CS 236 Winter 2023
Deep Generative Models
Mid Term Notes

November 13, 2023

SUNet ID: jchan7
Name: Jason Chan

Contents

1 Executive summary

The design space for generative models. Probability distributions p(x) are a key building block in generative modelling, (1) must be non-negative and (2) sum to one (normalised). Enforcing constraint (2) is actually very difficult and results in 'strange' model architectures.

Table 1: Executive Summary: Generative Models Comparison

Aspect	Autoregressive	VAE	Flow	GAN	Energy
Pros	• Explicit factorization of joint distribution • High-quality generatio	• Model features • Regularized by design	• Exact likelihood evaluation • Invertible transformations	• high-quality samples • No explicit density required	• Extreme flexibility • unsupervised and supervised learning
Cons	• Slow generation (sequential)	• Might produce blurry samples • Requires careful design of latent space	• Expensive • Requires careful design of transformations	• Mode collapse issues • Requires careful balancing of loss terms	• Requires careful choice of architecture and energy fn • sampling and learning is hard
Likelihood Est.	Exact	-	Exact	NA	
Loss $d(\cdot)$	Negative log-likelihood	ELBO (reconstruction + KL divergence)	Negative log-likelihood with change of variables	Min-max game (discriminator vs generator loss)	Energy function difference between data and generated samples
Learn features?	No	Yes by design	Yes by design f^{-1}	Yes with mods bi-GAN	Yes with mods
Sampling	Sequential, can be slow	From latent space, faster	Parallelizable, efficient	From random noise, efficient	MCMC-based, can be slow
Learning	RNNs, PixelCNN, etc.	Encoder-decoder architecture with stochastic units	Stacked invertible transformations	Generator and discriminator networks	Typically feedforward with specific energy function designs

2 Fundamentals

2.1 Joint, conditional, marginal probability

- Joint Probability $P(A, B)$: Probability both events A and B happen. Think of it as an intersection: where both A and B overlap.
- Conditional Probability $P(A|B)$: Probability event A happens given B has occurred. It's like asking, "Now that B happened, what's the chance of A?"
- Marginal Probability $P(A)$: Probability event A happens, irrespective of B. It's like summing (or "marginalizing") over all possibilities of B to focus solely on A.

$$P(A|B) = \frac{P(A,B)}{P(B)} \quad \text{given } P(B) \neq 0$$
$$P(A,B) = P(A|B) \times P(B)$$
$$P(A) = \sum_{\text{all } B} P(A,B)$$

2.2 Probability vs. Likelihood

Given a parameter θ, the **probability** describes how likely the data X is. *Given this model (with fixed parameters), what's the probability of observing this particular data?*. Given some observed data X, the **likelihood** tells us how likely different parameter values θ are *Given this observed data, how likely is this particular parameter value?*.

2.3 Probability Density Functions (PDFs)

Univariate PDF

$$p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1}$$

Multivariate PDF

$$p(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{D}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{2}$$

2.4 Partition Function

The partition function ensures that the probability distribution defined by the model is normalized (sums to 1 for discrete, integrates to 1 for continuous).

2.5 Information theory

Information theory provides a quantitative measure of information content, known as entropy. For a discrete probability distribution P, its entropy H(P) is given below. Entropy captures the average number of bits needed to encode the outcomes of the distribution, assuming an optimal coding scheme.

$$H(P) = -\sum_i P(x_i) \log P(x_i) \tag{3}$$

2.6 KL divergence

In essence, KL divergence can be viewed as measuring the "compression error" or inefficiency in using one distribution to represent or approximate another. A high KL divergence indicates that Q is a poor approximation, and you'd use many extra bits encoding P with a Q-optimized code.It is scalar and always greater than zero and not symmetric. In practical applications, we approximate KL given p_{data} by Monte Carlo sampling from the data.

Analytical solutions are available for two Gaussian distributions.

$$D_{KL}(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)} \tag{4}$$

2.7 Jensen's Inequality

The below is true for $f = ln(X)$, which is concave. If the function were convex then the inequality changes sign and the terms change sides.

$$f(\mathbb{E}[(X)]) \geq \mathbb{E}[f(X)] \tag{5}$$

$$log(\mathbb{E}[f(X)] \geq \mathbb{E}[logf(X)] \tag{6}$$

2.8 Maximum Likelihood Estimation

Minimising KL divergence is equivalent to Maximum Likelihood Estimation (MLE). The goal is to find the model parameters that maximises the probability that the model generates inputs from the dataset. MLE corresponds to a compression problem! MLE takes the log of the probability to (1) avoid the underflow problem of multiplying small numbers, which doesn't change the optimisation problem, (2) simplifies the derivative, (3) is convex so there's a global optimum

$$\hat{\theta} = \arg\max_\theta \sum_{i=1}^{M} \log p_\theta(\mathbf{x}_i) \quad \text{where } x_1, x_2, ..., x_M \sim p_{data}(\mathbf{x}) \tag{7}$$

2.8.1 Why MLE?

1. Optimal statistical efficiency: convergence is fasters among all statistical methods. Assuming sufficient model capacity, then there exitss a unique set of optimal parameters that satisfies $p^*_\theta = p$
2. Higher likelihood = better lossless compression.

2.8.2 No free lunch

1. MLE tells you nothing about the quality of the generated samples. They may not be pretty! Especially true for imperfect models (Theis et al. 2016)
   (a) Edge case: great log-likelihoods but poor samples. How? Risk increasers as x increases to very high dimensions.
   (b) Edge case: great samples but poor test log-liklihoods. How? Memorise the dataset!

2.9 Likelihood Free

2.9.1 Minimise Two Sample Test Objective

How to tell if a finite set of samples in one batch are generated from the same distribution as those in a second batch. Null hypothesis: they are from the same distribution. Alternative hypothesis: they are from a different distribution.

2.10 Test statistic

1. Compare the variances of both batch.

2.11 Cross Entropy

In classification, minimizing CE is equivalent to maximizing the likelihood of the observed data under the model. CE can be decomposed into the entropy of the true distribution and the KL divergence between the true and predicted distributions. In scenarios where the true distribution's entropy is constant (like fixed ground-truth labels in classification), minimizing cross entropy is equivalent to minimizing the KL divergence between the true and predicted distribution

$$H(p, q) = -\sum_{x} p(x) \log q(x) \tag{8}$$

2.12 Jacobians vs. Gradients

Gradients applies to scalar valued functions and outputs vector. Jacobians apply to vector valued functions and outputs matrix. Gradients tells us the direction in the domain where the function increases most rapidly, while Jacobian offers insight into the local behavior of vector-valued functions by providing a linear approximation around a particular point.
Jacobian, its relationship to derivatives, starting from the basics and gradually building up

1. Single variable calculus. One variable in one function. Derivate measures rate of change of a function. Akin to a single path.
2. multivariable calculus. More than one variable in one function. Instead of single path, now have multiple paths. Partial derivatives describe the slope in each direction.
3. jacobian. multiple functions. Jacobian is a collection of all the possible partial derivatives for each function. It creates a linear transformation matrix of how that transformation behaves near a particular point. It's utility is to tell us how small changes in input changes the output

- Scalar valued functions

$$
\begin{aligned}
f(x) &= x^2 + 2x + 1 \\
f(x, y) &= 3x^2 + 2y - 7 \\
\mathbf{v} &= [v_1, v_2, \ldots, v_n] \\
||\mathbf{v}|| &= \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}
\end{aligned}
$$

- Vector valued functions

$$
\begin{aligned}
\mathbf{r}(t) &= \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} t\sin(t) \\ t\cos(t) \\ t \end{bmatrix} \\
\mathbf{f}(x, y) &= \begin{bmatrix} x \\ y \\ x^2 + y^2 \end{bmatrix} \\
\mathbf{p}' &= R\mathbf{p}
\end{aligned}
$$

3 Autoregressive

An autoregressive model uses its own past values to predict future values.

$$p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)p(x_4|x_3, x_2, x_1) \tag{9}$$

Which is compactly represented as

$$p_\theta(\mathbf{x}) = \prod_{i=1}^{n} p_\theta(x_i|\mathbf{x}_{<i}) \tag{10}$$

- Non-binary Discrete variables e.g Pixel intensities from 0 to 255. Use softmax.
- Continuous e.g. speech. Model as mixture of K Gaussians.

3.1 Pros

1. Easy to sample from
2. Easy to compute probability
3. Easy to extend to continuous variables by choosing gaussian conditionals $p(x_t|x_{<t}) = \mathcal{N}(\mu_\theta(x_{<t}), \Sigma_\theta(x_{<t}))$ or a mixture of logistics

3.2 No free lunch

1. Exponential size.
   (a) Bayes net tries to simplify this by assuming that the conditional probabilities only depend on one previous node.
   (b) Neural models assumes specific functional form for the conditionals. A sufficiently deep neural net can approximate any function.
2. Can't learn features

3.3 Variations

- Fully Visible Sigmoid Belief Network (FVSBN) 2015. The conditional variables are bernoulli with parameters. Based on logistic regression.
- Neural Autoregressive Density Estimation (NADE) 2011. Replaced logistic regression with one layer neural network. Ties weights to reduce the number of parameters and speed up computation.
- RNADE

- Masked Autoencoder for Distribution Estimation (MADE) 2015. An autoencoder that is autoregressive. Note that autoencoders aren't autoregressive themselves nor are they generative. An autoencoder learns a feed-forward hidden representation h(x) of if its input via neural network. with one linear layer, an activation function and finally a sigmoid function. MADE modifies this to build an autoregressive model, which requires a DAG structure.
- RNN. Issues with RNN: a single hidden vector needs to encode a growing history. Sequential evaluation cannot be parallelised. Exploding/vanishing gradients when acessing information from many steps back.
- Attention based model. Compare a query vector to set of key vectors.
- GPT replaces RNN with transformer. Needs a masked self-attention to preserve autoregressive structure.
- Pixel RNN 2016. Model images pixel by pixel using raster scan order.
- Pixel CNN 2016. Use convolutional architecture to predict next pixel given context (a neighbourhood of pixels). Much faster than Pixel RNN.
- Pixel Defend 2018. Train a generative model p(x) on clean inputs, Pixel CNN. Given new input x*, evaluate p(x*).
- WaveNet 2016.

4 Latent Variable

4.1 VAE

1. Reparamerterize trick. Doesn't work if p(z) is bernoulli only normal.
2.

5 Normalising Flow

Transform simple distributions Z (easy to sample and evaluate densities) into more complex distributions via change of variables to match our data, X. Notation convention: the forward pass maps Z to X via f_θ that is $x = f_\theta(z)$ and the backwards pass maps X to Z via f_θ^{-1} that is $z = f_\theta^{-1}(x)$. There are computational tradeoffs in evaluating forward and inverse transformations.

Why is it called normalising? We need to ensure the sum of the mass equals one after the transformation. We rescale with the absolute determinant of the jacobian. The determinant of a jacobian returns a scalar value. This scalar value tells us about the magnitude and orientation of change of a multivariable multiple function mapping between inputs and outputs. Intuitively, linearize near data point x, and determinant tells us how much that volume expands or contracts around that datapoint. Jacobian of transformations should have tractable determinant for efficient learning and density estimation.

The likelihood can easily be found using the change of variable formula: invert img, x to compute corresponding z, compute how likely z is under the prior $p(z)$.

$$p_X(x;\theta) = p_Z(f_\theta^{-1}(x))\Big|\det\big(\frac{\partial f_\theta^{-1}(x)}{\partial x}\big)\Big| \tag{11}$$

5.1 Pros

1. Composibility: Invertible transformations can be composed with each other $z_m = f_\theta^m \cdot ... \cdot f_\theta^1(z_0)$, which is the deep learning idea of stacking layers f_θ. Note on notation θ in this case is the union of all the parameters for all of the layers. Start with simple distribution for z_0 e.g. Gaussian. Apply a sequence of M invertible transformations to finally obtain $x = z_m$. By change of variables

$$p_X(x;\theta) = p_Z(f_\theta^{-1}(x)) \prod_{m=1}^{M} \Big|\det \frac{\partial f_\theta^{-1}(z_m)}{\partial z_m}\Big| \tag{12}$$

2. Learning is easy because the maximum likelihood is exact. Just take the log of the probability formula.

$$\max_\theta \log p_X(\mathcal{D};\theta) = \sum_{x\in\mathcal{D}} \log p_Z(f_\theta^{-1}(x)) + \log|\det(\frac{\partial f_\theta^{-1}(x)}{\partial x})| \tag{13}$$

3. Sampling is easy via forward transformation $z \to x$

4. Latent representations are available and inferred via inverse transformation. It's questionable whether Normalising Flows are latent representations but has not compressive qualities.

5.2 No Free Lunch

1. X and Z need to have the same dimensionality. There is no compression unlike VAE.
2. X and Z need to be continuous. Therefore can't apply Normalising Flows to LLMs, which are discrete.
3. Invertible transformations need to be tractable
4. Jacobians need to be tractable. Key idea: choose transformations, whose Jacobian is triangular because the determinant of a triangular matrix is the product of its diagonals, which is an O(n) linear time operation.

5.3 Designing Invertible Transformation

1. NICE 2014. Volume preserving.
2. Real-NVP 2017. Non-volume preserving extension of NICE.
3. Continuous Autoregressive models as flow models
   (a) Inverse Autoregressive Flow 2016
   (b) Masked Autoregressive Flow 2017
4. Parallel Wavenet. Two part teacher and student model. Teacher parameterised by MAF. Teacher can be efficiently trained via MLE. Student model is parameterised by IAF. Improves sampling efficiency over original wavenet (vanilla autoregressive model) by 1000x.
5. I-Resnet 2018
6. Glow 2018
7. MINTNet 2019. Build invertible neural networks with masked convolutions like in autoregressive models like PixelCNN. Regular CNN is powerful but is not invertible and its Jacobian is expensive. Because of the ordering the Jacobian matrix is triangular.
8. Gaussian Flows 2020

6 GAN

Autoregressive, normalising flows and VAEs have restricted model architectures and are all trained by minimising KL divergence or equivalently maximising likelihoods (or their approximations). They are trained this way because they have a way to evaluate the probability of a specific data point

GANs, on the other hand, is on the other extreme, tries to have as much flexibility in model family. GANs define the probability distribution implicitly by instead defining the sampling procedure. Likelihood free training. Setting up a minimax game.

6.1 Pros

1. Very flexible model architectures
2. Likelihood free training. Two sample test objective learns by samples only (likelihood free). Wide range of two-sample test objectives covering f-divergences and Wasserstein distance (and more)
   (a) Under ideal conditions, the two sample statistic is equivalent to choosing the Jenson-Shannon Divergence
   (b) f-divergence. Is a generic positive definite scalar score that quantifies the differences between generated and real data distributions. It is the general case of KL divergence and Jenson-Shannon Divergence. Choose any f is any convex, lower-semi-continuous function with $f(1) = 0$. Need to rewrite

$$D_f(p, q) = \mathbb{E}_{x\sim q}\Big[f(\frac{p(x)}{qx})\Big] \tag{14}$$

   (c) Wasserstein (Earth-Mover) distance
3. Fast sampling (single forward pass)
4. Can train on negative data, i.e. the NOT cases.

6.2 No Free Lunch

Very difficult to train in practice. Largely given up today but it's a powerful idea that may return.

1. likelihood is intractable
2. training is unstable because of minimax optimisation
3. hard to evaluate
4. mode collapse issues
5. Need lots of hacks to make it work

6.3 Training Objective for GANs

$$min_\theta max_\phi \mathbb{E}_{x\sim p_{data}}[logD_\phi(x)] + \mathbb{E}_{z\sim p(z)}[log(1 - D_\phi(G_\theta(z)))] \tag{15}$$

6.3.1 Training Objective for Discriminator

$$\max_{D_\phi} V(p_\theta, D_\phi) = \mathbb{E}_{x\sim p_{data}}[logD_\phi(x)] + \mathbb{E}_{z\sim p(z)}[log(1 - D_\phi(x))] \tag{16}$$

$$\approx \sum_{x\in S_1} \log D_\phi(x) + \sum_{x\in S_2} [\log(1 - D_\phi(x))] \tag{17}$$

For fixed generative model p_θ, the discriminator is performing binary classification with the cross entropy objective. Assign probability 1 to true data points from set S_1 and 0 from set S_2. Optimal discriminator.

$$D^*_\theta(x) = \frac{p_{data}(x)}{p_{data}(x) + p_\theta(x)} \tag{18}$$

The ultimate goal of GAN training is to see that the discriminator's accuracy drop to 50 per cent indicating that the generator has mastered the data distribution perfectly. Sanity check if $p_\theta = p_{data}$ classifier cannot do better than chance.

6.3.2 Training objective for Generator

Looks like flow model $x = G_\theta(z)$, where z is a simple prior like a Gaussian but G_θ can be an arbitrary neural network and doesn't need to be invertible. We don't care to evaluate likelihood $p_\theta(x)$ over x, which is implicitly defined.

$$\min_G \max_D V(G, D) = \mathbb{E}_{x\sim p_{data}}[\log D_(x)] + \mathbb{E}_{p\sim p_G)}[\log(1 - D(x)] \tag{19}$$

$$= \mathbb{E}_{x\sim p_{data}}[\log D_(x)] + \mathbb{E}_{p\sim p_G)}[\log(1 - D(x)] \tag{20}$$

$$= \mathbb{E}_{x\sim p_{data}}[\log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}] + \mathbb{E}_{p\sim p_G)}[\log(1 - \frac{p_{data}(x)}{p_{data}(x) + p_G(x)})] \tag{21}$$

We know that the optimal discriminator depends on G $D^*_G(\cdot)$

$$\min_G \max_D V(G, D) = \mathbb{E}_{x\sim p_{data}}[\log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}] + \mathbb{E}_{p\sim p_G)}[\log(1 - \frac{p_{data}(x)}{p_{data}(x) + p_G(x)})] \tag{22}$$

We also need to normalise the distributions by dividing by two, hence the need to adjust the expression with a $-\log 4$ term

$$\min_G \max_D V(G, D^*_G(x)) = \mathbb{E}_{x\sim p_{data}}[\log \frac{p_{data}(x)}{0.5(p_{data}(x)p_G(x)})] \tag{23}$$

$$+ \mathbb{E}_{p\sim p_G)}[\log(1 - \frac{p_{data}(x)}{0.5(p_{data}(x) + p_G(x)})] - \log 4 \tag{24}$$

$$= D_{KL}[p_{data}, \frac{p_{data} + p_G}{2}] + D_{KL}[p_G, \frac{p_{data} + p_G}{2}] - \log 4 \tag{25}$$

$$= 2D_{JSD}[p_{data}, \frac{p_{data} + p_G}{2}] - \log 4 \tag{26}$$

6.4 Jenson-Shannon Divergence

$D_{JSD} \geq 0$, is zero iff $p = q$ and is symmetric.

$$D_{JSD}[p, q] = \frac{1}{2}(D_{KL}[p, \frac{p+q}{2} + D_{KL}[q, \frac{p+q}{2}]) \tag{27}$$

6.5 Implications for Training

We choose $d(p_{data}, p_\theta)$ to be a two sample test statistic. We learn this statistic by training a classifier, which under ideal conditions is equivalent to choosing that statistic to be the $D_{JSD}[p,p_\theta]$. Two sample tests. Can (approximately) optimise f-divergences and the Wasserstein distance.

6.6 Intuition

6.6.1 How GANs Work

GANs consist of two main components: the Generator and the Discriminator. The Generator is trained to produce data that is similar to the real data. It takes random noise as input and generates samples that resemble the real data. The Discriminator is trained to distinguish between real data and the synthetic data produced by the Generator. During training, these two networks are in a game-theoretic contest: the Generator tries to produce increasingly realistic data, while the Discriminator improves its ability to differentiate real data from fake data.

6.6.2 Implicit Definition of Distribution in GANs

In GANs, the Generator implicitly defines the probability distribution of the data it generates. This is because the Generator learns to transform input noise (sampled from a simple, known distribution like a Gaussian) into outputs that resemble the real data. However, GANs do not provide an explicit formula for this transformed distribution. Instead, they provide a way to generate samples that follow this distribution. By sampling noise and passing it through the trained Generator, you get samples that reflect the learned distribution of real data. This is the "sampling procedure" the statement refers to.

6.6.3 Understanding the Distribution through Samples:

While we cannot directly write down the probability distribution the Generator is modeling, we can understand its characteristics by observing the generated samples. For instance, if a GAN is trained on a dataset of images of cats, we won't have an explicit formula for the probability distribution of "cat-like" images. But by feeding noise to the Generator, we can generate new images of cats that give us insights into what the GAN has learned about the distribution of cat images. In summary, GANs implicitly define the probability distribution of the data they are trained on by learning to generate samples from that distribution, without ever providing an explicit formula or function for the distribution itself. This approach

contrasts with traditional statistical methods where distributions are often explicitly defined and parameterized.

6.7 GAN variations

6.7.1 f-GAN Variational Divergence Minimisation

Start from any f-divergence you want and then derive a loss that looks like a GAN. If you start with Jensen-Shannon then you will directly get vanilla GAN loss, called the f-GAN objective

$$\min_{\theta} \max_{\phi} F(\theta, \phi) = \mathbb{E}_{x \sim p_{data}}[T_\phi(x)] - \mathbb{E}_{x \sim p_{G_\theta}}[f^*(T_\phi(x))] \tag{28}$$

1. Fenchel conjugate
2. Lowerbound to an f-divergence via Fenchel conjugate

6.7.2 Wasserstein GAN

WGANs use a modified training algorithm with weight clipping or gradient penalty to enforce a Lipschitz constraint, which is necessary for the theoretical properties of the Wasserstein distance to hold. This has led to a new family of GANs that are easier to train and that produce higher quality results.

1. Kantorovich-Rubinstein duality. Intuitively, this ensure that f cannot change too rapidly

6.7.3 BiGAN

Infer latent variables. Change training objective to introduce latent variable, z. Introduce an encoder network.

7 Energy Based Models

Autoregressive models a products of normalised objects. Latent variable models are a mixture of normalised objects, like VAEs. Their volumes are known analytically as a function of θ. Volumes are required to construct the normalisation constant to guarantee the probability mass functions sum to one. Energy Based Models (EBMs) do away with the need for analytically defined volumes / normalisation constants to open the choice of models families.

7.1 Pros

1. The most flexible because you can define anything for f_θ (a neural network), which expands the space of model families. But still need $p_\theta(x)$ to be non-negative and sums to one, which is true if we can divide by a normalisation constant $Z(\theta)$. Why exp? because we want to capture very large variations in probability. Log-probability is the natural scale we want to work with. It's akin to softmax. Softmax is easier to calculate than $Z(\theta)$. These are called energy models $-f_\theta(x)$ is called energy, configurations x with low energy are more likely.

$$p_\theta(x) = \frac{\exp(f_\theta(x))}{Z(\theta)} = \frac{\exp(f_\theta(x))}{\int \exp(f_\theta(x))dx} = \frac{\exp(f_\theta(x))}{\text{Volume of } \exp(f_\theta(x))} \tag{29}$$

2. Stable training
3. Relatively high sample quality
4. Composibility. You can compose many different generative models together, which allows you to do interesting things.

7.2 No free lunch

Curse of dimensionality. The fundamental issue is that compute Z_θ numerically (when no analytical solution is available) scales **exponentially** in the number of dimensions of x.

1. $Z(\theta)$ is intractable so no access to likelihood. There's two implications.
   (a) Avoid calculating the normalisation constant altogether in a subset of problems e.g. object recognition, denoising, sequence labelling that only require relative comparisons between two data points. The normalisation contact is eliminated because we take the ratio of probabilities $\frac{p_\theta(x')}{p_\theta(x)} = exp(f_\theta(x') - f_\theta(x))$. Examples:
      i. Isling model. "Old school denoising algorithm".
      ii. Product of experts. Ensemble multiple models e.g. PixCNN, flow etc. by multiplying their probabilities then normalizing them. This is similar to an AND operation. If any model says zero then the ensemble model says zero. Unlike mixture models, which behave more like OR.

$$p_{\theta_1,\theta_2,\theta_3}(x) = \frac{1}{Z(\theta_1,\theta_2,\theta_3)} q_{\theta_1}(x) r_{\theta_2}(x) t_{\theta_3}(x) \tag{30}$$

iii. Restricted Boltzmann Machine (RBM) energy model with latent variables. One of the first deep generative models that worked in 2009. Called Restricted because there's no connections between each visible-visible units and hidden-hidden units. Deep Boltzmann Machines stack many RBMs together (deep learning).

(b) Numerically approximate the normalisation constant using Monte Carlo Sampling.

2. Learning is hard. We want to maximise $p_\theta(x) = \frac{\exp(f_\theta(x_{train})}{Z(\theta)}$ for MLE but this is hard because the normalisation constant is required to evaluate $p_\theta(x)$, every time the parameters are changed, you need to "push down" the probability of all the "wrong" answers. We reframe our goal to maximise the log-likelihood of $max_\theta(f_\theta(x_{train}) - logZ(\theta))$. But to do so we need the gradient of the log-likelihood

   (a) Contrastive Divergence gives us a way of evaluating gradients of log likelihoods. Sample correct samples vs. incorrect samples from the model using Monte Carlo. But this technique requires knowing how to sample from the model.

$$\nabla_\theta f_\theta(x_{train}) - \nabla_\theta f_\theta(x_{sample}) \text{ where } x_{sample} \sim p_\theta(x)$$

3. Sampling is hard. Generating samples from EBM is very expensive because it needs a large number of iterations and convergence slows down exponentially in dimensionality. There simply isn't a direct way to sample like in autoregressive of flow models because we cannot easily compute how likely each possible sample is. Instead, we can compare two samples x and x'

   (a) Metropolis-Hastings Markov Chain Monte Carlo (MH-MCMC). Use iterative approach. In theory, x^T converges to $p_\theta(x)$ when $T \to \inf$ why? Satisfies the *detailed balance condition.*

      i. Initialise x^0 randomly, $t = 0$

      ii. Repeat for $t = 0, 1, 2, ..., T-1$

         A. Propose new sample $x' = x^t + noise$

         B. if $f_\theta(x') \geq f_\theta(x^t)$ set $x^{t+1} = x'$

         C. else set $x^{t+1} = x'$ with probability $exp(f_\theta(x') - f_\theta(x^t))$ otherwise set $x^{t+1} = x^t$.

   (b) Unadjusted Langevin MCMC. Better than vanilla MCMC

4. No feature learning (but can add latent variables)

7.3 Topics

1. Contrastive Divergence does not require estimating the partition function. Can also train some f-divergences (e.g. KL) to train an EBM without estimating the partition function.

8 Previous Exam Questions

8.1 MCMC based Training of Latent Variable Model

So far, we have seen how variational methods (i.e. ELBO maximization) can be used to train the latent variable model $p_\theta(x, z)$ where x is observed and z is latent. In this question, we shall consider a popular alternative called Markov Chain Monte Carlo (MCMC). For the purposes of this question, we shall simply treat MCMC as a black-box method that—with enough computation time—reliably allows us to sample from (but not compute!) the posterior $p_\theta(z|x)$. Fortunately, the ability to sample from the posterior (even if we cannot compute it) is sufficient for constructing an unbiased estimate of the gradient of the log-likelihood, thanks to the following identity,

$$\nabla_\theta \ln p_\theta(x) = \mathbb{E}_{p_\theta(z|x)}[\nabla_\theta \ln p_\theta(x, z)]. \qquad (5)$$

Prove this identity using the formula for the gradient of the logarithm function (log-derivative trick):

$$\nabla_\theta \ln p_\theta(x) = \frac{1}{p_\theta(x)} \cdot \nabla_\theta p_\theta(x).$$

Answer. There are two CRITICAL mathematical expresions you MUST know in machine learning. Do not pass go on this question if you don't know these.

1. $\nabla_\theta \ln p_\theta(x) = \frac{\nabla_\theta p(x)}{p_\theta(x)}$
2. $\nabla_\theta \ln p_\theta(x, z) = \frac{\nabla_\theta p(x,z)}{p_\theta(x,z)}$
3. $p_\theta(x) = \int p(x, z)dz$

Start with the log-derivative trick and try to show left hand side equals the equation we need to prove. Use Leibniz$_i$ntegral$_r$uletopushthegradientintotheintegral.
Expanding $p_\theta(x)$ into the full definition of a joint probability because the problem hints that this direction is more useful than applying the chain rule, which is for useful compositions of functions.

$$p_\theta(x) = \int p_\theta(x, z)dz$$

Taking the gradient with respect to θ:

$$\nabla_\theta p_\theta(x) = \nabla_\theta \int p_\theta(x, z)dz$$

Moving the gradient inside the integral:

$$\nabla_\theta p_\theta(x) = \int \nabla_\theta p_\theta(x, z)dz$$

Substituting this back into the log-derivative formula:

$$\nabla_\theta \ln p_\theta(x) = \frac{1}{p_\theta(x)} \cdot \int \nabla_\theta p_\theta(x, z)dz$$

Using the definition of expected value:

$$\mathbb{E}_{p_\theta(z|x)}[\nabla_\theta \ln p_\theta(x,z)] = \int \nabla_\theta \ln p_\theta(x,z) \cdot p_\theta(z|x) dz$$

Relating to the gradient of the log joint probability:

$$\nabla_\theta \ln p_\theta(x,z) = \frac{\nabla_\theta p_\theta(x,z)}{p_\theta(x,z)}$$

Thus:

$$\mathbb{E}_{p_\theta(z|x)}[\nabla_\theta \ln p_\theta(x,z)] = \int \frac{\nabla_\theta p_\theta(x,z)}{p_\theta(x,z)} \cdot p_\theta(z|x) dz$$

Applying Bayes' theorem, $p_\theta(z|x) = \frac{p_\theta(x,z)}{p_\theta(x)}$, and simplifying:

$$\int \frac{\nabla_\theta p_\theta(x,z)}{p_\theta(x,z)} \cdot \frac{p_\theta(x,z)}{p_\theta(x)} dz = \frac{1}{p_\theta(x)} \cdot \int \nabla_\theta p_\theta(x,z) dz$$

This is the same as:

$$\frac{1}{p_\theta(x)} \cdot \nabla_\theta p_\theta(x)$$

Which is the right-hand side of our original log-derivative formula:

$$\nabla_\theta \ln p_\theta(x) = \frac{1}{p_\theta(x)} \cdot \nabla_\theta p_\theta(x)$$

Therefore, we have shown that:

$$\nabla_\theta \ln p_\theta(x) = \mathbb{E}_{p_\theta(z|x)}[\nabla_\theta \ln p_\theta(x,z)]$$

8.2 VAE

True or false? Suppose we are training a VAE where the prior p(z) is such that each dimension of z is Bernoulli distributed. We can use reparameterization with z to get an unbiased estimate of the gradient of the variational objective function.

Answer. False. The reparameterization trick only applies to Gaussian continuous distributions. Bernoulli is discrete and needs to use other methods.
The **vanilla reparameterization trick** is used in Variational Autoencoders (VAEs) to enable backpropagation of gradients through stochastic nodes, specifically through the random latent variables in the model's architecture. The main conditions and the process are:

1. The latent variables, z, are sampled from a distribution that can be reparameterized using deterministic functions of the model parameters and an independent noise source.
2. A differentiable transformation exists that can take parameters output by the model and a source of noise to produce a sample from the latent variable's distribution.

3. Noise used for sampling is typically standard normal $\mathcal{N}(0, I)$, which does not depend on the model parameters. In summary, noise is introduced in the reparameterization trick to maintain the stochastic nature of the latent variables while allowing the network to be trained using gradient-based optimization by making the sampling process differentiable with respect to the model parameters.

For a normally distributed latent variable with mean μ and standard deviation σ, both of which are functions of the input data determined by the encoder, the latent variable z is sampled using the equation:

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

where ϵ is sampled from a standard normal distribution $\mathcal{N}(0, I)$, and $\odot$ denotes element-wise multiplication.
This reparameterization allows for gradients of the expected lower bound (ELBO) to be backpropagated with respect to the parameters, facilitating the use of gradient descent optimization methods in deep learning.

True or false? Suppose we have trained a VAE parameterised by θ and ϕ, we can obtain a sample from $p_(x)$ by first drawing a sample $z \sim p_\theta(z)$, then drawing another sample $x \sim p_\theta(x|z)$.

Answer. False because we first draw from $p_\phi(z|X)$ then sample $p_\theta(x|z)$

True or false? After learning a VAE model on a dataset. Alice gives Bob the trained decoder $p_\theta(x|z)$ and the prior $p(z)$ she used. However, she forgets to give Bob the encoder. Given sufficient computation can Bob still infer a latent representation for a new test point x?
True or false? After learning a VAE model on a dataset. Alice gives Bob the trained decoder $p_\theta(x|z)$ and the prior $p(z)$ she used. However, she forgets to give Bob the encoder. Given sufficient computation can Bob still infer a latent representation for a new test point x? Walk me through your reasoning in enumerated steps. Start from the basics and gradually build up.

Answer. True.

Given a latent variable model $p_\theta(x, z)$ and a fixed choice of observation x, you can interpret the function $p_\theta(x, z)$ as an unnormalised distribution with respect to z, and whose partition function is $p_\theta(\mathrm{x})$.

Answer. True.

1. **All joint probabilities are normalized.**
   True.
   Rationale: A joint probability distribution $p(x, z)$ over two random variables x and z is defined such that when integrated (or summed) over all possible values of x and z,

the result is 1. This is a fundamental property of probability distributions: the total probability across the entire probability space must sum to 1.

$$\iint p(x, z)\, dx\, dz = 1$$

2. **All marginal probabilities are normalized.**
   True.
   Rationale: A marginal probability distribution is obtained by summing or integrating the joint probability distribution over all possible values of the other variable(s). The resulting marginal distribution for x is:

$$p(x) = \int p(x, z)\, dz$$

   This ensures that regardless of which variables are marginalized out, the resulting marginal distribution is a valid probability distribution.

$$\int p(x)\, dx = 1$$

3. **For a fixed choice of observation x, that distribution is not normalized. Similarly, for a fixed choice of z, that distribution is also not normalized.**
   True.
   Rationale: When we fix x and consider $p(x, z)$ as a function of z alone, it does not integrate to 1 over z, because it is not a complete distribution over z.

$$\int p(x, z)\, dz \neq 1 \quad \text{for fixed } x$$

   The same holds when z is fixed and we consider $p(x, z)$ as a function of x.

$$\int p(x, z)\, dx \neq 1 \quad \text{for fixed } z$$

4. **Dividing unnormalized distributions by a partition function normalizes that distribution. In other words, all conditional probabilities are normalized.**
   True.
   Rationale: The conditional distribution $p(z|x)$ is obtained by dividing the joint distribution by the marginal distribution of x, and is normalized with respect to z.

$$p(z|x) = \frac{p(x, z)}{p(x)}$$

   The marginal distribution $p(x)$ serves as the partition function, ensuring that $p(z|x)$ integrates to 1 over the space of z.

$$\int p(z|x)\, dz = 1$$

Because the variational autoencoder objective contains a Reconstruction term, an optimally trained VAE will always make us of the latent space and thus have non-zero KL-divergence to the prior $D(q(z|x)||p(z)) > 0$. Note we define an optimally trained VAE to be one with a tight ELBO (i.e. equality holds between the ELBO and the log-likelihood) and whose marginal distribution matches the data distribution.

Answer. False. **Rationale:**

- A VAE comprises two main components: an encoder that maps inputs to a latent space and a decoder that reconstructs the inputs from the latent space.
- The VAE loss function combines:
  1. *Reconstruction Loss*: Measures fidelity of input data reconstruction from the latent space.
  2. *Regularization Term (KL Divergence)*: Encourages the latent space distribution $q(z|x)$ to be similar to some prior distribution $p(z)$, typically a standard normal distribution.
- The total loss is expressed as:

$$\text{Loss} = \text{Reconstruction Loss} + \beta \cdot D_{\text{KL}}(q(z|x)||p(z))$$

  where β balances the two components.
- Optimal training seeks a balance between reconstruction and regularization, but does not ensure the latent space is fully utilized ($D_{\text{KL}} > 0$).
- If the data can be reconstructed without additional information beyond the prior, then D_{KL} could be zero.
- In practice, some latent space information is typically required ($D_{\text{KL}} > 0$) to capture the complexities of the data.
- Posterior collapse, where the encoder's output matches the prior, can result in low or zero D_{KL} but typically at the cost of high reconstruction loss.
- Thus, while the reconstruction term does imply a trade-off, it does not guarantee a non-zero D_{KL} in an optimally trained VAE.

8.2.1 KL Divergence and Bounds

In the VAE lectures we have seen how addition/subtraction of a KL divergence term can yield a bound (e.g. subtracting a KL term from the log-likelihood in a latent variable model yields the ELBO) We shall apply this same technique of adding/subtracting the KL term to answer the following questions.

Given a joint distribution $p(x, z)$ show that the below is true for any choice of q. Explicitly show the KL divergence term you are adding/subtracting in your work.

$$\mathbb{E}_{p(x,z)}[\ln \frac{p(x,z)}{p(x)p(z)}] \leq \mathbb{E}_{p(z)}[D(P(x|z)||q(x))]$$

Answer. Add KL and since its non-negative we know that it'll always be greater.

$$\begin{aligned}
\mathbb{E}_{p(x,z)}[\ln \frac{p(x,z)}{p(x)p(z)}] &\leq \mathbb{E}_{p(x,z)}[\ln \frac{p(x,z)}{p(x)p(z)}] + D_{KL}(p(x)||q(x)) \\
&= \mathbb{E}_{p(x,z)}[\ln \frac{p(x|z)p(z)}{p(x)p(z)}] + \mathbb{E}_{p(x)}[ln\frac{p(x)}{q(x)}] \\
&= \mathbb{E}_{p(x,z)}[\ln \frac{p(x|z)}{p(x)}] + \mathbb{E}_{p(x)}[\ln \frac{p(x)}{q(x)}] \\
&= \mathbb{E}_{p(x,z)}[\ln \frac{p(x|z)}{p(x)} + \ln \frac{p(x)}{q(x)}] \\
&= \mathbb{E}_{p(x,z)}[\ln \frac{p(x|z)}{q(x)}] \text{ the p(x) cancels each other out} \\
&= \mathbb{E}_{p(z)}\mathbb{E}_{p(x|z)}[\ln \frac{p(x|z)}{q(x)}] \text{ decompose the expectation} \\
&= \mathbb{E}_{p(z)}D_{KL}(P(x|z)||q(x))
\end{aligned}$$

Given a joint distribution $p(x, z)$ show that

$$\mathbb{E}_{p(x,z)}[ln\frac{p(x,z)}{p(x)p(z)}] \geq -\mathbb{E}_{p(z)}[\ln p(z)] + \mathbb{E}_{p(z,x)}[\ln q(z|x)] \tag{31}$$

for any choice of q. Explicitly show that the KL divergence term you are adding or subtracting in your work.

Answer. Subtract $\mathbb{E}_{p(x)}D_{KL}(p(z|x)||q(z|x))$ because this term is implied in the question where the right hand side has q(z—x) and we want the expectation to match the first, which is the joint distribution $p(x, z)$

$$\begin{aligned}
\mathbb{E}_{p(x,z)}[\ln \frac{p(x,z)}{p(x)p(z)}] &\geq \mathbb{E}_{p(x,z)}[\ln \frac{p(x,z)}{p(x)p(z)}] - \mathbb{E}_{p(x)}D_{KL}(p(z|x)||q(z|x)) \\
&= \mathbb{E}_{p(x,z)}[\ln \frac{p(x,z)}{p(x)p(z)}] - \mathbb{E}_{p(x)}\mathbb{E}_{p(z|x)} \ln \frac{p(z|x)}{q(z|x)} \\
&= \mathbb{E}_{p(x,z)}[\ln \frac{p(z|x)p(x)}{p(x)p(z)}] - \mathbb{E}_{p(x)}\mathbb{E}_{p(z|x)} \ln \frac{p(z|x)}{q(z|x)} \\
&= \mathbb{E}_{p(x,z)}[\ln \frac{p(z|x)}{p(z)}] - \mathbb{E}_{p(x)}\mathbb{E}_{p(z|x)} \ln \frac{p(z|x)}{q(z|x)} \\
&= \mathbb{E}_{p(x,z)}[\ln p(z|x) - \ln p(z)] - \mathbb{E}_{p(x,z)}[\ln p(z|x) - \ln q(z|x)] \\
&= -\mathbb{E}_{p(z)}[\ln p(z)] + \mathbb{E}_{p(x,z)}[\ \ln q(z|x)]
\end{aligned}$$

Consider the joint distribution of a latent variable model denoted by p(x,z). The model is capable of sampling only two images $\{x^(1), x^(2)\}$. You may image that these are two binarised MNIST images where $x^{(i)}\} \in \{0,1\}^7 84$. Note that this latent variable model is equipped with a scalar latent variable $z \in \mathbb{R}$. Further, this model is described by

$$p(z) = \mathcal{N}(z; 0, 1)$$

$$p(x \mid z) = \begin{cases} 1 & \text{if } z \geq 0 \text{ and } x = x^{(1)} \\ 0 & \text{if } z \geq 0 \text{ and } x \neq x^{(1)} \\ 1 & \text{if } z < 0 \text{ and } x = x^{(2)} \\ 0 & \text{if } z < 0 \text{ and } x \neq x^{(2)} \end{cases}$$

where $p(x \mid z)$ is a probability mass function and $p(z)$ is a probability density function. In other words, the generative model will *always* sample the first image $x^{(1)}$ when conditioned on $z \geq 0$, and the model will *always* sample the second image $x^{(2)}$ when conditioned on $z < 0$. $\mathcal{N}(z; 0, 1)$ indicates a Gaussian distribution with mean zero and variance 1.

1. We can consider the log-likelihood of some image x under our model as $l_{like} = logp(x)$. Based on the model described above, what is $l_{like}(x^{(1)}$

   Answer. (a) **Marginalize the Latent Variable**: To find $p(x)$, we need to integrate out the latent variable z from the joint distribution:

   $$p(x) = \int p(x, z)dz = \int p(x|z)p(z)dz$$

   (b) **Apply the Model's Specifics**: For the image $x^{(1)}$, the conditional probability $p(x|z)$ is 1 when $z \geq 0$ and 0 otherwise. Hence, the marginal probability $p(x^{(1)})$ simplifies to integrating the density of z over the non-negative domain:

   $$p(x^{(1)}) = \int_0^\infty p(z)dz$$

   (c) **Use the Properties of the Standard Normal Distribution**: Since $p(z)$ is the standard normal distribution $\mathcal{N}(z; 0, 1)$, the integral from 0 to infinity is the cumulative distribution function (CDF) of z evaluated at 0, which is $\Phi(0) = 0.5$.

   (d) **Calculate the Marginal Probability**: Therefore, we have:

   $$p(x^{(1)}) = 0.5$$

   (e) **Compute the Log-Likelihood**: Now, we can find the log-likelihood of $x^{(1)}$:

   $$l_{\text{like}}(x^{(1)}) = \log p(x^{(1)}) = \log 0.5$$

   (f) **Simplify the Expression**: Using logarithm properties:

   $$l_{\text{like}}(x^{(1)}) = \log 0.5 = -\log 2$$

2. What is the set of all points z for which the posterior density is positive $p(z|x^{(1)}) > 0$ when conditioned on $x^{(1)}$?

   Answer. Use baye's theorem. To determine the set of all points z for which the posterior density $p(z|x^{(1)})$ is positive when conditioned on $x^{(1)}$, we will proceed step by step:

   (a) **Understanding Bayes' Theorem**: Bayes' theorem relates the conditional and marginal probabilities of random variables. It is expressed as:

   $$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

   where:

   - $p(z|x)$ is the posterior density of z given x.
   - $p(x|z)$ is the likelihood of x given z.
   - $p(z)$ is the prior density of z.
   - $p(x)$ is the evidence or marginal likelihood of x.

   (b) **Prior Density** $p(z)$: The prior density of z, $p(z)$, is given as a Gaussian $\mathcal{N}(z; 0, 1)$, which is positive for all z in $\mathbb{R}$.

   (c) **Likelihood** $p(x|z)$: The likelihood $p(x|z)$ is defined piecewise:

   - It is 1 if $z \geq 0$ and $x = x^{(1)}$.
   - It is 0 if $z \geq 0$ and $x \neq x^{(1)}$.
   - It is 1 if $z < 0$ and $x = x^{(2)}$.
   - It is 0 if $z < 0$ and $x \neq x^{(2)}$.

   (d) **Marginal Likelihood** $p(x)$: The evidence $p(x)$ is a normalizing constant ensuring that the probabilities sum to 1. Since it's the sum (or integral) of the product of the prior and likelihood over all possible values of z, it is always positive.

   (e) **Posterior Density** $p(z|x^{(1)})$: We are interested in the posterior when $x = x^{(1)}$. According to the definition of the likelihood:

   - When $x = x^{(1)}$ and $z \geq 0$, $p(x|z) = 1$.
   - For $x = x^{(1)}$ and $z < 0$, $p(x|z) = 0$ because the condition for $x = x^{(2)}$ does not apply.

   (f) **Determining the Set of Positive Posterior Densities**: The posterior density $p(z|x^{(1)})$ is positive if and only if both $p(x|z)$ and $p(z)$ are positive (since $p(x)$ is always positive). From the likelihood definition, $p(x|z)$ is only positive when $z \geq 0$ for $x = x^{(1)}$.

   Therefore, the set of all points z for which $p(z|x^{(1)})$ is positive is the set of all non-negative real numbers:

   $$\{z \in \mathbb{R} : z \geq 0\}$$

This means that the posterior density $p(z|x^{(1)})$ will be greater than zero whenever z is zero or positive.

3. Prove that $E_{q(z)}[\log \frac{p(x,z)}{q(z)}] = \log p(x) - D(q(z)||p(z|x))$

Answer.

$$\begin{aligned} E_{q(z)}[log\frac{p(x,z)}{q(z)}] &= E_{q(z)}[log\frac{p(z|x)}{q(z)} + logp(x)] \\ &= -D(q(z)||p(z|x)) + E_{q(z)}logp(x) \\ &= -D(q(z)||p(z|x)) + logp(x) \end{aligned}$$

4. Suppose q(z) is a univariate gaussian with positive variance. What is the set o fall points z for which q(z) > 0?

Answer. Need to know the univariate formula. Need to know that the exp function is always positive.

(a) **Definition of a Gaussian Distribution:**
A Gaussian distribution, also known as a normal distribution, is defined by two parameters: the mean (μ) and the variance (σ^2). The probability density function (PDF) for a Gaussian distributed random variable Z with mean μ and variance σ^2 is given by:

$$q(z) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right)$$

(b) **Properties of the Gaussian Distribution:**

- The distribution is symmetric about the mean.
- It has a bell-shaped curve.
- The distribution is continuous.

(c) **Positivity of the Gaussian PDF:**
The exponential function $\exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right)$ is always positive for all real numbers z, since the square of any real number is non-negative, and the exponential of a non-negative number is positive.

(d) **The set of points where $q(z) > 0$:**
Given that $\sigma^2 > 0$, the variance is positive, meaning the factor $\frac{1}{\sqrt{2\pi\sigma^2}}$ is also positive. Thus, for any real number z, $q(z)$ is positive.

(e) **Conclusion:**
Since both factors of the Gaussian PDF are always positive for all real numbers z, it follows that $q(z) > 0$ for the entire set of real numbers, i.e., $\mathbb{R}$.

The log-likelihood $l_{like}(x^{(1)})$ is finite and negative, 0, finite and position, none of the above?

Answer. finite and negative because ln(p(x)¡1) is always negative

For any q that is a univariate Gaussian with positive variance $l_{ELBO}(x^{(1)}; q)$ is finite and negative, 0, finite and positive, none of the above

Answer. None of the above. The KL divergence is undefined since q(z) assigns non-zero probability mass to the interval $(\inf, 0)$, but $p(z|x(1))$ assigns zero probability mass to that interval.

8.3 Change of variables

You are dealing with a $32 \to 32$ greyscale image dataset whose pixel intensities are real-valued in the interval [0, 255]. A common pre-processing procedure is to scale your data by 1/127.5 and then shifting it by 1, so that your data lies in the interval [1, 1], before training your Gaussian autoregressive model $p_\theta(x)$, where x has dimensionality $32 \to 32$. You do so and report a test set log-likelihood of $\frac{1}{2}\sum_{i=1}^{N} \ln p_\theta(x^{(i)}) = 32.5$
where $\{x(i)\}_N^{i=1}$ is your test set and each x(i) is a processed $[1, 1]^{32\to32}$ image. However, Reviewer 2 requests that you report your model's test set log-likelihood in the original $[0, 255]^{32\to32}$ space for your report to be comparable with the literature. What is your test set log-likelihood in the original $[0, 255]^{32\to32}$? Report your value to the third significant digit in scientific notation. Explain how you got your answer for full credit

Answer. To revert the log-likelihood from the transformed scale $[-1, 1]$ back to the original scale $[0, 255]$, you need to account for the Jacobian of the transformation for each pixel in each image.
The forward transformation is $x* = \frac{x}{127.5} - 1$. The inverse is $x = (x * +1)127.5$. The determinant of the jacobian matrix of this transformation would be the absolute value of the scale factor raised to the power of the dimensionality of the data which is 32x32 per image.

$$|det(J)| = (127.5)^{32x32}$$

The log likelihood of the original data can be obtained by adjusting the log likelihood of the scaled data with the log of the determinant of the jacobian, as the change of variables in probability density function requires:

$$\ln(p_{original}(x)) = \ln p_\theta(x*) - \ln(|det(J)|)$$

The adjustment for a single image, which has 32×32 or 1024 pixels, due to the Jacobian determinant is:

$$\text{Adjustment per image} = -1024 \ln(127.5)$$

The negative sign comes from the fact that we're adjusting the log-likelihood from the scaled space to the original space, effectively "undoing" the earlier scaling. When you extend this adjustment to the entire test set of N images, the total adjustment to the log-likelihood is:

$$\text{Total adjustment} = N \times (-1024 \ln(127.5))$$

The log-likelihood in the original pixel intensity space across all N images would then be calculated as:

$$\frac{1}{2}\sum_{i=1}^{N} \ln p_{\text{original}}(x^{(i)}) = 32.5 - N \times 1024 \ln(127.5)$$

By substituting the approximate value of ln(127.5) into the adjustment per image, you get:

$$\text{Adjustment per image} \approx -1024 \times 4.846 \approx -4963.584$$

This is the amount to be subtracted from the log-likelihood for each image in the test set. Multiplying this adjustment by N will give you the total adjustment needed for the test set. If 32.5 is the sum of log-likelihoods for the entire test set, you adjust it as follows:

$$\frac{1}{2}\sum_{i=1}^{N} \ln p_{\text{original}}(x^{(i)}) = 32.5N - N \times 1024 \ln(127.5)$$

You would replace N with the actual number of images in your test set to compute the final value.

Given a univariate normal random variable Y, its exponentiation X = exp(Y) is said to have a Log-Normal distribution. If Y is distributed according to $\mathcal{N}(\mu, \sigma^2)$ then we denote X = exp(Y) as being distributed to $LN(\mu, \sigma^2)$. Using this definition of the gaussian probability density function for p_Y and the change of variables formula that relates p_Y to p_x prove that the probability density function for $X \sim LN(\mu, \sigma^2)$ is

$$p_X(x) = \frac{1}{x\sigma\sqrt{2\pi}} exp(-\frac{(lnx - \mu)^2}{2\sigma^2}) \tag{32}$$

Answer.

$$\begin{aligned}
p_X(x) &= p_Y(f^{-1}(x))\left|Det(\frac{\partial}{\partial x} f^{-1}(x))\right| \\
p_X(x = exp(y)) &= p_Y(y = lnx)\left|Det(\frac{\partial}{\partial x} ln(x))\right| \\
p_X(x = exp(y)) &= p_Y(y = lnx)\left|Det(\frac{1}{x})\right| \\
p_X(x = exp(y)) &= p_Y(y = lnx)\left|\frac{1}{x}\right| \\
&= p_Y(y = lnx)\frac{1}{x} \text{ because the derivative of lnx is always positive} \\
&= \frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-(\ln x - \mu)^2}{2\sigma^2} \frac{1}{x}
\end{aligned}$$

Let $Z \sim Uniform[-2, 3]$ and $X = \exp(Z)$. What is $p_x(5)$?

Answer. When sampling from uniform distribution, the probability is always the same regardless of the value of the realised random variable.

$$\frac{1}{25}$$

8.4 Normalising Flow Models

In Parallel Wavenet, evaluating the likelihood assigned by the student model for any external data point is computationally intractable (requires exponential time in the dimensions of the sample)

Answer. The key point of Parallel WaveNet is to make the generation of audio samples fast. It does not necessarily change the fundamental approach to likelihood evaluation. While the student model in Parallel WaveNet is trained to output the same distribution as the autoregressive teacher model, it is a flow-based model that allows direct and efficient likelihood evaluation for each sample. This is not an autoregressive process, and thus it does not require sequential operations that would make likelihood evaluation computationally intractable.

A permutation matrix is defined as a binary square matrix with 0,1 entries such that every column and every row sums to 1. The Jacobian for RealNVP can be expressed as the product of a series of upper or lower triangle matrices and permutation matrices.

Answer. True. Let's start by defining the terms involved and then build up to explain why the statement is true.

Permutation Matrix: A permutation matrix is indeed a binary square matrix (meaning it has the same number of rows and columns) where each row and each column has exactly one entry of 1 and all other entries are 0. This configuration ensures that each row and column sums to 1. Essentially, a permutation matrix represents a permutation of the standard basis in Euclidean space. For instance, in three-dimensional space, a permutation matrix can swap the x-axis with the y-axis, the y-axis with the z-axis, and so on, depending on which permutation of the identity matrix it represents.

RealNVP: RealNVP stands for "Real-valued Non-Volume Preserving" transformations, which is a type of flow-based generative model used in machine learning for the purpose of density estimation. RealNVP models are designed to learn complex probability distributions by using a series of transformations that are invertible and whose Jacobian determinant is easy to compute.

Jacobian: In the context of RealNVP and similar models, the Jacobian matrix represents the partial derivatives of the transformed variables with respect to the original variables. The determinant of the Jacobian (Jacobian determinant) is particularly important in these models because it measures the change in volume (or density) under the transformation. In flow-based models, it's essential that the determinant of the Jacobian can be computed efficiently because it's used in the likelihood computation.

Upper or Lower Triangular Matrices: In RealNVP, the transformations used are specifically

chosen to be a series of affine coupling layers that are invertible and where the Jacobian is a triangular matrix (either upper or lower triangular). Triangular matrices are useful because their determinants can be computed simply by taking the product of the diagonal entries, which makes the computation of the Jacobian determinant tractable.

Combining the Concepts: The statement that "The Jacobian for RealNVP can be expressed as the product of a series of upper or lower triangle matrices and permutation matrices" is true. In RealNVP, the transformations that are applied to the data involve alternations between affine coupling layers and fixed permutation layers. The permutation layers are included as permutation matrices and are used to ensure that the model does not just learn the identity function and that each input dimension can affect each output dimension.

The affine coupling layers result in a Jacobian that is triangular (because only part of the input is modified while the rest passes through unchanged). When these layers are composed, the overall Jacobian of the transformation is a product of the Jacobians of each layer.

The use of permutation matrices ensures that different dimensions are mixed, making the flow-based model more expressive and capable of modeling complex distributions. The final Jacobian of the transformation is, therefore, a product of triangular matrices (from the affine coupling layers) and permutation matrices. This structure allows for an efficient computation of the Jacobian determinant, which is crucial for training RealNVP models using maximum likelihood estimation.

A toy example Let's consider a 2D input vector $\mathbf{x} = [x_1, x_2]$.

Affine Coupling Layer: In an affine coupling layer, we transform one part of the input vector while the other part passes through unchanged. Suppose we transform x_2 as follows:

- Let $s(x_1) = 2$ (scale) and $t(x_1) = -1$ (translation) for simplicity.
- The transformed variable is $y_2 = s \cdot x_2 + t = 2 \cdot x_2 - 1$.
- x_1 passes through unchanged, so $y_1 = x_1$.

The output of the affine coupling layer is $\mathbf{y} = [y_1, y_2] = [x_1, 2x_2 - 1]$.
The Jacobian of this transformation is:

$$J_{\text{coupling}} = \begin{bmatrix} 1 & 0 \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

which is a lower triangular matrix with determinant $det(J_{\text{coupling}}) = 1 \cdot 2 = 2$.
Permutation: We then apply a permutation to $\mathbf{y}$ using the permutation matrix:

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

to get the permuted vector $\mathbf{z}$:

$$\mathbf{z} = P\mathbf{y} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} y_2 \\ y_1 \end{bmatrix}$$

which results in $\mathbf{z} = [2x_2 - 1, x_1]$.
The permutation matrix P has a Jacobian which is itself, and its determinant is 1 because it is an orthogonal matrix.

Overall Transformation: The overall transformation is the composition of the affine coupling layer and the permutation, and its Jacobian is the product of the Jacobians of each individual transformation.

Let R be a rotation matrix (i.e. such that $R^T R = I$) and $X = f(Z)$ be a flow model where $Z \sim \mathcal{N}(0, I)$ is distributed as a Gaussian with unit covariance I. Then $g(z) = f(RZ)$ is another flow model that achieves the same likelihood as f on any dataset.

Answer. True because the determinant after applying a rotation matrix (or it's inverse) doesn't change. The likelihood formula is normalized based on the absolute value of the determinant of the inverse transformation multiplied by the inverse transformation.

Theorem: Let R be a rotation matrix (i.e., such that $R^T R = I$) and $X = f(Z)$ be a flow model where $Z \sim \mathcal{N}(0, I)$ is distributed as a Gaussian with unit covariance I. Then $g(z) = f(RZ)$ is another flow model that achieves the same likelihood as f on any dataset.

Proof: *Rotation Matrix:* A rotation matrix R is an orthogonal matrix with the property that $R^T R = I$, where I is the identity matrix. This implies that $\det(R) = \pm 1$. For rotation matrices, $\det(R) = 1$, indicating volume preservation under rotation.
Flow Model: A flow model f is an invertible transformation that maps a simple base distribution Z, like a Gaussian $\mathcal{N}(0, I)$, to a complex data distribution.

Gaussian Distribution: The Gaussian distribution $\mathcal{N}(0, I)$ is spherically symmetric. Any rotation of a vector Z sampled from this distribution results in another vector with the same distribution.

Transformation of a Gaussian: If Z is distributed as $\mathcal{N}(0, I)$, then RZ is also distributed as $\mathcal{N}(0, I)$ because the rotation matrix R does not alter the distribution due to its spherical symmetry.

New Flow Model g: The function $g(z) = f(RZ)$ first applies a rotation to Z and then passes this through the flow model f. Since f can transform $\mathcal{N}(0, I)$ into the complex data distribution, so can g, as RZ has the same distribution as Z.

Likelihood: The likelihood of a data point x under model f is given by:

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

For the model g:

$$p_X(x) = p_Z(g^{-1}(x)) \left| \det \left(\frac{\partial g^{-1}(x)}{\partial x} \right) \right|$$

Since $g^{-1}(x) = R^T f^{-1}(x)$ and $\det(R) = 1$, the determinants of the Jacobians are the same, and thus the likelihoods under f and g are identical.

Therefore, $g(z) = f(RZ)$ achieves the same likelihood as f on any dataset, which proves our initial statement.

Questions and Clarifications:

1. **What is spherically symmetric? What does that look like for a Gaussian distribution with 0 mean and identity covariance matrix? Would it not be symmetric if these parameters are different?**

   Spherical symmetry in the context of a Gaussian distribution means that the distribution has the same appearance from all directions. In one dimension, this is a bell curve; in two dimensions, it's a symmetric bell-shaped curve around the origin. For a multivariate Gaussian, this occurs when the covariance matrix is the identity matrix, indicating no correlation between variables and equal variance in all directions. If the mean is not zero or the covariance is not identity, the distribution will lose its spherical symmetry, either being shifted (for non-zero mean) or stretched/compressed (for non-identity covariance).

2. **Transformation of a Gaussian: If the Gaussian was skewed, i.e., not 0 mean and identity covariance, and a rotation matrix was applied to it, does it alter the distribution?**

   Applying a rotation matrix to a non-spherically symmetric Gaussian (non-zero mean or non-identity covariance) does not alter the fundamental characteristics like skewness or kurtosis of the distribution. It merely changes the orientation of the Gaussian in space because rotation is a linear transformation that does not affect the probabilities, just reorients the axes.

3. **Likelihood vs. Probability:** Likelihood and probability are distinct concepts.

   - *Probability* is a measure of the chance of an event occurring given a fixed set of parameters and is denoted by $P(x)$ or $P(x|y)$.
   - *Likelihood*, denoted as $L(\theta|x)$, measures how well a set of parameters θ explains observed data x. While probability predicts future occurrences, likelihood evaluates parameter sets based on observed data. In formal terms, if θ represents the

parameters and x the data, the likelihood is $L(\theta|x)$, which is often proportional to $P(x|\theta)$ the probability of the data given the parameters, not the other way around.

- In summary, likelihood is often proportional to $P(x|\theta)$ when performing maximum likelihood estimation because the goal is to compare which parameters are most supported by the data, not to measure the actual probability of the data. When the goal is to calculate the probability distribution of the parameters given the data, as in Bayesian inference, the relationship is not merely proportional due to the inclusion of the prior and marginal likelihood.

In the context of generative models, $p_X(x)$ is referred to as the probability density function (PDF) of x under the model X. This is sometimes loosely referred to as the likelihood of the data under the model though technically "likelihood" would be the term for the function of the model parameters given the data.

A normalising flow model will map an observed random variable to lower dimensional latent variable z.

Answer. False. A normalising flow model doesn't not reduce the dimensionality when it performs it's mapping. Flow models indeed map observed variables X to latent variables Z in each transformation step of the flow.

8.4.1 FLOW + VAE: Augmenting variational posteriors

We wish to use flexible flow models for variational autoencoders. Let $x \in \mathbb{R}^D$ denote the inputs, z the latent variables, $p_\theta(z|x)$ the generative model, $p(z)$ the prior and $r_\phi(z|x)$ as the basic inference model representing a Gaussian distribution $\mathcal{N}(z; \mu_\phi(x), \text{diag}(\sigma_\phi(x)^2))$.
Instead of using $r_\phi(z|x)$ directly as the inference model, we will transform this distribution using a normalizing flow to obtain a richer variational posterior distribution $q_{\phi,\psi}(z|x)$. Specifically, let $f_\psi : \mathbb{R}^F \to \mathbb{R}^F$ be an invertible transformation, $\mu_\phi : \mathbb{R}^D \to \mathbb{R}^F$, and $\sigma_\phi : \mathbb{R}^D \to \mathbb{R}^F$. We use the following procedure to sample z from $q_{\phi,\psi}(z|x)$ given x:

- Sample $\tilde{z} \sim \mathcal{N}(z; \mu_\phi(x), \text{diag}(\sigma_\phi(x)^2))$
- Compute $z = f_\psi(\tilde{z})$

(a) [8 points] Derive an expression for $\log q_{\phi,\psi}(z|x)$. The function should take x and z as input, output a scalar value, and depend on μ_ϕ, σ_ϕ, and f_ψ. You can use $\mathcal{N}(u; \mu, \text{diag}(\sigma^2))$ to denote the pdf for normal distribution with mean μ and covariance $\text{diag}(\sigma^2)$ evaluated at u.

Answer. We use the normalizing flow formula to evaluate the density of the new variable. Qualitatively, we first evaluate the density of the old variable, then multiply that with the volume transformation to the new variable.

1. Recognize that the probability density function (pdf) of a random variable changes when the variable is transformed.
2. The change of variables formula for a continuous random variable states:

$$q_{\phi,\psi}(z|x) = p_{\tilde{z}}(f_\psi^{-1}(z)) \left| \det \frac{\partial}{\partial z}(f_\psi^{-1}(z)) \right|$$

   where $p_{\tilde{z}}$ is the pdf of the original variable $\tilde{z}$, and $z = f_\psi(\tilde{z})$ is the transformed variable.
3. For a normally distributed variable $\tilde{z} \sim \mathcal{N}(z; \mu_\phi(x), \text{diag}(\sigma_\phi(x)^2))$, the transformed pdf becomes:

$$q_{\phi,\psi}(z|x) = \mathcal{N}(f_\psi^{-1}(z); \mu_\phi(x), \text{diag}(\sigma_\phi(x)^2)) \left| \det \frac{\partial}{\partial z}(f_\psi^{-1}(z)) \right|$$

4. Taking the logarithm of the pdf yields:

$$\log q_{\phi,\psi}(z|x) = \log \mathcal{N}(f_\psi^{-1}(z); \mu_\phi(x), \text{diag}(\sigma_\phi(x)^2)) + \log \left| \det \frac{\partial}{\partial z}(f_\psi^{-1}(z)) \right|$$

(b) [12 points]

Consider $r_\phi(z|x)$ as the basic Gaussian inference model (without using the flow model), with the following sampling process:

- Sample $z \sim \mathcal{N}(z; \mu_\phi(x), \text{diag}(\sigma_\phi(x)^2))$

Show that the best evidence lower bound we can achieve with $q_{\phi,\psi}$ is at least as tight as the best one we can achieve with r_ϕ, i.e.,

$$\max_{\theta,\phi,\psi} \text{ELBO}(x; \rho, q_{\phi,\psi}) \geq \max_{\theta,\phi} \text{ELBO}(x; \rho, r_\phi)$$

where

$$\text{ELBO}(x; \rho, q) = \mathbb{E}_{q(z|x)}[\log p(x, z) - \log q(z|x)]$$

You may assume that f_ψ can represent any invertible function $\mathbb{R}^F \to \mathbb{R}^F$.

Answer. TODO

True or False: The use of a bijective transformation in normalizing flows means that the model can always perfectly reconstruct the input data from its latent representation without any loss of information.

Answer. Generally true. Bijective transformation means invertible.

True or False: Normalizing flow models can be used to approximate Bayesian posteriors, and in such a context, they can be seen as a form of implicit variational inference.

Answer. True. Flow models can be used in VAEs to approximate the posterior $p(z|x)$

True or False: The expressiveness of a normalizing flow is independent of the depth of the transformation, meaning that increasing the number of transformations does not affect the complexity of the distribution that can be modeled.

Answer. False. The expressiveness does depend on the number of layers.

True or False: In normalizing flow models, the choice of the base distribution is arbitrary and has no significant impact on the model's ability to learn the target distribution.

Answer. False. The initial distribution significantly affects the model's performance.

True or False: Autoregressive normalizing flows, which include models like RealNVP and Masked Autoregressive Flow (MAF), are designed to have tractable Jacobians, making them particularly suitable for density estimation tasks.

Answer. True.

8.5 Energy Based Models

In this question, we will consider energy-based models from a variational perspective.

(a) [5 points] Consider an EBM with an unnormalized distribution $\tilde{p}_\theta(x)$ and partition function $Z(\theta) = \int \tilde{p}_\theta(x)dx$. Computing the log-partition function $\ln Z(\theta)$ is usually intractable. So we shall look to bounding this quantity instead. In particular, if we introduce a proposal distribution $q(x)$ that is easy to compute and sample from, we can construct the following lower bound for the log-partition function,

$$\ln Z(\theta) \geq \mathbb{E}_{q(x)}\left[\ln \frac{\tilde{p}_\theta(x)}{q(x)}\right]. \qquad (6)$$

Prove that this bound holds for any choice of q. It may help to notice the strong resemblance between this expression and the Evidence Lower Bound for a latent variable model.

Answer. We want to exploit Jensen's inequality by first recognising that we need to relate $Z(\theta)$ to the form on the right hand side $\frac{p_\theta(x)}{q(x)}$

$$\begin{aligned} ln(Z(\theta)) &= ln(\int \frac{q(x)}{q(x)}\tilde{p}_\theta(x)dx) \\ &= ln(\mathbb{E}_{q(x)}[\frac{\tilde{p}_\theta(x)}{q(x)}]) \\ &\geq \mathbb{E}_{q(x)}[ln\frac{\tilde{p}_\theta(x)}{q(x)}] \end{aligned}$$

(b) [5 points] Consider again the EBM with an unnormalized $\tilde{p}_\theta(x)$ and partition function $Z(\theta) = \int \tilde{p}_\theta(x)dx$. Note that, when normalized, $p_\theta(x) = \frac{\tilde{p}_\theta(x)}{Z(\theta)}$. So far, we have learned from class that gradient-based optimization of the EBM's log-likelihood requires computing

$$\nabla_\theta \ln p_\theta(x) = \nabla_\theta \ln \tilde{p}_\theta(x) - \mathbb{E}_{p_\theta(x)}\nabla_\theta \ln \tilde{p}_\theta(x), \qquad (7)$$

We now take a variational perspective to this expression by introducing the variational family $\mathcal{Q}$, which we shall denote as the set of all possible distributions over x. Prove that

$$\nabla_\theta \ln p_\theta(x) = \nabla_\theta \ln \tilde{p}_\theta(x) - \mathbb{E}_{q^*(x)}\nabla_\theta \ln \tilde{p}_\theta(x), \qquad (8)$$

where

$$q^*(x) = \arg\max_{q\in\mathcal{Q}} \mathbb{E}_{q(x)}\left[\ln \frac{\tilde{p}_\theta(x)}{q(x)}\right]. \qquad (9)$$

You may make use of and do not have to prove Equation (??). [Hint: Prove that $q^* = p_\theta$.]

Answer.

$$\begin{aligned}\nabla_\theta \ln(p_\theta(x)) &= \nabla_\theta \ln \frac{\tilde{p}(x)}{Z(\theta)} = \nabla_\theta \ln \frac{\tilde{p}(x)}{\mathbb{E}_{p_\theta(x)}\tilde{p}(x)} \\ &= \nabla_\theta \ln \tilde{p}(x) - \mathbb{E}_{p_\theta(x)} \nabla_\theta \ln \tilde{p_\theta}(x)\end{aligned}$$

When $q^* = p_\theta(x)$ then the equivalence holds

$$\nabla_\theta \ln(p_\theta(x)) = \nabla_\theta \ln \tilde{p}(x) - \mathbb{E}_{q^*(x)} \nabla_\theta \ln \tilde{p_\theta}(x)$$

To prove $q^* = p_\theta(x)$, need to introduce KL divergence between $q(x)$ and $p(x)$. We need to recognise that $p_\theta(x) = \frac{\tilde{p}(x)}{Z(\theta)}$

$$\begin{aligned}KL(q(x)||p_\theta(x)) &= \mathbb{E}_{q(x)} \ln \frac{q(x)}{p_\theta(x)} \\ &= \mathbb{E}_{q(x)} \ln \frac{q(x)}{p_\theta(x)} \\ &= \mathbb{E}_{q(x)}[\ln q(x) - \ln p_\theta(x)] \\ &= \mathbb{E}_{q(x)}[\ln q(x) - \ln \frac{\tilde{p}(x)}{Z(\theta)}] \\ &= \mathbb{E}_{q(x)}[\ln q(x) - \ln \tilde{p}(x) + Z(\theta)] \\ &= \mathbb{E}_{q(x)}[\ln q(x) - \ln \tilde{p}(x)] + Z(\theta) \\ &= -\mathbb{E}_{q(x)}\left[\ln \frac{\tilde{p}(x)}{q(x)}\right] + Z(\theta)\end{aligned}$$

$q^*(x) = p_\theta(x)$ is the minimisation of the KL divergence.