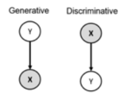
1) Intro -- Deep Generative Models

- **Generative Model**: Model a generative process that defines a joint distribution over random variables and their stochastic interactions.

- **Usage**: ***Data Generation*** *(sampling),* ***Density estimation*** *(anomaly detection),* ***Unsupervised representation learning*** *(Learn common features)*

- **Product rule**: Allows us to *factorize* the joint distribution in two manners -- p (x, y) = p(x|y) p(y) = p(y|x) p(x)
(Joint distribution could be represented as a product of a marginal distribution and a conditional distribution)
- **Sum rule**: If want calculate marginal distribution over one of the variables, must integrate/sum out other variable -- $p(\mathbf{x}) = \sum_y p\,(\mathbf{x}, \mathbf{y})$



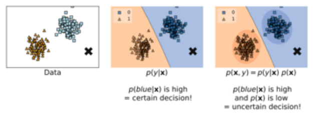
- **Deep G.M vs G.M:** Distribution is parameterized using deep neural networks (Rich and flexible parameterization of distributions)

- **Continuous Random Variable:**
  - If X is a continuous random variable, we can usually represent it using its probability density function (PDF) p_x
  - If X is a continuous **random vector**, we can usually represent it using its joint probability density function:

- **Discriminative approach VS Generative approach:**
  - *Discriminative approach model only p(y|x) ... Called a discriminative model because it is only useful for discriminating Y 's label when given X*
  - *Problem: Adding noise to images could result in completely false classification.*
  - *NNs used to parameterize conditional distribution p(y|x) seem to lack understanding, and no uncertainty quantification...*
  - *Generative approaches are key for models that understand and quantify beliefs (and for that, <u>estimating p(x) is crucial!!</u>)*

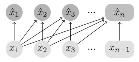
- **Explicit vs Implicit Generative Models:**
- *Explicit: Define a density function of the distribution*
- *Implicit: Learn mapping that generates samples by transforming an easy-to-sample random variable*

- *p(x, y) = p(x|y) p(y)* ***VS*** *p(x, y) = p(y|x)* ***p(x)***



- Now, the following models answers the key question of **<u>how to represent p(x):</u>**

2) Autoregressive Models

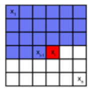


- Autoregressive approach casts p(x) as a product of conditional distributions...
- Define the joint distribution using conditionals over each feature, given the values of the previous features.
- The distribution over x is represented in an autoregressive manner: $p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i \mid x_{<i})$, *where $x_{<i}$ denotes all x's up to the index i.*

- Main idea: Leverage the chain rule of probability to factorize the joint probability distribution over the input space into a product of conditional distributions
- Each conditional probability is parameterized by a deep neural network whose architecture is chosen according to the required inductive biases for the data.

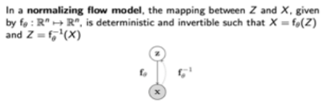
- Big challenge is to calculate these conditional likelihoods $p(x_i \mid x_1, \ldots, x_{i-1})$.
- How can we define these complex distributions in an expressive model that is also tractable and scalable? One solution is to use universal approximators (ex: DNN)

- Drawback: Parameterized in an autoregressive manner, hence, sampling is rather a slow process



- Example: PixelCNN
  - A pixel of interest i (in red) is defined by all previous pixels (in blue).
  - The PixelCNN can model their associations using convolutional layers along a deep neural network.

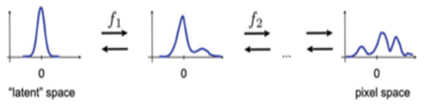
3) Normalizing Flows


In a **normalizing flow model**, the mapping between Z and X, given by $f_\theta : \mathbb{R}^n \mapsto \mathbb{R}^n$, is deterministic and invertible such that $X = f_\theta(Z)$ and $Z = f_\theta^{-1}(X)$

- Technique for building complex probability distributions by transforming simple ones
- Transform the simple distribution with some function *f*
- *f* represents a composition of a sequence of invertible transformations that we are trying to learn

- We are mapping x to its inverse z
- Evaluating the density of this z under p(z)
- Then multiplying by some scalar magnitude (Space Expansion/Contraction indice)
- The change of variables formula provides a principled manner of expressing a density of a random variable by transforming it with invertible transformation *f*.
- $\boldsymbol{p_\theta(x) = p(z)\,|\,J_{f(x)}|}$, **where $\boldsymbol{z = f_\theta(x)}$ and $J_{f(x)}$ denotes the Jacobian matrix.**

- Learn a Sequence of invertible transformations (f_k: $\mathbb{R}^D \rightarrow \mathbb{R}^D$)
- Start with a simple known distribution
- Then sequentially apply invertible transformations to obtain flexible distribution



- Each f is invertible (bijective)
$x = f_\theta(z) = f_k \circ \ldots \circ f_2 \circ f_1(z)$ $\qquad$ $f: Z \dashrightarrow X$ & $f^{-1}(x) = z$

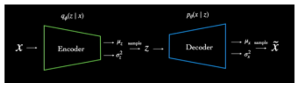
- Learning objective: log-likelihood function (Maximum likelihood objective where we directly maximize the density assigned to x by the model)
- Sampling via forward transformation ($z \rightarrow x$): $z \sim p_Z(z)$ $\qquad$ $x = f_\theta(z)$
- Latent representations inferred via inverse transformation: $z = f^{-1}_\theta(x)$

- Role of magnitude of the determinant of the Jacobian of F inverse:
  - Indicates how much transformation locally expands/contracts space
  - Necessary to ensure new density function p(x) satisfy requirement ∫ Pdf = 1
  - Hence the name "normalized" flows...

4) Latent Variable Models

A) Variational Autoencoder (VAE)

- **Encoder**: Produces a distribution over latent variables z given a data input x
- **Decoder**: Reconstructs the data by producing a distribution over data x given a latent input z



- **Training Objectives**: x observed variables z latent variables
  - Variationnal Lower Bound (Evidence Lower bound)
  - Lower bound on the marginal log likelihood $p_\theta(x)$
  - log $p_\theta(x) \geq$ Variationnal Lower Bound

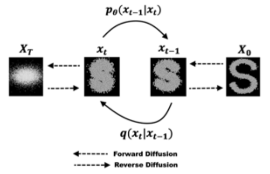
$$\log p_\theta(x) \geq \mathbb{E}_{q(z|x)}[\log p_\theta(x \mid z)] - D_{\mathrm{KL}}(q(z \mid x) \parallel p_\theta(z))$$

1) Likelihood term encourages model to max expected density assigned to the data
2) KL divergence term encourages the approximate posterior q(z|x) to be similar to the prior on the latent variable p (z)

- (For more details, see book or saved papers)

- Use an explicit density estimation but define an intractable density function with latent variables that cannot be optimized directly.
- To train the model, optimize the lower bound of likelihood instead (approximate density);
- Optimize the log-likelihood of the data by maximizing the evidence lower bound (**ELBO**)
- All distributions must be defined upfront and, therefore, they are called *prescribed models*.

B) Deep Diffusion Models



- Main idea: Iteratively destroy the structure in data through a forward diffusion process
- Afterward, learn a reverse diffusion process to restore the structure in data.
- *Inspiration from non-equilibrium statistical physics*

- **Two step process**: forward diffusion process and the reverse process (reconstruction)
1) Forward diffusion process: gaussian noise is introduced successively until the data becomes all noise. (Markov Chain) (No learning)
2) Reverse/ reconstruction process: Undoes noise by **learning the conditional probability densities using a DNN.**

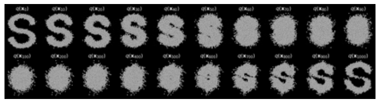
- Reverse process requires estimation of probability density at an earlier time step given the current state of the system.
- Hence generating data sample from isotropic Gaussian noise

- The forward noise step is restricted to be small (low ambiguity about x_{t-1})
- Hence we can model the posterior of the forward step that is $q(x_{t-1}|x_t)$ with a unimodal gaussian (eliminating contributions other modes)
- Shown theoretically that in the limit of infinitesimal step sizes, the true reverse process will have the same functional form as the forward process, which is a key principle used for diffusion models [1949 Feller] where we parameterize each learned reverse step to also be a unimodal diagonal gaussian.

- Unlike the forward process, the estimation of previous state from the current state requires the knowledge of all the previous gradients
- Which we can't obtain without having a learning model that can predict such estimates.
- Therefore, we train a NN that **<u>estimates the $p_\theta(x_{t-1}|x_t)$</u>** based on learned weights θ and current state at time t. *(Input: Sample x AND t)*
- Take t as input also as different timesteps are associated with different noise levels (model can learn to undo these individually)

- Model architecture often simpler compared to other architectures *(Single network vs VAE's 2 networks)*
  - The input layer has the same input size as that of the data dimensions.
  - There can be multiple hidden layers depending on the depth of the network requirements.
  - The middle layers are linear layers with respective activation functions.
  - The final layer is again of the same size as that of the original input layer, thereby reconstructing the original data.

- **Training**:
- Not maximum likelihood objective (max density assigned to x0 by model) as if we try to calculate p(x0), we see that we have to marginalize over all the possible trajectories/ways we could have arrived at x0 when starting from a noise sample
- As this is intractable, we maximize a lower bound, like the one seen in VAEs
- **Latent Variables: ($x_1, x_2, \ldots, x_T$) and Observed Variable: (x_0)**
- Forward process in diffusion models as analogous to the VAE encoder producing latency from data (fixed for this case)
- Reverse process as analogous to the VAE decoder producing data from latents
- See links for details about training targets *(https://youtu.be/fbLgFrlTnGU + books)*



- **Sample conditionally given some variable of interest** (ex: class label):
- One way: Feed the conditioning variable y as additional input during training
- Model should learn to use y as a helpful hint about what it should be reconstructing in practice

- *Rest coming at the end of this semester (And further details on each models, applications, comparaison, ...)*