log\left(\frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}\right)\right] + \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}\left[\log\left(\frac{p_\theta(\boldsymbol{x})}{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}\right)\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\log\left(\frac{p_{\text{data}}(\boldsymbol{x})}{\frac{p_\theta(\boldsymbol{x})+p_{\text{data}}(\boldsymbol{x})}{2}} \cdot \frac{1}{2}\right)\right] + \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}\left[\log\left(\frac{p_\theta(\boldsymbol{x})}{\frac{p_\theta(\boldsymbol{x})+p_{\text{data}}(\boldsymbol{x})}{2}} \cdot \frac{1}{2}\right)\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\log\left(\frac{p_{\text{data}}(\boldsymbol{x})}{\frac{p_\theta(\boldsymbol{x})+p_{\text{data}}(\boldsymbol{x})}{2}}\right) + \log\left(\frac{1}{2}\right)\right] + \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}\left[\log\left(\frac{p_\theta(\boldsymbol{x})}{\frac{p_\theta(\boldsymbol{x})+p_{\text{data}}(\boldsymbol{x})}{2}}\right) + \log\left(\frac{1}{2}\right)\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\log\left(\frac{p_{\text{data}}(\boldsymbol{x})}{\frac{p_\theta(\boldsymbol{x})+p_{\text{data}}(\boldsymbol{x})}{2}}\right)\right] + \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}\left[\log\left(\frac{p_\theta(\boldsymbol{x})}{\frac{p_\theta(\boldsymbol{x})+p_{\text{data}}(\boldsymbol{x})}{2}}\right)\right] + 2\log\left(\frac{1}{2}\right) \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\log\left(\frac{p_{\text{data}}(\boldsymbol{x})}{\frac{p_\theta(\boldsymbol{x})+p_{\text{data}}(\boldsymbol{x})}{2}}\right)\right] + \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}\left[\log\left(\frac{p_\theta(\boldsymbol{x})}{\frac{p_\theta(\boldsymbol{x})+p_{\text{data}}(\boldsymbol{x})}{2}}\right)\right] - \log 4 \\ &= D_{\text{KL}}\left(p_{\text{data}}(\boldsymbol{x}) \middle\| \frac{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}{2}\right) + D_{\text{KL}}\left(p_\theta(\boldsymbol{x}) \middle\| \frac{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}{2}\right) - \log 4 \\ &= 2 \cdot \underbrace{\frac{D_{\text{KL}}\left(p_{\text{data}}(\boldsymbol{x}) \middle\| \frac{p_\theta(\boldsymbol{x})+p_{\text{data}}(\boldsymbol{x})}{2}\right) + D_{\text{KL}}\left(p_\theta(\boldsymbol{x}) \middle\| \frac{p_\theta(\boldsymbol{x})+p_{\text{data}}(\boldsymbol{x})}{2}\right)}{2}}_{\text{Jensen-Shannon divergence (JSD)}} - \log 4 \\ &= 2D_{\text{JSD}}\left(p_{\text{data}}(\boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{x})\right) - \log 4 \end{aligned}$$

  Thus, the training objective becomes

$$\begin{aligned} \min_G \max_D V(G, D) &= \min_G V(G, D^*) \\ &= \min_G 2D_{\text{JSD}}\left(p_{\text{data}}(\boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{x})\right) - \log 4 \end{aligned}$$

- Issue: It often suffers from vanishing gradients when the discriminator becomes too strong, making it hard for the generator to learn.

– According to the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$,

$$\frac{\sigma'(x)}{1-\sigma(x)} = \frac{\frac{e^{-x}}{(1+e^{-x})^{-2}}}{1-\frac{1}{1+e^{-x}}} = \frac{\frac{e^{-x}}{(1+e^{-x})^{-1}}}{1+e^{-x}-1} = \frac{\frac{e^{-x}}{(1+e^{-x})^{-1}}}{e^{-x}} = \frac{1}{(1+e^{-x})} = \sigma(x)$$

Therefore, by calculation,

$$\begin{aligned}
\frac{\partial \mathcal{L}_G^{\text{minimax}}}{\partial \theta} &= \frac{\partial}{\partial \theta}\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1-\sigma(h_\phi(G_\theta(\boldsymbol{z}))))] \\
&= \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\frac{\partial}{\partial \theta}\log(1-\sigma(h_\phi(G_\theta(\boldsymbol{z}))))\right] \\
&= \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[-\frac{\sigma'(h_\phi(G_\theta(\boldsymbol{z})))}{1-\sigma(h_\phi(G_\theta(\boldsymbol{z})))}\cdot\frac{\partial}{\partial \theta}h_\phi(G_\theta(\boldsymbol{z}))\right] \\
&= -\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\frac{\sigma'(h_\phi(G_\theta(\boldsymbol{z})))}{1-\sigma(h_\phi(G_\theta(\boldsymbol{z})))}\cdot\frac{\partial}{\partial \theta}h_\phi(G_\theta(\boldsymbol{z}))\right] \\
&= -\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\sigma(h_\phi(G_\theta(\boldsymbol{z})))\cdot\frac{\partial}{\partial \theta}h_\phi(G_\theta(\boldsymbol{z}))\right] \\
&= -\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[D_\phi(G_\theta(\boldsymbol{z}))\cdot\frac{\partial}{\partial \theta}h_\phi(G_\theta(\boldsymbol{z}))\right]
\end{aligned}$$

When the discriminator is strong, it confidently predicts fake samples as fake, i.e., $D_\phi(G_\theta(\boldsymbol{z})) \approx 0$. Then the gradient of the generator's minimax loss is

$$\begin{aligned}
\frac{\partial \mathcal{L}_G^{\text{minimax}}}{\partial \theta} &= -\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[D_\phi(G_\theta(\boldsymbol{z}))\cdot\frac{\partial}{\partial \theta}h_\phi(G_\theta(\boldsymbol{z}))\right] \\
&\approx -\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[0\cdot\frac{\partial}{\partial \theta}h_\phi(G_\theta(\boldsymbol{z}))\right] \\
&= 0
\end{aligned}$$

Thus, the gradient vanishes, causing the generator to fail to improve and the training to stall, while the discriminator continues to classify generated samples as fake with high confidence.

6.12 Non-saturating GAN

- Definition:
  - Same as standard GAN
  - Generator non-saturating loss
- Note: To address the issue in the standard GAN, the non-saturating GAN uses an alternative generator loss, which encourages the generator to maximize the probability that the discriminator classifies its samples as real. Even when $D_\phi(G_\theta(\boldsymbol{z})) \approx 0$, the gradient remains large, providing strong learning signals and avoiding vanishing gradients. This results in stronger gradients for the generator and generally leads to more stable and faster training.

6.13 KL-based GAN

- Definition:
  - Same as standard GAN
  - Generator KL-based loss
    - When the discriminator is optimal, i.e., $D_{\phi}(\boldsymbol{x}) = D^{*}_{\phi}(\boldsymbol{x})$,

      $$\mathcal{L}_G(\theta; \phi) = D_{\mathrm{KL}}(p_{\theta}(\boldsymbol{x}) \,\|\, p_{\mathrm{data}}(\boldsymbol{x}))$$

      meaning the generator minimizes the KL divergence between its distribution and the data distribution.
- Issue: See case 1 in Type Issues

6.14 Conditional GAN with Projection Discriminator

- Definition:
  –
  –

6.15 Wasserstein GAN (WGAN)

- Definition:
  - WGAN critic loss
  - WGAN generator loss
- Issue: See case 2 in Type Issues

6.16 Wasserstein GAN with Gradient Penalty (WGAN-GP)

- Definition:
  - WGAN-GP critic loss
  - Same as WGAN
- Note: WGAN uses the Wasserstein distance instead of JS or KL divergence. This gives smoother gradients, reduces instability, and allows more reliable convergence.

6.17 Issue Analysis

Assume that $\epsilon \to 0$ and

$$p_\theta(x) = \mathcal{N}(x \mid \theta, \epsilon^2)$$
$$p_{\text{data}}(x) = \mathcal{N}(x \mid \theta_0, \epsilon^2)$$

- Case 1:

  $$D_\phi : \mathbb{R} \to [0, 1]$$
  $$\mathcal{L}_D(\phi; \theta) = -\mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D_\phi(x)] - \mathbb{E}_{x \sim p_\theta(x)}[\log(1 - D_\phi(x))]$$
  $$\mathcal{L}_G(\theta; \phi) = \mathbb{E}_{x \sim p_\theta(x)}[\log(1 - D_\phi(x))] - \mathbb{E}_{x \sim p_\theta(x)}[\log(1 - D_\phi(x))]$$

  By calculation,

  $$\mathcal{L}_G(\theta; \phi) = D_{\text{KL}}(p_\theta(x) \,\|\, p_{\text{data}}(x))$$
  $$D_{\text{KL}}(p_\theta(x) \,\|\, p_{\text{data}}(x)) = \frac{(\theta - \theta_0)^2}{2\epsilon^2}$$
  $$\frac{\partial}{\partial \theta} D_{\text{KL}}(p_\theta(x) \,\|\, p_{\text{data}}(x)) = \frac{\theta - \theta_0}{\epsilon^2}$$

  Thus, when $\epsilon \to 0$,

  $$\mathcal{L}_G(\theta; \phi) \to \infty, \quad \frac{\partial}{\partial \theta} \mathcal{L}_G(\theta; \phi) \to \infty$$

  This means even a tiny difference between θ and θ_0 leads to a huge generator loss and huge gradient.

  When $\epsilon \to 0$, the distributions

  $$p_\theta(x) = \mathcal{N}(x \mid \theta, \epsilon^2)$$
  $$p_{\text{data}}(x) = \mathcal{N}(x \mid \theta_0, \epsilon^2)$$

  become very narrow Gaussians, each concentrated around its mean. If $\epsilon \neq \epsilon_0$, the distributions have almost no overlap.

  This is problematic:

  – During GAN training, the generator overreacts to small errors:
    1. The loss becomes extreme large.
    2. Even tiny change in model parameters cause big unstable updates.
  – As a result, training becomes unstable and may fail to converge.

- Case 2:

  $$D_\phi : \mathbb{R} \to \mathbb{R}$$
  $$\mathcal{L}_D(\phi; \theta) = -\mathbb{E}_{x \sim p_{\text{data}}(x)}[D_\phi(x)] + \mathbb{E}_{x \sim p_\theta(x)}[D_\phi(x)]$$
  $$\mathcal{L}_G(\theta; \phi) = -\mathbb{E}_{x \sim p_\theta(x)}[D_\phi(x)]$$

  When $\epsilon \to 0$, both distributions collapse to single points,

  $$p_\theta(x) = \theta, \quad p_{\text{data}}(x) = \theta_0$$

Then the discriminator loss becomes,

$$\mathcal{L}_D(\phi;\theta) = D_\phi(\theta) - D_\phi(\theta_0)$$

Since $D_\phi \in (-\infty, \infty)$, let

$$D_\phi(\theta) \to -\infty, \quad D_\phi(\theta_0) \to \infty$$

so that

$$\mathcal{L}_D(\phi;\theta) \to -\infty$$

This means the discriminator loss can be made arbitrarily small, i.e., there is no finite minimum and no optimal discriminator exists.

- Case 3:

$$D_\phi : \mathbb{R} \to \mathbb{R}, \quad \frac{dD_\phi}{dx} : \mathbb{R} \to [-1, 1]$$

$$\mathcal{L}_D(\phi;\theta) = -\mathbb{E}_{x\sim p_{\mathbf{data}}(x)}[D_\phi(x)] + \mathbb{E}_{x\sim p_\theta(\boldsymbol{x})}[D_\phi(x)]$$

$$\mathcal{L}_G(\theta;\phi) = -\mathbb{E}_{x\sim p_\theta(x)}[D_\phi(x)]$$

6.18 f-divergence

f divergences

Many more f-divergences!

Name	$D_f(P\|Q)$	Generator $f(u)$
Total variation	$\frac{1}{2}\int \lvert p(x) - q(x)\rvert \, \mathrm{d}x$	$\frac{1}{2}\lvert u-1\rvert$
Kullback-Leibler	$\int p(x) \log \frac{p(x)}{q(x)} \, \mathrm{d}x$	$u \log u$
Reverse Kullback-Leibler	$\int q(x) \log \frac{q(x)}{p(x)} \, \mathrm{d}x$	$-\log u$
Pearson χ^2	$\int \frac{(q(x)-p(x))^2}{p(x)} \, \mathrm{d}x$	$(u-1)^2$
Neyman χ^2	$\int \frac{(p(x)-q(x))^2}{q(x)} \, \mathrm{d}x$	$\frac{(1-u)^2}{u}$
Squared Hellinger	$\int \left(\sqrt{p(x)} - \sqrt{q(x)}\right)^2 \, \mathrm{d}x$	$(\sqrt{u}-1)^2$
Jeffrey	$\int (p(x) - q(x)) \log\left(\frac{p(x)}{q(x)}\right) \, \mathrm{d}x$	$(u-1)\log u$
Jensen-Shannon	$\frac{1}{2}\int p(x) \log \frac{2p(x)}{p(x)+q(x)} + q(x) \log \frac{2q(x)}{p(x)+q(x)} \, \mathrm{d}x$	$-(u+1)\log\frac{1+u}{2} + u \log u$
Jensen-Shannon-weighted	$\int p(x)\pi \log \frac{p(x)}{\pi p(x)+(1-\pi)q(x)} + (1-\pi)q(x) \log \frac{q(x)}{\pi p(x)+(1-\pi)q(x)} \, \mathrm{d}x$	$\pi u \log u - (1-\pi+\pi u)\log(1-\pi+\pi u)$
GAN	$\int p(x) \log \frac{2p(x)}{p(x)+q(x)} + q(x) \log \frac{2q(x)}{p(x)+q(x)} \, \mathrm{d}x - \log(4)$	$u \log u - (u+1)\log(u+1)$
α-divergence ($\alpha \notin \{0,1\}$)	$\frac{1}{\alpha(\alpha-1)} \int \left(p(x)\left[\left(\frac{q(x)}{p(x)}\right)^\alpha - 1\right] - \alpha(q(x)-p(x))\right) \mathrm{d}x$	$\frac{1}{\alpha(\alpha-1)}\left(u^\alpha - 1 - \alpha(u-1)\right)$

Source: Nowozin et al., 2016