Background

$\log(x+1) \approx x$ if $|x| << 1$

Variance and covariance:

$\mathrm{Var}(X) = \mathrm{Cov}(X,X) = \mathbb{E}[(X-\mathbb{E}X)^2] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$

$\mathrm{Cov}(\mathbf{X},\mathbf{Y}) = \mathbb{E}[(\mathbf{X}-\mathbb{E}\mathbf{X})(\mathbf{Y}-\mathbb{E}\mathbf{Y})^T] = \mathbb{E}[\mathbf{X}\mathbf{Y}^T] - \mathbb{E}[\mathbf{X}]\mathbb{E}[\mathbf{Y}]^T$

$\mathrm{Cov}(aX+bY, cW+dV) = ac\mathrm{Cov}(X,W)+ad\mathrm{Cov}(X,V)+bc\mathrm{Cov}(Y,W)+bd\mathrm{Cov}(Y,V)$

Integration by parts: $\int f\cdot g' = f\cdot g - \int f'\cdot g$

Geometric sums & series: $\sum_{k=0}^{n} ar^k = a(\frac{1-r^{n+1}}{1-r}) \overset{n\to\infty}{\longrightarrow} \frac{a}{1-r}, |r|<1$

Hoeffding's inequality:

For $Z_i \in [0,1]$ iid, $\mathbb{P}(\frac{1}{n}\sum_{i=1}^n Z_i - \mathbb{E}[Z] \geq t) \leq \exp\{-2nt^2\}$.

For $Z_i \in [a_i,b_i]$ the upper bound is $\exp(-\frac{2nt^2}{\sum_{i=1}^n (b_i-a_i)^2})$

Boole's inequality - union bound $\mathbb{P}\left(\bigcup_i A_i\right) \leq \sum_i \mathbb{P}(A_i)$

Cauchy-Schwarz inequality: $v^T u \leq \|v\|_2\|u\|_2$

Hölder's inequality: $v^T u \leq \|v^T u\|_1 \leq \|v\|_p\|u\|_{p^*}$,

where $1/p + 1/p^* = 1$, equality if $|v|^p = \gamma|u|^{p^*}$

Jensen's inequality: f convex, then $f(\mathrm{E}[X]) \leq \mathrm{E}[f(X)]$

$\Rightarrow \log(\mathrm{E}[X]) \geq \mathrm{E}[\log(X)]$

Markov's inequality: $\mathrm{P}(X \geq a) \leq \frac{\mathrm{E}[X]}{a}$

KL Divergence Let Q, P be prob. distr. of a continuous RV $x \in \mathbb{R}^d$, with densities q,p. Then $KL(Q\|P) = \int_{-\infty}^{\infty} q(x)\log\frac{q(x)}{p(x)}\,dx$

Entropy For $X \in \{x_1,..,x_n\}$, $H(X) = -\sum P(x_i)\log P(x_i)$

$\left(\sum_i a_i\right)^2 = \sum_i a_i^2 + 2\sum_{i<j} a_i a_j$

For square matrices A, B: $\det A = \det A^t$, $\det AB = \det A\cdot\det B$, $\det A^{-1} = \frac{1}{\det A}$

Moment Generating Function $M_x : \mathbb{R}^n \to \mathbb{R}, M_x(t) = \mathbb{E}_x[\exp(t\cdot x)]$. $M_{x+y} = M_x \cdot M_y$

For $x \sim \mathcal{N}(\mu,\Sigma)$, we have $M_x(t) = \exp(t\cdot\mu + \frac{1}{2}t^T\Sigma t$

Losses

Cross entropy loss $H(x) = -\sum_{c=1}^{M} y_{o,c}\log P(x_{o,c})$

Distributions

Standard normal: $p(x|\mu,\sigma^2) = \frac{e^{-(x-\mu)^2/(2\sigma^2)}}{\sqrt{2\pi\sigma^2}}$

Multivariate normal: $x \sim \mathcal{N}(\mu,\Sigma)$

$p(x|\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{k}{2}}\det(\Sigma)^{\frac{1}{2}}}\exp(-\frac{1}{2}(x-\mu)^t\Sigma^{-1}(x-\mu))$

Exponential: $p(x|\lambda) = \lambda e^{-\lambda x}$

Bernoulli: $p(x|p) = p^x(1-p)^{1-x}$

Binomial: $p(x|n,p) = \binom{n}{x}p^x(1-p)^{n-x}$

Poisson: $p(x|\lambda) = \frac{\lambda^x \exp[-\lambda]}{x!}$

Connectionism

McCulloch & Pitts neuron: $f(x;\sigma,\theta) = \begin{cases}1, & \sum_{i=1}^n \sigma_i x_i \geq \theta\\ 0, & \text{else}\end{cases}$

$x \in \{0,1\}^n$, $\sigma \in \{\pm 1\}^n$, $\theta \in \mathbb{Z}$

Disjunctive Normal Form: $f(\mathbf{x};\sigma,\theta) = \vee_{I\in\mathcal{I}}(\wedge_{i\in I} x_i \wedge_{i\notin I}\bar{x}_i)$, $\mathcal{I} = \{I : \sum_{i\in I}\sigma_i \geq \theta\}$ (OR of ANDs)

Turing Type A machine (NAND):

$y(t+1) = 1 - x_1(t)x_2(t)$, $x_1(t), x_2(t) \in \{0,1\}$

Turing Type B machine:

$A(1,\mathrm{NAND}(x_1,x_1)) = x_1$, $A(0,\mathrm{NAND}(x_1,x_1)) = 1$

$\mathrm{NAND}(A,x_2,...,x_n) = \begin{cases}\mathrm{NAND}(x_2,...,x_n) & \text{if } A \leftarrow 1\\ \mathrm{NAND}(x_1,...,x_n) & \text{else}\end{cases}$

Perceptron

$(\mathbf{x},\theta) \to \mathrm{sgn}(\mathbf{x}\cdot\theta)$, update rule: $\Delta\theta = \begin{cases}\mathbf{0}, & y(\mathbf{x}\cdot\theta) \geq 0\\ y\mathbf{x}, & \text{otherwise}\end{cases}$

Path of updates always zig-zag since $\Delta\theta\cdot\theta < 0$ for updates

Update rule is SGD for the loss: $l(\mathbf{x},y;\theta) = \max\{0, -y\mathbf{x}\cdot\theta\}$

Lemma (Norm Growth): $(\mathbf{x}^t, y^t)$ perceptron mistakes inducing updates $\Delta\theta^t, \theta^s = \sum_{t=1}^s \Delta\theta^t$. Then:

$\|\theta^s\|^2 \leq \sum_{t=1}^s \|\mathbf{x}^t\|^2$ (prove by induction!)

Cor: If $\|\mathbf{x}^t\| \leq 1$ then $\|\theta^s\| \leq \sqrt{s}$

Def (Linear Separability): $\mathcal{S}$ linearly separable with margin $\gamma > 0$ if $\exists\theta^*, \|\theta^*\| = 1 : y\mathbf{x}\cdot\theta^* \geq \gamma > 0 \quad \forall(\mathbf{x},y) \in \mathcal{S}$

Novikov's convergence theorem: Converges in at most γ^{-2} steps (if assume $\|\mathbf{x}^t\| \leq 1$). Show $\gamma s \leq s$ and start with fact that $\Delta\theta^t\theta^* = y^t x^t\theta^* \geq \gamma + \theta^s = \sum\Delta\theta^t$ + Cauchy-Schwarz

Cover's theorem: Dichotomies possible with linear separators: $C(s,n) = 2\sum_{i=0}^{n-1}\binom{s-1}{i}$

Prove by showing recurrence relation $C(s+1,n) = C(s,n)+C(s,n-1)$ using Pascal's rule: $\binom{n-1}{k}+\binom{n-1}{k-1} = \binom{n}{k}$

VC dimension: largest set S that can be shattered. Corollary of Cover's thm: n points can be shattered by linear functions in n dimensions (meaning can realize all 2^n points). $m > n$ can not.

Willshaw Memory

Hebb rule: $\Delta\theta_{ij}^t \propto x_i^t x_j^t$ (neurons that fire together, wire together)

r-sparse Boolean vectors $\mathbb{B}_r^n = \{x \in \{0,1\}^n | \sum x_i \leq r\}$

Upper bound on number of patterns s: $s \leq \frac{n^2}{r\log n}$,

$\log\binom{n}{r} \approx r\log n$ is pattern information and n^2 total number of bits

Binary memory matrix: $\Theta_{ji} = \min\{1, \sum_{t=1}^s y_j^t x_i^t\}$, $\Theta_{ji} \in \{0,1\}^{nxn}$

Alternatively: $\Theta = \min\{\mathbf{1}, \sum_{t=1}^s \mathbf{y}^t(\mathbf{x}^t)^T\}$

Retrieve y from given x: $z = \Theta x$, $y_j = \begin{cases}0 & z_j < r\\ 1 & \text{else}\end{cases}$

Monotonicity: $x^t \mapsto y \geq y^t, \forall(x^t,y^t) \in \mathcal{S}$ (i.e. get at least as much as ask for). Proof: $\Theta = \min\{\mathbf{1}, \sum_\tau \Theta^\tau\} \geq \min\{\mathbf{1}, \Theta^t\} = \Theta^t$

$\mathbf{z}^t = \Theta\mathbf{x}^t \geq \Theta^t\mathbf{x}^t = \mathbf{y}^t(\mathbf{x}^t)^T\mathbf{x}^t = r\mathbf{y}^t$

Maximal capacity: $\max_q I(\text{patterns}) = (\log 2)n^2 \approx 0.693n^2$ in the limit, for $r = \log n$. Half of θ_{ji} will be 0 and half will be 1 as $n \to \infty$

Hopfield Networks

Idea: Store patterns in Θ, for recovery from corrupted patterns

$\Theta = \sum_{t=1}^s [x_t x_t^T - I_n] \in Z^{n\times n}$

Asynchronous update: $\bar{x}_i \leftarrow \mathrm{sign}(\sum_{j\neq i}\theta_{ij}\bar{x}_j + \theta_{i0})$ (Hopfield dynamics)

Synchronous updates can lead to limit cycles

Lyapunov function: $= -\sum_{i,j=1}^n \theta_{ij}x_i x_j - \sum_{i=1}^n \theta_{i0}x_i$

Hebb -Hopfield $\Theta \approx \sum_{t=1}^{\hat{s}} [x^{tT}x^t - I] \in \mathbb{R}^{nxn}$. Use fact that for $\hat{x}$ with hamming-dist k to x, $x\cdot\hat{x} - 1 = (n-2k-1)$ to show $-x, x$ both attractors. Capacity for num of patterns is $\frac{\alpha}{n} \leq \alpha^* \approx 0.138$. For Hopfield (without Hebb's rule, but with optimal weights) $\alpha^* = 2$

Linear Networks

Linear unit: $\iota(\mathbf{x};\boldsymbol{\theta}) = \mathbf{x}\cdot\boldsymbol{\theta}$, Affine unit: $\iota(\mathbf{x};\boldsymbol{\theta},b) = \mathbf{x}\cdot\boldsymbol{\theta} + b$

Level sets: $\iota(\mathbf{x}+\Delta\mathbf{x},\theta) = \iota(\mathbf{x},\theta)$, $\Delta\mathbf{x} \perp \theta$

Linear unit defines: direction of change via $\frac{\theta}{\|\theta\|}$, rate of change via $\|\theta\|$

Learning algo for linear units: $\Delta\theta = \eta(y - \mathbf{x}\cdot\theta)\mathbf{x}$

Homogeneity: $f(\alpha\mathbf{x}) = \alpha f(\mathbf{x})$

Additivity: $f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$

$\Rightarrow f$ is linear with these properties

f, g linear $\Rightarrow f\circ g$ linear

Affine functions: $f(\alpha\mathbf{x}+\beta\mathbf{y}) = \alpha f(\mathbf{x}) + \beta f(\mathbf{y})$, $\forall\mathbf{x},\mathbf{y} : \forall\alpha,\beta : \alpha+\beta = 1$

Linear Autoencoder

Def: $\mathbf{x} \mapsto \mathbf{z} \mapsto \mathbf{y}$, $\mathbf{z} = \mathbf{C}\mathbf{x}, \mathbf{y} = \mathbf{D}\mathbf{z}$, $\mathbf{C},\mathbf{D}^T \in \mathbb{R}^{m\times n}, m < n$

Loss: $l(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|^2$

Matrix notation: $\mathbf{X},\mathbf{Y} \in \mathbb{R}^{n\times s}$: $\theta = (\mathbf{C},\mathbf{D}) \to_{min} \frac{1}{2s}\|\mathbf{X} - \mathbf{D}\mathbf{C}\mathbf{X}\|_F^2$

$\frac{\delta l(\mathbf{x})}{\delta\mathbf{C}} = \mathbf{D}^T(\mathbf{y}-\mathbf{x})\mathbf{x}^T \in \mathbb{R}^{m\times n}$

$\frac{\delta l(\mathbf{x})}{\delta\mathbf{D}} = (\mathbf{y}-\mathbf{x})\mathbf{x}^T\mathbf{C}^T \in \mathbb{R}^{n\times m}$

$\mathrm{rank}(\mathbf{DC}) \leq \min\{\mathrm{rank}(\mathbf{C}), \mathrm{rank}(\mathbf{D})\} \leq m < n \quad \Rightarrow \mathrm{rank}(\mathbf{Y}) \leq m$

Eckhart-Young Theorem: $\|\mathbf{X} - \mathbf{X}_r\|_F = \min_{\mathrm{rank}(\mathbf{Y})\leq r}\|\mathbf{X}-\mathbf{Y}\|_F$

$\mathbf{C} = \mathbf{U}_m^T, \mathbf{D} = \mathbf{U}_m \Rightarrow \mathbf{DCX} = \mathbf{X}_m$

Gradients of deep linear networks

Assume $\mathbf{X},\mathbf{Y}$ centered, $\mathbf{X}$ whitened: $\mathbf{X} \mapsto \mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{U}^T\mathbf{X}$, s.t. $\frac{1}{s}\mathbf{X}\mathbf{X}^T = \mathbf{I}$

Then LS problem $\Theta \to_{min} \frac{1}{2s}\|\mathbf{Y} - \mathbf{\Theta}\mathbf{X}\|_F^2$ can be written:

$\Theta \to_{min} \frac{1}{2}\|\mathbf{\Theta} - \mathbf{\Gamma}\|_F^2$, $\mathbf{\Gamma} := \frac{1}{s}\mathbf{Y}\mathbf{X}^T$

Let $\mathbf{\Theta} = \mathbf{D}\mathbf{C}$, then: $\frac{\delta l}{\delta\mathbf{C}} = \mathbf{D}^T(\mathbf{\Theta}-\mathbf{\Gamma})$, $\frac{\delta l}{\delta\mathbf{D}} = (\mathbf{\Theta}-\mathbf{\Gamma})\mathbf{C}^T$

Now re-write: $\mathbf{DC} - \mathbf{\Gamma} = \mathbf{U}(\tilde{\mathbf{D}}\tilde{\mathbf{C}} - \Sigma)\mathbf{V}^T$, $\tilde{\mathbf{D}} = \mathbf{U}^T\mathbf{D}, \tilde{\mathbf{C}} = \mathbf{C}\mathbf{V}$

Then: $\frac{\delta l}{\delta\tilde{\mathbf{C}}} = (\tilde{\mathbf{D}}\tilde{\mathbf{C}} - \Sigma)^{\mathbf{T}}\tilde{\mathbf{D}}$, $\frac{\delta l}{\delta\tilde{\mathbf{D}}} = \tilde{\mathbf{C}}(\tilde{\mathbf{D}}\tilde{\mathbf{C}} - \Sigma)\mathbf{C}^T$

Let $\mathbf{d}_r$ be rows of $\tilde{\mathbf{D}}$ and $\mathbf{c}_r$ be columns of $\tilde{\mathbf{C}}$. Then can minimize: $\bar{l}(\mathbf{C},\mathbf{D}) = \frac{1}{2}\sum_r(\mathbf{d}_r\cdot\mathbf{c}_r - \sigma_r)^2 + \frac{1}{2}\sum_{r\neq q}(\mathbf{d}_r\cdot\mathbf{c}_q)^2$ (cooperative and competitive terms, second orthogonalizes)

Sigmoid Networks

Ridge function if can be written as a composition of an affine function: $f = \phi\circ\iota$, $f(\mathbf{x};\theta) = \phi(\mathbf{x}\cdot\theta)$, preserves level sets and directional sensitivity

But not rate of change: $\|\nabla_{\mathbf{x}} f(\mathbf{x};\theta)\| = |\phi'(\mathbf{x}\cdot\theta)|\cdot\|\theta\|$

Threshold units like Heavyside and sign gives no derivative info

$\Rightarrow$ **Logistic unit**: $\sigma(z) = \frac{1}{1+\exp[-z]}$, $\sigma^{-1}(t) = \log\frac{t}{1-t}$ (log-odds)

$\sigma'(z) = \sigma(z)(1-\sigma(z)) = \sigma(z)\sigma(-z)$, smooth since polynomials in σ

Hyperbolic tangent: $\tanh(z) := \frac{e^z - e^{-z}}{e^z + e^{-z}} = 2\sigma(2z) - 1$

$\tanh'(z) = 1 - \tanh^2(z)$

Sometimes preferred since range $(-1,1)$ symmetric around 0

Logistic regression

Cross-entropy loss $(y \in \{-1,+1\}) : l(\mathbf{x},y;\theta) = -\log\sigma(y\mathbf{x}\cdot\theta)$

$\Rightarrow \nabla_\theta l(\mathbf{x},y) = -\sigma(-y\mathbf{x}\cdot\theta)y\mathbf{x}$

Cross-entropy loss $(y \in \{0,1\}) :$

$l(\mathbf{x},y;\theta) = -y\log\sigma(\mathbf{x}\cdot\theta) - (1-y)\log(1-\sigma(\mathbf{x}\cdot\theta))$

Softmax

$\sigma_i^{\max}(\mathbf{x};\mathbf{\Theta}) = \frac{\exp[\mathbf{x}\cdot\theta_i]}{\sum_{j=1}^k\exp[\mathbf{x}\cdot\theta_j]}$, $\mathbf{\Theta} = [\theta_1,...,\theta_k]$

Over-parameterized (can add/subtract any constant vector)

Equal to sigmoid unit for $k = 2$, $\theta = \theta_1 - \theta_2$

$\nabla_{x_j}\sigma_i^{\max} = \begin{cases}\sigma_i^{\max}(1-\sigma_i^{\max}), & i = j\\ -\sigma_i^{\max}\sigma_j^{\max}, & i \neq j\end{cases}$

Softmax regression: $l(\mathbf{x},\mathbf{y};\mathbf{\Theta}) = -\mathbf{y}\cdot\log\sigma^{\max}(\mathbf{x};\Theta), \mathbf{y} \in \{\mathbf{e}_1,...,\mathbf{e}_k\}$

Alternatively: $l(\mathbf{x},\mathbf{y};\mathbf{\Theta}) = -\sum_{i=1}^k[y_i\mathbf{x}\cdot\theta_i + \log(\sum_{j=1}^k\exp\{\mathbf{x}\cdot\theta_j\})]$

$\Rightarrow \nabla_{\theta_i} l(\mathbf{x},\mathbf{y};\mathbf{\Theta}) = (\sigma_i^{\max} - y_i)\mathbf{x}$

Approximation Theory

Definitions and notation

$f \simeq \mathcal{G} \iff \text{approx-err}(f,\mathcal{G}) = \inf\{g \in \mathcal{G} \mid \|f-g\|_\infty\} = 0$

Uniform convergence: $(g_m) \overset{\infty}{\to} f \Leftrightarrow \forall\epsilon > 0 : \exists m \geq 1 : \|g_m - f\|_\infty < \epsilon$

Follows that: $\mathcal{G} \ni g_m \overset{\infty}{\to} f \Rightarrow f \simeq \mathcal{G}$

Denseness: $\mathcal{G} \subseteq \mathcal{F}$ dense in $\mathcal{F} \iff \mathcal{F} \simeq \mathcal{G} \iff \forall f \in \mathcal{F}, f \simeq \mathcal{G}$

Closure is all functions that can be approximated by $\mathcal{G}$: $\mathrm{cl}(\mathcal{G})$

$\mathcal{G}$ **universal approximator** $\iff C(S) \simeq \mathcal{G}(S) \forall$ compact $S \subset \mathbb{R}^n$ (compact = closed and bounded)

Weierstrass Theorem

Polynomials $\mathcal{P}$ are dense in $C([a,b]) \forall a,b \in \mathbb{R}$

Universal approximation theorem (1d)

Let $\sigma \in C^\infty(\mathbb{R})$ smooth & not polynomial,

$\mathcal{G}_\sigma^1 = \{g : g(x) = \sigma(ax+b) \quad a,b \in \mathbb{R}\}, H_\sigma^1 = \mathrm{span}(\mathcal{G}_\sigma^1)$,

then H_σ^1 is a universal approximator (can't pick polynomial because then we form polynomials of limited degree)

Ridge function theorem

$\mathcal{G}_\sigma^n = \{g : g(x) = \sigma(x\cdot\theta)\ \theta \in \mathbb{R}^n\}, \mathcal{G}^n = \bigcup_{\sigma\in C(\mathbb{R})}\mathcal{G}_\sigma^n, H_\sigma^n = \mathrm{span}(\mathcal{G}^n)$

Then H^n is a universal function approximator

(Problem: here we can pick any combination of ridge functions)

Dimension lifting theorem

H_σ^1 univ. approx for $C(\mathbb{R}) \Rightarrow H_\sigma^n$ univ. approx for $C(\mathbb{R}^n) \quad \forall n \geq 1$

Barron's theorem (number of units required)

Gradient regularity condition: $C_g = \int\|\omega\|\cdot|\hat{g}(\omega)|d\omega < \infty$

If g differentiable, then $\hat{\nabla g}(\omega) = \omega\cdot\hat{g}(\omega)$

Theorem: Let σ be bounded, monotonic s.t. $\lim_{t\to\infty}\sigma(t) = 1$ & $\lim_{t\to-\infty}\sigma(t) = 0$. Let $g : \mathbb{R}^n \to \mathbb{R}$ with $C_g < \infty$, & $r > 0$. Then $\exists(f_m(x))_{m=1}^\infty$ sequence defined as $f_m(x) = \sum_{j=1}^m\left(\beta_j\sigma(\theta_j + b_j) + b_0\right)$, s.t.

$\int_{r_{\mathbb{B}}}(g(x) - f_m(x))^2\mu(dx) \leq \mathcal{O}(\frac{1}{m})$,

$r_{\mathbb{B}} = \{x \in \mathbb{R}^n : \|x\| \leq r\}, \mu$ probability measure. Independent of n!

$\Rightarrow$ no curse of dimensionality when approx. certain functions

Benefits of depth

Function $g(\mathbf{x}) = \psi(\|\mathbf{x}\|)$ has exponential advantages in approximating with 2 layers vs 1:

Upper bound on how much space covered by 1 layer: me^{-n}

Backpropagation

Notation

Network $F = F_{k:1} = F_k \circ F_{k-1} \circ ... \circ F_1$

Width of a layer: $\mathrm{width}_l = n_l := \dim(\mathrm{range}(F_l))$

Using activations: $\mathbf{z}_l := F_{l:1}(\mathbf{x}) = (F_l \circ F_{l-1:1})(\mathbf{x}) = F_l(\mathbf{z}_{l-1})$

Jacobian map for $F : \mathbb{R}^n \to \mathbb{R}^m : \partial F = (\partial_{ij}F) = (\partial_i F_j) : \mathbb{R}^n \to \mathbb{R}^{m\times n}$

Chain rule for maps: $\underbrace{\partial(G\circ F)}_{\mathbb{R}^n\to\mathbb{R}^{m\times n}} = \underbrace{(\partial G\circ F)\cdot\partial F}_{\mathbb{R}^n\to(\mathbb{R}^{m\times k}\cdot\mathbb{R}^{k\times n})}$

The Jacobian is $\partial F = \prod_{l=k}^1 \partial F_l \circ F_{l-1:1}$, $\partial F(\mathbf{x}) = \prod_{l=k}^1 \partial F_l(\mathbf{z}_{l-1})$

With loss function: $f = l \circ F$

Gradient information provides direction of steepest descent:

$\lim_{\eta\to 0}\arg\min_{\boldsymbol{\delta}:\|\boldsymbol{\delta}\|=1} f(\mathbf{x};\boldsymbol{\theta} + \eta\boldsymbol{\delta}) = -\frac{\nabla_{\boldsymbol{\theta}} f(\mathbf{x};\boldsymbol{\theta})}{\|\nabla_{\boldsymbol{\theta}} f(\mathbf{x};\boldsymbol{\theta})\|}$

Derivative wrt parameter in layer: $h' \circ g$

Derivative wrt parameter in next layer: $(\partial h \circ g)\cdot g'$

Backpropagation

$\nabla_{\boldsymbol{\theta}_l} F = (\partial F_{k:l+1} \circ F_{l:1})\cdot(F_l' \circ F_{l-1:1})$

Or with activity vectors: $\nabla_{\boldsymbol{\theta}_l} F(\mathbf{x}) = \partial F_{k:l+1}(\mathbf{z}_l)\cdot F_l'(\mathbf{z}_{l-1})$

(downstream Jacobian $\times$ local Jacobian). $F = G \circ F_{\boldsymbol{\theta}} \circ H$, apply chain rule twice $(\partial G \circ F_{\boldsymbol{\theta}} \circ H)\cdot(F_{\boldsymbol{\theta}} \circ H)' = (\partial G \circ F_{\boldsymbol{\theta}} \circ H)\cdot(F_{\boldsymbol{\theta}}' \circ H)$.

∂l is a Jacobi vector. Get computational savings by multiplying it with Jacobi matrices in reverse order (this is exactly backprop!)

$\nabla_{\boldsymbol{\theta}_l} f(\mathbf{x}) = \partial(l \circ F_{k:l+1})(\mathbf{z}_l)\cdot F_l'(\mathbf{z}_{l-1}) =: \boldsymbol{\xi}_{l+1}\cdot F_l'(\mathbf{z}_{l-1})$

Row vectors $\boldsymbol{\xi}_l$ can be calculated through backward iteration: $\boldsymbol{\xi}_{k+1} = \partial l(\mathbf{z}_k)$, $\boldsymbol{\xi}_l = \boldsymbol{\xi}_{l+1}\partial F_l(\mathbf{z}_{l-1})$

Backpropagation algorithm can be written in three steps: (1) forward pass computing $\mathbf{z}_l's$, (2) backward pass computing $\boldsymbol{\xi}_l's$ and (3) local computations computing $\nabla_{\boldsymbol{\theta}_l} f(\mathbf{x})'s$

Automatic differentiation

Reverse mode (backpropagation) and **forward** mode

Forward mode: For each parameter, compute the derivatives for each of the (intermediate) outputs starting from the parameter (no forward pass here!)

For $f : \mathbb{R}^N \to \mathbb{R}^M$:

Reverse mode more efficient if $M < N$ (scales in $M, \mathcal{O}((V+E)M)$)

Forward mode more efficient if $M > N$ (scales in $N, \mathcal{O}(N(V+E))$)

For DL: $N =$ #parameters and $M = 1$ (scalar loss)

Reverse mode requires in general more memory due to storing results from forward pass

Derivatives

Element-wise function (e.g. activation fun): $\frac{\partial}{\partial\mathbf{x}} f(\mathbf{x}) = \mathrm{diag}(f'(\mathbf{x}))$

ReLU wrt pre-activations: $\frac{\partial}{\partial\mathbf{x}}\max(0,\mathbf{x}) = \mathrm{diag}(H(\mathbf{x}))$

ReLU wrt params: $\frac{\partial}{\partial\mathbf{W}_{ij}}\max(0,\mathbf{W}\mathbf{x})_k = \max(0,\mathrm{sign}(\mathbf{W}_i{}^T\mathbf{x}))x_j\delta_{ik}$

ReLU wrt activations: $\frac{\partial}{\partial\mathbf{x}_j}\max(0,\mathbf{W}\mathbf{x})_i = \max(0,\mathrm{sign}(\mathbf{W}_i^T\mathbf{x}))\cdot W_{ij}$

Rectified Networks

Sigmoids suffer from vanishing gradients / unreliable gradient information in deep networks: $\sigma'(z) = \sigma(z)\sigma(-z) \overset{z\to\pm\infty}{\to} 0$

Rectified units: continuous piecewise linear

Additional benefit of rectified units: computationally faster

Rectified Linear Unit (ReLU)

$(\mathbf{x},\boldsymbol{\theta}) \mapsto (\mathbf{x}\cdot\boldsymbol{\theta})_+ = \max\{0,\mathbf{x}\cdot\boldsymbol{\theta}\}$

Splits the input space in two half-spaces separated by hyper-

plane $\mathcal{H}^0_{\boldsymbol{\theta}}$: $\mathcal{H}^+_{\boldsymbol{\theta}} = \{\mathbf{x} : \mathbf{x}\cdot\boldsymbol{\theta} > 0\}$, $\mathcal{H}^-_{\boldsymbol{\theta}} = \{\mathbf{x} : \mathbf{x}\cdot\boldsymbol{\theta} < 0\}$

Subderivative: $\partial(z)_+ = \begin{cases} 1 & z > 0 \\ [0,1] & z = 0 \\ 0 & \text{else} \end{cases}$

Definition of subderivative of convex f at z_0:
$\partial f(z_0) = \{c : f(z) - f(z_0) \geq c(z - z_0)\}$

Activation patterns: units can be either active $\mathbf{x} \in \mathcal{H}^+_{\boldsymbol{\theta}}$ or inactive $\mathbf{x} \in \mathcal{H}^-_{\boldsymbol{\theta}}$, with patterns $H(\boldsymbol{\Theta}\mathbf{x}) \in \{0,1\}^m$

$|\{H(\boldsymbol{\Theta}\mathbf{x}) : \mathbf{x} \in \mathbb{R}^n\}| \leq 2^m$, but is less since not every boolean vector is a valid activation pattern (because not every activation pattern might actually have an input that produces it)

Vanishing gradient: For sigmoids the sensitivity (gradient wrt activation) can go to 0, which means that slight changes in parameters do not lead to any signal. With ReLUs: $\partial(z)_+ = 1 \quad \forall z > 0$

Backprop: If unit z_{lj} inactive then $\nabla_{\boldsymbol{\theta}_{lj}} z_{lj} = \nabla_{z_{l-1}} z_{lj} = \mathbf{0}$

Parameter gradients vanish and Jacobi matrix is sparse:

$$\partial F_l := \tilde{\boldsymbol{\theta}}_l := \begin{bmatrix} \tilde{\boldsymbol{\theta}}_{l1}^T \\ \dots \\ \tilde{\boldsymbol{\theta}}_{lm}^T \end{bmatrix}, \quad \tilde{\boldsymbol{\theta}}_{lj}^T = \begin{cases} \mathbf{0} & z_{lj} = 0 \\ \boldsymbol{\theta}_{l1}^T & \text{otherwise} \end{cases}$$

Dying ReLUs: If a specific unit is inactive for all inputs (can happen on initialization or during training) $\Rightarrow$ parameters will not be updated. Could prune or re-initialize

Absolute Value Unit (AbsU): $|z|$, $\partial|z| = \begin{cases} 1 & z > 0 \\ [-1,1] & z = 0 \\ -1 & z < 0 \end{cases}$

Relation to ReLU: $(z)_+ = \frac{z+|z|}{2}$ and $|z| = 2(z)_+ - z = (z)_+ + (-z)_+$

No sparseness property as in ReLUs, but symmetric

Smooth ReLU approximations: Combine rectification and smoothness

Softplus: $(\mathbf{x},\boldsymbol{\theta}) \mapsto \log(1 + \exp[\mathbf{x}\cdot\boldsymbol{\theta}]) \in (0,\infty)$

Exponential linear unit: $(\mathbf{x},\boldsymbol{\theta}) \mapsto \begin{cases} \mathbf{x}\cdot\boldsymbol{\theta} & \mathbf{x}\cdot\boldsymbol{\theta} \geq 0 \\ \exp[\mathbf{x}\cdot\boldsymbol{\theta}] - 1 & \text{else} \end{cases} \in (-1,\infty)$

Leaky ReLU: Gives some gradient information even in $\mathcal{H}^-_{\boldsymbol{\theta}}$ (low sensitivity instead of no sensitivity, typical $\epsilon = 0.01$)

$(\mathbf{x},\boldsymbol{\theta}) \mapsto \begin{cases} \mathbf{x}\cdot\boldsymbol{\theta} & \mathbf{x}\cdot\boldsymbol{\theta} \geq 0 \\ \epsilon\mathbf{x}\cdot\boldsymbol{\theta} & \text{else} \end{cases} \in \mathbb{R}$

Universal function approximators

Theorem: Piecewise linear functions are dense in $C([0,1])$

Theorem: A piecewise linear function with m pieces can be written as $g(x) = ax + b + \sum_{i=1}^{m-1} c_i(x - x_i)_+$ (sum of ReLU units)

(alternative representation exists with absolute value function)

Corollary: Networks with one hidden layer of ReLU or AbsU are universal function approximators

Minimal non-linearity

k-**Hinge functions**: $g(\mathbf{x}) = \max_{j=1}^k \{\boldsymbol{\theta}_j \cdot \mathbf{x} + b_j\}$ (aka maxout units)

Representational power: $2\max\{f,g\} = f + g + |f - g|$

Theorem: Every continuous piecewise linear function can be written as a signed sum of k-Hinges with $k \leq n+1$

Polyhedral functions: f is polyhedral $\Leftrightarrow$ epi(f) is polyhedral set S polyhedral $\Leftrightarrow$ S is finite intersection of closed half-spaces, i.e. $S = \{\mathbf{x} \in \mathbb{R}^n : \boldsymbol{\theta}_j \cdot \mathbf{x} + b_j \geq 0, j = 1,\dots,r\}$

Theorem: If f polyhedral then $\exists \mathcal{A} \subset \mathbb{R}^{n+1}, |\mathcal{A}| = k$ s.t. $f(\mathbf{x}) = \max_{(\boldsymbol{\theta},b)\in\mathcal{A}} \{\boldsymbol{\theta}\cdot\mathbf{x} + b\}$

Theorem: Every continuous piecewise linear function f can be written as the difference of two polyhedral functions

$\Rightarrow$ **Theorem**: Maxout networks with two maxout units (difference of two k-Hinges) are universal function approximators

Optimization

Losses

Squared loss: $l_{\mathbf{y}}(\boldsymbol{\nu}) = \frac{1}{2}\|\mathbf{y} - \boldsymbol{\nu}\|^2$

Zero-one loss: $l_y(v) = \begin{cases} 0, & v = y \\ 1, & \text{else} \end{cases}$

Log-loss (multiclass): $l_y(v) = -\log v_y$

Soft target cross-entropy ($\mathbf{y} \in [0,1]^m$):
$l_{\mathbf{y}}(\boldsymbol{\nu}) = -\sum_{j=1}^m y_j \log v_j \geq -\sum_{j=1}^m y_j \log y_j =: H(\mathbf{y})$

Probabilistic loss: $l_{\mathbf{y}}(\boldsymbol{\nu}) = -\log p(\mathbf{y};\boldsymbol{\nu})$ (e.g. replace with isotropic Gaussian to get squared loss)

Exponential family: $p(\mathbf{y};\boldsymbol{\nu}) = h(\mathbf{y})\exp[\mathbf{y}\cdot\boldsymbol{\nu} - \psi(\boldsymbol{\nu})]$, log partition/normalizing function ψ

Can construct losses by taking distributions in the exponential family in the log prob loss and replace $\boldsymbol{\nu}$ with $h(\boldsymbol{\theta}\cdot\mathbf{z})$, where $\mathbf{z}$ is produced by a Neural Network

Gradient Descent and Optimization Theory

$\mathbf{x}^{k+1} = \mathbf{x}^k - \eta\nabla f(\mathbf{x}^k)$

Solution to $\dot{\mathbf{x}} = -\nabla f(\mathbf{x})$ is called gradient flow - approximated by GD

Notes on η: Not typical to optimize by line search for neural networks since expensive to evaluate. Typically keep constant or use learning rate schedule. Selecting η is a trade-off between computational speed (larger better) and convergence & approximation of gradient flow (smaller better)

Quadratic Model: 2nd order Taylor approximation yields
$f(\mathbf{x} + \Delta\mathbf{x}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})\cdot\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T\nabla^2 f(\mathbf{x})\Delta\mathbf{x}$

$\Rightarrow$ minimizer is $\Delta\mathbf{x} = -[\nabla^2 f(\mathbf{x})]^{-1}\nabla f(\mathbf{x})$ (Newton's method)

Setting $\nabla^2 f(\mathbf{x}) = \frac{1}{\eta}\mathrm{I}$ yields gradient descent (η curvature of quadratic and step size, smaller η gives more curvature and hence smaller step)

Convex: $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y), \quad \forall\lambda \in [0,1]$

FOC of convexity: $f(x) \geq f(y) + \nabla f(y)^T(x - y)$

Lipschitz smoothness: $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$

$\Leftrightarrow f(x) \leq f(y) + \nabla f(y)^T(x - y) + \frac{L}{2}\|x - y\|^2$

Strong convexity: $f(x) \geq f(y) + \nabla f(y)^T(x - y) + \frac{\mu}{2}\|x - y\|^2$

Hessian of smooth and strongly convex function: $\mu\mathrm{I} \preccurlyeq \nabla^2 f \preccurlyeq L\mathrm{I}$

ϵ-**Stationarity**: $\|\nabla f(x)\| \leq \epsilon$

(to measure local convergence)

PL condition (generalization of strong convexity without convexity):

$\frac{1}{2}\|\nabla f(x)\|^2 \geq \mu(f(x) - f^*), \quad \forall x, f^* = \min f(x)$

SGD

Assume additive structure $f(x) = \sum_{i=1}^n f_i(x)$

$$x^{k+1} = x^k - \eta_k\nabla f_{I(k)}(x^k), \quad I(k) \sim \mathrm{Unif}(1,\dots,n)$$

Unbiased, $\mathbb{E}[\nabla f_I(x)] = \nabla f(x)$.

$\mathbf{V}(x) = \frac{1}{n}\sum_{i=0}^n \|\nabla f(x) - \nabla f_i(x)\|^2$

Polyak Averages: To combat variance around x^*

$\bar{x}^{k+1} = \frac{k}{k+1}\bar{x}^k + \frac{1}{k+1}x^{k+1}$

Minibatch SGD Sample r functions f_i. Unbiased and reduces variance αr. Smaller r implies more noise, yet better results in non-convex setting. Noise might help avoid getting stuck in bad regions. Batch size depending on concurrency model of GPU, has to fit in GPU memory.

Learning rate For theoretical results typically $\eta_k \propto \frac{1}{k}$, since $\sum^\infty \frac{1}{k} = \infty$ (min requirement, to ensure we can approach end point) and $\sum^\infty \frac{1}{k^2} \leq \infty$. However in practice: keep step size constant or reduce step size at a small number of points.

Theorem Assume $f = \sum_i f_i$, each f_i L_i-smooth, $\sup_i L_i \leq L$ and f μ-strongly convex. x^* minimizer of f and $\sigma^2 := V(x^*) = \frac{1}{n}\sum\|\nabla f_i(x)\|$. x^k SGD iterate generated with $\eta \leq 1/\mu$. Then

$$\mathbb{E}\|x^k - x^*\|^2 \leq A^k\|x^0 - x^*\|^2 + B,$$

where $A = 1 - 2\eta\mu(1 - \eta L)$, $B = \frac{\eta\sigma^2}{\mu - \eta L}$, the bigger the step size, the more stochasticity. Advantage over SGD: can return a set of visited parameters.

Variance reduction: $x^{k+1} = x^k - \eta[\nabla f_i(x^k) - \nabla f_i(\bar{x}) + \nabla f(\bar{x})]$

Gradient clipping: To avoid exploding gradients, can bound the norm of the gradient in the update step by a threshold (e.g. when reaching a "cliff")

Momentum

If gradients start going into a particular direction, keep going in that same direction (inertia in direction we had)

Nesterov's Acceleration Method

$$y_{k+1} = x_k + \beta(x_k - x_{k-1})$$
$$x_{k+1} = y_{k+1} - \eta\nabla f(y_{k+1})$$

Theorem Let f L-smooth and μ-strongly convex, $\kappa = \frac{L}{\mu}$, $\beta = \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$. Then $f(x^k) - f(x^*) \leq L\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}}\right)^k\|x^0 - x^*\|^2$.

Polyak's Heavy ball method

$x^{k+1} = x^k - \eta\nabla f(x^k) + \beta(x^k - x^{k-1}) \quad \beta \in (0,1)$.

Adaptivity

Adapt learning rate per parameter or dimension. Advantage in compositional models: adapt step size for different parameters in different layers (non-uniformity of parameters).

AdaGrad Increasing sequence:

$\gamma^k = \gamma^{k-1} + \nabla f(x^k)\odot\nabla f(x^k)$.

γ^k will be large for parameters that have received large updates. Use these as pre-conditioner matrix

$x^{k+1} = x^k - \eta\Lambda^k\nabla f(x^*)$,

where $\Lambda^k = \mathrm{diag}(\lambda_i^k)$, with $\lambda_i^k = \frac{1}{\sqrt{\gamma_i^k + \delta}}$, $\delta > 0$

If something has received very significant updates in the past, then we decay the learning rate faster.

Adam Momentum + adaptivity

$\mathbf{m}^k = \beta\mathbf{m}^{k-1} + (1-\beta)\nabla f(x^k)$, $\beta \in [0,1]$, $\mathbf{m}^0 = 0$

$x^{k+1} = x^k - \frac{\eta}{1-\beta^k}\mathbf{m}^k$.

Since $\mathbf{m}^0 = 0$ estimate is biased, $\frac{1}{(1-\beta^k)}$ corrects for this.

$\gamma^k = \alpha\gamma^{k-1} + (1-\alpha)\left[\nabla f(x^k)\odot\nabla f(x^k)\right]$

$x^{k+1} = x^k - \frac{\eta}{1-\beta^k}\Lambda^k\mathbf{m}^k$

$\Lambda^k = \mathrm{diag}(\lambda_i^k)$, with $\frac{1}{\lambda_i^k} = \sqrt{\frac{\gamma_i^k}{1-\alpha^k}} + \delta$.

As AdaGrad, not parametrization invariant (matrix depends on basis of chosen parametrization). Typical values $\beta = 0.9$, $\alpha = 0.99$.

AMSGrad: enforce monotonic increase of γ to ensure convergence, $\hat{\gamma}^{k+1} = \max\{\hat{\gamma}^k, \gamma^{k+1}\}$

Compressed stochastic gradients

SignSGD: $x^{k+1} = x^k - \eta_k\mathrm{sign}\left(\nabla f_{I(k)}(x^k)\right)$. Reduces computational and communication complexity.

Convolutional Networks

Convolution Operator

Integral operators Kernel $H : \mathbb{R}^2 \to \mathbb{R}$, $\quad -\infty \leq t_1 \leq t_2 \leq \infty$

$(Tf)(u) = \int_{t_1}^{t_2} H(u,t)f(t)dt$

Fourier Transform

$t_1 = -\infty$, $\quad t_2 = \infty$, $\quad H(u,t) = \exp\{-2\pi itu\}$

$(\mathcal{F}f)(u) := \int_{-\infty}^{\infty}\exp\{-2\pi itu\}f(t)dt$

Convolution

$(f*h)(u) := \int_{-\infty}^{\infty} h(u-t)f(t)dt = \int_{-\infty}^{\infty} f(u-t)h(t)dt$

Convolutions are **commutative** (exchange function and kernel)

And **shift-equivariant**: $f_\Delta(t) := f(t+\Delta), \Rightarrow f_\Delta * h = (f*h)_\Delta$

Linear shift-equivariant transforms

$T(\alpha f + \beta g) = \alpha Tf + \beta Tg, \quad \forall f,g; \quad \forall\alpha,\beta \in \mathbb{R}$

$(Tf_\Delta)(t) = (Tf)(t+\Delta)$

Theorem: Any linear translation-equivariant (shift-equivariant) transformation T can be written as a convolution with some h

Discrete convolutions

$f, h : \mathbb{Z} \to \mathbb{R}$. Define discrete convolution via

$(f*h)[u] := \sum_{t=-\infty}^{\infty} f[t]h[u-t]$

Typical choice of h: support over finite window, e.g. $h(t) = 0$ for $t \notin [t_{\min}, t_{\min}]$. In higher dimensions, replace vectors by matrices or fields $(F*G)[i,j] = \sum_k\sum_l F[i-k, j-l]G[k,l]$

Cross-correlation

$(f\star h)[u] := \sum_{t=-\infty}^{\infty} f[t]h[u+t]$

$(f\star h) = (\overline{f}*h)$, $\quad \overline{f}[t] := f[-t]$

Toeplitz matrices: Constant on the diagonals (from exercise: parameters of a 1D convolutional layer can be written as a dense layer through a Toeplitz matrix)

Convolutional Neural Networks

Exploiting translation equivariance

Exploiting locality and scale (temporal, spatial,...)

Increased efficiency through parameter sharing

Receptive fields and sparse connectivity Activity of one unit doesn't depend on all units from previous layer. Convolved signal **inherits** topology of original signal. Can create longer range dependencies (larger receptive field) by nesting of convolutions

Border handling: through *same* padding (with zeros) to retain dimension or *valid* padding to only retain values from windows fully contained in support of signal

Half (same) padding (Assuming unit strides) For any i and k odd $(k = 2n+1, \quad n \in \mathbb{N})$, $s = 1$ and $p = \lfloor\frac{k}{2}\rfloor = n$

$o = i + 2\lfloor\frac{k}{2}\rfloor - (k-1) = i$.

Full padding In this setting every possible partial or complete superimposition of the kernel on the input feature map is taken into account.

For any i, $\quad k$ and for $s = 1$ and $p = k - 1$

$o = i + k - 1$

Backpropagation

Exploit structural sparseness in computing $\frac{\partial x_i^l}{\partial x_j^{l-1}}$

Receptive field of $x_i^l : \mathcal{I}_i^l := \{j : w_{ij}^l \neq 0\}$, $\mathbf{W}^l$ Toeplitz matrix of the convolution. $\frac{\partial x_i^l}{\partial x_j^{l-1}} = 0$, for $j \notin \mathcal{I}_i^l$.

Weight sharing $\frac{\partial\mathcal{R}}{\partial h_j^l} = \sum_i \frac{\partial\mathcal{R}}{\partial x_i^l}\frac{\partial x_i^l}{\partial h_j^l}$, h_j^l kernel weight. Weights are reused for every unit within target layer.

Backprop example: Let $(\mathbf{x}*\mathbf{w})_{ij} = \sum_{k=1}^q\sum_{l=1}^q x_{i+q-k,j+q-l}w_{k,l}$, $1 \leq i,j \leq d-q+1$. Let $r = d-q+1$, then $(\mathbf{x}*\mathbf{w}) \in \mathbb{R}^{r\times r}$. Define: $f(\mathbf{x}) = \mathbf{v}^T\mathrm{vec}(\sigma(\mathbf{x}*\mathbf{w}))$. Then:

$\frac{\partial f(\mathbf{x})}{w_{ab}} = \sum_{k,l=1}^r \mathrm{mat}(\mathbf{v})_{kl}\sigma'((\mathbf{x}*\mathbf{w})_{kl})x_{k+q-a,l+q-b} = \dots =$

$= (\mathrm{rot}_\pi(\mathbf{x}) * (\mathrm{mat}(\mathbf{v})\odot\sigma'(\mathbf{x}*\mathbf{w})))_{ab}$,

where $\mathrm{rot}_\pi(\mathbf{A})$ flips rows and columns

Stages

Non-linearities

Pooling

Sub-sampling (strides)

Pooling 1D: $x_i^{\max} = \max\{x_{i+k} : 0 \leq k < r\}$

Pooling 2D: $x_{ij}^{\max} = \max\{x_{i+k,j+l} : 0 \leq k < r, 0 \leq l < r\}$

Stacking convolutional layers ($\mathbf{X}$, $\mathbf{Y}$ 3rd order tensors):

$y[r][s,t] = \sum_u\sum_{\Delta s,\Delta t} w[r,u][\Delta s,\Delta t]x[u][s+\Delta s, t+\Delta t]$

Convolution arithmetic - output dimensions

Dimension-wise: For any input size i, kernel size k, padding p and strides s,

$o = \left\lfloor\frac{i + 2p - k}{s}\right\rfloor + 1$

Convolution arithmetic - receptive field

Dimension-wise recursion formula: For receptive field r_l, stride s_l and kernel size k_l

$r_{l-1} = s_l \cdot r_l + (k_l - s_l), \quad r_L = 1$

Deep Gradients

Tricks and approaches to improve learning in deep networks

Short Connectivity

Motivation: Avoid vanishing gradients by using activations from previous layers until adjacent layer learns its weights
Use if get worse performance on train set by going deeper
Residual layers: $g(\mathbf{x}) = \mathbf{x} + f(\mathbf{x};\boldsymbol{\theta})$
If weights are initialized around zero, the gradient might not be able to propagate through properly. Now Jacobian at initialization ($f \approx 0$) will be: $\mathbf{J}_g = \mathbf{I} + \mathbf{J}_f \approx \mathbf{I}$, which is a better start since the gradient will flow through with identity map
If dimensions do not match: $g(\mathbf{x}) = \mathbf{W}\mathbf{x} + f(\mathbf{x};\boldsymbol{\theta}), \quad \mathbf{J}_g = \mathbf{W} + \mathbf{J}_f \approx \mathbf{W}$ at initialization $f \approx 0$
Can skip an arbitrary number of layers
Empirically see that ResNets allows the model to better take advantage of depth. Common in computer vision.
Note: They improve training, but do not increase representational power!

Dense Connectivity

Connect layer output to all downstream layers ($\Rightarrow$ get more channels in CNNs). Concatenate activations instead of adding as in ResNets
Residual connections: shortcut layers and *add* back in
Skip connections: shortcut layers and *concatenate* back in

Normalization

No agreement on what they do and how they help, just *that* they help
Observation: Poorly calibrated dynamic range of activities (different variances) $\Rightarrow$ poor backpropagation of errors (vanishing gradients)
Batch Normalization
Motivation: have strong dependencies between weights in layers, want to find a suitable learning rate to get similar scale
Broadly used in vision
Dependendence on batch size not suitable for all architectures (e.g. RNNs)
Normalize layer l activations for a batch $\mathrm{I} \subseteq [1:s]$: $\mu^l := \frac{1}{|\mathrm{I}|}\sum_{t\in\mathrm{I}} \mathbf{z}^l[t]$
$\sigma_i^l := \sqrt{\delta + \frac{1}{|\mathrm{I}|}\sum_{t\in\mathrm{I}}(z_i^l[t]-\mu_i)^2}$
$\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are functions of the weights: can be differentiated
Normalize (z-score): $\tilde{z}_i^l := \frac{z_i^l - \mu_i^l}{\sigma_i^l}$
Regain representational power / expressivity: $\hat{z}_i^l = \alpha_i^l \tilde{z}_i^l + \beta_i^l$
Ideally would do BN over whole dataset, but cannot for computational reasons
Weight Normalization
Normalize by the inner product of the weights instead of sd of activations
Easier to use in inference / test mode than BN
Permits distributed optimization as the normalization doesn't include dependencies on batch samples
Layer Normalization
Used in NLP
Fix data point but use population average in layer as reference:
$\mu^l[t] := \frac{1}{m^l}\sum_{i=1}^{m^l} z_i^l[t]$
$\sigma^l[t] := \sqrt{\delta + \frac{1}{m^l}\sum_{i=1}^{m^l}(z_i^l[t]-\mu^l[t])^2}$
Rest same as BN
BN, LN and WN are scale-invariant

Regularization

Intended to lower generalization error but not the training error

Norm-based

Regularized objective $\mathcal{R}_\Omega(\boldsymbol{\theta},\mathcal{S}) = \mathcal{R}(\boldsymbol{\theta},\mathcal{S}) + \Omega(\boldsymbol{\theta})$
L_2/Frobenius norm for deep networks
$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\sum_{l=1}^{L} \mu^l \|\boldsymbol{\Theta}^l\|_F^2, \quad \mu^l \geq 0$

Weight decay (a.k.a L_2 regularization)

$\nabla\Omega = \mu\boldsymbol{\theta}$ or $\frac{\partial\Omega}{\partial\theta_{ij}^l} = \mu\theta_{ij}^l$
Gradient descent update: $\boldsymbol{\theta}_{k+1} = (1-\eta\mu)\boldsymbol{\theta}_k - \eta\nabla\mathcal{R}(\boldsymbol{\theta}_k)$
favors weights of small magnitude (shrinkage). Network behavior won't change much, avoiding learning local noise from data. $\boldsymbol{\theta}$ still moves in the direction of the gradients, but also shrinks (if η is small enough s.t. $1-\eta > 0$). Forces network to learn only features which are seen often across the training set.
$\Rightarrow$ shrink weights where gradient vanishes!
Quadratic Taylor approximation of $\mathcal{R}$ around optimal $\boldsymbol{\theta}^*$ + first order optimality condition of $\mathcal{R}_\Omega$: $\boldsymbol{\theta} = \mathbf{Q}(\boldsymbol{\Lambda}+\mu\mathbf{I})^{-1}\boldsymbol{\Lambda}\mathbf{Q}^\intercal\boldsymbol{\theta}^*$, $\mathbf{H} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\intercal$,
$\lambda_i \gg \mu$: vanishing effect $\Rightarrow$ very small regularization and $\theta_i \approx \theta_i^*$
$\lambda_i \ll \mu$: shrinking effect $\Rightarrow$ large regularization and $\theta_i \approx 0$
Take-away: Small regularization in directions of large eigenvalues, which correspond to directions with a lot of curvature (steep), and vice versa
Ridge: Linear regression + L_2-regularization = ridge regression
$\boldsymbol{\theta} = (\mathbf{X}^\intercal\mathbf{X} + \mu\mathrm{I})^{-1}\mathbf{X}^\intercal\mathbf{y}$
Constrained optimization view: solve $\min_{\{\boldsymbol{\theta}:\|\boldsymbol{\theta}\|\leq r\}} \mathcal{R}(\boldsymbol{\theta})$, with projected GD
$\boldsymbol{\theta}(k+1) = \Pi_r[\boldsymbol{\theta}(k) - \eta\nabla\mathcal{R}(\boldsymbol{\theta}(k))], \quad \Pi_r(\mathbf{v}) := \min\{1, \frac{r}{\|\mathbf{v}\|}\}\mathbf{v}$
Benefit: Constraints do not affect initial learning where weights are small, only become active once weights are large

Early stopping

Stop learning after small number of iterations. Rely on validation data. If we can choose k, η s.t.
$(\mathbf{I}-\eta\boldsymbol{\Lambda})^k \stackrel{!}{=} \mu(\boldsymbol{\Lambda}+\mu\mathbf{I})^{-1}$, which for $\eta\lambda_i \ll 1$, $\lambda_i \ll \mu$ can be achieved in $k = \frac{1}{\eta\mu}$ steps. This indicates that a stronger regularization corresponds to stopping training earlier and vice versa.

Ensemble methods

Bagging: Bootstrap samples $\mathcal{S}_r^k$: sample r times from $\mathcal{S}$ with replacement (on average 2/3 distinct examples). Train model on each sample, then average model output probs $p(\mathbf{y}\,|\,\mathbf{x};\theta^{\mathbf{k}})$: $p(\mathbf{y}\,|\,\mathbf{x}) = \frac{1}{K}\sum_{k=1}^{K} p(\mathbf{y}\,|\,\mathbf{x};\theta^k)$
Almost always beneficial but expensive (not often used for NNs)
Knowledge Distillation - best of both worlds
Idea: Transfer knowledge from a complex [ensemble] model (source) into a simpler one (target). Useful if complex model is expensive to evaluate $\Rightarrow$ can deploy on less powerful hardware
Train a simple model to learn the soft outputs of a trained complex model, using a high value of the temperature parameter in softmax

Dropout

Idea: Randomly drop subsets of units for better robustness
Keep probability π_i^l for unit i in layer l. All models share same weights
Realizes an ensemble: $p(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{Z}} p(\mathbf{Z})p(\mathbf{y}|\mathbf{x},\mathbf{Z})$, $\mathbf{Z}$ zeroing mask
To predict, can sample and average. But to avoid sampling blow-up can use heuristic:
Weight Rescaling to avoid sampling 10-20 times, set $\tilde{\theta}_{ij}^l \leftarrow \pi_j^{l-1}\theta_{ij}^l$
Can be shown to be a (sometimes exact) approximation to a geometrically averaged ensemble

Data Augmentation

Powerful trick to generate larger training set & improve robustness
Generate virtual examples by applying transformations τ to each training example $(\mathbf{x},\mathbf{y}) \mapsto (\tau(\mathbf{x}),\mathbf{y})$. PCA, cropping resizing, rotations reflections, etc.
Can inject noise to inputs, weights (regularizing effect) and targets (create soft targets as labels might be noisy)

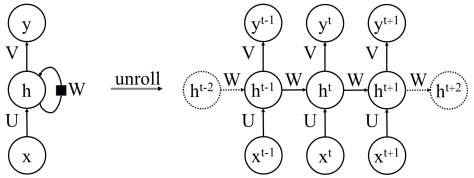
Task Augmentation

Pre-training: Pre-train parts of model on more generic task, where training data is cheaper. E.g. pre-trained word embeddings or image feature maps. After this: fine-tune
Semi-supervised learning: when not all data is labeled, create a prediction problem for unlabeled data. Define a generative model with corresponding log-likelihood. Generally more effective and more expensive than pre-training. Ex: predict relative position of image patch, predict coloring of gray-scale image
Multi-task learning: share (lower-level) representations across tasks and learn these jointly

Recurrent Neural Networks

Simple RNNs

Given a sequence of observations $\mathbf{x}^1,...,\mathbf{x}^s$, not iid
Discrete time evolution of hidden state space sequence:
$\mathbf{h}^t = F(\mathbf{h}^{t-1},\mathbf{x}^t;\boldsymbol{\theta}) \Rightarrow$ Markov property and time-invariant
$\mathbf{h}^t = F(\mathbf{h}^{t-1},\mathbf{x}^t;\boldsymbol{\theta}) = \sigma \circ \overline{F}(\mathbf{h}^{t-1},\mathbf{x}^t;\boldsymbol{\theta}), \quad \overline{F}(\mathbf{h},\mathbf{x};\boldsymbol{\theta}) = \mathbf{W}\mathbf{h} + \mathbf{U}\mathbf{x} + \mathbf{b}$
Optionally produce outputs: $\mathbf{y}^t = H(\mathbf{h}^t;\boldsymbol{\theta}), \quad H(\mathbf{h},\boldsymbol{\theta}) := \sigma(\mathbf{V}\mathbf{h} + \mathbf{c})$



Hidden state can be thought of as a noisy memory, compresses relevant aspects of sequence: $(\mathbf{x}^1,...,\mathbf{x}^{t-1}) \mapsto \mathbf{h}^t$ (conceptually)
For any fixed length s, the unrolled recurrent net corresponds to a feedforward net with s hidden layers. Difference to MLP: sharing of parameters and inputs processed sequentially..
Backpropagation (through time :P): sum over time steps
With $\dot{\sigma}_i^t := \sigma'(\overline{F}_i(h^{t-1},x^t))$:

$$\frac{\partial\mathcal{R}}{\partial w_{ij}} = \sum_{t=1}^{s}\frac{\partial\mathcal{R}}{\partial h_i^t}\cdot\frac{\partial h_i^t}{\partial w_{ij}} = \sum_{t=1}^{s}\frac{\partial\mathcal{R}}{\partial h_i^t}\cdot\dot{\sigma}_i^t\cdot h_j^{t-1}$$

$$\frac{\partial\mathcal{R}}{\partial u_{ik}} = \sum_{t=1}^{s}\frac{\partial\mathcal{R}}{\partial h_i^t}\cdot\frac{\partial h_i^t}{\partial u_{ik}} = \sum_{t=1}^{s}\frac{\partial\mathcal{R}}{\partial h_i^t}\cdot\dot{\sigma}_i^t\cdot x_k^t$$

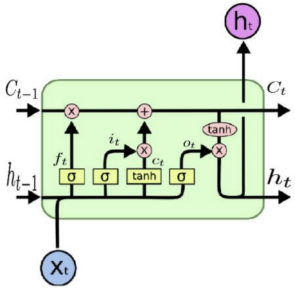
Example with setting: $y_{1:T}$ ground-truth outputs and $\hat{y}_{1:T}$ predictions, $a_t = F(x_t,h_{t-1},y_{t-1};\theta)$, $h_t = \sigma(a_t)$, $\hat{y}_t = G(h_t,\phi)$, $L_t = H(y_t,\hat{y}_t)$, $L = \sum_{t=1}^{T} L_t$. Derivative of L wrt θ:

$$\frac{\partial L}{\partial\theta} = \sum_{t=1}^{T}\frac{\partial L_t}{\partial\hat{y}_t}\frac{\partial\hat{y}_t}{\partial h_t}\sum_{i=0}^{t}(\prod_{j=t}^{i+1}\frac{\partial h_j}{\partial a_j}\frac{\partial a_j}{\partial h_{j-1}})\frac{\partial h_i}{\partial a_i}\frac{\partial a_i}{\partial\theta}$$

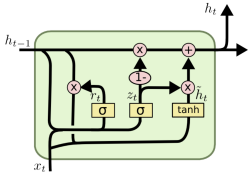
Exploding/Vanishing gradients: let output in last step $\mathbf{y} = \mathbf{y}^s$
Then $\nabla_{\mathbf{x}^t}\mathcal{R} = [\prod_{r=t+1}^{s}\mathbf{W}^T\mathbf{S}(\mathbf{h}^r)]\cdot\mathbf{J}_H\cdot\nabla_{\mathbf{y}}\mathcal{R}, \quad \mathbf{S}(\mathbf{h}^r) = \mathrm{diag}(\dot{\sigma}_1^r,...,\dot{\sigma}_n^r)$
Take spectral norm, and use $\|\mathbf{A}\mathbf{B}\|_2 \leq \|\mathbf{A}\|_2\cdot\|\mathbf{B}\|_2$. If $\sigma_{\max}(\mathbf{W}) < 1$:
$\|\nabla_{\mathbf{x}^t}\mathcal{R}\| \leq \sigma_{\max}(\mathbf{W})^{s-t}\cdot\|J_H\cdot\nabla_{\mathbf{y}}\mathcal{R}\| \stackrel{(s-t)\to\infty}{\to} 0$
Conversely may explode if $\sigma_{\max}(\mathbf{W}) > 1$
Bi-directional network: reverse order $\mathbf{g}^t = G(\mathbf{x}^t,\mathbf{g}^{t+1};\boldsymbol{\theta})$
Interweave the two hidden state sequences
Deep recurrent network: hierarchical hidden states of l layers
$\mathbf{h}^{t,1} = F(\mathbf{h}^{t-1,1},\mathbf{x}^t;\boldsymbol{\theta})$
$\mathbf{h}^{t,l} = F(\mathbf{h}^{t-1,l},\mathbf{h}^{t,l-1};\boldsymbol{\theta}), \quad l = 2,...,L$

Memory Units

Addressing the problem of vanishing/exploding gradients
Gated units to learn long-term dependencies
Difficult to understand what units learn, resource-hungry and slow in learning
LSTM



Content C has similar effect as ResNet for vanishing gradients
Forget gate: $f_t = \sigma(W_f\cdot[h_{t-1},x_t]+b_f)$
Input to memory:
$i_t = \sigma(W_i\cdot[h_{t-1},x_t]+b_i), \quad \tilde{C}_t = \tanh(W_C\cdot[h_{t-1},x_t]+b_C)$
Updating memory: $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$
Output gate:
$o_t = \sigma(W_o[h_{t-1},x_t]+b_o), \quad h_t = o_t * \tanh(C_t)$
Gates: input gate, output gate and forget gate
Gated Recurrent Unit (GRU)



$$z_t = \sigma(W_z\cdot[h_{t-1},x_t])$$
$$r_t = \sigma(W_r\cdot[h_{t-1},x_t])$$
$$\tilde{h}_t = \tanh(W\cdot[r_t * h_{t-1},x_t])$$
$$h_t = (1-z_t) * h_{t-1} + z_t * \tilde{h}_t$$

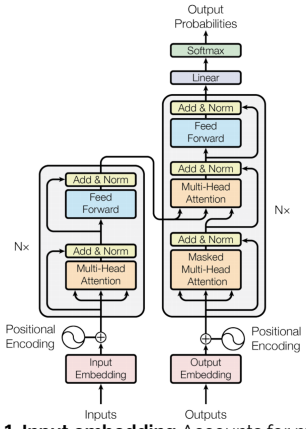
Gates: update gate and reset gate
Here h is both state and content, computed as a convex combination of old and new information
Learning sequences - Seq2Seq
Goal: learn $p(\mathbf{y}^{1:T}|\mathbf{x}^{1:T}) \approx \prod_{t=1}^{T} p(\mathbf{y}^t|\mathbf{x}^{1:t},\mathbf{y}^{1:t-1})$
Naive implementation: $p(\mathbf{y}^t)$ depends on $\mathbf{y}^{1:t-1}$ only through $\mathbf{h}^t$ $\Rightarrow$ remedy by feeding back previous outputs!
Teacher forcing: When training, compute loss on predicted output but feed back in correct output. During prediction feed back predicted outputs. Improves learning BUT gives exposure bias
Encoder-Decoder model:
Encoder: $(\mathbf{x}^1,...,\mathbf{x}^T) \mapsto \mathbf{z}, \quad \mathbf{z} = \mathbf{h}^T$ (RNN)
Decoder $\mathbf{z} \mapsto (\mathbf{y}^1,...,\mathbf{y}^S)$ (RNN with output feedback)
Can also be used for image captioning with CNN encoder

Attention

Instead of compressing the whole sequence into a vector, determine where to look
Self attention
Gating function: Given query $\boldsymbol{\xi} \in \mathrm{R}^n$ and values $\mathbf{x}^t \in \mathrm{R}^m$,

$$f_\phi(\boldsymbol{\xi},(\mathbf{x}^1,...,\mathbf{x}^s)) = \frac{1}{\sum_j \exp[\phi(\boldsymbol{\xi},\mathbf{x}^j)]}\begin{bmatrix}\exp[\phi(\boldsymbol{\xi},\mathbf{x}^1)]\\ ...\\ \exp[\phi(\boldsymbol{\xi},\mathbf{x}^s)]\end{bmatrix}$$

Simplest choice for similarity function when $m = n$: $\phi(\boldsymbol{\xi},\mathbf{x}) = \boldsymbol{\xi}\cdot\mathbf{x}$
Transfer function: $F(\boldsymbol{\xi},(\mathbf{x}^1,...,\mathbf{x}^s)) = [\mathbf{x}^1 \quad ... \quad \mathbf{x}^s]\cdot f_\phi(\boldsymbol{\xi},(\mathbf{x}^1,...,\mathbf{x}^s))$
i.e. computes convex combination of inputs wrt attention weights
RNN with attention (Seq2Seq): combining ideas of encoding information relevant for the future (RNNs) and selecting what is relevant in retrospective (attention)
Attend to hidden state sequence $(\mathbf{h}_e^1,...,\mathbf{h}_e^s)$ of encoder with query $\boldsymbol{\xi}^i$ produced as hidden states by decoder. Use this as input into decoder, which produces hidden states $(\boldsymbol{\xi}^1,...,\boldsymbol{\xi}^t)$ and output sequence $(\mathbf{y}^1,...,\mathbf{y}^t)$
Memory Networks
Recurrent attention model over possibly large external memory
Recursive associative recall: Given query $\mathbf{q}$ (e.g. question), find best matching memory cell i, use its content $\mathbf{m}_i$ and $\mathbf{x}$ to generate new query - repeat
Transformers
Attention is all you need
Key-Value attention map: given KV pairs $(\mathbf{x}^i,\mathbf{z}^i)$,
$F(\boldsymbol{\xi},((\mathbf{x}^1,\mathbf{z}^1),...,(\mathbf{x}^s,\mathbf{z}^s))) = [\mathbf{z}^1 \quad ... \quad \mathbf{z}^s]\cdot f(\boldsymbol{\xi},(\mathbf{x}^1,...,\mathbf{x}^s))$
$\Rightarrow$ keys determine where to look, values what features to extract
Similarity function: $\phi(\boldsymbol{\xi},\mathbf{x}) = \frac{\boldsymbol{\xi}\cdot\mathbf{x}}{\sqrt{n}}, \quad \boldsymbol{\xi},\mathbf{x} \in \mathrm{R}^n$
Assuming entries sampled from standard Gaussian, this gives standard scaling. Good since softmax can be sensitive to large input values (which kills gradient and slows down learning)
Multi-headed attention: $G(\boldsymbol{\xi},(\mathbf{x}^t,\mathbf{z}^t)_{t=1}^s) = \mathbf{W}\begin{bmatrix}F_1(\boldsymbol{\xi},(\mathbf{x}^t,\mathbf{z}^t))\\ ...\\ F_h(\boldsymbol{\xi},(\mathbf{x}^t,\mathbf{z}^t))\end{bmatrix}$
$F_j(\boldsymbol{\xi},(\mathbf{x}^t,\mathbf{z}^t)) = F(\mathbf{W}_j^q\boldsymbol{\xi},(\mathbf{W}_j^x\mathbf{x}^t,\mathbf{W}_j^z\mathbf{z}^t))$
Alt.: $\mathbf{Y} = \mathrm{softmax}(\mathbf{X}\mathbf{W}_q\mathbf{W}_k^T\mathbf{X}^T)\mathbf{X}\mathbf{W}_v, \mathbf{W}_{\{q,k,v\}} \in \mathrm{R}^{d\times hd}, \mathbf{X},\mathbf{Y} \in \mathrm{R}^{L\times d}$



1. Input embedding Accounts for meaning
2. Positional Encodings Accounts for position in the input sentence. Normally use sinusoidal functions. For position t and feature k:

$$p_{tk} = \begin{cases} \sin(t\omega_k) & k \text{ even} \\ \cos(t\omega_k) & k \text{ odd} \end{cases} \qquad \omega_k = C^{k/n}, C = 10000$$

3. Creating Masks In decoder, to prevent peaking ahead.
4. Multi-Head Attention Layer Split embedding into N heads and perform attention on each. Three types: Self, masked and mixed.
5. Feed-Forward layer

Applications in NLP

Construct word embeddings that reflect the context
ELMo: Input fixed embeddings on character level, build word representations with CNNs, then stack left-to-right LSTM and right-to-left LSTM. Output (for each word) is convex combination of hidden states over layers. Out-of-vocabulary (not restricted to fixed vocabulary). Train in task-specific manner.
BERT: Input sub-word unit embeddings. Leverage attention - take query and retrieve keys & values from words in context. Insight: do not need to learn a language model. Train by word masking task - predict missing word in text. Also do next sentence prediction in pre-training. Can fine-tune for specific downstream task. **BERTology** - representations hierarchical, struggles with negation, incomplete syntactic knowledge etc
GPT-n: Few, one or zero shot learning - add task description & examples to working memory and let model do the rest.

Theory of DNNs

VC Theory

Shattering coefficient: maximum number of ways in which n points can be classified by function class $\mathcal{F}$
$\Rightarrow \mathcal{S}_{\mathcal{F}}(n) = \sup_{(\mathbf{x}_1,...,\mathbf{x}_n)} |\{(f(\mathbf{x}_1),...,f(\mathbf{x}_n)) : f \in \mathcal{F}\}| \leq 2^n$
Say that $\mathcal{F}$ **shatters** a set of n points if $S_{\mathcal{F}}(n) = 2^n$
VC-dim The VC-dim of $\mathcal{F}$ is $n \iff$ there is a set of size n that is shattered by $\mathcal{F}$, and **no** set of size $n+1$ is shattered by $\mathcal{F}$.
VC-dim$(\mathcal{F}) \leq \log_2(|\mathcal{F}|)$
Vapnik-Chervonenkis Theorem: For any $\delta > 0$, with probability at least $1-\delta$:

$$\forall f \in \mathcal{F}, \quad \mathcal{R}(f) \leq \mathcal{R}_n(f) + 2\sqrt{2 \cdot \frac{\log \mathcal{S}_{\mathcal{F}}(2n) + \log\frac{2}{\delta}}{n}}$$

PAC Bayes Bounds

Donsker's Theorem: For any $P >> Q$, P-measurable function ϕ
$\mathrm{E}_Q[\phi] \leq \mathrm{KL}(Q\|P) + \log \mathrm{E}_P[e^{\phi}]$
McAllister theorem $\forall$ fixed P and any Q, $\epsilon \in (0,1)$ w.prob $\geq \epsilon$ over sample set S: $\mathbb{E}_Q[e_f] - \mathbb{E}_Q[e_f^S] \leq \sqrt{\frac{2}{|S|}\left[KL(Q\|P) + \ln\left(\frac{2\sqrt{|S|}}{\epsilon}\right)\right]}$.
Interpretation for DNNs: Take $P = \mathcal{N}(\theta_0, \lambda \mathrm{I})$ to be prior over parameter space. $Q = \mathcal{N}(\theta, \mathrm{diag}(\sigma^2))$ (learned from data) is a distribution over $\mathcal{F}$, where our function f is sampled from. Minimize bound together with (surrogate of) empirical risk.
A **stochastic neural network** is a network whose weights are drawn from a distribution Q each time data is propagated through the network.

Learning on Graphs

Motivation: Data may not be sampled iid but could be graph-structured
Use-cases: Node classification, link prediction, generative modeling
Focus here: GCN from paper *Semi-Supervised Classification with Graph Convolutional Networks*, Kipf & Welling 2017
Idea: To make prediction on node, gather information from context in graph
Let $\mathbf{X} \in \mathbb{R}^{d \times n}$ be graph of n nodes with d features, then define:
$\mathbf{X}^{l+1} = \sigma(\mathbf{W}^l \mathbf{X}^l \mathbf{Q})$, where $\mathbf{WX}$ sums over dimensions and $\mathbf{XQ}$ sums over data points / nodes (to learn from neighborhood)

Define degree-normalized matrix $Q = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, where $\tilde{D}$ is the diagonal degree matrix of $\tilde{A} := A + \mathrm{I}_n$ and A is adjacency matrix. It holds that:

$$q_{ij} = \frac{a_{ij} + \delta_{ij}}{\sqrt{\tilde{d}_i \tilde{d}_j}}, \quad \tilde{d}_i = 1 + \textstyle\sum_j A_{ij}, \quad \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

Activations from neighbors are mixed together with respective Q-weights: and $(X^l Q)_{ij} = ([x^l[1],...,x^l[n]] \cdot Q)_{ij} = \sum_{k=1}^n x_{ik} q_{kj}$
$\Rightarrow (X^l Q)_{\bullet j} = \sum_k q_{kj} \cdot x^l[k]$ (sum over neighborhood of node j)
Cyclic chain: For regular lattice graph, Q is Toeplitz matrix and thus related to convolutional layer with window size same as neighborhood size
Linear Shift Invariant Filter: linear function H over graph with adjacency matrix A such that $H(Ax) = A(Hx)$. H is A-shift invariant iff: $H = \theta_0 \mathrm{I} + \theta_1 A + ... + \theta_n A^n$

If we consider $H(\theta_0, \theta_1) = \theta_0 \mathrm{I} + \theta_1 (D^{-\frac{1}{2}} A D^{-\frac{1}{2}})$ and $\theta_0 = \theta_1$ then largest eigenvalue is 2, which can be a source of instabilities.

Motivates $\mathrm{I} + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \mapsto \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, which has largest EV 1

Adversarial Robustness

Adversarial examples are small manipulations of input which cause the model to change its prediction
Additive adversarial manipulation: Find $x \in \mathrm{R}^d$ s.t. $k(x+r) \neq k(x)$ and $x+r$ similar to x. As proxy for *similar*, use l_p-norm robustness - $\|r\|_p$ should be small
Two approaches to measure robustness to adversarial perturbations: unconstrained & constrained perturbations

Unconstrained Perturbations - DeepFool

Find smallest perturbation in l_p-norm that induces mistake

Let $f : \mathrm{R}^d \to \mathrm{R}^C$ be the logits, i.e. outputs before softmax, and wlog $k(x) = 1$
Disadvantage of approach: Typically get close to decision boundary which may be exploited to detect them (can e.g. train a classifier to detect them given some confidence threshold)

$\hat{\rho}_{\text{adv}} = \frac{1}{n} \sum_{i=1}^n \frac{\|r(x_i)\|_2}{\|x_i\|_2}$

DeepFool - Binary case: $\arg\min_r \|r\|_p$ s.t. $f_2(x+r) - f_1(x+r) > 0$

First-order Taylor approximation around x, rewrite constraint:
$(\nabla_x f_1(x) - \nabla_x f_2(x))^T r + f_1(x) - f_2(x) < 0 \Rightarrow$ linearized classifier
Solve iteratively until $k(x + r_1 + ... + r_n) \neq k(x)$
Optimum in each step, $1/p + 1/p^* = 1$:

$$r^* = \frac{f_1(x) - f_2(x)}{\|\nabla_x f_1(x) - \nabla_x f_2(x)\|_{p^*}^{p^*}} \cdot |\nabla_x f_1(x) - \nabla_x f_2(x)|^{p^*-1} \odot \mathrm{sgn}(\nabla_x f_1(x) - \nabla_x f_2(x))$$

DeepFool - General case: Optimize wrt r_i for each class and update in the direction of smallest r_i
$x_0 = x$

$$x_k = x_{k-1} + \min_{j \neq k(x)} \left\{ \frac{|h_j(x_{k-1})|}{\|\nabla_x h_j(x_{k-1})\|_{p^*}^{p^*}} \cdot |\nabla_x h_j(x_{k-1})|^{p^*-1} \odot \mathrm{sgn}(\nabla_x h_j(x_{k-1})) \right\}, \text{ where } h_j(x_{k-1}) = f_j(x_{k-1}) - f_{k(x)}(x_{k-1})$$

Constrained Perturbations - PGD

Given $\epsilon > 0$ find perturbation r with $\|r\|_p \leq \epsilon$
Disadvantage of approach: Perturbation found can be larger than the minimal one within ball
Projected Gradient Descent / Ascent:
$x_0 \sim \mathcal{U}(\mathcal{B}_\epsilon^p(x))$

$$x_k = \Pi_{\mathcal{B}_\epsilon^p(x)}(x_{k-1} + \alpha \underset{v_k : \|v_k\|_p \leq 1}{\arg\max} v_k^T \nabla_x l(y, f(x_{k-1})))$$

where $r_k = x_k - x$, $\Pi_{\mathcal{B}_\epsilon^p(x)}(\tilde{x}) = \arg\min_{x^* \in \mathcal{B}_\epsilon^p(x)} \|\tilde{x} - x^*\|_2$ and

$$v^* = \underset{v : \|v\|_p \leq 1}{\arg\max} v^T z = \frac{\mathrm{sign}(z) \odot |z|^{p^*-1}}{\|z^*\|_{p^*}^{p^*-1}}$$

Adversarial Training: Most effective method for improving robustness. Can combine with e.g. PGD:

$$\min_\theta \max_{\|r\|_p \leq \epsilon} (f_{\theta^T}(x+r), y)$$

Accuracy on clean data degrades $\to$ more data
Network overfits to attacks used in training $\to$ use many iterations for PGD
Takes longer to train $\to$ explicit regularization
Mode of failure 1: gradient wrt input norm becomes too small
Mode of failure 2: gradient wrt input becomes noisy $\to$ not find closest perturbation

Benefits beyond robustness and security

Interpretability - robust networks learn more human-interpretable features
Transfer learning - robust networks generalize better across domains
Generative modeling - robust networks perform better in generative tasks

Adversarial training and operator norm regularization

Linearization: $f(\mathbf{x} + \Delta\mathbf{x}) \approx f(\mathbf{x}) + J_f(\mathbf{x}) \Delta\mathbf{x}$

$$\Rightarrow \frac{\|f(\mathbf{x}+\Delta\mathbf{x}) - f(\mathbf{x})\|_2}{\|\Delta\mathbf{x}\|_2} \approx \frac{\|J_f(\mathbf{x})\Delta\mathbf{x}\|_2}{\|\Delta\mathbf{x}\|} \leq \sigma(J_f(\mathbf{x})) =: \max_{v : \|v\|_2 \leq 1} \|J_f(\mathbf{x}) v\|_2$$

So from a robustness perspective, want small operator norm
Data-independent spectral norm reg.: $\sigma(J_f(\mathbf{x})) \leq \prod_{l=1}^L \sigma(\mathbf{W}^l)$
Generalizes from train to test set but can be arbitrarily loose
Data-dependent: $\min_\theta \mathrm{E}_{(\mathbf{x},y)\sim\tilde{P}}[l(y, f(\mathbf{x})) + \frac{\lambda}{2} \sigma(J_f(\mathbf{x}))^2]$,
where $\sigma(J_f(\mathbf{x}))$ is computed by the power method
l_p-norm constrained PGA based adversarial training with an l_q-norm loss on the logits of clean and perturbed inputs is equivalent to data-dependent (p,q) operator norm regularization.

VAEs

Linear Factor Analysis

Latent variable prior $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathrm{I}), \quad \mathbf{z} \in \mathrm{R}^m$
Linear observation model for $\mathbf{x} \in \mathrm{R}^n$: $\mathbf{x} = \boldsymbol{\mu} + \mathbf{W}\mathbf{z} + \boldsymbol{\eta}$,
$\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$, $\boldsymbol{\Sigma} := \mathrm{diag}(\sigma_1^2, ..., \sigma_n^2)$
Can be shown that: $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^T + \boldsymbol{\Sigma})$
Non-identifiability, for orthogonal $\mathbf{Q}$: $(\mathbf{W}\mathbf{Q})(\mathbf{W}\mathbf{Q})^T = \mathbf{W}\mathbf{W}^T$
Posterior inference:
$\boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}} = \mathbf{W}^T (\mathbf{W}\mathbf{W}^T + \boldsymbol{\Sigma})^{-1} (\mathbf{x} - \boldsymbol{\mu})$
$\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}} = \mathbf{I} - \mathbf{W}^T (\mathbf{W}\mathbf{W}^T + \boldsymbol{\Sigma})^{-1} \mathbf{W}$
MLE: $\boldsymbol{\theta} = (\boldsymbol{\mu}, \mathbf{W}) \overset{\max}{\leftarrow} \log p(\mathbf{X}; \boldsymbol{\mu}, \mathbf{W})$, has no closed form solution
Probabilistic PCA: when $\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}$

Variational Autoencoders

Generalize factor analysis with depth
Noise variable: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
Density unavailable explicitly, so can't do MLE:
$\boldsymbol{\theta} \overset{\max}{\leftarrow} \sum_{i=1}^n \log p(\mathbf{x}[i]; \boldsymbol{\theta})$
Max. ELBO: $\log p(\mathbf{x}; \boldsymbol{\theta}) \geq \mathrm{E}_{\mathbf{z}\sim q(\mathbf{z};\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \boldsymbol{\theta})] - \mathrm{KL}(q(\mathbf{z};\mathbf{x}) \| p(\mathbf{z}))$
Can think of $q(\mathbf{z};\mathbf{x})$ as posterior (= $p(\mathbf{z}|\mathbf{x})$), restricted to (typically Gaussian) variational family:
$\mathcal{N}(\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\Sigma}(\mathbf{x}))$, $\boldsymbol{\Sigma}(\mathbf{x}) = \mathrm{diag}(\sigma_1^2(\mathbf{x}), ..., \sigma_n^2(\mathbf{x}))$ (output of encoder)
Optimizing over q involves gradients of expectations $\Rightarrow$ stochastic backpropagation / re-parameterization trick:
$\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad \Leftrightarrow \quad \mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\Sigma}^{1/2} \boldsymbol{\eta}, \quad \boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \mathrm{I})$
$\Rightarrow$ can now backpropagate wrt $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}^{1/2}$!

Generative Models

Density Estimation

Learn a parametrized model $p_\theta(\mathbf{x})$ to be indistinguishable from true generative process $p(\mathbf{x})$
1 - Prescribed model: Density explicitly specified (and accessible). Can do MLE. Challenge: Normalizing model
2 - Implicit model: Transformation of densities, can describe density through integration by substitution
Integration by substitution - change of variables formula:
$\mathbf{x} = F(\mathbf{z}), \quad p_X(\mathbf{x}) = p_Z(F^{-1}(\mathbf{x})) |\det(\partial F)|^{-1}$, p_X pushforward of p_Z
Normalizing Flows
Bijections F which are convenient to compute, invert and calculate $|\det(\partial F)|$
$F = F_L \circ \cdots \circ F_1, \quad F^{-1} = F_1^{-1} \circ \cdots \circ F_L - 1$
$\Rightarrow \det(\partial F) = \prod_{l=L}^1 \det(\partial F_l \circ F_{l-1:1}), \quad \det(\partial F^{-1}) = \det(\partial F)^{-1}$
Log-likelihood: $\log p(\mathbf{x}|\mathbf{z}) = -\sum_{l=1}^L \log|\det(\partial F_l \circ F_{1:l-1})|$
Linear flow: $F(\mathbf{x}) = \mathbf{A}\mathbf{z} + \mathbf{b} \Rightarrow \mathcal{O}(n^3)$ to compute determinant and inverse. If A diagonal, simple & efficient but not powerful.
If A triagonal: $\det(A) = \prod_i A_{ii}$, but A^{-1} expensive...
Invertible linear time flows: $F(\mathbf{z}) = \mathbf{z} + \mathbf{u}\sigma(\mathbf{w} \cdot \mathbf{z} + b)$
Determinant: $|\det(\partial F)| = |1 + \sigma' \mathbf{u}^T \mathbf{w}|$
In general, normalizing flows not powerful enough as Jacobian condition is too restrictive $\Rightarrow$ Likelihood-free methods
Generative Adversarial Networks (GANs)
Derive training signal for generator G from classifier D that discriminates data from model-generated samples
For sample $\mathbf{x}$ and model label y: $\tilde{p}_\theta(\mathbf{x}, y) = \frac{1}{2}(y p(\mathbf{x}) + (1-y) p_\theta(\mathbf{x}))$
$\Rightarrow$ balanced binary classification problem
Optimal discriminator - Bayes optimal classifier:
$q_\theta(\mathbf{x}) = \frac{p(\mathbf{x})}{p(\mathbf{x}) + p_\theta(\mathbf{x})} = \mathrm{P}(y = 1|\mathbf{x})$
Train generator by minimizing log-likelihood (=JS for BOC)
$l^*(\theta) := \mathrm{E}_{\tilde{p}_\theta}[y \log q_\theta(\mathbf{x}) + (1-y)\log(1 - q_\theta(\mathbf{x}))] = ... = \mathrm{JS}(p, p_\theta) - \log 2$
Bayes optimal classifier not accessible, define classification model: $q_\phi : \mathbf{x} \mapsto [0,1], \quad \phi \in \Phi$
It holds that: $l^*(\theta) \geq \sup_{\phi \in \Phi} l(\theta, \phi)$ (given a generator, the log-likelihood with the BOC is an upper bound for the log-likelihood with any other classifier)
General objective: $l(\theta, \phi) := \mathrm{E}_{\tilde{p}_\theta}[y \log q_\phi(\mathbf{x}) + (1-y)\log(1 - q_\phi(\mathbf{x}))]$
Saddle-point problem: $\theta^* := \arg\min_{\theta \in \Theta} \{\sup_{\phi \in \Phi} l(\theta, \phi)\}$
Inner sup is impractical $\Rightarrow$ SGD heuristic:
$\theta^{t+1} = \theta^t - \eta \nabla_\theta l(\theta^t, \phi^t), \quad \phi^{t+1} = \phi^t + \eta \nabla_\phi l(\theta^{t+1}, \phi^t)$
Alternatively, with $D(\cdot)$ the probability discriminator assigns that sample is from real data:
$\min_G \max_D \mathrm{E}_{\mathbf{x}\sim p(\mathbf{x})}[\log D(\mathbf{x})] + \mathrm{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$
Mode collapse (Helvetica scenario): Generator collapses too many values of z to similar values of x and does not capture the diversity in $p(\mathbf{x})$. Avoid by not training G too much without updating D. Can lead to good-looking samples without diversity
Vanishing gradient: If the discriminator becomes too strong the gradient wrt the parameters of the generator vanishes ($\to 0$), so there will be no learning signal for the generator and it will not improve. Can lead to noisy/blurry samples but good variability $\Rightarrow$ trade-off
Why does minimizing Jensen-Shannon divergence work?
Definition: $\mathrm{JS}(Q\|P) = \frac{1}{2}\mathrm{KL}(P\|\frac{P+Q}{2}) + \frac{1}{2}\mathrm{KL}(Q\|\frac{P+Q}{2})$
In MLE we minimize the forward KL-divergence $\mathrm{KL}(P\|Q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$, which leads $q(x)$ to spread its mass over the whole support of $p(x)$, which is why samples from e.g. VAEs tend to be blurry.
If we minimize the reverse KL-divergence $\mathrm{KL}(Q\|P)$, $q(x)$ can be set to zero when $p(x) > 0$, which leads to a $q(x)$ that focuses on certain mode(s) of $p(x)$.
Want something in between where whole support of $p(x)$ is covered while $q(x) = 0$ where $p(x) = 0 \Rightarrow$ **JS-divergence**!