1 Generative modelling

Learn $p_{\text{model}} \approx p_{\text{data}}$, sample from p_{model}.

- Explicit density:
  - Approximate:
    - Variational: VAE, Diffusion
    - Markov Chain: Boltzmann machine
  - Tractable:
    - Autoregressive: FVSBN/NADE/MADE, Pixel(C/R)NN, WaveNet/TCN, Autor. Transf.,
    - Normalizing Flows
- Implicit density:
  - Direct: Generative Adversarial Networks
  - MC: Generative Stochastic Networks

Autoencoder: $X \rightarrow Z \rightarrow X$, $g \circ f \approx$ id, f and g are NNs. Optimal linear autoencoder is PCA. Undercomplete: $|Z| < |X|$, else overcomplete. Overcomp. is for denoising, inpainting. Latent space should be continuious and interpolable. Autoencoder spaces are neither, so they are only good for reconstruction.

2 Variational AutoEncoder (VAE)

Sample z from prior $p_\theta(z)$, to decode use conditional $p_\theta(x \mid z)$ defined by a NN.

$D_{\text{KL}}(P \| Q) \coloneqq \int_x p(x) \log \frac{p(x)}{q(x)} \mathrm{d}x$: KL divergence, measure similarity of prob. distr. $D_{\text{KL}}(P\|Q) \neq D_{\text{KL}}(Q\|P)$, $D_{\text{KL}}(P\|Q) \geq 0$

Likelihood $p_\theta(x) = \int_z p_\theta(x \mid z) p_\theta(z) \mathrm{d}z$ is hard to max., let enc. NN be $q_\phi(z \mid x)$, $\log p_\theta(x^i) = \mathbb{E}_z \left[\log p_\theta(x^i \mid z)\right] - D_{\text{KL}}(q_\phi(z \mid x^i) \| p_\theta(z)) + D_{\text{KL}}(q_\phi(z \mid x^i) \| p_\theta(z \mid x^i))$. Red is intractable, use ≥ 0 to ignore it; Orange is reconstruction loss, clusters similar samples; Purple makes posterior close to prior, adds cont. and interp. Orange – Purple is **ELBO**, maximize it.

$$x \xrightarrow{\text{enc}} \mu_{z|x}, \Sigma_{z|x} \xrightarrow{\text{sample}} z \xrightarrow{\text{dec}} \mu_{x|z}, \Sigma_{x|z} \xrightarrow{\text{sample}} \hat{x}$$

Backprop through sample by reparametr.: $z = \mu + \sigma\epsilon$. For inference, use μ directly.

Disentanglement: features should correspond to distinct factors of variation. Can be done with semi-supervised learning by making z conditionally independent of given features y.

2.1 β-VAE

Disentangle by $\max_{\theta,\phi} \mathbb{E}_x \left[\mathbb{E}_{z \sim q_\phi} \log p_\theta(x \mid z)\right]$ s.t. $D_{\text{KL}}(q_\phi(z \mid x) \| p_\theta(z)) < \delta$, with KKT: max Orange – βPurple.

3 Autoregressive models

Autoregression: use data from the same input variable at previous time steps

Discriminative: $P(Y \mid X)$, generative: $P(X, Y)$, maybe with Y missing. Sequence models are generative: from $x_i \ldots x_{i+k}$ predict x_{i+k+1}.

Tabular approach: $p(\mathbf{x}) = \prod_i p(x_i \mid \mathbf{x}_{<i})$, needs 2^{i-1} params. Independence assumption is too strong. Let $p_{\theta_i}(x_i \mid \mathbf{x}_{<i}) = \text{Bern}(f_i(\mathbf{x}_{<i}))$, where f_i is a NN. **Fully Visible Sigmoid Belief Networks**: $f_i = \sigma(\alpha_0^{(i)} + \boldsymbol{\alpha}^{(i)} \mathbf{x}_{<i}^\intercal)$, complexity n^2, but model is linear.

Neural Autoregressive Density Estimator: add hidden layer. $\mathbf{h}_i = \sigma(\mathbf{b} + \mathbf{W}_{.,<i}\mathbf{x}_{<i})$, $\hat{x}_i = \sigma(c_i + \mathbf{V}_{i,.}\mathbf{h}_i)$. Order of $\mathbf{x}$ can be arbitrary but fixed. Train by max log-likelihood in $O(TD)$, can use 2nd order optimizers, can use **teacher forcing**: feed GT as previous output.

Extensions: Convolutional; Real-valued: conditionals by mixture of gaussians; Order-less and deep: one DNN predicts $p(x_k \mid x_{i_1} \ldots x_{i_j})$.

Masked Autoencoder Distribution Estimator: mask out weights s.t. no information flows from $x_d \ldots$ to $\hat{x}_d$. Large hidden layers needed. Trains as fast as autoencoders, but sampling needs D forward passes.

PixelRNN: generate pixels from corner, dependency on previous pixels is by RNN (LSTM). **PixelCNN**: also from corner, but condition by CNN over context region (perceptive field) $\Rightarrow$ parallelize. For conditionals use masked convolutions. Channels: model R from context, G from R + cont., B from G + R + cont. Training is parallel, but inference is sequential $\Rightarrow$ slow. Use conv. stacks to mask correctly.

NLL is a natural metric for autoreg. models, hard to evaluate others.

WaveNet: audio is high-dimensional. Use dilated convolutions to increase perceptive field with multiple layers.

AR does not work for high res images/video, convert the images into a series of tokens with an AE: Vector-quantized VAE. The codebook is a set of vectors. $x \xrightarrow{\text{enc}} z \xrightarrow{\text{codebook}} z_q \xrightarrow{\text{dec}} \hat{x}$.

We can run an AR model in the latent space.

3.1 Attention

$\mathbf{x}_t$ is a convex combination of the past steps, with access to all past steps. For $X \in \mathbb{R}^{T \times D}$: $K = XW_K, V = XW_V, Q = XW_Q$. Check pairwise similarity between query and keys via dot product: let attention weights be $\boldsymbol{\alpha} = \text{Softmax}(QK^\intercal/\sqrt{D})$, $\boldsymbol{\alpha} \in \mathbb{R}^{1 \times T}$. Adding mask M to avoid looking into the future:

$$X = \text{Softmax}\left(\frac{(XW_Q)(XW_K)^\intercal}{\sqrt{D}} + M\right)(XW_V)$$

Multi-head attn. splits W into h heads, then concatenates them. Positional encoding injects information about the position of the token. Attn. is $O(T^2 D)$.

4 Normalizing Flows

VAs dont have a tractable likelihood, AR models have no latent space. Want both. Change of variable for $x = f(z)$: $p_x(x) = p_z(f^{-1}(x))\left|\det \frac{\partial f^{-1}(x)}{\partial x}\right| = p_z(f^{-1}(x))\left|\det \frac{\partial f(z)}{\partial z}\right|^{-1}$. Map $Z \rightarrow X$ with a deterministic invertible f_θ. This can be a NN, but computing the determinant is $O(n^3)$. If the Jacobian is triangular, the determinant is $O(n)$. To do this, add a coupling layer:

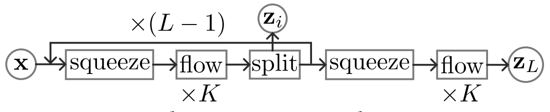
$$\begin{pmatrix} y^A \\ y^B \end{pmatrix} = \begin{pmatrix} h(x^A, \beta(x^B)) \\ x^B \end{pmatrix}$$

, where β is any model, and h is elementwise.

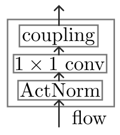
$$\begin{pmatrix} x^A \\ x^B \end{pmatrix} = \begin{pmatrix} h^{-1}(y^A, \beta(y^B)) \\ y^B \end{pmatrix}, J = \begin{pmatrix} h' & h'\beta' \\ 0 & 1 \end{pmatrix}$$

Stack these for expressivity, $f = f_k \circ \ldots f_1$. $p_x(x) = p_z(f^{-1}(x)) \prod_k \left|\det \frac{\partial f_k^{-1}(x)}{\partial x}\right|$.

Sample $z \sim p_z$ and get $x = f(z)$.



- Squeeze: reshape, increase chan.
- ActNorm: batchnorm with init. s.t. output $\sim \mathcal{N}(0, \mathbf{I})$. $\mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$, $\mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$, $\log\det = H \cdot W \cdot \sum_i \log|\mathbf{s}_i|$: linear.



- 1×1 conv: permutation along channel dim. Init $\mathbf{W}$ as rand. ortogonal $\in \mathbb{R}^{C \times C}$ with $\det \mathbf{W} = 1$. $\log\det = H \cdot W \cdot \log|\det \mathbf{W}|$: $O(C^3)$. Faster: $\mathbf{W} \coloneqq \mathbf{PL}(\mathbf{U} + \text{diag}(s))$, where $\mathbf{P}$ is a random fixed permut. matrix, $\mathbf{L}$ is lower triang. with 1s on diag., $\mathbf{U}$ is upper triang. with 0s on diag., s is a vector. Then $\log\det = \sum_i \log|\mathbf{s}_i|$: $O(C)$

Conditional coupling: add parameter $\mathbf{w}$ to β.

SRFlow: use flows to generate many high-res images from a low-res one. Adds affine injector between conv. and coupling layers. $\mathbf{h}^{n+1} = \exp(\beta^n_{\theta,s}(\mathbf{u})) \cdot \mathbf{h}^n + \beta_{\theta,b}(\mathbf{u})$, $\mathbf{h}^n = \exp(-\beta^n_{\theta,s}(\mathbf{u})) \cdot (\mathbf{h}^{n+1} - \beta^n_{\theta,b}(\mathbf{u}))$, $\log\det = \sum_{i,j,k} \beta^n_{\theta,s}(\mathbf{u}_{i,j,k})$.

StyleFlow: Take StyleGAN and replace the network $\mathbf{z} \rightarrow \mathbf{w}$ (aux. latent space) with a normalizing flow conditioned on attributes.

C-Flow: condition on other normalizing flows: multimodal flows. Encode original image $\mathbf{x}_B^1$: $\mathbf{z}_B^1 = f_\phi^{-1}(\mathbf{x}_B^1 \mid \mathbf{x}_A^1)$; encode extra info (image, segm. map, etc.) $\mathbf{x}_A^2$: $\mathbf{z}_A^2 = g_\theta^{-1}(\mathbf{x}_A^2)$; generate new image $\mathbf{x}_B^2$: $\mathbf{x}_B^2 = f_\phi(\mathbf{z}_B^1 \mid \mathbf{z}_A^2)$.

Flows are expensive for training and low res. The latent distr. of a flow needn't be $\mathcal{N}$.

5 Generative Adversarial Networks (GANs)

Log-likelihood is not a good metric. We can have high likelihood with poor quality by mixing in noise and not losing much likelihood; or low likelihood with good quality by remembering input data and having sharp peaks there.

Generator $G: \mathbb{R}^Q \rightarrow \mathbb{R}^D$ maps noise z to data, **discriminator** $D: \mathbb{R}^D \rightarrow [0, 1]$ tries to decide if data is real or fake, receiving both gen. outputs and training data. Train D for k steps for each step of G.

Training GANs is a min-max process, which are hard to optimize. $V(G, D) = \mathbb{E}_{\mathbf{x} \sim p_{\text{d}}} \log(D(\mathbf{x})) + \mathbb{E}_{\hat{\mathbf{x}} \sim p_{\text{m}}} \log(1 - D(\hat{\mathbf{x}}))$

For G the opt. $D^* = p_{\text{d}}(\mathbf{x})/(p_{\text{d}}(\mathbf{x}) + p_{\text{m}}(\mathbf{x}))$.

Jensen-Shannon divergence (symmetric): $D_{\text{JS}}(p\|q) = \frac{1}{2}D_{\text{KL}}(p\|\frac{p+q}{2}) + \frac{1}{2}D_{\text{KL}}(p\|\frac{p+q}{2})$.

Global minimum of $D_{\text{JS}}(p_{\text{d}}\|p_{\text{m}})$ is the glob. min. of $V(G, D)$ and $V(G, D^*) = -\log(4)$.

If G and D have enough capacity, at each update step D reaches D^* and p_{m} improves $V(p_{\text{m}}, D^*) \propto \sup_D \int_{\mathbf{x}} p_{\text{m}}(\mathbf{x}) \log(-D(\mathbf{x}))\mathrm{d}x$, then $p_{\text{m}} \rightarrow p_{\text{d}}$ by convexity of $V(p_{\text{m}}, D^*)$ wrt. p_{m}. These assumptions are too strong.

If D is too strong, G has near zero gradients and doesn't learn ($\log'(1 - D(G(z))) \approx 0$). Use gradient ascent on $\log(D(G(z)))$ instead.

Model collapse: G only produces one sample or one class of samples. Solution: **unrolling** — use k previous D for each G update.

DCGAN: pool $\rightarrow$ strided convolution, batchnorm, no FC, ReLU for G, LeakyReLU for D.

Wasserstein GAN: different loss, gradients don't vanish. Adding gradient penalty for D

stabilizes training. Hierarchical GAN: generate low-res image, then high-res during training. StyleGAN: learn intermediate latent space $\mathcal{W}$ with FCs, batchnorm with scale and mean from $\mathcal{W}$, add noise at each layer.
GAN **inversion**: find z s.t. $G(z) \approx x \Rightarrow$ manipulate images in latent space, inpainting. If G predicts image and segmentation mask, we can use inversion to predict mask for any image, even outside the training distribution.

5.1 3D GANs

3D GAN: voxels instead of pixels. Platonic-GAN: 2D input, 3D output differentiably rendered back to 2D for D.
HoloGAN: 3D GAN + 2D superresolution GAN
GRAF: radiance fields more effic. than voxels
GIRAFFE: GRAF + 2D conv. upscale
EG3D: use 3 2D images from StyleGAN for features, project each 3D point to tri-planes.

5.2 Image Translation

E.g. sketch $X \longrightarrow$ image Y. Pix2Pix: $G : X \longrightarrow Y$, $D : X, Y \longrightarrow [0, 1]$. GAN loss $+L_1$ loss between sketch and image. Needs pairs for training.
CycleGAN: unpaired. Two GANs $F : X \longrightarrow Y, G : Y \longrightarrow X$, cycle-consistency loss $F \circ G \approx$ id; $G \circ F \approx$ id plus GAN losses for F and G.
BicycleGAN: add noise input.
Vid2vid: video translation.