Contents

Lectures

Contents

Lectures **September 25, 2019**

Representation

Table of Contents Local *Written by Brandon McKinzie*

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. $1 + 2 + \cdots + 2^{n-1} = 2^n - 1$ (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 $p(x_i \mid \mathbf{x}_{\mathbf{A_i}})$ for set $\boldsymbol{X}_{A_i}$ of RVs. The model joint as

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

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.

[1] In general, you are "exponential in" the number of RVs appearing in a given CPD

Autoregressive Models

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:

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

$$p_{logit}(X_2{=}1 \mid x_1; \boldsymbol{\alpha}^{(2)}) = \sigma(\boldsymbol{\alpha}_0^{(2)} + \boldsymbol{\alpha}_1^{(2)} x_1) \tag{3}$$

$$p_{logit}(X_n{=}1 \mid x_1, \dots, x_{n-1}; \boldsymbol{\alpha}^{(n)}) = \sigma(\boldsymbol{\alpha}_0^{(n)} + \sum_{i=1}^{n-1} \boldsymbol{\alpha}_i^{(n)} x_i) \tag{4}$$

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

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

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

A vanilla AE is *not* a generative model: it doesn't define some $p(x)$ we can sample from[2].

[2] Why? 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_2, x_1:

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

$$p(x_1 \mid x_2, x_3) \qquad p(x_2) \qquad p(x_3 \mid x_2)$$

- 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

$$D_{KL}\left(p||q\right) = \sum_{\boldsymbol{x}} p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \tag{5}$$

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

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

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

- 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).

[4] Examples 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).

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

[6] Mathematically, you can indeed see that:

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

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:

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

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

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

Properties of the MC estimate $\hat{g}$:

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

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

$$\boldsymbol{z} \sim \text{Categorical}(1, \dots, K) \tag{10}$$
$$p(\boldsymbol{x} \mid \boldsymbol{z}{=}k) = \mathcal{N}(\mu_k, \Sigma_k) \tag{11}$$

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

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

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

$$\boldsymbol{z} \sim \mathcal{N}(0, I) \tag{13}$$
$$p(\boldsymbol{x} \mid \boldsymbol{z}) = \mathcal{N}\left(\mu_\theta(\boldsymbol{z}), \Sigma_\theta(\boldsymbol{z})\right) \tag{14}$$

where μ_θ, Σ_θ are neural networks.

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

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

$$p_\theta(x) = \sum_z p_\theta(x, z) = |\mathcal{Z}| \mathbb{E}_{z \sim U(\mathcal{Z})} \left[p_\theta(x, z)\right] \tag{15}$$

where $\mathcal{Z}$ is the set of all possible values that z can take. **NB:** the expectation is over (uniform) $U(\mathcal{Z})$, NOT $p(z)$ as we often see[7]; hence the scaling factor of $|\mathcal{Z}|$. The naive MC procedure is then to perform:

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

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

2. Approximate expectation with sample average:

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

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 $U(\mathcal{Z})$.

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

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

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 equivalent[8]

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

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

$$= \sum_z q(z) \log p_\theta(x,z) - \sum_z q(z) \log q(z) \tag{20}$$

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

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

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 $q(z) = p_\theta(z \mid x)$.
- $\log p_\theta(x) = \text{ELBo} + D_{KL}\left(q(z) || p(z \mid x)\right)$

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

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

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

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

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

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

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

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

$$\mathbb{E}_{q_\phi(z)}[r(z)] = \int q_\phi(z) r(z) \mathrm{d}z \tag{25}$$

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

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

$$z = \mu + \sigma\epsilon = g(\epsilon; \phi) \tag{27}$$

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

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**:

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

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 $q(z \mid x^i)$ using $q_\lambda(z \mid x)$.

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

$$\begin{aligned}\mathcal{L}(x;\theta,\phi) &= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x,z) - \log q_\phi(z \mid x)\right] && (30)\\ &= \mathbb{E}_{q_\phi(z|x)}\left[(\log p_\theta(x,z) - \log p(z)) - (-\log p(z) + \log q_\phi(z \mid x))\right] && (31)\\ &= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x \mid z)\right] - D_{KL}\left(q_\phi(z \mid x)||p(z)\right) && (32)\end{aligned}$$

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 $x \sim \sum_z p(x,z) = \mathbb{E}_{z\sim p(z)}\left[p(x \mid z)\right]$ (challenging).
2. *Importance/MC Sampling.* We can equivalently write $p_\theta(x) = \mathbb{E}_{z\sim q(z)}\left[p_\theta(x,z)/q(z)\right]$ 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 $\log \mathbb{E}\left[p/q\right]$. Therefore, we want to maximize this quantity.
4. *Variational Inference.* $\mathcal{L}$ is maximized when $q(z) = p_\theta(z \mid x)$. The goal of VI is to learn a parameterized function $q_\phi(z)$ as close as possible to $p_\theta(z \mid x)$. The challenge is that $p_\theta(z \mid x)$ is a function of x – $p_\theta(z \mid x^i)$ 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 phi^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 $q_\lambda(z \mid x)$.

[10] Called "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.

[11] Recall 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 $q_\phi(z \mid x^i)$
3. **Decoder**: Reconstruct $\hat{x}$ by MC sampling from $p_\theta(x \mid \hat{z})$

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 $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(z \mid x)]$. **Ok but why MC sampling from decoder? The first time in $\mathcal{L}$** implies we should simply *evaluate* $\log p(x \mid z)$ for each sample of z from the encoding step.

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

- $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)]$ encourages $\hat{x} \approx x^i$ (x^i likely under $p_\theta(x \mid \hat{z})$).
- $D_{KL}(q_\phi(z \mid x)||p(z))$ encourages $\hat{z}$ to be likely under $p(z)$.
- Our approximations work well if $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)]$ is *large*. The largest this expectation can get corresponds to how well large values of $q_\phi(z \mid x)$ align with large values of $p_\theta(x \mid z)$. If they *do* align well, then our samples from $z \sim q$ (which will naturally tend to be values of $\hat{z}$ for which $q_\phi(z \mid x)$ is large) will correspond to the large values of $p_\theta(x \mid \hat{z})$.
- As we learn, $D_{KL}(q_\phi(z \mid x)||p(z))$ encourages $q_\phi(z \mid x)$ (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 $\mathcal{L}(x; \theta, \phi)$:

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

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

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

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

Problem 1: Implementing the VAE. Setting:

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

Setup

The VAE for this problem is defined by the generative process

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z} \mid 0, I) \tag{37}$$

$$\textbf{[decoder]} \quad p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) = \text{Bern}(\boldsymbol{x} \mid f_\theta(\boldsymbol{z})) \tag{38}$$

Model and Sampling

where $\boldsymbol{z}$ has dimension `z_dim`[12]. The function $f_\theta(\boldsymbol{z})$ is parameterized by a neural network with weights θ. It outputs logits for each of the d values associated with $\boldsymbol{x}$. 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 $p_\theta(\boldsymbol{x})$ (needed for MLE) is intractable:

$$p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}, \boldsymbol{z}) \mathrm{d}\boldsymbol{z} \tag{39}$$

Intractability of $p(x)$

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

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

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

[12] Don't forget to sqrt the variance when computing $\boldsymbol{z}$ with the reparameterization trick!

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

$$\textbf{[encoder]} \quad q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) = \mathcal{N}\left(\boldsymbol{x}; \mu_\phi(\boldsymbol{x}), \operatorname{diag}\left(\sigma_\phi^2(\boldsymbol{x})\right)\right) \tag{41}$$

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

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

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 $\boldsymbol{z}$ MC sample estimate instead of dealing with the expectation. In other words, we literally compute $\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z}^{(1)})$ where $\boldsymbol{z}^{(1)}$ is our single sample from $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$. Apparently the KL term can be computed analytically if $q_\phi(\boldsymbol{z} \mid vecx)$ and $p(\boldsymbol{z})$ are both Gaussian:

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

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

$$p_\theta(\boldsymbol{z}) = \sum_{i=1}^{k} \frac{1}{k} \mathcal{N}\left(\boldsymbol{z}; \mu_i, \operatorname{diag}\left(\sigma_i^2\right)\right) \tag{44}$$

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 $\boldsymbol{z}^{(1)}$:

$$D_{KL}\left(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) || p(\boldsymbol{z})\right) \approx \log q_\phi(\boldsymbol{z}^{(1)} \mid \boldsymbol{x}) - \log p_\theta(\boldsymbol{z}^{(1)}) \tag{45}$$

$$= \log \mathcal{N}\left(\boldsymbol{z}^{(1)}; \mu_\phi(\boldsymbol{x}), \operatorname{diag}\left(\sigma_\phi^2(\boldsymbol{x})\right)\right) - \log \sum_{i=1}^{k} \frac{1}{k} \mathcal{N}\left(\boldsymbol{z}^{(1)}; \mu_i, \operatorname{diag}\left(\sigma_i^2\right)\right) \tag{46}$$

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

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

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

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

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

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

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

The SSVAE implements the generative process

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z};0,I) \tag{49}$$
$$p(y) = \mathrm{Cat}(y;\pi) = \frac{1}{10} \tag{50}$$
$$p_\theta(\boldsymbol{x}\mid y,\boldsymbol{z}) = \mathrm{Bern}(\boldsymbol{x};f_\theta(y,\boldsymbol{z})) \tag{51}$$

$$q_\phi(y,\boldsymbol{z}\mid\boldsymbol{x}) = q_\phi(y\mid\boldsymbol{x})q_\phi(\boldsymbol{z}\mid\boldsymbol{x},y) \tag{52}$$
$$q_\phi(y\mid\boldsymbol{x}) = \mathrm{Cat}(y;f_\phi(\boldsymbol{x})) \tag{53}$$
$$q_\phi(\boldsymbol{z}\mid\boldsymbol{x},y) = \mathcal{N}\left(\boldsymbol{z};\mu_\phi(\boldsymbol{x},y),\mathrm{diag}\left(\sigma_\phi^2(\boldsymbol{x},y)\right)\right) \tag{54}$$

For the homework, we'll maximize the objective

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

[14] They say this is because of the factor of $p_\theta(\boldsymbol{x})$ 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

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

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

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

Lectures **October 14, 2019**

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**: $p_\theta(x) = \int p_\theta(x, z)\mathrm{d}z$. **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 Case[15]]

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

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

$$= p_Z(\boldsymbol{z})\left|\det \boldsymbol{J}_{\boldsymbol{f}^{-1}(\boldsymbol{x})\to\boldsymbol{x}}\right| \tag{59}$$

$$= p_Z(\boldsymbol{z})\left|\det \boldsymbol{J}_{\boldsymbol{f}(\boldsymbol{z})\to\boldsymbol{z}}\right|^{-1} \tag{60}$$

$$\tag{61}$$

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

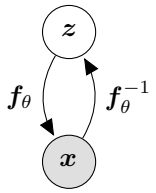
- $\boldsymbol{x}, \boldsymbol{z}$ *must be continuous and have same dimension.*
- *For linear* $\boldsymbol{x} = \boldsymbol{A}\boldsymbol{z}$, *change in volume is* $\det A$.
- *For non-linear* $\boldsymbol{f}(\cdot)$, *the linearized (?) change in volume is* $\det \boldsymbol{J}$[16]
- *For any invertible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

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

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

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

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



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

$$\boldsymbol{x} \triangleq \boldsymbol{z}_M = \boldsymbol{f}_\theta^M \circ \cdots \circ \boldsymbol{f}_\theta^1(\boldsymbol{z}_0) \triangleq \boldsymbol{f}_\theta(\boldsymbol{z}_0) \tag{63}$$

$$\boldsymbol{z}_0 = \left(\boldsymbol{f}_\theta^1\right)^{-1} \circ \cdots \circ \left(\boldsymbol{f}_\theta^M\right)^{-1}(\boldsymbol{x}) \tag{64}$$

$$p_X(\boldsymbol{x};\theta) = p_{Z_0}(\boldsymbol{f}_\theta^{-1}(\boldsymbol{x})) \prod_{m=1}^{M} \left|\det \boldsymbol{J}_{(f_\theta^m)^{-1} \to z_m}\right| \tag{65}$$

Planar Flows. Invertible transformation

$$\boldsymbol{x} = \boldsymbol{f}_\theta(\boldsymbol{z}) = \boldsymbol{z} + \boldsymbol{u} h\left(\boldsymbol{w}^T \boldsymbol{z} + b\right) \tag{66}$$

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

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

Note that we need to restrict parameters and non-linearity to ensure the mapping is invertible[17].

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

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

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 $\boldsymbol{z} = \boldsymbol{z}_\perp + \boldsymbol{z}_\parallel$, defined relative to $\boldsymbol{w}$, i.e. such that $\boldsymbol{z}_\perp^T \boldsymbol{w} = 0$ and $\boldsymbol{z}_\parallel = \alpha \frac{\boldsymbol{w}}{||\boldsymbol{w}||^2}$ for some $\alpha \in \mathbb{R}$. This gives

$$f(\boldsymbol{z}) = \boldsymbol{z}_\perp + \boldsymbol{z}_\parallel + \boldsymbol{u}\tanh\left(\boldsymbol{w}^T \boldsymbol{z}_\parallel + b\right) \tag{70}$$

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

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

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

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

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

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

$$m(x) = -1 + \log(1 + e^x) \tag{75}$$

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

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

✓ **Exact likelihood evaluation** via inverse transformation $\boldsymbol{x} \mapsto \boldsymbol{z}$ and c.o.v. formula (instead of intractable summation over all $\boldsymbol{z}$).

✗ Requires evaluation of $\det \boldsymbol{J}$, where J is an $n \times n$ Jacobian[18]. Computing such a determinant in general is $\mathcal{O}(n^3)$.

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

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

$$\boldsymbol{z} \sim p_Z(\boldsymbol{z}) \qquad \boldsymbol{x} = \boldsymbol{f}_\theta(\boldsymbol{z}) \tag{78}$$

$\theta = \{\boldsymbol{w}, \boldsymbol{u}, b\}$

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

---

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

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

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.

NICE[19] **[9:00]** composes two kinds of invertible transformations:

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

$$[\boldsymbol{z} \mapsto \boldsymbol{x}] \qquad \boldsymbol{x}_{1:d} = \boldsymbol{z}_{1:d} \tag{79}$$

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

$$[\boldsymbol{x} \mapsto \boldsymbol{z}] \qquad \boldsymbol{z}_{1:d} = \boldsymbol{x}_{1:d} \tag{81}$$

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

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

$$\boldsymbol{J}_{\boldsymbol{x}\to\boldsymbol{z}} \triangleq \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{z}} = \begin{pmatrix} \boldsymbol{I}_d & \boldsymbol{0} \\ \frac{\partial \boldsymbol{x}_{d+1:n}}{\partial \boldsymbol{z}_{1:d}} & \boldsymbol{I}_{n-d} \end{pmatrix} \tag{83}$$

$$\det J = 1 \tag{84}$$

  Volume preserving transformation since det is 1 **[17:17]**.
- **Rescaling layers** (final layer). The forward/inverse mappings and J:

$$[\boldsymbol{z} \mapsto \boldsymbol{x}] \qquad x_i = s_i z_i \tag{85}$$

$$[\boldsymbol{x} \mapsto \boldsymbol{z}] \qquad z_i = \frac{x_i}{s_i} \tag{86}$$

$$\boldsymbol{J}_{\boldsymbol{x}\mapsto\boldsymbol{z}} = \operatorname{diag}(\boldsymbol{s}) \tag{87}$$

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

Real-NVP[21]. Coupling layers now shift *and* scale.

$$[\boldsymbol{z} \mapsto \boldsymbol{x}] \qquad \boldsymbol{x}_{1:d} = \boldsymbol{z}_{1:d} \tag{88}$$

$$\boldsymbol{x}_{d+1:n} = \boldsymbol{z}_{d+1:n} \odot \exp\left(\alpha_\theta\left(\boldsymbol{z}_{1:d}\right)\right) + \mu_\theta(\boldsymbol{z}_{1:d}) \tag{89}$$

$$[\boldsymbol{x} \mapsto \boldsymbol{z}] \qquad \boldsymbol{z}_{1:d} = \boldsymbol{x}_{1:d} \tag{90}$$

$$\boldsymbol{z}_{d+1:n} = \left(\boldsymbol{x}_{d+1:n} - \mu_\theta(\boldsymbol{x}_{1:d})\right) \odot \left(\exp\left(-\alpha_\theta\left(\boldsymbol{x}_{1:d}\right)\right)\right) \tag{91}$$

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

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

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

[19] Nonlinear Independent Components Estimation (Dinh et al., 2014).
[20] In practice, we assign d randomly for each layer.
[21] Non-volume preserving extension of NICE.

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

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

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

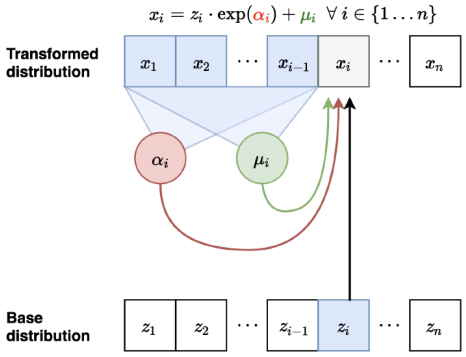
where both μ_i and α_i are functions (neural nets) of $x_1, \ldots x_{i-1}$ (constants for i=1). The sampling procedure is defined as follows:

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

$$D_{KL}(s||t) = \mathbb{E}_{\boldsymbol{x} \sim s}\left[\log s(\boldsymbol{x}) - \log t(\boldsymbol{x})\right] \tag{97}$$

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

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

$$p_Z(\boldsymbol{z}) \triangleq \frac{\partial P(Z \leq \boldsymbol{z})}{\partial \boldsymbol{z}} \tag{98}$$

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

$$P(X \leq \boldsymbol{x}) = P\left(Z \in \{\boldsymbol{z} | \boldsymbol{f}(\boldsymbol{z}) \leq x\}\right) \tag{99}$$

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

$$\begin{aligned}
p_X(\boldsymbol{x}) &\triangleq \frac{\partial P(X \leq \boldsymbol{x})}{\partial \boldsymbol{x}} && (100)\\
&= \frac{\partial P\left(Z \in \{\boldsymbol{z} | \boldsymbol{f}(\boldsymbol{z}) \leq \boldsymbol{x}\}\right)}{\partial \boldsymbol{x}} && (101)\\
&= \frac{\partial P\left(Z \in \{\boldsymbol{z} | \boldsymbol{f}(\boldsymbol{z}) \leq \boldsymbol{x}\}\right)}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} && (102)\\
&= \frac{\partial P\left(Z \leq \boldsymbol{f}^{-1}(\boldsymbol{x})\right)}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} && (103)\\
&= p_Z(\boldsymbol{f}^{-1}(\boldsymbol{x})) \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} && (104)\\
&= p_Z(\boldsymbol{f}^{-1}(\boldsymbol{x})) \boldsymbol{J}_{\boldsymbol{z} \to \boldsymbol{x}} && (105)
\end{aligned}$$

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

$$p_X(\boldsymbol{x}) = p_Z\left(\boldsymbol{z} = \boldsymbol{f}^{-1}(\boldsymbol{x})\right) \tag{106}$$

TODO: FINISH THIS AFTER THE STATS BOOKS ARRIVES

[a] This part is intuitively clear to me, yet I'm not sure how I'd actually *prove* it.

1.4.1 Homework 3

Problem 1: Flow Models. Implement a **Masked Autoregressive Flow** (MAF) model. MAF models are themselves composed of **Masked Autoregressive Distribution Estimator** (MADE) blocks (each $\boldsymbol{f}(\cdot)$ 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** $[\boldsymbol{z} \mapsto \boldsymbol{x}]$: Assuming that $\boldsymbol{z}$ comes from some base noise distribution like $\mathcal{N}(0, I)$ (**is this still valid when e.g. $\boldsymbol{z}$ 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:

$$x_1 = \mu_1 + z_1 \exp(\alpha_1) \tag{107}$$
$$x_2 = \mu_2(x_1) + z_2 \exp(\alpha_2(x_1)) \tag{108}$$
$$\vdots \tag{109}$$
$$x_n = \mu_2(x_1, \ldots, x_{n-1}) + z_n \exp(\alpha_n(x_1, \ldots, x_{n-1})) \tag{110}$$

  Since this is an invertible transformation,

$$\log \left| \det \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} \right| = \log \left| \det \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{z}} \right|^{-1} \tag{111}$$
$$= \log \left| \prod_{i=1}^{n} \exp(\alpha_i) \right|^{-1} \tag{112}$$
$$= -\sum_{i=1}^{n} \alpha_i \tag{113}$$

  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.

Lectures **October 21, 2019**

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

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

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

Consider a noise mixture model $p_\theta(\boldsymbol{x})$ where $\boldsymbol{x}$ 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 $\boldsymbol{x} \sim p_{data}(\boldsymbol{x})$.
3. Else, return sample $\boldsymbol{x} \sim p_{noise}(\boldsymbol{x})$.

Note that the above procedure can be written as

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

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

$$\log p_\theta(\boldsymbol{x}) = \log\left(0.01 p_{\text{data}}(\boldsymbol{x}) + 0.99 p_{\text{noise}}(\boldsymbol{x})\right) \tag{115}$$

$$\geq \log 0.01 p_{\text{data}}(\boldsymbol{x}) \tag{116}$$

$$= \log p_{\text{data}}(\boldsymbol{x}) - \log 100 \tag{117}$$

$$\mathbb{E}_{p_{data}}\left[\log p_\theta(\boldsymbol{x})\right] \geq \mathbb{E}_{p_{data}}\left[\log p_{\text{data}}(\boldsymbol{x})\right] - \log 100 \tag{118}$$

$$\mathbb{E}_{p_{data}}\left[\log p_{\text{data}}(\boldsymbol{x})\right] \geq \mathbb{E}_{p_{data}}\left[\log p_\theta(\boldsymbol{x})\right] \tag{119}$$

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

$$\mathbb{E}_{p_{data}}\left[\log p_{\text{data}}(\boldsymbol{x})\right] \geq \mathbb{E}_{p_{data}}\left[\log p_\theta(\boldsymbol{x})\right] \geq \mathbb{E}_{p_{data}}\left[\log p_{\text{data}}(\boldsymbol{x})\right] - \log 100 \tag{120}$$

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

$$\mathbb{E}_{p_{data}}\left[\log p_\theta(\boldsymbol{x})\right] \approx \mathbb{E}_{p_{data}}\left[\log p_{\text{data}}(\boldsymbol{x})\right] \tag{121}$$

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

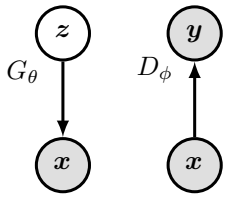
[a] Think about it. Higher dimensional spaces have more unique possible values for $\boldsymbol{x}$. For higher dimensional spaces, the observed points $\boldsymbol{x}$ represent a much smaller fraction of the total number of points, and will thus likely get assigned lower probabilities (and thus larger $|\log p|$) 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 $S_1 = \{\boldsymbol{x} \sim P\}$ and $S_2 = \{\boldsymbol{x} \sim Q\}$, 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** $G_\theta : \boldsymbol{z} \mapsto \boldsymbol{x}$. Deterministic directed LVM. **Minimizes** the two-sample test objective in support of $H_0 : p_{data} = p_\theta$.
- **Discriminator** $D_\phi : \boldsymbol{x} \mapsto \boldsymbol{y}$ where $\boldsymbol{y}$ 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.

[22] We 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

[23] The 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]

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

Optimal Discriminator

The optimal discriminator function D for a given generator G is

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

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

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

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

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

The GAN Training Algorithm [52:00]

1. Sample minibatch of m training points from data.
$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)} \sim \mathcal{D}$$
2. Sample minibatch of m noise vectors from p_Z.
$$\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \ldots, \boldsymbol{z}^{(m)} \sim p_Z$$
3. Update generator parameters θ by stochastic gradient **descent**.
$$\nabla_\theta V(G_\theta, D_\phi) = \frac{1}{m}\nabla_\theta \sum_{i=1}^{m} \log\left(1 - D_\phi\left(G_\theta\left(\boldsymbol{z}^{(i)}\right)\right)\right) \tag{130}$$
4. Update discriminator parameters ϕ by stochastic gradient **ascent**.
$$\nabla_\phi V(G_\theta, D_\phi) = \frac{1}{m}\nabla_\phi \sum_{i=1}^{m} \left[\log D_\phi\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D_\phi\left(G_\theta\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right] \tag{131}$$
5. Repeat for fixed number of epochs.

[24] Properties of $D_{JSD}\left(p||q\right)$:

$$D_{JSD}\left(p||q\right) \geq 0 \tag{126}$$

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

$$D_{JSD}\left(p||q\right) = D_{JSD}\left(q||p\right) \tag{128}$$

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

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.

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

f-divergence [8:30]

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

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

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

Fenchel Conjugate (Complex Conjugate)

*For any function[26] $f(\cdot)$, its **Fenchel conjugate** (a.k.a. the complex conjugate) is defined as[27]*

$$f^*(t) \triangleq \sup_{u\in dom_f}\left(ut - f(u)\right) \tag{134}$$

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

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

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

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).

[26] Technically, 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.

[27] Recall 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 samples[28]. We can obtain a lower bound that is likelihood-free wrt p and q to any f-divergence via its Fenchel conjugate **[20:00]**.

$$
\begin{aligned}
D_f\left(p||q\right) &\triangleq \mathbb{E}_{\boldsymbol{x}\sim q}\left[f\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right)\right] && (135)\\
&= \mathbb{E}_{\boldsymbol{x}\sim q}\left[f^{**}\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right)\right] && (136)\\
&= \mathbb{E}_{\boldsymbol{x}\sim q}\left[\sup_{t\in \mathrm{dom}_{f^*}}\left(t\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} - f^*(t)\right)\right] && (137)\\
&:= \mathbb{E}_{\boldsymbol{x}\sim q}\left[T(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} - f^*\left(T(\boldsymbol{x})\right)\right] && (138)\\
&= \int \mathrm{d}\boldsymbol{x}\left[T(\boldsymbol{x})p(\boldsymbol{x}) - f^*\left(T(\boldsymbol{x})\right)q(\boldsymbol{x})\right] && (139)\\
&\geq \sup_{\hat{T}\in\mathcal{T}}\int \mathrm{d}\boldsymbol{x}\left[\hat{T}(\boldsymbol{x})p(\boldsymbol{x}) - f^*\left(\hat{T}(\boldsymbol{x})\right)q(\boldsymbol{x})\right] && (140)\\
&= \sup_{\hat{T}\in\mathcal{T}}\left(\mathbb{E}_{\boldsymbol{x}\sim p}\left[\hat{T}(\boldsymbol{x})\right] - \mathbb{E}_{\boldsymbol{x}\sim q}\left[f^*\left(\hat{T}\left(\boldsymbol{x}\right)\right)\right]\right) && (141)
\end{aligned}
$$

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

$$
\begin{aligned}
D_f\left(p||q\right) &\geq \sup_{\hat{T}\in\mathcal{T}}\left(\mathbb{E}_{\boldsymbol{x}\sim p}\left[\hat{T}(\boldsymbol{x})\right] - \mathbb{E}_{\boldsymbol{x}\sim q}\left[f^*\left(\hat{T}\left(\boldsymbol{x}\right)\right)\right]\right) && (142)\\
\min_\theta\max_\phi F(\theta,\phi) &= \mathbb{E}_{\boldsymbol{x}\sim p_{data}}\left[T_\phi(\boldsymbol{x})\right] - \mathbb{E}_{\boldsymbol{x}\sim p_{G_\theta}}\left[f^*\left(T_\phi\left(\boldsymbol{x}\right)\right)\right] && (143)
\end{aligned}
$$

where the interpretations of ϕ (discriminator) and θ (generator) are consistent with our original GAN formulation[29]

[28] Why can't we already? Technically, all MLE applications minimize $D_{KL}\left(p_{data}||p_\theta\right)$ 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)

[29] For convenience:

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

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

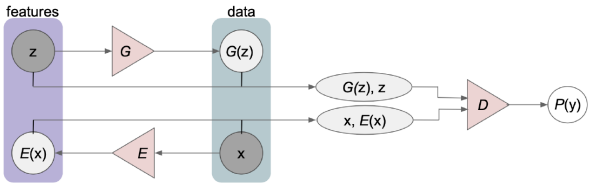
- Normalizing flow models: $\boldsymbol{z} = \boldsymbol{f}_\theta^{-1}(\boldsymbol{x})$.
- VAEs: $\boldsymbol{z} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$.

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

- [Use D] For any point $\boldsymbol{x}$, use the activations of the prefinal layer of D as feature representation[30]
- [Change learning algorithm] Compare samples of $(\boldsymbol{z}, \boldsymbol{x})$ (instead of just $\boldsymbol{x}$) from joint distributions of model/data. Note that we don't have access to $\boldsymbol{z}$ for the $\boldsymbol{x}$ sampled from p_{data}.

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

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



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 $\mathcal{X} \rightleftarrows \mathcal{Y}$ in an unsupervised manner? **CycleGAN** learns two parameterized conditional generative models:

$$G : \mathcal{X} \mapsto \mathcal{Y} \qquad F : \mathcal{Y} \mapsto \mathcal{X} \tag{145}$$

$$F(G(X)) \approx X \qquad G(F(Y)) \approx Y \tag{146}$$

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

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

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

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

[30] **What? How is this even remotely a valid approach? What does the prefinal layer have to $\boldsymbol{z}$??**

Energy-Based Models I

Written by Brandon McKinzie

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

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

$$\int g_\theta(x)\mathrm{d}x = 1 \tag{151}$$

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.

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

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

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

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

$$Z(\theta) = \int \exp\left(f_\theta\left(x\right)\right) \mathrm{d}x \tag{155}$$

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 $Z(\theta)$:

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

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

Concepts **October 20, 2019**

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:

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

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 $0 \leq \lambda \leq 1$:*

$$f\left(\lambda x_1 + (1-\lambda)x_2\right) \leq \lambda f(x_1) + (1-\lambda) f(x_2) \tag{158}$$

$$f\left(\mathbb{E}\left[x\right]\right) \leq \mathbb{E}\left[f(x)\right] \qquad \textit{[Jensen's Inequality]} \tag{159}$$

Observe that log is a **concave function**. In other words, $\log(\mathbb{E}\left[x\right]) \geq \mathbb{E}\left[\log x\right]$. This means that $\mathbb{E}\left[\log x\right]$ is a *lower bound* on $\log \mathbb{E}\left[x\right]$. 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 $\mathcal{L}(x; \theta, \phi)$, as[31]

$$\begin{aligned}
\mathcal{L}(x;\theta,\phi) &\triangleq \mathbb{E}_{z\sim q(z)} \left[\log \frac{p_\theta(x, z)}{q(z)} \right] && (166)\\
&= \mathbb{E}_{z\sim q(z)} \left[\log p_\theta(x, z)\right] + H(q) && (167)\\
&= \log p_\theta(x) - D_{KL}\left(q(z)||p_\theta(z \mid x)\right) && (168)\\
&= \mathbb{E}_{z\sim q(z)} \left[\log p_\theta(x \mid z)\right] - D_{KL}\left(q(z)||p(z)\right) && (169)\\
\log p_\theta(x) &\geq \mathcal{L}(x;\theta,\phi) && (170)
\end{aligned}$$

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

$$\begin{aligned}
D_{KL}\left(q(z)||p_\theta(z \mid x)\right) &\triangleq \mathbb{E}_q \left[\log \frac{q(z)}{p_\theta(z \mid x)} \right] && (160)\\
&= -H(q) - \mathbb{E}_q \left[\log p_\theta(z \mid x)\right] && (161)\\
&= -H(q) - \mathbb{E}_q \left[\log p_\theta(x, z)/p_\theta(x)\right] && (162)\\
&= -H(q) - \mathbb{E}_q \left[\log p_\theta(x, z)\right] + \mathbb{E}_q \left[\log p_\theta(x)\right] && (163)\\
&= -H(q) - \mathbb{E}_q \left[\log p_\theta(x, z)\right] + \log p_\theta(x) && (164)\\
&= -\mathcal{L}(x;\theta,\phi) + \log p_\theta(x) && (165)
\end{aligned}$$

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 $D_{KL}\left(q(z)||p_\theta(z \mid x)\right) = 0$. 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 $p_\theta(z \mid x)$.
- **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 $D_{KL}\left(q(z)||p(z)\right) \to 0$. Doesn't this just encourage our encoder to learn $p(\boldsymbol{z})$? How is that even useful (since we already know what $p(\boldsymbol{z})$ is? What if we just use $p(\boldsymbol{z})$ directly?)
  - **A**: yo good question idk tho