Lecture 15

Generative Chemistry and Variational Autoencoder

Contents

Starting from this lecture, we will discuss generative models and their applications in chemistry.

1 Generative Chemistry

While a large component of chemical data science focuses on making plausible predictions, another central task in chemistry is to **create new chemicals**. Conventional discovery often relies on years of expertise combined with trial and error in the laboratory.

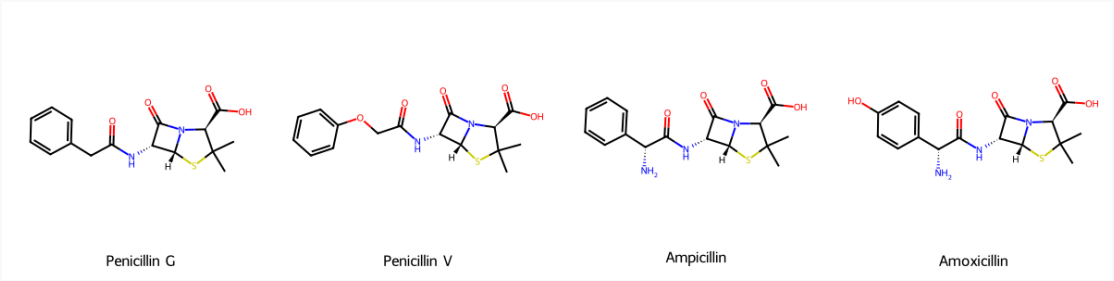
How do chemists decide which new chemical to try? A common strategy is to start from existing chemicals that already achieve the desired goal to some extent, but are not entirely satisfactory. Chemists then explore nearby chemical variations.

We illustrate this strategy with two examples: the design of semi-synthetic penicillins and the development of lithium-ion battery cathode materials.

Example: Designing Semi-Synthetic Penicillins

Natural penicillin was discovered by Alexander Fleming in 1928. Although penicillin revolutionized antibiotic therapy, early penicillins had several limitations, including poor stability in acidic environments, a limited antibacterial spectrum, and the rapid emergence of bacterial resistance.

Rather than searching randomly for new antibiotics, chemists modified the side chains attached to the penicillin core structure, which contains a characteristic β-lactam ring. By systematically varying the side chains while preserving the core scaffold, many improved antibiotics were developed. Examples are illustrated below, where penicillin G represents the original structure.



Example: Lithium-Ion Battery Cathodes

The first practical lithium-ion battery cathode material was $LiCoO_2$, discovered by John B. Goodenough and collaborators in 1980. This layered transition-metal oxide enabled reversible lithium intercalation and became the foundation of modern lithium-ion batteries.
Starting from this structure, researchers explored chemically related compounds by substituting different transition metals. This led to the development of materials such as $LiNiO_2$, $LiMnO_2$, and mixed-metal cathodes including NMC ($LiNiMnCoO_2$).
This **scaffold + substitution** strategy has also guided the development of sodium-ion batteries. In this case, Li^+ is replaced by Na^+, leading to analogous layered materials such as $NaCoO_2$, after which the transition-metal composition can again be varied to optimize performance.

In summary, traditional chemical discovery often follows the pattern

Known working system → small modifications

1.1 Probability View of Discovery

This empirical strategy can also be interpreted from a probabilistic perspective. If one compound is known to possess a desired property, then other compounds with similar structures are more **likely** to exhibit related properties. In other words, promising candidates are often located in the *neighborhood* of known successful compounds in chemical space.

A natural strategy is therefore to sample compounds from a probability distribution that favors such regions of chemical space. To construct such a distribution, two key questions arise:

1. What random variables $\mathbf{x}$ should represent a chemical compound?
2. How can we approximate the probability distribution function $p(\mathbf{x})$?

Generative models provide principled approaches to address both of these questions in an automatic manner:

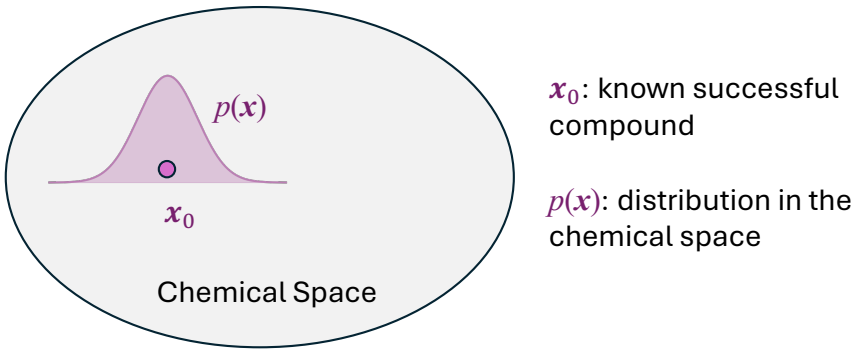


Figure 1: Probabilistic view of chemical discovery.

- *Answer to Question 2:* As in many machine learning models, we introduce a **parameterized** probability distribution for the data,

$$\mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x}), \tag{1}$$

where $\boldsymbol{\theta}$ denotes model parameters. The goal of training is to optimize $\boldsymbol{\theta}$ so that the model distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$ approximates the distribution of the observed chemical data.

- *Answer to Question 1:* The random variable $\mathbf{x}$ can represent a chemical system using a suitable structural representation (for example, molecular graphs, SMILES strings, or atomic coordinates). A common modeling strategy is to introduce additional **latent variables z** that capture hidden factors governing the data. The generative model then assumes

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\, p(\mathbf{z})\, \mathrm{d}\mathbf{z}, \tag{2}$$

where $p(\mathbf{z})$ is a simple prior distribution and $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ describes how the latent variables generate observable structures.

Different ways of modeling the above components lead to different classes of generative models. These models vary in how the distribution is represented, how the latent variables are handled, and how the parameters are trained. In the following sections we will discuss several major types of generative models and their applications in chemistry.

1.2 Overview of Generative Models and Chemical Applications

In the following lectures, we will cover three widely used families of generative models: variational autoencoders (VAEs), autoregressive models, and diffusion models. Each type of generative model has its own strengths. Autoregressive models are well suited for sequential

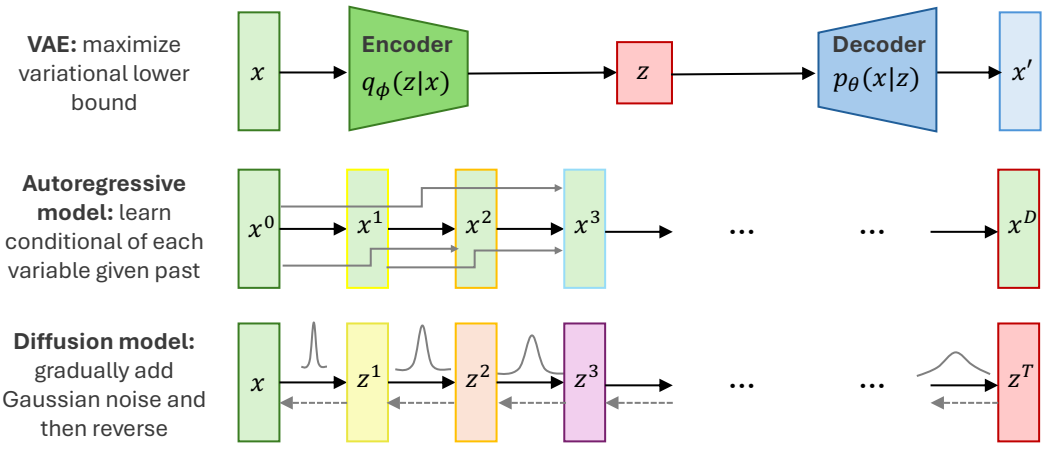


Figure 2: Architectures of three generative models.

chemical representations, VAEs provide an interpretable latent space for exploring chemical variations, and diffusion models are powerful tools for generating complex molecular and materials structures. We provide a high-level overview of the three methods in this section, and will discuss each model and their applications in the following lectures. A summary of the three models can be found in Fig. 2.

- **Variational Autoencoders (VAEs).** VAE models introduce a continuous latent space **z** to describe the variability of the data. A VAE consists of two neural networks: an encoder that maps a data point **x** to a latent representation **z**, and a decoder that reconstructs **x** from **z**. In chemistry, VAEs are commonly used for *molecular design*, *property optimization*, and *exploration of chemical space*, where new molecules can be generated by sampling or modifying latent variables.
- **Autoregressive Models.**Autoregressive models construct the probability distribution of the data by factorizing it into a **sequence** of conditional probabilities,

$$p(\mathbf{x}) = \prod_i p(x_i \mid x_1, \ldots, x_{i-1}). \tag{3}$$

  This formulation is natural when the data can be represented as a sequence.
  In chemistry, a common representation is the SMILES string of a molecule, which encodes molecular graphs as linear sequences of tokens. These models are widely used for *molecular generation*, *reaction prediction* and *retrosynthesis*.
- **Diffusion Models.** Diffusion models generate data by simulating a gradual denoising process. Starting from random noise, the model iteratively removes noise through a sequence of

learned transformations until a structured sample is obtained. In chemistry and materials science, diffusion models have been applied to *molecular graph generation*, *3D molecular structure generation*, *protein structure design*, and *crystal structure generation*. Their ability to model complex high-dimensional distributions makes them particularly suitable for problems involving atomic coordinates and spatial structures.

2 Variational Autoencoder (VAE)

When modeling a chemical system, we often use features that are easily interpretable to humans, such as molecular structures or structure-related descriptors (e.g., fingerprints). These representations typically lead to a high-dimensional feature space. However, many learning tasks do not require such high dimensionality. High-dimensional feature spaces can make machine learning training more difficult, since many features may be irrelevant to the target task, while others may be strongly correlated with each other.

We have previously encountered dimensionality-reduction techniques such as feature selection and principal component analysis (PCA). These methods are typically applied as preprocessing steps before training a model. As a result, they can remove obviously irrelevant features or reduce correlations among features, but they are not directly optimized for the downstream learning task.

How can we learn a "cleaned-up" feature space that is tailored to the task itself? Variational autoencoders (VAEs) provide an automatic approach for learning such representations. We refer to this lower-dimensional representation as the **latent space**. We use $\mathbf{x}$ to denote the original data, and $\mathbf{z}$ to denote the **latent variables**.

To connect the original variables and the latent variables, we introduce two components: an **encoder** and a **decoder**.

$$\text{original variable } \mathbf{x} \xrightarrow{\text{encoder}} \text{latent variable } \mathbf{z} \xrightarrow{\text{decoder}} \text{reconstructed variable } \mathbf{x}'$$

We have already encountered a model with an encoder-decoder structure: the Seq2Seq model. In Seq2Seq models, both the encoder and the decoder are typically implemented using recurrent neural networks (RNNs). The input data $\mathbf{x}$ is a sequence, and the latent representation $\mathbf{z}$ is often taken as the final hidden state of the encoder RNN.

In a VAE, however, the goal is not merely to map inputs to outputs, but to learn the **probability distribution** of the data. To achieve this, we introduce a latent-variable generative model:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\, p(\mathbf{z})\, \mathrm{d}\mathbf{z}, \tag{4}$$

where $p(\mathbf{z})$ is called the **prior distribution**.

In the following sections, we describe how to construct the encoder and decoder networks that connect $\mathbf{x}$ and $\mathbf{z}$, and how to design the loss function used to train a VAE model.

2.1 Encoder

In a standard autoencoder, the encoder maps the input $\mathbf{x}$ to a single latent vector $\mathbf{z}$. In a variational autoencoder, the encoder instead *outputs the parameters of a probability distribution over the latent variables*, as illustrated in Fig. 3.

In practice, the encoder produces two vectors,

$$\boldsymbol{\mu}(\mathbf{x}), \quad \log \boldsymbol{\sigma}^2(\mathbf{x}), \tag{5}$$

which define a Gaussian distribution

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}\Big(\boldsymbol{\mu}(\mathbf{x}), \mathrm{diag}(\boldsymbol{\sigma}^2(\mathbf{x}))\Big). \tag{6}$$

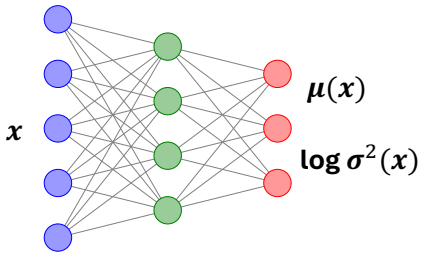


Figure 3: The encoder neural network in a variational autoencoder.

Here $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is called the **approximate posterior distribution**, and the parameters $\boldsymbol{\phi}$ denote the weights of the encoder neural network.

Instead of producing a single latent vector, the encoder therefore describes a distribution of possible latent representations that could have generated the observed data $\mathbf{x}$. A latent vector $\mathbf{z}$ is then **sampled** from this distribution.

The encoder step is also called the **inference step**, since we infer latent variables from the observed data.

2.2 Decoder

Once we obtain $(\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\sigma}^2(\mathbf{x}))$ from the encoder, we can sample a latent variable $\mathbf{z}$ from the Gaussian distribution

$$\mathcal{N}\Big(\boldsymbol{\mu}(\mathbf{x}), \mathrm{diag}(\boldsymbol{\sigma}^2(\mathbf{x}))\Big) \xrightarrow{\text{sample}} \mathbf{z}. \tag{7}$$

To allow gradients to propagate through the sampling step during training, the sampling is implemented using the **reparameterization trick**:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \tag{8}$$

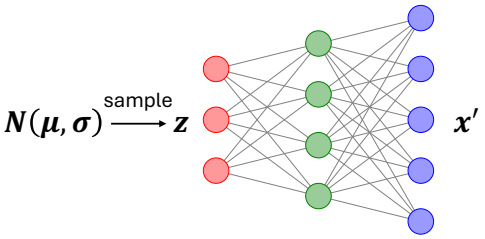


Figure 4: The decoder neural network in a variational autoencoder.

where $\boldsymbol{\epsilon}$ is a noise vector drawn from a standard normal distribution,

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I).$$

This formulation separates the stochastic sampling from the deterministic network outputs, which makes gradient-based optimization possible.

Once a latent vector $\mathbf{z}$ is obtained, the decoder takes $\mathbf{z}$ as input and generates a *reconstruction of the original data*. From a probabilistic perspective, the decoder represents the conditional distribution

$$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}), \tag{9}$$

which describes how latent variables generate observable data.

The decoding step is also called the **generative** step.

2.3 Choices of Neural Networks

The neural networks used to connect the data $\mathbf{x}$ and the latent variable $\mathbf{z}$ can take many different forms.

A common choice is a feed-forward neural network, also called a multi-layer perceptron (MLP). However, different architectures are often used depending on the structure of the data. Below is a summary of common neural network choices for the encoder and decoder.

- **Feed-forward neural networks (FNN).** Used when the input $\mathbf{x}$ is a fixed-length vector, such as molecular descriptors or fingerprints. The VAE architecture based on FNN is shown in Fig. 5.
- **Graph neural networks (GNN).** When $\mathbf{x}$ is represented as a molecular graph, we can use GNNs to encode structural information:

  $$\textbf{Encoder:} \quad \text{graph} \rightarrow \text{GNN} \rightarrow (\boldsymbol{\mu}, \boldsymbol{\sigma})$$
  $$\textbf{Decoder:} \quad \mathbf{z} \rightarrow \text{graph generator}$$

  For the graph generator, one simple approach is to train an MLP that maps $\mathbf{z}$ to the matrices describing the graph (such as node features and adjacency matrices). More advanced graph generators include sequential graph construction or fragment-based generation.
- **Recurrent neural networks (RNN).** When the input data $\mathbf{x}$ is a sequence (for example SMILES strings), we can use RNNs or transformers (which will be discussed in later lectures).

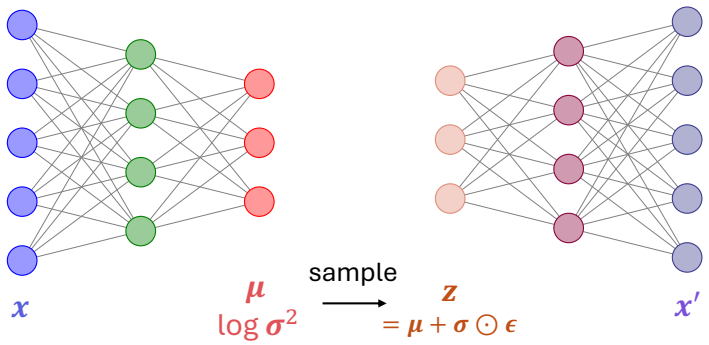


Figure 5: VAE architecture with FNNs as the encoder and decoder.

In this case, the encoder processes the sequence and produces a final hidden state $\mathbf{h}_T$. Instead of using $\mathbf{h}_T$ directly as the latent representation, we map it to the parameters of a Gaussian distribution:

$$\begin{aligned} \boldsymbol{\mu} &= W_\mu \mathbf{h}_T + \mathbf{b}_\mu \\ \log \boldsymbol{\sigma}^2 &= W_\sigma \mathbf{h}_T + \mathbf{b}_\sigma \end{aligned} \tag{10}$$

The latent variable $\mathbf{z}$ is then sampled from this distribution.

2.4 Training Objective

Our goal is to learn the probability distribution of the data, $p_{\boldsymbol{\theta}}(\mathbf{x})$. Given a dataset $\{\mathbf{x}\}$, a natural objective is to maximize the likelihood of observing the data under this model. Therefore, the training objective is

$$\text{Training objective:} \quad \max \ \log p_{\boldsymbol{\theta}}(\mathbf{x}). \tag{11}$$

However, evaluating $p_{\boldsymbol{\theta}}(\mathbf{x})$ directly is difficult because it involves integrating over the latent variable $\mathbf{z}$:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\, p(\mathbf{z})\, \mathrm{d}\mathbf{z}. \tag{12}$$

To make this tractable, we introduce an approximate posterior distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, represented by the encoder network. Inserting this distribution into the integral gives

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) p(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\, \mathrm{d}\mathbf{z}. \tag{13}$$

Applying Jensen's inequality yields a lower bound of $\log p_{\boldsymbol{\theta}}(\mathbf{x})$, known as the **Evidence Lower Bound (ELBO)**:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right]}_{\text{reconstruction term}} - \underbrace{D_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})\right)}_{\text{regularization term}}. \tag{14}$$

Here $D_{\mathrm{KL}}(\cdot\|\cdot)$ denotes the **Kullback–Leibler (KL) divergence**, formally defined as

$$\text{KL Divergence: } D_{\mathrm{KL}}\left(q(x)\|p(x)\right) = \mathbb{E}_{q(x)}\left(\log\frac{q(x)}{p(x)}\right). \tag{15}$$

The first term encourages the decoder to reconstruct the input data accurately, while the second term regularizes the latent variables so that their distribution remains close to the prior $p(\mathbf{z})$. The ELBO balances *accuracy of reconstruction* with *smooth, organized latent space* to make both learning and generation work well.

In practice, training a VAE corresponds to *maximizing the ELBO*, or equivalently *minimizing the negative ELBO*:

$$\mathcal{L} = -\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] + D_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})\right). \tag{16}$$

2.4.1 Practical implementation

Unlike basic loss functions such as MSE and cross entropy, there is no built-in implementation of ELBO in ML packages such as `PyTorch` or `TensorFlow`. However, the implementation is straightforward.

Reconstruction term

This measures how well the decoder can reproduce the input from the latent variable $\mathbf{z}$, i.e., how close between the input data $\mathbf{x}$ and the reconstructed data $\mathbf{x}'$.

Therefore, we can use the standard loss for this term. For continuous $\mathbf{x}$, we use the mean sqaured error (MSE), and for discrete $\mathbf{x}$, we use cross entropy.

$$\boxed{-\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] \rightarrow \mathrm{MSE}(\mathbf{x},\mathbf{x}') \text{ or } \mathrm{CrossEntropy}(\mathbf{x},\mathbf{x}')} \tag{17}$$

KL regularization term

This ensures that the latent space is smooth and well-behaved by keeping the encoded distributions close to the standard Gaussian prior $p(\mathbf{z}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

We have closed forms for both $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{z})$:

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \sim \mathcal{N}(\boldsymbol{\mu}(\mathbf{x}), \mathrm{diag}(\boldsymbol{\sigma}^2(\mathbf{x}))). \tag{18}$$

The final form for the KL-divergence is

$$\boxed{D_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})\right) = -\frac{1}{2}\sum_i (1 + \log\sigma_i^2 - \mu_i^2 - \sigma_i^2)} \tag{19}$$

Therefore, one can easily implement the negative ELBO to train a VAE.

2.5 Generating New Data

Once the model is trained, new chemical candidates can be generated by sampling latent variables $\mathbf{z}$ from the **prior** distribution $p(\mathbf{z})$, and then passing these latent variables through the decoder network with fixed parameters to produce new samples of $\mathbf{x}$.

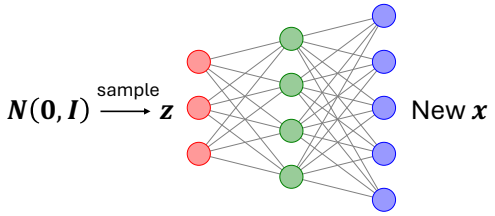


Figure 6: Generating New Data from trained decoder.

The reason we sample $\mathbf{z}$ from the prior rather than the posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is that the posterior represents the distribution of $\mathbf{z}$ conditioned on **observed data**. In contrast, when generating new data, the corresponding $\mathbf{x}$ is not known in advance, so the posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ cannot be evaluated.

Instead, we sample $\mathbf{z}$ from the prior distribution. During training, the KL divergence term encourages the approximate posterior to remain close to the prior, ensuring that latent variables drawn from the prior lie in regions of the latent space that the decoder has learned to map to realistic data.

3 Conditional Variational Autoencoder (CVAE)

A Conditional VAE (CVAE) is an extension of the standard VAE that allows us to control the generated output based on some known attribute or property, denoted by $\mathbf{y}$. This is especially useful in chemistry, where we may want to generate molecules with a *specific target property* (e.g., logP, molecular weight, activity against a target).

In a standard VAE, the encoder maps $\mathbf{x} \rightarrow q_{\phi}(\mathbf{z}|\mathbf{x})$ and the decoder reconstructs $\mathbf{x}$ from $\mathbf{z}$ via

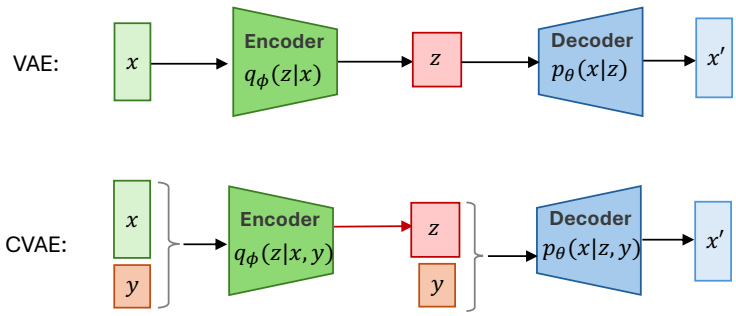


Figure 7: VAE architecture (top) vs. conditional VAE (CVAE) architecture (bottom).

$p_\theta(\mathbf{x}|\mathbf{z})$. - In a CVAE, both the encoder and decoder are **conditioned on y**:

$$q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y}), \quad p_\theta(\mathbf{x}|\mathbf{z},\mathbf{y}) \tag{20}$$

Intuitively, we are learning a latent space specific to the property **y**, so that sampling from the latent space while fixing **y** will generate molecules matching the desired property.

3.1 Architecture Changes

1. **Encoder:** takes both **x** and **y** as input, and outputs the $\boldsymbol{\mu}$ and $\log \boldsymbol{\sigma}^2$.
2. **Decoder:** takes **z** and **y** as input and output the reconstructed $\mathbf{x}'$.

Everything else (latent distribution, KL divergence, ELBO) remains the same. The loss function is unchanged except that the conditional variables are included:

$$\mathcal{L} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}\left[\log p_\theta(\mathbf{x}|\mathbf{z},\mathbf{y})\right] - D_{\mathrm{KL}}\Big(q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})\|p(\mathbf{z})\Big) \tag{21}$$

A comparison between VAE and conditional VAE (CVAE) is shown in Fig. 7.

4 Lab: Latent Space Learning with VAE

In addition to generating new data, such as molecules, VAE can also be used to construct a **compressed, structured representation** of high-dimensional representations, i.e., the **latent space**. We can use the latent vectors **z** as the principal components.

Let's take molecules as an example, the latent space learned by VAE has the following properties:

- The latent variables are encouraged to follow a *standard Gaussian distribution*, which ensures a smooth and continuous space.
- Molecules with similar structures or properties are mapped to nearby points.

- Interpolation between two points $\mathbf{z}_1$ and $\mathbf{z}_2$, i.e.,

$$\mathbf{z}_\alpha = (1-\alpha)\mathbf{z}_1 + \alpha\mathbf{z}_2, \quad \alpha \in [0,1].$$

  can be decoded into a smooth transformation from molecule 1 to molecule 2.
- Sampling from the latent space allows generation of new molecules that are chemically plausible.

4.1 Latent Space Visualization

For visualization, we can choose 2-3 latent dimensions for visualization. The latent dimension should roughly follow a Gaussian distribution.

We can also use the t-SNE (t-distributed Stochastic Neighbor Embedding) visualization to map the high-dimensional latent space to two or three dimensions.

With t-SNE visualization, points that are closer to each other in the latent space are placed near each other in the 2D plot.

We can assign a color corresponding to a certain chemical property (e.g., logP, molecular weight) to the point, then we can see the trends in the latent space.

The t-SNE visualization is provided in `scikit-learn`:

```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2)
z_2d = tsne.fit_transform(z)
```

Link:

`https://colab.research.google.com/drive/1EDJ-8JU_l2F120V5NgamyTuI_rXLuXmI?usp=sharing`