1 ML Basics

MLE: $\hat{\theta} = \arg\max_\theta \sum_{i=1}^{n} \log p(x_i \mid \theta)$
MAP: $\theta^* \in \arg\max p(\theta \mid X, y) = \arg\max p(y \mid X, \theta) p(\theta)$
CE: $H(p,q) = -\mathbb{E}_p[\log q] = -\sum_x p(x) \log q(x)$
KL: $\mathrm{KL}(p \parallel q) = \mathbb{E}_p[\log(p/q)] = \sum_x p(x) \log \frac{p(x)}{q(x)}$
Min. KL $\iff$ max. MLE (for infinite sample size)

$H(p,q) = H(p) + \mathrm{KL}(p \parallel q)$

For empirical distribution $p = \sum_i \delta_{x_i}$: $H(p,q) = \mathrm{NLL}(q) = H(p) + \mathrm{KL}(p \parallel q)$
BCE: $L(\theta) = -\sum_i [y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$

1.1 Neural Networks

Perceptron: $\hat{y} = \mathbf{1}\{\mathbf{w}^\top \mathbf{x} + b > 0\}$.
Learning: $\theta \leftarrow \theta + \eta(y_i - \hat{y}_i)\mathbf{x}_i$
Converges in finite time iff data is linearly separable.
Multi-Layer Perceptron: $\hat{y} = \sigma(\mathbf{W}_k \sigma(\mathbf{W}_{k-1} \cdots \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \cdots + \mathbf{b}_{k-1}) + \mathbf{b}_k)$
Universal Approx. Thr.: For any cont. func. f on compact set, $\exists$ NN g with single hidden layer and nonlin. act. s.t. $|g(x) - f(x)| < \varepsilon\ \forall x$.

1.2 Activation Functions

Function	Formula	Derivative	Range
Sigmoid	$\frac{1}{1+e^{-x}}$	$\sigma(x)(1-\sigma(x))$	$(0,1)$
Tanh	$\frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - \tanh^2(x)$	$(-1,1)$
ReLU	$\max(0,x)$	$\mathbf{1}\{x \geq 0\}$	$[0,\infty)$

Properties: Sigmoid/tanh saturate $\Rightarrow$ vanishing gradients. Tanh zero-centered. ReLU: fast convergence, can explode, dead neurons possible.
Vector derivatives: $\frac{\partial \sigma(\mathbf{x})}{\partial \mathbf{x}} = \mathrm{diag}(\sigma(\mathbf{x}) \odot (1 - \sigma(\mathbf{x})))$
$\frac{\partial \tanh(\mathbf{x})}{\partial \mathbf{x}} = \mathrm{diag}(1 - \tanh^2(\mathbf{x}))$ $\frac{\partial \mathrm{softmax}(\mathbf{x})}{\partial \mathbf{x}} = \mathrm{diag}(\mathbf{y}) - \mathbf{y}\mathbf{y}^\top$

1.3 Backpropagation & Training

Backprop is a way to compute chain rule that avoids recomp. same terms multiple times (Dyn. Prog.)
Chain rule: $\mathbf{y} = g(\mathbf{x}), \mathbf{z} = f(\mathbf{y}) \Rightarrow \frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial \mathbf{z}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}}$
MLP backprop:
Forward: $\mathbf{z}^{(l)} = \mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$, $\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)})$
Backward inputs: $\delta^{(l)} = \delta^{(l+1)} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{z}^{(l)}}$
Backward weights: $\frac{\partial L}{\partial \mathbf{W}^{(l)}_{ij}} = \delta^{(l)} \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{W}^{(l)}_{ij}}$
Useful derivatives: $\frac{\partial \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = \mathbf{A}$, $\frac{\partial \mathbf{x} \odot \mathbf{y}}{\partial \mathbf{x}} = \mathrm{diag}(\mathbf{y})$
SGD: $\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$ (single sample, high variance, may escape local minima)
Momentum: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta L(\theta)$, $\quad \theta \leftarrow \theta + \mathbf{v}$

1.4 Training Techniques & Generalization

Bias-Variance Trade-off: *High bias:* underfitting, model too simple; *High variance:* overfitting, model too complex, sensitive to training data

Effective Model Complexity (EMC): $\mathrm{EMC}_{D,\varepsilon}(\mathcal{T})$ measures model's effective complexity given dataset D, tolerance ε, and training procedure $\mathcal{T}$.
- $\mathrm{EMC}_{D,\varepsilon}(\mathcal{T}) \ll n$: Under-parameterized (increasing complexity decreases test error)
- $\mathrm{EMC}_{D,\varepsilon}(\mathcal{T}) \gg n$: Over-parameterized (increasing complexity decreases test error)
- $\mathrm{EMC}_{D,\varepsilon}(\mathcal{T}) \approx n$: Critical regime (increasing complexity may increase or decrease test error)

Double Descent: Test error decreases, then increases (classical overfitting), then decreases again as model size grows. Occurs at interpolation threshold where model starts fitting training data perfectly.
Grokking: Phenomenon where generalization performance suddenly improves long after achieving perfect training accuracy. Model continues learning meaningful patterns after memorizing train data.

Regularization Techniques:
- L_1 penalty: $\lambda \|\mathbf{w}\|_1$ (promotes sparsity)
- L_2 penalty: $\lambda \|\mathbf{w}\|_2^2$ (weight decay, prevents large weights)
- **Dropout:** Rand. set neurons to 0 during training
- **Data augmentation:** Random transformations (rotation, scaling, noise)
- **Early stopping:** Stop when validation error increases for p consecutive epochs
- **Batch normalization:** Addresses internal covariate shift, normalizes layer inputs

Normalization: *Batch normalization:* Normalize inputs to each layer, addresses internal covariate shift; *Layer normalization:* Normalize across features instead of batch
Residual connections: $\mathbf{h}_{l+1} = f(\mathbf{h}_l) + \mathbf{h}_l$ (skip connections); Prevents vanishing gradients, enables training deeper networks; Allows gradient flow and propagates high-frequency information

2 Convolutional Neural Networks

Neural Findings: (1) Rapid Serial Visual Presentation (RSVP), (2) Hubel & Wiesel receptive fields, (3) HMAX model with simple/complex cells
Historical Models: Neurocognitron, LeNet-5.
Image Filtering: Modify pixels in image based on some function of a local neighborhood of pixels.
Three Key Properties:
(1) **Linear:** $T(\alpha\mathbf{u} + \beta\mathbf{v}) = \alpha T(\mathbf{u}) + \beta T(\mathbf{v})$
(2) **Translation Invariant:** $T(f(\mathbf{u})) = T(\mathbf{u})$
(3) **Translation Equivariant:** $T(f(\mathbf{u})) = f(T(\mathbf{u}))$

Any linear, shift-equivariant transform can be written as a convolution

Advantages over FNN: *Parameter sharing:* Same weights reused across spatial locations; *Local connectivity:* Each neuron connects only to local region.

2.1 Convolution Operations

Correlation ($\star$) (template matching):

$\mathbf{I}'(i,j) = \sum_{m=-k}^{k} \sum_{n=-k}^{k} \mathbf{K}(m,n)\mathbf{I}(i+m, j+n)$

Convolution ($*$): $K \in \mathbb{R}^{M \times N}$, $k = \mathrm{floor}(M/2)$

$\mathbf{I}'(i,j) = \sum_{m=-k}^{k} \sum_{n=-k}^{k} \mathbf{K}(-m,-n)\mathbf{I}(i+m, j+n)$

Convolution = Correlation iff $\mathbf{K}(i,j) = \mathbf{K}(-i,-j)$ (kernel is symmetric)

Forward Pass: $z^{(l)}_{i,j} = w^{(l)} * z^{(l-1)}(i,j) + b = \sum_m \sum_n w^{(l)}_{m,n} z^{(l-1)}_{i-m,j-n} + b$
Backprop on inputs z: $\delta^{(l-1)}_{i,j} = \frac{\partial \mathcal{L}}{\partial z^{(l-1)}_{i,j}} = \sum_{i',j'} \delta^{(l)}_{i',j'} w^{(l)}_{i'-i,j'-j} = \delta^{(l)} * \mathrm{FLIP}(w^{(l)})$
Backprop on weights w: $\frac{\partial \mathcal{L}}{\partial w^{(l)}_{m,n}} = \sum_{i',j'} \delta^{(l)}_{i',j'} z^{(l-1)}_{i'-m,j'-n} = \delta^{(l)} * \mathrm{FLIP}(z^{(l-1)})$

$L_{\text{out}} = \left\lfloor \frac{L_{\text{in}} + 2 \cdot \text{Pad} - \text{Dilation} \cdot (\text{Kernel} - 1) - 1}{\text{Stride}} + 1 \right\rfloor$

Parameter Count: $(k^2 \cdot C_{\text{in}} + 1) \cdot C_{\text{out}}$
Connections per neuron: $3k^2$ (for RGB input)

2.2 Pooling Operations

Max Pooling: Forward: $z^{(l)} = \max\{z^{(l-1)}_i\}$
Backward: $\frac{\partial z^{(l)}}{\partial z^{(l-1)}_i} = \mathbf{1}_{\{i=i^*\}}$ and $\delta^{(l-1)} = \{\delta^{(l)}\}_{i^*}$ with $i^* = \text{a}\max_i\{z^{(l-1)}_i\}$
Goals: (1) Increase receptive field exponentially, (2) Noise robustness, (3) Translation invariance
Strided Convolutions: Skip pixels when sliding kernel (stride > 1). Reduces spatial dimensions without pooling. More efficient alternative to conv + pool for downsampling.
Dilated Convolutions: Insert gaps between kernel elements. Increases receptive field without adding parameters or reducing resolution. Preserves spatial detail while capturing wider context.

2.3 Famous CNN Architectures

AlexNet: Smaller filters, more layers, ReLU activation, dropout. **VGG:** Depper, more params
GoogLeNet: Inception modules, 1×1 convs for dimension reduction, auxiliary classification heads
ResNet: Residual connections $\mathbf{h}_{l+1} = f(\mathbf{h}_l) + \mathbf{h}_l$ (1) Better convergence, (2) Propagate high-freq. info

2.4 Fully Convolutional Networks (FCNN)

Applications: Semantic segmentation, image-to-image translation, human pose estimation
Strategy: *Downsample* with convolutions/pooling, then *upsample* to original resolution

Upsampling Methods:
- **Nearest Neighbor:** Duplicate values
- **Bed of Nails:** Place value top-left, fill rest with 0's
- **Max Unpooling:** Remember positions from max-pooling, place values back
- **Strided Upconvolution or Transposed Conv** *(learned!)*: Insert $(s-1)$ zeros between pixels, $(k-p-1)$ padding, then convolve. Transposed Convolution Output Size: $L_{\text{out}} = (L_{\text{in}} - 1) \cdot \text{Stride} - 2 \cdot \text{Pad} + \text{Dilation} \cdot (\text{Kernel} - 1) + \text{OutPad} + 1$

U-Net: Skip connections between corresponding down/upsampling layers; Combines global context (from skip connections) with local information (from previous layer); Essential for preserving fine-grained spatial information.

3 Recurrent Neural Networks

$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \mathbf{W}), \hat{\mathbf{y}}_t = \mathbf{W}_{hy}\mathbf{h}_t$

Vanilla RNN: $\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + b)$
Loss Function: $L = \sum_t L_t = \sum_t L(\mathbf{y}_t, \hat{\mathbf{y}}_t)$
Classification: $\mathrm{softmax}(\mathbf{o}_t)_k = \frac{e^{o_{t,k}}}{\sum_{j=1}^{K} e^{o_{t,j}}} =: \hat{y}_{t,k}$
$\Rightarrow$ *CE Loss:* $L_t = -\sum_k y_{t,k} \log \hat{y}_{t,k}$ (for one-hot $\mathbf{y}_t$)
Use Cases: $1 \to 1$: POS tagging; $1 \to N$: Image captioning; $N \to 1$: Sent. classif.; $N \to N$: Machine trans.

3.1 Backpropagation Through Time (BPTT)

$\frac{\partial L}{\partial \mathbf{W}} = \sum_t \frac{\partial L_t}{\partial \mathbf{W}}$ $\quad \frac{\partial L_t}{\partial \mathbf{W}} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial^+ \mathbf{h}_k}{\partial \mathbf{W}}$
$\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}} = \frac{\partial^+ \mathbf{h}_t}{\partial \mathbf{W}} + \sum_{k=1}^{t-1} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial^+ \mathbf{h}_k}{\partial \mathbf{W}}$
Chain Rule Term: $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{i=k+1}^{t} \mathbf{W}_{hh}^T \mathrm{diag}[f'(\mathbf{h}_{i-1})]$
For Elman RNN: $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{i=k+1}^{t} \mathbf{W}_{hh} \mathrm{diag}(1 - \mathbf{h}_i^2)$
Recursive Formulation: $\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L_t}{\partial \mathbf{h}_t} + \frac{\partial L}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}$
Key Gradients: $\frac{\partial L_t}{\partial \mathbf{o}_t} = \hat{\mathbf{y}}_t - \mathbf{y}_t$ (for CE loss); $\frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} = \mathbf{V}$ (linear output layer); $\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} = \mathrm{diag}(1 - \mathbf{h}_{t+1}^2)\mathbf{W}_{hh}$ (tanh)

3.2 Gradient Vanishing/Exploding

Mathematical Analysis: If $\mathbf{W}_{hh}$ is diagonalizable: $\mathbf{W}_{hh} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$ Then: $\mathbf{h}_t = (\mathbf{Q}\mathbf{\Lambda}^t\mathbf{Q}^T)\mathbf{h}_1 \Rightarrow \mathbf{h}_t$ becomes dominant eigenvector. **Condition:** Let $\lambda_1 = \max \lambda(\mathbf{W}_{hh}), \gamma > \|\mathrm{diag}(f'(\mathbf{h}_{i-1}))\|$
- If $\lambda_1 > \gamma^{-1}$: gradients **explode**
- If $\lambda_1 < \gamma^{-1}$: gradients **vanish**

Problems: (1) Training instabilities (NaN/∞), (2) Hard to capture long-term dependencies, (3) Large gradients jump over local minima

3.3 Solutions to Gradient Problems

Truncated BPTT: Limit sum to last κ steps to reduce computational cost: $\frac{\partial L_t}{\partial \mathbf{W}} \approx \sum_{k=t-\kappa}^{t} \frac{\partial L_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial^+ \mathbf{h}_k}{\partial \mathbf{W}}$

Gradient Clipping: $\theta \leftarrow \begin{cases} \theta - \eta \mathbf{g}, & \|\mathbf{g}\| \le k \\ \theta - \eta \frac{k}{\|\mathbf{g}\|} \mathbf{g}, & \text{otherwise} \end{cases}$

Leaky Unit: Constant error flow to solve vanishing gradient: $\hat{\mathbf{h}}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$, $\mathbf{h}_t = \alpha \mathbf{h}_{t-1} + (1-\alpha)\hat{\mathbf{h}}_t$

3.4 Long Short-Term Memory (LSTM)

Key Idea: Constant error flow through cell state

$$\begin{pmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \mathbf{W} \begin{pmatrix} \mathbf{x}_t \\ \mathbf{h}_{t-1} \end{pmatrix}$$

$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \mathbf{g}$ $\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$

Where $\mathbf{W} \in \mathbb{R}^{4n \times (d+n)}$ (concatenated weight matrix), $\mathbf{i}, \mathbf{f}, \mathbf{o} \in [0,1]$, gate $\mathbf{g} \in \mathbb{R}$

- **Input gate i**: Which values to write to cell state
- **Forget gate f**: What to forget (reset)
- **Output gate o**: Which values to read from cell state to hidden state
- **Gate gate g**: Candidate values to write to cell state

Information Highway: Gates form pathway for easy error propagation through minimal modifications to cell state, solving vanishing gradient problem

4 Autoencoders & Variational Autoencoders

Standard Autoencoder: Encoder $f: \mathbf{x} \in \mathbb{R}^n \to \mathbf{z} \in \mathbb{R}^d$ (where $d \ll n$), Decoder $g: \mathbf{z} \to \hat{\mathbf{x}}$

Loss: $\mathcal{L} = \text{Diff}(\mathbf{x}_{\text{in}}, \mathbf{x}_{\text{recon}})$ (e.g., MSE, cross-entropy)

Equivalent to PCA with closed-form solution

Applications: (1) Denoising AE: corrupt input, recover clean version, (2) Inpainting AE: mask parts of image then recover, (3) 3D human motion: add noise to skeleton, recover with AE

Limitation: Good reconstruction, poor generation. Latent space not well-structured: no continuity, no interpolation, sparse regions.

4.1 Variational Autoencoder (VAE)

$$\mathbf{x} \xrightarrow{\text{enc}} \boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}}, \sigma_{\mathbf{z}|\mathbf{x}} \xrightarrow{\text{sample}} \mathbf{z} \xrightarrow{\text{dec}} \boldsymbol{\mu}_{\mathbf{x}|\mathbf{z}}, \sigma_{\mathbf{x}|\mathbf{z}} \xrightarrow{\text{sample}} \hat{\mathbf{x}}$$

Encoder: $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_{\text{nn}}(\mathbf{x}), \text{diag}(\sigma^2_{\text{nn}}(\mathbf{x})))$

Decoder: $p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}_{\text{nn}}(\mathbf{z}), \text{diag}(\sigma^2_{\text{nn}}(\mathbf{z})))$

Prior: $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$

ELBO: $\log p_\theta(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \text{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z})) + \text{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z}|\mathbf{x}))$

$\text{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z}|\mathbf{x})) \ge 0$ (intractable), we get:

$$\log p_\theta(\mathbf{x}) \ge \text{ELBO}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \text{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z}))$$

- $\mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}|\mathbf{z})]$: **Reconstruction loss** (encourages clustering of similar samples in latent space)
- $-\text{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z}))$: **Latent regularization** (posterior close to prior, ensures continuity and interpolation)

Objective: $\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_i \text{ELBO}(\mathbf{x}_i, \theta, \phi)$

Reparam. Trick: To backpropagate through sampling: $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2) \Leftrightarrow \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \mathbf{z} = \boldsymbol{\mu} + \sigma \odot \boldsymbol{\epsilon}$

Generation: Sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, decode to get $\hat{\mathbf{x}}$

Inference: Use $\boldsymbol{\mu}$ directly (no sampling)

4.2 Disentangled Representations

Goal: Each latent dimension should control a single factor of variation

Semi-supervised: $p(\mathbf{x}|\mathbf{y}, \mathbf{z})$ where $\mathbf{y}$ are known factors, $\mathbf{z}$ are style factors

β-VAE (Unsupervised): Assume $p(\mathbf{x}|\mathbf{z}) \approx p(\mathbf{x}|\mathbf{v}, \mathbf{w})$ where $\mathbf{v}$ are conditionally independent, $\mathbf{w}$ are conditionally dependent

Constrained Opt.: $\max_{\phi,\theta} \mathbb{E}_{\mathbf{x}, q_\phi}[\log p_\theta(\mathbf{x}|\mathbf{z})]$ subject to $\text{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z})) < \delta$

Lagrangian Solution: $\mathcal{L}_{\beta,\phi,\theta} = \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta(\text{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z})) - \delta)$

Where $\beta > 1$ enforces stronger constraint on latent space than standard VAE

4.3 Gaussian Integrals & KL Divergences

For $p = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $q = \mathcal{N}(\mathbf{m}, \mathbf{L})$:

Entropy: $H(p) = \frac{1}{2}[D(\ln 2\pi + 1) + \ln|\boldsymbol{\Sigma}|]$

Cross-entropy: $H(p,q) = \frac{1}{2}[D \ln 2\pi + \ln|\mathbf{L}| + \text{tr}(\mathbf{L}^{-1}\boldsymbol{\Sigma}) + (\boldsymbol{\mu} - \mathbf{m})^T \mathbf{L}^{-1}(\boldsymbol{\mu} - \mathbf{m})]$

KL Divergence: $\text{KL}(p \parallel q) = \frac{1}{2}[\ln \frac{|\mathbf{L}|}{|\boldsymbol{\Sigma}|} + \text{tr}(\mathbf{L}^{-1}\boldsymbol{\Sigma}) - D + (\boldsymbol{\mu} - \mathbf{m})^T \mathbf{L}^{-1}(\boldsymbol{\mu} - \mathbf{m})]$

Diagonal Case: If $\boldsymbol{\Sigma} = \text{diag}\{\sigma_i^2\}, \mathbf{L} = \text{diag}\{l_i^2\}$: $\text{KL}(p \parallel q) = \frac{1}{2}\sum_{i=1}^D \left[\ln \frac{l_i^2}{\sigma_i^2} + \frac{\sigma_i^2}{l_i^2} - 1 + \frac{(\mu_i - m_i)^2}{l_i^2}\right]$

5 Autoregressive Models

Goal: Tractable density estimation using chain rule factorization $p(\mathbf{x}) = \prod_{i=1}^n p(x_i|\mathbf{x}_{<i})$

Discriminative vs. Generative: Discriminative: $P(Y|X)$; Generative: $P(X,Y)$ (possibly missing Y)

Seq. modeling: From $x_i, \ldots, x_{i+k}$ predict x_{i+k+1}

Tabular: 2^{i-1} params for $p(x_i|\mathbf{x}_{<i})$ - not scalable

Fully Visible Sigmoid Belief Networks (FVSBN):

$f_i(\mathbf{x}_{1:i-1}) = \sigma(\alpha_0^{(i)} + \sum_{j=1}^{i-1} \alpha_j^{(i)} x_j)$ $p_{\theta_i}(x_i|\mathbf{x}_{<i}) = \text{Bernoulli}(f_i(\mathbf{x}_{1:i-1}))$ $\Rightarrow \sum_i i = (n^2+n)/2$ params needed

Neural Autoreg. Density Estimator (NADE):

$\mathbf{h}_i = \sigma(\mathbf{b} + \mathbf{W}_{:,1:i-1}\mathbf{x}_{1:i-1})$ $\hat{x}_i = \sigma(c_i + \mathbf{V}_{i,:}\mathbf{h}_i)$

Efficiency trick: $\mathbf{h}_i = \mathbf{h}_{i-1} + \mathbf{W}_i x_i$ (inc. comp.)

Training: $\mathcal{L} = \sum_{t=1}^T \sum_{i=1}^D \log p(x_i^{(t)}|\mathbf{x}_{<i}^{(t)})$

Advantages: (1) $\mathcal{O}(TD)$ complexity, (2) Second-order optimization OK, (3) Arbitrary ordering

Masked Autoencoder Distribution Estimator (MADE) **Idea:** Constrain autoencoder to satisfy autoregressive property; **Constraint:** No computational path between output $\hat{x}_d$ and inputs $x_{d+1:D}$

Implementation: Mask weights to prevent inform. flow from future inputs; **Training trick:** Randomly re-mask during training (similar to dropout)

Trade-off: Fast training like autoencoders, but sampling requires D forward passes.

PixelRNN: Generate pixels from corner using LSTM dependencies; **Cons:** Slow due to seq. generation

PixelCNN: Use CNN over context region instead of LSTM; **Training:** Parallelizable with masked convolutions; **Inference:** Still sequential (slow)

WaveNet (Audio): PixelCNN for audio with dilated convolutions; **Problem:** Audio dimension $> 16000/s$; **Solution:** Dilated conv for exponential receptive field growth

5.1 Self-Attention & Transformers

$$\mathbf{Y} = \text{softmax}\left(\frac{\mathbf{X}\mathbf{W}_Q(\mathbf{X}\mathbf{W}_K)^T}{\sqrt{D}} + \mathbf{M}\right)\mathbf{X}\mathbf{W}_V$$

- Learnable weights: $W_K, W_V, W_Q \in \mathbb{R}^{D \times D}$
- $\mathbf{M}$ is causal mask (prevents accessing future tokens)
- Key: $\mathbf{K} = \mathbf{X}\mathbf{W}_K$, Value: $\mathbf{V} = \mathbf{X}\mathbf{W}_V$, Query: $\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$
- Attention weights: $\boldsymbol{\alpha} = \text{softmax}(\mathbf{Q}\mathbf{K}^T/\sqrt{D})$
- **Intuition:** Compute similarity between queries and keys to determine how much to attend to each value

Positional Encoding: Inject position information with sinusoidal functions

$$\text{PE}_{i,t} = \begin{cases} \sin(\omega_k t) & \text{if } i = 2k \\ \cos(\omega_k t) & \text{if } i = 2k+1 \end{cases}$$

Where $\omega_k = 10000^{-2k/d}$, t position, i embedding

Complexity: $\mathcal{O}(T^2 D)$ vs. $\mathcal{O}(TD^2)$ for RNNs

Multi-head attention: split attention into h heads; each head calculates a chunk of the representation.

5.2 Large Language Models (LLMs) & GPT

LLMs predict next token given seq. of input tokens

- **Long-range dependencies:** Self-attention handles context modeling better than RNNs
- **Scalability:** Transformers parallelizable, scale to billions of parameters and large datasets

Tokenization Process: (1) Convert text into tokens (e.g., sub-words), (2) Assign each token a unique integer, (3) Replace integers with learned embeddings.

GPT Training: *(1) Unsupervised pretraining:* $L_1(X) = \sum_i \log P(x_i|x_{i-1}, x_{i-2}, \ldots, x_{i-k})$; *(2) Supervised finetuning:* $L_2(X,Y) = \sum_{(x,y)} \log P(y|x^m, \ldots, x^m)$; *(2) Combined objective:* $L_3 = L_2(X,Y) + \lambda \cdot L_1(X)$

5.3 Vector-Quantized VAE (VQ-VAE)

Problem: Autoreg. models struggle with high-res. images/video; **Solution:** Convert images to discrete token seq. using learned codebook (Set of learned vectors for quantization). $\mathbf{x} \xrightarrow{\text{enc}} \mathbf{z} \xrightarrow{\text{codebook}} \mathbf{z}_q \xrightarrow{\text{dec}} \hat{\mathbf{x}}$

Run model in quantized latent space (latent tokens are semantically more meaningful than pixel values).

6 Generative Adversarial Networks (GANs)

Two failure modes for Likelihood-based Models:

- **High likelihood, poor quality:** Mixing with noise barely affects likelihood. E.g., let p be a good model, q be noise. Then $0.01p + 0.99q$ has likelihood: $\log(0.01p(\mathbf{x}) + 0.99q(\mathbf{x})) \ge \log p(\mathbf{x}) - \log 100$
- **Low likelihood, good quality:** Overfitting gives sharp peaks on training data

GAN Solution: Two-player adversarial game

- **Generator:** $G: \mathbb{R}^Q \to \mathbb{R}^D$ (noise $\mathbf{z}$ to data $\mathbf{x}$)
- **Discriminator:** $D: \mathbb{R}^D \to [0,1]$ (real vs. fake)

$\min_G \max_D V(G,D) = \mathbb{E}_{p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{p_z}[\log(1 - D(G(\mathbf{z})))]$

We train with *alternating optimization* (aim is to keep D near optimum and G changes only slowly):

GAN Training Algorithm:

While not converged **do**

1. Freeze G, **for** k steps **do**:
   i. Draw N samples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$ from $p_{\text{data}}(\mathbf{x})$
   ii. Draw N samples $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(N)}\}$ from $p(\mathbf{z})$
   iii. Update D by ascending:
   $\nabla_{\theta_D} \frac{1}{N}\sum_{i=1}^N [\log D(\mathbf{x}^{(i)}) + \log(1 - D(G(\mathbf{z}^{(i)})))]$
2. Freeze D, draw N samples $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(N)}\}$ from $p(\mathbf{z})$
3. Update G by descending:
   $\nabla_{\theta_G} \frac{1}{N}\sum_{i=1}^N \log(1 - D(G(\mathbf{z}^{(i)})))$

(Or ascending: $\nabla_{\theta_G} \frac{1}{N}\sum_{i=1}^N \log(D(G(\mathbf{z}^{(i)})))$ avoid saturat.)

Optimal Discriminator: $D^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_{\text{model}}(\mathbf{x})}$

Jensen-Shannon Divergence: $\text{JS}(p \parallel q) = \frac{1}{2}\text{KL}\left(p \parallel \frac{p+q}{2}\right) + \frac{1}{2}\text{KL}\left(q \parallel \frac{p+q}{2}\right)$ (symmetric!)

$V(G, D^*) = -\log 4 + 2 \cdot \text{JS}(p_{\text{data}} \parallel p_{\text{model}})$

Global Opt.: $p_{\text{data}} \equiv p_{\text{model}} \Rightarrow V(G^*, D^*) = -\log 4$

Convergence Guarantee: If G and D have sufficient capacity, at each step $D \to D^*$, and p_{model} improves: $V(D^*, p_{\text{model}}) \propto \sup_D \int p_{\text{model}}(\mathbf{x}) \log(1 - D(\mathbf{x}))d\mathbf{x}$. Since $V(D^*, p_{\text{model}})$ is convex in p_{model}, global optimum is achievable. In Reality: Assumptions too strong - finite capacity networks, D doesn't reach D^*, non-convex objectives.

6.1 Training Challenges & Solutions

1. Gradient Saturation: Early in training, $D(G(\mathbf{z})) \approx 0 \Rightarrow \log(1 - D(G(\mathbf{z})))$ saturates. $\Rightarrow$ **Solution:** Use gradient ascent on $\max_G \mathbb{E}_{p_z}[\log D(G(\mathbf{z}))]$ instead

2. Mode Collapse: Generator produces limited diversity, only high-quality samples from few modes. $\Rightarrow$ **Sol:** Unrolled GAN (optimize G w.r.t. last k discr.)

3. Training Instability: Two-player games hard to optimize, finding Nash equilibria difficult $\Rightarrow$ **Sol:** Gradient penalty: $V(G,D) = \mathbb{E}_{x \sim p_d}[\log D(\mathbf{x}) - \lambda\|\nabla_x D(\mathbf{x})\|^2] + \mathbb{E}_{x \sim p_m}[\log(1 - D(G(\mathbf{z})))]$

6.2 GAN Variants & Improvements

Progressive GAN: Grow generator and discriminator resolution by adding layers during training

Feature Modulation: Control image generation by modulating intermediate feature maps. **AdaIN:** (Instance Normalization + Feature Modulation) $\mathbf{c}' = \gamma(\mathbf{s}) \odot \frac{\mathbf{c}-\mu(\mathbf{c})}{\sigma(\mathbf{c})} + \beta(\mathbf{s})$. First normalize features, then apply affine transformation. Transfer style statistics from style vector to content features.

StyleGAN: Layer-wise style conditioning. **Mapping Network:** $\mathbf{z} \in \mathcal{Z} \to \mathbf{w} \in \mathcal{W}$ (intermediate latent space). **Style Injection:** AdaIN with different $\mathbf{s}_i$ at each resolution + per-layer noise $\boldsymbol{\varepsilon}_i$. Better disentanglement than standard GAN.

6.3 Conditional GANs & Applications

Pix2Pix (Paired Translation): sketch $X \to$ img Y

$\mathcal{L}(G,D) = \mathcal{L}_{\text{cGAN}}(G,D) + \lambda \mathcal{L}_{L1}(G)$

$\mathcal{L}_{\text{cGAN}}(G,D) = \mathbb{E}_{\mathbf{x},\mathbf{y}}[\log D(\mathbf{x},\mathbf{y})] + \mathbb{E}_{\mathbf{x},\mathbf{z}}[\log(1-D(\mathbf{x},G(\mathbf{x},\mathbf{z})))]$

$\mathcal{L}_{L1}(G) = \mathbb{E}_{\mathbf{x},\mathbf{y},\mathbf{z}}[\|\mathbf{y}-G(\mathbf{x},\mathbf{z})\|_1]$ **Limitations:** Required paired images as train data, maps to unique output.

CycleGAN (Unpaired Translation): Two conditional GANs with cycle consistency

$\mathcal{L}(G,F,D_X,D_Y) = \mathcal{L}_{\text{GAN}}(G,D_Y,X,Y) + \mathcal{L}_{\text{GAN}}(F,D_X,Y,X) + \lambda \mathcal{L}_{\text{cyc}}(G,F)$

$\mathcal{L}_{\text{cyc}}(G,F) = \mathbb{E}_{\mathbf{x}}[\|F(G(\mathbf{x}))-\mathbf{x}\|_1] + \mathbb{E}_{\mathbf{y}}[\|G(F(\mathbf{y}))-\mathbf{y}\|_1]$

Vid2vid: Video-to-video translation extending pix2pix to temporal domain (with temporal discriminator and optical flow for consistency)

BicycleGAN: Addresses mode collapse in pix2pix by enforcing bidir. consistency (cVAE-GAN + cLR-GAN losses) for diverse outputs from same input.

6.4 3D GANs (Voxel-based 3D generation)

PlatonicGAN: 2D input → 3D output → differentiable 2D rendering for discriminator

HoloGAN: 3D GAN + 2D super-resolution GAN

EG3D: Tri-plane representation (uses $O(N^2)$ instead of $O(N^3)$ for voxel) to project 3D points to three 2D feature planes (from StyleGAN).

6.5 Advantages & Limitations

Pros: High expressiveness and flexibility; No explicit density modeling required; Often produces realistic samples; Efficient sampling (single forward pass)

Cons: No explicit likelihood computation; Training instability and mode collapse; Difficult to evaluate and compare models; Requires careful balancing of generator and discriminator

7 Normalizing Flows

Problem: VAEs lack tractable likelihood, autoregressive models have no latent space representation

$\Rightarrow$ **Goal:** Combine benefits of both - tractable likelihood AND meaningful latent space

Change of Variables: (probabilistic mass preserved)

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

$\Rightarrow$ change of volume given by det. of Jacobian

Consider directed, latent variable model over *observed* variables X and *latent* variables Z. In normalizing flows: mapping between Z and X given by deterministic and invertible functions:

$$f_\theta : \mathbb{R}^d \to \mathbb{R}^d, \text{s.t.}, X = f_\theta(Z), Z = f_\theta^{-1}(X)$$

$$p_{\mathbf{x}}(\mathbf{x}) = p_{\mathbf{z}}(f^{-1}(\mathbf{x}))\left|\det\left(\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}\right)\right|^{-1}$$

Requirements for f: (1) *Invertible*: f^{-1} must exist and (2) be *differentiable*, (3) *Dimension-preserving*: $\dim(\mathbf{z}) = \dim(\mathbf{x})$, i.e., bijective mapping (and (4) $\det \frac{\partial f}{\partial \mathbf{z}}$ efficiently computable in $\mathcal{O}(d)$ not $\mathcal{O}(d^3)$, Trick: Design f with triangular Jacobian $\Rightarrow$ determinant = product of diagonal elements)

Useful Identities: $\log|\det(\mathbf{A}^{-1})| = -\log|\det(\mathbf{A})|$

$\det(\mathbf{I} + \mathbf{u}h'\mathbf{w}^T) = 1 + h'\mathbf{u}^T\mathbf{w}$ $\quad \mathbf{x} \odot \mathbf{y} = \text{diag}(\mathbf{x})\mathbf{y}$

7.1 Coupling Layers

$$f : \begin{bmatrix} \mathbf{x}_A \\ \mathbf{x}_B \end{bmatrix} \mapsto \begin{bmatrix} h(\mathbf{x}_A, \beta(\mathbf{x}_B)) \\ \mathbf{x}_B \end{bmatrix}$$

Inverse: $f^{-1} : \begin{bmatrix} \mathbf{y}_A \\ \mathbf{y}_B \end{bmatrix} \mapsto \begin{bmatrix} h^{-1}(\mathbf{y}_A, \beta(\mathbf{y}_B)) \\ \mathbf{y}_B \end{bmatrix}$

Jacobian: $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} h'(\mathbf{x}_A, \beta(\mathbf{x}_B)) & h'(\mathbf{x}_A, \beta(\mathbf{x}_B))\beta'(\mathbf{x}_B) \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$

Where β can be any neural network (not invertible) and h is element-wise invertible

Flow of Transformations:

$\mathbf{x} = f(\mathbf{z}) = (f_K \circ f_{K-1} \circ \cdots \circ f_1)(\mathbf{z})$

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

Training Objective (max. *exact* log likelihood):

$$\log p_X(\mathcal{D}) = \sum_{\mathbf{x}\in\mathcal{D}} \left[\log p_Z(f^{-1}(\mathbf{x})) + \sum_{k=1}^K \log\left|\det \frac{\partial f_k^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right|\right]$$

- Sample $\mathbf{x}$: draw $\mathbf{z} \sim p_{\mathbf{z}}$, apply transform: $\mathbf{x} = f(\mathbf{z})$.
- Compute likelihood of $\mathbf{x}$: $\mathbf{z} = f^{-1}(\mathbf{x})$, eval. $p_{\mathbf{z}}(\mathbf{z})$.

NICE (additive): $\begin{bmatrix} \mathbf{y}_A \\ \mathbf{y}_B \end{bmatrix} = \begin{bmatrix} \mathbf{x}_A + \beta(\mathbf{x}_B) \\ \mathbf{x}_B \end{bmatrix}$

RealNVP: $\begin{bmatrix} \mathbf{y}_A \\ \mathbf{y}_B \end{bmatrix} = \begin{bmatrix} \mathbf{x}_A \odot \exp(\mathbf{s}(\mathbf{x}_B)) + \mathbf{t}(\mathbf{x}_B) \\ \mathbf{x}_B \end{bmatrix}$

GLOW: Flow Blocks for Computer Vision

L Levels and K steps per level (*depth*). The *squeeze* operation reduces spatial dimensions. For comp. eff., only half of the features are passed on (*split*).

Flow Block Structure: (1) ActNorm → (2) Invertible 1×1 Conv → (Conditional) Coupling Layer

1. Activation Normalization (ActNorm): Similar to batch norm. Forward: $\mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$; Inverse: $\mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$; Log-det: $H \cdot W \cdot \sum_i \log|\mathbf{s}_i|$

2. Invertible 1×1 Convolution: Generalization f a permutation in the channel dimension. Forward: $\mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$; Inverse: $\mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$; Log-det: $H \cdot W \cdot \log|\det \mathbf{W}|$ where $\mathbf{W} \in \mathbb{R}^{C\times C}, \det(W) = 1$.

Efficient 1×1 Conv: $\mathbf{W} = \mathbf{PL}(\mathbf{U} + \text{diag}(\mathbf{s}))$.

Where $\mathbf{P}$: fixed permutation; $\mathbf{L}$: lower triangular; $\mathbf{U}$: upper triangular. Log-det: $\sum_i \log|\mathbf{s}_i|$ (reduces $\mathcal{O}(C^3)$ to $\mathcal{O}(C)$)

3. Affine Coupling Layer: Split: $\mathbf{x}_a, \mathbf{x}_b = \text{split}(\mathbf{x})$; Transform: $(\log \mathbf{s}, \mathbf{t}) = \text{NN}(\mathbf{x}_b), \mathbf{s} = \exp(\log \mathbf{s})$; Forward: $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}, \mathbf{y}_b = \mathbf{x}_b$; Inverse: $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}, \mathbf{x}_b = \mathbf{y}_b$; Log-det: $\sum_i \log|\mathbf{s}_i|$

Squeeze Operation: $C \times H \times W \to 4C \times H/2 \times W/2$ (redistribute spatial dimensions to channels)

7.2 Applications

SRFlow: **(Super-Resolution)** Learn distribution of high-res variants conditioned on low-res image.

- Forward: $\mathbf{h}^{n+1} = \exp(f^n_{\theta,s}(\mathbf{u})) \cdot \mathbf{h}^n + f^n_{\theta,b}(\mathbf{u})$
- Inverse: $\mathbf{h}^n = \exp(-f^n_{\theta,s}(\mathbf{u})) \cdot (\mathbf{h}^{n+1} - f^n_{\theta,b}(\mathbf{u}))$
- Log-det: $\sum_{i,j,k} f^n_{\theta,s}(\mathbf{u})_{i,j,k}$

StyleFlow: Replace StyleGAN mapping network $z \to w$ (aux. latent space) with cond. norm. flow

C-Flow: **(Conditional Flow)** Two coupled flows for multimodal image-to-image translation. *Flow A (Conditioning):* Standard affine coupling layer for images; *Flow B (Conditioned):* Transformation parameters conditioned on Flow A. **Process:** (1) Encode original image: $\mathbf{z}_B^1 = f_\phi^{-1}(\mathbf{x}_B^1|\mathbf{x}_A^1)$, (2) Encode extra modality: $\mathbf{z}_A^2 = g_\theta^{-1}(\mathbf{x}_A^2)$, (3) Generate new image: $\mathbf{x}_B^2 = f_\phi(\mathbf{z}_B^1|\mathbf{z}_A^2)$

Applications: Multimodal image translation, style transfer, 3D point cloud reconstruction from images

7.3 Advantages & Limitations

Pros: (1) Exact likelihood computation, (2) Meaningful latent space, (3) Stable training

Cons: (1) Expensive training, (2) Limited to equal dimensions, (3) Architecture constraints for efficient Jacobians, (4) Typically low resolution

8 Diffusion Models

Advantages: High quality generations, better diversity, more stable training than GANs. **Two-step:**

- **Forward (diffusion):** Add noise to $\mathbf{x}_t$ (not learned)
- **Reverse (denoising):** Remove noise (learned p_θ)

Modeled as Markov stochastic process, where $x_0 \sim p(x_0)$ and $x_T \sim \mathcal{N}(0, I)$

8.1 Forward Diffusion Process

Noise schedule: $\{\beta_t\}_{t=1}^T$ (mon. increasing)

Single step: $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$

Closed-form sol.: $\alpha_t = 1 - \beta_t$, $\bar{\alpha}_t = \prod_{i=1}^t \alpha_i$

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon \qquad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

8.2 Reverse Process & Training

Problem: $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ intract. (req. $\int q(\mathbf{x}_t|\mathbf{x}_0)q(\mathbf{x}_0)d\mathbf{x}_0$)

Key insight: Cond. on $\mathbf{x}_0$ makes distr. tract:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}|\mu_q(\mathbf{x}_t, \mathbf{x}_0), \sigma_q^2(t)\mathbf{I})$$

(both mean and variance known!)

Solution: Learn $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \approx q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$

Parametr.: $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I})$

ELBO Objective: $\log p(\mathbf{x}_0) \geq \mathbb{E}_{q(\mathbf{x}_1|\mathbf{x}_0)}[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)] - \text{KL}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p(\mathbf{x}_T)) - \sum_{t=2}^T \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}[\text{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))]$

Noise pred.: NN to predict noise $\epsilon_\theta(\mathbf{x}_t, t)$:

$\boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}\epsilon$

$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\sqrt{\alpha_t}}\epsilon_\theta(\mathbf{x}_t, t)$

$\Rightarrow$ **Goal:** Optimize $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$ to match $\boldsymbol{\mu}_q(\mathbf{x}_t, \mathbf{x}_0)$

Loss: $\mathcal{L} = \|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\|^2$

Training:

1. $\mathbf{x}_0 \sim q(x_0), t \sim \text{Uniform}(1, \ldots, T), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2. Compute $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$
3. Update θ: $\nabla_\theta\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2$

Sampling:

1. Sample $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2. **For** $t = T, \ldots, 1$:
   $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
   $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t)\right) + \sigma_t\mathbf{z}$

Where $\sigma_t^2 = \beta_t$ in practice, and t can be continuous.

Classifier-free Guidance: Add conditional variable c to model. Mix conditional and unconditional predictions for better quality:

$$\epsilon^*(\mathbf{x}, c; t) = (1+\rho)\epsilon_\theta(\mathbf{x}, c; t) - \rho\epsilon_\theta(\mathbf{x}; t)$$

Where $\rho \uparrow$: more guidance (better quality, less divers)

ControlNet: Zero-convolution approach to add control without destroying pretrained model: $y_c = \mathcal{F}(x) + Z_2(\mathcal{F}'(x + Z_1(c)))$ (Where $\mathcal{F}$: Original network (locked), $\mathcal{F}'$: Trainable copy, Z_1, Z_2: Zero conv. layers init. to 0, so initially $y_c = \mathcal{F}(x)$)

8.3 Latent Diffusion Models

Motivation: High-res. images expensive to model $\Rightarrow$ **Idea:** perform diff./denoising in latent space

Approach: 1. Train VAE: $\mathbf{x} \xrightarrow{\text{encode}} \mathbf{z} \xrightarrow{\text{decode}} \hat{\mathbf{x}}$ 2. Perform diffusion in latent space $\mathbf{z}$ (lower dimensional) 3. Decoder reconstructs final image

Benefits: Efficiency + diffusion focuses on «semantic»generation while VAE handles pixel-level details

8.4 Key Properties

VAEs are essentially 1-step diffusion models

Advantages: Stable training, high quality, good diversity, scalable. **Disadvantages:** Slow sampling (requires T denoising steps), computationally expensive

9 Foundation Models

NLP Found. Model: 1st Gen: Encoder + Task Decoder (BERT, ELMO) $\rightarrow$ 2nd Gen: Model + Finetuning (GPT-3) $\rightarrow$ 3rd Gen: LLMs (ChatGPT)

Vision Transformer (ViT): Images as tokens, patch embeddings + positional encoding + transformer

Masked AutoEncoder (MAE): Self-supervised pretr. by masking image patches, then reconstructing

CLIP: Contrastive learning for shared image-text feature space (Image enc. + Text enc.). Max. cosine similarity for matching pairs, min. for non-matching.

Segment Anything (SAM): Heavy encoder: MAE-pretrained ViT on SA-1B dataset; Light decoder: Prompt-based segmentation; Generalizes to any object without retraining

DINOv2: Self-supervised learning for general-purpose visual features. Self-distillation: Student matches teacher (built from past student iterations). No text supervision needed (unlike CLIP).

DreamBooth: Finetune diffusion models for customized text-to-image generation with few examples

Zero-1-to-3: View synthesis from single image. Condition diffusion model on CLIP image embedding + 3D rotation (R,T). Channel-concatenate image to U-Net features. Caveat: Poor perf. on humans

SiTH: Single-view textured human reconstruction addressing Zero-1-to-3 limitations

Key Insights **Deterministic tasks:** Pretrained encoder + task-specific decoder often superior; **Generative tasks:** Model finetuning enables 2D→3D adaptation and custom.; **Current state:** Vision lacks truly gener. model equivalent to LLMs in NLP

10 Parametric Human Body Models

Goal: Equip AI with human-level perception and modeling to understand and mimic people.

10.1 2D Feature Learning Approaches

Direct Regression (DeepPose): CNN directly outputs (x,y) coord. for body parts with refinement

Heatmaps: CNN outputs separate binary heatmap for each keypoint. Keypoint positive (often Gaussian-blurred), elsewhere negative

Convolutional Pose Machine: Iteratively generate more accurate heatmaps using original image.

OpenPose (Real-Time Multi-Person 2D Keypoint Est.): *bottom-up* approach similar to Conv. Pose Machines. Introduces Part Affinity Fields (PAFs): for each limb pixel p, define unit vector v along the limb direction. Helps associating keypoints into full skeletons and assigning them to individuals.

10.2 Statistical Shape Models (Basel Face Model)

Core Idea: Represent any shape as linear combination of basis shapes: $S = \sum_{i=1}^{m} a_i S_i$. **Requirements:**

1. **Dense Correspondences:** Register template mesh to all input scans (difficult chicken-egg problem)
2. **Sensible Coefficients:** Use PCA to ensure meaningful parameter space

(1) Dense Corr. Challenge: Chicken-and-egg problem: need morphable model to help registration, but need registration to build morphable model.

Bootstrapping Sol: Start with simple algo (ICP) for easy samples $\rightarrow$ build initial model $\mathcal{M}_0$ $\rightarrow$ fit to diff. examples $\rightarrow$ update model $\mathcal{M}_1$ $\rightarrow$ repeat.

(2) PCA for Shape Space: Compute average shape $\bar{S}$, center data $x_i = S_i - \bar{S}$, compute covariance $C = \frac{1}{n} X X^T$, find eigenvectors U via SVD ($C = \frac{1}{m} U \operatorname{diag}(\sigma_i^2) U^T$). Model: $S = \bar{S} + \sum_{i=1}^{m} c_i \sigma_i \mathbf{u}_i$

10.3 SMPL: Skinned Multi-Person Linear Model

SMPL uses a 3D mesh (vertices and faces, N = 6,890).

$$M(\boldsymbol{\beta},\boldsymbol{\theta}) = W(T_P(\boldsymbol{\beta},\boldsymbol{\theta}), J(\boldsymbol{\beta}), \boldsymbol{\theta}, \mathcal{W})$$
$$T_P(\boldsymbol{\beta},\boldsymbol{\theta}) = \overline{\mathbf{T}} + B_S(\boldsymbol{\beta}) + B_P(\boldsymbol{\theta})$$

$\boldsymbol{\beta} \in \mathbb{R}^{10}$: Shape params; $\boldsymbol{\theta} \in \mathbb{R}^{3K+3}$: Pose params ($K = 23$ joints + global orientation); W: Linear Blend Skinning func; T_P: Template with shape and pose correctives; $\mathcal{W} \in \mathbb{R}^{K' \times 3N}, K' \ll K$: Skinning weights; $\overline{\mathbf{T}} \in \mathbb{R}^{3N}$: Rest pose (T-pose)

Shape Blend Shapes: $B_S(\boldsymbol{\beta}) = \sum_{n=1}^{|\beta|} \beta_n \mathbf{S}_n$, Per vertex linear offset (learned via PCA on dataset)

Joint Locations: $J(\boldsymbol{\beta}) = \mathcal{J}(\overline{\mathbf{T}} + B_S(\boldsymbol{\beta}))$ where $\mathcal{J}$ is learned regression matrix from meshes in T-pose.

Pose Representation: Angle-axis formulation: rotate by angle ϕ around norm. axis ω, so $\|\mathbf{a}\| = \phi$ where $\mathbf{a} = \phi\omega$. Rotations are relative (*Kin. chain*): $\mathbf{R}_k^{\text{global}} = \mathbf{R}_k^{\text{local}} \cdot \mathbf{R}_{A(k)}^{\text{global}}$ where $A(k)$ parent of joint k.

Linear Blend Skinning (LBS): Posed vertices t_i' are lin. comb. of transformed template vertices t_i:

$$\mathbf{t}_i' = \sum_{k=1}^{K} w_{k,i} G_k'(\boldsymbol{\theta}, J(\boldsymbol{\beta}))(\mathbf{t}_i + \mathbf{b}_{S,i}(\boldsymbol{\beta}) + \mathbf{b}_{P,i}(\boldsymbol{\theta}))$$

LBS Artifacts: Candy-wrapper effect, collapsing joints $\Rightarrow$ Pose correctives $B_P(\boldsymbol{\theta})$ learned from data.

Pose Blend Shapes: $B_P(\boldsymbol{\theta}) = \sum_{n=1}^{9K} (R_n(\boldsymbol{\theta}) - R_n(\boldsymbol{\theta}^*))\mathbf{P}_n$ where $\boldsymbol{\theta}^*$ is rest pose

Root rotation applied to origin of SMPL template (not root joint): $V' = \mathbf{R}_0(T_P(\boldsymbol{\beta},\boldsymbol{\theta}) - \mathbf{j}_0) + \mathbf{j}_0 + \mathbf{t}$

Parameters: $\boldsymbol{\beta}, \boldsymbol{\theta}$ and $\Phi = \{\overline{\mathbf{T}}, \mathcal{W}, \mathcal{S}, \mathcal{J}, \mathcal{P}\}$

Training Process: (first registration)

1. Train $\{\mathcal{W}, \mathcal{J}, \mathcal{P}\}$ on pose dataset minimizing Euclidean surface reconstruction error + regularization ($\mathcal{P} \approx 0$, $\mathcal{W}$ kept close, influence $\mathcal{J}$ only local)
2. Train $\{\overline{\mathbf{T}}, \mathcal{S}\}$ on dataset with pose norm.: $\hat{\mathbf{T}}_j^S = \arg\min_{\hat{\mathbf{T}}} \left\| W\left(\mathbf{T} + B_P(\vec{\theta}_j; \mathbf{P}), \mathcal{J}\hat{\mathbf{T}}, \vec{\theta}_j, \mathcal{W}\right) - \mathbf{V}_j^S \right\|^2$. Then PCA on $\hat{\mathbf{T}}_j^S$.

10.4 SMPL Applications

Human Mesh Recovery (HMR): Direct regression from image to SMPL parameters $(\boldsymbol{\beta}, \boldsymbol{\theta})$ using CNN + discriminator for realism.

SMPLify: Opt. based fitting: $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \|J_{\text{projected}} - J_{\text{2D}}\|^2 + L_{\text{prior}}$ (Prio learned from DB of SMPL poses).

Issues: Hand-crafted optimization, sensitive to initialization, slow convergence. $\Rightarrow$ *Learned Gradient Descent* (**LGD**): $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t + F(\frac{\partial L_{\text{reproj}}}{\partial \boldsymbol{\theta}}, \boldsymbol{\theta}^t, x)$ where F is neural network predicting parameter updates.

IMU Fitting: Optimize SMPL pose to match orientation and acceleration from inertial sensors.

Electromag. Fit.: Use EM measur. for pose/ shape est. with learned iter. fitting using RNN + LGD.

Clothing Modeling: SMPL provides naked body $\rightarrow$ template reconstruction for clothed appearance.

11 Egocentric Computer Vision

Def: First-person vision from wearable cameras (head-mounted, chest-mounted). **vs. Third-Person:** Dynamic content, embodied (pros), Motion blur, occlusions (cons). **Key Tasks:** Action recognition/anticipation, hand-object interaction, gaze prediction, 3D scene understanding, localization

12 Implicit Surfaces & Neural Radiance

- **Voxels:** $\mathcal{O}(n^3)$ memory cost, discret. artifacts
- **Point clouds:** Doesn't model connect./topology
- **Meshes:** Approx. err., limit. granular., self-intersec.

Solution: Implicit shape repr.: Represent surface as level-set of a continous function: $S = \{\mathbf{x} : f(\mathbf{x}) = 0\}$

Advantages: Non-rigid transf., continuous normals, infinite precision, learnable, continous, arb. topology, memory efficient (simply the weights of network f_θ)

Disadvantages: Expensive intersection computation, requires root-finding, ill-defined UV space

12.1 Neural Implicit Representations (NIR)

Both cond. on input $\mathcal{X}$ (2D image, class labels, etc.)

Occupancy Networks: $f_\theta : \mathbb{R}^3 \times \mathcal{X} \rightarrow [0,1]$ (probability of being inside mesh). f_θ a MLP.

DeepSDF: $f_\theta : \mathbb{R}^3 \times \mathcal{X} \rightarrow \mathbb{R}$ (signed dist. to surface)

Normals: Gradient of implicit function.

12.2 Training NIR

1. From Watertight Meshes: Sample points inside/outside mesh. Occupancy: Binary cross-entropy loss. DeepSDF: Regression loss on distance to mesh.

2. From Point Clouds: Use Eikonal regularization:

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} |f_\theta(\mathbf{x}_i)|^2 + \lambda \mathbb{E}_{\mathbf{x}}[(\|\nabla_{\mathbf{x}} f_\theta(\mathbf{x})\| - 1)^2]$$

Eikonal term rationale: $\|\nabla f\| = 1$ (single answer to "how far away am I from the surface?").

3. From Images: Need Differentiable Volumetric Rendering (**DVR**): Networks: $f_\theta(\mathbf{p}) \in [0,1]$ (occupancy), $\mathbf{t}_\theta(\mathbf{p}) \in \mathbb{R}^3$ (color). **Forward:** For each pixel $\mathbf{u}$, cast ray $\hat{\mathbf{p}} = \mathbf{r}_0 + \hat{d}\mathbf{w}$, then **Secant method:** Find smallest j s.t. $f_\theta(x_{j+1}) \geq \tau > f_\theta(x_j)$, then: $x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)}$ (iter. find surface intersec.). **Loss:** $\mathcal{L}(\hat{\mathbf{I}}, \mathbf{I}) = \sum_{\mathbf{u}} \|\hat{\mathbf{I}}_{\mathbf{u}} - \mathbf{I}_{\mathbf{u}}\|$.

Backward: From $f_\theta(\hat{\mathbf{p}}) = \tau$, implicit differentiation gives: $\frac{\partial \hat{\mathbf{p}}}{\partial \theta} = -\mathbf{w}\left(\frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \cdot \mathbf{w}\right)^{-1} \frac{\partial f_\theta(\hat{\mathbf{p}})}{\partial \theta}$

12.3 Neural Radiance Fields (NeRF)

Implicit surfaces struggle with transp., thin struct.

$\Rightarrow$ **Network:** $F_\theta(x,y,z,\theta,\phi) \rightarrow (r,g,b,\sigma)$, where (θ,ϕ): viewing direction, σ: density.

Constraint: View direction applied late, limiting view effects and encouraging fine geometry. Density σ independent of viewing direction.

Forwad: Shoot ray, sample points along it & blend (Alpha Compositing): $\alpha_i = 1 - \exp(-\sigma_i \delta_i)$, $\delta_i = t_{i+1} - t_i$, $T_i = \prod_{j=1}^{i-1}(1 - \alpha_j)$ (prob. light reaches point i)

$\mathbf{c} = \sum_{i=1}^{N} T_i \alpha_i \mathbf{c}_i$ (Hierarchical Samp. for better effic.).

Training: $\min_\theta \sum_i \|\text{render}_i(F_\theta) - \mathbf{I}_i\|^2$

Key difference from DVR: Sample multiple points along ray, not just surface intersection

Positional Encoding: *Problem:* NNs biased toward low-frequency functions, but need high-frequency details. *Solution:* Replace (x,y,z) coordinates with positional encoding or rand. Fourier features.

Caveat: Slow render. due to many ray samples (50+)

12.4 3D Gaussian Splatting

Idea: Model scene as collection of 3D Gaussians.

Process: 1. Initialize with point cloud 2. Project 3D Gaussians to 2D image plane («splatting ») 3. Differentiable rendering with depth sorting 4. Adaptive density control: move/clone/merge Gaussians

(1) 2D Gauss. Proj.: $\alpha_i(\mathbf{u}) = o_i \cdot \exp\left(-\frac{1}{2}(\mathbf{x} - \mu_i')^T (\boldsymbol{\Sigma}_i')^{-1} (\mathbf{x} - \mu_i')\right)$

3D Gaussian parameterization: $\boldsymbol{\Sigma} = \mathbf{R}\mathbf{S}\mathbf{S}^T\mathbf{R}^T$. $\mathbf{R}$: rotation, $\mathbf{S}$: scaling, o_i: opacity

Rendering: Same as NeRF volume rendering

Advantages: Fast rendering, real-time capable, good quality, no neural networks required