Course Materials for GEN-AI

Northeastern University

These materials have been prepared and sourced for the course **GEN-AI** at Northeastern University. Every effort has been made to provide proper citations and credit for all referenced works.

If you believe any material has been inadequately cited or requires correction, please contact me at:

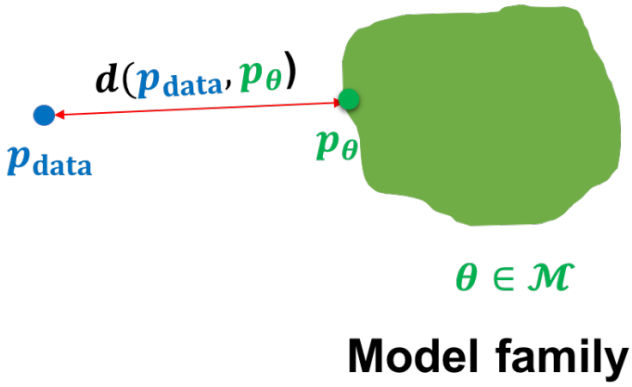
Instructor: Ramin Mohammadi
r.mohammadi@northeastern.edu

Thank you for your understanding and collaboration.

Content sourced from multiple references - see Reference section for details

Evaluation Of Generative AI Models

1 Model Family



Generative models can be broadly classified based on how they model probability densities or mass functions. Here are the main types:

1.1 Probability Density and Mass Functions

- **Autoregressive Models:** Autoregressive models decompose the joint probability of data $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ as a product of conditional probabilities:

$$p_\phi(\mathbf{x}) = \prod_{i=1}^{d} p_\phi(x_i \mid x_{<i}),$$

  where $x_{<i} = (x_1, x_2, \ldots, x_{i-1})$. Examples include PixelCNN and WaveNet.

- **Normalizing Flow Models:** Normalizing flows transform a simple base distribution $p_Z(\mathbf{z})$ (e.g., Gaussian) into a complex data distribution $p_X(\mathbf{x})$ through a sequence of invertible transformations:

$$p_\phi(\mathbf{x}) = p_Z\left(f_\phi^{-1}(\mathbf{x})\right) \left| \det \left(\frac{\partial f_\phi^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right|,$$

  where f_ϕ is an invertible mapping parameterized by ϕ.

- **Latent Variable Models (e.g., Variational Autoencoders):** Latent variable models introduce a set of latent variables $\mathbf{z}$ to model complex data distributions:

$$p_\phi(\mathbf{x}) = \int p_\phi(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) \, d\mathbf{z}.$$

Variational Autoencoders (VAEs) optimize a variational bound on the log-likelihood.

- **Energy-Based Models (EBMs):** EBMs define the data distribution in terms of an energy function $E_\phi(\mathbf{x})$:

$$p_\phi(\mathbf{x}) = \frac{\exp(-E_\phi(\mathbf{x}))}{Z(\phi)},$$

where $Z(\phi) = \int \exp(-E_\phi(\mathbf{x}))d\mathbf{x}$ is the partition function that normalizes the distribution.

All the above families are typically trained by minimizing the KL divergence between the data distribution $p_{\text{data}}(\mathbf{x})$ and the model distribution $p_\phi(\mathbf{x})$:

$$D_{\text{KL}}(p_{\text{data}} \| p_\phi) = \int p_{\text{data}}(\mathbf{x}) \log \frac{p_{\text{data}}(\mathbf{x})}{p_\phi(\mathbf{x})} d\mathbf{x}.$$

Minimizing D_{KL} is equivalent to maximizing the likelihood of the data under the model.

1.2 Sample Generation Processes

In practice, samples from generative models can be obtained through processes such as:

- **Generative Adversarial Networks (GANs):** GANs generate samples using a generator G that maps a noise variable $\mathbf{z} \sim p(\mathbf{z})$ to the data space. The discriminator D distinguishes between real data and generated samples. The objective function is:

$$\min_G \max_D V(G, D) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))],$$

where D and G play a minimax game to approximate the data distribution.

- **Two-Sample Tests:** These approaches can approximately optimize f-divergences (e.g., Jensen-Shannon, Kullback-Leibler) or Wasserstein distances to learn the generative model. Examples include methods based on Maximum Mean Discrepancy (MMD) or Sinkhorn distances.

1.3 Score Functions

1.3.1 Score Matching

In score-based generative models, the probability density $p_\phi(\mathbf{x})$ is parameterized through its gradient, known as the **score function**:

$$\nabla_{\mathbf{x}} \log p_\phi(\mathbf{x}).$$

These models use score matching techniques to estimate the score function, which can then be used in Langevin Dynamics or other sampling methods to generate samples.

$$= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[\frac{1}{2} \|\nabla_{\mathbf{x}} \log p_\phi(\mathbf{x})\|_2^2 + \text{tr}\left(\nabla_{\mathbf{x}}^2 \log p_\phi(\mathbf{x})\right) \right] + \text{const.}$$

Very flexible model architectures. But likelihood is intractable, training is unstable, hard to evaluate, and has mode collapse issues.

1.3.2 Noise-Perturbed Score Matching

Given a data distribution $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$, the goal of score matching is to estimate the score function $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$. In noise-perturbed score matching, a perturbation distribution $q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$ is applied to the data distribution, resulting in a noise-perturbed distribution $\tilde{\mathbf{x}} \sim q_\sigma(\tilde{\mathbf{x}})$.

The loss function for noise-perturbed score matching is expressed as:

$$\frac{1}{2}\mathbb{E}_{\tilde{\mathbf{x}} \sim p_{\text{data}}}\left[\|s_\phi(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}})\|_2^2\right], \tag{1}$$

where $s_\phi(\tilde{\mathbf{x}})$ is the estimated score function. This loss can be rewritten as:

$$\frac{1}{2}\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x}), \tilde{\mathbf{x}} \sim q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})}\left[\|s_\phi(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})\|_2^2\right] + \text{const.} \tag{2}$$

Assuming Gaussian perturbations, where $\tilde{\mathbf{x}} = \mathbf{x} + \sigma \mathbf{z}$ and $\mathbf{z} \sim \mathcal{N}(0, I)$, the loss simplifies to:

$$\frac{1}{2}\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}, \mathbf{z} \sim \mathcal{N}(0,I)}\left[\left\|s_\phi(\mathbf{x} + \sigma \mathbf{z}) + \frac{\mathbf{z}}{\sigma}\right\|_2^2\right] + \text{const.} \tag{3}$$

Pros:

- Much more scalable than standard score matching.
- Reduces score estimation to a denoising task.

Con:

- Estimates the score of noise-perturbed data rather than the true data distribution:

$$s_\phi(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x}) \neq \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}).$$

1.4 Diffusion Models

Diffusion models are generative models that learn to reverse a noising process applied to data. By adding noise to data progressively, these models learn to denoise and sample from the true data distribution. Different types of diffusion models are discussed below, each with its own mathematical formulation and sampling approach.

1.4.1 General Diffusion Models

General diffusion models define the forward process as a Markov chain that progressively adds Gaussian noise to the data. The reverse process is trained to reconstruct the original data from noise.

$$q(z_t|x) = \mathcal{N}(z_t; \sqrt{\alpha_t}x, (1 - \alpha_t)I)$$
$$p_\phi(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\phi(x_t, t), \sigma_t^2 I)$$

Where:

- x is the input data.
- z_t is the latent variable at time t.
- α_t controls the variance schedule.

The model is trained by minimizing the following MSE loss:

$$L = \mathbb{E}_{t,x,\epsilon}\left[\|\epsilon - \epsilon_\phi(z_t, t)\|^2\right]$$

1.4.2 Stochastic Differential Equations (SDEs)

In SDE-based diffusion models, the forward process is modeled as a continuous-time stochastic differential equation (SDE):

$$dx_t = f(x_t, t)\,dt + g(t)\,dw_t$$

The reverse process follows a similar but time-reversed SDE. The model is trained to predict the score function $\nabla_x \log p_t(x)$, which can be used to sample from the reverse SDE.

1.4.3 Probability Flow ODEs

Instead of using an SDE, probability flow ODEs use a deterministic ODE to model the reverse process. This approach avoids randomness, making sampling more efficient.

$$\frac{dx_t}{dt} = f(x_t, t) - \frac{1}{2}g(t)^2 \nabla_x \log p_t(x)$$

This ODE can be solved using standard numerical ODE solvers (like Euler or Runge-Kutta). Since there is no noise, it can generate precise samples with fewer steps than traditional SDEs.

1.4.4 Score-Based Diffusion Models

Score-based models use the score function $\nabla_x \log p_t(x)$ to define the reverse process. The forward process is similar to general diffusion, but the key difference is the use of score matching instead of direct likelihood estimation.

$$\nabla_{\mathbf{x}} \log p_\phi(\mathbf{x})$$

The model is trained to predict the score directly using score matching, and the reverse process is achieved using Langevin Dynamics or other MCMC-based methods.

1.4.5 Latent Diffusion Models

Latent diffusion models operate on a lower-dimensional latent space, unlike general diffusion models that operate in the pixel space. This allows for faster and more efficient sampling.

- **Forward Process:**
$$z_t = \sqrt{\alpha_t} z_{t-1} + \sqrt{1 - \alpha_t}\epsilon$$
- **Reverse Process:**
$$p_\phi(z_{t-1}|z_t) = \mathcal{N}(z_{t-1}; \mu_\phi(z_t, t), \sigma_t^2 I)$$

The latent space representation is learned using a Variational Autoencoder (VAE), and the noise is added directly in the latent space, reducing computational complexity.

1.4.6 Stable Diffusion Models

Stable diffusion is a specific type of latent diffusion model that introduces conditioning (e.g., via text prompts) to guide the generation process. It utilizes the following components:

1. Pre-trained VAE Encoder/Decoder: Encodes images into a compact latent space. 2. Diffusion Process in Latent Space: Adds noise and performs diffusion in a lower-dimensional space. 3. Conditioning: Enables control over the generation process (e.g., image prompts).

$$z_t = \sqrt{\alpha_t} z_{t-1} + \sqrt{1 - \alpha_t}\epsilon$$

Where z_t is the latent variable.

2 Distances of Probability Distributions

Various generative models optimize distances between probability distributions using different divergence measures. The most commonly used distances are as follows:

- **KL Divergence (Maximum Likelihood):**

$$D_{\mathrm{KL}}(p_{\mathrm{data}} \| p_\phi) = \int p_{\mathrm{data}}(x) \log \frac{p_{\mathrm{data}}(x)}{p_\phi(x)} dx.$$

Models that minimize KL divergence include:

  - **Autoregressive Models:**

$$\log p_\phi(\mathbf{x}) = \sum_{i=1}^{d} \log p_\phi(x_i \mid x_{<i})$$

  - **Normalizing Flow Models:**

$$\log p_\phi(\mathbf{x}) = \log p_Z(f_\phi^{-1}(\mathbf{x})) + \log \left| \det \left(\frac{\partial f_\phi^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right|$$

  - **Variational Autoencoders (VAEs):**

$$\mathcal{L}_{\mathrm{ELBO}} = \mathbb{E}_{q(z|x)} \left[\log p_\phi(x \mid z) \right] - D_{\mathrm{KL}}(q(z \mid x) \| p(z))$$

  - **Diffusion Models:**

$$D_{\mathrm{KL}}(q(z_t \mid x) \| p_\phi(z_t \mid x)) = \mathbb{E}_q \left[\log \frac{q(z_t \mid x)}{p_\phi(z_t \mid x)} \right]$$

    The objective is to minimize the reverse KL divergence, typically written as:

$$\mathbb{E}_{t,x,\epsilon} \left[\|\epsilon - \epsilon_\phi(z_t, t)\|^2 \right]$$

- **f-Divergences and Wasserstein Distances:** Models that optimize f-divergences (e.g., Jensen-Shannon divergence) or Wasserstein distances include:

– **Generative Adversarial Networks (GANs):**

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

– **Wasserstein GANs (WGANs):**

$$\min_G \max_{\|D\|_L \leq 1} \mathbb{E}_{x\sim p_{\text{data}}}[D(x)] - \mathbb{E}_{z\sim p(z)}[D(G(z))]$$

– **Diffusion Models:** Score-based models can be seen as minimizing an implicit Wasserstein distance, especially when using the Probability Flow ODE.

$$\frac{dx_t}{dt} = f(x_t, t) - \frac{1}{2}g(t)^2 \nabla_x \log p_t(x)$$

- **Fisher Divergence (Score Matching):** Fisher divergence is used in score-based methods to match gradients of log-probability densities. It is computed as:

$$D_{\text{Fisher}}(p_{\text{data}} \| p_\phi) = \mathbb{E}_{p_{\text{data}}}\left[\|\nabla_x \log p_{\text{data}}(x) - \nabla_x \log p_\phi(x)\|^2\right]$$

Examples of models include:

– **Energy-Based Models (EBMs):**

$$\nabla_x \log p_\phi(x) = -\nabla_x E_\phi(x)$$

– **Score-Based Diffusion Models:** The training objective minimizes the Fisher divergence using score matching, which can be written as:

$$\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\frac{1}{2}\|\nabla_{\mathbf{x}} \log p_\phi(\mathbf{x})\|_2^2 + \operatorname{tr}\left(\nabla_{\mathbf{x}}^2 \log p_\phi(\mathbf{x})\right)\right]$$

– **Diffusion Models (via Denoising Score Matching):** The loss for denoising score matching is:

$$\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}, \mathbf{z}\sim\mathcal{N}(0,I)}\left[\left\|s_\phi(\mathbf{x} + \sigma\mathbf{z}) + \frac{\mathbf{z}}{\sigma}\right\|_2^2\right]$$

- **Noise-Contrastive Estimation (NCE):** Noise-contrastive estimation is used as an alternative to maximum likelihood for training generative models. It is particularly useful for:

– **Energy-Based Models (EBMs):**

$$\log p_\phi(x) = \log \frac{\exp(-E_\phi(x))}{\int \exp(-E_\phi(x'))dx'}$$

– **Diffusion Models (via Reverse KL Divergence):** The objective for diffusion models is to minimize the reverse KL divergence:

$$D_{\text{KL}}(q(z_t \mid x) \| p_\phi(z_t \mid x)) = \mathbb{E}_q\left[\log \frac{q(z_t \mid x)}{p_\phi(z_t \mid x)}\right]$$

This is often rewritten as a mean squared error loss:

$$\mathbb{E}_{t,x,\epsilon}\left[\|\epsilon - \epsilon_\phi(z_t, t)\|^2\right]$$

3 Evaluating Generative Models

In any research field, **evaluation drives progress**. A critical question arises: *How do we evaluate generative models effectively?*

The evaluation of *discriminative models* (e.g., classifiers) is well understood. The common approach is to:

- Compare task-specific loss (e.g., top-1 accuracy).
- Evaluate performance on unseen test data.

Evaluating generative models, however, is **highly non-trivial**. Unlike discriminative models, there is no universal evaluation metric, as it often depends on the specific use case.

Key Question: What Is the Task That You Care About?

To evaluate generative models, the task being addressed plays a critical role. Common tasks include:

- **Density Estimation:** How well does the model approximate the data distribution? Caring about evaluating probabilities of images, texts.
- **Compression:** Can the model achieve efficient representation of the data?
- **Sampling/Generation:** Is the model capable of generating high-quality, realistic samples?
- **Latent Representation Learning:** Does the model learn useful latent representations that can be leveraged for downstream tasks? e.g., classification
- **Multi-Task Evaluation:** Custom tasks such as:
  - **Semi-Supervised Learning:** How effectively does the model use both labeled and unlabeled data for tasks like classification or regression?
  - **Image Translation:** Can the model transform images between domains, such as converting sketches into realistic images or day-time photos into night-time scenes?
  - **Compressive Sensing:** Can the model reconstruct signals or images from limited measurements, such as in medical imaging or sparse signal recovery?
- **Few-Shot/Zero-Shot Performance:** For models like LLMs (Large Language Models), how well do they perform through prompting without fine-tuning?

The choice of evaluation metrics and tasks should be driven by the specific goals of the application. Generative models often require a combination of quantitative and qualitative assessments to ensure robust evaluation.

4 Evaluation:

4.1 Density Estimation

4.1.1 Likelihood as a Metric for Density Estimation

Likelihood serves as a fundamental metric for evaluating the quality of density estimation in generative models. The process can be summarized as follows:

- **Split the Dataset:** Divide the dataset into training, validation, and test sets.
- **Learn the Model:** Train the model $p_\phi(\mathbf{x})$ on the training set.
- **Tune Hyperparameters:** Use the validation set to optimize hyperparameters.
- **Evaluate Generalization:** Compute the likelihood of the test set to assess generalization:

$$\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\log p_\phi(\mathbf{x})\right]$$

  which is an approximation to the average log−likelihood that the model assigns to the drawn samples from the data distribution.

This is essentially the same thing as compression as maximizing the likelihood is the same as minimizing the KL divergence which is the same as compressing the data.

4.1.2 Compression as a Measure of Model Efficiency

Generative models can also be evaluated by how efficiently they compress data. This involves mapping data points to shorter representations based on their likelihood. We can compare models based on how well they can compress

the data.

There are ways to take a probabilistic model and map it to a compression scheme where we want to map data point x to some string that can be decoded back in a unique way where the length of the string depends on the probability of the data point .

One way to achieve this via Shannon Coding

- Assign a codeword of length $\lceil \log \frac{1}{p_\phi(\mathbf{x})} \rceil$ to a data point $\mathbf{x}$.
- Intuitively, frequent data points $\mathbf{x}$ are assigned shorter codes and infrequent data points are assigned longer codes. This idea recalls the Morse Code system:

  $$\text{Example: } E = \cdot, \quad A = \cdot -, \quad Q = - - \cdot -$$

  if we train a generative model on maximum likelihood we are basically we are trying to do good in compression. Basically if we are comparing models based on their likelihood we are comparing them based on how well they compress.
- Average Code Length - length of the code that we assign to data point x is proportional to log of 1 over p then we can see that the average code length for data compressed by the model is given by:

  $$\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\lceil \log \frac{1}{p_\phi(\mathbf{x})} \rceil\right] \approx \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\log \frac{1}{p_\phi(\mathbf{x})}\right] = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[-\log p_\phi(\mathbf{x})\right]$$

  Basically if we are ignoring the fact that the length should be integer then we can approximate it via negative log−likelihood. This implies that better density estimation models result in shorter average code lengths.

4.1.3 Practical Considerations for Compression

- Shannon and Huffman codes are theoretically optimal but often intractable to construct for large datasets and they are expensive.
- Practical alternatives include compression schemes such as **Arithmetic Coding**, which approximates optimal performance.
- This is the same as Perplexity as a Metric for Language Models For language models e.g. GPT models, compression efficiency is often evaluated using **perplexity**, defined as:

  $$2^{-\frac{1}{D}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log p_\phi(\mathbf{x})]} \quad \text{for } \mathbf{x} \in \mathbb{R}^D$$

  Perplexity quantifies how well the model predicts sequences, with lower perplexity indicating better performance which is scaled version of the likelihood.

The question is why compression?

- Compression evaluates how well the model has identified patterns or redundancy in the data and the only way to have a good compression is by identifying the redundancy patterns which requires understanding structure in the data.
- If we are able to get a KL divergence to go to 0 that means we have learned a good compression of the data distribution.
- **Intuition:** If we think about physical laws (e.g., $F = ma$) it enables compression. Knowing force F allows us to exactly recover acceleration a through the equation without a need to have both.

Hutter Prize: A Benchmark for Compression The Hutter Prize offers $\$500,000$ for compressing Wikipedia effectively. This competition serves as a benchmark for measuring intelligence in compressors.

"Being able to compress well is closely related to intelligence. While intelligence is a slippery concept, file sizes are hard numbers. Wikipedia is an extensive snapshot of Human Knowledge. If you can compress the first 1GB of Wikipedia better than your predecessors, your (de)compressor likely has to be smart(er). The intention of this prize is to encourage the development of intelligent compressors/programs as a path to AGI."

Human vs. Model Performance

- Humans achieve 1.2 to 1.3 bits per character on text (Shannon, 1951).
- Large Language Models (LLMs) achieved 0.94 bits per character in 2019 which is better than humans.

A Critical Issue: Not All Bits Are Created Equal The problem is that maybe compression is not exactly what we want as not all bits hold the same importance. For example:

- A bit encoding **life (0) vs. death (1)** is more significant than a bit encoding **rain (0) vs. no rain (1)**.

So the compression is the same as optimizing for maximum likelihood or KL divergence but its not a reflective of downstream tasks. For example if we think of an image, changing a pixel color might not be important from perspective of compression but might be important for the downstream classification task which is the main limitation of the density estimation.

Compression or likelihood is straightforward for models with **tractable likelihoods**. Models that fall under this category can efficiently evaluate data compression through their ability to compute log-likelihoods.

Not all models have tractable likelihoods, for example:

- Variational Autoencoders (VAEs)
- Generative Adversarial Networks (GANs)
- Energy-Based Models (EBMs)

For VAEs, we can compare **evidence lower bounds (ELBO)** to log-likelihoods. However:

- **GANs:** How can we estimate the model likelihood when we only have generated samples?

Key Insight:

- In general, unbiased estimation of probability density functions from samples is impossible.
- *Approximation methods like MLE or kernel density estimation (KDE) could be used as a biased estimation.*

4.2 Kernel Density Estimation - KDE

Given: A model $p_\theta(x)$ with an intractable or ill-defined density. Let $\mathcal{S} = \{x^{(1)}, x^{(2)}, \ldots, x^{(6)}\}$ be 6 data points drawn from p_θ:

$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$x^{(4)}$	$x^{(5)}$	$x^{(6)}$
-2.1	-1.3	-0.4	1.9	5.1	6.2

Question: What is $p_\theta(-0.5)$?

Answer 1: Since $-0.5 \notin \mathcal{S}$, we conclude that:

$$p_\theta(-0.5) = 0$$

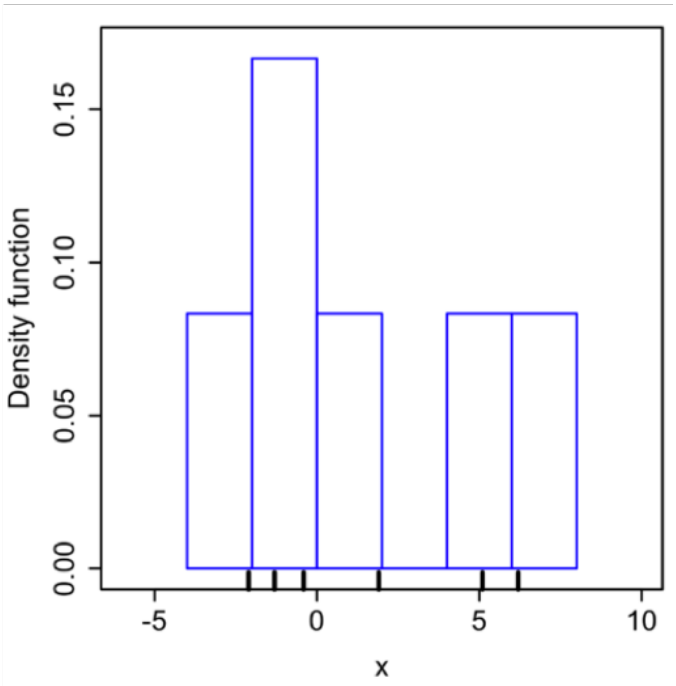
Answer 2: Compute a histogram by binning the samples.

- **Bin width:** This determines how the density is distributed over the range of the bins. Here we pick a bin width as 2
- **Minimum height:**

$$\text{Height of the bin} = \frac{\text{Probability mass in the bin}}{\text{Bin width}}$$
$$= \frac{\frac{1}{6}}{2} = \frac{1}{12}$$

What is $p_\theta(-0.5)$? To approximate $p_\theta(-0.5)$, the density can be estimated based on the binning of samples.

$$p_\theta(-0.5) = \frac{1}{6}, \quad p_\theta(-1.99) = \frac{1}{6}, \quad p_\theta(-2.01) = \frac{1}{12}$$



Answer 3: Compute Kernel Density Estimate (KDE) Over $\mathcal{S}$

$$\hat{p}(x) = \frac{1}{n} \sum_{x^{(i)} \in \mathcal{S}} K\left(\frac{x - x^{(i)}}{\sigma}\right)$$

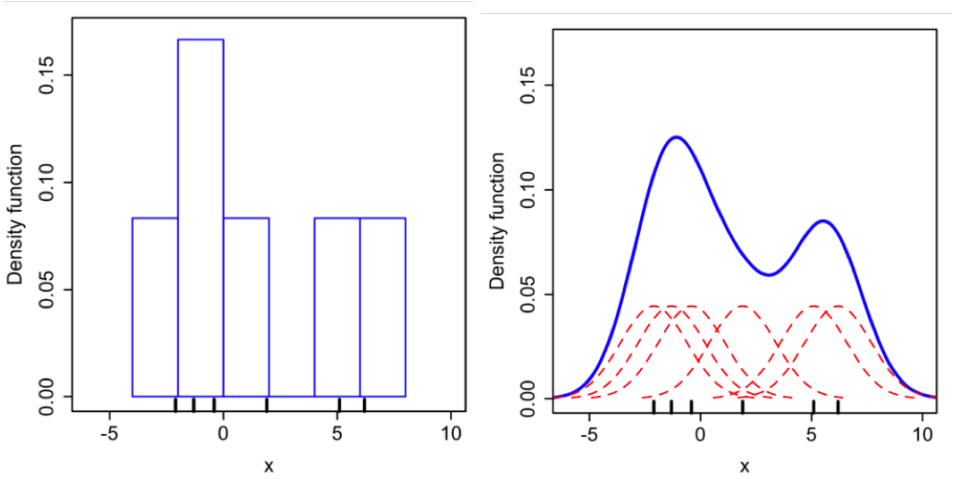
where:

- σ is the **bandwidth parameter**.
- K is the **kernel function** - Some sort of similarity.

Example: Gaussian Kernel The Gaussian kernel is defined as:

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}u^2\right)$$

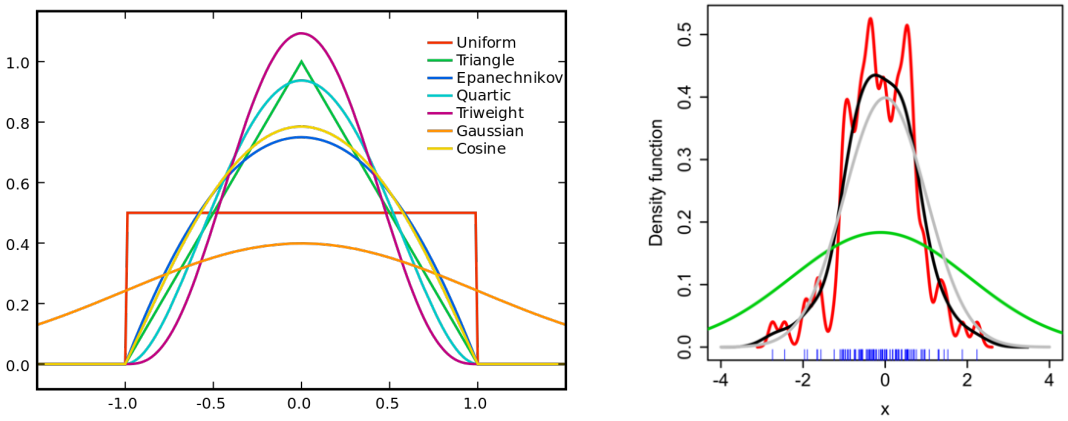
Using kernels we get a more natural / smoother interpolation. We do this by fitting a KDE to each sample (red curves) which can be used to generate an estimated likelihood (blue curve) for all samples.



(a) Histogram Density Estimate (b) KDE Estimate with Gaussian Kernel

Kernel Functions and Bandwidth in KDE

- A **kernel** K is any non-negative function satisfying two properties:
  - **Normalization:**
    $$\int_{-\infty}^{\infty} K(u)\,du = 1$$
    This ensures that the Kernel Density Estimate (KDE) is also normalized.
  - **Symmetry:**
    $$K(u) = K(-u) \quad \text{for all } u$$
- Intuitively, a kernel is a measure of similarity between pairs of points (u). The kernel function is higher when the difference between points is closer to 0.
- **Con:** KDE is very unreliable in higher dimensions.



Left: Examples of kernel functions. Right: Effect of bandwidth σ on KDE smoothness.

- **Bandwidth** σ controls the smoothness of the KDE. As shown in the figure (right):
  - **Optimal sigma (black):** KDE is close to the true density (grey curve).
  - **Low sigma (red curve):** Results in an undersmoothed KDE.
  - **High sigma (green curve):** Results in an oversmoothed KDE.
  - Bandwidth σ is often tuned using cross-validation.

4.3 Latent Variables

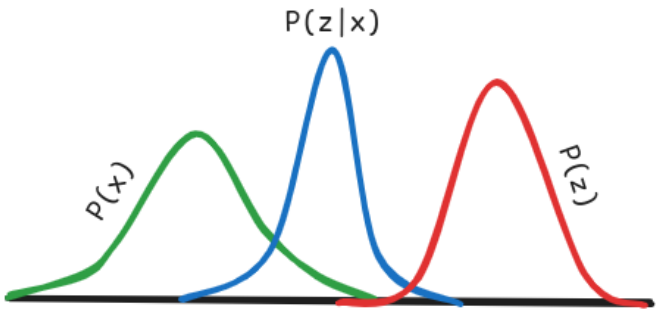
For latent variable models, determining the likelihood of a datapoint x theoretically involves integrating over all possible latent variables z. However, as previously discussed, this approach is computationally intractable.

4.3.1 Likelihood Weighting:

Assume a Gaussian likelihood function $p(x \mid z)$:

$$p(x) = \mathbb{E}_{p(z)}\left[p(x \mid z)\right]$$

However, this method can have **high variance** if $p(z)$ is far from $p(z \mid x)$.



4.3.2 Annealed Importance Sampling (AIS):

Construct a sequence of intermediate distributions that gradually interpolate from $p(z)$ to the unnormalized estimate of $p(z \mid x)$. This is a general-purpose technique to estimate ratios of normalizing constants $\mathcal{N}_2/\mathcal{N}_1$ for any two distributions using importance sampling.

Which is practically a procedure to draw z samples between the naive choice $p(z)$ (prior) and the optimal choice $p(z \mid x)(posterior)$.

For estimating $p(x)$:

- The first distribution is $p(z)$ (with $\mathcal{N}_1 = 1$).
- The second distribution is $p(x \mid z)$ (with $\mathcal{N}_2 = p(x) = \int_x p(x, z)\, dz$).

> It gives an unbiased estimate of the likelihood but a biased estimate of log−likelihood. This technique could provide a good estimate of likelihood for models like VAE.

A practical implementation of AIS is available in TensorFlow Probability under:

`tfp.mcmc.sample_annealed_importance_chain`

5 Evaluation - Sample Quality

$S_1 = \{\mathbf{x} \sim P\}$ $S_2 = \{\mathbf{x} \sim Q\}$

What if we have two generative−ai models and we dont cate about the likelihood or compression but interested in the quality of the images generated by them? This is not an obvious task and is pretty challenging.

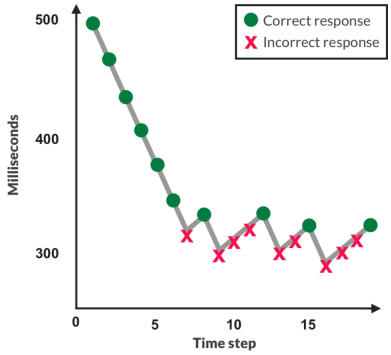
Which of these two sets of generated samples *"look"* better?

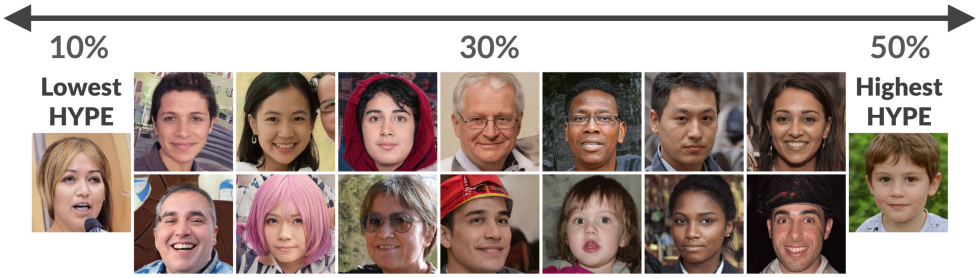
How about using:

- Human evaluations (e.g., Mechanical Turk)?

5.1 HYPE: Human Eye Perceptual Evaluation (Zhou et al., 2019)

- **HYPE$_{\text{time}}$:** The minimum time people needed to make accurate classifications. **The larger, the better.**
- **HYPE$_{\infty}$:** The percentage of samples that deceive people under unlimited time. **The larger, the better.**
- https://stanfordhci.github.io/gen-eval/





HYPE$_{\infty}$ scores for samples generated from a StyleGAN.

Challenges of human evaluation:

- They are expensive, biased, and hard to reproduce.
- It's hard to define and assess generalization: memorizing the training set would give excellent samples but is clearly undesirable.

In general quantitative evaluation of a qualitative task can have many answers. However, we could evaluate the quality of samples using popular metrics such as:

- Inception Scores
- Frechet Inception Distance
- Kernel Inception Distance

5.2 Inception Scores

Assumption 1: We are evaluating sample quality for generative models trained on labelled datasets.
Assumption 2: We have a good probabilistic classifier $c(y \mid \mathbf{x})$ for predicting the label y for any point $\mathbf{x}$.

We want samples from a good generative model to satisfy two criteria:

- **Sharpness**
- **Diversity**

The Inception Score evaluates p_θ by:

- Drawing samples $\mathbf{x} \sim p_\theta$. e.g. generate sample by the model.
- Using a pretrained classifier $c(y \mid \mathbf{x})$ to assess sharpness and diversity:
  - **Sharpness:** Assessed via $c(y \mid \mathbf{x})$, the classifier's confidence in predictions for individual samples.
  - **Diversity:** Assessed via the marginal $c(y) = \mathbb{E}_{\mathbf{x} \sim p_\theta}[c(y \mid \mathbf{x})]$, computed over all samples from p_θ.

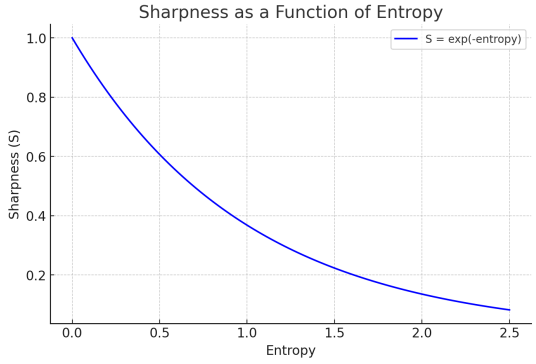
5.2.1 Sharpness (S):

Low sharpness | High sharpness

Sharpness is measured as:

$$S = \exp\left(\mathbb{E}_{\mathbf{x} \sim p_\theta}\left[\int c(y \mid \mathbf{x}) \log c(y \mid \mathbf{x})\, dy\right]\right)$$

- High sharpness implies the classifier is confident in making predictions for generated images.
- This means the classifier's predictive distribution $c(y \mid \mathbf{x})$ has low entropy.



Why is there no explicit negative term in the formula?
The integral term inside the formula:

$$\int c(y \mid \mathbf{x}) \log c(y \mid \mathbf{x})\, dy$$

already computes a **negative value**, because:

- $c(y \mid \mathbf{x})$ is a valid probability distribution ($0 \leq c(y \mid \mathbf{x}) \leq 1$).
- The logarithm of probabilities ($\log c(y \mid \mathbf{x})$) is always negative for $c(y \mid \mathbf{x}) \in (0, 1]$.
- The product $c(y \mid \mathbf{x}) \cdot \log c(y \mid \mathbf{x})$ results in a negative value.

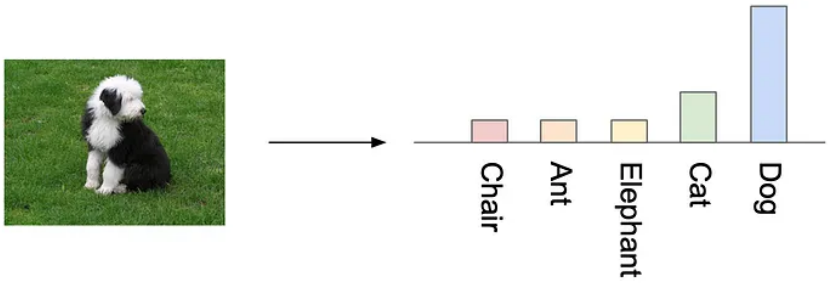
Thus, the integral itself represents the **negative entropy** of the classifier's predictive distribution $c(y \mid \mathbf{x})$.
Sharpness formula implicitly handles the negativity:

$$S = \exp\left(\mathbb{E}_{\mathbf{x}\sim p_\theta}\left[\int c(y \mid \mathbf{x}) \log c(y \mid \mathbf{x})\, dy\right]\right)$$

is equivalent to:

$$S = \exp\left(-H_{\text{avg}}\right),$$

where H_{avg} is the expected entropy. The absence of an explicit negative sign is due to the integral itself evaluating to a negative value.



Summary: - **Low entropy** of the classifier's predictive distribution leads to **high sharpness** S, reflecting confident and accurate predictions. - The formula implicitly accounts for the negativity of entropy, ensuring S increases as entropy decreases.

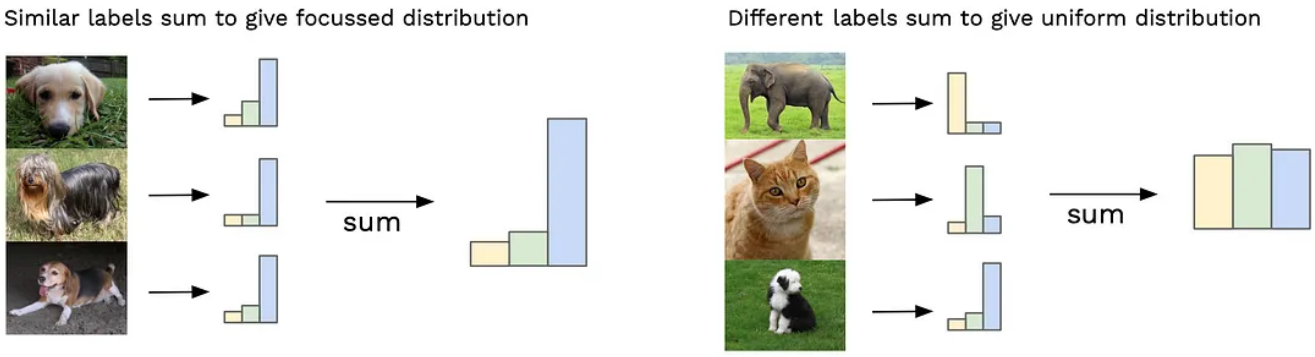
5.2.2 Diversity (D)

Diversity is measured as:

$$D = \exp\left(-\mathbb{E}_{\mathbf{x}\sim p_\theta}\left[\int c(y)\log c(y)\,dy\right]\right),$$

where $c(y) = \mathbb{E}_{\mathbf{x}\sim p_\theta}[c(y \mid \mathbf{x})]$ is the classifier's marginal predictive distribution.

- High diversity implies $c(y)$ has high entropy.
- This technically means the average number of times we predicted each class should be something uniform (or similar).
- However, this doesn't look at the prediction capability within each class. e.g. how often we predicted different breeds of dogs within the dog class. (mode failure within class)

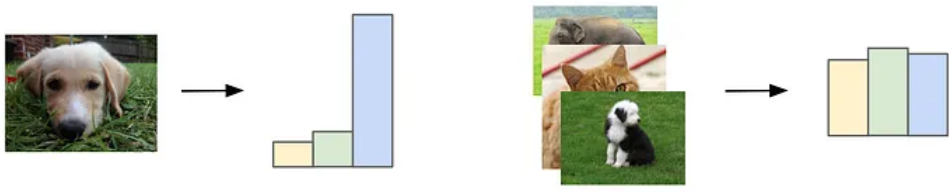


5.2.3 Incpection Score:

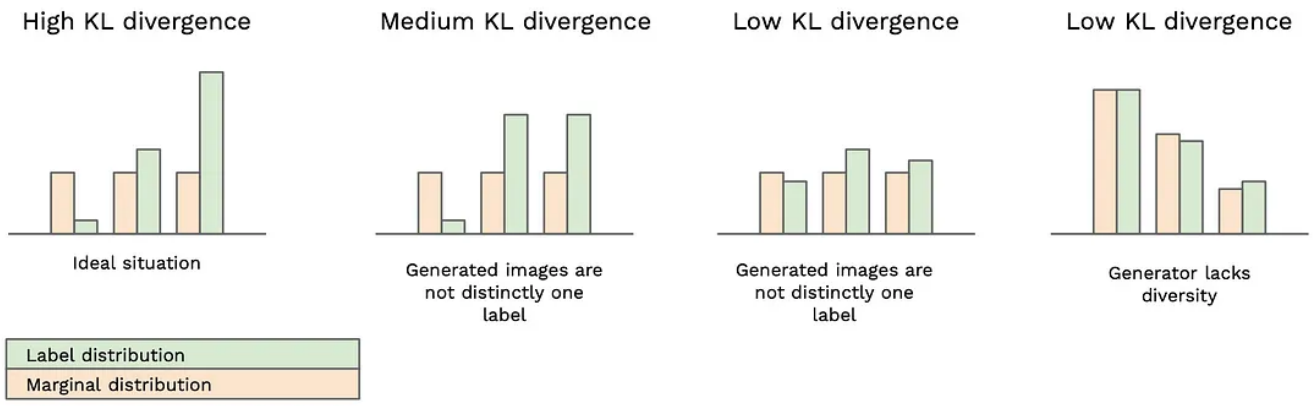
- Inception Scores (IS) combine the two criteria of sharpness and diversity into a simple metric:

$$IS = D \times S$$

- Higher IS corresponds to better quality.
- If a classifier is not available, a classifier trained on a large dataset, e.g., Inception Net trained on the ImageNet dataset, can be used.



5.2.4 Relationship Between Inception Score and KL Divergence



The Inception Score (IS) can be written as:

$$IS = \exp\left(\mathbb{E}_{\mathbf{x}\sim p_\theta}\left[\int c(y \mid \mathbf{x}) \log \frac{c(y \mid \mathbf{x})}{c(y)}\, dy\right]\right).$$

We want to show that this is equivalent to:

$$IS = S \times D,$$

where S represents **sharpness** and D represents **diversity**.

Sharpness (S):

$$S = \exp\left(-\mathbb{E}_{\mathbf{x}\sim p_\theta}\left[\int c(y \mid \mathbf{x}) \log c(y \mid \mathbf{x})\, dy\right]\right),$$

which measures how confident the classifier $c(y \mid \mathbf{x})$ is for individual images.

Diversity (D):

$$D = \exp\left(\int c(y \mid \mathbf{x}) \log c(y)\, dy\right).$$

$$=$$

$$\exp\left(-\int c(y) \log c(y)\, dy\right),$$

which measures the entropy of the marginal class distribution $c(y) = \mathbb{E}_{\mathbf{x}\sim p_\theta}[c(y \mid \mathbf{x})]$, ensuring diversity across generated samples.

Proof:

1. Starting with the definition of $S \times D$:

$$S \times D = \exp\left(-\mathbb{E}_{\mathbf{x}\sim p_\theta}\left[\int c(y \mid \mathbf{x}) \log c(y \mid \mathbf{x})\, dy\right]\right) \times \exp\left(-\int c(y) \log c(y)\, dy\right).$$

2. Combine the two exponential terms:

$$S \times D = \exp\left(-\mathbb{E}_{\mathbf{x}\sim p_\theta}\left[\int c(y \mid \mathbf{x}) \log c(y \mid \mathbf{x})\, dy\right] - \int c(y) \log c(y)\, dy\right).$$

3. Reorganize the terms to reflect the KL divergence:

$$S \times D = \exp\left(\mathbb{E}_{\mathbf{x}\sim p_\theta}\left[\int c(y \mid \mathbf{x}) \log \frac{c(y \mid \mathbf{x})}{c(y)}\, dy\right]\right).$$

4. This is exactly the definition of the Inception Score:

$$IS = \exp\left(\mathbb{E}_{\mathbf{x}\sim p_\theta}\left[\int c(y \mid \mathbf{x}) \log \frac{c(y \mid \mathbf{x})}{c(y)}\, dy\right]\right).$$

The Inception Score promotes:

- High-quality images (sharp predictions from the classifier).
- A diverse set of generated images (broad coverage of labels).

Conclusion:

- The Inception Score can be written as $IS = S \times D$, where S measures the sharpness of individual predictions and D measures the diversity of the overall class distribution.
- Both formulations are mathematically equivalent and promote high-quality (sharp and diverse) generated samples.

5.3 Frechet Inception Distance (FID)

- Inception Scores only require samples from p_θ (synthetic samples) and do not take into account the desired data distribution p_{data} (real data) directly (only implicitly via a classifier).
- **Frechet Inception Distance (FID)** measures similarities in the feature representations (e.g., those learned by a pretrained classifier) for datapoints sampled from p_θ and the test dataset (real data).
- Lower FID implies better sample quality.

- **Computing FID:**
  - Let $\mathcal{G}$ denote the generated samples and $\mathcal{T}$ denote the test dataset. Generally the p_θ is a NNs called **Inception Network**.
  - Compute feature representations $F_\mathcal{G}$ and $F_\mathcal{T}$ for $\mathcal{G}$ and $\mathcal{T}$, respectively (e.g., prefinal layer of Inception Net).
  - Fit a multivariate Gaussian to each of $F_\mathcal{G}$ and $F_\mathcal{T}$. Let $(\mu_\mathcal{G}, \Sigma_\mathcal{G})$ and $(\mu_\mathcal{T}, \Sigma_\mathcal{T})$ denote the means and covariances of the two Gaussians.
  - FID is defined as the Wasserstein-2 distance between these two Gaussians:

$$\text{FID} = \|\mu_\mathcal{T} - \mu_\mathcal{G}\|^2 + \text{Tr}(\Sigma_\mathcal{T} + \Sigma_\mathcal{G} - 2(\Sigma_\mathcal{T}\Sigma_\mathcal{G})^{1/2}).$$

  - The main reason we select multivariate Gaussian is that we can use Wasserstein-2 distance which has a closed form solution but its not particularly true in principal.

5.4 Kernel Inception Distance (KID)

Kernel Inception Distance (KID) is also computed over the features of samples, not the raw samples themselves. Like the Frechet Inception Distance (FID), KID compares the feature representations extracted from the generated samples and the test samples.

- **Maximum Mean Discrepancy (MMD)** is a two-sample test statistic that compares samples from two distributions p and q by computing differences in their moments (e.g., mean, variances, etc.).
- **Key idea:** Use a suitable kernel (e.g., Gaussian) to measure similarity between points.

- The MMD between two distributions p and q is given by:

$$\text{MMD}(p,q) = \underbrace{\mathbb{E}_{\mathbf{x},\mathbf{x}'\sim p}[K(\mathbf{x},\mathbf{x}')]}_{\text{similarity between fake samples}} + \underbrace{\mathbb{E}_{\mathbf{x},\mathbf{x}'\sim q}[K(\mathbf{x},\mathbf{x}')]}_{\text{similarity between real samples}} - \underbrace{2\mathbb{E}_{\mathbf{x}\sim p,\mathbf{x}'\sim q}[K(\mathbf{x},\mathbf{x}')]}_{\text{similarity between real and fake samples}}.$$

- Intuitively, MMD compares the "similarity" between samples within p and q individually to the samples from the mixture of p and q.
- **Kernel Inception Distance (KID):** Compute the MMD in the feature space of a classifier (e.g., Inception Network).

5.4.1 FID vs. KID:

- FID is biased (can only be positive), whereas KID is unbiased.
- FID can be evaluated in $O(n)$ time, whereas KID evaluation requires $O(n^2)$ time.

6 Evaluating Sample Quality for Text-to-Image Models

- **Metrics for Quality Evaluation:** Evaluating the quality of text-to-image models involves multiple metrics such as:
  - Quality metrics: FID, Inception Score, KID, etc.
  - Alignment with provided captions: CLIPScore.
  - Biases and fairness evaluations.
- **Holistic Evaluation of Text-to-Image Models (HEIM):**
  - HEIM provides a comprehensive evaluation framework for text-to-image models.
  - Covers 26 models, 29 scenarios, and 33 metrics (both automated and human evaluations).
  - Includes metrics for alignment, aesthetics, originality, reasoning, knowledge, bias, toxicity, fairness, robustness, multilingualism, and efficiency.

Examples of Evaluation Aspects:

- **Alignment:** CLIPScore and overall alignment between the generated image and the provided caption.
- **Quality:** Metrics like FID and photorealism.
- **Aesthetics:** LAION aesthetic scores, facial coherence, and spatial clarity.
- **Originality:** Watermark detection and overall originality of the generated content.
- **Reasoning:** Object detection and logical consistency in generated images.
- **Knowledge:** CLIPScore-based evaluations for general knowledge representation.
- **Bias and Fairness:** Evaluates gender proportion, skin tone proportion, and fairness of the generated content.
- **Robustness:** Checks resilience against variations in input prompts.
- **Multilingualism:** Tests the model's capability to handle prompts in multiple languages.
- **Efficiency:** Measures inference time and overall computational efficiency.

More Information:

- HEIM: https://crfm.stanford.edu/heim/latest/

7 Evaluating Latent Representations

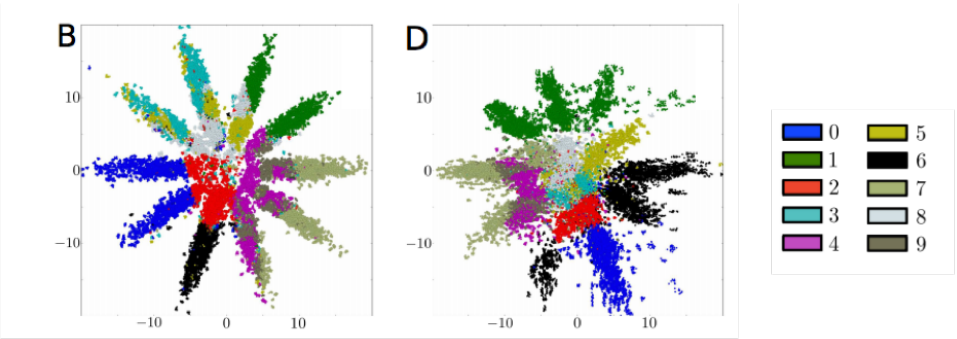
What does it mean to learn "good" latent representations?

- For a downstream task, the representations can be evaluated based on the corresponding performance metrics, e.g., accuracy for semi-supervised learning or reconstruction quality for denoising.
- For unsupervised tasks, there is no one-size-fits-all approach.

Three commonly used notions for evaluating unsupervised latent representations:

- Clustering
- Compression
- Disentanglement

8 Clustering



Source: Makhzani et al., 2018

- Representations that can group together points based on some semantic attribute are potentially useful (e.g., semi-supervised classification).
- Clusters can be obtained by applying k-means or any other algorithm in the latent space of the generative model.

8.1 Metrics for Clustering Evaluation

- For labeled datasets, there exist many quantitative evaluation metrics.
- Labels are only used for evaluation, not for obtaining clusters themselves (clustering is unsupervised).
- `from sklearn.metrics.cluster import completeness_score, homogeneity_score, v_measure_score`

Example metrics:

- Completeness score (between 0 and 1): Maximized when all data points that are members of a given class are elements of the same cluster.
- Homogeneity score (between 0 and 1): Maximized when all clusters contain only data points that are members of a single class.
- V-measure score: Harmonic mean of completeness and homogeneity scores.

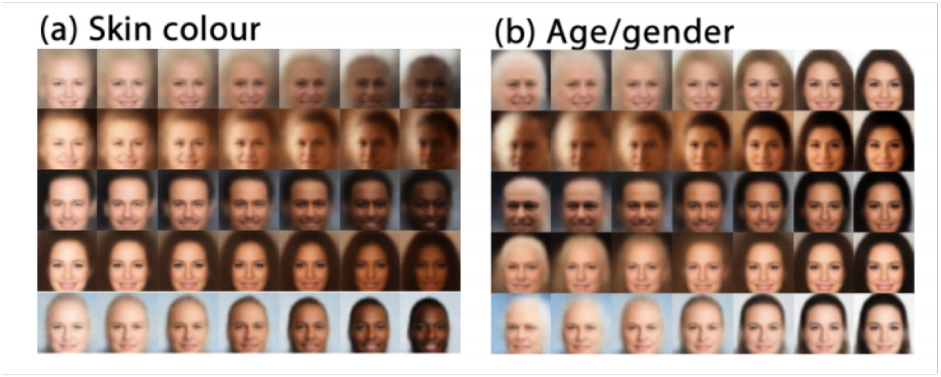
9 Lossy Compression or Reconstruction

Source: Santurkar et al., 2018

- Latent representations can be evaluated based on the maximum compression they can achieve without significant loss in reconstruction accuracy.

10 Disentanglement

- Intuitively, we want representations that disentangle **independent and interpretable** attributes of the observed data.



Source: Higgins et al., 2018

Examples of disentanglement for attributes such as skin color and age/gender. Source: Higgins et al., 2018.

This provides user control over the attributes of the generated data:

- When one latent variable Z_1 is fixed, the size of the generated object does not change.
- When Z_1 is changed, the change is restricted to the size of the generated object.

10.1 Quantitative Evaluation Metrics for Disentanglement

- Many quantitative evaluation metrics:
  - **Beta-VAE metric (Higgins et al., 2017):** Accuracy of a linear classifier that predicts a fixed factor of variation.
  - Many other metrics: Factor-VAE metric, Mutual Information Gap, SAP score, DCI disentanglement, Modularity.

  - Check `disentanglement_lib` for implementations of these metrics.

- Disentangling generative factors using only unlabeled data is theoretically impossible without additional assumptions.

Despite some empirical success its not well understood why these approaches work and some theoretical results suggest that these approaches might not really work.

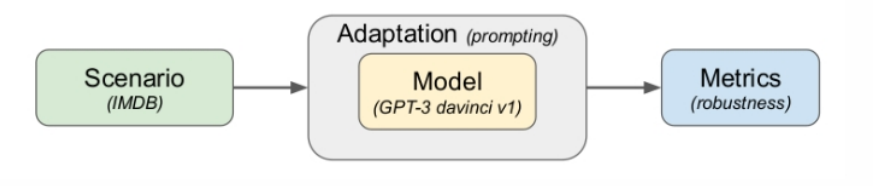
10.2 Beta-VAE

The Beta-VAE metric evaluates the disentanglement of latent representations learned by a generative model, such as a Variational Autoencoder (VAE). Below is the detailed explanation:

- **Latent Factors of Variation:** Many datasets have underlying generative factors (e.g., rotation, size, color, or shape of an object). These are the "true factors" responsible for creating variations in the data.
- **Goal of Disentanglement:** A well-disentangled representation maps each of these factors to a separate latent variable in the model. For example, one latent variable may represent "rotation," another may represent "color," with minimal overlap or entanglement between variables.
- **Latent Variables to Predict Over:**
  - The latent space of the generative model consists of variables $\mathbf{z} = [z_1, z_2, \ldots, z_n]$.
  - The Beta-VAE metric evaluates how well each latent variable z_i corresponds to one specific *ground truth generative factor* (e.g., rotation, color, size, etc.).
  - For example:
    * z_1: Encodes the rotation of an object.
    * z_2: Encodes the size of an object.
    * z_3: Encodes the object's color.
- **How the Beta-VAE Metric Works:**
  - A **linear classifier** is trained on the latent representations $\mathbf{z}$ to predict one of the ground truth generative factors (e.g., "color" or "rotation").
  - The **accuracy** of this classifier indicates how well the latent variable captures the specific factor of variation:
    * **High accuracy:** The latent variable strongly corresponds to the factor, suggesting good disentanglement.
    * **Low accuracy:** The latent variable does not effectively capture the factor, indicating poor disentanglement.
- **Procedure:**
  - The generative model is first trained on the dataset.
  - After training, the encoder produces latent representations $\mathbf{z} = [z_1, z_2, \ldots, z_n]$ for the data points.
  - A linear classifier is trained for each ground truth factor to predict it using the corresponding z_i.
  - The performance of the classifier determines how well z_i captures the specific factor of variation.

Summary: The Beta-VAE metric measures disentanglement by assessing how accurately a linear classifier can predict a single generative factor (e.g., "color" or "size") based on the learned latent representations. High accuracy suggests that the model has learned a well-disentangled representation.

11 Solving Tasks Through Prompting



A Language Model as a Generative Model of Text: A language model generates text by modeling the conditional probability:

$$p_\theta(\text{next word} \mid \text{sentence}).$$

A popular idea in recent times is that when working with language models (LMs), we might not necessarily need to follow the traditional process of mapping data to a latent space and using these latent representations to improve performance on downstream tasks.
Instead, if we have a generative language model, we can directly leverage it to solve language-related tasks by specifying the task in natural language. This approach highlights two distinct ways of utilizing a generative model:

- **Latent Representation Approach:** Train the model in an unsupervised manner, allowing it to uncover meaningful latent representations of the data. These representations can then be used to train a classifier for specific tasks.
- **Prompting Approach:** Pre-train the model on a large corpus of unlabeled data and adapt it using prompting techniques. By carefully crafting prompts, the model can directly solve a wide variety of tasks without the need for additional fine-tuning.

We know if language models which are trained by maximum likelihood (same as compression) and doing well that means they have learns something about the structure of the data and they have memorized lot of interesting things about the data and the hope is that we can leverage these learning's for different kinds of downstream tasks.

For instance, in **sentiment classification**, given a text (e.g., movie review), the goal is to predict the sentiment (positive or negative).

- **Task:** Classify the sentiment of movie reviews as either "Positive" or "Negative".
- **Choose sentence** = *Classify the sentiment of the movie reviews below as either "Positive" or "Negative".*

  Example 1 Movie Review: This has got to be one of the best episodes of Doctor Who... I cannot WAIT until next week's episode to find how they get out of this mess. **Sentiment: Positive**

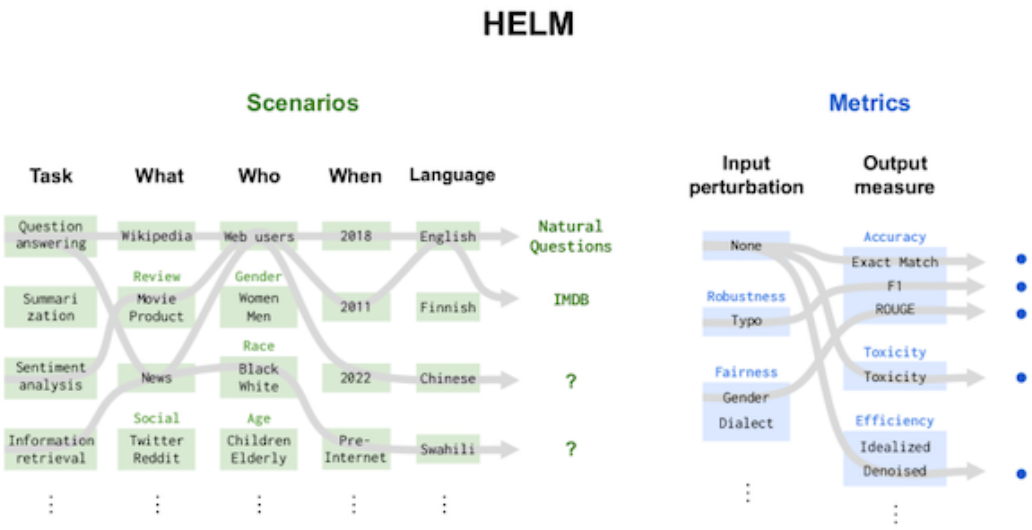
  Example 2 Movie Review: The fifth collaboration between... in such a dull, incoherent film. **Sentiment: Negative**

  Example 3 Movie Review: I viewed the first two nights before coming to IMDb... Now I am not so sure. **Sentiment:**(___)
- Using $p_\theta(\text{next word} \mid \text{sentence})$, the model predicts the next word (___). Is it "Positive" or "Negative"?

Adaptation Strategies:

- Many prompting strategies can be employed (Prompt Engineering).
- Fine-tuning the model is also widely used for specific tasks.

12 Holistic Evaluation of Language Models (HELM)



Evaluation Across Scenarios and Metrics: HELM provides a comprehensive evaluation framework for language models by testing on various scenarios and metrics.

- Tasks include question answering, summarization, sentiment analysis, information retrieval, etc.
- Scenarios are characterized by **what**, **who**, **when**, and **language**.
- Metrics include accuracy, calibration, robustness, fairness, bias, toxicity, and efficiency.

Examples:

- **Task: Sentiment Analysis.**
  - Scenario: News articles about race.
  - Metrics: Fairness (e.g., gender/dialect biases), robustness, and accuracy.
- **Task: Summarization.**
  - Scenario: Summarizing movie or product reviews.
  - Metrics: ROUGE, efficiency (e.g., inference time).

HELM Resources:

- Holistic evaluation of language models: `https://crfm.stanford.edu/helm/latest/`
  - 73 scenarios, 65 metrics, 81 models.
- BigBench: `https://github.com/google/BIG-bench`
  - Over 200 tasks including QA, reasoning, math, riddles, etc.

References

[1] Stefano Ermon. CS236.