

CONTENTS

1	Lectures	2
1.1	Representation	3
1.2	Autoregressive Models	4
1.3	Variational Autoencoders	7
1.3.1	Homework 2	12
1.4	Normalizing Flow Models	16
1.4.1	Homework 3	22
1.5	Generative Adversarial Networks	23
1.6	Energy-Based Models I	29
2	Concepts	31
2.1	Evidence Lower Bound (ELBo)	32

LECTURES

CONTENTS

1.1	Representation	3
1.2	Autoregressive Models	4
1.3	Variational Autoencoders	7
1.3.1	Homework 2	12
1.4	Normalizing Flow Models	16
1.4.1	Homework 3	22
1.5	Generative Adversarial Networks	23
1.6	Energy-Based Models I	29

Representation

Learning a Generative Model. We are given a set of examples $\{x\}$. Want to learn $p(x)$ such that

- **Generation:** $x \sim p(x)$ should look like an example from the data.
- **Density estimation:** $p(x)$ should be high if x looks like the data, and low otherwise (*anomaly detection*).
- **Unsupervised representation learning.** Learn what the examples have in common (features).

Structure through Conditional Independence [27:30]

1. How many parameters to specify joint distribution $p(x_1, \dots, x_n)$ with the chain rule?
2. Now suppose $X_{i+1} \perp X_1, \dots, X_{i-1} \mid X_i$

Answers:

1. $1 + 2 + \dots + 2^{n-1} = 2^n - 1$ (duh, chain rule is fully general).
2. $2n - 1$

Bayes Networks [34:00]. Use conditional parameterization (instead of joint). For each RV X_i specify $p(x_i \mid \mathbf{x}_{A_i})$ for set \mathbf{X}_{A_i} of RVs. The model joint as

$$p(x_1, \dots, x_n) = \prod_i p(x_i \mid \mathbf{x}_{A_i}) \quad (1)$$

A **Bayesian network** is a DAG $G = (V, E)$. RVs are nodes, edges specify conditional dependencies. **Economical representation:** we are now exponential in $|Pa(i)|$ (instead of $|V|$)¹.

Stopped at [46:00]. Lecture seems to be transitioning from CPDs to generative models. Watched the rest on the bus, nothing too informative.

¹In general, you are “exponential in” the number of RVs appearing in a given CPD

Autoregressive Models

Table of Contents Local

Written by Brandon McKinzie

Autoregressive Models [23:00]. AR models assume an ordering of the modeling variables x_1, \dots, x_n , and assume that $X_i \mid X_1, \dots, X_{i-1}$ can be modeled as a parameterized function of X_1, \dots, X_{i-1} with number of params $\mathcal{O}(i-1)$. For example, consider the problem where each x_i is binary bernoulli RV. We could model the joint distribution via the chain rule with the following conditionals:

$$p_{CPT}(X_1=1; \alpha^{(1)}) = \alpha^{(1)} \quad (2)$$

$$p_{logit}(X_2=1 \mid x_1; \alpha^{(2)}) = \sigma(\alpha_0^{(2)} + \alpha_1^{(2)} x_1) \quad (3)$$

$$p_{logit}(X_n=1 \mid x_1, \dots, x_{n-1}; \alpha^{(n)}) = \sigma(\alpha_0^{(n)} + \sum_{i=1}^{n-1} \alpha_i^{(n)} x_i) \quad (4)$$

which has $\mathcal{O}(n^2)$ parameters.

AR Models vs. Autoencoders (AE) [1:02:00]. AEs consist of two stages:

1. **Encoder**: maps inputs $\mathbf{x} = x_1, \dots, x_n$ to some hidden representation $e(\mathbf{x})$.
2. **Decoder**: function d such that $d(e(\mathbf{x})) \approx \mathbf{x}$.

A vanilla AE is *not* a generative model: it doesn't define some $p(x)$ we can sample from².

²Why? Because it doesn't specify an *ordering* of the variables.

MADE (Masked Autoencoder for Distribution Estimation) is an AE architecture that is autoregressive. It defines an ordering of the variable via masking. Consider a 3-variable model over x_1, x_2, x_3 with ordering x_2, x_3, x_1 :

- The unit producing the parameters for $p(x_2)$ cannot depend on any inputs.
- Any weights leading to $p(x_3 | x_2)$ can only be traceable back to x_2 .
- Any weights leading to $p(x_1 | x_2, x_3)$ can only be traceable back to x_2 or x_3 .

To accomplish this, we perform the following masking operations:

- Assign each of the n input units a **degree** i that defines their index in the ordering. For our working example, assume we got degrees $\{x_1=3, x_2=1, x_3=2\}$.
- This defines the output conditionals of the model as

$$p(x_1 | x_2, x_3) \quad p(x_2) \quad p(x_3 | x_2)$$

- For each unit in a hidden layer, pick a random integer i in $[1, n-1]^3$. That unit only has connections from units j (in the previous layer) that were assigned a number less than or equal to i . Conceptually, a hidden unit with degree i can depend on inputs in the (inclusive) range $[1..i]$.

Learning Setting [48:30]. Given dataset \mathcal{D} of m IID samples from P_{data} . Also given some family of models \mathcal{M}^4 , and our task is to learn some “good” model $\hat{\mathcal{M}} \in \mathcal{M}$.

The **KL-divergence** between two distributions p and q is defined as

$$D_{KL}(p||q) = \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} \quad (5)$$

$$0 \log \frac{1}{0} = 0$$

- (Gibb’s inequality) $D(p||q) \geq 0 \forall p, q$ with equality iff $p=q$. **Proof**⁵:

$$\mathbb{E}_{x \sim p} \left[-\log \frac{q(x)}{p(x)} \right] \geq -\log \left(\mathbb{E}_{x \sim p} \left[\frac{q(x)}{p(x)} \right] \right) = -\log \left(\sum_x p(x) \frac{q(x)}{p(x)} \right) = 0 \quad (6)$$

- Asymmetry: $D(p||q) \neq D(q||p)$.
- Measures the expected number of extra bits required to describe samples from $p(x)$ using a code based on q instead of p ⁶.

³ $n-1$ (not n) because none of the outputs (and thus hidden layers, too) can be conditioned on the final variable (given by index n).

⁴Examples of model families \mathcal{M} are things like (a) all Bayes nets with a specific structure, or (b) all neural networks with a given architecture (our goal is to search for the correct params in that space).

⁵This proof relies on Jensen’s Inequality, which says that for any convex function $f(x)$: $f(\mathbb{E}[x]) \leq \mathbb{E}[f(x)]$. See my exercises from PGM chapter 2 for related proofs.

⁶Mathematically, you can indeed see that:

$$D(p||q) = \mathbb{E}_p \left[\log \frac{1}{q(x)} \right] - \mathbb{E}_p \left[\log \frac{1}{p(x)} \right] \quad (7)$$

In terms of parameter optimization with some model $q(x) := P_\theta(x)$, minimizing $D(P_{data}||P_\theta)$ is equivalent to maximizing the expected log-likelihood, $\log P_\theta(x)$ [1:15:00].

Monte Carlo Estimation [1:21:16].

1. Express quantity of interest as the expected value of a random variable:

$$\mathbb{E}_{x \sim P(x)} [g(x)] = \sum_x g(x)P(x) \quad (8)$$

2. Generate T samples $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}$ from P .
3. Estimate the expected value using:

$$\hat{g} \triangleq \frac{1}{T} \sum_{t=1}^T g(\mathbf{x}^{(t)}) \quad (9)$$

Properties of the MC estimate \hat{g} :

- **Unbiased:** $\mathbb{E}_P [\hat{g}] = \mathbb{E}_P [g(x)]$
- **Convergence:** $\hat{g} \rightarrow \mathbb{E}_P [g(x)]$ as $T \rightarrow \infty$.
- **Variance:** $\text{Var} [\hat{g}] = \frac{1}{T} \text{Var} [g(x)]$

Variational Autoencoders

Mixture of Gaussians [55:40]. A shallow LVM.

$$\mathbf{z} \sim \text{Categorical}(1, \dots, K) \quad (10)$$

$$p(\mathbf{x} \mid \mathbf{z}=k) = \mathcal{N}(\mu_k, \Sigma_k) \quad (11)$$

which also provides the generative process: sample a component $k \sim p(\mathbf{z})$, then generate data point \mathbf{x} by sampling from $p(\mathbf{x} \mid \mathbf{z}=k)$. We can also do clustering using the posterior $p(\mathbf{z} \mid \mathbf{x})$. Another interpretation/motivation for LVMs is that we can *combine simple models into a more complex and expressive one*. E.g. MoG results in a complex distribution $p(\mathbf{x})$ even though it is combining very simple Gaussians components:

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{z}) p(\mathbf{x} \mid \mathbf{z}) = \sum_{k=1}^K p(\mathbf{z}=k) \underbrace{\mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)}_{\text{component}} \quad (12)$$

Variational Autoencoder (VAE) [1:09:15]: a mixture of an infinite number of Gaussians.

$$\mathbf{z} \sim \mathcal{N}(0, I) \quad (13)$$

$$p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mu_{\theta}(\mathbf{z}), \Sigma_{\theta}(\mathbf{z})) \quad (14)$$

where $\mu_{\theta}, \Sigma_{\theta}$ are neural networks.

Need way of approximating $\nabla \log \sum_s p(\mathbf{x}, \mathbf{z}; \theta)$.

Naive Monte Carlo [13:50]. As usual, we can express $p_{\theta}(\mathbf{x})$ as

$$p_{\theta}(\mathbf{x}) = \sum_{\mathbf{z}} p_{\theta}(\mathbf{x}, \mathbf{z}) = |\mathcal{Z}| \mathbb{E}_{\mathbf{z} \sim U(\mathcal{Z})} [p_{\theta}(\mathbf{x}, \mathbf{z})] \quad (15)$$

where \mathcal{Z} is the set of all possible values that \mathbf{z} can take. **NB:** the expectation is over (uniform) $U(\mathcal{Z})$, NOT $p(\mathbf{z})$ as we often see⁷; hence the scaling factor of $|\mathcal{Z}|$. The naive MC procedure is then to perform:

1. Sample $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(k)}$ uniformly at random.

⁷The form we typically write is $\mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} [p(\mathbf{x} \mid \mathbf{z})]$, an equivalent expression.

2. Approximate expectation with sample average:

$$\sum_z p_\theta(x, z) \approx |\mathcal{Z}| \frac{1}{k} \sum_{j=1}^k p_\theta(x, z^{(j)}) \quad (16)$$

Problem: for most values of z , $p_\theta(x, z)$ is very low. Results in very high estimator variance.

Importance Sampling [19:20]. Sample instead from a distribution $q(z)$ instead of $U(\mathcal{Z})$.

$$p_\theta(x) = \sum_{z \in \mathcal{Z}} \frac{q(z)}{q(z)} p_\theta(x, z) = \mathbb{E}_{z \sim q(z)} \left[\frac{p_\theta(x, z)}{q(z)} \right] \quad (17)$$

$$p_\theta(x) \approx \frac{1}{k} \sum_{j=1}^k \frac{p_\theta(x, z)}{q(z)} \quad (18)$$

Want to choose $q(z)$ that assigns high probabilities to $p_\theta(x, z)$, since (a) those terms contribute most to the true expectation, and (b) it reduces the variance of our estimator. **Problem:** we want to work with $\log p_\theta(x) = \log \mathbb{E}_z [p/q]$ as usual, but this process would *actually* result in us working with $\mathbb{E}_z [\log p/q]$ which is not equivalent⁸

Evidence Lower Bound [29:00]. Idea: use Jensen's inequality to get a lower bound on $\log p_\theta(x)$:

$$\log \left(\mathbb{E}_{z \sim q(z)} \left[\frac{p_\theta(x, z)}{q(z)} \right] \right) \geq \mathbb{E}_{z \sim q(z)} \left[\log \frac{p_\theta(x, z)}{q(z)} \right] \quad (19)$$

$$= \sum_z q(z) \log p_\theta(x, z) - \sum_z q(z) \log q(z) \quad (20)$$

$$= \sum_z q(z) \log p_\theta(x, z) + H(q) \quad (21)$$

$$\triangleq \mathcal{L}_\theta(x) \quad (22)$$

which is called the **evidence lower bound (ELBo)**⁹. Some properties/facts:

- Equality holds (trivially) if you can remove the z dependence of $p(x, z)/q(z)$. This occurs if $q(z) = p_\theta(z | x)$.
- $\log p_\theta(x) = \text{ELBo} + D_{KL}(q(z) || p(z | x))$

Variational inference: learn a parameterized function $q_\phi(z)$ such that it's as close as possible to $p_\theta(z | x)$.

⁸**TODO:** As everyone in the class has asked: *why can't we just sample from p directly and then compute the log after??*

⁹This can also be derived via $D_{KL}(q(z) || p_\theta(z | x))$

Learning Deep Generative Models. We can replace $\log p$, when computing maximum likelihood, with \mathcal{L} as follows:

$$\ell(\theta; \mathcal{D}) = \sum_{x^{(i)} \in \mathcal{D}} \log p_{\theta}(x^{(i)}) \geq \sum_{x^{(i)} \in \mathcal{D}} \mathcal{L}(x^{(i)}; \theta, \phi^{(i)}) \quad (23)$$

$$\max_{\theta} \ell(\theta; \mathcal{D}) \geq \max_{\theta, \phi^{(1)}, \dots, \phi^{(M)}} \sum_{x^{(i)} \in \mathcal{D}} \mathcal{L}(x^{(i)}; \theta, \phi^{(i)}) \quad (24)$$

We use different **variational parameters** $\phi^{(i)}$ for each data point $x^{(i)}$ because the true posterior $p_{\theta}(z \mid x^{(i)})$ is different for each $x^{(i)}$. We can optimize via **stochastic variational inference** (SVI), which is just performing SGD on \mathcal{L} .

Stochastic Variational Inference

1. Initialize $\theta, \phi^1, \dots, \phi^M$.
2. Randomly sample $\mathbf{x}^{(i)}$ from \mathcal{D} .
3. Optimize $\mathcal{L}(\mathbf{x}^{(i)}, \theta, \phi^i)$ as a function of ϕ^i :
 - (a) Repeat $\phi^i = \phi^i + \eta \nabla_{\phi^i} \mathcal{L}(\mathbf{x}^{(i)}; \theta, \phi^i)$
 - (b) Until convergence to $\phi^{i*} \approx \arg \max_{\phi} \mathcal{L}(\mathbf{x}^{(i)}; \theta, \phi)$
4. Update θ using $\nabla_{\theta} \mathcal{L}(\mathbf{x}^{(i)}; \theta, \phi^{i*})$

Computing $\nabla_{\theta} \mathcal{L}$ is easy: do MC sampling and compute via backprop as usual. To compute $\nabla_{\phi^i} \mathcal{L}$, we can't just MC sample because the sampling procedure itself depends on $q_{\phi^i}(z)$. One approach is a general technique called **REINFORCE** [1:18:00].

Reparameterization. A better but less general alternative to REINFORCE that only works for [some] continuous z . Again, our goal is to compute the gradient wrt ϕ of

$$\mathbb{E}_{q_{\phi}(z)} [r(z)] = \int q_{\phi}(z) r(z) dz \quad (25)$$

Suppose $q_{\phi}(z) = \mathcal{N}(\mu, \sigma^2 I)$. We can then do:

$$\epsilon \sim \mathcal{N}(0, I) \quad (26)$$

$$z = \mu + \sigma \epsilon = g(\epsilon; \phi) \quad (27)$$

$$\mathbb{E}_{z \sim q_{\phi}(z)} [r(z)] = \mathbb{E}_{\epsilon \sim \mathcal{N}(0, I)} [r(g(\epsilon; \phi))] \quad (28)$$

The primary result here is that we've *reparameterized* z to be a function of auxiliary variable ϵ such that **we can push the gradient inside the expectation**:

$$\nabla_{\phi} \mathbb{E}_{q_{\phi}} [r(z)] = \nabla_{\phi} \mathbb{E}_{\epsilon} [r(g(\epsilon; \phi))] = \mathbb{E}_{\epsilon} [\nabla_{\phi} r(g(\epsilon; \phi))] \quad (29)$$

which we can estimate via Monte Carlo if r and g are differentiable wrt ϕ , and if ϵ is easy to sample from (backpropagation). Typically much lower variance than REINFORCE.

Amortized Inference [16:00]. Having a different ϕ^i for each example x^i doesn't scale to larger datasets. **Amortization**¹⁰: learn a single parametric function f_λ that maps each x to a set of (good) variational parameters. Like doing regression on $x^i \mapsto \phi^{i,*}$. We approximate the posteriors $q(z | x^i)$ using $q_\lambda(z | x)$.

Autoencoder Perspective [26:00]. We can rewrite our variational objective \mathcal{L} as follows:

$$\mathcal{L}(x; \theta, \phi) = \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x, z) - \log q_\phi(z | x)] \quad (30)$$

$$= \mathbb{E}_{q_\phi(z|x)} [(\log p_\theta(x, z) - \log p(z)) - (-\log p(z) + \log q_\phi(z | x))] \quad (31)$$

$$= \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x | z)] - D_{KL}(q_\phi(z | x) || p(z)) \quad (32)$$

Interpretation Break. Remember, our overall goal here is to have some way of sampling $x \sim p(x)$. The steps we've covered in this class so far can be summarized as follows:

1. *LVMs and difficulties of sampling.* We are also using latent variables z ¹¹, so that means we need to figure out how to sample $x \sim \sum_z p(x, z) = \mathbb{E}_{z \sim p(z)} [p(x | z)]$ (challenging).
2. *Importance/MC Sampling.* We can equivalently write $p_\theta(x) = \mathbb{E}_{z \sim q(z)} [p_\theta(x, z)/q(z)]$ for any distribution $q(z)$. We can approximate this expectation with a MC estimator $\hat{p} = \frac{1}{k} \sum_{j=1}^k p_\theta(x, z)/q(z)$.
3. *Variational objective (ELBo).* **For reasons unknown**, we only are able to sample $\log(p/q)$ (not from p/q directly). This leads to the variational objective, a.k.a. the ELBo:

$$\mathcal{L}_\theta(x) = \sum_z q(z) \log p_\theta(x, z) + H(q) \quad (33)$$

which is guaranteed to be less than or equal to $\log \mathbb{E} [p/q]$. Therefore, we want to maximize this quantity.

4. *Variational Inference.* \mathcal{L} is maximized when $q(z) = p_\theta(z | x)$. The goal of VI is to learn a parameterized function $q_\phi(z)$ as close as possible to $p_\theta(z | x)$. The challenge is that $p_\theta(z | x)$ is a function of x – $p_\theta(z | x^i)$ is different for each x^i . Therefore, we use different **variational parameters** ϕ^i for each x^i .
5. *Stochastic Variational Inference.* One method for learning the parameters $\{\theta, \{\phi^i\}\}$ is **SVI**. Unfortunately, in order to approximate $\nabla_{\phi^i} \mathcal{L}(x^i)$, we first need to sample from $q_{\phi^i}(z)$, which itself is a function of ϕ^i .
6. *Reparameterization & Amortized Inference.* **Reparameterization** allows us to push the gradient of ϕ inside the expectation, which means we can *first* draw our samples, and then compute gradients of those samples via e.g. backprop. Also, instead of having a different ϕ^i for every data point x^i , we can use **amortized inference**: learn a single parametric function $q_\lambda(z | x)$.

¹⁰Called “amortized” because, by removing the constraint that every single x^i must have it's own ϕ^i , we remove the need to train the ϕ^i when trying to do inference at test time on new x^i . Although learning a generic f_λ is *more* challenging than learning a separate ϕ^i for each x^i , those costs are “paid off” at test time, since we no longer have to learn parameters every time we want to do inference.

¹¹Recall that LVMs are useful for generating samples and for combining simple models into a more complex/expressive one.

Now, let's see how we can use this knowledge to outline the algorithm for a VAE.

VAE

1. Take a data point x^i .
2. **Encoder:** Map it to \hat{z} by MC sampling from $q_\phi(z | x^i)$
3. **Decoder:** Reconstruct \hat{x} by MC sampling from $p_\theta(x | \hat{z})$

The reason we are doing MC samples from the encoder (instead of just single top-1 sampling) is because we are eventually going to use these to evaluate \mathcal{L} . They represent the approximation, for example, to $\mathbb{E}_{q_\phi(z|x)} [\log p_\theta(z | x)]$. **Ok but why MC sampling from decoder? The first time in \mathcal{L}** implies we should simply *evaluate* $\log p(x | z)$ for each sample of z from the encoding step.

Analysis of \mathcal{L} (from VAE perspective).

- $\mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x | z)]$ encourages $\hat{x} \approx x^i$ (x^i likely under $p_\theta(x | \hat{z})$).
- $D_{KL}(q_\phi(z | x) || p(z))$ encourages \hat{z} to be likely under $p(z)$.
- Our approximations work well if $\mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x | z)]$ is *large*. The largest this expectation can get corresponds to how well large values of $q_\phi(z | x)$ align with large values of $p_\theta(x | z)$. If they *do* align well, then our samples from $z \sim q$ (which will naturally tend to be values of \hat{z} for which $q_\phi(z | x)$ is large) will correspond to the large values of $p_\theta(x | \hat{z})$.
- As we learn, $D_{KL}(q_\phi(z | x) || p(z))$ encourages $q_\phi(z | x)$ (which remember is a function of x !) to be shaped like the *prior* $p(z)$ (independent of x). This is useful because if we know $p(z)$ and don't have access to data points x in the future, we can still generate reasonable values of \hat{x} by sampling $\hat{z} \sim p(z)$ instead!

Some good paper references shown in the slides around [42:00].

1.3.1 HOMEWORK 2

VAEs utilize the following form of $\mathcal{L}(x; \theta, \phi)$:

$$\mathcal{L}(x; \theta, \phi) = \mathbb{E}_{z \sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x) || p(z)) \quad (34)$$

We can interpret $q_\phi(z|x)$ as an *encoder* and $p_\theta(x|z)$ as a *decoder*.

$$p_\theta(z) := \mathcal{N}(z; 0, I) \quad (35)$$

$$\log q_\phi(z|x^{(i)}) := \log \mathcal{N}(z; \mu^{(i)}, \sigma^2(i)I) \quad (36)$$

Problem 1: Implementing the VAE. Setting:

- Task: learn probabilistic model of MNIST.
- Observed variables: $\mathbf{x} \in \{0, 1\}^d$ (binary pixel sequence)
- Latent variables $\mathbf{z} \in \mathbb{R}^k$.
- Goal: Learn LVM $p_\theta(\mathbf{x})$ of the high-dimensional data distribution $p_{data}(\mathbf{x})$.

Setup

The VAE for this problem is defined by the generative process

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z} | 0, I) \quad (37)$$

$$\text{[decoder]} \quad p_\theta(\mathbf{x} | \mathbf{z}) = \text{Bern}(\mathbf{x} | f_\theta(\mathbf{z})) \quad (38)$$

Model and Sampling

where \mathbf{z} has dimension $\mathbf{z_dim}$ ¹². The function $f_\theta(\mathbf{z})$ is parameterized by a neural network with weights θ . It outputs logits for each of the d values associated with \mathbf{x} . We can apply a sigmoid to obtain the bernoulli probabilities of each value being 1.

Although sampling is easy, we need to train our parameters first. Computing $p_\theta(\mathbf{x})$ (needed for MLE) is intractable:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} \quad (39)$$

Intractability of $p(\mathbf{x})$

since this requires integrating over all of $\mathbb{R}^{z_{dim}}$. In MLE, we need to compute $\log p_\theta(\mathbf{x})$. Therefore, we need some approximation of this value that's tractable to compute.

We know that, for any distribution $q(\mathbf{z})$, we can obtain a lower bound on $\log p_\theta(\mathbf{x})$ with

$$\mathcal{L}(x; \theta, \phi) \triangleq \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z})} \left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right] \quad (40)$$

¹²Don't forget to sqrt the variance when computing \mathbf{z} with the reparameterization trick!

Ok, so how do we implement $q(\mathbf{z})$? Note that, for a given \mathbf{z} , the optimal choice for q is $p(\mathbf{z} | \mathbf{x})$ ¹³. VAEs exploit this via an **encoder** $q_\phi(\mathbf{z} | \mathbf{x})$ parameterized by ϕ .

$$\text{[encoder]} \quad q_\phi(\mathbf{z} | \mathbf{x}) = \mathcal{N}(\mathbf{x}; \mu_\phi(\mathbf{x}), \text{diag}(\sigma_\phi^2(\mathbf{x}))) \quad (41)$$

Of the many ways to write $\mathcal{L}(x; \theta, \phi)$, the formula useful for our VAE here will be

$$\mathcal{L}(x; \theta, \phi) = \mathbb{E}_{q_\phi(\mathbf{z} | \mathbf{x})} [\log p_\theta(\mathbf{x} | \mathbf{z})] - D_{KL}(q_\phi(\mathbf{z} | \mathbf{x}) || p(\mathbf{z})) \quad (42)$$

consisting of the **reconstruction loss** and **KL** terms. So how do we actually *implement* this? Well, one hacky way (and the way we do in the homework) for implementing the reconstruction loss is to take a single \mathbf{z} MC sample estimate instead of dealing with the expectation. In other words, we literally compute $\log p_\theta(\mathbf{x} | \mathbf{z}^{(1)})$ where $\mathbf{z}^{(1)}$ is our single sample from $q_\phi(\mathbf{z} | \mathbf{x})$. Apparently the KL term can be computed analytically if $q_\phi(\mathbf{z} | \text{vec } \mathbf{x})$ and $p(\mathbf{z})$ are both Gaussian:

$$D_{KL}(q_\phi(\mathbf{z} | \mathbf{x}) || p(\mathbf{z})) = \frac{1}{2} \left[\log \frac{\sigma_p^2}{\sigma_q^2} + \frac{\sigma_p^2}{\sigma_q^2} + \frac{(\mu_q - \mu_p)^2}{\sigma_p^2} - 1 \right] \quad (43)$$

Problem 2: GMVAE. Mixture of Gaussians VAE. Instead of the parameter-free isotropic Gaussian $p(\mathbf{z})$ from problem 1, we now have:

$$p_\theta(\mathbf{z}) = \sum_{i=1}^k \frac{1}{k} \mathcal{N}(\mathbf{z}; \mu_i, \text{diag}(\sigma_i^2)) \quad (44)$$

One challenge is that the KL term can't be computed analytically between a Gaussian q and *mixture* of Gaussians p . In the homework, we use the unbiased MC estimator with a single sample $\mathbf{z}^{(1)}$:

$$D_{KL}(q_\phi(\mathbf{z} | \mathbf{x}) || p(\mathbf{z})) \approx \log q_\phi(\mathbf{z}^{(1)} | \mathbf{x}) - \log p_\theta(\mathbf{z}^{(1)}) \quad (45)$$

$$= \log \mathcal{N}(\mathbf{z}^{(1)}; \mu_\phi(\mathbf{x}), \text{diag}(\sigma_\phi^2(\mathbf{x}))) - \log \sum_{i=1}^k \frac{1}{k} \mathcal{N}(\mathbf{z}^{(1)}; \mu_i, \text{diag}(\sigma_i^2)) \quad (46)$$

¹³Why? because it removes the dependence on \mathbf{z} from the fraction, thus making the lower bound equal to $\log \mathbb{E}[p/q]$

Problem 3: Importance Weighted Autoencoder (IWAE). Note that we can rewrite the ELBo in the form

$$\mathcal{L}(x; \theta, \phi) = \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z} | \mathbf{x})} \left[\log \frac{p_\theta(\mathbf{z} | \mathbf{x})}{q_\phi(\mathbf{z} | \mathbf{x})} \cdot p_\theta(\mathbf{x}) \right] \quad (47)$$

The term in brackets is *unnormalized*¹⁴. We can obtain a tighter [lower] bound by averaging $m > 1$ samples from the approximate posterior $q_\phi(\mathbf{z} | \mathbf{x})$.

$$\mathcal{L}_m(\mathbf{x}; \theta, \phi) = \mathbb{E}_{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)} \sim q_\phi(\mathbf{z} | \mathbf{x})} \left[\log \frac{1}{m} \sum_{i=1}^m \frac{p_\theta(\mathbf{x}, \mathbf{z}^{(i)})}{q_\phi(\mathbf{z}^{(i)} | \mathbf{x})} \right] \quad (48)$$

Problem 4: Semi-Supervised VAE (SSVAE). Setting:

- Small number of labeled pairs $\mathbf{x}_\ell = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{100}$ where the labels $y^{(i)}$ are the integer that the MNIST image $\mathbf{x}^{(i)}$ represents.
- Large amount of unlabeled data $\mathbf{x}_u = \{\mathbf{x}^{(i)}\}_{i=101}^{60000}$.

The SSVAE implements the generative process

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, I) \quad (49)$$

$$p(y) = \text{Cat}(y; \pi) = \frac{1}{10} \quad (50)$$

$$p_\theta(\mathbf{x} | y, \mathbf{z}) = \text{Bern}(\mathbf{x}; f_\theta(y, \mathbf{z})) \quad (51)$$

$$q_\phi(y, \mathbf{z} | \mathbf{x}) = q_\phi(y | \mathbf{x}) q_\phi(\mathbf{z} | \mathbf{x}, y) \quad (52)$$

$$q_\phi(y | \mathbf{x}) = \text{Cat}(y; f_\phi(\mathbf{x})) \quad (53)$$

$$q_\phi(\mathbf{z} | \mathbf{x}, y) = \mathcal{N}(\mathbf{z}; \mu_\phi(\mathbf{x}, y), \text{diag}(\sigma_\phi^2(\mathbf{x}, y))) \quad (54)$$

For the homework, we'll maximize the objective

$$\max_{\theta, \phi} \sum_{\mathbf{x} \in \mathbf{X}} \mathcal{L}(x; \theta, \phi) + \alpha \sum_{\mathbf{x}, y \in \mathbf{X}_\ell} \log q_\phi(y | \mathbf{x}) \quad (55)$$

¹⁴They say this is because of the factor of $p_\theta(\mathbf{x})$ but I don't see how the fraction is normalized at all.

other/idk leave me alone.

Notes while I work through some homework/coding. Recall that for VAEs, we prefer to express the ELBo in the form

$$\mathcal{L}(x; \theta, \phi) = \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x | z)] - D_{KL}(q_\phi(z | x) || p(z)) \quad (56)$$

Consider the case where both $p(z)$ and $q_\theta(z | x)$ are Gaussian, i.e. that $p(z) = \mathcal{N}(z; \mu_p, \sigma_p^2)$ and $q_\theta(z | x) = \mathcal{N}(z; \mu_q, \sigma_q^2)$. To evaluate the $D_{KL}(q || p(z))$ term,

$$D_{KL}(q || p) \triangleq \int \mathcal{N}(z; \mu_q, \sigma_q^2) \left(\log \mathcal{N}(z; \mu_q, \sigma_q^2) - \log \mathcal{N}(z; \mu_p, \sigma_p^2) \right) dz \quad (57)$$

Normalizing Flow Models

Table of Contents Local

Written by Brandon McKinzie

Recap of likelihood-based learning [48:30]. We've seen the following two model families so far, along with their respective pros/cons:

- **Autoregressive Models:** $p_\theta(x) = \prod_i^n p_\theta(x_i \mid x_{<i})$. **Pro:** tractable likelihoods. **Con:** no direct mechanism for learning features.
- **VAEs:** $p_\theta(x) = \int p_\theta(x, z) dz$. **Pro:** can learn feature representations (via latent z). **Con:** intractable marginal likelihoods.

Key question: Can we design an LVM with tractable likelihoods? Yes! With caveats.

Change of Variables [General Case¹⁵]

Define invertible $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^n$, where $X = \mathbf{f}(Z)$, and $Z = \mathbf{f}^{-1}(X)$.

$$p_X(\mathbf{x}) = p_Z(\mathbf{f}^{-1}(\mathbf{x})) \left| \det \left(\frac{\partial \mathbf{f}^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right| \quad (58)$$

$$= p_Z(\mathbf{z}) \left| \det \mathbf{J}_{\mathbf{f}^{-1}(\mathbf{x}) \rightarrow \mathbf{x}} \right| \quad (59)$$

$$= p_Z(\mathbf{z}) \left| \det \mathbf{J}_{\mathbf{f}(\mathbf{z}) \rightarrow \mathbf{z}} \right|^{-1} \quad (60)$$

$$(61)$$

$$[\mathbf{J}_{\mathbf{z} \rightarrow \mathbf{x}}]_{i,j} \triangleq \frac{\partial z_i}{\partial x_j}$$

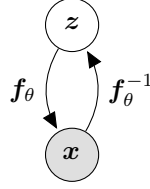
- \mathbf{x}, \mathbf{z} must be continuous and have same dimension.
- For linear $\mathbf{x} = \mathbf{A}\mathbf{z}$, change in volume is $\det \mathbf{A}$.
- For non-linear $\mathbf{f}(\cdot)$, the linearized (?) change in volume is $\det \mathbf{J}$ ¹⁶
- For any invertible matrix \mathbf{A} : $\det \mathbf{A}^{-1} = \det(\mathbf{A})^{-1}$.

¹⁵Hands-down best explanation/proof of this is this S.O. answer.

¹⁶Note that, for invertible function $f : z \mapsto x$:

$$\mathbf{J}_{\mathbf{f}(\mathbf{z}) \mapsto \mathbf{z}}^{-1} = \left(\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} \right)^{-1} = \frac{\partial \mathbf{f}^{-1}(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{J}_{\mathbf{f}^{-1}(\mathbf{x}) \mapsto \mathbf{x}} \quad (62)$$

Normalizing Flow Models [1:12:30]. A **normalizing flow model** (NFM) defines a deterministic/invertible mapping $\mathbf{f}_\theta : \mathbb{R}^n \mapsto \mathbb{R}^n$, with $X = \mathbf{f}_\theta(Z)$ and $Z = \mathbf{f}_\theta^{-1}(X)$.



- **Normalizing:** C.o.V. gives normalized density after applying an invertible transformation.
- **Flow:** Invertible transformations can be composed with each other.

$$\mathbf{x} \triangleq \mathbf{z}_M = \mathbf{f}_\theta^M \circ \dots \circ \mathbf{f}_\theta^1(\mathbf{z}_0) \triangleq \mathbf{f}_\theta(\mathbf{z}_0) \quad (63)$$

$$\mathbf{z}_0 = \left(\mathbf{f}_\theta^1\right)^{-1} \circ \dots \circ \left(\mathbf{f}_\theta^M\right)^{-1}(\mathbf{x}) \quad (64)$$

$$p_X(\mathbf{x}; \theta) = p_{Z_0}(\mathbf{f}_\theta^{-1}(\mathbf{x})) \prod_{m=1}^M \left| \det \mathbf{J}_{(\mathbf{f}_\theta^m)^{-1} \rightarrow \mathbf{z}_m} \right| \quad (65)$$

Planar Flows. Invertible transformation

$$\mathbf{x} = \mathbf{f}_\theta(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^T \mathbf{z} + b) \quad (66)$$

$$\left| \det \mathbf{J}_{\mathbf{f}(\mathbf{z}) \rightarrow \mathbf{z}} \right| = \left| \det \left(\mathbf{I} + h'(\mathbf{w}^T \mathbf{z} + b) \mathbf{u} \mathbf{w}^T \right) \right| \quad (67)$$

$$= \left| 1 + h'(\mathbf{w}^T \mathbf{z} + b) \mathbf{u}^T \mathbf{w} \right| \quad (68)$$

Note that we need to restrict parameters and non-linearity to ensure the mapping is invertible¹⁷.

¹⁷ For example, let $h = \tanh$. Need to ensure that $\mathbf{f}_\theta^{-1}(\mathbf{x})$ exists such that

$$\mathbf{f}_\theta^{-1}(\mathbf{x}) = \mathbf{f}_\theta^{-1}(\mathbf{z} + \mathbf{u} \tanh(\mathbf{w}^T \mathbf{z} + b)) = \mathbf{z} \quad (69)$$

Recall that any function f is invertible IFF it is a bijection. Equivalently, any function f is invertible IFF it is either strictly increasing or decreasing (with no local maxima/minima). Let $\mathbf{z} = \mathbf{z}_\perp + \mathbf{z}_\parallel$, defined relative to \mathbf{w} , i.e. such that $\mathbf{z}_\perp^T \mathbf{w} = 0$ and $\mathbf{z}_\parallel = \alpha \frac{\mathbf{w}}{\|\mathbf{w}\|^2}$ for some $\alpha \in \mathbb{R}$. This gives

$$f(\mathbf{z}) = \mathbf{z}_\perp + \mathbf{z}_\parallel + \mathbf{u} \tanh(\mathbf{w}^T \mathbf{z}_\parallel + b) \quad (70)$$

$$\mathbf{w}^T f(\mathbf{z}) = \alpha + \mathbf{w}^T \mathbf{u} \tanh(\alpha + b) \triangleq f_\alpha(\alpha) \quad (71)$$

Again, note that $f_\alpha(\alpha)$ is invertible IFF it's monotonic:

$$\frac{df_\alpha(\alpha)}{d\alpha} = 1 + \mathbf{w}^T \mathbf{u} \tanh'(\alpha + b) \quad (72)$$

$$1 + \mathbf{w}^T \mathbf{u} \tanh'(\alpha + b) \geq 0 \iff \mathbf{w}^T \mathbf{u} \geq -\frac{1}{\tanh'(\alpha + b)} \quad (73)$$

Since $0 < \tanh'(\cdot) \leq 1$, we simply need $\mathbf{w}^T \mathbf{u} \geq -1$. The planar flow authors accomplish this via

$$\hat{u}(\mathbf{w}, \mathbf{u}) = \mathbf{u} + \left[m(\mathbf{w}^T \mathbf{u}) - (\mathbf{w}^T \mathbf{u}) \right] \frac{\mathbf{w}}{\|\mathbf{w}\|^2} \quad (74)$$

$$m(x) = -1 + \log(1 + e^x) \quad (75)$$

Learning and Inference [1:20:10]. Learning via maximum likelihood over \mathcal{D} :

$$\max_{\theta} \log p_X(\mathcal{D}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log p_Z(\mathbf{f}_{\theta}^{-1}(\mathbf{x})) + \log \left| \det \mathbf{J}_{\mathbf{f}^{-1}(\mathbf{x}) \rightarrow \mathbf{x}} \right| \quad (76)$$

- ✓ **Exact likelihood evaluation** via inverse transformation $\mathbf{x} \mapsto \mathbf{z}$ and c.o.v. formula (instead of intractable summation over all \mathbf{z}).
- ✗ Requires evaluation of $\det \mathbf{J}$, where J is an $n \times n$ Jacobian¹⁸. Computing such a determinant in general is $\mathcal{O}(n^3)$.

Key Idea: choose transformations s.t. \mathbf{J} has special structure allowing for fast computation of $\det J$. For example, if \mathbf{J} is triangular, then $\det \mathbf{J} = \prod_i J_{ii}$. Conceptually, if \mathbf{J} is lower triangular, for example, then $(\forall i, j > i), \frac{\partial \mathbf{f}^{-1}(\mathbf{x})_i}{\partial x_j} = \frac{\partial f(z)_j}{\partial z_i} = 0$. This would be true if e.g. x_i depended only on $z_{\geq i}$.

- ✓ **Sampling** via forward transformation $\mathbf{z} \mapsto \mathbf{x}$:

$$\mathbf{z} \sim p_Z(\mathbf{z}) \quad \mathbf{x} = \mathbf{f}_{\theta}(\mathbf{z}) \quad (78) \quad \theta = \{\mathbf{w}, \mathbf{u}, b\}$$

- ✓ **Latent representations** inferred via inverse transformation $\mathbf{z} = \mathbf{f}_{\theta}^{-1}(\mathbf{x})$.

¹⁸Determinant review: For 3×3 matrices, you do that technique from physics for calculating vector cross products. For arbitrary $n \times n$ matrices,

$$\det A \triangleq \sum_{c=1}^{N_c} (-1)^{c-1} a_{1c} \det A_{11} \quad (77)$$

where a_{ij} is element at row i , column j , and A_{ij} is the $(n-1) \times (n-1)$ sub-matrix resulting from removing row i column j from the original matrix A .

Designing Invertible Transformations. In what follows, we look at two models, NICE and Real-NVP, that have invertible transformations with diagonal Jacobians.

NICE¹⁹ [9:00] composes two kinds of invertible transformations:

- **Additive coupling layers**²⁰. Partition \mathbf{z} into two disjoint subsets $\mathbf{z}_{1:d}$ and $\mathbf{z}_{d+1:n}$ for any $1 \leq d < n$.

$$[\mathbf{z} \mapsto \mathbf{x}] \quad \mathbf{x}_{1:d} = \mathbf{z}_{1:d} \quad (79)$$

$$\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} + m_\theta(\mathbf{z}_{1:d}) \quad (80)$$

$$[\mathbf{x} \mapsto \mathbf{z}] \quad \mathbf{z}_{1:d} = \mathbf{x}_{1:d} \quad (81)$$

$$\mathbf{z}_{d+1:n} = \mathbf{x}_{d+1:n} - m_\theta(\mathbf{x}_{1:d}) \quad (82)$$

where m_θ is a NN mapping d inputs to $n - d$ outputs. Jacobian of forward mapping:

$$\mathbf{J}_{\mathbf{x} \rightarrow \mathbf{z}} \triangleq \frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \begin{pmatrix} \mathbf{I}_d & \mathbf{0} \\ \frac{\partial \mathbf{x}_{d+1:n}}{\partial \mathbf{z}_{1:d}} & \mathbf{I}_{n-d} \end{pmatrix} \quad (83)$$

$$\det J = 1 \quad (84)$$

Volume preserving transformation since \det is 1 [17:17].

- **Rescaling layers** (final layer). The forward/inverse mappings and J :

$$[\mathbf{z} \mapsto \mathbf{x}] \quad x_i = s_i z_i \quad (85)$$

$$[\mathbf{x} \mapsto \mathbf{z}] \quad z_i = \frac{x_i}{s_i} \quad (86)$$

$$\mathbf{J}_{\mathbf{x} \mapsto \mathbf{z}} = \text{diag}(\mathbf{s}) \quad (87)$$

where $s_i > 0$ is the scaling factor of the i th dimension.

Real-NVP²¹. Coupling layers now shift and scale.

$$[\mathbf{z} \mapsto \mathbf{x}] \quad \mathbf{x}_{1:d} = \mathbf{z}_{1:d} \quad (88)$$

$$\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} \odot \exp(\alpha_\theta(\mathbf{z}_{1:d})) + \mu_\theta(\mathbf{z}_{1:d}) \quad (89)$$

$$[\mathbf{x} \mapsto \mathbf{z}] \quad \mathbf{z}_{1:d} = \mathbf{x}_{1:d} \quad (90)$$

$$\mathbf{z}_{d+1:n} = (\mathbf{x}_{d+1:n} - \mu_\theta(\mathbf{x}_{1:d})) \odot (\exp(-\alpha_\theta(\mathbf{x}_{1:d}))) \quad (91)$$

$$\mathbf{J}_{\mathbf{x} \rightarrow \mathbf{z}} \triangleq \frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \begin{pmatrix} \mathbf{I}_d & \mathbf{0} \\ \frac{\partial \mathbf{x}_{d+1:n}}{\partial \mathbf{z}_{1:d}} & \text{diag}(\exp(\alpha_\theta(\mathbf{z}_{1:d}))) \end{pmatrix} \quad (92)$$

$$\det J = \prod_{i=d+1}^n \exp(\alpha_\theta(\mathbf{z}_{1:d})_i) \quad (93)$$

$$= \exp\left(\sum_{i=d+1}^n \alpha_\theta(\mathbf{z}_{1:d})_i\right) \quad (94)$$

¹⁹Nonlinear Independent Components Estimation (Dinh et al., 2014).

²⁰In practice, we assign d randomly for each layer.

²¹Non-volume preserving extension of NICE.

Autoregressive Models as Flow Models [43:00]. Consider a Gaussian AR model:

$$p(\mathbf{x}) = \prod_{i=1}^n p(x_i \mid \mathbf{x}_{<i}) \quad (95)$$

$$= \prod_{i=1}^n \mathcal{N}(x_i; \mu_i, \exp(\alpha_i)^2) \quad (96)$$

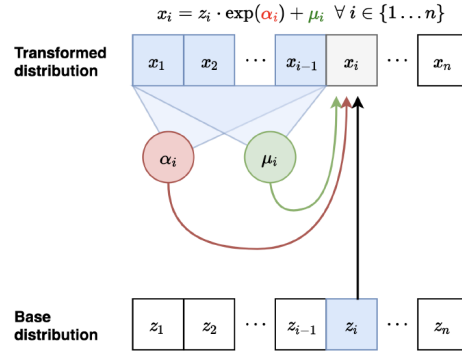
where both μ_i and α_i are functions (neural nets) of x_1, \dots, x_{i-1} (constants for $i=1$). The sampling procedure is defined as follows:

1. Sample $z_i \sim \mathcal{N}(0, 1)$ for $i = 1, \dots, n$.
2. Set $x_1 := \exp(\alpha_1)z_1 + \mu_1$.
3. For i in $[2..n]$ (inclusive): do
 - (a) Compute $\mu_i(x_1, \dots, x_{i-1})$ and $\alpha_i(x_1, \dots, x_{i-1})$.
 - (b) Set $x_i := \exp(\alpha_i)z_i + \mu_i$.

Flow Interpretation [45:57]

Sampled each z_i from a (simple) Gaussian, $\mathcal{N}(0, 1)$. Sampled the associated x_i via invertible transformations parameterized by $\mu_i(\cdot), \alpha_i(\cdot)$.

We’ve just described the **Masked Autoregressive Flow** (MAF) [46:11] model. The sampling for the forward mapping $\mathbf{z} \mapsto \mathbf{x}$ is illustrated below.



Inverse Autoregressive Flow (IAF) [52:00].

Probability Density Distillation

Student distribution is trained to minimize D_{KL} between student s and teacher t :

$$D_{KL}(s||t) = \mathbb{E}_{\mathbf{x} \sim s} [\log s(\mathbf{x}) - \log t(\mathbf{x})] \quad (97)$$

Parallel Wavenet

Training.

1. Train teacher model (**MAF**) via MLE.
2. Train student model (**IAF**) to minimize D_{KL} .

Testing. Use student model.

Improves sampling efficiency over Wavenet by 1000x!

Change of Variables: Derivation

Given some [continuous] $\mathbf{z} \sim p_Z$, and monotonic $\mathbf{f}(\mathbf{z}) = \mathbf{x}$, derive $p_X(\mathbf{x})$. Both \mathbf{z} and \mathbf{x} must have the same dimension.

1. The density for continuous RV \mathbf{z} is defined as

$$p_Z(\mathbf{z}) \triangleq \frac{\partial P(Z \leq \mathbf{z})}{\partial \mathbf{z}} \quad (98)$$

2. Since $\mathbf{f}(\mathbf{z})$ is monotonic, we know that^a the CDF for X can be defined in terms of the CDF for Z as

$$P(X \leq \mathbf{x}) = P(Z \in \{\mathbf{z} | \mathbf{f}(\mathbf{z}) \leq \mathbf{x}\}) \quad (99)$$

3. We can use the above, along with the fact that any monotonic function is invertible, to define the density $p_X(\mathbf{x})$ as

$$p_X(\mathbf{x}) \triangleq \frac{\partial P(X \leq \mathbf{x})}{\partial \mathbf{x}} \quad (100)$$

$$= \frac{\partial P(Z \in \{\mathbf{z} | \mathbf{f}(\mathbf{z}) \leq \mathbf{x}\})}{\partial \mathbf{x}} \quad (101)$$

$$= \frac{\partial P(Z \in \{\mathbf{z} | \mathbf{f}(\mathbf{z}) \leq \mathbf{x}\})}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \quad (102)$$

$$= \frac{\partial P(Z \leq \mathbf{f}^{-1}(\mathbf{x}))}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \quad (103)$$

$$= p_Z(\mathbf{f}^{-1}(\mathbf{x})) \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \quad (104)$$

$$= p_Z(\mathbf{f}^{-1}(\mathbf{x})) \mathbf{J}_{\mathbf{z} \rightarrow \mathbf{x}} \quad (105)$$

In summary, the density $p_X(\mathbf{x})$, where $\mathbf{x} = \mathbf{f}(\mathbf{z})$, can be written as:

$$p_X(\mathbf{x}) = p_Z(\mathbf{z} = \mathbf{f}^{-1}(\mathbf{x})) \quad (106)$$

TODO: FINISH THIS AFTER THE STATS BOOKS ARRIVES

^aThis part is intuitively clear to me, yet I'm not sure how I'd actually *prove* it.

Problem 1: Flow Models. Implement a **Masked Autoregressive Flow** (MAF) model. MAF models are themselves composed of **Masked Autoregressive Distribution Estimator** (MADE) blocks (each $\mathbf{f}(\cdot)$ will be a MADE block).

```
nf_blocks = []
for i in range(self.n_flows):
    nf_blocks.append(
        MADE(self.input_size, self.hidden_size, self.n_hidden))
    nf_blocks.append(PermuteLayer(self.input_size)) # permute dims
self.nf = nn.Sequential(*nf_blocks)
```

Each MADE block is consists of some number of hidden layers, each with an associated mask.

- **Forward Pass** [$\mathbf{z} \mapsto \mathbf{x}$]: Assuming that \mathbf{z} comes from some base noise distribution like $\mathcal{N}(0, I)$ (is this still valid when e.g. \mathbf{z} is just the output of the previous layer (as in MAF)), we can use the reparameterization trick for Gaussian sampling to obtain each x_i :

$$x_1 = \mu_1 + z_1 \exp(\alpha_1) \tag{107}$$

$$x_2 = \mu_2(x_1) + z_2 \exp(\alpha_2(x_1)) \tag{108}$$

$$\vdots \tag{109}$$

$$x_n = \mu_n(x_1, \dots, x_{n-1}) + z_n \exp(\alpha_n(x_1, \dots, x_{n-1})) \tag{110}$$

Since this is an invertible transformation,

$$\log \left| \det \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right| = \log \left| \det \frac{\partial \mathbf{x}}{\partial \mathbf{z}} \right|^{-1} \tag{111}$$

$$= \log \left| \prod_{i=1}^n \exp(\alpha_i) \right|^{-1} \tag{112}$$

$$= - \sum_{i=1}^n \alpha_i \tag{113}$$

Note that everything we've seen so far has been within a single MADE layer. Our full MAF model consists of 5 such MADE layers.

Generative Adversarial Networks

Table of Contents Local

Written by Brandon McKinzie

Recap [8:00]. We've seen the following model families so far:

- **Autoregressive Models:** $p_\theta(\mathbf{x}) = \prod_i^n p_\theta(x_i \mid x_{<i})$. **Pro:** tractable likelihoods. **Con:** no direct mechanism for learning features.
- **VAEs:** $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, z) dz$. **Pro:** can learn feature representations (via latent z). **Con:** intractable marginal likelihoods.
- **Normalizing Flow Models:** $p_{\theta, X}(\mathbf{x}) = p_{\theta, Z}(\mathbf{f}_\theta^{-1}(\mathbf{x})) |\det \mathbf{J}_{\mathbf{z} \rightarrow \mathbf{x}}|$

All the above are based on maximizing likelihoods (or approximations).

Example: great $\log p_{\text{test}}(\mathbf{x})$; poor samples [12:14]

Consider a noise mixture model $p_\theta(\mathbf{x})$ where \mathbf{x} is some d_x -dimensional vector, with sampling procedure

1. Sample binary-valued $b \sim \text{Bern}(0.01)$.
2. If $b = 1$ (probability 1 percent), then return sample $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$.
3. Else, return sample $\mathbf{x} \sim p_{\text{noise}}(\mathbf{x})$.

Note that the above procedure can be written as

$$p_\theta(\mathbf{x}) = 0.01 p_{\text{data}}(\mathbf{x}) + 0.99 p_{\text{noise}}(\mathbf{x}) \quad (114)$$

In what follows, we find an upper and lower bound on $\log p_\theta(\mathbf{x})$ to prove it can have great likelihoods.

$$\log p_\theta(\mathbf{x}) = \log(0.01 p_{\text{data}}(\mathbf{x}) + 0.99 p_{\text{noise}}(\mathbf{x})) \quad (115)$$

$$\geq \log 0.01 p_{\text{data}}(\mathbf{x}) \quad (116)$$

$$= \log p_{\text{data}}(\mathbf{x}) - \log 100 \quad (117)$$

$$\mathbb{E}_{p_{\text{data}}} [\log p_\theta(\mathbf{x})] \geq \mathbb{E}_{p_{\text{data}}} [\log p_{\text{data}}(\mathbf{x})] - \log 100 \quad (118)$$

$$\mathbb{E}_{p_{\text{data}}} [\log p_{\text{data}}(\mathbf{x})] \geq \mathbb{E}_{p_{\text{data}}} [\log p_\theta(\mathbf{x})] \quad (119)$$

where the last line is true because $D_{KL}(p_{\text{data}} \| p_\theta) \geq 0$. This gives us the following upper and lower bound:

$$\mathbb{E}_{p_{\text{data}}} [\log p_{\text{data}}(\mathbf{x})] \geq \mathbb{E}_{p_{\text{data}}} [\log p_\theta(\mathbf{x})] \geq \mathbb{E}_{p_{\text{data}}} [\log p_{\text{data}}(\mathbf{x})] - \log 100 \quad (120)$$

For larger and larger d_x , the absolute values of $\log p_{\text{data}}(\mathbf{x})$ increases^a. and the contribution of $-\log 100$ becomes less and less significant, until

$$\mathbb{E}_{p_{\text{data}}} [\log p_\theta(\mathbf{x})] \approx \mathbb{E}_{p_{\text{data}}} [\log p_{\text{data}}(\mathbf{x})] \quad (121)$$

which results in a model with great likelihoods, but horrible samples (mostly noise).

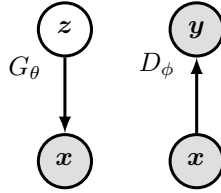
^aThink about it. Higher dimensional spaces have more unique possible values for \mathbf{x} . For higher dimensional spaces, the observed points \mathbf{x} represent a much smaller fraction of the total number of points, and will thus likely get assigned lower probabilities (and thus larger $|\log p|$) than e.g. observed data in small dimensional spaces.

Towards likelihood-free learning [11:23]. GANs can be motivated by asking whether optimizing *likelihoods* is a good approach. There exist cases where model can generate poor samples with great likelihoods, and vice-versa. For example, memorizing training set can result in great samples, but have horrible test likelihoods. Can we disentangle likelihoods and samples?

Two-Sample Tests [23:48]

- Given $S_1 = \{\mathbf{x} \sim P\}$ and $S_2 = \{\mathbf{x} \sim Q\}$, a **two-sample test** considers the following hypotheses:
 - Null hypothesis $H_0 : P = Q$.
 - Alternate hypothesis $H_1 : P \neq Q$.
- Test statistic T compares S_1 and S_2 (e.g. difference in means)²².
- If $T < \alpha$, then accept H_0 else reject it.
- **Key observation:** T is likelihood-free since it doesn't involve densities P or Q (only samples).

GANs [30:40]. Key idea: *learn* a statistic that *maximizes* a suitable notion of distance between S_1 and S_2 ²³. GANs involve a two player minimax game between a **generator** G_θ and a **discriminator**



- **Generator** $G_\theta : z \mapsto x$. Deterministic directed LVM. **Minimizes** the two-sample test objective in support of $H_0 : p_{data} = p_\theta$.
- **Discriminator** $D_\phi : x \mapsto y$ where y is a binary RV. **Maximizes** the two-sample test objective in support of $H_1 : p_{data} \neq p_\theta$. For fixed G , the discriminator D is performing binary classification with cross entropy.

²²We interpret large T as meaning the samples appear to come from *different* distributions, and low T as meaning the samples appear to come from the *same* distribution

²³The wording is subtle/confusing here. What we mean is we want to somehow learn a high-quality statistic T that's very large when the samples are not from the same distribution. Why? Because such a T makes the task of minimizing its value (job of the generator) difficult. If T was low to begin with, there wouldn't be much of a task. Stated another way: given that G wants to *minimize* the test statistic, the job of D is to *find* a test statistic that is hard for G to minimize.

The GAN objective function is defined as [38:00]

$$\min_{\theta} \max_{\phi} V(G_{\theta}, D_{\phi}) = \mathbb{E}_{\mathbf{x} \sim p_{data}} [\log D_{\phi}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} [\log(1 - D_{\phi}(G_{\theta}(\mathbf{z})))] \quad (122)$$

Optimal Discriminator

The optimal discriminator function D for a given generator G is

$$D_G^*(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_G(\mathbf{x})} \quad (123)$$

If we plug this into $V(G, D)$, we obtain

$$V(G, D_G^*(\mathbf{x})) = 2D_{JSD}(p_{data} || p_G) - \log 4 \quad (124)$$

$$D_{JSD}(p || q) \triangleq \frac{1}{2} \left(D_{KL} \left(p || \frac{p+q}{2} \right) + D_{KL} \left(q || \frac{p+q}{2} \right) \right) \quad (125)$$

where $D_{JSD}(p || q)$ is the **Jensen-Shannon Divergence**²⁴.

The GAN Training Algorithm [52:00]

1. Sample minibatch of m training points from data.

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(m)} \sim \mathcal{D}$$

2. Sample minibatch of m noise vectors from p_Z .

$$\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \dots, \mathbf{z}^{(m)} \sim p_Z$$

3. Update generator parameters θ by stochastic gradient **descent**.

$$\nabla_{\theta} V(G_{\theta}, D_{\phi}) = \frac{1}{m} \nabla_{\theta} \sum_{i=1}^m \log(1 - D_{\phi}(G_{\theta}(\mathbf{z}^{(i)}))) \quad (130)$$

4. Update discriminator parameters ϕ by stochastic gradient **ascent**.

$$\nabla_{\phi} V(G_{\theta}, D_{\phi}) = \frac{1}{m} \nabla_{\phi} \sum_{i=1}^m [\log D_{\phi}(\mathbf{x}^{(i)}) + \log(1 - D_{\phi}(G_{\theta}(\mathbf{z}^{(i)})))] \quad (131)$$

5. Repeat for fixed number of epochs.

²⁴Properties of $D_{JSD}(p || q)$:

$$D_{JSD}(p || q) \geq 0 \quad (126)$$

$$D_{JSD}(p || q) = 0 \quad \text{iff} \quad p = q \quad (127)$$

$$D_{JSD}(p || q) = D_{JSD}(q || p) \quad (128)$$

$$\sqrt{D_{JSD}(p || q)} \leq \sqrt{D_{JSD}(p || r)} + \sqrt{D_{JSD}(r || q)} \quad (129)$$

Challenges.

- **Unstable Optimization.** G and D often oscillate during training without converging. No robust stopping criteria.
- **Mode Collapse [1:11:00].** Generator collapses to one or few samples (dubbed as “modes”). For example, if p_{data} is multimodal, the generator may collapse to modeling just one of these modes.
- **Evaluation.**

Beyond KL and JSD.

- $D_{KL}(p||q)$: used by AR and Flow models.
- $D_{JSD}(p||q)$ (scaled & shifted): original GAN objective.

f-divergence [8:30]

Given two densities p and q , the **f-divergence** is given by

$$D_f(p||q) \triangleq \mathbb{E}_{\mathbf{x} \sim q} \left[f \left(\frac{p(\mathbf{x})}{q(\mathbf{x})} \right) \right] \quad (132)$$

where f is any convex, **lower-semicontinuous**²⁵ function with $f(1) = 0$. Note that if $f(u) = u \log u$, $D_f(p||q) \equiv D_{KL}(p||q)$.

Fenchel Conjugate (Complex Conjugate)

For any function²⁶ $f(\cdot)$, its **Fenchel conjugate** (a.k.a. the complex conjugate) is defined as²⁷

$$f^*(t) \triangleq \sup_{u \in \text{dom}_f} (ut - f(u)) \quad (134)$$

- f^* is always lower semi-continuous.
- $f^{**} = f$ IFF f is convex and lower semi-continuous.

²⁵Ok, I understand this now. Remember how piecewise functions over real numbers are defined like

$$f(x) = \begin{cases} 1 & x < 0 \\ -1 & x \geq 0 \end{cases} \quad (133)$$

In particular, we say that $f(0) = -1$, while $f(0 + \epsilon) = 1$. For this reason, we say f is **lower-semicontinuous**, because $\forall x$, the “function value” at x is either literally just $f(x)$ (true everywhere if e.g. f were continuous), or it’s larger. Stated even more sloppily, it just means that whenever you try to evaluate f at x , you are guaranteed to get back exactly $f(x)$ or a value that’s larger than $f(x)$ (which occurs if you are at a discontinuity).

²⁶Technically, any function $f : X \mapsto \mathbb{R} \cup \{-\infty, +\infty\}$ where X is a real vector space. Similarly $f^* : X^* \mapsto \mathbb{R} \cup \{-\infty, +\infty\}$, where X^* is the dual space to X .

²⁷Recall that the **supremum** of a set $S \subset T$ is the smallest element of T that’s greater than or equal to all elements in S .

To use f -divergences as our 2-sample test objective for likelihood-free learning, we need to be able to estimate it only via samples²⁸. We can obtain a lower bound that is likelihood-free wrt p and q to any f -divergence via its Fenchel conjugate [20:00].

$$D_f(p||q) \triangleq \mathbb{E}_{\mathbf{x} \sim q} \left[f \left(\frac{p(\mathbf{x})}{q(\mathbf{x})} \right) \right] \quad (135)$$

$$= \mathbb{E}_{\mathbf{x} \sim q} \left[f^{**} \left(\frac{p(\mathbf{x})}{q(\mathbf{x})} \right) \right] \quad (136)$$

$$= \mathbb{E}_{\mathbf{x} \sim q} \left[\sup_{t \in \text{dom}_{f^*}} \left(t \frac{p(\mathbf{x})}{q(\mathbf{x})} - f^*(t) \right) \right] \quad (137)$$

$$:= \mathbb{E}_{\mathbf{x} \sim q} \left[T(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} - f^*(T(\mathbf{x})) \right] \quad (138)$$

$$= \int d\mathbf{x} [T(\mathbf{x})p(\mathbf{x}) - f^*(T(\mathbf{x}))q(\mathbf{x})] \quad (139)$$

$$\geq \sup_{\hat{T} \in \mathcal{T}} \int d\mathbf{x} [\hat{T}(\mathbf{x})p(\mathbf{x}) - f^*(\hat{T}(\mathbf{x}))q(\mathbf{x})] \quad (140)$$

$$= \sup_{\hat{T} \in \mathcal{T}} \left(\mathbb{E}_{\mathbf{x} \sim p} [\hat{T}(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim q} [f^*(\hat{T}(\mathbf{x}))] \right) \quad (141)$$

where $\mathcal{T} : \mathcal{X} \mapsto \mathbb{R}$ is an arbitrary class of functions.

f-GAN: Variational Divergence Minimization. We can rewrite the result above, plugging in $p := p_{data}$, $q := p_{G_\theta}$, $\hat{T} := T_\phi$, to obtain

$$D_f(p||q) \geq \sup_{\hat{T} \in \mathcal{T}} \left(\mathbb{E}_{\mathbf{x} \sim p} [\hat{T}(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim q} [f^*(\hat{T}(\mathbf{x}))] \right) \quad (142)$$

$$\min_{\theta} \max_{\phi} F(\theta, \phi) = \mathbb{E}_{\mathbf{x} \sim p_{data}} [T_\phi(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_{G_\theta}} [f^*(T_\phi(\mathbf{x}))] \quad (143)$$

where the interpretations of ϕ (discriminator) and θ (generator) are consistent with our original GAN formulation²⁹

²⁸Why can't we already? Technically, all MLE applications minimize $D_{KL}(p_{data}||p_\theta)$ but (obviously) never have direct access to p_{data} . Confused in general by the notion of likelihood-free learning, since it seems no different *in practice* (as opposed to in theory/formulas)

²⁹For convenience:

$$\min_{\theta} \max_{\phi} V(G_\theta, D_\phi) = \mathbb{E}_{\mathbf{x} \sim p_{data}} [\log D_\phi(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} [\log(1 - D_\phi(G_\theta(\mathbf{z})))] \quad (144)$$

Inferring Latent Representations in GANs [30:50]. Unlike...

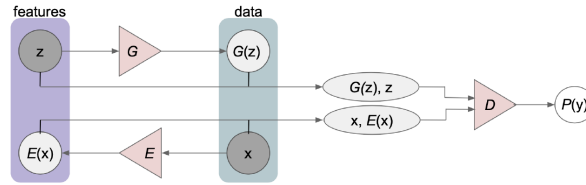
- Normalizing flow models: $\mathbf{z} = \mathbf{f}_\theta^{-1}(\mathbf{x})$.
- VAEs: $\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{x})$.

...the GAN generator is a directed LVM $\mathbf{z} \mapsto \mathbf{x}$ and it need not be invertible. In practice, there are a few ways to get latent representations [32:00]:

- [Use D] For any point \mathbf{x} , use the activations of the prefinal layer of D as feature representation³⁰
- [Change learning algorithm] Compare samples of (\mathbf{z}, \mathbf{x}) (instead of just \mathbf{x}) from joint distributions of model/data. Note that we don't have access to \mathbf{z} for the \mathbf{x} sampled from p_{data} .

BiGAN [35:00]. Introduces an encoder $E : \mathbf{x} \mapsto \mathbf{z}$ to obtain latent representation from $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$.

- Discriminator maximizes two-sample test objective between $(\mathbf{z}, G(\mathbf{z}))$ and $(E(\mathbf{x}), \mathbf{x})$.
- After training, we can **sample** $\mathbf{x} \sim p_{G_\theta}(\mathbf{x})$ (as usual), but now we can also **infer** $\mathbf{z} = E(\mathbf{x})$.



CycleGAN. Adversarial training across two domains. Setting: we have unpaired sets of samples $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$. Can we translate $\mathcal{X} \rightleftharpoons \mathcal{Y}$ in an unsupervised manner? **CycleGAN** learns two parameterized conditional generative models:

$$G : \mathcal{X} \mapsto \mathcal{Y} \quad F : \mathcal{Y} \mapsto \mathcal{X} \quad (145)$$

$$F(G(X)) \approx X \quad G(F(Y)) \approx Y \quad (146)$$

Discriminator D_Y compares observed Y with samples $\hat{Y} = G(X)$, while D_X compares observed X with samples $\hat{X} = F(Y)$. The CycleGAN loss function is as follows, where for brevity I've defined $\mathcal{L}(G, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y)$

$$\min_{F, G, D_X, D_Y} \mathcal{L}_G + \mathcal{L}_F + \lambda (\mathbb{E}_X [\|F(G(X)) - X\|_1] + \mathbb{E}_Y [\|G(F(Y)) - Y\|_1]) \quad (147)$$

$$\mathcal{L}_G := \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \quad (148)$$

$$\mathcal{L}_F := \mathcal{L}_{\text{GAN}}(F, D_X, X, Y) \quad (149)$$

³⁰What? How is this even remotely a valid approach? What does the prefinal layer have to do???

Energy-Based Models I

Say we have some function $g_\theta(x)$ and we want to model $p(x)$. This obviously means we must satisfy:

$$g_\theta(x) \geq 0 \quad \forall x \quad (150)$$

$$\int g_\theta(x) dx = 1 \quad (151)$$

We'll be referring to the integral above as the *volume* of g_θ . We typically choose g_θ s.t. we know the volume *analytically*, e.g.

$$\text{[Gaussian]} \quad g_{\mu,\sigma}(x) = e^{-\frac{(x-\mu)^2}{2\sigma^2}} \longrightarrow \sqrt{2\pi\sigma^2} \quad (152)$$

$$\text{[Exponential]} \quad g_\lambda(x) = e^{-\lambda x} \longrightarrow \frac{1}{\lambda} \quad (153)$$

Energy-Based Model. “Give up” on computing volume analytically and just do:

$$p_\theta(x) = \frac{1}{Z(\theta)} \exp(f_\theta(x)) \quad (154)$$

$$Z(\theta) = \int \exp(f_\theta(x)) dx \quad (155)$$

Exponentials allow us to work in log-probability space with f_θ . Allows for capturing large variations in probability. Also because exponential families, etc. In physics, $-f_\theta(x)$ represents the **energy**.

- ✓ Extremely flexible. Works with basically any f_θ .
- ✗ Sampling from $p_\theta(x)$ is hard.
- ✗ Evaluating/optimizing likelihood $p_\theta(x)$ is hard (learning is hard).
- ✗ No feature learning.
- ✗ Computing $Z(\theta)$ scales exponentially in dimensionality of x .

Applications. Note that probability ratios for two points x and x' don't require knowing $Z(\theta)$:

$$\frac{p_{\theta}(x)}{p_{\theta}(x')} = \exp(f_{\theta}(x) - f_{\theta}(x')) \quad (156)$$

Useful for e.g. anomaly detection and denoising.

Training. Goal: maximize $f_{\theta}(x_{train})/Z(\theta)$ by increasing numerator, decreasing denominator. **Contrastive divergence** uses MC estimates to approximate $Z(\theta)$. Takes gradients $\nabla_{\theta}(f_{\theta}(x_{train}) - f_{\theta}(x_{sample}))$.

CONCEPTS

CONTENTS

2.1	Evidence Lower Bound (ELBo)	32
-----	---------------------------------------	----

Evidence Lower Bound (ELBo)

Table of Contents Local

Written by Brandon McKinzie

The **evidence** (a.k.a. the marginal likelihood) is defined as $p(x) = \sum_z p(x, z)$. The goal here to find a *lower bound* on the evidence. Computing the evidence is typically intractable, due to the expectation over latent variables:

$$\log p_\theta(x) = \log \mathbb{E}_{z \sim q(z)} \left[\frac{p_\theta(x, z)}{q(z)} \right] \quad (157)$$

Note that the above is true for *any* distribution $q(z)$.

Convex functions and Jensen's Inequality

A function $f(x)$ is convex over the interval $[x = a, x = b]$ if every chord of the function lies above the function. That is, $\forall x_1, x_2 \in [a, b]$ and $0 \leq \lambda \leq 1$:

$$f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2) \quad (158)$$

$$f(\mathbb{E}[x]) \leq \mathbb{E}[f(x)] \quad \text{[Jensen's Inequality]} \quad (159)$$

Observe that \log is a **concave function**. In other words, $\log(\mathbb{E}[x]) \geq \mathbb{E}[\log x]$. This means that $\mathbb{E}[\log x]$ is a *lower bound* on $\log \mathbb{E}[x]$. From above, we see that $\log p_\theta(x)$ can be rewritten as a log of an expectation. As such, we define the **evidence lower bound (ELBo)**, denoted $\mathcal{L}(x; \theta, \phi)$, as³¹

$$\mathcal{L}(x; \theta, \phi) \triangleq \mathbb{E}_{z \sim q(z)} \left[\log \frac{p_\theta(x, z)}{q(z)} \right] \quad (166)$$

$$= \mathbb{E}_{z \sim q(z)} [\log p_\theta(x, z)] + H(q) \quad (167)$$

$$= \log p_\theta(x) - D_{KL}(q(z) || p_\theta(z | x)) \quad (168)$$

$$= \mathbb{E}_{z \sim q(z)} [\log p_\theta(x | z)] - D_{KL}(q(z) || p(z)) \quad (169)$$

$$\log p_\theta(x) \geq \mathcal{L}(x; \theta, \phi) \quad (170)$$

³¹Derivation of $\mathcal{L}(x; \theta, \phi)$ expressed with D_{KL} :

$$D_{KL}(q(z) || p_\theta(z | x)) \triangleq \mathbb{E}_q \left[\log \frac{q(z)}{p_\theta(z | x)} \right] \quad (160)$$

$$= -H(q) - \mathbb{E}_q [\log p_\theta(z | x)] \quad (161)$$

$$= -H(q) - \mathbb{E}_q [\log p_\theta(x, z) / p_\theta(x)] \quad (162)$$

$$= -H(q) - \mathbb{E}_q [\log p_\theta(x, z)] + \mathbb{E}_q [\log p_\theta(x)] \quad (163)$$

$$= -H(q) - \mathbb{E}_q [\log p_\theta(x, z)] + \log p_\theta(x) \quad (164)$$

$$= -\mathcal{L}(x; \theta, \phi) + \log p_\theta(x) \quad (165)$$

Questions/Answers

- **Q:** Why is a lower bound useful at all?
 - **A:** In this case it's useful because $\log p_\theta(x) = \mathcal{L}(x; \theta, \phi)$ when $D_{KL}(q(z) || p_\theta(z | x)) = 0$. The fact that the formula for $\mathcal{L}(x; \theta, \phi)$ is a lower bound on $\log p_\theta(x)$ isn't useful by itself; the useful part is that we can drive $\mathcal{L}(x; \theta, \phi)$ closer and closer to $\log p_\theta(x)$ by minimizing the KL-divergence of q with the posterior $p_\theta(z | x)$.
- **Q:** Why is $\log p(x)$ what we want? Why not just $p(x)$?
 - **A:** idk. Probably because logs are better for computational reasons.
- **Q:** Look at the form of $\mathcal{L}(x; \theta, \phi)$ given by 169. It implies that our lower bound gets maximized when $D_{KL}(q(z) || p(z)) \rightarrow 0$. Doesn't this just encourage our encoder to learn $p(z)$? How is that even useful (since we already know what $p(z)$ is? What if we just use $p(z)$ directly?)
 - **A:** yo good question idk tho