# Likelihood-based Generative Models

Generative and Graphical Models AI60201, Module 3

Adway Mitra

Indian Institute of Technology Kharagpur

19 October 2022

# Contents

# Background

# Generation of Complex Structures

- Aim of generative model: to synthesize new data-points
- Data-points may be complex structures like images, videos, speech etc
- They should be similar, but not identical to, the examples already present in a dataset
- For example, if a dataset has 1000 face images from 10 different persons, generative model should produce a face image, but not necessarily from any of these persons
- Conditional generation: generated datapoint should correspond to an input from the user
- Example: user provides a text caption, according to which an image is generated

# Image Generation

- An image is basically a $M \times N$ matrix, where each value (pixel) lies between 0 and 255
- Images are highly complex structures with spatial properties
- Deep Neural Networks, especially Convolutional Neural Network are suitable for representing images
- Deep Neural Networks like U-Net and autoencoders can produce an image as output against a low-dimensional input
- In Generative Models, the low-dimensional input can be sampled from a prior
- The low-dimensional input, or the prior, can encode the condition specified by the user
- A likelihood-based generative model defines a distribution over the space of all images
- The *desired* images (eg. visually meaningful images containing some specified object) should have high probability

# Classification of Generative Models

- Likelihood-based (define probability distribution over image space)
  - Fully observed (no latent variable)
    - Autoregressive Models
  - Latent variable based
    - Variational Autoencoders
    - Normalizing Flows
- Without Likelihood (doesn't define any such distribution)
  - Generative Adversarial Networks (and variants)

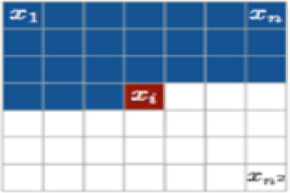
# Deep Autoregressive Models

# Autoregressive Models

- Generate each pixel sequentially, conditioned on those already generated
- The image is scanned, pixel by pixel, in a fixed order (row-wise or column-wise)
- For each pixel, a probability distribution is defined over possible values $[0, 255]$, parameterized with values of previous pixels (according to sequential order)probability
- Image likelihood $p(X;\theta) = \prod_{i,j} p(X_{i,j}|X_{1,1},\ldots,X_{i,j-1})$
- Given image $X$, probability $p(X;\theta) = \prod_{i,j} p(X_{i,j}|X_{1,1},\ldots,X_{i,j-1})$
- Drawback: scan order is artificial, may not carry enough information to insert new objects in image
- Good for evaluating likelihood of a given image, not good for sampling/synthesizing new images

# PixelRNN

- Each pixel's distribution parameterized by Neural Network
- Sequence of pixel values $(X_{1,1}, \ldots, X_{i,j-1})$ in scan order input to Recurrent Neural Network



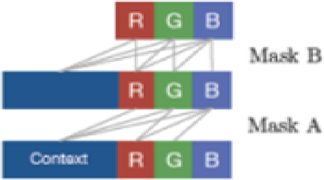




Figure: Pixel Recurrent Neural Network

# PixelCNN

- Faster but less accurate than PixelRNN
- For each pixel, focus only on pixels within receptive field
- Two filters for horizontal and vertical neighboring pixels

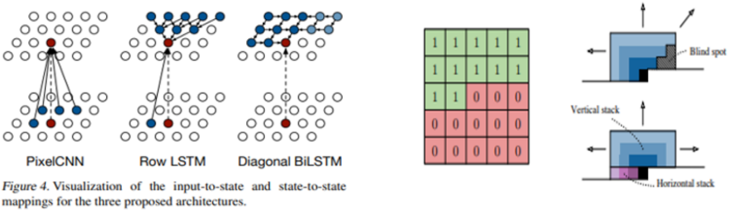



Figure: Pixel Convolutional Neural Network

# Variational Autoencoder

# Latent Variable Formulation

- Start with *image seed* $Z$, generated from prior $Z \sim P$
- It is then passed through a neural network $g_\theta$ which produces an image as output
- $g$ specifies neural network architecture, $\theta$ its parameters
- Add random noise to each pixel independently
- $Z \sim \mathcal{N}(0, 1)$, $X \sim \mathcal{N}(g_\theta(Z), \sigma^2 I_{m \times n})$
- Probability of any image $p(X) = \int p(X|Z)p(Z)dZ$
- Aim: estimate the neural network parameters $\theta$ by maximizing probability of desired images (training)
- Problem 1: The marginal likelihood $P(X)$ cannot be calculated analytically due to presence of $g_\theta$
- Problem 2: $p(X_i, Z_i)$ is easy to calculate, but we don't know $Z_i$ for each training image $X_i$

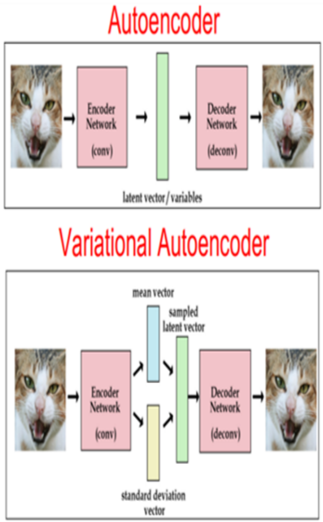
# Variational Inference Network

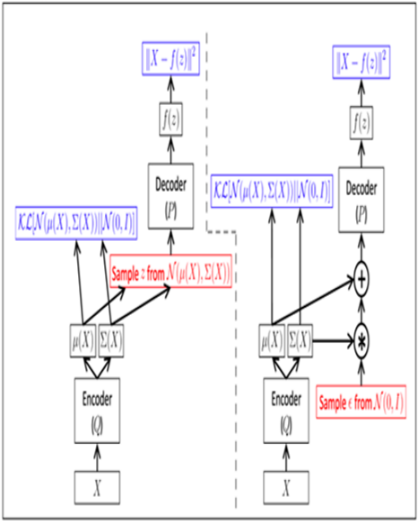
- For each training image $X_i$, estimate the corresponding $Z_i$
- $p(Z_i|X_i)$ cannot be calculated analytically, as $p(X)$ cannot be
- Alternative: variational inference $q(Z|X)$ to approximate $p(Z|X)$
- $q(Z|X) \sim \mathcal{N}(\mu(X_i), \Sigma(X_i))$
- $\{\mu(X_i), \Sigma(X_i)\} = h_\phi(X_i)$, where $h$ is another neural network with parameters $\phi$
- $h_\phi$ is called the *encoder*, while $g_\theta$ is called *decoder*
- Parameters $\theta, \phi$ to be estimated simultaneously
- Loss function with respect to both $X_i$ and $Z_i$
- $X_i$ must have high probability according to $p_{\theta^*}$, while $q_\phi(Z_i|X_i)$ should be close to $p(Z_i)$

# Objective Function

- $\theta^* = argmax_{\theta,\phi} \prod_{i=1}^{N} p(X_i|g_\phi(Z_i))$, where $Z_i \sim \mathcal{N}(h(\phi)(X_i)$
- Neural Network parameters $\theta, \phi$, may be estimated by backpropagation after defining suitable loss function
- $\ell(\theta,\phi) = \sum_{i=1}^{N} ||X_i - g_\theta(Z_i)|| + \sum_{i=1}^{N} KL(\mathcal{N}(h_\phi(X_i))||\mathcal{N}(0,1))$
- Equivalent to maximizing the evidence lower bound (ELBO)
- Backpropagation cannot be applied directly as it includes sampling $Z_i$ as an intermediate step
- Solution: Decouple the sampling from the backpropagation
- **Reparameterization Trick:** $\epsilon_i \sim \mathcal{N}(0,1)$ sampled independently, $Z_i = \mu_\phi(X_i) + \epsilon_i \sigma_\phi(X_i)$
- Now backpropagation can be applied to estimate $\theta, \phi$

# VaE Graphical Model



# Synthesizing New Images

- Once $\theta^*$ is estimated, new images can be generated by first sampling $Z$ from prior and then running it through decoder network $g_{\theta^*}$
- Use of parameters $\theta^*$ ensures that produced image will be *similar* to the images used for training (eg. if training set had handwritten digit images, only such images will be synthesize)
- Since $Z$ is generated independently for each image, no two images are going to be identical
- The differences may be manifested in any attribute, or a set of attributes, that can be encoded by a single real number like $Z$
- We can use inference network to carry out post-processing analysis of $Z$ to find which attributes(s) it represents

# Supervised VaE

- The datapoints used for training may be accompanied by class labels $(X_i, Y_i)$
- The generator/decoder network should be sensitive to the class label, i.e. $g_\theta(Y_i, Z_i)$
- Similarly, the inference/encoder network should be able to predict probability distribution $q(Y_i|X_i)$
- The training process will remain unchanged
- The loss function will include cross-entropy loss function for $Y_i$ (prediction by encoder and actual label)
- For new image generation, the user should specify $Y_i$, and the image will be generated accordingly

# VaE Variants Graphical Model

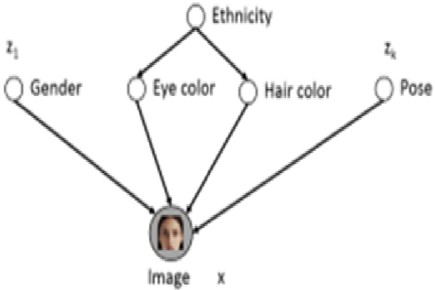
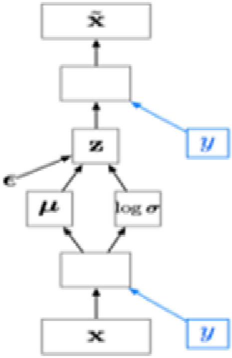



Image attributes through Latent Variables




Supervised VaE

# Interpretation

- The latent variable represents some attribute, or combination of attributes of the image
- However, these attributes are not specified by the user
- They can be stylistic (eg. thickness, level of blurring etc) or semantic (eg. complexion, gender etc)
- Basically, $Z$ captures the most prominent directions of variation in the dataset (rough analogy with non-linear PCA)
- Nature of $Z$ (real/discrete/binary) determines how much variation can be accommodated by it
- May be possible to include more variations with more latent variables!

# Examples of VaE images

- Generating new images via latent variable is effectively an interpolation of training images
- Hence, VaE images often look blurry as the RMSE error fails to retain sharpness of edges


Figure: Images generated by VaE

# Deep Generative Models

# Deep Latent Variables

- We may want several latent variables to capture the variability of training dataset
- The variables can be arranged in layers to indicate hierarchical attributes (eg. gender, ethnicity may be at a higher level of variability than complexion)
- Assumption: one layer of latent variables connected with another layer

# Deep Boltzmann Machine

- An undirected graphical model over layers of variables $Z^L, Z^{L-1}, \ldots, Z^2, Z^1, Z^0$, where $Z^0 = X$ (observations)
- Each layer of latent variables $Z^l$ connected to neighboring layers only ($Z^{l-1}$ and $Z^{l+1}$)
- Each variable $i$ in layer $l$ ($Z^l_i$) connected to each variable $j$ in layer $(l-1)$ through $W^l_{ij}$
- Joint distribution of all variables represented as product of edge potential functions
- $\phi(Z^l_i, Z^{l-1}_j) = exp(-W^l_{ij} Z^l_i Z^{l-1}_j)$
- $p(Z) \propto exp(-\sum_{l=1}^{L} (Z^{l-1})^T W^l Z^l)$
- The last layer connecting $Z^1$ with $Z^0 = X$ may be represented by a more complex neural network like $g_\theta$ (especially if $X$ is a complex object like image)

# Inference in DBM

- We may be interested to find the meaning of every latent variable (i.e. which attribute it represents)
- Approach: carry out inference of all latent variables to compute posteriors $p(Z^l_{ij} = 1|X)$
- Cannot be estimated directly, so we can use Gibbs Sampling
- According to D-separation rules, each variable in layer $l$ is independent of all variables except those in the neighboring layers
- For simplicity, assume all variables are binary
- $p(Z^l_i = 1| - Z^l_i) = \sigma(-(Z^{l-1})^T W^l_i - W^{l+1}_i Z^{l+1})$ where $\sigma$ denotes the sigmoid function
- We sample each variable in one iteration, keeping the rest constant
- We repeat this for many iterations, collecting samples of each variable regularly. Posteriors estimated from these samples

# Training of DBM by Contrastive Divergence

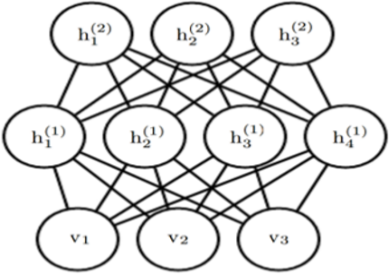
- Training of DBM involves estimating the parameters $W$
- Maximum Likelihood: $W^* = \textit{argmax}_W \sum_{Z^1,\dots,Z^L} p(X,Z)$
- Difficult to compute because there are too many combinatorial configurations to sum over
- Alternative approach: through sampling
- Starting from $X$, sample values of $Z^1,\dots,Z^L$, let these values be $\{\hat{Z}_l\}_{l=0}^L$
- Starting from $Z^L$, sample values of $z^{L-1},\dots,Z^1,Z^0$, call these values $\{Z_l^*\}_{l=0}^L$
- Sampling from the conditional distributions mentioned earlier
- At every layer $l$, update $W^l$ according to $\Delta W^l = \hat{Z}_{l-1}\hat{Z}_l^T - Z_{l-1}^*(Z_l^*)^T$
- $W^l = W^l - \alpha\Delta W^l$ where $\Delta W$ is called *contrastive divergence*

# DBM Training Algorithm

## Variational Stochastic MLE Algorithm:

For training a 2-layer DBM

Set $\epsilon$, the step size, to a small positive number
Set $k$, the number of Gibbs steps, high enough to allow a Markov chain of $p(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}; \boldsymbol{\theta} + \epsilon\Delta_{\boldsymbol{\theta}})$ to burn in, starting from samples from $p(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}; \boldsymbol{\theta})$.
Initialize three matrices, $\tilde{\boldsymbol{V}}$, $\tilde{\boldsymbol{H}}^{(1)}$ and $\tilde{\boldsymbol{H}}^{(2)}$ each with $m$ rows set to random values (e.g., from Bernoulli distributions, possibly with marginals matched to the model's marginals).
**while** not converged (learning loop) **do**
  Sample a minibatch of $m$ examples from the training data and arrange them as the rows of a design matrix $\boldsymbol{V}$.
  Initialize matrices $\hat{\boldsymbol{H}}^{(1)}$ and $\hat{\boldsymbol{H}}^{(2)}$, possibly to the model's marginals.
  **while** not converged (mean field inference loop) **do**
    $\hat{\boldsymbol{H}}^{(1)} \leftarrow \sigma\left(\boldsymbol{V}\boldsymbol{W}^{(1)} + \hat{\boldsymbol{H}}^{(2)}\boldsymbol{W}^{(2)\top}\right)$.
    $\hat{\boldsymbol{H}}^{(2)} \leftarrow \sigma\left(\hat{\boldsymbol{H}}^{(1)}\boldsymbol{W}^{(2)}\right)$.
  **end while**
  $\Delta_{\boldsymbol{W}^{(1)}} \leftarrow \frac{1}{m}\boldsymbol{V}^\top\hat{\boldsymbol{H}}^{(1)}$
  $\Delta_{\boldsymbol{W}^{(2)}} \leftarrow \frac{1}{m}\hat{\boldsymbol{H}}^{(1)\top}\hat{\boldsymbol{H}}^{(2)}$
  **for** $l = 1$ to $k$ (Gibbs sampling) **do**
    Gibbs block 1:
    $\forall i, j, \tilde{V}_{i,j}$ sampled from $P(\tilde{V}_{i,j} = 1) = \sigma\left(\boldsymbol{W}^{(1)}_{j,:}\left(\tilde{\boldsymbol{H}}^{(1)}_{i,:}\right)^\top\right)$.
    $\forall i, j, \tilde{H}^{(2)}_{i,j}$ sampled from $P(\tilde{H}^{(2)}_{i,j} = 1) = \sigma\left(\tilde{\boldsymbol{H}}^{(1)}_{i,:}\boldsymbol{W}^{(2)}_{:,j}\right)$.
    Gibbs block 2:
    $\forall i, j, \tilde{H}^{(1)}_{i,j}$ sampled from $P(\tilde{H}^{(1)}_{i,j} = 1) = \sigma\left(\tilde{\boldsymbol{V}}_{i,:}\boldsymbol{W}^{(1)}_{:,j} + \tilde{\boldsymbol{H}}^{(2)}_{i,:}\boldsymbol{W}^{(2)\top}_{j,:}\right)$.
  **end for**
  $\Delta_{\boldsymbol{W}^{(1)}} \leftarrow \Delta_{\boldsymbol{W}^{(1)}} - \frac{1}{m}\boldsymbol{V}^\top\tilde{\boldsymbol{H}}^{(1)}$
  $\Delta_{\boldsymbol{W}^{(2)}} \leftarrow \Delta_{\boldsymbol{W}^{(2)}} - \frac{1}{m}\tilde{\boldsymbol{H}}^{(1)\top}\tilde{\boldsymbol{H}}^{(2)}$
  $\boldsymbol{W}^{(1)} \leftarrow \boldsymbol{W}^{(1)} + \epsilon\Delta_{\boldsymbol{W}^{(1)}}$ (this is a cartoon illustration, in practice use a more effective algorithm, such as momentum with a decaying learning rate)
  $\boldsymbol{W}^{(2)} \leftarrow \boldsymbol{W}^{(2)} + \epsilon\Delta_{\boldsymbol{W}^{(2)}}$
**end while**



# Sampling from Deep Generative Model

- No direct way to sample from the joint distribution $p$
- Two samples can be compared using marginal likelihood $p(X)$
- Start with an initial sample $x^0$. Perturb by adding random noise, and accept new sample if it has higher likelihood
- $x^{t+1} = x^t + z^t$ where $z^t \sim \mathcal{N}(0, \sigma)$
- Accept $x^{t+1}$ if $p(x^{t+1}) \geq p(x^t)$
- If gradient of $p$ at $x$ can be calculated, we can use Langevin Dynamics for sampling
- Initialize $x^0$ randomly, navigate $x^{t+1} = x^t + \epsilon \nabla_x p(x_t) + \sqrt{2\epsilon} z_t$ where $z_t \sim \mathcal{N}(0, 1)$
- As $t \to \infty$ and $\epsilon \to 0$, it can be shown that $x^t \sim p(x)$

# Inversion Model

- We want a generative model where parameter estimation, sampling and inference are all easy
- Plus, there should be many latent variables to support variations in the training dataset
- Possible solution: *invertible latent variable models*!
- $Z \sim \pi$, $X = g_\theta(Z), Z = h(X)$ where $h = g_\theta^{-1}$
- Some special neural networks are invertible

# Inversion Model

- Inversion formula: helps us to define distribution of the transformed variable $X$ in terms of the original latent variable $Z$
- $p_X(x) = p_Z(h(x))|h'(x)|$ where $h$ is the inverse function
- In case $h'(x)$ is a matrix, $|.|$ is the determinant
- Example: if $X = AZ$ where $A$ is an invertible matrix, then $h(X) = A^{-1}X$, and $h'(X) = A^{-1}$
- So according to formula, $p_X(x) = p_Z(A^{-1}x)\frac{1}{det(A)}$ where $p_Z$ is the prior distribution on latent variable $Z$
- This becomes more complex if we consider non-linear transformations

# Flow of Transformations

- Let us consider a sequence of latent variables $Z^0, Z^1, \ldots, Z^L$ as in a deep generative model
- Flow of transformations: $Z^L \sim \pi$, $Z^{L-1} = f_L(Z^L)$, ..., $Z^1 = f_2(Z^2)$, $X = f_1(Z^1)$
- Essentially a composition function $X = f(Z^L) = f_1(f_2(\ldots f_L(Z^L))$
- Each of these functions is invertible (deterministic or probabilistic)
- Then the output distribution $p_X(x;\theta) = p_Z(f^{-1}(x)) \prod_{l=1}^{L} |det(\frac{\partial (f_\theta^l)^{-1}(Z^l)}{\partial Z^l})|$
- Here $\theta$ is the set of parameters of all the invertible distributions
- This allows us to define a closed-form distribution over the output variable $X$ (we could not evaluate it in case of DBM due to the partition function)

# Learning and Inference Problem

- Sampling from flow models is easy (just follow the transformations in sequence)
- The main problem: estimate the parameters $\theta$
- $\theta^* = argmax_\theta log(p(X;\theta)) = \sum_i (p_Z(f^{-1}(X_i)) + log|det(\frac{\partial (f_\theta^l)^{-1}(X)}{\partial X})|_{X=X_i})$
- Inference: need to apply the inversion formula to estimate $Z$ from $X$
- In case of flows, we need to compute the Jacobian matrix as the derivative
- The $(i,j)$-th entry is $\frac{\partial h_i}{\partial z_j}$ where $h_i = f_j^{-1}$
- In case of deep models with $L$ latent variables, it will be expensive to calculate the determinant of an $L \times L$ Jacobian
- Solution: the Jacobian should be triangular, as $h_L$ involves just $z_L$, $h_l$ involves $\{z_L, \ldots, z_l\}$ etc

# Examples of Flow Models

- NICE: Non-linear Independent Components Estimation
- $L$ latent variables and $L$ observations
- $X_{1:l} = s_{1:l} \odot Z_{1:l}$, $X_{l+1:L} = Z_{l+1:L} + g_\theta(Z_{1:l})$
- Alternative formulation: $X_{l+1:L} = Z_{l+1:L} \odot exp(-h_\theta(Z_{1:l})) + g_\theta(Z_{1:l})$
- Here $s = \{s_1, \ldots, s_l\}$ is a set of scaling factors and $g_\theta, h_\theta$ are non-linear functions that may be represented by neural networks
- Inverse mapping: $Z_{1:l} = X_{1:l} \odot \frac{1}{s_{1:l}}$, $Z_{l+1:L} = X_{l+1:L} - g_\theta(X_{1:l}) \odot \frac{1}{s_{1:l}}$
- The Jacobian here is upper triangular by design
- Can be trained on image datasets to generate new images

# Examples of Flow Models

- It is possible to build an invertible neural network using specific invertible units
- MintNet uses Masked Convolutions, which are invertible unlike normal convolutions
- Masked convolution causes the input elements (eg. image pixels) to have a sequential generative structure, and corresponding Jacobian is triangular
- In a masked convolution filter, the receptive field is restricted by setting some elements to 0

# Thank you!