XCS236 Deep Generative Models | Jonathan Hague (CF) | jhague@stanford.edu

Exploring Variational Auto Encoders and their variations with PyTorch implementation.

	Variational Auto Encoder (VAE)	Mixture of Gaussian VAE (GMVAE)	Importance Weight Auto Encoder (IWAE)
The Idea	The VAE models the data distribution p(x) and enables sample generation by optimizing the Evidence Lower Bound (ELBO) instead of the intractable logp(x). It uses an **encoder-decoder architecture** where the encoder outputs a Gaussian distribution qφ(z|x) over latent variables, enabling stochastic sampling via the reparameterization trick. A fixed prior p(z)=N(0,I) regularizes the latent space. During training, a single latent sample per data point is used to estimate the ELBO, allowing scalable optimization.	Uses a **Mixture of Gaussians** prior instead of a single Gaussian over the latent variable z. This enables the latent space to form clusters and discover subgroups, offering greater expressivity for modeling complex data distributions. The log probability density for each element in a multivariate normal distribution with independent dimensions is: Log p(x)=−1/2∑(log v_i+((x_i−m_i)^2)/vi+log (2π))	Uses **multiple latent samples** for a **tighter bound** on marginal likelihood than the ELBO by using multiple samples per input. This results in improved density estimation and more accurate modeling of the data distribution.
Prior p(*z*)	The VAE uses a **fixed standard Gaussian** as its latent prior: p(z)=N(0,I) This prior reflects the assumption that, in the absence of strong evidence from the data, latent representations z should remain close to the origin in a unit Gaussian space. The **likelihood model** (decoder) defines the conditional distribution over the input given the latent code: pθ(x|z)=Bernoulli(x;fθ(z)) where fθ(z) is the output of the decoder network. This formulation is suitable for binary data, where each pixel is treated as an independent Bernoulli random variable.	**Learnable Mixture of Gaussians:** $p(z) = \frac{1}{k}\sum_{i=1}^{k}\mathcal{N}(\mu_i, \sigma_i^2)$ Defined in gmvae.py under self.z_pre. The first k rows correspond to the means μi, and the next k rows to the variances σi^2. The initialization is scaled to prevent instability during training. The prior is parameterized as: $z_{\text{pre}} \sim \frac{1}{\sqrt{k \cdot \text{zdim}}} \cdot \mathcal{N}(0, I)$ **Likelihood model (decoder):** $p_\theta(x \mid z) = \text{Bernoulli}(x; f_\theta(z))$	A fixed standard Gaussian prior: $\boldsymbol{\mathcal{N}}(\mathbf{0}, \mathbf{I})$ Same as in the standard VAE, serving as a regularizing baseline for the latent space.

	Variational Auto Encoder (VAE)	Mixture of Gaussian VAE (GMVAE)	Importance Weight Auto Encoder (IWAE)
KL Term	The KL divergence in the VAE is **computed analytically in closed form** between two multivariate Gaussian distributions with diagonal covariances: DKL(qφ(z|x) ‖ p(z)) where qφ(z|x)=N(μ,diag(σ2)) is the approximate posterior output by the encoder, and p(z)=N(0,I) is the fixed standard normal prior. Because both distributions are Gaussian with tractable forms, the KL divergence has a closed-form expression and does not require sampling or approximation.	Computed against a mixture of Gaussians, making it non-trivial and requiring approximation via Monte Carlo sampling. Unlike the standard VAE, the KL divergence cannot be computed in closed form and is estimated by evaluating log q(z|x) − log p(z) using a single sample from the approximate posterior.	Approximated via **Monte Carlo sampling**. Multiple latent samples are drawn from q(z|x), and the KL term is estimated indirectly via the difference log q(z|x)−log p(z) within the importance weighting framework.
Inference Model (encoder output) $\mathbf{q_\phi(z\|x)}$	The encoder network approximates the true posterior p(z|x)p(z|x) with a diagonal Gaussian: qϕ(z|x)=N(z|μϕ(x),diag(σϕ2(x))) This form assumes conditional independence across latent dimensions and enables efficient sampling via the reparameterization trick. The approach is known as amortized inference, as the encoder learns a shared mapping from inputs to posterior parameters. Typically, one sample is drawn per data point during training.	A single latent sample is drawn per data point from the approximate posterior qφ(z|x) = N(μφ(x),diag(σφ^2(x))), output by the encoder network.	Draws multiple latent samples per data point from the approximate posterior qφ(z|x). Increasing the number of samples *m* improves the tightness of the bound and yields a closer approximation to the true log-likelihood than in standard VAEs.
Loss Function	The VAE minimizes the negative ELBO: −ELBO=Reconstruction Loss+KL Divergence The reconstruction loss measures how well pθ(x|z) matches the input, computed as binary cross-entropy for MNIST. The KL term regularizes qφ(z|x) toward the prior p(z)=N(0,I). A single latent sample per data point is drawn via the reparameterization trick to keep training efficient and differentiable.	Modifies **the prior distribution** in the ELBO objective: $L(x;\theta,\phi) = \mathbb{E}_{q_\phi(z\|x)}[\log p_\theta(x\|z)] -$ $D_{KL}(q_\phi(z\|x)\|\|p_\theta(z))$ where the prior $p_\theta(z)$ is a **Mixture of Gaussians** instead of a standard normal requiring KL divergence estimation via sampling.: $p_\theta(z) = \sum_{i=1}^{k} \frac{1}{k} \mathcal{N}(z\|\mu_i, \sigma_i^2)$	Refines ELBO by using multiple latent samples $\{z^{(i)}\}_{i=1}^m$: IWAE Objective: $\mathcal{L}_m(x;\theta,\phi) = \mathbb{E}_{z^{(1)},\ldots,z^{(m)}\sim q_\phi(z\|x)}\left[\log\left(\frac{1}{m}\sum_{i=1}^{m}\frac{p_\theta(x,z^{(i)})}{q_\phi(z^{(i)}\|x)}\right)\right]$ The ELBO is a loose lower bound on log p(x) when the approximate posterior q(z|x) diverges significantly from the true posterior. IWAE tightens this bound by averaging over mm samples from q(z|x). As m increases, Lm approaches the true marginal log-likelihood log p(x). When m=1, IWAE reduces to the standard ELBO.

	Variational Auto Encoder (VAE)	Mixture of Gaussian VAE (GMVAE)	Importance Weight Auto Encoder (IWAE)
Code Hints	**Use the reparameterization trick** to enable gradient-based optimization through stochastic sampling: sample $\epsilon \sim N(0,I)$ and compute $z=\mu+\sigma\cdot\epsilon$ This ensures that the sampling operation is differentiable during backpropagation. **Compute the KL divergence analytically** between the approximate posterior $q\varphi(z\|x)=N(\mu,\text{diag}(\sigma 2))$ and the prior $p(z)=N(0,I)$. The KL is computed element-wise and summed across the latent dimensions. The **prior $p(z)\sim N(0,I)$** is hard-coded as non-trainable parameters in the VAE.__init__() method. This makes it convenient to reuse the same prior across all ELBO calculations without redefining it. **Disentanglement:** scaling factor β>on the KL term (as in β-VAE). Increasing β forces the approximate posterior $q\varphi(z\|x)$ to more closely align with the prior $p(z)$, promoting better-disentangled latent representations at the cost of reconstruction quality.	**Mixture of Gaussians Prior Initialization:** The prior encodes both means and variances for the k mixture components in self.z_pre, which is then split into m_mixture and v_mixture for use in log probability calculations. **Posterior Sampling and Decoder Pass:** Start by passing x through the encoder to get the approximate posterior parameters m and v. Sample $z\sim q(z\|x)$, then feed z into the decoder to obtain logits. Use these to compute the reconstruction term and estimate the KL divergence as: Log $q(z\|x)$ – $\log p(z)$ **Negative ELBO and Batch Averaging:** Compute the negative ELBO by adding the reconstruction and KL terms. Take the mean over the batch to make the loss independent of batch size. **Mixture Model KL Evaluation:** When computing $\log p(z)$ under the mixture prior, expand z with .unsqueeze()to broadcast against all k mixture components. This ensures the mixture density is evaluated for each sample.	**Sampling Expansion:** IWAE draws multiple samples per data point (iw=m), so we expand the mean, variance, and input x to match the number of samples. The helper function ut.duplicate() creates iw copies of the input tensor along the batch dimension, effectively scaling up the input size for sampling and decoding. **Multiple-Sample Estimation:** Instead of estimating the expectation with a single sample, IWAE uses multiple samples from $q(z\|x)$. The **log-mean-exp trick** is applied to maintain numerical stability when computing the average of importance-weighted terms. **Jensen's Inequality and Bound Tightness:** We use the Jensen's inequality to pull the log inside the expectation. Log $p\theta(x) \geq Lm(x) \geq L1(x)$ where: Log $p\theta(x)$ is the **true marginal log-likelihood**. $\mathcal{L}m(x)$ is the **IWAE bound with m samples**. $L1(x)$ is the **standard ELBO (when m=1)**. **Jensen's Inequality** states that for a convex function f(x), we have: $f(E[X]) \leq E[f(X)]$ Since log is a **concave function**, we can **move it inside the expectation**. This step shows that **IWAE ($\mathcal{L}m(x)$) is always a lower bound on the true log-likelihood**.

	Semi-Supervised VAE (SSVAE)	Fully-Supervised VAE (FSVAE)
The Idea	The SSVAE extends the VAE for **semi-supervised classification**. It models both: A **generative process** p(x,y,z) And a **discriminative process** through the approximate posterior qφ(y,z|x)= qφ(y|x)qφ(z|x,y) This allows training on **labeled data** (x,y) ∈ Xℓ and **unlabeled data** x ∈ Xu.	FSVAE explicitly separates **content** (label y) and **style** (latent variable z) in a conditional generative model. This model is a **Conditional VAE** where the generation of the observed variable x depends on two latent factors: y: observed label (digit class: the **content**) z: unobserved latent variable (representing **style** or other intra-class variations) FSVAE is useful for **style-content disentanglement**, especially in the SVHN dataset where digits have varying visual styles. Basically, y controls **what digit** is generated and z controls **how it looks.**
Prior p(*z*)	Fixed standard gaussian as the latent prior: $p(z) = \mathcal{N}(z \mid 0, I)$ $p(y) = \text{Categorical}(y \mid \pi), \quad \pi = \frac{1}{10} \text{ for MNIST classes}$ $p(x \mid y, z) = \text{Bernoulli}(x \mid f_\theta(y, z))$	Prior over latent style variable z: a standard multivariate normal. $p(z) = \mathcal{N}(z \mid 0, I)$ Our distribution captures the idea that, independent of any particular image or class, style vectors are expected to lie near the origin in latent space. Uniformed prior over style. Conditional likelihood of image given label and style: $p(x \mid y, z) = \mathcal{N}(x \mid \mu_\theta(y, z), \frac{1}{10} I)$ The image x is drawn from a multivariate Gaussian with: Mean μθ(y,z): output of the decoder Covariance fixed to 1/10 I: simplifies training and likelihood computation The decoder takes both y and z as input to generate the mean image. <u>This means that:</u> The **class label** y determines the **content** (which digit it is). The **latent variable** z controls the **style** (font thickness, slant, background).
KL Term	The KL divergence to this prior acts as a **regularizer**: it prevents the encoder from pushing q(z|x,y) arbitrarily far from a centered, normalized latent space. Computed **analytically**. There are two KL divergence terms in SSVAE: KL Divergence for the Latent Variable z (for labeled data): it Measures how far the encoder's posterior over latent space is from the standard normal prior. KL Divergence for the Label Variable y (for unlabeled data): p(y) is a uniform categorical over digits: 1/10. This KL penalizes the classifier if it becomes too confident or too unbalanced across classes.	Computed between the approximate posterior qφ(z|x,y)=N(μφ,σφ2) and the standard normal prior p(z)=N(0,I). Since both distributions are Gaussians with diagonal covariances, the KL term admits a closed-form solution and is implemented analytically. This allows efficient and exact computation of the KL divergence without sampling.

	Semi-Supervised VAE (SSVAE)	Fully-Supervised VAE (FSVAE)
Inference Model (encoder output) $\mathbf{q_\phi(z\|x)}$	The encoder is multi-headed: qφ(y|x) Label classifier: A categorical distribution over the digit classes. (implemented by passing logits on softmax). qφ(z|x,y) Latent posterior: A Gaussian distribution whose parameters depend on both x and y. μ(x,y), σ2(x,y) are output by the encoder network.	Remember, since the true posterior p(z|x,y) is intractable, we use a neural network that takes x and y as input to approximate it: $q_\phi(z \mid x, y) = \mathcal{N}(z \mid \mu_\phi(x, y), \mathrm{diag}(\sigma_\phi^2(x, y)))$ Where: μφ(x,y): predicted mean of the latent style vector given the image and label σφ2(x,y): predicted diagonal variance The encoder '**inverts**' the generation: given an image and its class label, it infers the style. We **amortized inference**: the same network is used for all x,y pairs, and learns to generalize from data. During training, we sample z ∼ qφ(z|x,y) using the **reparameterization trick**: z=μ+σ⊙ϵ, ϵ∼N(0,I) Basically, our **encoder infers** style given image + label.
Loss Function	$\max_{\theta,\phi} \sum_{x\in\mathcal{X}} \mathrm{ELBO}(x;\theta,\phi) + \alpha \sum_{(x,y)\in\mathcal{X}_\ell} \log q_\phi(y \mid x)$ Where: For unlabeled data: $\mathrm{ELBO}(x;\theta,\phi) = \mathbb{E}_{q(y\|x)}\mathrm{ELBO}(x,y;\theta,\phi)$ Implemented by repeating each x across 10 one-hot label values, then Calculating q(z|x,y), sampling z, computing reconstruction loss and both KL terms, and taking expectation via weighting by q(y|x). For labeled data: $\mathrm{ELBO}(x,y;\theta,\phi) = \mathbb{E}_{q(z\|x,y)}[\log p(x \mid y,z)] - D_{\mathrm{KL}}(q(z\|x,y)\\|p(z))$ No KL term for y since it's known. Classification loss: $\log q(y \mid x)$	This is the **standard ELBO** for conditional VAEs. Because this is a **fully supervised** setup with known y, we compute: $\log p(x \mid y) \geq \mathbb{E}_{q_\phi(z\|x,y)}[\log p_\theta(x \mid y, z)] - \mathrm{KL}(q_\phi(z\|x,y) \\| p(z))$

	Semi-Supervised VAE (SSVAE)	Fully-Supervised VAE (FSVAE)
	Encourages the classifier (the encoder's categorical head) to predict the correct label for the labeled samples.	
Code Hints	For unlabeled data, you'll marginalize over the 10 possible labels. Use ut.duplicate(x, 10) to tile each image across all 10 label possibilities. For labeled data, use known one-hot labels. Do not marginalize over y.	Once trained, we Fix each **label** y (rows: 0–9), and sample **20 random** z vectors (columns: 0–19) to generate x~p(x|y,z) by decoding from each pair. We Use torch.clip() on the is the mean of the Gaussian distribution used to model the pixels of the reconstructed image to prevent pixel overflow. We clip, because the neural networks are free to output any real value, but our reconstructed image pixels are modeled as real values **between 0 and 1**, because the true images are normalized pixel intensities. So, we **clip** the values to valid pixel range 0 and 1.