Statistics 414 – From Predictive AI to Generative AI

Xiaofeng Lin
bernardo1998@g.ucla.edu

Department of Statistics & Data Science, UCLA

Februray 20, 2025

Introduction to Sampling from Distributions

- **Goal**: To sample from a target distribution to model real-world data.
  - Example: Sampling from the distribution of real-world images or complex patterns.
- **Simple Known Distributions**:
  - When we know the probability density function (pdf), methods like **inverse CDF** and **rejection sampling** allow us to sample effectively.
- **Example of Sampling Techniques**:
  - First, sample from a simpler, known distribution (e.g., **uniform** or **normal**).
  - Then, transform these samples to match the target distribution.

Neural Networks for Complex Real Distributions

- **Challenge with Real Data**:
  - Complex real-world data (e.g., images) lacks a straightforward pdf or cdf.
- **Neural Networks as Approximation**:
  - Use neural networks to approximate complex distributions and learn mappings from a simple proposal distribution (usually **normal**) to the target distribution.
- **Why Normal Distribution as Seed?**:
  - The normal distribution is **smooth, continuous**, and **symmetric**, making it efficient and easier to map to complex data distributions compared to a uniform distribution.

Latent Variable Models in Generative Modeling

- **Latent Variable Idea:** Generative models often use latent variables z to represent complex dependencies within data. The model samples a latent variable z (e.g., which digit to generate) and then uses it to guide the generation of realistic data samples X.
- **Generating Samples:** The generation process can be viewed as a deterministic function $f(z;\theta)$ where θ represents model parameters. Given $z \sim P(z)$, the model optimizes θ to ensure $f(z;\theta)$ resembles data points in the training set.
- **Modeling Objectives:** The model aims to maximize the probability of data X by integrating over latent variables:

$$P(X) = \int P(X|z;\theta)P(z)\,dz$$

- **Presentation Outline:** We will explore three types of latent variable models in ascending order of complexity:
  1. **GANs (Generative Adversarial Networks)**
  2. **VAEs (Variational Autoencoders)**
  3. **DDPMs (Denoising Diffusion Probabilistic Models)**

Training Pipeline of GAN

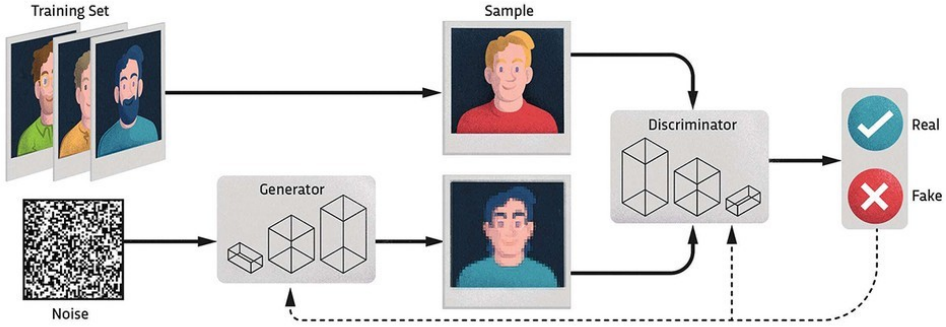


Figure: Step 1: discriminator learns to distinguish the real data sample from the fake ones. Step 2: generator learn to improve the quality of fake samples to mimic real data points.

Training Steps of a GAN Network

GAN Loss Function:

$$\min_{\theta_G} \max_{\theta_D} \left(\mathbb{E}_{x \sim p_{\text{data}}} [\log D_{\theta_D}(x)] + \mathbb{E}_{z \sim p(z)} [\log(1 - D_{\theta_D}(G_{\theta_G}(z)))] \right)$$

- **Initialize:** Start with random initial weights networks.

Training Steps of a GAN Network

GAN Loss Function:

$$\min_{\theta_G} \max_{\theta_D} \left(\mathbb{E}_{x \sim p_{\text{data}}} [\log D_{\theta_D}(x)] + \mathbb{E}_{z \sim p(z)} [\log(1 - D_{\theta_D}(G_{\theta_G}(z)))] \right)$$

- **Initialize:** Start with random initial weights networks.
- **Train Discriminator:**

Training Steps of a GAN Network

GAN Loss Function:

$$\min_{\theta_G} \max_{\theta_D} \left(\mathbb{E}_{x \sim p_{\text{data}}} [\log D_{\theta_D}(x)] + \mathbb{E}_{z \sim p(z)} [\log(1 - D_{\theta_D}(G_{\theta_G}(z)))] \right)$$

- **Initialize:** Start with random initial weights networks.
- **Train Discriminator:**
  - Combine generated fake data from the generator with real data.

Training Steps of a GAN Network

GAN Loss Function:

$$\min_{\theta_G} \max_{\theta_D} \left(\mathbb{E}_{x \sim p_{\text{data}}} [\log D_{\theta_D}(x)] + \mathbb{E}_{z \sim p(z)} [\log(1 - D_{\theta_D}(G_{\theta_G}(z)))] \right)$$

- **Initialize:** Start with random initial weights networks.
- **Train Discriminator:**
  - Combine generated fake data from the generator with real data.
  - Train the discriminator to classify real data as real and fake data as fake, adjusting its weights to minimize classification errors.

Training Steps of a GAN Network

GAN Loss Function:

$$\min_{\theta_G} \max_{\theta_D} \left(\mathbb{E}_{x \sim p_{\text{data}}}[\log D_{\theta_D}(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D_{\theta_D}(G_{\theta_G}(z)))] \right)$$

- **Initialize:** Start with random initial weights networks.
- **Train Discriminator:**
  - Combine generated fake data from the generator with real data.
  - Train the discriminator to classify real data as real and fake data as fake, adjusting its weights to minimize classification errors.
- **Train Generator:**

Training Steps of a GAN Network

GAN Loss Function:

$$\min_{\theta_G} \max_{\theta_D} \left(\mathbb{E}_{x \sim p_{\text{data}}}[\log D_{\theta_D}(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D_{\theta_D}(G_{\theta_G}(z)))] \right)$$

- **Initialize:** Start with random initial weights networks.
- **Train Discriminator:**
  - Combine generated fake data from the generator with real data.
  - Train the discriminator to classify real data as real and fake data as fake, adjusting its weights to minimize classification errors.
- **Train Generator:**
  - Freeze the discriminator's weights and pass the fake data through the discriminator again.

Training Steps of a GAN Network

GAN Loss Function:

$$\min_{\theta_G} \max_{\theta_D} \left(\mathbb{E}_{x \sim p_{\text{data}}}[\log D_{\theta_D}(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D_{\theta_D}(G_{\theta_G}(z)))] \right)$$

- **Initialize:** Start with random initial weights networks.
- **Train Discriminator:**
  - Combine generated fake data from the generator with real data.
  - Train the discriminator to classify real data as real and fake data as fake, adjusting its weights to minimize classification errors.
- **Train Generator:**
  - Freeze the discriminator's weights and pass the fake data through the discriminator again.
  - Adjust the generator's weights to maximize the discriminator's classification error.

Training Steps of a GAN Network

GAN Loss Function:

$$\min_{\theta_G} \max_{\theta_D} \left(\mathbb{E}_{x \sim p_{\text{data}}} [\log D_{\theta_D}(x)] + \mathbb{E}_{z \sim p(z)} [\log(1 - D_{\theta_D}(G_{\theta_G}(z)))] \right)$$

- **Initialize:** Start with random initial weights networks.
- **Train Discriminator:**
  - Combine generated fake data from the generator with real data.
  - Train the discriminator to classify real data as real and fake data as fake, adjusting its weights to minimize classification errors.
- **Train Generator:**
  - Freeze the discriminator's weights and pass the fake data through the discriminator again.
  - Adjust the generator's weights to maximize the discriminator's classification error.
- **Repeat:** Iterate steps 2-4.

Challenges in Training GANs

- **Mode Collapse:** A situation where the generator learns to produce a limited variety of samples, or even the same sample, regardless of the input noise. This happens because the generator finds a particular data point that consistently fools the discriminator, thus missing out on the broader data distribution.

Challenges in Training GANs

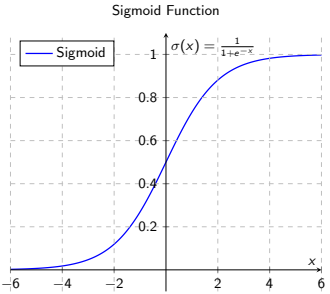
- **Mode Collapse:** A situation where the generator learns to produce a limited variety of samples, or even the same sample, regardless of the input noise. This happens because the generator finds a particular data point that consistently fools the discriminator, thus missing out on the broader data distribution.
- **Lack of Diversity:** Closely related to mode collapse, lack of diversity refers to the generator's failure to capture the full complexity and variability of the data distribution.

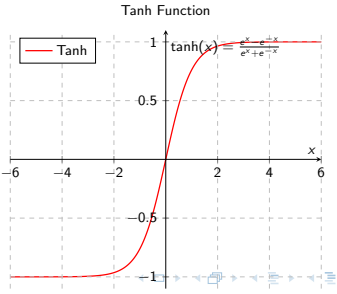
Challenges in Training GANs

- **Mode Collapse:** A situation where the generator learns to produce a limited variety of samples, or even the same sample, regardless of the input noise. This happens because the generator finds a particular data point that consistently fools the discriminator, thus missing out on the broader data distribution.
- **Lack of Diversity:** Closely related to mode collapse, lack of diversity refers to the generator's failure to capture the full complexity and variability of the data distribution.
- **Vanishing Gradients:** A problem where the discriminator becomes too effective, correctly classifying all samples. As a result, the gradient the generator receives during training becomes very small or vanishes, making it difficult for the generator to improve further.

Challenges in Generating Tabular Data with GANs - Part 1

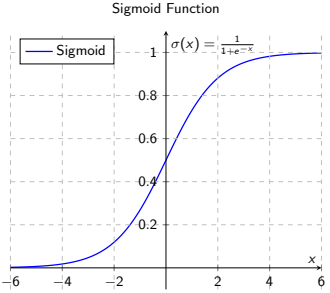
- **Mixed Data Types:** Real-world tabular data often comprises both discrete and continuous columns. GANs face the challenge of generating these mixed types simultaneously, typically requiring the application of different activation functions.

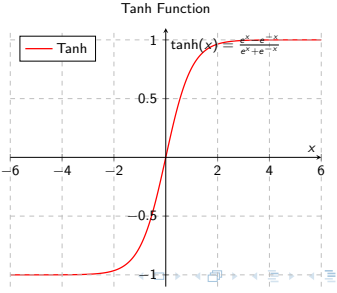




Challenges in Generating Tabular Data with GANs - Part 1

- **Mixed Data Types:** Real-world tabular data often comprises both discrete and continuous columns. GANs face the challenge of generating these mixed types simultaneously, typically requiring the application of different activation functions.
- **Non-Gaussian Distributions:** Unlike pixel values in images that follow a Gaussian-like distribution, continuous values in tabular data are usually non-Gaussian. Applying a min-max transformation and a tanh can lead to the vanishing gradient problem.

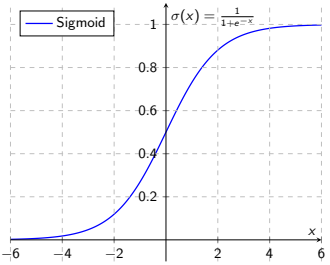




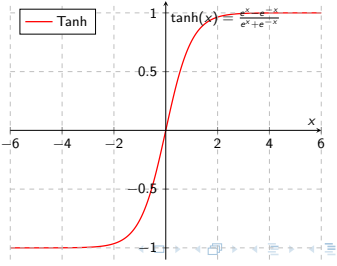
Challenges in Generating Tabular Data with GANs - Part 1

- **Mixed Data Types:** Real-world tabular data often comprises both discrete and continuous columns. GANs face the challenge of generating these mixed types simultaneously, typically requiring the application of different activation functions.
- **Non-Gaussian Distributions:** Unlike pixel values in images that follow a Gaussian-like distribution, continuous values in tabular data are usually non-Gaussian. Applying a min-max transformation and a tanh can lead to the vanishing gradient problem.
- **Multimodal Distributions:** Real-world datasets often exhibit multimodal distributions in continuous columns, challenging GANs to model all modes effectively.

Sigmoid Function



Tanh Function



Challenges in Generating Tabular Data with GANs - Part 1

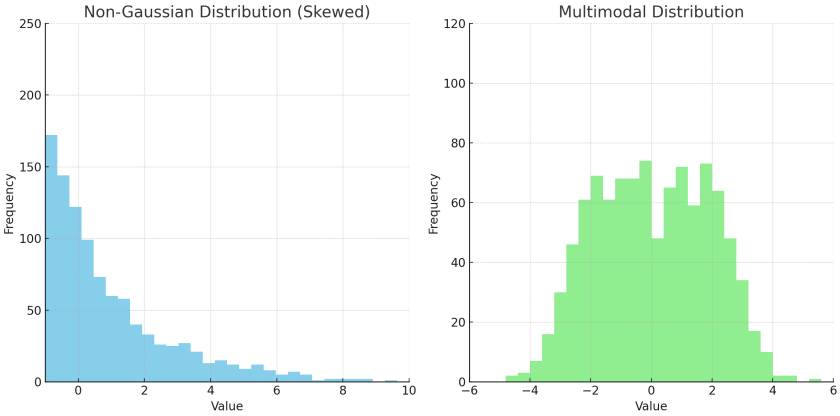


Figure: Left: Non-Gaussian (Skewed) Distribution. Right: Multimodal Distribution.

Challenges in Generating Tabular Data with GANs - Part 2

Which Vector is Real?

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0.025 & 0.9 & 0.025 & 0.025 & 0.025 \end{bmatrix}$$

Challenges in Generating Tabular Data with GANs - Part 2

Which Vector is Real?

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0.025 & 0.9 & 0.025 & 0.025 & 0.025 \end{bmatrix}$$

- **Learning from Sparse One-Hot-Encoded Vectors:** Generative models trained on synthetic samples must generate a probability distribution over all categories using softmax, while real data is represented in sparse one-hot vectors. This discrepancy makes it easy for a trivial discriminator to distinguish between real and fake data by checking the sparseness of the distribution rather than its authenticity.

Challenges in Generating Tabular Data with GANs - Part 2

Which Vector is Real?

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0.025 & 0.9 & 0.025 & 0.025 & 0.025 \end{bmatrix}$$

- **Learning from Sparse One-Hot-Encoded Vectors:** Generative models trained on synthetic samples must generate a probability distribution over all categories using softmax, while real data is represented in sparse one-hot vectors. This discrepancy makes it easy for a trivial discriminator to distinguish between real and fake data by checking the sparseness of the distribution rather than its authenticity.
- **Highly Imbalanced Categorical Columns:** Many datasets contain categorical columns with a dominant category appearing in more than 90% of rows. This imbalance leads to severe mode collapse, where missing minor categories cause only minor changes in the data distribution, making it difficult for the discriminator to detect and for the model to learn effectively from minor classes.

CTGAN: Conditional GAN for Tabular Data Generation

CTGAN: Conditional Generative Adversarial Network for Tabular Data
CTGAN [Xu+19] tackles key challenges in tabular generation, notably the handling of mixed data types and multimodal distributions.

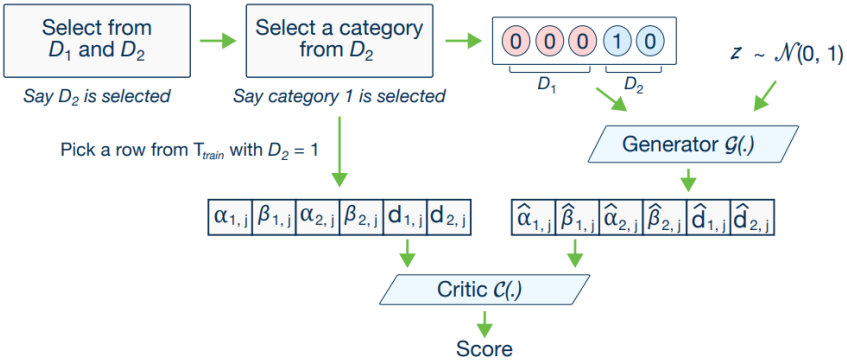
- **Mode-Specific Normalization:** CTGAN implements a unique preprocessing step that normalizes data based on its mode. This approach is particularly effective in addressing the challenge of non-Gaussian and multimodal distributions in continuous columns, enabling the model to learn and generate more accurate synthetic data.

CTGAN: Conditional GAN for Tabular Data Generation

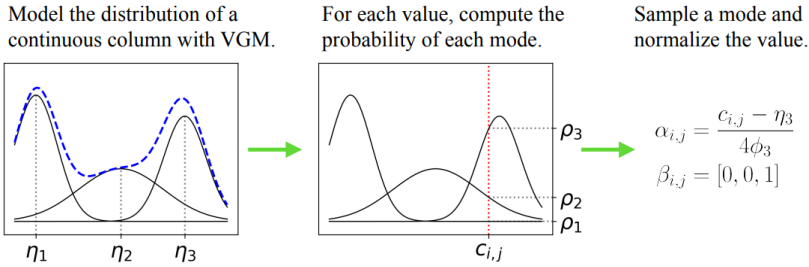
CTGAN: Conditional Generative Adversarial Network for Tabular Data
CTGAN [Xu+19] tackles key challenges in tabular generation, notably the handling of mixed data types and multimodal distributions.

- **Mode-Specific Normalization:** CTGAN implements a unique preprocessing step that normalizes data based on its mode. This approach is particularly effective in addressing the challenge of non-Gaussian and multimodal distributions in continuous columns, enabling the model to learn and generate more accurate synthetic data.
- **Conditional GAN + Train-by-Sampling:** Leveraging a conditional GAN framework, CTGAN can better manage the complexity of tabular data, including the presence of categorical variables with high cardinality and sparse matrices. The train-by-sampling technique further enhances the model's ability to focus on and learn from underrepresented modes, significantly reducing the risk of mode collapse and improving the diversity of the generated data.

CTGAN: Overview



Mode-Specific Normalization in CTGAN



Significance: This mode-specific normalization allows CTGAN to effectively handle the complexity of continuous columns with multiple modes, enhancing the diversity and realism of generated synthetic data.

Mode-Specific Normalization in CTGAN

Overview of Mode-Specific Normalization Process

1. **Identify Modes with VGM:** For each continuous column C_i, a Variational Gaussian Mixture (VGM) model is used to estimate the number of modes m_i and fit a Gaussian mixture. The Gaussian mixture for column C_i, given a data point $c_{i,j}$, is represented as $P_{C_i}(c_{i,j}) = \sum_{k=1}^{m_i} \mu_k \mathcal{N}(c_{i,j}; \eta_k, \phi_k)$, where μ_k, η_k, and ϕ_k denote the weight, mean, and standard deviation of mode k, respectively.

Mode-Specific Normalization in CTGAN

Overview of Mode-Specific Normalization Process

1. **Identify Modes with VGM:** For each continuous column C_i, a Variational Gaussian Mixture (VGM) model is used to estimate the number of modes m_i and fit a Gaussian mixture. The Gaussian mixture for column C_i, given a data point $c_{i,j}$, is represented as $P_{C_i}(c_{i,j}) = \sum_{k=1}^{m_i} \mu_k \mathcal{N}(c_{i,j}; \eta_k, \phi_k)$, where μ_k, η_k, and ϕ_k denote the weight, mean, and standard deviation of mode k, respectively.
2. **Compute Mode Probabilities:** For each value $c_{i,j}$ in C_i, calculate the probability of $c_{i,j}$ belonging to each mode. This is done by computing probability densities $\rho_k = \mu_k \mathcal{N}(c_{i,j}; \eta_k, \phi_k)$ for each mode k.

Mode-Specific Normalization in CTGAN

Overview of Mode-Specific Normalization Process

1. **Identify Modes with VGM:** For each continuous column C_i, a Variational Gaussian Mixture (VGM) model is used to estimate the number of modes m_i and fit a Gaussian mixture. The Gaussian mixture for column C_i, given a data point $c_{i,j}$, is represented as $P_{C_i}(c_{i,j}) = \sum_{k=1}^{m_i} \mu_k \mathcal{N}(c_{i,j}; \eta_k, \phi_k)$, where μ_k, η_k, and ϕ_k denote the weight, mean, and standard deviation of mode k, respectively.
2. **Compute Mode Probabilities:** For each value $c_{i,j}$ in C_i, calculate the probability of $c_{i,j}$ belonging to each mode. This is done by computing probability densities $\rho_k = \mu_k \mathcal{N}(c_{i,j}; \eta_k, \phi_k)$ for each mode k.
3. **Mode Sampling and Normalization:** Based on the calculated probability densities, sample a mode for normalization. For instance, if the third mode is selected, $c_{i,j}$ is represented as a one-hot vector $\beta_{i,j} = [0, 0, 1]$ indicating the chosen mode, and a scalar $\alpha_{i,j} = \frac{c_{i,j} - \eta_3}{4\phi_3}$ to represent the normalized value within that mode.

Addressing Imbalance with Conditional GAN

- Traditional GANs, fed with vectors from a standard multivariate normal distribution, do not account for imbalances in categorical columns.

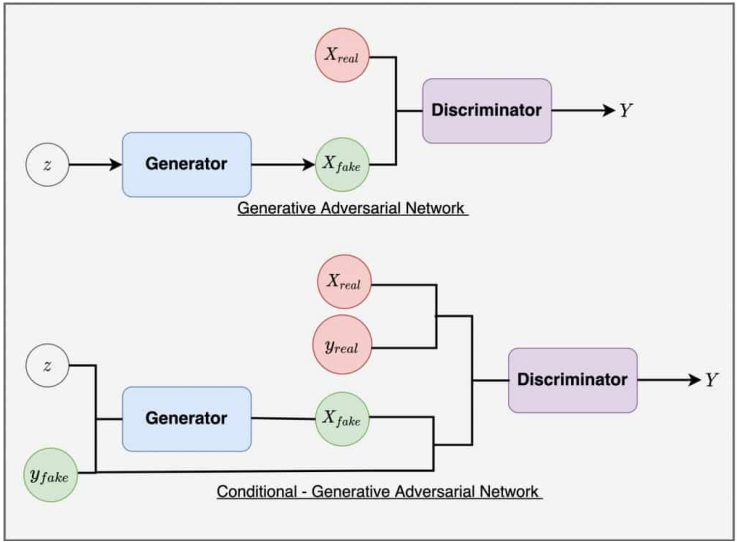
Addressing Imbalance with Conditional GAN

- Traditional GANs, fed with vectors from a standard multivariate normal distribution, do not account for imbalances in categorical columns.
- The conditional GAN aims to evenly sample all categories from discrete attributes during training, ensuring the synthetic data accurately reflects the real distribution, not a resampled one. This is akin to solving the "class imbalance" problem but in the generative modeling context.

Addressing Imbalance with Conditional GAN

- Traditional GANs, fed with vectors from a standard multivariate normal distribution, do not account for imbalances in categorical columns.
- The conditional GAN aims to evenly sample all categories from discrete attributes during training, ensuring the synthetic data accurately reflects the real distribution, not a resampled one. This is akin to solving the "class imbalance" problem but in the generative modeling context.
- The conditional generator is designed to output data conditioned on specific category values, effectively modeling $\hat{r} \sim P_G(\text{row}|D_{i*} = k^*)$. This approach enables the generation of balanced synthetic data across all categories.

Conditional GAN



Train-by-Sampling Procedure in CTGAN

Enhancing Conditional Generator Training:

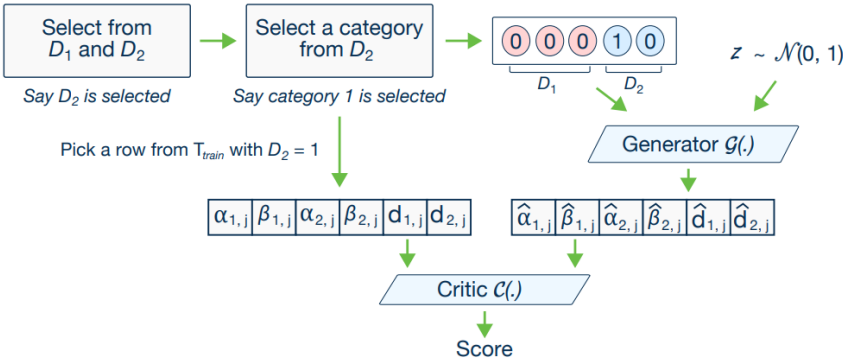
The train-by-sampling method aims to optimize the interaction between the conditional generator and the critic by focusing on the conditional distribution of the generated and real data. Here's a step-by-step summary:

1. **Mask Vector Initialization:** For each discrete column D_i, initialize a zero-filled mask vector m_i, where each element corresponds to a category in that column.
2. **Column Selection:** Randomly select one of the N_d discrete columns with equal probability. The selected column, denoted as D_{i*}, will be the focus for generating the condition vector.
3. **Probability Mass Function (PMF) Construction:** Build a PMF for the selected column D_{i*} where the probability mass for each value is proportional to the logarithm of its frequency, enhancing representation of less frequent categories.
4. **Value Selection:** Randomly select a value k^* from D_{i*} based on the constructed PMF. This value specifies the condition to be met by the generated data.
5. **Mask Update:** Set the k^*th element of the i^*th mask vector to one, indicating the selected condition.
6. **Condition Vector Construction:** Combine all mask vectors into a single condition vector *cond* through concatenation. This vector guides the conditional generation process, ensuring a balanced exploration of category values.

Generator Loss in Conditional GANs

- During training, the conditional generator produces a set of one-hot discrete vectors $\{\hat{d}_1, \ldots, \hat{d}_{N_d}\}$. Given a condition $D_{i^*} = k^*$ in the form of a cond vector, there is initially no constraint preventing the generation of $\hat{d}_{i^*}^{(k^*)} = 0$ or $\hat{d}_{i^*}^{(k)} = 1$ for $k \neq k^*$.
- To ensure the generator accurately replicates the specified condition in its output, a mechanism is introduced where the generator's loss is penalized by adding the cross-entropy between the expected condition m_{i^*} and the produced condition $\hat{d}_{i^*}$, averaged over all instances in the batch.
- This loss penalization drives the generator towards making an exact copy of the given m_{i^*} in $\hat{d}_{i^*}$, effectively learning to preserve the specified condition as the training progresses.

Put Together the CTGAN



CTGAN Generator Model Structure

Description:

- The generator begins by combining the noise vector z with the conditional information $cond$.
- It then passes through two fully-connected layers with ReLU activations and batch normalization.
- The generator produces synthetic data, with different activations for different types of output: α_i (scalar values) use tanh, while β_i (mode indicators) and d_i (discrete values) use Gumbel Softmax.

Network:

$$h_0 = z \oplus cond$$
$$h_1 = h_0 \oplus \text{ReLU}(\text{BN}(FC_{|cond|+|z|\to 256}(h_0)))$$
$$h_2 = h_1 \oplus \text{ReLU}(\text{BN}(FC_{|cond|+|z|+256\to 256}(h_1)))$$
$$\hat{\alpha}_i = \tanh(FC_{|cond|+|z|+512\to 1}(h_2))$$
$$\hat{\beta}_i = \text{gumbel}_{0.2}(FC_{|cond|+|z|+512\to m_i}(h_2))$$
$$\hat{d}_i = \text{gumbel}_{0.2}(FC_{|cond|+|z|+512\to |D_i|}(h_2))$$

CTGAN Critic Model Structure

Description:

- The critic model concatenates all input samples $r_1, \ldots, r_{10}$ with their respective conditional information.
- It features two fully-connected layers with Leaky ReLU activations and dropout to prevent overfitting.
- The final output is a single value representing the critic's assessment of the input's authenticity.

Network:

$$
\begin{aligned}
h_0 &= r_1 \oplus \ldots \oplus r_{10} \oplus cond_1 \oplus \ldots \oplus cond_{10} \\
h_1 &= \text{drop}(\text{leaky}_{0.2}(FC_{10|r|+10|cond| \to 256}(h_0))) \\
h_2 &= \text{drop}(\text{leaky}_{0.2}(FC_{256 \to 256}(h_1))) \\
C(\cdot) &= FC_{256 \to 1}(h_2)
\end{aligned}
$$

Addressing All Tabular Challenges for GAN

- Multi-mode / Non-gaussian Distributions: Mode-specific normalization.
- Mode-collapse / Class Imbalance: conditional gan + training by sampling.
- Learning from sparse vector: mask-predicting generator loss.

Autoencoder (AE): The Basics

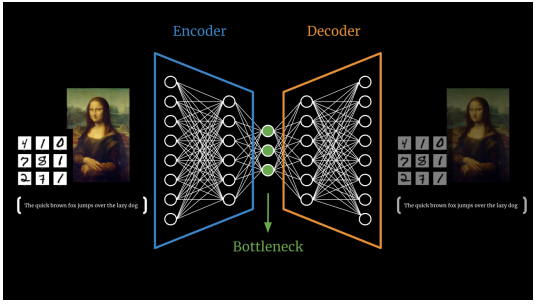


Figure: Autoencoder structure.

- **Autoencoder (AE)**:
  - An autoencoder is a neural network that learns to compress (encode) data into a lower-dimensional space and then reconstruct (decode) it back to the original form.
- **Limitation for Generative Tasks**:
  - AEs only reconstruct the input data they see during training and do not learn a structured latent space suitable for generating new samples.

Step 1: Define $p_\theta(X)$ as an Integral

- In a Variational Autoencoder (VAE), we want to maximize the probability of the observed data $p_\theta(X)$.

Step 1: Define $p_\theta(X)$ as an Integral

- In a Variational Autoencoder (VAE), we want to maximize the probability of the observed data $p_\theta(X)$.
- Let X represent the data and Z represent the latent variables.

Step 1: Define $p_\theta(X)$ as an Integral

- In a Variational Autoencoder (VAE), we want to maximize the probability of the observed data $p_\theta(X)$.
- Let X represent the data and Z represent the latent variables.
- Define $p_\theta(X)$ as an integral over the latent variable space Z:

$$p_\theta(X) = \int p(Z)\, p_\theta(X|Z)\, dZ$$

Step 1: Define $p_\theta(X)$ as an Integral

- In a Variational Autoencoder (VAE), we want to maximize the probability of the observed data $p_\theta(X)$.
- Let X represent the data and Z represent the latent variables.
- Define $p_\theta(X)$ as an integral over the latent variable space Z:

$$p_\theta(X) = \int p(Z)\, p_\theta(X|Z)\, dZ$$

- Here, $p(Z)$ is the prior distribution over latent variables, and $p_\theta(X|Z)$ is the likelihood of X given Z, parameterized by θ.

Step 2: Using Bayes' Theorem to Express $p_\theta(x)$

- Using Bayes' theorem, we can write $p_\theta(x)$ as:

$$p_\theta(x) = \frac{p_\theta(x, z)}{p_\theta(z|x)}$$

Step 2: Using Bayes' Theorem to Express $p_\theta(x)$

- Using Bayes' theorem, we can write $p_\theta(x)$ as:

$$p_\theta(x) = \frac{p_\theta(x, z)}{p_\theta(z|x)}$$

- The numerator $p_\theta(x, z) = p(Z)p_\theta(X|Z)$ is tractable because we have explicit forms for $p(Z)$ and $p_\theta(X|Z)$.

Step 2: Using Bayes' Theorem to Express $p_\theta(x)$

- Using Bayes' theorem, we can write $p_\theta(x)$ as:

$$p_\theta(x) = \frac{p_\theta(x, z)}{p_\theta(z|x)}$$

- The numerator $p_\theta(x, z) = p(Z)p_\theta(X|Z)$ is tractable because we have explicit forms for $p(Z)$ and $p_\theta(X|Z)$.
- The denominator $p_\theta(z|x)$ is intractable.

Step 2: Using Bayes' Theorem to Express $p_\theta(x)$

- Using Bayes' theorem, we can write $p_\theta(x)$ as:

$$p_\theta(x) = \frac{p_\theta(x, z)}{p_\theta(z|x)}$$

- The numerator $p_\theta(x, z) = p(Z)p_\theta(X|Z)$ is tractable because we have explicit forms for $p(Z)$ and $p_\theta(X|Z)$.
- The denominator $p_\theta(z|x)$ is intractable.
- We can instead model an approximate posterior $q_\phi(z|x_i)$.

Step 3: Deriving the Evidence Lower Bound (ELBO)

$$D_{\mathrm{KL}}(q_\phi(z|x_i)\|p(z|x_i)) = -\int q_\phi(z|x_i)\log\left(\frac{p(z|x_i)}{q_\phi(z|x_i)}\right)dz$$

$$= -\int q_\phi(z|x_i)\log\left(\frac{p_\theta(x_i|z)p(z)}{q_\phi(z|x_i)p_\theta(x_i)}\right)dz$$

$$= -\int q_\phi(z|x_i)\left(\log\frac{p_\theta(x_i|z)p(z)}{q_\phi(z|x_i)} - \log p_\theta(x_i)\right)dz$$

$$= -\int q_\phi(z|x_i)\log\frac{p_\theta(x_i|z)p(z)}{q_\phi(z|x_i)}\,dz + \int q_\phi(z|x_i)\log p_\theta(x_i)\,dz$$

$$= -\int q_\phi(z|x_i)\log\frac{p_\theta(x_i|z)p(z)}{q_\phi(z|x_i)}\,dz + \log p_\theta(x_i)\int q_\phi(z|x_i)\,dz$$

$$= -\int q_\phi(z|x_i)\log\frac{p_\theta(x_i|z)p(z)}{q_\phi(z|x_i)}\,dz + \log p_\theta(x_i)$$

$$\log p_\theta(x_i) \geq \int q_\phi(z|x_i)\left(\log p_\theta(x_i|z) + \log p(z) - \log q_\phi(z|x_i)\right)dz$$

Step 4: Interpretation of the ELBO as the VAE Loss Function

- The ELBO consists of two terms, each corresponding to a part of the VAE loss function:

Step 4: Interpretation of the ELBO as the VAE Loss Function

- The ELBO consists of two terms, each corresponding to a part of the VAE loss function:
- Reconstruction Loss $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$

Step 4: Interpretation of the ELBO as the VAE Loss Function

- The ELBO consists of two terms, each corresponding to a part of the VAE loss function:
- Reconstruction Loss $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$
  - Encourages the model to reconstruct X from the latent variable Z.

Step 4: Interpretation of the ELBO as the VAE Loss Function

- The ELBO consists of two terms, each corresponding to a part of the VAE loss function:
- Reconstruction Loss $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$
  - Encourages the model to reconstruct X from the latent variable Z.
  - Implemented as the negative log-likelihood, such as mean squared error or cross-entropy.

Step 4: Interpretation of the ELBO as the VAE Loss Function

- The ELBO consists of two terms, each corresponding to a part of the VAE loss function:
- Reconstruction Loss $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$
  - Encourages the model to reconstruct X from the latent variable Z.
  - Implemented as the negative log-likelihood, such as mean squared error or cross-entropy.
- KL Divergence Regularization $D_{\mathrm{KL}}(q_\phi(z|x)\|p(Z))$

Step 4: Interpretation of the ELBO as the VAE Loss Function

- The ELBO consists of two terms, each corresponding to a part of the VAE loss function:
- Reconstruction Loss $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$
  - Encourages the model to reconstruct X from the latent variable Z.
  - Implemented as the negative log-likelihood, such as mean squared error or cross-entropy.
- KL Divergence Regularization $D_{\mathrm{KL}}(q_\phi(z|x)\|p(Z))$
  - Penalizes the divergence between $q_\phi(z|x)$ and the prior $p(Z)$.

Step 4: Interpretation of the ELBO as the VAE Loss Function

- The ELBO consists of two terms, each corresponding to a part of the VAE loss function:
- Reconstruction Loss $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$
  - Encourages the model to reconstruct X from the latent variable Z.
  - Implemented as the negative log-likelihood, such as mean squared error or cross-entropy.
- KL Divergence Regularization $D_{\mathrm{KL}}(q_\phi(z|x)\|p(Z))$
  - Penalizes the divergence between $q_\phi(z|x)$ and the prior $p(Z)$.
  - Helps structure the latent space and prevents overfitting.

VAE Framework

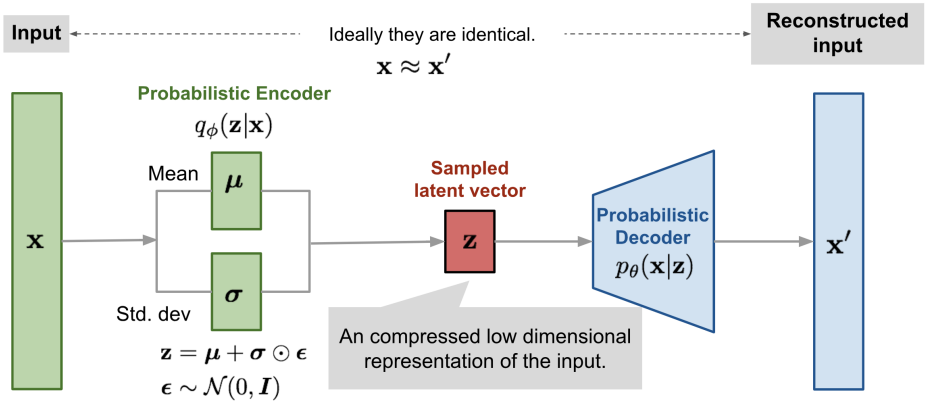


Figure: VAE Framework

How VAE Ensures a Structured Latent Space

- **Latent Space Regularization**:
  - The KL Divergence term enforces the latent space to follow a normal distribution.
  - Encourages each input to be encoded as a distribution with similar properties.
- **Sampling from Latent Space**:
  - By sampling from the normal distribution in latent space, we can generate realistic, diverse data points after decoding.
- **Benefits of Structured Latent Space**:
  - Ensures smooth transitions and meaningful variations in generated data, as the latent space is continuous and well-organized.

Conditional Variational Autoencoder (CVAE)

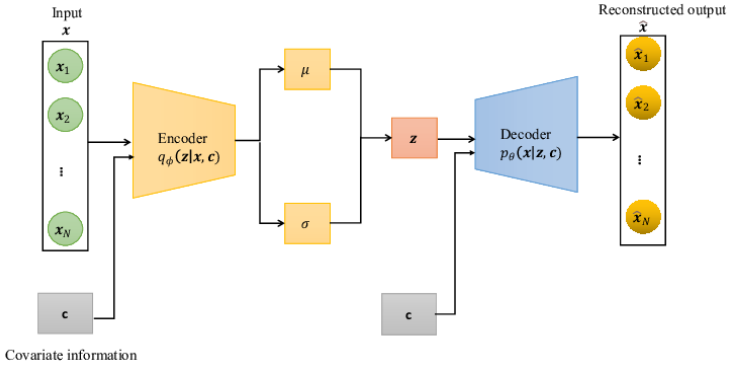
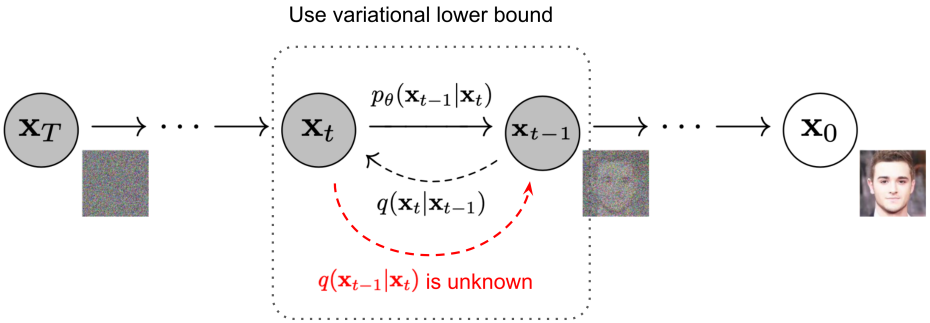


Figure: Structure of conditional VAE

Tabular Variational Autoencoder (TVAE)

- **Overview of TVAE**:
  - TVAE adapts the Variational Autoencoder (VAE) framework for tabular data.
  - Uses separate neural networks to model $p_\theta(r_j|z_j)$ (decoder) and $q_\phi(z_j|r_j)$ (encoder).
- **TVAE Loss Function**:
  - Trained using Evidence Lower Bound (ELBO) loss, which balances reconstruction accuracy and regularization.
  - Ensures that the latent space is regularized to follow a normal distribution for easy sampling.
- **Structured Latent Space**:
  - By regularizing the latent vector, TVAE maintains a structured distribution, making it suitable for generating diverse synthetic data.
  - Continuous features follow a Gaussian distribution; discrete features are generated with a categorical distribution.

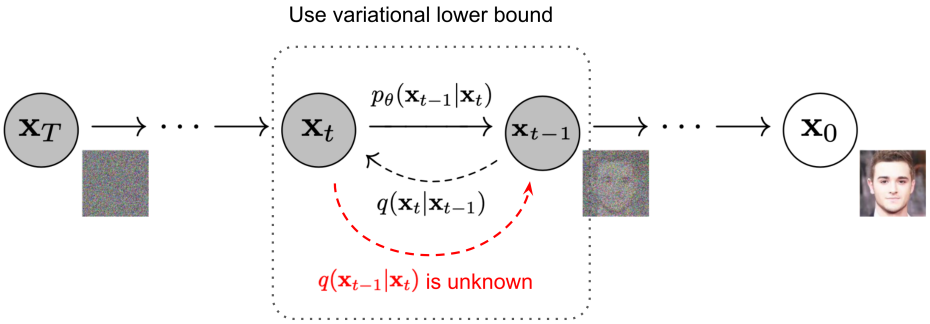
Denoising Diffusion Probabilistic Model (DDPM)



The Denoising Diffusion Probabilistic Model (DDPM) [HJA20] comprised of two main processes:

- **Forward Process (Noise Adding):** Noise is incrementally added to the original data over a series of steps, transforming the data into a nearly unrecognizable noisy state.

Denoising Diffusion Probabilistic Model (DDPM)



The Denoising Diffusion Probabilistic Model (DDPM) [HJA20] comprised of two main processes:

- **Forward Process (Noise Adding):**Noise is incrementally added to the original data over a series of steps, transforming the data into a nearly unrecognizable noisy state.
- **Backward Process (Noise Removal):** The reverse of the forward process, reconstruct the original data from its noisy representation.

Gaussian Diffusion Models

Gaussian diffusion models are a class of generative models that operate in continuous spaces and leverage Gaussian distributions for both forward and reverse diffusion processes. Key characteristics and formulas include:

- **Forward Process:** The addition of noise to data samples is modeled as a Gaussian process with:

$$q(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t \mathbf{I}\right)$$

Gaussian Diffusion Models

Gaussian diffusion models are a class of generative models that operate in continuous spaces and leverage Gaussian distributions for both forward and reverse diffusion processes. Key characteristics and formulas include:

- **Forward Process:** The addition of noise to data samples is modeled as a Gaussian process with:

$$q(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t \mathbf{I}\right)$$

- **Reverse Process:** The model for reversing noise addition, approximated by a neural network with parameters θ, is given by:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)\right)$$

Gaussian Diffusion Models

Gaussian diffusion models are a class of generative models that operate in continuous spaces and leverage Gaussian distributions for both forward and reverse diffusion processes. Key characteristics and formulas include:

- **Forward Process:** The addition of noise to data samples is modeled as a Gaussian process with:

$$q(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t \mathbf{I}\right)$$

- **Reverse Process:** The model for reversing noise addition, approximated by a neural network with parameters θ, is given by:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)\right)$$

- **Parameterization:** Ho et al. (2020) proved that the training objective can be simplified to mean-squared error between predicted and actual noise:

$$L^t_{\text{simple}} = \mathbb{E}_{x_0,\epsilon,t}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|_2^2\right]$$

Multinomial Diffusion Models

Multinomial diffusion models are designed for generating categorical data and operate using a different set of principles tailored to discrete spaces:

- **Forward Process:** The process corrupts data samples with uniform noise over K classes, modeled as a categorical distribution:

$$q(x_t|x_{t-1}) = \text{Cat}\left(x_t; (1-\beta_t)x_{t-1} + \frac{\beta_t}{K}\right)$$

Multinomial Diffusion Models

Multinomial diffusion models are designed for generating categorical data and operate using a different set of principles tailored to discrete spaces:

- **Forward Process:** The process corrupts data samples with uniform noise over K classes, modeled as a categorical distribution:

$$q(x_t|x_{t-1}) = \text{Cat}\left(x_t; (1-\beta_t)x_{t-1} + \frac{\beta_t}{K}\right)$$

- **Reverse Process:** Defines the reverse diffusion from the noised data back to the original data as:

$$q(x_t|x_0) = \text{Cat}\left(x_t; \bar{\alpha}_t x_0 + \frac{1-\bar{\alpha}_t}{K}\right)$$

Multinomial Diffusion Models

Multinomial diffusion models are designed for generating categorical data and operate using a different set of principles tailored to discrete spaces:

- **Forward Process:** The process corrupts data samples with uniform noise over K classes, modeled as a categorical distribution:

$$q(x_t|x_{t-1}) = \text{Cat}\left(x_t; (1-\beta_t)x_{t-1} + \frac{\beta_t}{K}\right)$$

- **Reverse Process:** Defines the reverse diffusion from the noised data back to the original data as:

$$q(x_t|x_0) = \text{Cat}\left(x_t; \bar{\alpha}_t x_0 + \frac{1-\bar{\alpha}_t}{K}\right)$$

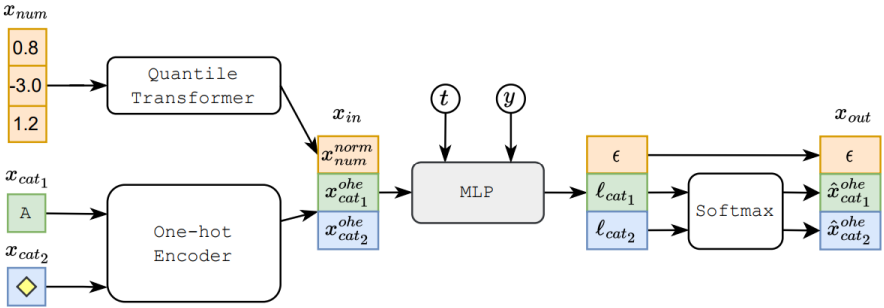
- **Posterior Derivation:** The posterior distribution, which is critical for the reverse process, is given by:

$$q(x_{t-1}|x_t, x_0) = \text{Cat}\left(x_{t-1}; \frac{\pi}{\sum_{k=1}^{K} \pi_k}\right)$$

where

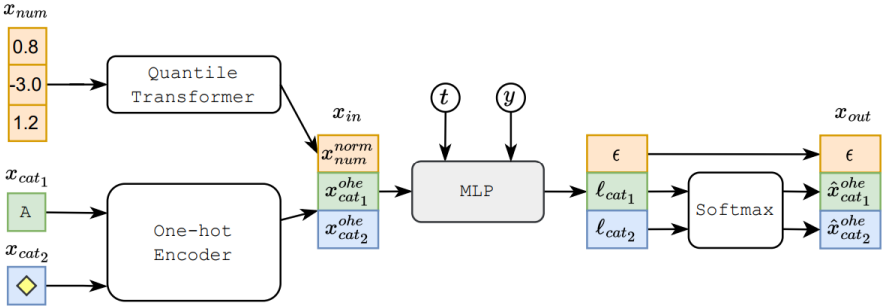
$$\pi = [\alpha_t x_t + (1-\alpha_t)/K] \odot [\bar{\alpha}_{t-1} x_0 + (1-\bar{\alpha}_{t-1})/K].$$

TabDDPM



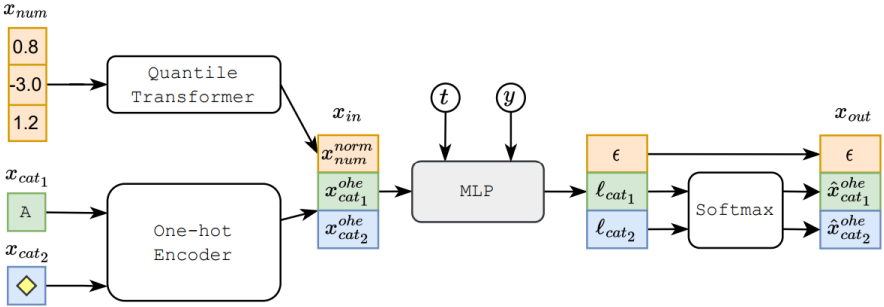
- The tabDDPM [Kot+23] model preprocesses input data by converting categorical features into one-hot encoded vectors ($x_{\text{ohe\_cat}_i} \in \{0,1\}^{K_i}$) and normalizing numerical features.

TabDDPM



- The tabDDPM [Kot+23] model preprocesses input data by converting categorical features into one-hot encoded vectors ($x_{\text{ohe\_cat}_i} \in \{0,1\}^{K_i}$) and normalizing numerical features.
- For the preprocessing of numerical features, tabDDPM employs the Gaussian quantile transformation. This ensures that the transformed values is close to a normal distribution.

TabDDPM



- The tabDDPM [Kot+23] model preprocesses input data by converting categorical features into one-hot encoded vectors ($x_{\text{ohe\_cat}_i} \in \{0,1\}^{K_i}$) and normalizing numerical features.
- For the preprocessing of numerical features, tabDDPM employs the Gaussian quantile transformation. This ensures that the transformed values is close to a normal distribution.
- Categorical feature undergoes a distinct forward diffusion process with independently sampled noise components.

Loss Function and Model Structure in TabDDPM

- **Model Structure:** The reverse diffusion step is modeled by a Multi-Layer Perceptron neural network, with the output dimension matching that of the input x_0.

Loss Function and Model Structure in TabDDPM

- **Model Structure:** The reverse diffusion step is modeled by a Multi-Layer Perceptron neural network, with the output dimension matching that of the input x_0.
- **Loss Function:** The model is trained by minimizing a combination of mean-squared error L^t_{simple} for the Gaussian diffusion component, and the KL divergences L^t_i for each multinomial diffusion term. The total loss:

$$L^t_{\text{TabDDPM}} = L^t_{\text{simple}} + \frac{1}{C} \sum_{i \leq C} L^t_i$$

Loss Function and Model Structure in TabDDPM

- **Model Structure:** The reverse diffusion step is modeled by a Multi-Layer Perceptron neural network, with the output dimension matching that of the input x_0.
- **Loss Function:** The model is trained by minimizing a combination of mean-squared error L^t_{simple} for the Gaussian diffusion component, and the KL divergences L^t_i for each multinomial diffusion term. The total loss:

$$L^t_{\text{TabDDPM}} = L^t_{\text{simple}} + \frac{1}{C}\sum_{i \le C} L^t_i$$

- **Class-Conditional Model:** For classification problems, TabDDPM employs a class-conditional model, learning $p_\theta(x_{t-1}|x_t, y)$, where y represents class labels. In regression scenarios, the target value is treated as an additional numerical feature.

Comparing GANs, Diffusion Models, and VAEs

Generative Adversarial Networks (GANs), diffusion models, and Variational Autoencoders (VAEs) represent three powerful approaches to generative modeling, each with unique strengths and challenges. Here's a comparison, focusing on stability, performance, and latent space structure:

- **Training Stability:**

Comparing GANs, Diffusion Models, and VAEs

Generative Adversarial Networks (GANs), diffusion models, and Variational Autoencoders (VAEs) represent three powerful approaches to generative modeling, each with unique strengths and challenges. Here's a comparison, focusing on stability, performance, and latent space structure:

- **Training Stability:**
  - **GANs:** Training involves a min-max optimization between the generator and discriminator, which can lead to instability, mode collapse, and convergence issues.

Comparing GANs, Diffusion Models, and VAEs

Generative Adversarial Networks (GANs), diffusion models, and Variational Autoencoders (VAEs) represent three powerful approaches to generative modeling, each with unique strengths and challenges. Here's a comparison, focusing on stability, performance, and latent space structure:

- **Training Stability:**
  - **GANs:** Training involves a min-max optimization between the generator and discriminator, which can lead to instability, mode collapse, and convergence issues.
  - **VAEs:** Use variational inference to approximate the posterior, resulting in a stable training process, though reconstructions may be slightly blurry due to the probabilistic nature of latent sampling.

Comparing GANs, Diffusion Models, and VAEs

Generative Adversarial Networks (GANs), diffusion models, and Variational Autoencoders (VAEs) represent three powerful approaches to generative modeling, each with unique strengths and challenges. Here's a comparison, focusing on stability, performance, and latent space structure:

- **Training Stability:**
  - **GANs:** Training involves a min-max optimization between the generator and discriminator, which can lead to instability, mode collapse, and convergence issues.
  - **VAEs:** Use variational inference to approximate the posterior, resulting in a stable training process, though reconstructions may be slightly blurry due to the probabilistic nature of latent sampling.
  - **Diffusion Models:** Employ a more straightforward optimization with a well-defined objective function, leading to stable training dynamics and avoiding adversarial interactions.

Comparing GANs, Diffusion Models, and VAEs

Generative Adversarial Networks (GANs), diffusion models, and Variational Autoencoders (VAEs) represent three powerful approaches to generative modeling, each with unique strengths and challenges. Here's a comparison, focusing on stability, performance, and latent space structure:

- **Training Stability:**
  - **GANs:** Training involves a min-max optimization between the generator and discriminator, which can lead to instability, mode collapse, and convergence issues.
  - **VAEs:** Use variational inference to approximate the posterior, resulting in a stable training process, though reconstructions may be slightly blurry due to the probabilistic nature of latent sampling.
  - **Diffusion Models:** Employ a more straightforward optimization with a well-defined objective function, leading to stable training dynamics and avoiding adversarial interactions.
- **Sample Quality and Diversity:**

Comparing GANs, Diffusion Models, and VAEs

Generative Adversarial Networks (GANs), diffusion models, and Variational Autoencoders (VAEs) represent three powerful approaches to generative modeling, each with unique strengths and challenges. Here's a comparison, focusing on stability, performance, and latent space structure:

- **Training Stability:**
  - **GANs:** Training involves a min-max optimization between the generator and discriminator, which can lead to instability, mode collapse, and convergence issues.
  - **VAEs:** Use variational inference to approximate the posterior, resulting in a stable training process, though reconstructions may be slightly blurry due to the probabilistic nature of latent sampling.
  - **Diffusion Models:** Employ a more straightforward optimization with a well-defined objective function, leading to stable training dynamics and avoiding adversarial interactions.
- **Sample Quality and Diversity:**
  - **GANs:** Can generate high-quality, sharp samples but may suffer from mode collapse, reducing diversity.

Comparing GANs, Diffusion Models, and VAEs

Generative Adversarial Networks (GANs), diffusion models, and Variational Autoencoders (VAEs) represent three powerful approaches to generative modeling, each with unique strengths and challenges. Here's a comparison, focusing on stability, performance, and latent space structure:

- **Training Stability:**
  - **GANs:** Training involves a min-max optimization between the generator and discriminator, which can lead to instability, mode collapse, and convergence issues.
  - **VAEs:** Use variational inference to approximate the posterior, resulting in a stable training process, though reconstructions may be slightly blurry due to the probabilistic nature of latent sampling.
  - **Diffusion Models:** Employ a more straightforward optimization with a well-defined objective function, leading to stable training dynamics and avoiding adversarial interactions.
- **Sample Quality and Diversity:**
  - **GANs:** Can generate high-quality, sharp samples but may suffer from mode collapse, reducing diversity.
  - **VAEs:** Often produce diverse but slightly less sharp samples compared to GANs, as the reconstruction quality can be affected by the Gaussian assumption in the latent space.

Comparing GANs, Diffusion Models, and VAEs

Generative Adversarial Networks (GANs), diffusion models, and Variational Autoencoders (VAEs) represent three powerful approaches to generative modeling, each with unique strengths and challenges. Here's a comparison, focusing on stability, performance, and latent space structure:

- **Training Stability:**
  - **GANs:** Training involves a min-max optimization between the generator and discriminator, which can lead to instability, mode collapse, and convergence issues.
  - **VAEs:** Use variational inference to approximate the posterior, resulting in a stable training process, though reconstructions may be slightly blurry due to the probabilistic nature of latent sampling.
  - **Diffusion Models:** Employ a more straightforward optimization with a well-defined objective function, leading to stable training dynamics and avoiding adversarial interactions.
- **Sample Quality and Diversity:**
  - **GANs:** Can generate high-quality, sharp samples but may suffer from mode collapse, reducing diversity.
  - **VAEs:** Often produce diverse but slightly less sharp samples compared to GANs, as the reconstruction quality can be affected by the Gaussian assumption in the latent space.
  - **Diffusion Models:** Tend to produce high-quality, diverse samples, as they directly model the data distribution over multiple noise scales.

Comparing GANs, Diffusion Models, and VAEs

Generative Adversarial Networks (GANs), diffusion models, and Variational Autoencoders (VAEs) represent three powerful approaches to generative modeling, each with unique strengths and challenges. Here's a comparison, focusing on stability, performance, and latent space structure:

- **Training Stability:**
  - **GANs:** Training involves a min-max optimization between the generator and discriminator, which can lead to instability, mode collapse, and convergence issues.
  - **VAEs:** Use variational inference to approximate the posterior, resulting in a stable training process, though reconstructions may be slightly blurry due to the probabilistic nature of latent sampling.
  - **Diffusion Models:** Employ a more straightforward optimization with a well-defined objective function, leading to stable training dynamics and avoiding adversarial interactions.
- **Sample Quality and Diversity:**
  - **GANs:** Can generate high-quality, sharp samples but may suffer from mode collapse, reducing diversity.
  - **VAEs:** Often produce diverse but slightly less sharp samples compared to GANs, as the reconstruction quality can be affected by the Gaussian assumption in the latent space.
  - **Diffusion Models:** Tend to produce high-quality, diverse samples, as they directly model the data distribution over multiple noise scales.
- **Latent Space Structure:**

Comparing GANs, Diffusion Models, and VAEs

Generative Adversarial Networks (GANs), diffusion models, and Variational Autoencoders (VAEs) represent three powerful approaches to generative modeling, each with unique strengths and challenges. Here's a comparison, focusing on stability, performance, and latent space structure:

- **Training Stability:**
  - **GANs:** Training involves a min-max optimization between the generator and discriminator, which can lead to instability, mode collapse, and convergence issues.
  - **VAEs:** Use variational inference to approximate the posterior, resulting in a stable training process, though reconstructions may be slightly blurry due to the probabilistic nature of latent sampling.
  - **Diffusion Models:** Employ a more straightforward optimization with a well-defined objective function, leading to stable training dynamics and avoiding adversarial interactions.
- **Sample Quality and Diversity:**
  - **GANs:** Can generate high-quality, sharp samples but may suffer from mode collapse, reducing diversity.
  - **VAEs:** Often produce diverse but slightly less sharp samples compared to GANs, as the reconstruction quality can be affected by the Gaussian assumption in the latent space.
  - **Diffusion Models:** Tend to produce high-quality, diverse samples, as they directly model the data distribution over multiple noise scales.
- **Latent Space Structure:**
  - **GANs:** Do not explicitly model a structured latent space, making interpolation and representation learning challenging.

Comparing GANs, Diffusion Models, and VAEs

Generative Adversarial Networks (GANs), diffusion models, and Variational Autoencoders (VAEs) represent three powerful approaches to generative modeling, each with unique strengths and challenges. Here's a comparison, focusing on stability, performance, and latent space structure:

- **Training Stability:**
  - **GANs:** Training involves a min-max optimization between the generator and discriminator, which can lead to instability, mode collapse, and convergence issues.
  - **VAEs:** Use variational inference to approximate the posterior, resulting in a stable training process, though reconstructions may be slightly blurry due to the probabilistic nature of latent sampling.
  - **Diffusion Models:** Employ a more straightforward optimization with a well-defined objective function, leading to stable training dynamics and avoiding adversarial interactions.
- **Sample Quality and Diversity:**
  - **GANs:** Can generate high-quality, sharp samples but may suffer from mode collapse, reducing diversity.
  - **VAEs:** Often produce diverse but slightly less sharp samples compared to GANs, as the reconstruction quality can be affected by the Gaussian assumption in the latent space.
  - **Diffusion Models:** Tend to produce high-quality, diverse samples, as they directly model the data distribution over multiple noise scales.
- **Latent Space Structure:**
  - **GANs:** Do not explicitly model a structured latent space, making interpolation and representation learning challenging.
  - **VAEs:** Learn a well-organized, interpretable latent space, allowing for smooth interpolation and meaningful representation learning.

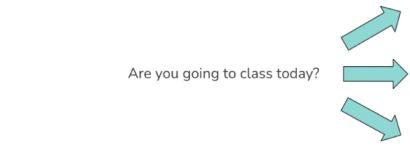
Comparing GANs, Diffusion Models, and VAEs

Generative Adversarial Networks (GANs), diffusion models, and Variational Autoencoders (VAEs) represent three powerful approaches to generative modeling, each with unique strengths and challenges. Here's a comparison, focusing on stability, performance, and latent space structure:

- **Training Stability:**
  - **GANs:** Training involves a min-max optimization between the generator and discriminator, which can lead to instability, mode collapse, and convergence issues.
  - **VAEs:** Use variational inference to approximate the posterior, resulting in a stable training process, though reconstructions may be slightly blurry due to the probabilistic nature of latent sampling.
  - **Diffusion Models:** Employ a more straightforward optimization with a well-defined objective function, leading to stable training dynamics and avoiding adversarial interactions.
- **Sample Quality and Diversity:**
  - **GANs:** Can generate high-quality, sharp samples but may suffer from mode collapse, reducing diversity.
  - **VAEs:** Often produce diverse but slightly less sharp samples compared to GANs, as the reconstruction quality can be affected by the Gaussian assumption in the latent space.
  - **Diffusion Models:** Tend to produce high-quality, diverse samples, as they directly model the data distribution over multiple noise scales.
- **Latent Space Structure:**
  - **GANs:** Do not explicitly model a structured latent space, making interpolation and representation learning challenging.
  - **VAEs:** Learn a well-organized, interpretable latent space, allowing for smooth interpolation and meaningful representation learning.
  - **Diffusion Models:** Do not rely on a latent space; instead, they generate samples by progressively

Language As Sequences

Text As Autoregressive Sequences.



Text Token

$$p(\boldsymbol{t}) = p(w_1, \ldots, w_j) = \prod_{k=1}^{j} p(w_k | w_1, ..., w_{k-1}).$$

Probability of tokens in natural language sequence depend on both the input, and preceding tokens already generated in the output.

Tokenization and Training

What is a Tokenizer?

- A tokenizer breaks down text into manageable pieces (tokens).
- Tokens can be words, subwords, or characters.
- **Example:** "cat" might remain "cat", but "cats" could be broken into "cat" + "s".

Role in NLP

- Tokenizers are the first step in text processing.
- They convert raw text into a format that models can understand.

Training Tokenizers

- Typically trained on a large corpus of text.
- Learning common patterns and subword frequencies.

GPT Tokenizer Example

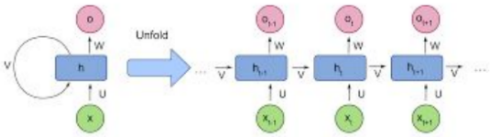
- Uses a byte pair encoding (BPE) algorithm.
- Starts with basic characters, then merges the most frequent pairs.
- Continues until a specified vocabulary size is reached.

Pre-LLM Modeling

Sequence data is very common in real life. Examples of it includes sentences, audios, time series, etc.

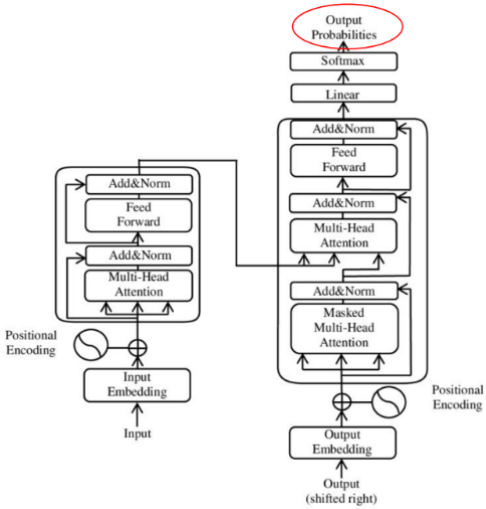
Recursive Neural Network was developed to handle such sequential and autoregressive datasets.

The recurrent nature of it often leads to forgetting of earlier information in the sequence.

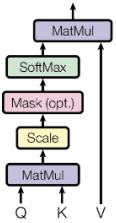


Transformer

- Vaswani, et.al introduced the transformer architecture, which utilizes attention mechanism to learn seq2seq tasks.
- The figure on the right from the paper shows the encoder-decoder architecture. GPT is a decoder-only model which only has the right hand side. It takes a sequence of tokens (as indices in vocabulary), and predicts the right shifted version of input.
- Transformer outperforms RNNs because it is able to attend to the entire sequence.



Attention



Look-ahead masking

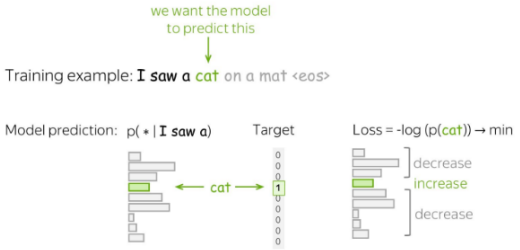
$$\begin{bmatrix} \text{Softmax}\left(\frac{Q_1K_1^T}{\sqrt{d_K}}\right) & 0 & \cdots & 0 \\ \text{Softmax}\left(\frac{Q_2K_1^T}{\sqrt{d_K}}\right) & \text{Softmax}\left(\frac{Q_2K_2^T}{\sqrt{d_K}}\right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \text{Softmax}\left(\frac{Q_tK_1^T}{\sqrt{d_K}}\right) & \text{Softmax}\left(\frac{Q_tK_2^T}{\sqrt{d_K}}\right) & \cdots & \text{Softmax}\left(\frac{Q_tK_t^T}{\sqrt{d_K}}\right) \end{bmatrix} \times \begin{bmatrix} V_{1,1} & V_{1,2} & \cdots & V_{1,d_k} \\ V_{2,1} & V_{2,2} & \cdots & V_{2,d_k} \\ \vdots & \vdots & \cdots & \vdots \\ V_{t,1} & V_{t,2} & \cdots & V_{t,d_k} \end{bmatrix}$$

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

- The matrix Query, Key and Values are acquired by learnable linear layers.
- The resulting attention matrix represents how much attention should be paid to every other token in the sequence when creating the output representation for a token.
- For GPT setting, upper triangle masking ensures that each token only pays attention to its preceding tokens. LLMs trained on other tasks might have other masking choices.

What LLM Outputs Are Random?

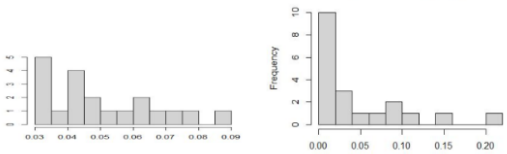
- Using transformers architectures,LLMs themselves are deterministics.
- But they can learn the distribution of tokens.
- By sampling on these distributions, we get probabilistic outputs (text).
- By conditioning on all previous input and output tokens via attention mechanism, LLMs also learn the correlation between tokens.



How Do We Control LLM Randomness?

- The temperature parameter in the softmax function controls how "spread out" the output probabilities are. With higher temperature, we are more likely to get rare outputs.
- The idea comes from thermodynamics, where low energy states are more likely to occur when temperature is high.
- The common choice when we want a diverse output is T = 0.7.

$$P_i = \frac{e^{\frac{y_i}{T}}}{\sum_{k=1}^{n} e^{\frac{y_k}{T}}}$$



How Can We Sample?

- Randomly sampling: use the predicted probabilities as the probability for inclusion.
- Top-K sampling: only sample from the vocabulary with top-K largest probabilities.
- Nucleus Sampling: only sample from the top vocabularies, whose probabilities add up to a hyperparameter p. Applied by OpenAI.

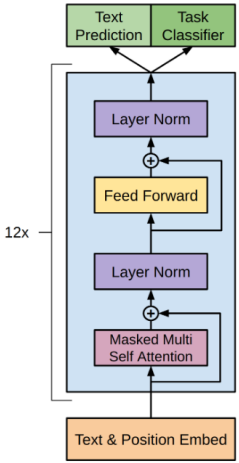
Sampling strategies can affect the generation of imbalance variables. For example, rare values can be consistently ignored by top-K/nucleus sampling.

Autoregressive Model

- Autoregressive Language Learning Models (LLMs) generate text by predicting one word at a time.
- Each new word is predicted based on the sequence of words that came before it.
- This type of model is *autoregressive* in that the prediction of the next word is a regression on previous words.
- **Example:** The GPT (Generative Pre-trained Transformer) series uses this approach to produce coherent and contextually relevant text by conditioning each new word prediction on all the previous words in a sentence.

Autoregressive Model

- **Decoder-Only Structure:**
  - A "decoder-only" architecture, like that used in GPT models, consists exclusively of decoder blocks from the traditional Transformer architecture.
  - This structure is well-suited for tasks that involve generating text from a given prompt because it excels in handling a sequence of inputs where the entire sequence is available from the start.
  - The decoder processes input by looking at all previous positions in the sequence, making it powerful for generating coherent and context-aware text.
  - **Efficiency and Scalability:** This streamlined architecture allows for more efficient training and scalability to large datasets, a hallmark of the GPT series.

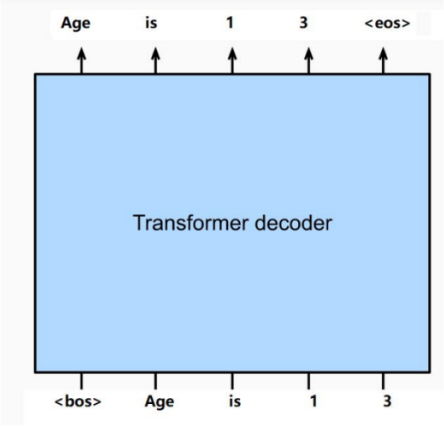


Autoregressive Prediction

In the case of GPT2, the model is trained to predict the next token based on previous tokens.

For example, the input "<sos> Age is 13, Gender is female. <eos>" will have target output "Age is 13, Gender is female <eos> <eot> ", where <eot> is the special end of text token.

During training, prediction for all tokens in the target sequence is done all at once.



Masked-Language Model

Masked Language Modeling (MLM):

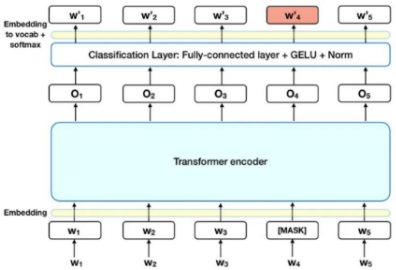
- MLM is a task where certain words in a text are masked (i.e., hidden), and the objective is to predict these masked words based on the context provided by the non-masked words in the text.
- This approach helps the model to focus on understanding bidirectional contexts, thereby gaining a deeper understanding of language structure and usage.
- **Example:** In BERT, random words are replaced with a special token (e.g., "[MASK]") during training, and the model learns to predict the original word based solely on its context.

Masked-Language Model

- **BERT as an Example:**
  - BERT (Bidirectional Encoder Representations from Transformers) utilizes the MLM task to train its bidirectional capabilities.
  - Unlike traditional autoregressive models that predict words in a unidirectional manner, BERT examines the context from both directions (left-to-right and right-to-left).
  - This bidirectionality allows for a more nuanced understanding of language, making BERT effective for a variety of NLP tasks beyond text generation, such as sentiment analysis and question answering.

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com



AR vs MLM

Comparison with Autoregressive Models:

- **Pros of MLM and BERT:**
  - *Contextual Awareness:* MLMs like BERT can understand and utilize context from both directions, providing a richer understanding of language.
  - *Versatility:* BERT's training on the MLM task makes it versatile for different types of NLP tasks that require understanding of context and relationships between words.
- **Cons of MLM and BERT:**
  - *Computational Intensity:* BERT models are generally more resource-intensive due to their deep bidirectional nature.
  - *Less Efficient for Generation:* MLMs are not as efficient as autoregressive models for tasks like text generation where the flow and continuity of text are crucial.
- **Pros of Autoregressive Models:**
  - *Efficiency in Text Generation:* Autoregressive models are more streamlined for generating coherent and contextually appropriate text one word at a time.
  - *Speed:* Typically, these models are faster to train on tasks that do not require deep bidirectional context understanding.
- **Cons of Autoregressive Models:**
  - *Limited Context:* The unidirectional nature limits the understanding of full context, potentially reducing the richness of language comprehension compared to MLMs.

Semantic Search

1. **Use Cases**
   - Information Retrieval: Enhancing the accuracy of search engines by understanding query context.
   - Data Management: Organizing large datasets by semantic similarity rather than keywords or tags.
   - Customer Support: Improving response quality in chatbots and virtual assistants by understanding user intent.
2. **Common Types and Methods**
   - Vector Space Models: Using TF-IDF, word embeddings (Word2Vec, GloVe) to convert text into vectors.
   - Graph-Based Models: Leveraging knowledge graphs for semantic understanding and link prediction.
   - Neural Approaches: Employing deep learning, BERT, and transformer models to capture deeper semantic meanings.
3. **Limitations and Challenges**
   - Complexity: High computational resources needed for processing and model training.
   - Ambiguity: Difficulty in handling words with multiple meanings based on context.
   - Data Bias: Models may inherit biases present in training data, affecting impartiality and performance.

Sentence Transformers

Purpose of Sentence Embeddings

- Sentence embeddings are used to convert text into high-dimensional vectors.
- These vectors capture not only the words used but also the semantic meaning of the sentences.
- Enables tasks such as semantic search, where search results are based on meaning rather than just keyword matches.

How are Sentence Embeddings Generated?

- Training Data: Large datasets of text pairs are used, often annotated for similar meaning.
- Model Architecture: Typically involve Transformer-based models like BERT or RoBERTa.
- Siamese and Triplet Networks: These networks compare multiple sentences at once to understand semantic similarities and differences.
- Fine-Tuning: Models are fine-tuned on specific tasks to optimize performance for particular types of semantic understanding.

Benefits of Sentence Embeddings

- Improved Accuracy: More accurate search results by understanding context and nuances in language.
- Language Independence: Effective across different languages without direct translation.
- Scalability: Once trained, embeddings can be used to process large volumes of text efficiently.

Retrieval-Augmented Generation (RAG)

What is RAG?

- Retrieval Step: Uses semantic search techniques to find documents relevant to a query.
  - Sentence embeddings from models like BERT or Sentence Transformers are used to understand and match query semantics.
- Generation Step: Utilizes a Transformer-based model to synthesize information from retrieved texts into coherent answers.

How Does RAG Improve Language Models?

1. **Contextual Relevance**: By integrating retrieval into generation, RAG models can access a broader context, improving the relevance and accuracy of responses.
2. **Factuality**: Enhances the factuality of generated content by grounding responses in retrieved documents.
3. **Diversity**: Produces more diverse responses by drawing from a wide range of sources.

Boosting Robustness in LLMs through RAG

Contributions to Robustness and Reliability

- Data-Driven Contextualization: By incorporating external data during the generation phase, RAG models produce responses that are not only relevant but also contextually rich.
- Error Reduction: Leveraging retrieved information reduces the likelihood of generating nonsensical or factually incorrect statements.
- Consistency and Coherence: Responses are more consistent and coherent as they are based on curated content from reliable sources.

Mechanism of Action

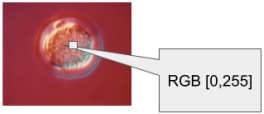
1. **Retrieval Step**: Uses semantic search to identify and fetch documents relevant to the input query, employing techniques like sentence embeddings for accurate matches.
2. **Generation Step**: Processes the input and the retrieved documents using a Transformer-based model to generate informed and precise responses.

Impact on Reliability

- Factual Accuracy: Grounding responses in actual documents enhances factual accuracy, crucial for applications like medical advice, legal assistance, and educational tools. RAG reduces the chance of hallucination.
- Adaptive Learning: RAG models can adapt over time by learning from new and diverse datasets, thereby continuously improving their accuracy and reliability.
- Reduction in Biases: By using a wide array of sources for retrieval, RAG can mitigate model biases

Challenge: The Heterogeneity of Tabular Data

Pixels are homogenous:



Tables are heterogeneous:

Age	Country	Weight
35	U.S	65.3
27	U.K	48.1

Ordinal Categorical Continuous

- Vision and Language Models operate in a homogeneous data space, utilizing normalized pixels or fixed vocabularies of tokens.
- In contrast, **tabular data are inherently heterogeneous**, combining:
  - Numerical, ordinal, categorical, and boolean data types.
  - This diversity often necessitates data-specific structures to effectively manage features.
- Structured and heterogeneous tables requires a more flexible structures.

Challenge: The Heterogeneity of Tabular Data

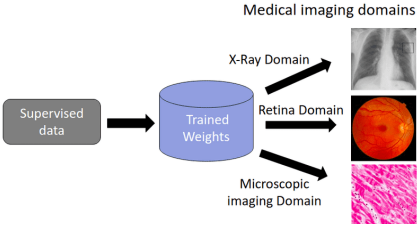
Challenges for Deep Learning

- Varied Data Types: Integrating diverse data types (e.g., binary, continuous, ordinal) into a uniform model is non-trivial.
- Scaling and Normalization: Different scales or distributions of data require careful normalization to avoid skewing model performance.
- Feature Engineering: Unlike CV/NLP where features can often be learned automatically, structured data often requires manual feature selection and engineering.

Implications for Model Development

- Developing models for heterogeneous structured data often requires more domain knowledge and preprocessing effort.
- These challenges necessitate advanced techniques in data preprocessing, feature engineering, and sometimes, specialized model architectures.

Challenge: Pre-training on Tables Needs a Transferable Representation



?	?	?
0	1	1
1	1	1
1	0	1
0	0	0
0	1	0

Apple / Melon / Peach ?

Apple / Google / Meta?

- Images and languages have **transferable local structures** or patterns.
- Conversely, tabular data encoded as numbers have **no context/semantic meaning**.
- Identically formed representations may come from vastly different contexts, and vice versa.

Feature Transferability in CV/NLP vs. Tabular Data

Challenges with Tabular Data

- Tabular data encompasses a wide variety of data types and structures (continuous, categorical, different schemas).
- Encoder Limitations: Current methods struggle to encode this heterogeneity uniformly; some resemble text conversion models that miss deeper structural nuances of tables.
- Permutation Invariance: Tabular data is unique in that rows and columns can be permuted without changing the underlying information, a property not addressed well by existing models.

Implications for Transfer Learning

1. Transfer learning involves applying knowledge gained from one domain to another; in CV and NLP, features are robust and broadly applicable.
2. Due to its diverse and schema-dependent nature, tabular data does not lend itself easily to feature transfer across different datasets or tasks.
3. This requires more customized or adaptable encoding strategies to effectively use tabular data in transfer learning scenarios.

Reconstruction Challenges in Tabular Pre-Training

Complexity of Reconstruction Targets

- Unlike text tokens, tabular data features can range widely - from tightly bounded age categories to wide-ranging lab results.
- This heterogeneity requires adaptive scaling and normalization strategies to model effectively.

Non-Aligned Categorization Issues

- Clinical diagnoses and other categorical data often lack a universal schema - complicating the learning of useful embeddings.
- Strategies such as embedding alignment or schema standardization are potential solutions but remain challenging to implement effectively.

Implications for Model Training

- Models must handle a variety of data types and distributions, necessitating complex and flexible neural network architectures.
- Accurate and efficient data imputation techniques become crucial when dealing with missing or incomplete data fields.

Building Contrastive Pairs in Tabular Data

Difficulty in Defining "Positive" Alterations

- Common CV techniques like image rotation or color jittering cannot be directly applied to numerical or categorical table data.
- Developing domain-specific alterations that maintain the "identity" of the data while providing meaningful augmentation is a key challenge.

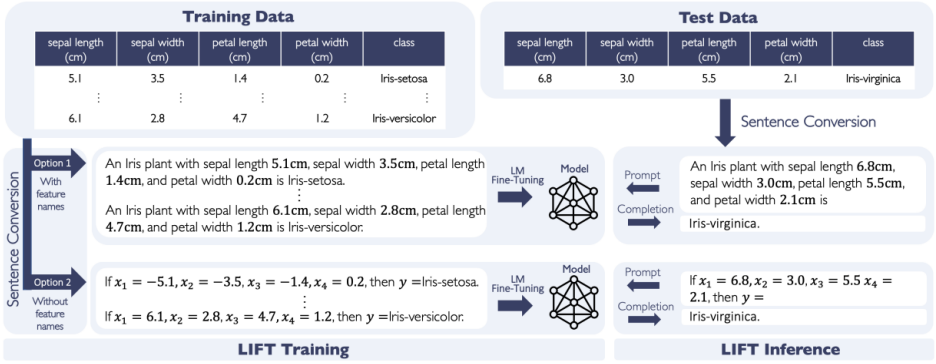
Lack of Natural Pairing Mechanisms

- Tabular data does not naturally support transformations that preserve underlying class or characteristics like images or text.
- Innovative approaches such as synthetic minority over-sampling or generative adversarial networks may be required to create effective training pairs.

Strategies for Effective Contrastive Learning

- Implementing task-specific metrics for similarity could aid in the better construction of contrastive pairs.
- Exploring cross-modal techniques, using structured explanations of data points to generate pairs, offers a promising direction.

Language As Interface



LIFT: Language Interface

Dinh et.al test LIFT method on several language models, and on both real/synthetic tabular datasets. The results are compared with LR/SVM/Decision Tree.

- In regression/classification tasks. LIFT showed performance comparable to most baselines.
- LIFT is robust against outlier and low level of random noise, though failed to defend PGD/transfer attacks.
- LIFT produces models that are calibrated and mimic the real decision boundaries.

Language Interface Enables Pre-training and Finetuning

- Converting table rows into text strings **preserves all information**, mitigating the issue of data heterogeneity and loss of semantic meaning in traditional numerical conversions [**borisov2022language**; **dinh2022lift**].
- In scenarios where real data is scarce, LLMs can **reasonably extrapolate to unseen modes** of the data, providing valuable insights beyond the initial dataset [**seedat2023curated**].
- By **fine-tuning LLMs on downstream datasets** within relevant domains, it's possible to leverage data from similar fields to enhance performance on tabular tasks specific to a particular domain [**hegselmann2023tabllm**].

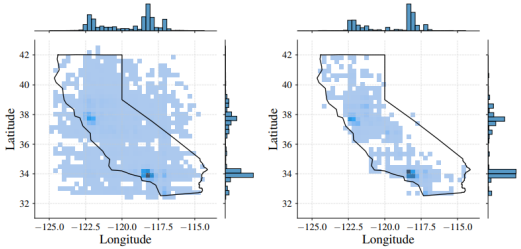
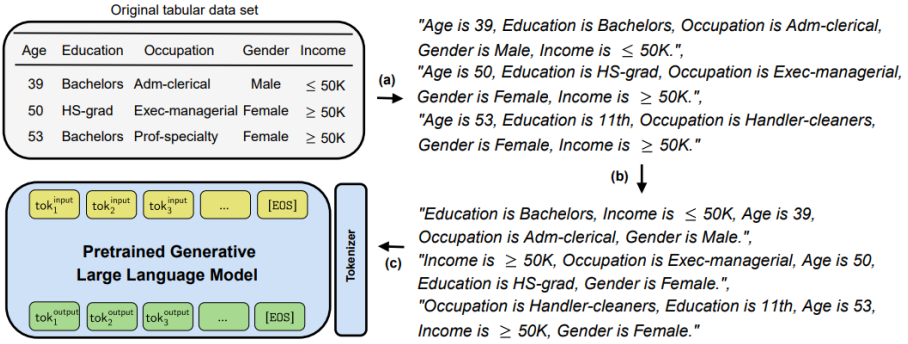


Figure: Illustration of pre-trained common sense knowledge in tabular LLM by [**borisov2022language**]. A pre-trained GPT2 generated more realistic coordinates of California Houses (Right) than CTGAN (left).

GReaT: LMs Are Realistic Table Geneators



GReaT: Formulation

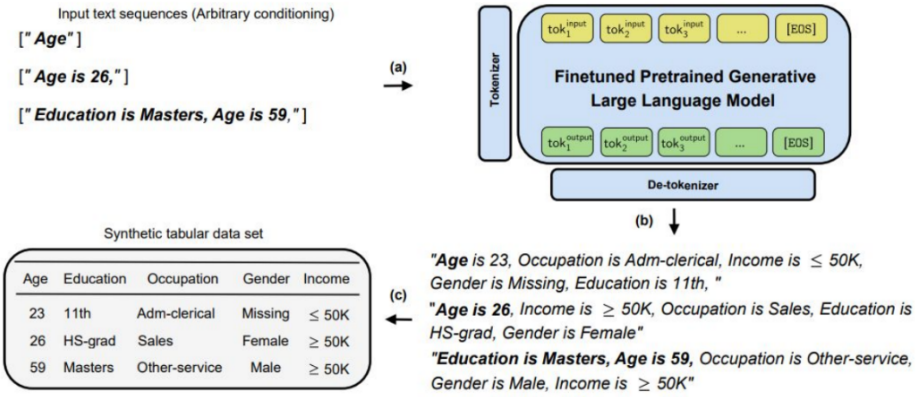
Definition 1 (Textual encoder) *Given a tabular data set of m columns with feature names $f_1, f_2, \ldots, f_m$ and n rows of samples $\mathbf{s}_1, \ldots, \mathbf{s}_n$, we let the entry $v_{i,j}, i \in \{1, ..., n\}, j \in \{1, ..., m\}$ represent the value of the j-th feature of the i-th data point. Taking the feature name and value into account, each sample $\mathbf{s}_i$ of the table is transformed into a textual representation $\mathbf{t}_i$ using the following subject-predicate-object transformation:*

$$t_{i,j} = [f_j, \text{“}is\text{”}, v_{i,j}, \text{“,”}] \qquad \forall i \in \{1, ..., n\}, j \in \{1, ..., m\}, \tag{1}$$

$$\mathbf{t}_i = [t_{i,1}, t_{i,2}, ..., t_{i,m}] \qquad \forall i \in \{1, ..., n\}, \tag{2}$$

where $t_{i,j}$, the textually encoded feature, is a clause with information about a single value and its corresponding feature name, and $[\cdot]$ denotes the concatenation operator.

GReaT: Sampling

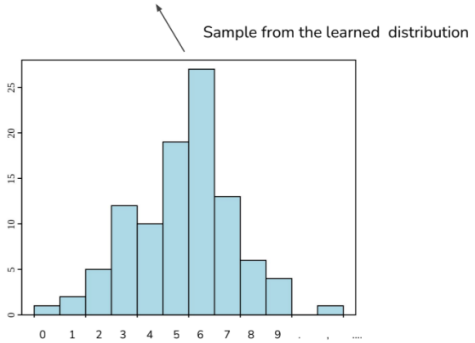


GReaT: Different Data Types

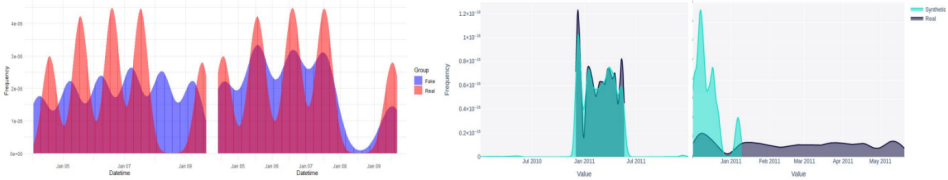
- The model predicts the distribution for the next token; we sample from that distribution and the sampled token is then added to the model input. We repeat this process until <eos> is produced.
- Combinations of tokens can represent both numerical / categorical tokens.
- Let $\boldsymbol{t}$ be a text representing a row, w be individual tokens in the text. The joint probability distribution is

$$p(\boldsymbol{t}) = p(w_1, \ldots, w_j) = \prod_{k=1}^{j} p(w_k | w_1, ..., w_{k-1}).$$

Input: age is 6, gender is female, ... , <eos>



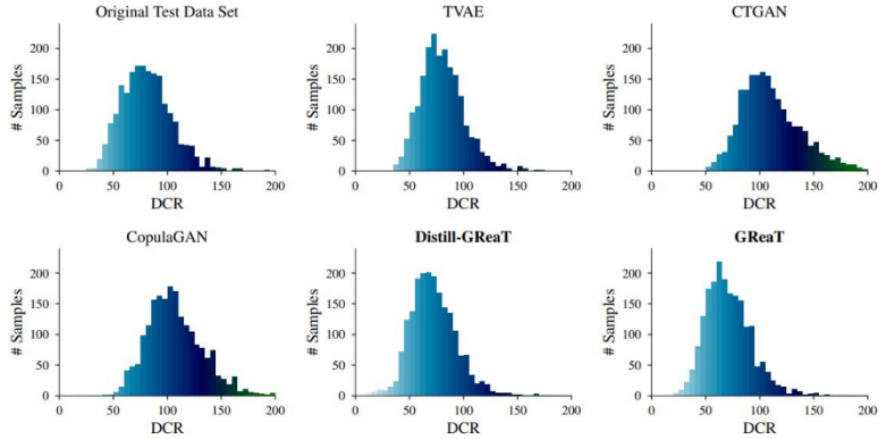
GReaT: Numerical Handling



CTGAN (cyclic encoder) GReaT CTGAN (cyclic encoder) GReaT

Synthesized InvoiceDate from Ecommerce dataset. GReat better preserved the weekly and daily pattern, but failed to capture the overall time range.

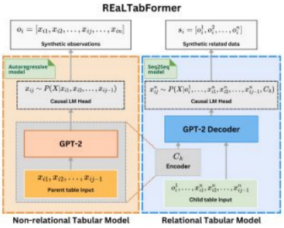
GReaT: Not Just Memorizing



Multi-Table

REaLTabFormer: Generating Realistic Relational and Tabular Data using Transformers

Aivin V. Solatorio [1] Olivier Dupriez [1]



- Solatorio et.al build on GReaT and improved the efficiency of training by reducing the vocabulary the models need to predict. It also applied masking on the target column to reduce data copying.
- The model is also able to model relational tables by first sampling a parent table, then generate child tables conditioned on the parent.

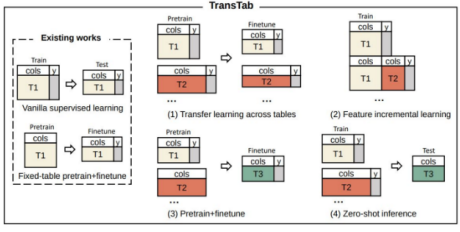
TransTab

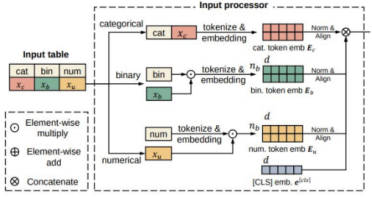
Wang et.al tokenizes feature name/value pairs into continuous embeddings. This converts tabular data into a feature space with fixed feature dimensions, thus allowing transfer learning and pretrain/finetune framework. The downstream tasks are performed in regular tabular form.

TransTab: Learning Transferable Tabular Transformers Across Tables

Zifeng Wang
UIUC
zifengw2@illinois.edu

Jimeng Sun
UIUC
jimeng@illinois.edu





Preprocessing Methods for Categorical, Binary, and Numerical Features

Categorical/Textual Features

- Combine column name and feature value to create a token sequence.
- Tokenize the sequence and map to an embedding matrix to produce embeddings $E_c \in \mathbb{R}^{n_c \times d}$.
- Here, d is the embedding dimension and n_c is the number of tokens.

Binary Features

- Binary feature values $\{0, 1\}$ are processed based on their state:
- If value is 1, tokenize and encode to embeddings $E_b \in \mathbb{R}^{n_b \times d}$.
- If value is 0, skip to reduce computational and memory overhead.

Numerical Features

- Avoid direct tokenization-embedding due to poor performance in number discrimination.
- Encode using the same method as categorical data to get $E_{u,\text{col}} \in \mathbb{R}^{n_u \times d}$.
- Multiply the numerical value by its column embedding to obtain the final embedding $E_u = x_u \times E^2_{u,\text{col}}$.
- This method is empirically found to be effective against more complex techniques.

Table Contrastive Loss: Self-VPCL Method

Overview of Self-VPCL

- A novel self-supervised learning model that reduces computational costs and minimizes overfitting.
- Utilizes vertical partitions of the dataset to build diverse and effective positive and negative samples.

Constructing Samples

- Positive Samples: Generated from different partitions within the same table record.
- Negative Samples: Constructed using partitions from different records.
- This method leverages overlapping regions between neighbouring partitions to enhance the robustness of the embeddings.

Mathematical Formulation

$$L(X) = -\sum_{i=1}^{B}\sum_{k=1}^{K}\sum_{k' \neq k} \log \frac{\exp(\psi(v_{ki}, v_{k'i}))}{\sum_{j=1}^{B}\sum_{k^*=1}^{K} \exp(\psi(v_{ki}, v_{k^*j}))} \tag{1}$$

Where:

- B is the batch size, K is the number of partitions.
- $\psi(.,.)$ is the cosine similarity function between the linear projections of the partitions' embeddings.

Supervised VPCL for Tabular Data

Introduction to Supervised VPCL

- Traditional supervised pre-training often uses a task-specific predicting head with cross-entropy loss.
- In Supervised VPCL, we drop these heads post pre-training and introduce a new head during fine-tuning to improve model transferability.

Why Move Beyond Traditional Supervised Methods?

- Bias Towards Major Tasks: Supervised loss can cause the encoder to become biased to predominant classes and tasks.
- Hyperparameter Sensitivity: Difficulty in selecting optimal hyperparameters due to varying dataset characteristics.
- These factors lead to reduced effectiveness in model transferability across different tabular datasets.

Mathematical Formulation of Supervised VPCL

$$L(X,y) = -\sum_{i=1}^{B}\sum_{j=1}^{B}\sum_{k=1}^{K}\sum_{k'=1}^{K} \mathbf{1}_{y_j=y_i} \log \frac{\exp(\psi(v_{ki}, v_{k'j}))}{\sum_{j^*=1}^{B}\sum_{k^*=1}^{K} \mathbf{1}_{y_{j^*}\neq y_i} \exp(\psi(v_{ki}, v_{k^*j^*}))} \tag{2}$$

Where:

- $\mathbf{1}_{y_j=y_i}$ is an indicator function denoting if samples i and j are from the same class.
- $\psi(\cdot,\cdot)$ is the cosine similarity function between embeddings.

In-Context Learning in Large Language Models

What is In-Context Learning?

- In-context learning refers to the ability of LLMs to infer and apply knowledge from provided examples within a single prompt, without prior explicit training on these examples.
- The model uses the context of the input itself to generate appropriate outputs, essentially learning "on the fly."

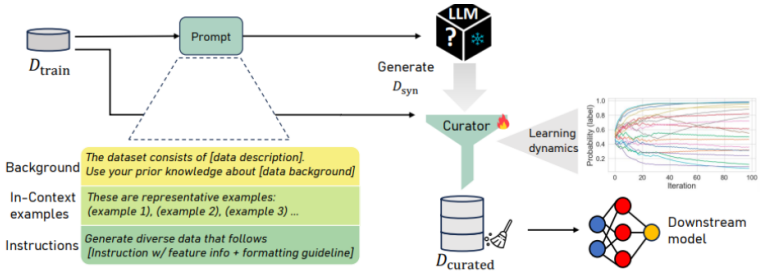
Key Characteristics

- Flexibility: Adapts to new tasks or data types based solely on the examples included in the prompt.
- Zero-shot or Few-shot: Effective even when very few examples are provided (few-shot) or none at all (zero-shot).

Difference from Meta-Learning

- Training Requirement: Meta-learning involves training a model on a variety of learning tasks, intending to generalize well on new tasks. In contrast, in-context learning does not use explicit training on tasks but relies on the context provided in the input.
- Generalization: Meta-learning explicitly aims for fast adaptation to new tasks using learned knowledge and skills, whereas in-context learning implicitly adjusts its responses based on given examples.
- Mechanism: In meta-learning, models are typically trained through algorithms like Model-Agnostic Meta-Learning (MAML) which adjust internal model parameters. In-context learning, however, leverages the existing knowledge embedded in the model weights.

LLM as Few-Shot Tabular Synthesizer



- Augmentation using pre-training knowledge: the diverse pre-training corpus of LLMs carries valuable prior knowledge
- a post-generation data curation mechanism based on sample learning dynamic. This addresses the overlooked aspect that not all of the synthetic samples are useful to downstream model performance.

Generation By Prompt Engineering

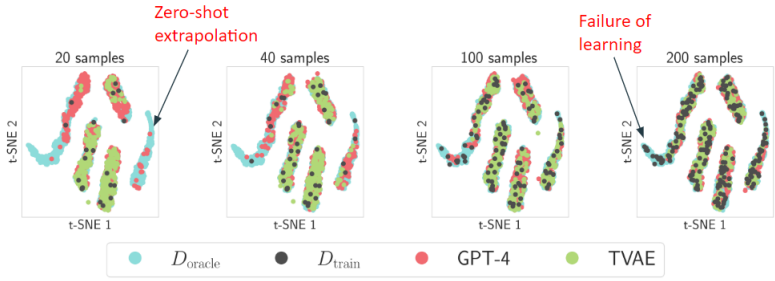
```
System role: 'You are a tabular synthetic data generation model.'

You are a synthetic data generator.
Your goal is to produce data which mirrors \
the given examples in causal structure and feature and label distributions \
but also produce as diverse samples as possible.

I will give you real examples first.

```

Extrapolation By Prior Knowledge



T-Sne plot of real and synthetic examples. GPT-4 is able to extrapolate to regions of the oracle (true manifold) even where there is no training data covering them, as can be seen by the overlap with the turquoise dots, with the effect more pronounced when Dtrain is small.

Summary: How Are the Challenges Addressed?

- **Heterogenous Tables**: LLM-based method unified them into token or embedding spaces.
- **Untransferable Features**: LLM-based models add semantic meaning to table representations that are consistent throughout models.
- **Proper Training Loss**: By reducing heterogeneous categories/number prediction to token prediction/contrastive projection, we are able to find meaningful pretraining objects that are generalizable to different downstream tasks, opening oppurtunity of pre-training / fine-tuning diagrams that enables the success of DL in other domains.

Problems: Redundancy and Biases

- **Tokens are Expensive**: text-serialization of table rows introduces redundant tokens, which can incur prohibitive computing costs.
- **Syntactic vs Semantic:** Reducing tabular problems to language modeling risk LMs confusing syntactic information with statistical distribution and induces biases from pre-training data.
  - Example: A language model might simplify the relationship between symptoms and diagnosis by associating common co-occurrences in training data (e.g., cough and fever) directly with frequent diagnoses like the flu. This could lead to overlooking more serious conditions with similar symptoms, such as a pulmonary embolism, which could be fatal if misdiagnosed.
- **Catastrophic Forgetting:** Finetuning LMs on tabular data might lead to forgetting previously learned knowledge from natural language corpora, a phenomenon known as catastrophic forgetting.

Problems: Tokenization Losses Table Structure

- **Tokenization**: Language models decompose both categorical labels and numeric values into tokens, which can obscure the original structural and numerical meanings.
- **Categories & Hierarchy**:Textual representations often fail to distinguish between columns that contain categorical data with a finite set of options and those that hold free-form text.
- **Numeracy**: The tokenization process can handle floating-point numbers inconsistently, reducing LMs‘ ability to model complex numerical data.