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 continuous 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. $D_{\text{KL}}(P\|Q) := \int_x p(x) \log \frac{p(x)}{q(x)} \mathrm{d}x$: 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

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

3 Autoregressive generative 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**: $f_i = \sigma(\alpha_0^{(i)} + \boldsymbol{\alpha}^{(i)}\mathbf{x}_{<i}^{\mathsf{T}})$, complexity n^2, but model is linear.

Neural Autoregressive Density Estimator: add hidden layer. $\mathbf{h}_i = \sigma(\mathbf{b} + \mathbf{W}_{\cdot,<i}\mathbf{x}_{<i})$, $\hat{x}_i = \sigma(c_i + \mathbf{V}_{i,\cdot}\mathbf{h}_i)$. 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 $\boldsymbol{\alpha} = \text{Softmax}(QK^{\mathsf{T}}/\sqrt{D})$, $\boldsymbol{\alpha} \in \mathbb{R}^{1 \times T}$. Adding mask M to avoid looking into the future:

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

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

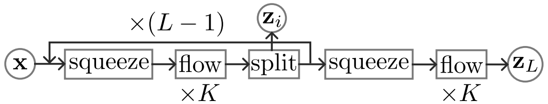
4 Normalizing Flows

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.

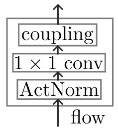
$$\begin{pmatrix} x^A \\ x^B \end{pmatrix} = \begin{pmatrix} h^{-1}(y^A, \beta(y^B)) \\ y^B \end{pmatrix}, J = \begin{pmatrix} h' & h'\beta' \\ 0 & 1 \end{pmatrix}$$

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 $\sim \mathcal{N}(0, \mathbf{I})$ for first minibatch. $\mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$, $\mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$, $\log\det = H \cdot W \cdot \sum_i \log|\mathbf{s}_i|$: 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: $\mathbf{W} := \mathbf{PL}(\mathbf{U} + \text{diag}(s))$, 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. $\mathbf{h}^{n+1} = \exp(\beta_{\theta,s}^n(\mathbf{u})) \cdot \mathbf{h}^n + \beta_{\theta,b}(\mathbf{u})$, $\mathbf{h}^n = \exp(-\beta_{\theta,s}^n(\mathbf{u})) \cdot (\mathbf{h}^{n+1} - \beta_{\theta,b}^n(\mathbf{u}))$, $\log\det = \sum_{i,j,k} \beta_{\theta,s}^n(\mathbf{u}_{i,j,k})$.

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 $V(p_{\text{m}}, D^*) \propto \sup_D \int_{\mathbf{x}} p_{\text{m}}(\mathbf{x}) \log(-D(\mathbf{x}))\mathrm{d}\mathbf{x}$, 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. PlatonicGAN: 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 $X \to$ image Y. Pix2Pix: $G: X \to Y$, $D: X, Y \to [0, 1]$. GAN loss $+L_1$ loss between sketch and image. Needs pairs for training.
CycleGAN: unpaired. Two GANs $F: X \to Y, G: Y \to X$, cycle–consistency loss $F \circ G \approx \text{id}; G \circ F \approx \text{id}$ plus GAN losses for F and G.
BicycleGAN: add noise input.
Vid2vid: video translation.

6 Diffusion models
High quality generations, better diversity, more stable/scalable.
Diffusion (forward) step q: adds noise to $\mathbf{x}_t$ (not learned). Denoising (reverse) step p_θ: removes noise from $\mathbf{x}_t$ (learned).
$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\sqrt{1-\beta}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$
$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I})$
β_t is the variance schedule (monotone ↑). Let $\alpha_t := 1 - \beta_t, \overline{\alpha}_t := \prod \alpha_i$, then $q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\sqrt{\overline{\alpha}_t}\mathbf{x}_0, (1-\overline{\alpha}_t)\mathbf{I}) \Rightarrow \mathbf{x}_t = \sqrt{\overline{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\overline{\alpha}_t}\epsilon$.
Denoising is not tractable naively: $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) q(\mathbf{x}_{t-1}) / q(\mathbf{x}_t)$, $q(\mathbf{x}_t) = \int q(\mathbf{x}_t \mid \mathbf{x}_0) q(\mathbf{x}_0) \mathrm{d}\mathbf{x}_0$.
Conditioning on $\mathbf{x}_0$ we get a Gaussian. Learn model $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \approx q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ by predicting the mean.
$\log p(\mathbf{x}_0) \geq \mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \log\left(\frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right) = \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)} \log p_\theta(\mathbf{x}_0|\mathbf{x}_1) - D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \| p(\mathbf{x}_T)) - \sum_{t=2}^T \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)} D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))$, where orange and purple are the same as in VAEs, and blue are the extra loss functions. In a sense VAEs are 1-step diffusion models.
t-th denoising is just $\arg\min_\theta \frac{1}{2\sigma_q^2(t)} \left\|\mu_\theta - \mu_q\right\|_2^2$, so we want $\mu_\theta(\mathbf{x}_t, t) \approx \mu_q(\mathbf{x}_t, \mathbf{x}_0)$. $\mu_q(\mathbf{x}_t, \mathbf{x}_0)$ can be written as $\frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}\sqrt{\alpha_t}}\epsilon_0$, and $\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}\sqrt{\alpha_t}}\hat{\epsilon}_\theta(\mathbf{x}_t, t)$, so the NN learns to predict the added noise.
Training: img $\mathbf{x}_0, t \sim \text{Unif}(1 \dots T), \epsilon \sim \mathcal{N}(0, \mathbf{I})$, GD on $\nabla_\theta \left\|\epsilon - \epsilon_\theta(\sqrt{\overline{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\overline{\alpha}_t}\epsilon, t)\right\|^2$.
Sampling: $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$, for $t = T$ downto 1: $\mathbf{z} \sim \mathcal{N}(0, I)$ if $t > 1$ else $\mathbf{z} = 0$;
$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t)) + \sigma_t \mathbf{z}$.
$\sigma_t^2 = \beta_t$ in practice. t can be continuous.

6.1 Conditional generation
Add input y to the model.
ControlNet: don't retrain model, add layers that add something to block outputs.
Guidance: mix predictions of a conditional and unconditional model, because conditional models are not diverse.

6.2 Latent diffusion models
High-res images are expensive to model. Predict in latent space, decode with a decoder.

7 Reinforcement learning
Environment is a Markov Decision Process: states S, actions A, reward $r: S \times A \to \mathbb{R}$, transition $p: S \times A \to S$, initial $s_0 \in S$, discount factor γ. r and p are deterministic, can be a distribution. Learn policy $\pi: S \to A$. Value $V_\pi: S \to \mathbb{R}$, the reward from s under π. **Bellman eq.**: $G_t := \sum_{k=0}^\infty \gamma^k R_{t+k+1}, v_\pi(s) := \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']] = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]]$. Can be solved via dynamic programming (needs knowledge of p), Monte-Carlo or Temporal Difference learning.

7.1 Dynamic programming
Value iteration: compute optimal v_*, then π_*.
Policy iteration: compute v_π and π together.
For any V_π the greedy policy (optimal) is $\pi'(s) = \arg\max_{a \in A}(r(s, a) + \gamma V_\gamma(p(s, a)))$.
Bellman optimality: $v_*(s) = \max_a q_*(s, a) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')] \Rightarrow$ update step: $V^*_{\text{new}}(s) = \max_{a \in A}(r(s, a) + \gamma V^*_{\text{old}}(s'))$, when $V^*_{\text{old}} = V^*_{\text{new}}$, we have optimal policy.
Converges in finite steps, more efficient than policy iteration. But needs knowledge of p, iterates over all states and $O(|S|)$ memory.

7.2 Monte Carlo sampling
Sample trajectories, estimate v_π by averaging returns. Doesn't need full p, is unbiased, but high variance, exploration/exploitation dilemma, may not reach term. state.

7.3 Temporal Difference learning
For each $s \to s'$ by action a update: $\Delta V(s) = r(s, a) + \gamma V(s') - V(s)$. **$\varepsilon$-greedy policy**: with prob. ε choose random action, else greedy.

7.4 Q-learning
Q-value f.: $q_\pi(s, a) := \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$.
SARSA (on-policy): For each $S \to S'$ by action A update: $\Delta Q(S, A) = r(S, A) + \gamma Q(S', A') - Q(S, A)$, $Q(S, A) \mathrel{+}= \alpha \Delta Q(S, A)$, α is LR.
Q-learning (off-policy/offline): $\Delta Q(S, A) = R_{t+1} + \gamma \max_a Q(S', a) - Q(S, A)$
All these approaches do not approximate values of states that have not been visited.

7.5 Deep Q-learning
Use NN to predict Q-values. Loss is $(R + \gamma \max_{a'} Q_\theta(S', a') - Q_\theta(S, A))^2$, backprop only through $Q_\theta(S, A)$. Store history in replay buffer, sample from it for training $\Rightarrow$ no correlation in samples.

7.6 Deep Q-networks
Encode state to low dimensionality with NN.

7.7 Policy gradients
Q-learning does not handle continuous action spaces. Learn a policy directly instead, $\pi(a_t \mid s_t) = \mathcal{N}(\mu_t, \sigma_t^2 \mid s_t)$. Sample trajectories: $p(\tau) = p(s_1, a_1, \dots, s_T, a_T) = p(s_1) \prod \pi(a_t|s_t) p(s_{t+1}|a_t, s_t)$. This is on-policy. Eval: $J(\theta) := \mathbb{E}_{\tau \sim p_\theta(\tau)}[\sum_t \gamma^t r(s_t, a_t)]$. To optimize, need to compute $\mathbb{E}$ (see proofs).
REINFORCE: MC sampling of τ. To reduce variance, subtract baseline $b(s_t)$ from reward.

7.8 Actor-Critic
$\nabla_\theta J(\theta) = \frac{1}{N} \sum_i \sum_t \nabla \log \pi_\theta(a_t^i \mid s_t^i)(r(s_t^i, a_t^i) + \gamma V(s_{t+1})^i - V(s_t^i)))$. π = actor, V = critic.

7.9 Motion synthesis
Data-driven: bad perf. out of distribution, needs expensive mocap.
DeepMimic: RL to imitate reference motions while satisfying task objectives.
SFV: use pose estimation: videos $\to$ train data.

8 Neural Implicit Representations
Voxels/volum. primitives are inefficient (cubic complexity). Meshes have limited granularity and have self-intersections. **Implicit representation**: $S = \{x \mid f(x) = 0\}$. Usually represented as signed distance function values on a grid. But this is again cubic. By UAT, approx. f with NN. **Occupancy networks**: predict probability that point is inside the shape. **DeepSDF**: predict SDF. Both conditioned on input (2D image, class, etc.). Continuous, any topology/resolution, memory-efficient. NFs can model other properties (color, force, etc.).

8.1 Learning 3D Implicit Shapes
Inference: to get a mesh, sample points, predict occupancy/SDF, use marching cubes.

8.1.1 From watertight meshes
Sample points in space, compute GT occupancy/SDF, CE loss.

8.1.2 From point clouds
Only have samples on the surface. Weak supervision: loss $= |f_\theta(x_i)|^2 + \lambda \mathbb{E}_x (\|\nabla_x f_\theta(x)\| - 1)^2$, edge points should have $\|\nabla f\| \approx 1$ by def. of SDF, $f \approx 0$.

8.1.3 From images
Need differentiable rendering 3D $\to$ 2D. **Differentiable Volumetric Rendering**: for a point conditioned on encoded image, predict occupancy $f(x)$ and RGB color $c(x)$. Forward: for a pixel, raymarch and root find $x: f(x) = 0$ with secant. Set pixel color to $c(x)$.

9 Proofs

Policy gradients $J(\theta) = \mathbb{E}_{\tau\sim p(\tau)}[r(\tau)] = \int p(\tau)r(\tau)\mathrm{d}\tau$. $\nabla_\theta J(\theta) = \int \nabla_\theta p(\tau)r(\tau)\mathrm{d}\tau = \int p(\tau)\nabla_\theta \log p(\tau)r(\tau)\mathrm{d}\tau = \mathbb{E}_{\tau\sim p(\tau)}[\nabla_\theta \log p(\tau)r(\tau)] = \mathbb{E}_{\tau\sim p(\tau)}[\nabla_\theta \log p(\tau)r(\tau)]$.
$\log p(\tau) = \log[p(s_1)\prod \pi_\theta(a_t \mid s_t)p(s_{t+1} \mid a_t, s_t)] = 0 + \sum_t \log \pi_\theta(a_t \mid s_t) + 0$
$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p(\tau)}[(\sum_t \nabla \log p_\theta(a_t^i \mid s_t^i))(\sum_t \gamma^t r(s_t^i, a_t^i))]$: max likelihood, trajectory reward scales the gradient.

Implicit differentiation $\frac{\mathrm{d}y}{\mathrm{d}x}$ of $x^2 + y^2 = 1$:
$\frac{\mathrm{d}}{\mathrm{d}x}(x^2 + y^2) = \frac{\mathrm{d}}{\mathrm{d}x}(1) \Rightarrow \frac{\mathrm{d}}{\mathrm{d}x}x^2 + \frac{\mathrm{d}}{\mathrm{d}x}y^2 = 0 \Rightarrow 2x + (\frac{\mathrm{d}}{\mathrm{d}y}y^2)\frac{\mathrm{d}y}{\mathrm{d}x} = 0$
$\Rightarrow 2x + 2y\frac{\mathrm{d}y}{\mathrm{d}x} = 0 \Rightarrow \frac{\mathrm{d}y}{\mathrm{d}x} = -\frac{x}{y}$

10 Appendix

Secant Method Line $(x_0, f(x_0)) \to (x_1, f(x_1))$, approx.: $y = \frac{f(x_1)-f(x_0)}{x_1-x_0}(x - x_1) + f(x_1)$, $y = 0$ at $x_2 = x_1 - f(x_1)\frac{x_1-x_0}{f(x_1)-f(x_0)}$. Approximates Newton's method without derivatives.