

# 1 Connectionism

## 1.1 Perceptron

**Threshold Unit**  $f[w, b](x) = \text{sign}(x \cdot w + b)$  with **Decision Boundary**  $x \cdot w + b = 0$ .  $-\frac{b}{\|w\|}$  is the signed distance of the hyperplane from 0.  
**Geometric Margin**  $\gamma[w, b](x, y) = \frac{y(x \cdot w + b)}{\|w\|}$ .  
**Maximum Margin Classifier**  $(w^*, b^*) \in \text{argmax}_{w, b} \gamma[w, b](\mathcal{S})$ , with margin  $\gamma[w, b](\mathcal{S}) := \min_{(x, y) \in \mathcal{S}} \gamma[w, b](x, y)$ .

### Perception Learning

if  $f[w, b](x) \neq y: w \leftarrow w + yx, b \leftarrow b + y$ .  
Aims to find some solution; assumes version space is non-empty. Does not aim for small err.  
 $w_0 \in \text{span}(x_1, \dots, x_s) \Rightarrow w_t \in \text{span}(x_1, \dots, x_s) \forall t$

### Convergence - Novikoff's Theorem

- If  $\exists w^*, \|w^*\| = 1$ , s.t.  $\gamma[w^*](\mathcal{S}) = \gamma > 0 \Rightarrow w_t \cdot w^* \geq t\gamma$ .
- Let  $R = \max_{x \in \mathcal{S}} \|x\|$ . Then  $\|w_t\| \leq R\sqrt{t}$ .  
 $\cos \angle(w^*, w_t) = \frac{w^* \cdot w_t}{\|w^*\| \|w_t\|} \geq \frac{t\gamma}{\sqrt{t}R} = \sqrt{t} \frac{\gamma}{R} \leq 1 \Rightarrow t \leq \frac{R^2}{\gamma^2}$ .

### Cover's Theorem for $\mathcal{S} \subset \mathbb{R}^n, |\mathcal{S}| = s$

$C(\mathcal{S}, n)$ : # of ways to separate  $\mathcal{S}$  in  $n$  dimensions.  
Position of pts does not matter (as long as they are in general position).  $C(s+1, n) = 2 \sum_{i=0}^{n-1} \binom{s}{i}$ .  
 $C(s, n) = 2^s$  for  $s \leq n$ . Phase transition at  $s = 2n$ . For  $s < 2n$  empty version space is the exception, otherwise the rule.

## 1.2 Hopfield Networks

$E(X) = -\frac{1}{2} \sum_{i \neq j} w_{ij} X_i X_j + \sum_i b_i X_i$ , where  $X_i \in \{\pm 1\}$ .  $w_{ij} = w_{ji}, w_{ii} = 0$ .

### Hebbian Learning

Choose patterns  $\{x^t\}_{t=1}^s \in \{\pm 1\}^n$ , compute weights once using them:  $w_{ij} = \frac{1}{n} \sum_{t=1}^s x_i^t x_j^t$ ,  $w_{ii} = 0$ . For inference, update  $X$  iteratively:  $X_i^{t+1} = \text{sign}(\sum_j w_{ij} X_j^t + b_i)$  asynchronously.  
Capacity for random, uncorrelated patterns:  $s_{\max} \approx 0.138n$ . Requiring pattern to be retrieved with high probability:  $s \leq \frac{n}{2 \log_2 n}$ .

If  $X = \text{diag}(1, \dots, 1)$ , no reconstruction happens. Under async update step, any Hopfield network is guaranteed to converge.

# 2 Feedforward Networks

## 2.1 Linear Models

**Linear regression** (MSE)  
 $L[w](X, y) = \frac{\|Xw - y\|^2}{2n}$ ,  $\nabla_w L = \frac{X^T X w - X^T y}{n}$ .  
**Moore-Penrose inverse solution**  
 $w^* = X^+ y \in \text{argmin}_w L[w](X, y)$ , where  $X^+ = \lim_{\delta \rightarrow 0} (X^T X + \delta I)^{-1} X^T$  Moore-Penrose inv.

**Stochastic gradient descent update**  $w_{t+1} = w_t + \eta(y_{i_t} - w_t^T x_{i_t}) x_{i_t}, i_t \sim \mathcal{U}([1, n])$ .

**Gaussian noise model**  $y_i = w^T x_i + \varepsilon_i, \varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ , LSQ equivalent to NLL of gaussian noise model.

**Ridge regression**  $h_\lambda[w] = h[w] + \frac{\lambda}{2} \|w\|^2, w^* = (X^T X + \lambda I)^{-1} X^T y$ .

### Logistic function

$\sigma(z) = \frac{1}{1+e^{-z}}, \sigma'(z) = \sigma(-z) = 1$ .  
 $\sigma' = \sigma(1 - \sigma), \sigma'' = \sigma(1 - \sigma)(1 - 2\sigma)$   
**Cross entropy loss** for  $y \in \{0, 1\}$   
 $\ell(y, z) = -y \log \sigma(z) - (1 - y) \log(1 - \sigma(z)) = -\log \sigma((2y - 1)z)$ .

**Logistic regression with Cross Entropy loss:**  
 $L[w] = \frac{1}{n} \sum_{i=1}^n \ell_i(y_i, w^T x_i)$ ,  
 $\nabla \ell_i = [\sigma(w^T x_i) - y_i] x_i$ .

## 2.2 Feedforward Networks

### Generic feedforward layers

$F: \frac{\mathbb{R}^{m(n+1)}}{\text{parameters}} \times \frac{\mathbb{R}^n}{\text{input}} \rightarrow \frac{\mathbb{R}^m}{\text{output}}, F[\theta](x) = \varphi(Wx + b)$ .

**Softmax**( $z$ ) $_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$ , CE-loss in terms of log-its  $\ell(y, z) = \frac{1}{\ln(2)} \left[ -z_y + \ln \sum_j e^{z_j} \right]$ . CE between two pmfs:  $\ell(p; q) = -\sum_i p_i \log q_i$ .  
CE with hard labels is NLL-loss.

$\frac{d}{dx_j} \text{softmax}(x)_i = \text{sm}(x)_i (\delta_{ij} - \text{sm}(x)_j)$

**Residual layer**  $F[W, b](x) = x + (\varphi(Wx + b) - \varphi(0))$ , therefore  $F[0, 0] = \text{id}$ . Link that propagates  $x$  forward is called a **skip connection**. Composing residual layers: number of paths grows exponentially, can include projections for flexibility of changing dimensionality.

## 2.3 Sigmoid Networks

**Sigmoid activation**  $\sigma(z) = \frac{1}{1+e^{-z}}$ .  
**Hyperbolic tangent activation**  
 $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = 2\sigma(2z) - 1$ .  
 $\tanh'(z) = 1 - \tanh^2(z)$ .

**Smooth function approximation**  
Polynomials, ridge functions (functions of the form  $\varphi(a^T x + b)$ ) and MLPs with  $C^\infty$  activations are universal approximators.

**Weierstrass:** Polynomials are universal approximators of  $C(\mathbb{R})$  on any given compact  $I$ .  
**Barron's Theorem:** For  $f: \mathbb{R}^d \rightarrow \mathbb{R}$  with  $C_f = \int \|w\| |f'(w)| dw < \infty, \exists$  width- $m$  MLP  $g_m$  s.t.:  $\int_B |f - g_m|^2 dx \leq O(\frac{1}{m})$ .

## 2.4 ReLU( $z$ ) = max(0, $z$ ) networks

**Zalavsky's Theorem: Activation patterns**  
 $m$  ReLU neurons in  $\mathbb{R}^n$ . Each neuron's hyperplane  $\{w_i^T x = 0\}$  partitions  $\mathbb{R}^n$  into  $R(m)$  connected regions of constant activation pattern.  
 $R(m) \leq \sum_{i=0}^{\min(n, m)} \binom{m}{i} \ll 2^m$ .  
**Montufar: Connected linear regions in ReLU network**  $R(m, L) \geq R(m) \left[ \frac{m}{n} \right]^{n(L-1)}$ ,  $L$ : layers,  $m$ : width.

**Shektman:** Piecewise linear functions are dense in  $C([0, 1])$ .  
**Lebesgue:** Piecewise linear function with  $m$  pieces can be written  $g(x) = ax + b + \sum_{i=1}^{m-1} c_i(x - x_i)_+$ ;  $m+1$  parameters,  $a, b, c_i$ .  
**ReLU networks with 1 hidden layer are universal approximators.**

**Wang and Sun:** Every continuous piecewise linear function  $g: \mathbb{R}^n \rightarrow \mathbb{R}$  can be written as a signed sum of  $k$ -Hinges with  $k \leq n+1$ . A  $k$ -Hinge is a function  $g(x) = \max_{j=1}^k \{w_j^T x + b_j\}$ , generalizes ReLU, known as Maxout unit.

**Linear Autoencoder:** Optimal  $A = DE$ , s.t. frobenius norm reconstruction err of  $AX$  is minimized, is  $D = U_k, E = U_k^T$ , not jointly convex in  $E$  and  $D$ , but individually.  $\hat{X}^* = \arg \min_{\text{rank}(\hat{X})=k} \|X - \hat{X}\|_F^2 = U \Sigma_k V^T$  (SVD).

# 3 Gradient-Based Learning

Forward mode is more memory efficient, but backward mode is more runtime efficient. Fwd is  $O(\#\text{params})$ , reverse is  $O(d_{\text{out}})$ .

**Numerator layout:** For  $f: \mathbb{R}^n \rightarrow \mathbb{R}^m, \left(\frac{\partial y_i}{\partial x}\right)_{ij} = \frac{\partial y_i}{\partial x_j} \in \mathbb{R}^{m \times n}$  and  $f: \mathbb{R}^{n_1 \times n_2} \rightarrow \mathbb{R}, \nabla f(X)_{ij} = \frac{\partial f}{\partial X_{ij}} \in \mathbb{R}^{n_1 \times n_2}$ .

## 3.1 Backpropagation

$x^\ell = \varphi(W^\ell x^{\ell-1} + b^\ell), \frac{\partial \mathcal{L}}{\partial W^\ell} = \delta^\ell (x^{\ell-1})^T, \frac{\partial \mathcal{L}}{\partial b^\ell} = \delta^\ell, \delta^\ell = \frac{\partial \mathcal{L}}{\partial x^\ell} \odot \varphi'(W^\ell x^{\ell-1} + b^\ell), \frac{\partial \mathcal{L}}{\partial x^\ell} = (W^{\ell+1})^T \delta^{\ell+1}$ .

$\frac{\partial L}{\partial \bar{y}} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial \bar{y}}$ . For  $x \in \mathbb{R}^n$  different  $z: \frac{\partial(Wx)}{\partial x} = W$ , element wise  $f$  gives:  $\frac{\partial f(x)}{\partial x} = \text{diag}(f'(x))$ ,  $\frac{\partial \|y - \bar{y}\|^2}{\partial \bar{y}} = 2(\bar{y} - y)^T, \frac{\partial L}{\partial \bar{y}} \frac{\partial(W\bar{h})}{\partial W} = h \cdot \frac{\partial L}{\partial \bar{y}}$ .

## 3.2 Gradient Descent

**Update:**  $x_{t+1} = x_t - \eta \nabla f(x_t)$ .

**Gradient flow ODE**  $\frac{dx}{dt} = -\nabla f(x)$  gives ideal trajectory to be approximated by GD.

**Newton's method** gives optimal step for quadratic model:  $\Delta x = -[\nabla^2 f(x)]^{-1} \nabla f(x)$ .  
 $\nabla_x^2 [x^T A x + b^T x + c] = A + A^T$ .

### Optimal LR for Convex Quadratics

For  $f(x) = \frac{1}{2} x^T Q x, \eta^* = \frac{2}{\lambda_{\max}(Q) + \lambda_{\min}(Q)}$ . Stability requires  $\eta \leq \frac{2}{\lambda_{\max}(Q)}$ . Quadratic approx. of  $f: f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x + \frac{1}{2} \Delta x^T \nabla^2 f(x) \Delta x$ . Condition number of  $Q: \frac{\lambda_{\max}}{\lambda_{\min}}$

**L-smooth:**  $\|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|$ . Equivalently (if  $f$  twice diff) (for  $L = 0 \Rightarrow$ ):  $f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{L}{2} \|y - x\|^2$ . Implies  $\lambda_i(\nabla^2 f(x)) \leq L$  for all EVs  $\lambda_i$  of  $\nabla^2 f(x)$ .

**Convexity** ( $\lambda \in [0, 1]$ ):  
 $f(\lambda w + (1 - \lambda)w') \leq \lambda f(w) + (1 - \lambda)f(w')$ .  
 **$\mu$ -Strong convexity:** ( $\mu = 0 \Leftrightarrow$  convex + diff)  
 $\Leftrightarrow f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{\mu}{2} \|y - x\|^2$ . Implies  $\lambda_i(\nabla^2 f(x)) \geq \mu$  for all EVs  $\lambda_i$  of  $\nabla^2 f(x)$ .  
For  $f: \mathbb{R} \rightarrow \mathbb{R}$ , these become:  $L \geq f''(x) \geq \mu \quad \forall x$ .

**Strict convexity:** Replace  $\leq$  in convexity with  $<$ , for twice differentiable  $f: \nabla^2 f(x) \succ 0 \quad \forall x$ ; strict convexity is **weaker** than strong convexity ( $\leftarrow$ )!

If  $f \mu$ -strongly convex,  $L$ -smooth, GD iterates  $x_t$  with  $0 < \eta \leq \frac{1}{L}$  converge to unique minimizer  $x^*$  at rate  $\|x_t - x^*\|^2 \leq (1 - \eta\mu)^k \|x_0 - x^*\|^2$ .  
If  $f$  convex, diff and  $L$ -smooth, with  $\eta \leq \frac{1}{L}, f(x_t) - f(x^*) \leq \frac{1}{2\eta t} \|x_0 - x^*\|^2$ .  
**Non-Convex case:** If  $f$  diff,  $L$ -smooth, with minimum  $f^*$ , GD iterates with  $\eta \leq \frac{1}{L}$  satisfy  $\min_{i=0}^t \|\nabla f(x_i)\|^2 \leq \frac{2(f(x_0) - f^*)}{\eta(t+1)}$ .

If  $f$  diff and  $L$ -smooth:  $f(x) - f(x^*) \geq \frac{1}{2L} \|\nabla f(x)\|^2$ .

### Polyak-Lojasiewicz condition

$\frac{1}{2} \|\nabla f(x)\|^2 \geq \mu(f(x) - \min f) (\forall x)$ .

$\mu$ -strong convex  $\Rightarrow \mu$ -PL.

### GD Convergence Rates & Learning Rates

**L-smooth only:**  $\eta^* = \frac{1}{L}$ . To reach  $\varepsilon$ -stationary point ( $\|\nabla f\| \leq \varepsilon$ ) needs at most  $\frac{2\varepsilon}{\mu^2} (f(x_0) - \min f)$  steps.

**$\mu$ -PL +  $L$ -smooth:** Use  $\eta^* = \frac{2}{L+\mu}$ . Convergence:  $f(x_t) - f(x^*) \leq (1 - \frac{\mu}{L})^t (f(x_0) - f(x^*))$ .

## 3.3 Stochastic Gradient Descent

### SGD variance

$V[\theta](S) = \frac{1}{s} \sum_{i=1}^s \|\nabla f[\theta](S) - \nabla f[\theta](x_i, y_i)\|^2$ .

Polyak averages:  $\bar{x}_{k+1} = \frac{k}{k+1} \bar{x}_k + \frac{1}{k+1} x_{k+1}$

**SGD convergence rate** with Polyak averaging and  $\eta_k \propto \frac{1}{k}$

$\mathbb{E}[f(\bar{\theta}_t)] - \min f \leq O\left(\frac{1}{\sqrt{t}}\right)$  (general)  
 $\mathbb{E}[f(\bar{\theta}_t)] - \min f \leq O\left(\frac{\log t}{t}\right)$  (strongly convex)  
 $\mathbb{E}[f(\bar{\theta}_t)] - \min f \leq O\left(\frac{1}{t}\right)$  (additionally smooth)

**Minibatch SGD:** Variance  $\downarrow$  by  $\propto r$ . Can  $\uparrow \eta \propto r$ .

**Var. Reduction with SVRG** w/ occasional snapshot  $\bar{\theta}: \theta_{t+1} = \theta_t - \eta [\nabla f_i(\theta_t) - \nabla f_i(\bar{\theta}) + \nabla f(\bar{\theta})]$ .

## 3.4 Acceleration and Adaptivity

### Heavy ball momentum update

$\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t) + \beta(\theta_t - \theta_{t-1})$

### Nesterov acceleration

$\bar{\theta}_{t+1} = \theta_t + \beta(\theta_t - \theta_{t-1})$   
 $\theta_{t+1} = \bar{\theta}_{t+1} - \eta \nabla h(\bar{\theta}_{t+1})$   
More theoretical grounding than heavy ball.

### AdaGrad updates

$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\gamma_t + \varepsilon}} \odot \nabla f(\theta_t), \gamma_t = \gamma^{t-1} + \nabla f(\theta_t) \odot \nabla f(\theta_t)$ .

### Adam updates

$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla f(\theta_t), \hat{m}_t = \frac{m_t}{1 - \beta_1^t}$   
 $v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla f(\theta_t))^2, \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$   
 $\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$ .

**RMSprop:** Adam without momentum term.

**signSGD:**  $\theta_{t+1} = \theta_t - \eta \text{sign}(\nabla f_{i_k}(\theta_t))$ .

$x^4$  is strictly convex but not strongly convex, since near 0 the growth of  $x^4$  is slower than  $x^2$ , violating the uniform lower bound on curvature.

With  $f$   $L$ -smooth and  $\mu$ -PL, GD with optimal step-size  $\arg \min_\eta f(\theta_t - \eta \nabla f(\theta_t))$  converges globally at linear rate.  
 $f(w) = \frac{\|Xw - y\|^2}{2} + \lambda \|w\|^2$  satisfies PL-condition.  
**Muon:** Orthogonalize gradient, should increase the scale of other rare directions which have small magnitude in update but are important.  
 $\Delta W = -\|\nabla L(W)\|_* \cdot \frac{d_{\text{out}}}{d_{\text{in}}} \cdot U^T V \quad (\nabla L(W) = U \Sigma V^T)$  minimizes RHS of:  $L(W + \Delta W) \leq L(W) + \langle \nabla_W L(W), \Delta W \rangle_F + \frac{1}{2} \frac{d_{\text{out}}}{d_{\text{in}}} \|\Delta W\|_F^2$ .  
GD Trajectory always orthogonal to level set.

# 4 Convolutional Networks

Convolution  $(f * g)(u) = (g * f)(u) = \int_{-\infty}^{\infty} f(t)g(u - t) dt = \int_{-\infty}^{\infty} f(u - t)g(t) dt$ .  
 $(f * g) = \text{Toeplitz-Matrix}(g)f$ .

### Fourier transform convolution property

$\mathcal{F}(f * g) = \mathcal{F}(f) \cdot \mathcal{F}(g)$ .

### Cross-correlation:

$(g * f)[u] = \sum_t g[t]f[u + t]$ .

$(f(t) * g(t))(y) = (f(-t) * g(t))(y)$ .  
 $(f(t) * g(t))(-y) = (f(-t) * g(t))(y) = (f(t) * g(-t))(y)$ .  
Translation-Equivariant Operators = Convolutions.

For  $x \in \mathbb{R}^D, h \in \mathbb{R}^K, x * h = W_h x$ . **Toeplitz Matrix:**  $W_h \in \mathbb{R}^{(D-K+1) \times D}$   
 $(W_h)_{k,j} = \begin{cases} h_{K+k-j} & \text{if } k \leq j \leq k+K-1 \\ 0 & \text{otherwise} \end{cases}$  for  $h \in \mathbb{R}^K$ .

$\nabla_w v^T \text{vec}(\sigma(x * w)) = \text{flip}[x] * [\text{mat}(v) \odot \sigma'(x * w)]$ , flip rows and columns.

Normal convolution of image  $x \in \mathbb{R}^{d \times d}$ , kernel  $w \in \mathbb{R}^{q \times q} x * w$  requires  $\mathcal{O}((d - q)^2 q^2)$ . If  $w$  separable st  $w = uv^T, u \in \mathbb{R}^q, v \in \mathbb{R}^p, \mathcal{O}(dq(d - q))$

Output size:  $H_{\text{out}} = \lfloor \frac{H_{\text{in}} + 2P - K}{2} \rfloor + 1$ , Height, Input size, padding, Kernel size, stride.

For 2D convolution  $y = x * w, \frac{\partial L}{\partial w} = x * \frac{\partial L}{\partial y}$

## 4.1 Convolutional Networks

**Conventions** for Padding: Add 0 around input.

**ConvNets for Images** ( $r$  out channel,  $u$  in ch)

$y[r][s, t] = \sum_u \sum_{i,j} w[r, u][i, j] * x[u][s + i, t + j]$ .

### # parameters of a convolutional layer

$(|r| \times |u|) \cdot (|i| \times |j|)$  : fully connected  $\times$  patch-size.

## 4.2 Word2Vec

Per word  $w$ , have input embedding  $x_w$  and output embedding  $y_w$ . Predict context word  $v$  given center word  $w: P(v | w) = \frac{\exp(x_w^T y_v)}{\sum_{\mu} \exp(x_w^T y_\mu)}$ .  
NLL loss:  $\ell_{w,v} = -x_w^T y_v + \ln \sum_{\mu} \exp(x_w^T y_\mu)$ .  
Total:  $h(\{x_w\}, \{y_v\}) = \sum_{(w,v)} \ell_{w,v}$  over observed pairs. Use only input embeddings after training.

5 Geometric Deep Learning

Group is set  $G$  with a binary operation s.t.:  
1)  $(gh)f = g(hf)$ , 2)  $\exists e \in G$  s.t.  $ef = fe = f$ ,  
3)  $\forall g \exists g^{-1} \in G$  s.t.  $gg^{-1} = g^{-1}g = e$ , 4)  $gh \in G \forall g, h$ .     **Abelian** if  $gh = hg$ .

5.1 Sets and Points

**Order-invariance property:**  
 $f(x_1, ..., x_M) = f(x_{\pi(1)}, ..., x_{\pi(M)})$  (perturbs).

**(Permutation) Equivariance property:**  
 $f(x_1, ..., x_M) = (y_1, ..., y_M) \Rightarrow$   
 $f(x_{\pi(1)}, ..., x_{\pi(M)}) = (y_{\pi(1)}, ..., y_{\pi(M)})$

**Attn wo/ mask is permutation-equivariant**

**Deep Sets model** (invariant layer):  
 $f(x_1, ..., x_M) = \rho(\sum_{m=1}^M \varphi(x_m))$ .

**Equivariant map construction:**  
 $\rho: \mathbb{R} \times \mathbb{R}^N \rightarrow Y, (x_m, \sum_{k=1}^M \varphi(x_k)) \mapsto y_m$

5.2 Graph Convolutional Networks

**Feature and adjacency matrices**

$X = \text{mat}(x_1^\top; ...; x_M^\top), A = (a_{nm})$   
with  $a_{nm} = 1 \Leftrightarrow \{v_n, v_m\} \in E$ .

**Permutation matrix constraints**  
 $P \in \{0, 1\}^{M \times M}$  w/ single 1 in each row and col.

**Graph invariance definition**  
 $f(X, A) \stackrel{!}{=} f(PX, PAP^\top), \forall P \in \Pi_M$ .

**Graph equivariance definition**  
 $f(X, A) = Pf(PX, PAP^\top), \forall P \in \Pi_M$ .

**Node neighborhood features**  
 $X_m = \{\{x_n : \{v_n, v_m\} \in E\}\}, \{\{\}\} = \text{multiset}$

**Message passing scheme**  
 $\varphi(x_m, X_m) = \varphi(x_m, \bigoplus_{x \in X_m} \psi(x))$ ,  
 $\oplus$  is some permutation-invariant operation.

Normalized adjacency matrix

$\bar{A} = D^{-\frac{1}{2}}(A + I)D^{-\frac{1}{2}}, \quad D = \text{diag}(d_1, ..., d_M),$   
 $d_m = 1 + \sum_{n=1}^M a_{nm} \cdot \lambda_{\max}(\bar{A}) = 1, \lambda_{\min}(\bar{A}) \geq -1.$   
For  $\bar{L} = I - \bar{A}, \lambda_{\min}(\bar{L}) = 0$  and  $\lambda_{\max}(\bar{L}) \leq 2$ .

If  $\bar{A}$  symmetric,  $\bar{A} = \sum_i \lambda_i v_i v_i^\top$ .

For  $\bar{A}, v_1 \propto [-\sqrt{d_i} - 1]^\top$ , where d is degree+1.

**One GCN layer**  
 $X^+ = \sigma(\bar{A}XW), W \in \mathbb{R}^{M \times N}$ .

5.2.1 Spectral Graph Theory

**Laplacian operator**  
 $\Delta f = \sum_{n=1}^N \frac{\partial^2 f}{\partial x_n^2}, f: \mathbb{R}^N \rightarrow \mathbb{R}.$

**Graph Laplacian**  
 $L = D - A, (Lx)_n = \sum_{m=1}^M a_{nm}(x_n - x_m).$   
 $x^\top Lx = \frac{1}{2} \sum_u \sum_v A_{uv}(x_u - x_v)^2 \geq 0$  (psd).

**Normalized Laplacian**  
 $\bar{L} = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}}.$

Graph Fourier transform

$L = D - A = U\Lambda U^\top, \hat{f} = U^\top f, f = U\hat{f}.$   
 $\Lambda := \text{diag}(\lambda_1, ..., \lambda_M), \lambda_i \geq \lambda_{i+1}.$

**Convolution:**  $x * y = U((U^\top x) \odot (U^\top y)).$

**Filtering operation**  $G_\theta(L)x = UG_\theta(\Lambda)U^\top x$   
**Polynomial kernels**  
 $U(\sum_{k=0}^K \alpha_k \Lambda^k)U^\top = \sum_{k=0}^K \alpha_k L^k$   
**Polynomial kernel network layer**  
 $x_i^{l+1} = \sum_j p_{ij}(L)x_j^l + b_i; p_{ij}(L) = \sum_{k=0}^K \alpha_{ijk} L^k$

GNNs cannot distinguish between certain graphs that are topologically different. Unconstrained set architectures are more powerful. If **WL-test** says graphs are different, they are; but if it says they're the same, they might still be different.

6 Theory of DNNs

6.1 Statistical Learning Theory

**Risk decomposition**  
 $f^*$ : optimal predictor over all functions,  
 $f_H^* = \text{argmin}_{f \in H} \mathcal{R}(f), \hat{f}_H$ : learned from finite data.  
 $\underbrace{\mathcal{R}(\hat{f}_H) - \mathcal{R}(f^*)}_{\text{excess risk}} = \underbrace{\mathcal{R}(\hat{f}_H) - \mathcal{R}(\hat{f}_H)}_{\text{estimation error}} + \underbrace{\mathcal{R}(\hat{f}_H) - \mathcal{R}(f^*)}_{\text{approximation error}}.$

**Rademacher Complexity**  $G = \{g_h \mid h \in H\}$ :  
For  $\sigma \in \{-1, 1\}$ , measures how well  $G$  can fit random noise:  $\hat{\mathfrak{R}}_{D_n}(G) = \mathbb{E}_\sigma [\sup_{g \in G} g \frac{1}{n} \sum_{i=1}^n \sigma_i g(z_i)].$   
 $\mathbb{E} [\sup_{h \in H} \mathcal{R}(h) - \mathcal{R}_{D_n}(h)] \leq 2\hat{\mathfrak{R}}_{D_n}(G),$   
 $\mathbb{E} [\mathcal{R}(\hat{h}_H)] \leq \mathcal{R}(h_H^*) + 2\hat{\mathfrak{R}}_{D_n}(G).$

**Double descent:**  
Beyond the interpolation point, models eventually may level out at a lower generalization error.

**Implicit bias towards min norm solutions:**  
Any convergent algorithm with iterates in  $\text{span}\{x_1, ..., x_n\}$  finds minimum norm solution.

6.1.1 A PAC-Bayesian result

$P$  prior distribution over functions before seeing data,  $Q$  posterior after training.

PAC-Bayesian theorem

Bounds generalization gap for stochastic classifiers ( $f \sim Q$ ):  $E_Q[\mathcal{R}(f)] - E_Q[\hat{\mathcal{R}}_n(f)] \leq \sqrt{\frac{2}{n} [\text{KL}(Q\|P) + \ln(2\sqrt{n}/\varepsilon)]}$   
•  $P$ : prior,  $Q$ : posterior (learned). Rate  $\tilde{O}(1/\sqrt{n})$   
•  $\text{KL}(Q\|P)$ : "information cost" of moving  $P \rightarrow Q$   
• Insight: generalization depends on **distance moved**, not parameter count

PAC-Bayesian for DNNs

$P = \mathcal{N}(0, \lambda I), Q = \mathcal{N}(\theta, \text{diag}(\sigma_i^2))$   
 $\text{KL}(Q\|P) = \sum_i [\log \frac{\lambda}{\sigma_i^2} + \frac{\sigma_i^2 + \theta_i^2}{2\lambda^2} - \frac{1}{2}]$   
Minimize directly:  $E_Q[\hat{\mathcal{R}}] + \sqrt{\frac{2}{n} [\text{KL}(Q\|P) + ...]}$   
 $\Rightarrow$  encourages wide/flat minima (perturbations  $\theta + \varepsilon$  must also perform well)

**Implementation:** Reparameterization:  $\tilde{\theta} = \theta + \text{diag}(\sigma_i), \eta \sim \mathcal{N}(0, I)$ , Backprop through  $\theta$  and  $\sigma$ .

6.2 Linearized DNNs and NTK

Training neural network  $f(\theta)(x)$  can be approximated by **linearizing** around initialization  $\theta_0$  when parameters change slowly.

**Linearization  $\rightarrow$  Kernel Regression:**  
Taylor approximation:  $h(\beta)(x) = f(\theta_0)(x) + \beta \cdot \nabla f(\theta_0)(x), \quad \beta = \theta - \theta_0.$   
With residuals  $\tilde{y}_i = y_i - f(\theta_0)(x_i)$ , training becomes **linear regression** with features  $\nabla f(\theta_0)(x_i): \min_\beta \sum_i \|\tilde{y}_i - \beta^\top \nabla f(\theta_0)(x_i)\|^2$

**Neural Tangent Kernel (NTK):**  
Definition:  $k_\theta(x, x') := \nabla f(\theta)(x) \cdot \nabla f(\theta)(x')$ .

**Dual representation:**  
 $h(\alpha)(x) = f(\theta_0)(x) + \sum_{i=1}^n \alpha_i k_{\theta_0}(x_i, x).$   
**Optimization problem:**  $\min_\alpha \frac{1}{2} \|\mathbf{K}_{\theta_0} \alpha - \tilde{\mathbf{y}}\|^2$   
**Optimal solution (kernel regression):**  
 $\alpha^* = \mathbf{K}_{\theta_0}^{-1}(\mathbf{y} - f(\theta_0)), h^*(x) = \mathbf{k}_{\theta_0}(x)^\top \alpha^*$

Functional Gradient Flow

Training dynamics in function space:  
 $\dot{\mathbf{f}}(\theta) = \mathbf{K}(\theta)(\mathbf{y} - \mathbf{f}(\theta))$   
• If  $\mathbf{K}(\theta)$  constant  $\rightarrow$  **linear ODE** with closed-form solution  
• If  $\mathbf{K}(\theta)$  evolves  $\rightarrow$  **nonlinear dynamics**, feature learning

**Infinite-Width Limit**  
Init:  $w_{ij}^{(\ell)} \sim \frac{\sigma_w}{\sqrt{m_\ell}} \mathcal{N}(0, 1)$ . Result: As width  $m \rightarrow \infty: \mathbf{k}_{\theta(t)} \rightarrow k_\infty$  (constant during training).  
• Kernel becomes **deterministic** (depends only on architecture/init scheme)  
• Training = kernel regression with frozen  $k_\infty$   
• No feature learning

**Finite-width:**  $\|\mathbf{K}(\theta_0) - \mathbf{K}(\theta(t))\| = \mathcal{O}(\frac{1}{m})$

Why Kernel Stays Constant:

Kernel  $\nabla K = \nabla^2 f(x) \nabla f(z) + \nabla^2 f(z) \nabla f(x)$ , at  $m = \infty: \nabla^2 f \rightarrow 0 \Rightarrow \nabla K \rightarrow 0$  kernel frozen.

Lazy Training (NTK Regime)	Feature Learning (Rich Regime)
$m \rightarrow \infty$ , small LR	Finite width, normal LR
$\mathbf{K}$ const $\rightarrow$ linear dynamics	$\mathbf{K}$ evolves $\rightarrow$ nonlinear dynamics
No feature learning	Learns representations
Theoretically tractable	SOTA performance

**Takeaways:**  
**Linearization** turns NN training into kernel regression with features  $\nabla f(\theta_0)(x)$ .  
**NTK**  $k_\theta = \nabla f(\theta)(x) \cdot \nabla f(\theta)(x')$  governs training dynamics via  $\dot{\mathbf{f}} = \mathbf{K}(\mathbf{y} - \mathbf{f})$ .  
**Infinite width**  $\rightarrow$  kernel constant  $\rightarrow$  NN = kernel machine (no feature learning).  
**Finite width**  $\rightarrow$  kernel evolves  $\mathcal{O}(\frac{1}{m}) \rightarrow$  enables feature learning.  
**NTK explains lazy regime but NOT why deep learning works**  $\rightarrow$  real power is feature learning when kernel changes.

6.3 Random NNs and GPs

Marginals + Conditionals of MV Gaussians

Let  $X \in \mathbb{R}^d \sim \mathcal{N}(\mu, \Sigma)$  with partition:  
 $X = \begin{pmatrix} x_A \\ x_B \end{pmatrix}, \quad \mu = \begin{pmatrix} \mu_A \\ \mu_B \end{pmatrix}, \quad \Sigma = \begin{bmatrix} \Sigma_{AA} & \Sigma_{AB} \\ \Sigma_{BA} & \Sigma_{BB} \end{bmatrix}$

**Marginal:**  $X_A \sim \mathcal{N}(\mu_A, \Sigma_{AA})$

**Conditional:**  $X_B \mid X_A \sim \mathcal{N}(\mu_{B|A}, \Sigma_{B|A})$

$\mu_{B|A} = \mu_B + \Sigma_{BA} \Sigma_{AA}^{-1} (X_A - \mu_A)$

$\Sigma_{B|A} = \Sigma_{BB} - \Sigma_{BA} \Sigma_{AA}^{-1} \Sigma_{AB}$

6.3.1 Bayesian Linear Regression

**Least-squares:**  $\hat{w} = \arg \min_w \frac{1}{2n\sigma^2} \|\mathbf{y} - \mathbf{X}w\|^2.$

**Closed-form solution:**  $\hat{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}.$

**MLE interpretation:**  $y_i = x_i^\top w + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2), \mathcal{L}(w) = \log \prod_{i=1}^n p(y_i \mid x_i, w),$   
 $y_i \mid x_i, w \sim \mathcal{N}(x_i^\top w, \sigma^2).$

**Prior:**  $p(w) = \mathcal{N}(0, I_d).$

**Posterior:**  $p(w \mid \mathbf{y}, \mathbf{X}) = \frac{p(\mathbf{y} \mid \mathbf{X}, w)p(w)}{p(\mathbf{y} \mid \mathbf{X})}.$

**Predictive distribution:**  $p(y_{n+1}) = \int p(y_{n+1} \mid w)p(w \mid \mathbf{y})dw.$

$\Sigma_{ww} = I_d, \Sigma_{yw} = \mathbf{X}, \Sigma_{yy} = \mathbf{X} \mathbf{X}^\top + \sigma^2 I_n.$

$\mu_{w|y} = \mathbf{X}^\top \Sigma_{yy}^{-1} \mathbf{y} = (\mathbf{X}^\top \mathbf{X} + \sigma^2 I_d)^{-1} \mathbf{X}^\top \mathbf{y},$   
 $\Sigma_{w|y} = I_d - \mathbf{X}^\top \Sigma_{yy}^{-1} \mathbf{X} = \sigma^2 (\mathbf{X}^\top \mathbf{X} + \sigma^2 I_d)^{-1}.$

Same result as ridge:  
 $\hat{w} = (\mathbf{X}^\top \mathbf{X} + \sigma^2 I_d)^{-1} \mathbf{X}^\top \mathbf{y}.$

Equiv. to GP with linear kernel:  $f(\cdot) \sim \mathcal{GP}(0, k(\cdot, \cdot)):$   $k(x, x') = \phi(x)^\top \phi(x'), \quad y = f + \varepsilon, \quad f \sim \mathcal{N}(0, K), \quad \varepsilon \sim \mathcal{N}(0, \sigma^2 I_n),$  posterior  $p(f|y) = \mathcal{N}(\mu_{f|y}, \Sigma_{f|y}), \mu_{f|y} = \mathbf{K}(\mathbf{K} + \sigma^2 I_n)^{-1} \mathbf{y}.$

For  $y = f(x) + \varepsilon, \quad f(x) = x^\top w, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2), p(y \mid X, w) = \mathcal{N}(X^\top w, \sigma^2 I).$  If  $w \sim \mathcal{N}(0, \Sigma), \quad p(w \mid x, X) = \mathcal{N}(\Sigma_{\text{post}}^{-1} \frac{1}{\sigma^2} Xy, \Sigma_{\text{post}}),$  where  $\Sigma_{\text{post}}^{-1} = \Sigma^{-1} + \frac{1}{\sigma^2} X X^\top.$  Maximizing  $\log p(w \mid x, X)$  is same as minimizing least-squares with  $\ell_2$  penalty  $\frac{1}{2} w^\top \Sigma^{-1} w$ , ridge when  $\Sigma = \lambda^{-1} I.$  **Predictive:**  $f_* \mid x_*, y, X = x_*^\top w \sim \mathcal{N}(x_*^\top \mu_{\text{post}}, x_*^\top \Sigma_{\text{post}} x_*).$   $y_* \mid x_*, y, X$  adds  $\sigma^2.$   $f = x^\top w$  with  $w \sim \mathcal{N}(0, \Sigma)$  is GP w  $k(x, x') = x^\top \Sigma x'.$

6.3.2 NNGPs

**Setup:** Random 1-hidden-layer NN with  $m$  units:  $f(x) = v_0 + \frac{1}{\sqrt{m}} \sum_{j=1}^m v_j \varphi(\theta_j^\top x).$  Random init:  $v_0 \sim \mathcal{N}(0, \sigma_0^2), \mathbb{E}[v_j^2] = \sigma_v^2, \text{Cov}(\theta_j) = \Sigma_\theta.$

**Result:** As  $m \rightarrow \infty, f(\cdot) \rightarrow \text{GP}(0, k)$  where  $k(x, x') = \sigma_0^2 + \sigma_v^2 \mathbb{E}_\theta [\varphi(\theta^\top x) \varphi(\theta^\top x')]$

**Monte Carlo approximation:** Sample  $B$  random NNs  $\{f_b\}_{b=1}^B$ , define features:

- $\varphi(x) = \frac{1}{\sqrt{B}} (f_1(x), ..., f_B(x))^\top$
- $\Phi = [f_1 \dots f_B] \in \mathbb{R}^{n \times B}$  (feature matrix)
- $\hat{K} = \Phi \Phi^\top$  (approximate kernel matrix)

**GP regression:** Posterior mean and variance:  
 $\mathbb{E}[f(x) \mid \mathbf{y}] = \varphi(x)^\top (\Phi^\top \Phi + \sigma^2 I_B)^{-1} \Phi^\top \mathbf{y}$   
 $\text{Var}[f(x) \mid \mathbf{y}] = \sigma^2 \varphi(x)^\top (\Phi^\top \Phi + \sigma^2 I_B)^{-1} \varphi(x)$

**Key advantage:** Inverts  $B \times B$  matrix instead of  $n \times n$  when  $B \ll n$ .

7 Generative Models

7.0.1 Linear Autoencoders

**Setup.** Encoder  $C \in \mathbb{R}^{k \times d}$ , decoder  $D \in \mathbb{R}^{d \times k}$ , data  $X \in \mathbb{R}^{d \times n}$  (centered cols):  $\min_{C, D} \|X - DCX\|_F^2.$

**Optimal Solution (PCA).** Let  $S = X X^\top$  with eigendecomposition  $S = Q \Lambda^2 Q^\top, \quad \lambda_1 \geq ... \geq \lambda_d \geq 0.$  Optimal reconstruction via rank- $k$  projection:  $\hat{X} = U_k U_k^{*\top} X$  where  $U_k^* = Q_{[:,1:k]}$  are top- $k$  eigenvectors of  $S$  (equiv, top- $k$  left singular vectors of  $X$ ).  
• Any  $C = A^{-1} (U_k^*)^\top, D = U_k^* A$  is optimal  
• Reduces to truncated SVD:  $\hat{X} = U^* \Lambda_k V^\top$  with  $\Lambda_k = \text{Diag}(\lambda_1, ..., \lambda_k, 0, ..., 0)$   
• Convex objective with no spurious local minima (gradient descent finds global optimum)  
• Singular vectors may not be uniquely identified

7.0.2 Factor analysis

**Latent Variable Models** are a generic way to describe generative models. Latent variable  $z \sim p(z)$ , conditional models for observables  $x, p(x|z)$ , observed data model:  $p(x) = \int p(x|z)p(z) dz.$

**Mixture models:** simple discrete models:  $z \in [K], p(z)$  mixing proportions,  $p(x|z)$  condit. densities.

$$p(x; \mu, \Sigma) = \frac{\exp[-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)]}{\sqrt{(2\pi)^n \det(\Sigma)}}$$

7.0.3 Linear Factor Analysis

- Latent prior:  $z \sim \mathcal{N}(0, I), z \in \mathbb{R}^m$
- Observation:  $x = \mu + Wz + \eta, \eta \sim \mathcal{N}(0, \Sigma)$
- Independence:  $\eta \perp z$
- Typically  $m < n$  (fewer factors than features)

**Marginal distr:**  $x \sim \mathcal{N}(\mu, WW^\top + \Sigma)$

- $WW^\top$ : shared variance (low-rank, explained by latent factors)
- $\Sigma$ : unique variance (diagonal, observation-specific)

Non-identifiability:  $(WQ)(WQ)^\top = WQQ^\top W^\top = WW^\top$  for any orthogonal  $Q$ . Factors only identifiable up to rotations/reflections.  $\Rightarrow$  Use factor rotations (varimax, etc.) for interpretability.

**MLE estimation:**  $\theta = (\mu, W) \xleftarrow{\max} \log p(X; \mu, W)$   
•  $\hat{\mu} = \frac{1}{s} \sum_{i=1}^s x_i$  (closed form)  
• No closed form for  $W \rightarrow$  use GD or EM algorithm

**Posterior (encoder):**  $p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)}.$   
 $\mu_{z|x} = W^\top (WW^\top + \Sigma)^{-1} (x - \mu).$   
 $\Sigma_{z|x} = I - W^\top (WW^\top + \Sigma)^{-1} W.$



**Probabilistic PCA:** Special case  $\Sigma = \sigma^2 I$ . Optimal  $i$ -th column:  $w_i = \rho_i u_i$ ,  $\rho_i^2 = \max\{0, \lambda_i - \sigma^2\}$ .  $W = U_m L_m$ , where  $(\lambda_i, u_i)$  is  $i$ -th eigenpair of data covariance.

As  $\sigma \rightarrow 0$ :  $\mu_{z|x} \rightarrow W^\dagger(x - \mu)$  (standard PCA). If  $W$  has orthogonal columns, then  $W^\dagger = W^\top$ .

7.1 Variational Autoencoders

$z \in \mathbb{R}^d$  is learned embedding of  $x$ . For generation,  $z \sim \mathcal{N}(0, I)$ , decoder  $p_\theta(x|z)$  maps latent to data.

**Problem:**  $p_\theta(x) = \int p(z)p_\theta(x|z) dz$  intractable. **Solution:** Maximize ELBO instead:  $\log p_\theta(x) \geq \underbrace{\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]}_{\text{reconstruction}} - \underbrace{D_{\text{KL}}(q_\phi(z|x) \parallel p(z))}_{\text{regularization}}$

- **Reconstruction:** Encode  $x \rightarrow z$ , decode back
- **KL term:** Keep encoder output close to prior  $p(z) \sim \mathcal{N}(0, I)$ , ensures generation using latents

**Encoder**  $q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \text{diag}(\sigma_\phi^2(x)))$ .

**KL closed form:**  $D_{\text{KL}}(q_\phi(z|x) \parallel p(z)) = \frac{1}{2} \sum_{j=1}^d (\sigma_{\phi,j}^2 + \mu_{\phi,j}^2 - 1 - \log \sigma_{\phi,j}^2)$ .

$\text{KL}(\mathcal{N}(\mu_0, \sigma_0^2) \parallel \mathcal{N}(\mu_1, \sigma_1^2)) = \frac{1}{2} \left( \frac{\sigma_0^2}{\sigma_1^2} + \frac{(\mu_0 - \mu_1)^2}{\sigma_1^2} - 1 + \log \frac{\sigma_1^2}{\sigma_0^2} \right)$

$\text{KL}(p \parallel q) = \mathbb{E}_p \left[ \log \frac{p(x)}{q(x)} \right]$

- **Fwd KL:**  $q_1^* = \arg \min_{q \in Q} \text{KL}(p \parallel q)$
- **Rev KL:**  $q_2^* = \arg \min_{q \in Q} \text{KL}(q \parallel p)$

Rev KL: Mode-seeking ( $p = 0 \Rightarrow q = 0$ ), FwdKL: Mean-seeking ( $p \neq 0 \Rightarrow q \neq 0$ ). MLE minimizes fwd KL to empirical  $D$ .

$\log p_\theta(x|z) = -\frac{1}{2\sigma^2} \|x - \mu_\theta(z)\|_2^2 - \frac{d}{2} \log(2\pi\sigma^2)$ .

**Reparameterization trick:**  $z = \mu_\varphi(x) + \sigma_\varphi(x) \epsilon$ ,  $\epsilon \sim \mathcal{N}(0, I)$ , enables backprop through sampling.

$\log p_\theta(x) - \text{ELBO} = D_{\text{KL}}(q_\phi(z|x) \parallel p_\theta(z|x))$ , **tight when  $q_\phi = \text{true posterior}$ .**

**MC estimation:**  $\frac{E_{q_\phi(z|x)}[\log p_\theta(x|z)]}{-\frac{1}{2\sigma^2 K} \sum_{k=1}^K \|x - \mu_\theta(z_k)\|_2^2 - \frac{d}{2} \log(2\pi\sigma^2)}$ .

**Generative Classifiers** Given  $y \in \{0, 1\}$ ,  $p(y=1) = p(y=0) = \frac{1}{2}$ ,  $p(x|y) = \mathcal{N}(x; \mu_y, I_d)$ , where  $\mu_0, \mu_1 \in \mathbb{R}^d$ ,  $p(y=1|x) = \frac{p(y=1)p(\frac{x}{\sigma_0} | y=1) + p(y=0)p(x|y=0)}{\frac{1}{2}(2\pi)^{-\frac{d}{2}} \exp(-\frac{1}{2}\|x-\mu_1\|^2)}$  =  $\frac{\frac{1}{2}(2\pi)^{-\frac{d}{2}} \exp(-\frac{1}{2}\|x-\mu_1\|^2) + \frac{1}{2}(2\pi)^{-\frac{d}{2}} \exp(-\frac{1}{2}\|x-\mu_0\|^2)}{1 + \exp(\frac{1}{2}\|x-\mu_1\|^2 - \frac{1}{2}\|x-\mu_0\|^2)}$  =  $\frac{1}{(1 + \exp(-(\mu_1 - \mu_0)x + \frac{1}{2}\|\mu_0\|^2 - \|\mu_1\|^2))}$  equiv to logistic regression where  $p(y=1|x) = \sigma(w^\top x + b)$  with  $w = \mu_1 - \mu_0$ ,  $b = \frac{1}{2}(\|\mu_0\|^2 - \|\mu_1\|^2)$ .

**ELBO for Hierarchical VAEs:** Model  $x$  by decoding from latents  $z = (z_1, \dots, z_L)$ .  $p_\theta(x, z) = p_\theta(x|z_1) \prod_{i=1}^{L-1} p_\theta(z_i|z_{i+1})p(z_L)$ . **Inference** top-down:  $q_\phi(z|x) = q_\phi(z_L|x) \prod_{i=1}^{L-1} q_\phi(z_i|z_{i+1})$ . **ELBO** for HVAE:  $\mathcal{L}(x) = \mathbb{E}_{z|x \sim q_\phi} \left[ \log \frac{p_\theta(x, z)}{q_\phi(z|x)} \right] = \mathbb{E}_{z|x \sim q_\phi} \left[ \log p_\theta(x|z_1) + \log \frac{p_\theta(z_1|z_2)}{q_\phi(z_1|x)} + \sum_{i=2}^{L-1} \log \frac{p_\theta(z_i|z_{i+1})}{q_\phi(z_i|z_{i-1})} + \log \frac{p_\theta(z_L)}{q_\phi(z_L|z_{L-1})} \right]$ .

**Change of variables** spherical to 3D euclidian  $(x, y, z) \mapsto (r \cos \theta \cos \phi, r \cos \theta \sin \phi, r \sin \theta)$ . Lengths of the three sides of an infinitesimal cuboid whose diagonally opposite vertices are at  $r, \theta, \phi$  and  $(r + dr, \theta + d\theta, \phi + d\phi)$  are  $(dr, r d\theta, r \cos \theta d\phi)$ . Volume is  $r^2 \cos \theta dr d\theta d\phi$ . Determinant of jacobian  $|\frac{\partial(x,y,z)}{\partial(r,\theta,\phi)}| = r^2 \cos \theta$ . Density on spherical coordinates  $p(r, \theta, \phi) \rightarrow$  density on Euclidian coordinates is  $p(x, y, z) = p(r, \theta, \phi) |\frac{\partial(x,y,z)}{\partial(r,\theta,\phi)}|^{-1}$ . Infinitesimal probability mass of the cuboid above is equal to the mass of a Euclidian cuboid of size  $(dx, dy, dz)$  at  $(x, y, z)$ .

7.2 Normalizing Flows

Transform simple distribution  $z \sim \mathcal{N}(0, I)$  through invertible map  $T$  to get complex  $x = T(z)$ . Exact likelihood (no ELBO like VAEs), easy sampling.

**Change of Variables Formula:**  $p_x(x) = p_z(T^{-1}(x)) \cdot |\det J_{T^{-1}}(x)|$ ,  $|\det J_{T^{-1}}(x)| = \frac{1}{|\det J_T(T^{-1}(x))|}$ .

**Diffeomorphism:**  $T$  is bijective, differentiable, w/ differentiable inverse. Guarantees  $\det J_T \neq 0$ .

**Computational problem:** Computing  $\det J$  is  $O(d^3)$  for dense Jacobian. **Solution:** Design  $T$  s.t. Jacobian is **triangular**, then only  $O(d)!$

Two architectures with triangular Jacobians:

	MAF	IAF
Fast / parallel	Density eval	Sampling
Slow / sequential	Sampling	Density

**Coupling layers:** Trick that makes both directions fast, at the cost of being less expressive per layer.

7.3 Generative Adversarial Networks

Likelihood-free generative model: train via adversarial game between two networks: **Generator**  $G_\theta$  maps latent  $z \sim p_z$  (typically Gaussian) to fake samples; **Discriminator**  $D_\varphi$ : outputs prob that input is **real**. **GAN Objective:**  $\min_{\theta} \max_{\varphi} \underbrace{\mathbb{E}_{z \sim p_z} [\log D_\varphi(x)]}_{\text{real samples}} + \underbrace{\mathbb{E}_{z \sim p_z} [\log(1 - D_\varphi(G_\theta(z)))]}_{\text{fake samples}}$ 

- **Discriminator** maximizes: correctly classify real (high  $D$ ) and fake (low  $D$ )
- **Generator** minimizes: fool discriminator (make  $D(G(z))$  high)

Common alternative objective for the generator is to maximize  $\mathbb{E}_{z \sim p(z)} [\log D(G(z))]$  instead of minimizing  $\mathbb{E}_{z \sim p(z)} [\log(1 - D(G(z)))]$  to help mitigate vanishing gradient problem when discriminator becomes too good, i.e.  $D(G(z)) \rightarrow 0$ . For a fixed generator  $G$ , **optimal discriminator**  $D^*$  is given by  $D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}$ . If **discriminator optimal**, GAN objective reduces to  $V(D^*, G) = 2D_{\text{JS}}(p_{\text{data}} \parallel p_G) - \log 4$ .

7.3.1 Theoretical Foundation

Binary clfcation with  $p(y=1) = p(y=0) = \frac{1}{2}$ :

- $y = 1$ : sample from real  $p_{r(x)}$
- $y = 0$ : sample from generator  $p_{\theta(x)}$

**Bayes Optimal Classifier** (prob that  $x$  is real):  $q_\theta(x) = P(y=1|x) = \frac{p_r(x)}{p_r(x) + p_\theta(x)}$ .

**Generator Logistic Objective = JS Div:**  $\ell^*(\theta) = \mathbb{E}_{p_\theta(y,x)}[y \ln q_\theta(x) + (1-y) \ln(1 - q_\theta(x))] = \text{JS}(p_r \parallel p_\theta) - \ln 2$ .

**Jensen-Shannon Divergence:**  $\text{JS}(p_r \parallel p_\theta) = \frac{1}{2} D_{\text{KL}}(p_r \parallel p_m) + \frac{1}{2} D_{\text{KL}}(p_\theta \parallel p_m)$ ,  $p_m = \frac{p_r + p_\theta}{2}$ . **Bounded:**  $0 \leq \text{JS}(p_r \parallel p_\theta) \leq \log 2$ .

**Jensen Inequality**  
If  $\varphi$  convex:  $\varphi(\mathbb{E}[X]) \leq \mathbb{E}[\varphi(X)]$ . If concave, other way around.

7.3.2 Training

Alternating SGD (heuristic, may diverge!). Training is **Saddle-point problem**, unstable!

**JS Divergence Saturates** when distributions don't overlap. If  $p_r$  and  $p_\theta$  have disjoint supports: discriminator perfect, no gradient for generator!

**Wasserstein Distance (Earth Mover's Distance):**

$W(p_r, p_\theta) = \inf_{\gamma \in \Pi(p_r, p_\theta)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|]$   
Minimum total "work" to transport mass from  $p_r$  to  $p_\theta$ . Provides meaningful gradients even without overlap.

**Dual (Kantorovich-Rubinstein):**

$W(p_r, p_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim p_r}[f(x)] - \mathbb{E}_{x \sim p_\theta}[f(x)]$ . Maximize gap by adv score of real vs fake samples w.r.t. Lipschitz constraint. Max achievable gap = Wasserstein distance.

**WGAN** uses critic  $f_w$  (not classical discriminator!):  $\min_{\theta} \max_w \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_z}[f_w(G_\theta(z))]$ . **Enforcing Lipschitz:**

- **Weight clipping** (original): crude, problematic
- **Gradient penalty:** add  $\lambda \mathbb{E}_{\hat{x}} [(\|\nabla_{\hat{x}} f_w(\hat{x})\|_2 - 1)^2]$

**Mode Collapse:** Generator produces only few samples that fool discriminator, ignoring full distribution diversity.

7.4 Diffusion Models

**Fixed Fwd process:** Gradually add Gaussian noise over  $T$  steps until data becomes pure noise.

Forward step:  $q(x_t | x_{t-1}) = \mathcal{N}(x_t; \sqrt{1 - \beta_t} x_{t-1}, \beta_t I)$ . Full fwd proces:  $q(x_{1:T} | x_0) = \prod_{t=1}^T q(x_t | x_{t-1})$ .

**Noise schedule:**  $\{\beta_t \in (0, 1)\}_{t=1}^T$  noise added at each step. **Define:**  $\alpha_t = 1 - \beta_t$  and  $\bar{\alpha}_t = \prod_{i=1}^t \alpha_i$ .

**Direct sampling (reparameterization trick):**  
 $q(x_t | x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t) I)$ .  
 $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon_0$ ,  $\epsilon_0 \sim \mathcal{N}(0, I)$ .

**Reverse process (learned):** Train NN to denoise step by step:  $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$ .

For small  $\beta_t$ , reverse  $q(x_{t-1}|x_t)$  is also Gaussian.

$\log p_\theta(x_0)$  intractable, so derive **Variational Lower Bound (VLB):**  $-\log p_\theta(x_0) \leq \mathcal{L}_{\text{VLB}} = \mathbb{E}_q \left[ \log \frac{q(x_{1:T} | x_0)}{p(x_{0:T})} \right]$ . **Decomposition into 3 terms:**  $\mathcal{L}_{\text{VLB}} = \underbrace{D_{\text{KL}}(q(x_T|x_0) \parallel p(x_T))}_{L_T} +$

$\underbrace{\sum_{t=2}^T \mathbb{E}_{q(x_t|x_0)} [D_{\text{KL}}(q(x_{t-1} | x_t, x_0) \parallel p_\theta(x_{t-1} | x_t))]}_{L_{t-1}} - \underbrace{\mathbb{E}_{q(x_1|x_0)} [\log p_\theta(x_0 | x_1)]}_{L_0}$ 

- $L_T$ : Is  $q(x_T | x_0) \approx \mathcal{N}(0, I)$ ? Not optimized.
- $L_{t-1}$ : Match learned reverse to true reverse
- $L_0$ : Reconstruction term

**Tractable Reverse Posterior**  $q(x_{t-1} | x_t, x_0)$  is Gaussian with closed form (product of Gaussians):  $q(x_{t-1} | x_t, x_0) = \mathcal{N}(x_{t-1}; \mu_{q,t}(x_t, x_0), \sigma_t^2 I)$ , with:

$\mu_{q,t}(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_0 \right)$ ,  $\sigma_t^2 = \frac{1 - \alpha_t}{1 - \bar{\alpha}_t} \beta_t$ .

7.5 Noise Prediction Parameterization

Predict the **noise**  $\epsilon_\theta(x_t, t)$  instead of mean directly. Parameterize learned mean to mirror true posterior:  $\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right)$ . Since both distributions are Gaussian with same variance:  $D_{\text{KL}}(\mathcal{N}(\mu_q, \sigma^2 I) \parallel \mathcal{N}(\mu_p, \sigma^2 I)) = \frac{1}{2\sigma^2} \|\mu_q - \mu_p\|^2$ . This simplifies  $L_{t-1}$  to comparing noise:  $L_{t-1} = \mathbb{E}_{x_0, \epsilon_0} \left[ \frac{(1 - \alpha_t)^2}{2\alpha_t(1 - \bar{\alpha}_t)\sigma_\epsilon^2} \|\epsilon_0 - \epsilon_\theta(x_t, t)\|^2 \right]$ .  $\mathcal{L}_{\text{simple}} = \mathbb{E}_{t \sim [1, T], x_0, \epsilon_0} [\|\epsilon_0 - \epsilon_\theta(x_t, t)\|^2]$ .

Training	Sampling
1. Sample real image $x_0 \sim q(x_0)$ 2. Sample random timestep $t \sim \text{Uniform}(\{1, \dots, T\})$ 3. Sample noise $\epsilon \sim \mathcal{N}(0, I)$ 4. Compute noisy image: $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$ 5. GD step on $\nabla_{\epsilon} \ \epsilon - \epsilon_\theta(x_t, t)\ ^2$	1. Sample $x_T \sim \mathcal{N}(0, I)$ 2. For $t = T, \dots, 1$ : $z \sim \mathcal{N}(0, I)$ if $t > 1$ , else $z = 0$ $x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sigma_t z$ Return $x_0$

Cosine noise schedule performs better than linear.

Used architecture is U-Net. **Input:** Noisy image  $x_t + \text{timestep } t$ ; **Output:** Predicted noise  $\epsilon_\theta(x_t, t)$ .

Model **conditional distribution**  $p_\theta(x_{0:T} | y)$  where  $y$  is condition (class, text, image). Extend denoiser to take  $y$  as input.

**Latent Diffusion Models (LDM)** run diffusion in **compressed latent space** instead of pixel space.

**Autoregressive Models**  
 $p(x) = \prod_{i=1}^d p(x_i | x_{<i})$ .

8 Tricks

**Short Connections** in DNs: Add less deep paths to a very deep network. **Residual connections:** shortcut and add back in. **Skip connections:** concatenate.

For  $y_1 = \alpha f(x, \Theta_1) + x$ ,  $x \in \mathbb{R}^d, y_i \in \mathbb{R}^d, \Theta_i \in \mathbb{R}^{d \times d}$  and  $y_{i \geq 2} = \alpha f(y_{i-1}, \Theta_i) + y_{i-1}$ ,  $l = L(y_n)$  it holds that  $\frac{\partial y_{k-1}}{\partial \Theta_k} = \alpha \frac{\partial f(y_{k-1}, \Theta_k)}{\partial y_{k-1}} + I_d$  and  $\frac{\partial y_k}{\partial \Theta_k} = \alpha \frac{\partial f(y_{k-1}, \Theta_k)}{\partial \Theta_k} \frac{\partial y_{k-1}}{\partial \Theta_k}$ . By applying the chain rule we have  $\frac{\partial}{\partial \Theta_k} = \frac{\partial L(y_n)}{\partial y_n} \frac{\partial y_n}{\partial y_{n-1}} \dots \frac{\partial y_{k+1}}{\partial y_k} \frac{\alpha \partial f(y_{k-1}, \Theta_k)}{\partial \Theta_k}$ . Set  $\alpha^2 = \frac{a}{n}$  for  $a > 0$  s.t.  $\lim_{n \rightarrow \infty} \mathbb{E} \|y\|^2 < \infty$ .

**8.1 Weight Decay & Early Stopping**  
**L2 regularization**  $\mathcal{R}_\Omega(\theta; \mathcal{S}) = \mathcal{R}(\theta; \mathcal{S}) + \Omega(\theta)$ ,  $\Omega_\mu(\theta) = \frac{\mu}{2} \|\theta\|^2, \mu \geq 0$ . Only penalize weights, not biases. **GD upd w/ WD:**  $\Delta \theta = -\eta \nabla \mathcal{R}(\theta) - \eta \nabla \Omega_\mu(\theta) = -\eta \nabla \mathcal{R}(\theta) - \eta \mu \theta$ .

Geometric interpration (local quadratic approx): Regularized optimum:  $\theta_\mu^* = (H + \mu I)^{-1} H \theta^*$ , where  $H = Q^\top \Lambda Q$  gives  $\theta_\mu^* = Q \text{diag}\left(\frac{\lambda_i}{\lambda_i + \mu}\right) Q^\top \theta^*$ .

$\lambda_i \gg \mu: \frac{\lambda_i}{\lambda_i + \mu} \approx 1 \rightarrow$  weak shrinkage (important dirs).  
 $\lambda_i \ll \mu: \frac{\lambda_i}{\lambda_i + \mu} \approx 0 \rightarrow$  strong shrinkage (flat dirs).

Adaptively shrinks based on loss geometry, preserves important dirs, removes unnecessary complexity.

**Early stopping:** Rather than training to convergence, stop when validation performance plateaus. Analysis shows that this is approximately equivalent to L2 regularization. GD trajectories can be approximated as  $\theta(k) = [I - (I - \eta \Lambda)^k] \theta^*$ . For small step sizes, behaves like weight decay when  $k = \frac{1}{\eta \mu}$ .

$L^1$  regularized second-order approximation of an arbitrary loss function around optimal  $\theta^*$  is  $R_{L^1}(\theta) \approx R(\theta^*) + \frac{1}{2}(\theta - \theta^*)^\top H(\theta - \theta^*) + \lambda \|\theta\|_1$ . Assuming  $H = \text{diag}(h_1, \dots, h_d)$ , we get  $R_{L^1}(\theta) \approx \sum_{i=1}^d \left[ \frac{h_i}{2} (\theta_i - \theta_i^*)^2 + \lambda |\theta_i| \right] + \text{const}$  so we need to minize  $f(a) = \frac{1}{2}(a - b)^2 +$

$\beta|a|$  and this gives  $a^* = \text{sgn}(b) \max\{0, |b| - \beta\}$ , so  $\theta_i = \text{sgn}(\theta_i^*) \max\{0, |\theta_i^*| - \frac{\lambda}{h_i}\}$ . For  $L^2$  regularization we get  $\theta_i = \frac{h_i}{h_i + \lambda} \theta_i^*$ . **Connecting  $L^2$  w early stopping:**  $R(w) \approx R(w^*) + \frac{1}{2}(w - w^*)^\top H(w - w^*)$ .  $\nabla R(w) = H(w - w^*)$ . GD update:  $w^t = w^{t-1} - \eta H(w^{t-1} - w^*)$  gives  $w^t - w^* = (I_d - \eta H)(w^{t-1} - w^*)$ . Using  $H = Q\Lambda Q^\top$ :  $Q^\top(w^t - w^*) = (I_d - \eta\Lambda)Q^\top(w^{t-1} - w^*)$ . If  $w^0 = 0$ ,  $Q^\top(w^t - w^*) = (I_d - \eta\Lambda)^t Q^\top(0 - w^*) \Rightarrow Q^\top w^t = [I_d - (I_d - \eta\Lambda)^t] Q^\top w^*$ . Optimal  $w$  under  $L^2$  reg gives  $Q^\top w = [I_d - \lambda(\Lambda + \lambda I_d)^{-1}] Q^\top w^*$ . Matching both gives  $t \approx \frac{1}{\eta\lambda}$ . **Weight normalization** is like Batch-Norm, with the covariance matrix replaced by the identity matrix.

### 8.2 Ensemble Methods

**Bagging:** Create  $K$  bootstrap samples of Data (sampling with replacement), train separate models, and average predictions:  $p(y|x) = \frac{1}{K} \sum_{k=1}^K p(y \mid x; \theta_k)$ .

**Dropout:** Randomly drop units during training with probability  $1 - \pi$ . Creates an exponential ensemble of sub-networks sharing weights. Test time: Scale weights by  $\pi$  to approximate the ensemble average.

### 8.3 Normalization

**Batch Norm:** Normalize activations across mini-batch:  $\tilde{z} = \frac{z - \mu_{\text{batch}}}{\sigma_{\text{batch}}}$ ,  $\hat{z} = \alpha \tilde{z} + \beta$ ,  $\mu_{\text{batch}} = \frac{1}{b} \sum_{i=1}^b z_i$ ,  $\sigma_{\text{batch}} = \sqrt{\frac{1}{b} \sum_{i=1}^b (z_i - \mu_{\text{batch}})^2}$ . **Layer Norm:** Normalize features in a layer instead; particularly effective for RNNs (batch statistics are less stable).

### 8.4 Data / Task Augmentation

Augment Data by applying valid transformations. Semi-supervised Learning: Train jointly on labeled and unlabeled data w combined loss. Pre-training & Fine-tuning. Multi-task Learning. Self-supervised Learning: Create free supervision from data.

## 9 Recurrent Neural Networks

**Evolution:**  $z_t = F[\theta](z_{t-1}, x_t)$ , with  $z_0 = 0$ . Optional output:  $y_t = G[\theta](z_t)$ .

**Simple RNN:**  $z_t = \phi(Wz_{t-1} + Ux_t)$  where  $W \in \mathbb{R}^{m \times m}$ ,  $U \in \mathbb{R}^{m \times n}$

**Backpropagation Through Time** (param sharing):  $\frac{\partial R}{\partial w_{i,j}} = \sum_t \frac{\partial R}{\partial z_i^t} \cdot \dot{\phi}_i^t \cdot z_j^{t-1}$ .  $\frac{\partial R}{\partial u_{i,k}} = \sum_t \frac{\partial R}{\partial z_i^t} \cdot \dot{\phi}_i^t \cdot x_k^t$ ;  $\dot{\phi}_i^t = \phi'(F_i(z^{t-1}, x^t))$ . **Gradient flow backward through time:**  $\nabla_{x_t} \mathcal{R} = [\prod_{r=t+1}^s W^\top S(z^r)] \cdot J_G \cdot \nabla_y \mathcal{R}$

**Spectral analysis:**  $\|\prod W^\top S(z^r)\|_2 \leq [\sigma_{\max}(W)]^{s-t}$  **Root cause:** Repeated matmul through time.

$\Rightarrow$  Simple RNNs cannot learn long dependencies.

**Deep RNNs** stack layers vertically:  $z^{t,\ell} = \varphi(W_\ell z^{t-1,\ell} + U_\ell z^{t,\ell-1})$  where  $z^{t,0} = x_t$ .

For RNN with  $z_{t+1} = \varphi(Uz_t + Vx_{t+1})$ ,  $L = \sum_{t=1}^T \ell(\hat{y}_t, y_t)$ , where  $\hat{y}_t$  depends on  $z_t$ . Then  $\frac{\partial L}{\partial U} = \sum_{t=1}^T \frac{\partial L}{\partial z_t} \cdot (\varphi'_t \cdot z_t)$ ,  $\frac{\partial L}{\partial V} = \sum_{t=1}^T \frac{\partial L}{\partial z_t} \cdot (\varphi'_t \cdot x_{t+1})$ .

**Weight Sharing in RNNs (LSTM):**

**Proof idea:** Introduce dummy parameters  $\tilde{W}_i = f(W)$  for each time step. By chain rule:  $\frac{\partial L}{\partial \tilde{W}_i} = \sum_j \frac{\partial L}{\partial \tilde{W}_i} \frac{\partial \tilde{W}_i}{\partial W}$ . With constraint  $\tilde{W}_i = W$ , we have  $\frac{\partial L}{\partial \tilde{W}_i} = I$ , giving the sum. Initialization of bias in RNNs: Use 1.

**Backpropagation for example network**

- $a_t = F(x_t, h^{(t-1)}, y^{(t-1)}; \theta)$
- $h_t = \sigma(a_t)$
- $\hat{y}_t = G(h_t; \varphi)$
- $L_t = H(\hat{y}_t, y_t)$
- $L = \sum_{t=1}^T L_t$

$$\frac{\partial L}{\partial \theta} = \sum_{t=1}^T \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \sum_{i=1}^t \left( \prod_{j=t}^{i+1} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_i}{\partial \theta} \\ \frac{\partial h_j}{\partial h_{j-1}} = \frac{\partial h_j}{\partial a_j} \frac{\partial a_j}{\partial h_{j-1}}, \frac{\partial h_i}{\partial \theta} = \frac{\partial h_i}{\partial a_i} \frac{\partial a_i}{\partial \theta}$$

### 9.1 Long Short-Term Memory (LSTM)

- $C_t$ : cell state (internal memory, protected high-way)
- $z_t$ : hidden state (external output, filtered view)

$$C_t = \underbrace{\sigma(F\tilde{x}^t)}_{\text{forget}} \odot C_{t-1} + \underbrace{\sigma(G\tilde{x}^t) \odot \tanh(V\tilde{x}^t)}_{\text{input}} \\ z_t = \underbrace{\sigma(H\tilde{x}^t)}_{\text{output}} \odot \tanh(C_t), \text{ where } \tilde{x}^t = [x_t, z_{t-1}].$$

### 9.2 Gated Recurrent Unit (GRU)

**Single state  $z_t$ . Input:**  $\tilde{x}^t = [x_t, z_{t-1}]$ .  $u_t = \sigma(U\tilde{x}^t)$ ,  $r_t = \sigma(R\tilde{x}^t)$ ,  $z_t = u_t \odot z_{t-1} + (1 - u_t) \odot \tanh(W[r_t \odot z_{t-1}, x_t])$

Often comparable to LSTM with fewer resources. Gating creates identity paths  $\rightarrow$  better gradient flow.

### 9.3 Linear Recurrent Models

RNNs not parallelizable during training. LRU has linear dynamics:  $z_{t+1} = Az_t + Bx_t$ . Diagonalize to  $A = P\Lambda P^{-1}$ ,  $\lambda_i \in \mathbb{C}$ , change basis  $\zeta_t = P^{-1}z_t$ . Then:  $\zeta_{t+1} = \Lambda \zeta_t + Cx_t$ . Each dimension evolves independently (no channel mixing). Compensate with expressive output:  $y_t = \text{MLP}(\text{Re}(Gz_t))$ .

**Stability:** Require  $\max|\lambda_j| \leq 1$  (spectral radius  $\leq 1$ ).

**Parameterization:**  $\lambda_i = \exp(-\exp(\nu_i) + i\varphi_i)$  ensures  $|\lambda_i| \in (0, 1)$  automatically,  $|\lambda_i| \approx 1$ : Long-term memory,  $|\lambda_i| \approx 0$ : Short-term pat-

terns.

**Provably universal** as sequence-to-sequence map.

#### 9.3.1 Connectionist Temporal Classification

**Problem:** Unsegmented sequences (e.g., speech). **Solution:** RNN outputs prob distribution over vocabulary at each time step. Model all alignments with blank symbol  $^{\text{“-”}}$ :  $p(\ell \mid x) = \sum_{\pi \in \mathcal{B}^{-1}(\ell)} \prod_t y_{\pi_t}$ .  $\mathcal{B}$  removes blanks and repeated symbols.

### 9.4 Sequence Learning

**Teacher Forcing:**  $p(y^t)$  depends on  $y^{1:t-1}$  only through  $z^t$ , means during autoregressive generation, model doesn't see its own predictions. **Solution:** Add feedback connections from  $y^{t-1}$  to  $z^t$ :  $z^t = \text{RNN}(z^{t-1}, x^t, y^{t-1})$ , now model conditions on its own previous predictions  $\rightarrow$  more coherent gen.

**Professor Forcing:** Train two networks (teacher-forced + free-running), discriminator matches hidden states  $\rightarrow$  improved generalization.

**Exposure bias:** Model relies on itself where inputs come from the previous output because of the non-availability of the ground truth.

**Seq2Seq:** Input and output sequences have different lengths: Use encoder-decoder framework.

Gradients in bi-directional RNNs are computed by making a forward and backward run, then at timestep  $t$  we combine (concatenate/add) and continue with the backpropagation. This happens at every bidirectional layer.

To initialize points uniformly on the unit ring in  $\mathbb{C}$  between circles of radii  $r_2 > r_1$ , sample from  $\exp(\frac{1}{2} \log(u_1(r_2^2 - r_1^2) + r_1^2) + i2\pi u_2)$  w  $u_1, u_2 \sim \mathcal{U}(0, 1)$ .

## 10 Attention and Transformers

**Seq2Seq with Attention:** Encoder generates hidden state sequence. Decoding RNNs output attends to encoder states and gets used as input in next step.

**Attention:** Learn to index, multiplicative gating to combine bottom-up and top-down information.

**KV - attention map:**

$F(\xi, ((x_1, z_1)), \dots, (x_s, z_s)) = [z_1, \dots, z_s] \cdot f(\xi, (x_1, \dots, x_s))$  consisting of a query  $\xi$  (what to look for?), keys  $x_i$  (index) and values  $z_i$ .

**Scaled Dot-Product Attention:**  $f(\xi, x) = \frac{\xi \cdot x}{\sqrt{n}}$

**Multi-headed attention:**

$$G\left(\xi, (x^t, z^t)^s_{t=1}\right) = W \begin{bmatrix} F_1(\xi, (x^t, z^t)) \\ \vdots \\ F_h(\xi, (x^t, z^t)) \end{bmatrix},$$

where

$$F_j(\xi, (x^t, z^t)) = F\left(W_j^q \xi, \left(W_j^x x^t, W_j^z z^t\right)\right).$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

where

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V \quad \text{and} \quad X \in \mathbb{R}^{T \times d_{\text{model}}}, \quad W_Q, W_K \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad W_V \in \mathbb{R}^{d_{\text{model}} \times d_v}, \quad Q, K \in \mathbb{R}^{T \times d_k}, \quad V \in \mathbb{R}^{T \times d_v}.$$

**Positional encodings** necessary since no use of recurrence. Can use predefined approaches, or learned.

Self attention used in encoder, masked self-attention in decoder. Can also add cross-attention to decoder.

**ELMo** consists of multiple layers of 2 LSTMs working in opposite directions. Can then be used to collapse all layers  $2L + 1$  and train in task-specific manner.

**BERT** is trained on two simultaneous tasks (Masked Language Modeling and Binary Prediction whether sentence B follows A). Has bidirectional encoder, and task-specific heads. Can do FT for various tasks.

**Vision Transformers** split images into patches, add pos embeddings and a [CLS] token, then process with a standard transformer encoder.

**Complexity metrics for different layer types in terms of input sequence length  $n$ .**

**Key Metrics:**

- Complexity per Layer:** Total computational operations per layer
- Sequential Operations:** Number of operations needed to connect any two input positions
- Maximum Path Length:** Longest path between any two input positions in the network

**Self-Attention:**

- Complexity per Layer:  $\mathcal{O}(n^2 d)$  – quadratic in sequence length due to all-pairs attention
- Sequential Operations:  $\mathcal{O}(1)$  – fully parallelizable
- Maximum Path Length:  $\mathcal{O}(1)$  – direct connections between all positions

**RNN:**

- Complexity per Layer:  $\mathcal{O}(nd^2)$  – linear in sequence length
- Sequential Operations:  $\mathcal{O}(n)$  – must process sequentially
- Maximum Path Length:  $\mathcal{O}(n)$  – information flows through entire sequence

**Trade-off:** Self-attention enables parallel processing and direct long-range connections but has quadratic complexity, while RNNs have linear complexity but require sequential processing.

There is **Linear Attention**, where compute and memory scales  $\mathcal{O}(1)$  with incrementing sequence length.

#### 11 Ethics

**Adversarial examples** (given  $f(x) = y$  correctly):

- Untargeted:  $\|\delta\| \leq \epsilon$  s.t.  $f(x + \delta) \neq y$ .
  - Optimize  $\max_{\|\delta\| \leq \epsilon} L(f(x + \delta), y)$ .
- Targeted:  $\|\delta\| \leq \epsilon$  s.t.  $f(x + \delta) = t \neq y$ .
  - Optimize  $\min_{\|\delta\| \leq \epsilon} L(f(x + \delta), t)$ .

**Linear Binary** ( $y \in \{-1, 1\}$ ,  $f(x) = w^\top x + b$ ):

**Correct:**  $y(w^\top x + b) > 0$ . Adv. flips when  $|y w^\top \delta| \leq -y(w^\top x + b)$  cross hyperplane. **L2 optimal:**  $\delta^* = \frac{-w^\top x + b}{\|w\|_2^2} w$ ,  $\|\delta^*\|_2 = \frac{|w^\top x + b|}{\|w\|_2}$ .  $L_\infty$  optimal:  $\delta = -\epsilon \text{ sign}(yw)$ . **Multiclass:**  $f_k(x) = w_k^\top x + b_k$ , use  $\text{argmax}_k f_k(x)$ .

**Margin** to class  $j$ :  $m_j(x) = (w_j - w_j^*)^\top x + (b_j - b_j^*)$ .  $f_y(x) = f_j(x) \Leftrightarrow m_j(x) = 0$ . **Correct** if  $f_y(x) > f_j(x) \ \forall j \neq y$ . **adversarial** if  $\exists j \neq y$  s.t.  $f_y(x + \delta) < f_j(x + \delta)$ . Distance to boundary:  $\frac{m_j(x)}{\|w_j\|_2}$

$\|w_y - w_j\|_2$ . Adversarial attacks for **NNs**: Approximate boundary by  $f(x + \delta) \approx f(x) + \nabla f(x)^\top \delta$ . **FGSM** is a one-step  $L_\infty$  attack:  $\delta = \epsilon \text{ sign}(\nabla_x L(f(x), y))$ . **PGD** is multi-step  $\delta_{\{t+1\}} = \text{Proj}_{\|\delta\| \leq \epsilon}(\delta_t + \alpha \text{ sign}(g_t))$ .

**Distributionally Robust Optimization:**

$\min_f \sup_{Q \in \mathcal{U}(P)} E_Q[L(f(x))]$ , where  $\mathcal{U}$  means close. Can use upper bound on Wasserstein distance e.g.

**Robust training**  $\min_f \mathbb{E}[\max_{S \in \mathcal{S}} L(f(x + \delta), y)]$ .

Adversarial training can be viewed as robustness to distribution shift measured by Wasserstein distance.

**Interpretability:** Local - explain pred for specific  $x$ , Global - explain model behaviour on avg over data.

**Local:** Ceteris paribus (vary  $x_j$ , fix  $x_{-j}$ ), Sensitivity ( $\partial_{x_j} f(x)$ ), missing info ( $f(x) - \mathbb{E}[f(X) \mid X_{-j} = x_{-j}]$ ). **Global:** Mutual info ( $I(X_j; Y \mid X_{-j})$ ), Predictive util (train  $f$  w/ and w/o  $x_j$ ). For log-loss predictive util  $\approx$  conditional mutual information.

SHAP attributes predictions, while SAGE attributes risk reduction.  $A$  protected attribute,  $Y$  target outcome,  $\hat{Y}$  prediction. Demographic Parity:  $\hat{Y} \perp A$ ; Equalized Odds:  $\hat{Y} \perp A \mid Y$ , Equality of Opportunity:  $\hat{Y} \perp A \mid Y = 1$ .