Lecture 11: Variational Autoencoders - II
COMP 5801H/4900A: Generative AI and LLMs

2026-02-10

Sriram Subramanian
Assistant Professor & Canada Research Chair, Carleton University
Faculty Affiliate, Vector Institute for Artificial Intelligence
Faculty Affiliate, Schwartz Reisman Institute for Technology and Society

Carleton University

VECTOR INSTITUTE

SCHWARTZ REISMAN INSTITUTE FOR TECHNOLOGY AND SOCIETY

Lecture Outline

- The Reparameterization Trick
- Latent Space Properties
- Conclusion

The Stochastic Bottleneck

- **The Problem: Randomness is a Dead End**
  - Backpropagation relies on the Chain Rule. To update the weights of the Encoder, we need to calculate the gradient of the Loss with respect to the Encoder's parameters (ϕ)
  - In a standard network, the data flows like water through a pipe
  - In a VAE, we introduce a random sampling step: $z \sim \mathcal{N}(\mu, \sigma^2)$
- **Why can't we differentiate "Random"?**
  - To find a gradient, you need to know how a small change in the input affects the output.
  - If you change μ slightly, how does z change?
  - You can't know. Because z is being picked by a "roll of the dice" (the sampling process), the output is non-deterministic
  - You cannot take the derivative of a random variable. The "static" of the sampling step blocks the gradient from flowing back to the Encoder

The Encoder is Left in the Dark

- The Decoder can still learn (it's downstream of the noise)
- The Encoder, however, never receives any feedback. It doesn't know if the μ and σ it predicted were helpful or harmful to the final reconstruction
- Without a fix, the Encoder remains "**untrained**," and the **whole system collapses**

The Reparameterization Trick

- The Transformation Formula
  - Instead of letting the Encoder output a random variable z directly, we express z as a deterministic function of μ,σ, and a separate source of noise ϵ:

$$z = \mu + \sigma \odot \epsilon$$

  - μ and σ: Deterministic outputs of the Encoder (we can calculate gradients for these)
  - ϵ: A "noise" variable sampled from a fixed Standard Normal Distribution $\mathcal{N}(0,I)$
  - $\odot$: Element-wise multiplication

The Reparameterization Trick

- **Moving the Randomness "Outside"**
  - By doing this, we treat the randomness as an input from the environment rather than a "broken link" in the middle of the network
  - The sampling happens at ϵ, which doesn't have any weights to train
  - The path from the Loss, through the Decoder, and back to μ and σ is now smooth and continuous.
- **Summary of the Flow**
  - Encoder predicts μ and σ
  - Generate a random ϵ (e.g., 0.42)
  - Calculate $z = \mu + (\sigma \times 0.42)$
  - Decoder uses this z to reconstruct the image.
  - Gradients flow back through z, then through the addition and multiplication, directly into μ and σ

Flow of Gradients after Reparameterization

- **Visualizing the Gradient Highway**
  - The Deterministic Path: The path from the Loss $\rightarrow$ Decoder $\rightarrow z \rightarrow \mu$ (or σ) is now composed entirely of differentiable operations (addition and multiplication)
  - The Stochastic Branch: The random variable ϵ is treated as a constant input (like the data x itself). Since we don't need to find the gradient of the noise, the "randomness" no longer blocks the flow
- **The Chain Rule in Action**
  - When we calculate the gradient of the loss $\mathscr{L}$ with respect to the Encoder parameters ϕ, the chain rule now looks like this:

$$\frac{\partial \mathscr{L}}{\partial \phi} = \frac{\partial \mathscr{L}}{\partial \hat{x}} \cdot \frac{\partial \hat{x}}{\partial z} \cdot \frac{\partial z}{\partial \phi}$$

  - Because $z = \mu_\phi(x) + \sigma_\phi(x) \cdot \epsilon$, the derivative $\frac{\partial z}{\partial \phi}$ is perfectly well-defined. It's just the derivative of the network's outputs

Flow of Gradients

- This flow of gradients tells the Encoder exactly **how to adjust** its "predictions" of μ and σ
  - If the reconstruction is blurry: The gradient flows back and tells the Encoder to adjust μ to a better location or to shrink σ to be more precise
  - If the latent space is too messy: The gradient from the KL term flows back and tells the Encoder to pull μ closer to zero
- **Why "Standard" SGD Works?**
  - Because we've turned a stochastic process into a functional one, we don't need specialized "Reinforce" algorithms or complex Bayesian estimators. We can use:
    - Adam / RMSProp / SGD: Any standard optimizer works.
    - Automatic Differentiation: Frameworks like PyTorch and TensorFlow can handle the entire VAE training loop just like a normal neural network.

Implementation Details

- **The Problem with Direct Variance**
  - If we forced the network to output σ or σ^2 directly, we would run into two major technical "disasters":
    - **The Positivity Constraint**: Variance must always be positive ($\sigma^2 > 0$). A neural network's linear output layer can produce any number from $-\infty$ to $+\infty$. If the network outputs a negative number for variance, the math (like $\sqrt{\sigma^2}$ or $\ln \sigma^2$) crashes.
    - **Exploding Gradients**: Variance can be a very small fraction (0.0001) or a very large number. These extremes lead to unstable gradients that make training "diverge" or fail.

Implementation Details

- **The Solution: The Exponential Map**
  - By having the network predict $s = \log(\sigma^2)$ instead, we solve both problems:
    - **Natural Positivity**: We calculate the actual standard deviation using the exponential function:

$$\sigma = \exp(0.5 \cdot \log \sigma^2)$$

    - Since e^x is always positive, the model can output any real number for the "log-variance" without ever breaking the laws of math

Implementation Details

- **Mathematical "Ease of Use"**
  - The KL Divergence formula requires us to calculate $\ln(\sigma^2)$
  - If our network outputs σ^2, we have to take the log of it: $\ln(\text{output})$
  - If our network outputs the log-variance directly, we just use the output as-is! This simplifies the computation and makes the backpropagation smoother

Latent Space Interpolation

- **What is Interpolation?**
  - Interpolation is the process of picking two points in the latent space, z_A (e.g., a "Digit 0") and z_B (e.g., a "Digit 6”), and traveling along a straight line between them
  - We take a step-by-step path: $z_{new} = (1 - \alpha)z_A + \alpha z_B$, where α ranges from 0 to 1
  - We feed each intermediate z_{new} into the Decoder.
- **Smooth Transitions vs. Sudden Jumps**
  - In a Standard Autoencoder: The space is fragmented. If you move between two points, the output usually looks like "static" or a messy blur until you hit the next cluster.
  - In a VAE: The output changes semantically. You might see a "0" gradually grow a tail and close its loop to become a "6." This proves the model has learned the features of the data, not just memorized the pixels

Latent Space Interpolation

- **Traversal of Latent Attributes**
  - By moving along a single dimension of the latent vector while keeping others fixed, we can discover what specific neurons have learned
    - Dimension 5: Might control the "thickness" of a line
    - Dimension 12: Might control the "rotation" of an object
    - Dimension 20: Might control the "smile" on a face
- **Why This Matters: Creativity and Control**
  - Interpolation isn't just a cool trick; it’s why VAEs are used in generative design:
    - Morphing: Creating smooth animations between two different states
    - Denoising: Finding the "clean" version of a noisy image by projecting it into the structured manifold
    - Feature Engineering: Generating new data samples that share properties of existing ones but are unique

Latent Vector Arithmetic

- **The Concept of Attribute Vectors**
  - To manipulate a feature like "sunglasses," we first identify the direction in latent space that represents that feature:
    - Take the average latent vector of many faces with sunglasses ($z_{sunglasses}$)
    - Take the average latent vector of many faces without sunglasses (z_{plain})
    - Calculate the difference: $v_{attr} = z_{sunglasses} - z_{plain}$
    - This vector v_{attr} now represents the abstract concept of "sunglasses" in the model's mind
- **Performing the Arithmetic**
  - Once we have the attribute vector, we can apply it to a completely new person (z_{new_person}):

$$z_{modified} = z_{new\_person} + \alpha \cdot v_{attr}$$

    - Addition: Adding the vector "puts on" the sunglasses
    - Subtraction: Subtracting the vector can "remove" a feature
    - Scaling (α): Adjusting the intensity (e.g., making the sunglasses darker or larger)

Latent Vector Arithmetic

- **Why This Works: Linear Subspaces**
  - In a well-trained VAE, the complex, non-linear relationships of pixels are untangled into linear relationships in the latent space.
    - Rotation, lighting, gender, and accessories often become linear directions
    - This is the "Holy Grail" of generative modelling: **Controllable Synthesis**. You aren't just generating a random face; you are editing specific properties of that face
- **Limitations: Entanglement**
  - If the model isn't perfectly "disentangled," the arithmetic might be "dirty":
    - Adding "sunglasses" might accidentally make the hair darker or change the skin tone
    - This is why researchers use β-VAEs, which put extra pressure on the KL term to ensure each dimension is as independent as possible

Disentanglement in VAEs

- **What is Entanglement?**
  - In a poorly trained or unconstrained model, features are "tangled" together
    - Moving along Dimension 1 might change the shape of an object and its colour simultaneously
    - This makes the model a "black box", you can generate data, but you can't control it precisely.
- **The Goal of Disentanglement**
  - We want a Factorized Representation. In a perfectly disentangled latent space:
    - z_1: Only controls Rotation.
    - z_2: Only controls Scale/Size.
    - z_3: Only controls Hue/Color.

Disentanglement in VAEs

- **How the VAE Encourages this?**
  - The secret lies in the Identity Matrix (I) in the KL Divergence term:
    - The prior $p(z) = \mathcal{N}(0, I)$ assumes that all dimensions are independent (zero correlation)
    - To minimize the KL Divergence, the Encoder is under constant pressure to find features in the data that are also independent
    - If the Encoder can find a way to separate "Face Shape" from "Hair Color," it can match the prior more efficiently and lower the loss

Applications of VAEs

- **Anomaly Detection (The "Surprise" Metric)**
  - VAEs are exceptional at finding the "odd one out" in complex datasets (e.g., credit card fraud, factory sensor failures)
  - How it works: Train a VAE only on "normal" data. When an anomaly appears, the model will struggle to reconstruct it because it doesn't fit the learned Gaussian distribution
- **Image Denoising and Inpainting**
  - Because the VAE maps data to a clean, structured latent space, it can "fix" corrupted inputs.
  - The Process: You feed a noisy or blurry image into the Encoder. The Encoder maps it to the nearest "clean" point in the latent manifold.
  - The Result: The Decoder outputs a version of the image that follows the rules of the training data, effectively "wiping away" the noise.

Applications of VAEs

- **Molecular Discovery & Chemical Design**
  - In drug discovery, scientists represent molecules as strings or graphs. VAEs can learn the "Grammar" of chemistry
    - Optimization: Once molecules are in the latent space, we can perform gradient-based optimization to find new vectors that maximize "Drug-likeness" or "Solubility"
    - Generation: The Decoder then translates these optimized vectors back into brand-new chemical structures that have never existed before
- **Data Compression and Dimensionality Reduction**
  - VAEs serve as a non-linear version of PCA (Principal Component Analysis)
    - They can compress massive datasets into tiny latent vectors while preserving the most important semantic features
    - This is used for efficient data storage and faster searching in large-scale image databases