Lin. Fcn A function $f: \mathbf{R}^n \to \mathbf{R}$ is a linear function if:
(1) $f(x+x') = f(x) + f(x'), (\forall x, x' \in \mathbf{R}^n)$
(2) $f(\alpha x) = \alpha f(x), (\forall \alpha \in \mathbf{R})$ P: f linear $\Leftrightarrow f(x) = w^\top x$ for some $w \in \mathbf{R}^n$. Proof: "$\Leftarrow$" Properties of scalar product: (1) $f(x+x') = ... = f(x) + f(x')$; (2) $f(\alpha x) = ... = \alpha f(x)$; "$\Rightarrow$" Write $x = \sum_{i=1}^n x_i e_i$. Linearity implies: $f(\mathbf{x}) = \sum_{i=1}^n x_i f(e_i)$ identify $w_i := f(e_i)$ **Hyperplane** A hyperplane is an affine subspace of co-dimension 1. **Level set** The level sets of a fucntion $f: \mathbf{R}^n \to \mathbf{R}$ is a one-parametric family of sets defined as $L_f(c) := \{x : f(x) = c\} = f^{-1}(c) \subseteq \mathbf{R}^n$ **Level sets of linear functions** Let $f: \mathbf{R}^n \to \mathbf{R}$ be linear, $f(x) = w^\top x + b$, then $L_f(c) = \{x : w^\top x = c - b\} =$ hyperplane $\perp w$
Linear (affine) maps $F: \mathbf{R}^n \to \mathbf{R}^m$ with
$$F(x) = \begin{pmatrix} f_1(x) \\ f_m(x) \end{pmatrix} = \begin{pmatrix} w_1^\top x + b_1 \\ w_m^\top x + b_m \end{pmatrix} = \begin{pmatrix} w_1^\top \\ w_m^\top \end{pmatrix} x + \begin{pmatrix} b_1 \\ b_m \end{pmatrix}$$
Composition of linear maps P: Let $F_1, .., F_L$ be linear maps, then $F = F_L \circ \cdots \circ F_1$ is also a linear map. Proof: $F(\mathbf{x}) = (\mathbf{W}_L ... (\mathbf{W}_2(\mathbf{W}_1 \mathbf{x}))...) = (\mathbf{W}_L ... \mathbf{W}_2 \mathbf{W}_1)\mathbf{x} = \mathbf{W}\mathbf{x}$
- every L-level hierarchy collapses to one level
- note that $rank(F) \equiv dim(im(F)) \leq \min_l rank(F_l)$

Conclusion: Need to move beyond linearity!

Approximation Theory **Ridge function:** $f: \mathbf{R}^n \to \mathbf{R}$ is a ridge function, if it can be written as $f(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$. Limit set: $L_f(c) = \cup_{d \in \sigma^{-1}(c)} L_{\bar{f}}(d)$, if linear part of f denoted by $\bar{f}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$. If σ is differentiable at $z = \mathbf{w}^\top \mathbf{x} + b$ then $\nabla_x f \overset{\text{chain rule}}{=} \sigma'(z) \nabla_x \bar{f} = \sigma'(z)\mathbf{w}$
Theorem Let $f: \mathbf{R}^n \to \mathbf{R}$ differentiable at $\mathbf{x}$. Then either $\nabla f(\mathbf{x}) = 0$ or $\nabla f(\mathbf{x}) \perp L_f(f(\mathbf{x}))$.
Dense Subsets: A function class $\mathcal{H} \subseteq C(\mathbf{R}^d)$ is dense in $C(\mathbf{R}^d)$ iff $\forall f \in C(\mathbf{R}^d) \forall \epsilon > 0 \forall K \subset \mathbf{R}^d$, compact: $\exists h \in \mathcal{H} s.t. \max_{\mathbf{x} \in K} |f(\mathbf{x}) - h(\mathbf{x})| = \|f - h\|_{\infty, K} < \epsilon$

Conclusion: We cann approximate any continuous f to arbitrary accuracy (on K) with a suitable member of $\mathcal{H}$. $\Rightarrow$ uniform approximation on compacta (i.e. use of ∞-norm) $\Rightarrow$ sup $\to$ max (Bolzano-Weierstrass)

Universal Approximation with Ridge Functions Let $\sigma: \mathbf{R} \to \mathbf{R}$ be a scalar function
$\mathcal{G}_\sigma^n := \{g : g(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$ for some $\mathbf{w} \in \mathbf{R}^n, b \in \mathbf{R}\}$
$\mathcal{G}^n := \cup_{\sigma \in C(\mathbf{R})} \mathcal{G}_\sigma^n$, universe of continuous ridge functions
Theorem: Vostrecov and Kreines, 1961
$\mathcal{H}^n := \text{span}(\mathcal{G}^n)$ is dense in $C(\mathbf{R}^n)$.
Dimension Lifting Lemma (Pinkus) L(Pinkus 1999): The density of $\mathcal{H}_\sigma^1$ in $C(\mathbf{R})$ with $\mathcal{H}_\sigma^1 := \text{span}(\mathcal{G}_\sigma^1) = \text{span}\{\sigma(\lambda t + \theta) : \lambda, \theta \in \mathbf{R}\}$ implies the density of $\mathcal{H}_\sigma^n := \text{span}(\mathcal{G}_\sigma^1) = \text{span}\{\sigma(\mathbf{w}^\top \mathbf{x} + b) : \mathbf{w} \in \mathbf{R}^n, b \in \mathbf{R}\}$ in $C(\mathbf{R}^n)$ for any $n \geq 1$.
Conclusion: We can lift density property of ridge function families from $C(\mathbf{R})$ to $C(\mathbf{R}^n)$.

- Continuous functions can be well approximated by linear combinations of ridge functions (universal function approximation).
- Justifies use of computational units which apply a scalar nonlinearity to a linear function of the inputs.

Rectification Networks ReLU: $(x)_+ := \max(0, x)$; $\partial(x)_+ = 1(x > 0), 0(x < 0), [0; 1](x = 0)$; AVU: $|x| := x(x \geq 0), -x(\text{otw.}); \partial|x| = 1(x > 0), [-1; 1](x = 0), -1(x < 0)$

Shektman (1982): Any $f \in C[0;1]$ can be uniformly approximated to arbitrary precision by a polygonal line.
Lebesgue (1898): A polygonal line with m pieces can be written: $\boxed{g(x) = ax + b + \sum_{i=1}^{m-1} c_i (x - x_i)_+}$ knots: $0 = x_0 < x_1 < \cdots < x_{m-1} < x_m = 1$; $m+1$ parameters $a, b, c_i \in \mathbf{R}$; ReLU function approximation in 1D; $\boxed{g(x) = a'x + b' + \sum_{i=1}^{m-1} c_i' |x - x_i|}$

Weierstrass: $C[0;1]$ functions can be uniformly approximated by polynomials. Lebesgue: proof for W. theorem by showing that $|x|$ can be uniformly approximated on $[-1;1]$ by polynomials

T: Networks with one hidden layer of ReLU or AVU are universal function approximators

Linear Combinations of Rectified Units
By linearly combining m rectified units, into how many $(R(m))$ cells is $\mathbf{R}^n$ maximally partioned? (Zaslavsky, 1975)
$R(m) \leq \sum_{i=0}^{\min\{m,n\}} \binom{m}{i}$; for $m \leq n$, $R(m) = 2^m$ (exponential growth); for given n, asymptotically, $R(m) \in \Omega(m^n)$ (bounded by m^n), i.e. there is a polynomial slow-down, which is induced by the limitation of the input space dimension.
Deep Combinations of Rectified Units Process n inputs through L ReLU layers with widths $m_1, ..., m_L \in O(m)$. Into how many $(R(m, L))$ cells can $\mathbf{R}^n$ be maximally partitioned? T(Montufar, 2014): $R(m, L) \in \Omega\left(\left(\frac{m}{n}\right)^{n(L-1)} m^n\right)$. For any fixed n, exponential growth can be ensured by making layers sufficiently wide $(m > n)$ and increasing the level of functional nesting (i.e. depth L).
Hinging Hyperplanes D.: Hinge function: If $f: \mathbf{R}^n \to \mathbf{R}$ can be wrriten with parameters $w_1, w_2 \in \mathbf{R}^n$ and $b_1, b_2 \in \mathbf{R}$ as below it is called a hinge function: $g(\mathbf{x}) = \max(\mathbf{w}_1^\top \mathbf{x} + b_1, \mathbf{w}_2^\top \mathbf{x} + b_2)$
- face: $(\mathbf{w}_1 - \mathbf{w}_2)^\top \mathbf{x} + (b_1 - b_2) = 0$
- representational power: $2\max(f, g) = f + g + |f - g|$
- k-Hinge function: $g(\mathbf{x}) = \max(\mathbf{w}_1^\top \mathbf{x} + b_1, ..., \mathbf{w}_k^\top \mathbf{x} + b_k)$
T (Wang and Sun, 2004): Every continuous piecewise linear function from $\mathbf{R}^n \to \mathbf{R}$ can be written as a signed sum of k-Hinges with $k \leq n+1$.
- exact representation (not approximation as ReLU, AVU).
- to represent k-Hinge with ReLU: need depth log. in k.
Polyhedral Function (Convex functions)
= convex and continuous piecewise linear functions
- f polyhedral $\leftrightarrow$ epi(f) is a polyhedral set
- epigraph of f (all points above the graph of f):
epi(f):= $\{(\mathbf{x}, t) \in \mathbf{R}^{n+1} : f(\mathbf{x}) \leq t\}$
- polyhedral set S: finite intersection of closed half-spaces
$S = \{\mathbf{x} \in \mathbf{R}^n : \mathbf{w}_j^\top \mathbf{x} + b_j \geq 0, j = 1, ...r\}$
Max-Representation of Polyhedral Functions For every polyhedral f, there exists $\mathcal{A} \subset \mathbf{R}^{n+1}, |\mathcal{A}| = m$ s.t. $f(x) = \max_{(w,b) \in \mathcal{A}} \{\mathbf{w}^\top \mathbf{x} + b\}$ - each polyhedral f can be repres. as max. of supp. hyperplanes - linear functions in $\mathcal{A}$ describe supporting hyperplanes of epi(f). **Continuous Piecewise Linear Functions** T(Wang, 2004): Every cont. piecewise linear fcn f can be written as the difference of two polyhedral fcns; with finite $\mathcal{A}^+, \mathcal{A}^-$. $f(x) = \max_{(w,b) \in \mathcal{A}^+} \{\mathbf{w}^\top \mathbf{x} + b\} - \max_{(w,b) \in \mathcal{A}^-} \{\mathbf{w}^\top \mathbf{x} + b\}$ **2 × Maxout = Allout** T(Goodfellow, 2013): Maxout networks with two maxout units are universal function approximators.
Sigmoid Fcns: Approximation Theorem
T(Lencho, Lin, Pinkus, Schocken, 1993): Let $\sigma \in C^\infty(\mathbf{R})$, not polynomial, then $\mathcal{H}_\sigma^1$ is dense in $C(\mathbf{R})$; i.e. results in dense function approximation. C: MLPs with one hidden layer and any non-polynomial, smooth activation function are universal function approximators. L: MLPs with one hidden layer and a poly. activation fcn are **not** univ. fcn approximators.
Sigmoidal MLP: Approximation Guarantees
T(Barron, 1993): For every $F: \mathbf{R}^n \to \mathbf{R}$ with absolutely continuous Fourier transform and for every m there is a function of the form $\tilde{f}_m$ such that $\int_{B_r} (f(\mathbf{x} - \tilde{f}_m(\mathbf{x}))^2 \mu(d\mathbf{x}) \leq O(1/m)$ where $B_r = \{\mathbf{x} \in \mathbf{R}^n : \|\mathbf{x}\| \leq r\}$ and μ is any probability measure on B_r. Residual bound doesn't depend on n.
Feedforward networks: A set of computational units arranged in a DAG (directed acyclic graph). **Loss function:** A non-negative function $l: \mathcal{Y} \times \mathcal{Y} \to \mathbf{R}_{\geq 0}$, $(y^*, y) \to l(\mathbf{y}^*, \mathbf{y})$, output space: $\mathcal{Y}$, squared error: $\mathcal{Y} = \mathbf{R}^m, l(\mathbf{y}^*, \mathbf{y}) = \|\mathbf{y}^* - \mathbf{y}\|_2^2 = \sum_{i=1}^m (y_i^* - y_i)^2$, class. error: $\mathcal{Y} = [1:m], l(\mathbf{y}^*, \mathbf{y}) = 1 - \delta_{\mathbf{y}^*\mathbf{y}}$ **Expected risk:** Assume inputs and outputs are governed by a distribution $p(\mathbf{x}, \mathbf{y})$ over $\mathcal{X} \times \mathcal{Y}, \mathcal{X} \subset \mathbf{R}^n$. The expected risk of F is given by $J^*(F) = \mathbf{E}_{x,y}[l(\mathbf{y}, F(\mathbf{x})]$ **Training risk:** Assume we have a random sample of N input-output pairs $\mathcal{S}_N := \{(\mathbf{x}_i, \mathbf{y}_i)$ iid distr. $\{p : i = 1, ..., N\}$. The training risk of F on a training sample is $J(F; \mathcal{S}_N) = \frac{1}{N} \sum_{i=1}^N l(y_i, F(x_i))$. training risk is the expected risk under the empirial distribution induced by the sample $\mathcal{S}_N$. **Empirical risk minimizer:** $\hat{F}(\mathcal{S}_N) = \arg\min_{F \in \mathcal{F}} J(F; \mathcal{S}_N)$ with parameter $\hat{\theta}(\mathcal{S}_N)$. **Generalized linear models:** predict the mean of the output distribution: $\mathbf{E}[y|x] = \sigma(\mathbf{w}^\top \mathbf{x})$ **Log.-Likelihood:** $J(\theta; (\mathbf{x}, \mathbf{y})) = -\log p(\mathbf{y}|\mathbf{x}; \theta)$ **Logistic Log Likelihood** $J(F; (x, y)) = -\log p(y|z) = -\log \sigma((2y-1)z) = \zeta((1-2y)z)$ with $z := \bar{F}(\mathbf{x}) \in \mathbf{R}$, $\zeta = \log(1 + \exp(\cdot))$ (soft-plus) **Likelihood for logistic regression** $L = \prod_{i=1}^n p(x_i)^{y_i} (1 - p(x_i))^{1-y_i}$ **Multinomial Log Likelihood** $J(F; (\mathbf{x}, y)) = -\log p(y|\mathbf{x}; F) = -\log \left[\frac{e^{z_y}}{\sum_{i=1}^m e^{z_i}}\right] = -z_y + \log \sum_{i=1}^m \exp[z_i]$ with $\mathbf{z} := \bar{F}_i(\mathbf{x}) = \mathbf{w}_i^\top \mathbf{x} \in \mathbf{R}^m$
Backpropagation 1. perform a forward pass to compute activations for all units 2. compute gradient of J wrt. output layer activations 3. iteratively propagate activation gradient information from outputs to inputs 4. compute local gradients of activations wrt. weights **Jacobi matrix**
$$\mathbf{J}_F := \begin{bmatrix} \nabla^\top F_1 \\ \nabla^\top F_m \end{bmatrix} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \dots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_m}{\partial x_1} & \frac{\partial F_m}{\partial x_2} & \dots & \frac{\partial F_m}{\partial x_n} \end{bmatrix} \in \mathbf{R}^{m \times n}$$

Jacobi Matrix Chain Rule
Vector-valued funtions $G: \mathbf{R}^n \to \mathbf{R}^q, F: \mathbf{R}^q \to \mathbf{R}^m$ componentwise rule: $\frac{\partial (F \circ G)}{\partial x_i}|_{x=x_0} = \sum_{k=1}^q \frac{\partial F_j}{\partial z_k}|_{\mathbf{z}=G(\mathbf{x}_0)} \cdot \frac{\partial G_k}{\partial x_i}|_{x=x_0}$
Jacobi matrix chain rule (do not commute!)
$\mathbf{J}_{F\circ G}|_{x=x_0} = \mathbf{J}_F|_{z=G(x_0)} \cdot \mathbf{J}_G|_{x=x_0}$
Function Composition $G: \mathbf{R}^n \to \mathbf{R}^m, f: \mathbf{R}^m \to \mathbf{R}, f \circ G: \mathbf{R}^n \to \mathbf{R}$ in other words $\mathbf{R}^n \ni \mathbf{x} \stackrel{G}{\to} \mathbf{y} \stackrel{f}{\to} z \in \mathbf{R}$
Lemma(Chain rule for "activations"):
$\boxed{\nabla_\mathbf{x} z = \mathbf{J}_G^\top \nabla_\mathbf{y} z,}$ $\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial y_j}{\partial x_i}\frac{\partial z}{\partial y_j}$
z: output, x:input, for f don't need Jacobian
Activity Backpropagation $F = F^L \circ \cdots \circ F^1: \mathbf{R}^n \to \mathbf{R}^m$
$\mathbf{x} = \mathbf{x}^0 \stackrel{F^1}{\to} \mathbf{x}^1 \stackrel{F^2}{\to} \mathbf{x}^2 \to \cdots \stackrel{F^L}{\to} \mathbf{x}^L = \mathbf{y} \stackrel{J}{\to} J * \theta; \mathbf{y})$
$\nabla_\mathbf{x} J = \mathbf{J}_{F^1}^\top \cdots \mathbf{J}_{F^L}^\top \nabla_\mathbf{y} J$
Jacobian for ridge function
$\mathbf{x}^l = F^l(\mathbf{x}^{l-1} = \sigma(\mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l)$
$\frac{\partial x_i^l}{\partial x_j^{l-1}} = \sigma'(\langle \mathbf{w}_i^l, \mathbf{x}^{l-1}\rangle + b_i^l) W_{ij}^l := \bar{W}_{ij}^l$
Multinomial Logistic Regression
$\mathbf{z} := \bar{F}_i(\mathbf{x}) \in \mathbf{R}^m$
$J(F; (\mathbf{x}, y)) = -\log p(y|\mathbf{x}; F) = -\log\left[\frac{e^{z_y}}{\sum_{i=1}^m e^{z_i}}\right]$
$= -z_y + \log \sum_{i=1}^m \exp[z_i] = \log\left[1 + \sum_{i \neq y} \exp[z_i - z_y]\right]$
Multivariate logistic loss
$-\frac{\partial J(x, y^*)}{\partial z_y} = \frac{\partial}{\partial z_y}\left[z_{y^*} - \log \sum_i \exp[z_i]\right]$
$= \delta_{yy^*} - \frac{\exp[z_y]}{\sum_i \exp[z_i]} = \delta_{yy^*} - p(y|x)$ Quadratic loss (neg. gradient: in what direction want to move): $-\nabla_\mathbf{y} J(\mathbf{x}, \mathbf{y}^*) = -\nabla_\mathbf{y} \frac{1}{2}\|\mathbf{y}^* - \mathbf{y}\|^2 = \mathbf{y}^* - \mathbf{y}$
From Activations to Weights
$\frac{\partial J}{\partial W_{ij}^l} = \frac{\partial J}{\partial x_i^l}\frac{\partial x_i^l}{\partial W_{ij}^l} = \frac{\partial J}{\partial x_i^l} \cdot \sigma'(\langle \mathbf{w}_i^l, \mathbf{x}^{l-1}\rangle + b_i^l) \cdot x_j^{l-1}$
$\frac{\partial J}{\partial b_i^l} = \frac{\partial J}{\partial x_i^l}\frac{\partial x_i^l}{\partial b_i^l} = \frac{\partial J}{\partial x_i^l} \cdot \sigma'(\langle \mathbf{w}_i^l, \mathbf{x}^{l-1}\rangle + b_i^l) \cdot 1$
Optimization for Deep Networks
Gradient Descent: $\theta(t+1) = \theta(t) - \eta \nabla_\theta \mathcal{R}$, cont.: $\dot\theta = -\nabla_\theta \mathcal{R}$; Convex objective $\mathcal{R}$, $\mathcal{R}$ has L-Lipschitz-continuous gradients: $\mathcal{R}(\theta(t)) - \mathcal{R}^* \leq \frac{2L}{t+1}\|\theta(0) - \theta^*\|^2 \in \mathbf{O}(t^{-1})$ **Analy.: Gradient Descent:** $\mathcal{R}$ is μ-strongly convex in θ: $\mathcal{R}(\theta(t)) - \mathcal{R}^* \leq (1 - \frac{\mu}{L})^t \mathcal{R}(\theta(t)) - \mathcal{R}^*)$; exponential convergence ("linear rate"); rate depends adversely on condition number L/μ; Lower bound (general case): $\mathbf{O}(t^{-2})$, achieved by Neterov acceleration.
Curvature of objective function
$\mathcal{R}(\theta - \eta\nabla\mathcal{R}) \stackrel{Taylor}{\approx} \mathcal{R}(\theta) - \eta\|\nabla\mathcal{R}\|^2 + \frac{\eta^2}{2}\nabla\mathcal{R}^\top \mathbf{H} \nabla\mathcal{R}$ with $\nabla\mathcal{R}^\top \mathbf{H}\nabla\mathcal{R} = \|\nabla\mathcal{R}\|_\mathbf{H}^2$, $\mathbf{H} = \nabla^2\mathcal{R}$
ill-conditioning: $\frac{\eta}{2}\|\nabla\mathcal{R}\|_\mathbf{H}^2 \gtrsim \|\nabla\mathcal{R}\|^2$
Least-Squares: Single Layer Linear Network
$\mathcal{R}(\mathbf{A}) = \mathbf{E}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|^2 = \mathrm{Tr}\,\mathbf{E}[(\mathbf{y} - \mathbf{A}\mathbf{x})(\mathbf{y} - \mathbf{A}\mathbf{x})^\top]$
$= \mathrm{Tr}\,\mathbf{E}[\mathbf{y}\mathbf{y}^\top] + \mathrm{Tr}(\mathbf{A}\mathbf{E}[\mathbf{x}\mathbf{x}^\top]\mathbf{A}^\top) - 2\,\mathrm{Tr}(\mathbf{A}\mathbf{E}[\mathbf{x}\mathbf{y}^\top])$
$\nabla_\mathbf{A}\mathcal{R} = \nabla\mathbf{A}\,\mathrm{Tr}(\mathbf{A}\mathbf{A}^\top) - 2\nabla\mathbf{A}\,\mathrm{Tr}(\mathbf{A}\mathbf{\Gamma}^\top) = 2(\mathbf{A} - \mathbf{\Gamma})$
Least-Squares: Two Layer Linear Network
$(\mathbf{A} = \mathbf{Q}\mathbf{W})$ $\mathcal{R}(\mathbf{Q}, \mathbf{W}) = \text{const.} + \mathrm{Tr}(\mathbf{Q}\mathbf{W} \cdot (\mathbf{Q}\mathbf{W})^\top) - 2\,\mathrm{Tr}(\mathbf{Q}\mathbf{W} \cdot \mathbf{\Gamma}^\top)$; $\frac{1}{2}\nabla_\mathbf{Q}\mathcal{R} = (\mathbf{Q}\mathbf{W})\mathbf{Q}^\top - \mathbf{\Gamma}\mathbf{W}^\top = (\mathbf{A} - \mathbf{\Gamma})\mathbf{W}^\top \in \mathbf{R}^{m\times k}$; $\frac{1}{2}\nabla_\mathbf{W}\mathcal{R} = \mathbf{Q}\top(\mathbf{A} - \mathbf{\Gamma}) \in \mathbf{R}^{k\times n}$; $\frac{1}{2}\nabla_{\tilde{\mathbf{Q}}}\mathcal{R} = \mathbf{U}\mathbf{U}^\top(\tilde{\mathbf{Q}}\tilde{\mathbf{W}} - \mathbf{\Sigma})\mathbf{V}\mathbf{V}^\top\tilde{\mathbf{W}}^\top = (\tilde{\mathbf{Q}}\tilde{\mathbf{W}} - \mathbf{\Sigma})\tilde{\mathbf{W}}^\top$; $\frac{1}{2}\nabla_{\tilde{\mathbf{W}}}\mathcal{R} = \tilde{\mathbf{Q}}^\top(\tilde{\mathbf{Q}}\tilde{\mathbf{W}} - \mathbf{\Sigma})$; $\frac{1}{2}\nabla_{\mathbf{q}_r}\mathcal{R} = (\mathbf{q}_\mathbf{r}^\top\mathbf{w}_\mathbf{r} - \sigma_r)\mathbf{w}_\mathbf{r} + \sum_{\mathbf{s}\neq\mathbf{r}}(\mathbf{q}_\mathbf{r}^\top\mathbf{w}_\mathbf{s})\mathbf{w}_\mathbf{s}$; $\frac{1}{2}\nabla_{\mathbf{w}_r}\mathcal{R} = (\mathbf{q}_\mathbf{r}^\top\mathbf{w}_\mathbf{r} - \sigma_r)\mathbf{q}_\mathbf{r} + \sum_{\mathbf{s}\neq\mathbf{r}}(\mathbf{q}_\mathbf{s}^\top\mathbf{w}_\mathbf{r})\mathbf{q}_\mathbf{s}$; Equivalent energy function: $\tilde{\mathcal{R}}(\tilde{\mathbf{Q}}, \tilde{\mathbf{W}}) = \sum_\mathbf{r}(\mathbf{q}_\mathbf{r}^\top\mathbf{w}_\mathbf{r} - \sigma_r)^2 + \sigma_{\mathbf{s}\neq\mathbf{r}}(\mathbf{q}_\mathbf{s}^\top\mathbf{w}_\mathbf{r})^2$; cooperation: same input-output mode weight vector align; competition: different mode weight vectors are decoupled
Stochastic Gradient Descent Choose update direction $\mathbf{v}$ at random such that $\mathbf{E}[\mathbf{v}] = -\nabla\mathcal{R}$; $\mathcal{S}_K \subseteq \mathcal{S}_N, K \leq N$; $\mathbf{E}\mathcal{R}(\mathcal{S}_K) = \mathcal{R}(\mathcal{S}_N) \Rightarrow \mathbf{E}\nabla\mathcal{R}(\mathcal{S}_K) = \nabla\mathcal{R}(\mathcal{S}_N)$ Update step: $\theta(t+1) = \theta(t) - \eta\nabla\mathcal{R}(t), \mathcal{R} := \mathcal{R}(\mathcal{S}_K(t))$; Conv. to optimum: convex or strongly convex objective, Lipschitz gradients, $\sum_{t=1}^\infty \eta^2(t) < \infty, \sum_{t=1}^\infty \eta(t) = \infty$, e.g. $\eta(t) = Ct^{-\alpha}, \frac{1}{2} < \alpha \leq 1$, iterate (Polyak) averaging; Conv. rates: strongly-convex case: $\mathcal{O}(1/t)$, non-strongly convex: $\mathcal{O}(1/\sqrt{t})$
Heavy Ball Method Update: $\theta(t+1) = \theta(t) - \eta\nabla\mathcal{R} + \alpha(\theta(t) - \theta(t-1)), \alpha \in [0;1)$; Gradients are constant $\Rightarrow$ update steps are boosted by $1/(1-\alpha)$: $\eta\|\nabla J\|(1 + \alpha + \alpha^2 + \alpha^3 + \ldots) \to \frac{\eta\|\nabla J\|}{1-\alpha}$, $\alpha = 0.9 \Rightarrow 10\times$. Accelerate for high curvature, small but consistent gradient, or noisy gradients. Solve poor conditioning of Hessian matrix and variance in stochastic gradient. hlgrayAdaGrad Consider the entire history of gradients: gradient matrix: $\theta \in \mathbf{R}^d, \mathbf{G} \in \mathbf{R}^{d\times t_{max}}, g_{it} = \frac{\partial\mathcal{R}(t)}{\partial\theta_i}|_{\theta=\theta(t)}$ Learning rate decays faster for weights that have seen significant updates. Compute (partial) row sums of $\mathbf{G}$: $\gamma_i^2(t) := \sum_{s=1}^t g_{is}^2$ Adapt learning rate per parameter $\theta_i(t+1) = \theta_i(t) - \frac{\eta}{\delta + \gamma_i(t)}\nabla\mathcal{R}(t), \delta > 0$ (small) Non-convex variant: **RMSprop** (moving average, expon. weighted): $\gamma_i^2(t) := \sum_{s=1}^t \rho^{t-1} g_{is}^2, \rho < 1$ **BFGS/LBFGS** (advantages of Newton, without comp. burden) Newton method: $\theta(t+1) = \theta(t) - (\nabla^2\mathcal{R})^{-1}\nabla\mathcal{R}|_{\theta=\theta(t)}$ BFGS: $(\nabla^2\mathcal{R})^{-1} \approx \mathbf{M}(t)$: $\theta(t+1) = \theta(t) - M(t)\nabla\mathcal{R}|_{\theta=\theta(t)}$ where $\mathbf{M}(t+1) = \mathbf{M}(t)$ + rank one update with $\nabla\mathcal{R}$ via line search; LBFGS: Reduce memory footprint with $\tilde{\mathbf{M}} \approx \mathbf{M}(t)$ with $k \approx 30$ rank one matrices (pairs of vectors), mini-batch
Optimization Heuristics **Polyak averaging** (Average over iterates, red. fluctuation): Linear (convex case): $\bar\theta(t) = \frac{1}{t}\sum_{s=1}^t \theta(s)$; Running (non-convex): $\bar\theta(t) = \alpha\theta(t-1) + (1-\alpha)\theta(t), \alpha \in [0;1)$ **Batch normalization** Hard to find suitable learning rate for all layers (strong dependencies between weights in layers exist) $\Rightarrow$ normalize the layer activations + backpropagate through normalizations; Fix layer l, fix set of example $I \subseteq [1:N]$: $\mu_j^l := \frac{1}{|I|}\sum_{i\in I}(F_j^l \circ \ldots \circ F^1)(\mathbf{x}[i]) \in \mathbf{R}^{m_l}$; $\sigma_j^l := \sqrt{\delta + \frac{1}{|I|}\sum_i (F_j^l \circ \ldots \circ F^1)(\mathbf{x}[i]) - \mu_j)^2}, \delta > 0$; Normalized activities: $\tilde{\mathbf{x}}_j^l := \frac{\mathbf{x}_j^l - \mu_j}{\sigma_j}$; Regain representational power: $\tilde{\tilde{\mathbf{x}}}_j^l = \alpha_j\tilde{\mathbf{x}}_j^l + \beta_j$ **Batch normalization (simplified)** (1) **Input**: mini-batch of real values $X = (x_1, .., x_n) \in \mathbf{R}^n$ (2) **Learnable parameters**: $\gamma, \beta \in \mathbf{R}$ (3) **Output**: $Y = (y_1, .., y_n) \in \mathbf{R}^n$, where we have (a) Mini-batch mean: $\mu := \frac{1}{n}\sum_i x_i$ (b) Mini-batch variance: $\sigma^2 := \frac{1}{n}\sum_i (x_i - \mu)^2$ (c) Norm. mini-batch (matrix form): $\hat{X} := (X - \mu)/(\sqrt{\sigma^2 + \epsilon})$ (d) Output: $Y = BN_{\gamma,\beta}(X) := \gamma\hat{X} + \beta$
Regularization Any aspect of a learning algorithm that is intended to lower the generalization error but not the training error. E.g.: Informed regularization: encode specific prior knowledge. simplicity bias: preference for simpler models (Occam's razor). Data augmentation and cross-task learning. Model averaging, e.g. ensemble methods, drop-out. **Norm-based Regularization** Standard regularization: $\mathcal{R}_\Omega(\theta; \mathcal{S}) = \mathcal{R}(\theta; \mathcal{S}) + \Omega(\theta)$; Deep networks: $\Omega(\theta) = \frac{1}{2}\sum_{l=1}^L \mu^l \|\mathbf{W}^l\|_F^2, \mu^l \geq 0$ **Weight decay** Regularization based on L_2-norm is also called weight decay: $(\partial\Omega)/(\partial W_{ij}^l) = \mu^l w_{ij}^l$ Weights in l-th layer get pulled towards zero with "gain" μ^l. Naturally favors weights of small magnitude. GD update: $\theta(t+1) = (1-\mu) \cdot \theta(t) - \eta \cdot \nabla_\theta\mathcal{R}$ **Weight decay (Analysis)** Taylor: $\mathcal{R}(\theta) \approx \mathcal{R}(\theta^*) + \frac{1}{2}(\theta - \theta^*)^\top \mathbf{H}(\theta - \theta^*)$, where $\mathbf{H}_\mathcal{R}$ is the Hessian of $\mathcal{R}$: $\mathbf{H}_\mathcal{R} = \left(\frac{\partial^2\mathcal{R}}{\partial\theta_i\partial\theta_j}\right)$, and $\mathbf{H} := \mathbf{H}_\mathcal{R}|_{\theta=\theta^*} \Rightarrow \nabla_\theta\mathcal{R}_\Omega \stackrel{!}{=} 0$ with $\mathbf{H} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top, \mathbf{\Lambda} = diag(\lambda_1, \ldots, \lambda_d) \Rightarrow \theta = \mathbf{Q}(\Lambda + \mu I)^{-1}\Lambda Q^\top\theta^* \Rightarrow$ Along directions in parameter space with large eigenvalues of $\mathbf{H}$ (i.e. $\lambda_i \gg \mu$): vanishing effect. Along directions in parameter space with small eigenvalues of $\mathbf{H}$ (i.e. $\lambda_i \ll \mu$): shrunk to nearly zero magnitude. Linear regression: $\mathcal{R}_\Omega(\theta) = \frac{1}{2}(\mathbf{X}\theta - y)^\top(\mathbf{X}\theta - y) + \frac{\mu}{2}\|\theta\|^2 \Rightarrow \theta = (\mathbf{X}^\top\mathbf{X} + \mu\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}$ **Regularization via Constrained Optimization** $\min_{\theta: \|\theta\| \leq r}\mathcal{R}(\theta)$ Optimization approach: Projected gradient descent: $\theta(t+1) = \Pi_r(\theta(t) - \eta\nabla\mathcal{R}), \Pi_r(\mathbf{v}) := \min\left\{1, \frac{r}{\|\mathbf{v}\|}\right\}\mathbf{v}$; Only active when weights are (too) large **Early Stopping** Stop learning after finite (small) number of iterations. E.g. use validation data to estimate risk. Stop when flat or worsening. Keep best solution. Taylor: $\nabla_\theta\mathcal{R}|_{\theta_0} \approx \nabla_\theta\mathcal{R}|_{\theta^*} + \mathbf{H}_{\nabla\mathcal{R}}|_{\theta^*}(\theta_0 - \theta^*) = \mathbf{H}(\theta_0 - \theta^*) \Rightarrow (\mathbf{I} - \eta\mathbf{\Lambda})^t \stackrel{!}{=} \mu(\mathbf{\Lambda} + \mu\mathbf{I})^{-1}$ which for $\eta\lambda_i \ll 1, \lambda_i \ll \mu$ can be achieved approximately via performing $t = \frac{1}{\eta\mu}$ steps. Early stopping = approximate L_2 regularizer.
Dataset Augmentation Generate virtual examples by applying transf. τ to each training example $(\mathbf{x}, \mathbf{y})$ to get $(\tau(\mathbf{x}), \mathbf{y})$): e.g. crop, resize, rotate, reflect, add transf. through PCA. - Inject noise: to inputs, to weights (regularizing effect), to targets (soft targets, robustness wrt. label errors) **Semi-supervised training** (more unlabeled data) - define generative model with corresponding log-likelihood - Opt. additive combination of supervised and unsupervised risk, sharing parameters **Multi-Task Learning** Share representations across tasks and learn jointly (i.e. minimize combined objective); typically: share low level representations, learn high level representations per task. **Ensemble Methods: Bagging** Ensemble method that combines model trained on bootstrap samples (BS); BS $\tilde{\mathcal{S}}_N^k$: sample N times from $\mathcal{S}_N$ with replacement for $k = 1, .., K$; train model on $\tilde{\mathcal{S}}_N^k \to \theta^k$.

Prediction: average model output probabilities $p(\mathbf{y}|\mathbf{x};\theta^{\mathbf{k}})$: $p(\mathbf{y}|\mathbf{x}) = \frac{1}{K}\sum_{k=1}^K p(\mathbf{y}|\mathbf{x};\theta^k)$ **Dropout** Randomly "drop" subsets of units in network; keep probability π_i^l for unit i in layer l. Typically: $\pi_i^0 = 0.8, \pi_i^{l\geq 1} = 0.5$ **Dropout Ensembles** Dropout realizes an ensemble $p(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{Z}} p(\mathbf{Z})p(\mathbf{y}|\mathbf{x};\mathbf{Z})$, where $\mathbf{Z}$ denotes the binary "zeroing" mask. **Weight Rescaling** Approximation to geometrically averaged ensemble, to avoid 10-20× sampling blowup: Scale each weight w_{ij}^l by probability of unit j being active: $\tilde{w}_{ij}^l \leftarrow \pi_j^{l-1} w_{ij}^l$ Make sure, net input to unit i is calibrated, i.e. $\sum_j \tilde{w}_{ij}^l x_j \overset{!}{=} \mathbf{E}_{\mathbf{Z}} \sum_j z_j^{l-1} w_{ij}^l x_j = \sum_j \pi_j^{l-1} w_{ij}^l x_j$ **Convolutional Layers** Contin. Conv.: $(f*h)(u) := \int_{-\infty}^{\infty} h(u-t)f(t)dt = \int_{-\infty}^{\infty} f(u-t)h(t)dt$; Discr. Conv: $(f*h)[u] := \sum_{t=-\infty}^{\infty} f[t]h[u-t]$; $(F*G)[i,j] = \sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} F[i-k,j-l]\cdot G[k,l]$. T: Any linear, translation-invariant transformation T can be written as a convolution with a suitable h. Discr. Cross-Correlation (sliding inner product): $(f\star h)[u] := \sum_{t=-\infty}^{\infty} f[t]h[u+t]$; Border handling: same padding, valid padding; "Kernels" (across channels) form a linear map: $h: \mathbf{R}^{r^2\times d} \to \mathbf{R}^k$, where $r\times r$ is the window size (of convolution) and d is the depth (RGB).; Sub-sampling (strides) to reduce temporal/spatial resolution; Learn multiple convolution kernels (or filters) = multiple channels **Toeplitz matrix** A matrix $\mathbf{H}\in\mathbf{R}^{k\times n}$ is a Toeplitz matrix, if there exists $n+k-1$ numbers $c_l (l \in [-(n-1):(k-1)]\cup\mathbf{Z})$ s.t. $H_{i,j} = c_{i-j}$ **Backpropagation** Exploit structural sparseness in computing $\frac{\partial x_i^l}{\partial x_j^{l-1}}$; Receptive field of x_i^l : $\mathcal{I}_i^l := \{j : W_{ij}^l \neq 0\}$, where $\mathbf{W}^l$ is the Toeplitz matrix of the convolution; also $\frac{\partial x_i^l}{\partial x_j^{l-1}} = 0$ for $j\notin \mathcal{I}_i^l$; Weight sharing in computing $\frac{\mathcal{R}}{\partial h_j^l}$ where h_j^l is a kernel weight: $\frac{\mathcal{R}}{\partial h_j^l} = \sum_i \frac{\mathcal{R}}{\partial x_i^l}\frac{\partial x_i^l}{\partial h_j^l}$, weight is re-used for every unit within target layer ⇒ additive combination **CNNs (dimension)** CNN input: $H_1\times W_1\times C_1$, output of conv. layer with N filters, kernel size K, stride S and zero padding P: $H2 = (H_1 - K + 2P)/S + 1$, $W_2 = (W_1-K+2P)/S+1$, $C_2 = N$; $H2 = (H_1-K)/S+1$, $W_2 = (W_1-K)/S+1$, $C_2 = C_1$ **BProp:** Single input channel, single output channel; input $x\in\mathbf{R}^{d\times d}$, weights $w\in\mathbf{R}^{k\times k}$; output (before nonlin.) $y = x*w$; $\frac{\partial\mathcal{L}}{\partial w_{uv}} = \sum_i\sum_j \frac{\partial\mathcal{L}}{\partial y_{ij}}\frac{\partial y_{ij}}{\partial w_{uv}} = \sum_i\sum_j \partial\delta_{ij}\frac{\partial}{\partial w_{uv}}\sum_a\sum_b x_{i-a,j-b}w_{ab} = \sum_i\sum_j \partial\delta_{ij} x_{i-u,j-v} = \sum_i\sum_j \text{rot}_{180}(x_{u-i,v-j})\delta_{ij} = (\text{rot}_{180}(x)*\delta)_{u,v}$; $\frac{\partial\mathcal{L}}{\partial x_{uv}} = \sum_i\sum_j \frac{\partial\mathcal{L}}{\partial y_{ij}}\frac{\partial y_{ij}}{\partial x_{ij}} = \ldots = (\text{rot}_{180}(w)*\delta)_{u,v}$ **FFT** (compute convolutions faster, $\mathbf{O}(n\log n)$) $(f*h) = \mathcal{F}^{-1}((\mathcal{F}f))\cdot(\mathcal{F}h))$; pays off if many channels; small kernels ($m<\log n$): favor time/space domain **Convolutional Layers: Stages** Input to layer → Convolution stage: affine transform → Detector stage: nonlinearity (e.g. rectified linear) → pooling stage (locally combine activities) → next layer **Max Pooling** Maximum over a small "patch" of units: $1D: x_i^{max} = \max\{x_{i+k}: 0\leq k<r\}$; $2D: x_{ij}^{max} = \max\{x_{i+k,j+l}: 0\leq k,l<r\}$; $\mathcal{T}$-invariance through maximization $f_{\mathcal{T}}(\mathbf{x}) := \max_{\tau\in\mathcal{T}} f(\tau\mathbf{x})$; $f_{\mathcal{T}}$ is invariant under $\tau\in\mathcal{T}$: $f_{\mathcal{T}}(\tau\mathbf{x}) = \max_{\rho\in\mathcal{T}} f(\rho(\tau\mathbf{x})) = \max_{\rho\in\mathcal{T}} f((\rho\circ\tau)\mathbf{x}) = \max_{\sigma\in\mathcal{T}} f(\sigma\mathbf{x})$, as $\forall\sigma, \sigma=\rho\circ\tau$ with $\rho = \sigma\circ\tau^{-1}$

Conv. Networks for Natural Language Point-wise mutual information (pmi): $\text{pmi}(v,w) = \log\frac{p(v,w)}{p(v)p(w)} = \log\frac{p(v|w}{p(v)} = \mathbf{x}\top\mathbf{x}_w + \text{const.}$; Skip-gram objective: $\mathcal{L}(\theta;\mathbf{w}) = \sum_{(i,j)\in\mathcal{C}_{\mathcal{R}}}\log\left[\frac{p_\theta(w_i|w_j)}{p(w_i)}\right]$ with co-occurence index set $\mathcal{C}_R := \{(i,j)\in[1:T]^2 : 1\leq|i-j|\leq R\}$; Skip-gram model (soft-max): $\log p_\theta(v|w) = \mathbf{x}_v^\top\mathbf{z}_w - \log\sum_{u\in\mathcal{V}}\exp[\mathbf{x}_u^\top\mathbf{z}_w]$; Skip-gram model (negative sampling, logistic regression): $\mathcal{L}(\theta;\mathbf{w}) = \sum_{(i,j)\in\mathcal{C}_{\mathcal{R}}}\left[\log\sigma(\mathbf{x}_{w_i}^\top\mathbf{z}_{w_j}) + k\mathbf{E}_{v\sim p_n}[\log\sigma(-\mathbf{x}_{w_i}^\top\mathbf{z}_{w_j})]\right]$ **Recurrent Networks** Markov property; Time-invariance, share weights; $\bar{F}(h,x;\theta) := Wh+Ux+b$; $y = H(h;\theta), H(h;\theta) := \sigma(Vh+c)$

RNN-Backpropagation

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

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

with $\dot{\sigma}_i^t := \sigma'(\bar{F}_i(h^{t-1},x^t))$

MLP: $\nabla_{\mathbf{x}}\mathcal{R} = \mathbf{J}_{F^1}\ldots\mathbf{J}_{F^L}\nabla_{\mathbf{y}}\mathcal{R}$

RNN ($F^t=F$): $\nabla_{\mathbf{x}^t}\mathcal{R} = \left[\prod_{s=t+1}^T \mathbf{W}^\top\mathbf{S}(\mathbf{h}^s)\right]\cdot\mathbf{J}_H\nabla_{\mathbf{y}}\mathcal{R}$

where $\mathbf{S}(\mathbf{h}^s) = diag(\dot{\sigma}_1^s,\ldots,\dot{\sigma}_n^s)$;

RNN: Loss depends on all outputs loss $L=\sum_{t=1}^T L_t$, input $\mathbf{x}^t$, state $\mathbf{h}^t$: $\mathbf{h}^t = F(\mathbf{h}^{t-1},\mathbf{x}^t,\theta) = \alpha(\mathbf{W}\mathbf{h}^{t-1}+\mathbf{U}\mathbf{x}^t+\mathbf{b})$; $\frac{\partial L}{\partial\theta} = \sum_{t=1}^T\frac{\partial}{\partial\theta}L_t$; Sum over all the paths in the (unfolded) network leading from the parameters to the loss: $\frac{\partial L_t}{\partial\theta} = \sum_{k=1}^t\frac{\partial L_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial\theta}$; Expansion along a single path: $\frac{\partial h_t}{\partial h_k} = \prod_{i=k}^t\frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k}^t W^\top\text{diag}(\alpha'(\cdot))$

RNN: Loss depends only on last output $\bar{h}_t = F(x_t,x_{t-1};\theta)$, $h_t = \sigma(t)$, $y_t = G(h_t;\kappa)$, $L_T := L(y_T)+\frac{\lambda}{2}\|\theta\|_2^2$, $\frac{\partial L_T}{\partial\theta} = \frac{\partial L(y_T)}{\partial\theta}+\lambda = \frac{\partial L(y_T)}{\partial y_T}\frac{\partial G(h_T;\kappa)}{\partial h_T}\sum_{k=t}^T\frac{\partial h_T}{\partial h_t}\frac{\partial h_t}{\partial\theta} = \frac{\partial L(y_T)}{\partial y_T}\frac{\partial G(h_T;\kappa)}{\partial h_T}\sum_{k=t}^T\prod\frac{\partial h_T}{\partial h_t}\frac{\partial h_t}{\partial\theta}$ **Bi-Directional Recurrent Networks** $g^t = G(x^t,g^{t+1};\theta)$ **Deep Recurrent Networks** $h^{t,1} = F^1(h^{t-1,1},x^t;\theta)$; $h^{t,l} = F^l(h^{t-1,l},h^{t,l-1};\theta), l=2,\ldots,L$; $y^t = H(h^{t,L};\theta)$;

Memory Units / LSTM input processing (1), input g. (2), forget g. (3), output g. (4); $F^\kappa = \sigma\circ\bar{F}^\kappa$, $\bar{F}^\kappa = W^\kappa h^{t-1}+U^\kappa x^t+b^\kappa, \kappa\in\{1,2,3,4\}$; Next state: $h^t = F^3(\ldots)\circ h^{t-1}+F^2(\ldots)\circ F^1(\ldots)$; Output: $y^t = F^4(\ldots)\circ\tanh(h^t)$; LSTM = building unit for RNN. A common LSTM unit is composed of a cell, an input gate, an output gate, and a forget gate. $f_t = \sigma_g(W_fx_t+U_fh_{t-1}+b_f)\in R^h$ (forget gate); $i_t = \sigma_g(W_ix_t+U_ih_{t-1}+b_i)\in R^h$ (input gate); $o_t = \sigma_g(W_ox_t+U_oh_{t-1}+b_o)\in R^h$ (output gate); $c_t = f_t\circ c_{t-1}+i_t\circ\sigma_c(W_cx_t+U_ch_{t-1}+b_c)$ (cell state); $h_t = o_t\circ\sigma_h(c_t)$ (output vector), $W\in R^{h\times d}$, $U\in R^{h\times h}$ and $\in R^h$: weight matrices and bias vector parameters which need to be learned during training, $x_t\in R^d$: input vector to the LSTM unit. **LSTM with peephole connections** allow the gates to access the constant error carousel (CEC), whose activation is the cell state. h_{t-1} is not used, c_{t-1} is used instead in most places. $f_t = \sigma_g(W_fx_t+U_fc_{t-1}+b_f)$; $i_t = \sigma_g(W_ix_t+U_ic_{t-1}+b_i)$; $o_t = \sigma_g(W_ox_t+U_oc_{t-1}+b_o)$; $c_t = f_t\circ c_{t-1}+i_t\circ\sigma_c(W_cx_t+b_c)$; $h_t = o_t\circ\sigma_h(c_t)$ **Gated Memory Units** Memory state = output (lack output gate) $z_t = \sigma_g(W_zx_t+U_zh_{t-1}+b_z)$, $r_t = \sigma_g(W_rx_t+U_rh_{t-1}+b_r)$, $h_t = z_t\circ h_{t-1}+(1-z_t)\circ\sigma_h(W_hx_t+U_h(r_t\circ h_{t-1})+b_h)$

Differentiable Memory/Neural Turing Machine Able to learn to read from and write arbitrary content to memory cells. To read, they take a weighted average of many cells. $r\leftarrow\sum_i\alpha_iM_i, \alpha\geq 0, \sum_i\alpha_i = 1$; To write, they modify multiple cells by different amounts. $M_i\leftarrow(1-\beta_i)M_i+\beta_iw, \beta_i\in[0;1]$; Weights with nonzero derivatives (softmax) enables the functions controlling access to the memory to be optimized using GD. **Attention** Selectively attend to inputs or feature representations computed from inputs; select what is relevant from the past in hindsight **Recursive Networks** For a sequence of length τ, the depth can be reduced from τ to $O(\log\tau)$.

Autoencoders Linear auto-encoding (hidden layer $\mathbf{z}\in\mathbf{R}^m$, input dimension n, data points $i=1,\ldots,k$); $\mathbf{x}\in\mathbf{R}^n\overset{\mathbf{C}}{\to}\mathbf{z}\in\mathbf{R}^m (m\leq n)\overset{\mathbf{D}}{\to}\hat{\mathbf{x}}\in\mathbf{R}^n\overset{\mathcal{R}}{\to}\frac{1}{2}\|\mathbf{x}-\hat{\mathbf{x}}\|^2$. Optimal choice of $\mathbf{C}\in\mathbf{R}^{n\times m}$ and $\mathbf{D}\in\mathbf{R}^{m\times n}$ s.t. $\frac{1}{2k}\sum_{k=1}^k\|\mathbf{x}_i - \mathbf{DCx}_i\|^2\leftarrow\min$ **Eckart-Young Theorem** (for $m\leq\min(n,k)$) $\arg\min_{\hat{\mathbf{X}}:rank(\hat{\mathbf{X}})=m}\|\mathbf{X}-\hat{\mathbf{X}}\|_F^2 = \mathbf{U}_m\cdot diag(\sigma_1,\ldots,\sigma_m)\cdot\mathbf{V}_m^\top$. No linear auto-encoder with m hidden units can improve on SVD as rank$(CD)\leq m$. Given data $\mathbf{X} = \mathbf{U}diag(\sigma_1,..,\sigma_n)\mathbf{V}^\top$. The choice $\mathbf{C} = \mathbf{U}_m^\top$ and $\mathbf{D}=\mathbf{U}_m$ minimizes the squared reconstruction error of a two layer linear eauto-encoder with m hidden units. $\tilde{D}\tilde{C} = (U_mA^{-1})\cdot(AU_m^\top) = U_mU_m^\top$. Solutions restricted to $\mathbf{D}=\mathbf{C}^\top$ (weight-sharing) ⇒ $A^{-1}=A^\top$ (orthogonal) ⇒ mapping $x\to z$ only determined up to rotations. **Non-linear auto-encoder** $\min\mathbf{E}_{\mathbf{x}}[l(\mathbf{x},(H\circ G)(\mathbf{x})]$, e.g. $l(\mathbf{x},\hat{\mathbf{x}}) = \frac{1}{2}\|\mathbf{x}-\hat{\mathbf{x}}\|^2$; Encoder: $G = F_l\circ\cdots\circ F_1:\mathbf{R}^n\to\mathbf{R}^m, \mathbf{x}\to\mathbf{z}:=\mathbf{x}^l$; Decoder: $H = F_L\circ\cdots\circ F_{l+1}:\mathbf{R}^m\to\mathbf{R}^n, \mathbf{z}\to\mathbf{y}:=\hat{\mathbf{x}}$ **Denoising non-linear auto-encoder** $\min\mathbf{E}_{\mathbf{x}}\mathbf{E}_\eta[l(\mathbf{x},(H\circ G)(\mathbf{x}_\eta)]$ with $\mathbf{x}_\eta = \mathbf{x}+\eta$, $\eta\sim\mathcal{N}(\mathbf{0},\sigma^2\mathbf{I})$ **Denoising non-linear auto-encoder** code sparseness (sparse activity vector): $\Omega(\mathbf{z}) = \lambda\|\mathbf{z}\|_1$; contractive AE (stable wrt. changes in input): $\Omega(\mathbf{z}) = \lambda\|\frac{\partial\mathbf{z}}{\partial\mathbf{x}}\|_F^2$ **Factor Analysis** $x = \mu+Wz+\eta$, $z\sim\mathcal{N}(0,I), \eta\sim\mathcal{N}(0,\Sigma)$ then $x\sim\mathcal{N}(\mu,WW^\top+\Sigma)$, posterior $p(z|x) = \mathcal{N}(\mu_{z|x},\Sigma_{z|x})$

$\mu_{z|x} = (W^\top(WW^\top+\Sigma)^{-1}(x-\mu)$

$\Sigma_{z|x} = I - W\top(WW^\top+\Sigma)^{-1}W$

Pseudo-inverse: $W^\dagger := W^\top(WW^\top+\sigma^2I)^{-1}$ for $\sigma^2\to 0$

$W^\dagger = W^\top$ if W orthogonal columns

Moment generating functions MGF of random vector $\mathbf{x}$: $M_{\mathbf{x}}:\mathbf{R}^n\to\mathbf{R}, M_{\mathbf{x}} := \mathbf{E}_x\exp[\mathbf{t}^T\mathbf{x}]$

Uniqueness thereom: If M_x, M_y exist for RVs $\mathbf{x},\mathbf{y}$ and $M_x = M_y$ then (essentially) $p(\mathbf{x}) = p(\mathbf{y})$.

$\mathbf{E}[x_1^{k_1}\cdot x_n^{k_n}] = \frac{\partial^k}{\partial t_1^{k_1}\ldots\partial t_n^{k_n}}M_{\mathbf{x}}|_{t=0}$

$\mathbf{E}\mathbf{x}=\mu,\ \Sigma=\mathbf{E}(\mathbf{x}-\mu)(\mathbf{x}-\mu)^\top$

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

MGF: $M_x(\mathbf{t})=\exp[\mathbf{t}^\top\mu+\frac{1}{2}\mathbf{t}^\top\mathbf{\Sigma}\mathbf{t}]$

Deep Latent Gaussian Models (DLGMs) Noise variables $\mathbf{z}^l\overset{iid}{\sim}\mathcal{N}(\mathbf{0},\mathbf{I}),l=1,\ldots L$. Hidden activities (top-down: $\mathbf{h}^L\to\mathbf{h}^1$): $\mathbf{h}^L=\mathbf{W}^L\mathbf{z}^L$, $\mathbf{h}^l=F^l(\mathbf{h}^{l+1})+\mathbf{W}^l\mathbf{z}^l$. Hidden layer (conditional) distribution: $h|h^+\sim\mathcal{N}(F(\mathbf{h}^+),\mathbf{W}\mathbf{W}^\top)$

Jensen's inequality If g is a real-valued function that is μ-integrable, and if φ is a convex function, then: $\varphi(\int_\Omega g d\mu)\leq\int_\Omega\varphi\circ g d\mu$ OR $f(\mathbf{E}[x])\leq\mathbf{E}[f(x)]$

ELBO: Evidence lower BOund $\min -\log p_\theta(\mathbf{x})=-\log\int p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}=-\log\int q(\mathbf{x})\left[p_\theta(\mathbf{x}|\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}\right]d\mathbf{z}\leq-\int q(\mathbf{z})\log p_\theta(\mathbf{x}|\mathbf{z})d\mathbf{z}+\int q(\mathbf{z})\log\frac{q(\mathbf{z})}{p(\mathbf{z})}d\mathbf{z}=:\mathcal{F}(\theta,q;x)$ with $D_{KL}(q||p)=\int q(\mathbf{z})\log\frac{q(\mathbf{z})}{p(\mathbf{z})}d\mathbf{z}$ which we want to minimize optimal (often intractable): $q(z;x)=p(z|x)$ (posterior) (EM-alg.). Restrict q to (possibly) simpler familiy: Variational distributions, e.g.: $q(z;x)=\prod_{l=1}^L q(z^l;x),z^l\sim\mathcal{N}(\mu^l(x),\Sigma^l(x))$; need to learn functions $x\to\mu^l(x)$ (similar for cov.). q variational distr. approx. true intractable posterior $p(\mathbf{z}|\mathbf{x})$. Use another DNN: inference (recogn. network).

Recognition model: $\mathbf{x}\overset{\vartheta}{\to}(\mu^l,\mathbf{\Sigma}^l)_{l=1}^L\to q\sim\mathcal{N}(\ldots)$. Parametric form with parameters ϑ: generalization across x, aka amortized inference. KL-divergence can be thought of as regul.. Constr. prec. matrix: $\Sigma^{-1}=D(\text{diagonal})+uu^\top(\text{rank }1)$

Generative model opt. Assume for given x, $q(z|x)$ is fixed, opt. θ? Sample noise variables $(z^1,\ldots,z^L)\sim q(z|x)$. Perf. BP and SGD step for θ.

$\log p_\theta(\mathbf{x})\geq\mathbf{E}_q[\log p_\theta(\mathbf{x},\mathbf{z})]+KL(q(\mathbf{z})||p_\theta(\mathbf{z}))\overset{\max\mathbf{E}}{\longleftarrow}\theta,q.$

Stochastic Backpropagation Optimizing over q involves gradients of expectations! $\mathbf{z}\sim\mathcal{N}(\mu,\mathbf{\Sigma})$, f: smooth and integrable, then $\nabla_\mu\mathbf{E}f(\mathbf{z})]=\mathbf{E}[\nabla_\mathbf{z}f(\mathbf{z})]$, $\nabla_\mathbf{\Sigma}\mathbf{E}[f(\mathbf{z})]=\frac{1}{2}\mathbf{E}[\nabla_\mathbf{z}^2 f(\mathbf{z})]$
$\nabla_\mu\mathbf{E}f(z)=\int f(z)\nabla_\mu p(z)dz=-\int f(z)\nabla_z p(z)dz=\int\nabla_z f(z)p(z)dz-[f\cdot p]_{-\infty}^{\infty}=\mathbf{E}[\nabla_z f(z)]$.

DLGM top-down (generative), bottom-up (recognition). Forward pass: deterministic recognition, sampled generative. Backward pass: deterministic, but stoch. BP.

Density Estimation: Prescribed model: Use observer likelihoods and assume observation noise, Implicit models: Likelihood-free models

Partition Function $p(x;\theta)=\frac{1}{Z(\theta)}\tilde{p}(x;\theta)=\frac{1}{\sum_x\tilde{p}(x)}\tilde{p}(x;\theta)$
$\nabla_\theta\log p(x;\theta)=\nabla_\theta\log\tilde{p}(x;\theta)-\nabla_\theta\log Z(\theta)$
$\nabla_\theta\log Z(\theta)=\mathbf{E}_{x\sim p(x)}\nabla_\theta\log\tilde{p}(x)$

Score Matching (Alternative to MLE); avoids computing quantities related to the partition function; score $=\nabla_x\log p(x)$; Minimize the expected squared difference between the derivatives of the model's log density wrt the input and the derivatives of the data's log density wrt the input: $\psi_\theta:=\nabla\log\tilde{p}_\theta,\psi=\nabla\log p$, minimize $J(\theta)=\mathbf{E}\|\psi_\theta-\psi\|^2\Rightarrow J(\theta)\overset{\pm c}{=}\mathbf{E}\left[\sum_i\partial_i\psi_{\theta,i}-\frac{1}{2}\psi_{\theta,i}^2\right]$; Partition function Z is not a function of $x\Rightarrow\nabla_\mathbf{x}Z=0$; not applicable to models of discrete data; need to evaluate $\log\tilde{p}(x)$ and its derivatives; not compatible if only lower bound available

Noise Constrastive Estimation (NCE) The probability distribution estimated by the model is represented explicitly as $\log p_{\text{model}}(x)=\log\tilde{p}_{\text{model}}(x;\theta)+c$ where c approximates $-\log Z(\theta)$. Reduces density estimation to binary classification; $\tilde{p}(\mathbf{x},y=1)=\frac{1}{2}p_{\text{model}}(x),\tilde{p}(\mathbf{x},y=0)=\frac{1}{2}p_{\text{noise}}(\mathbf{x})$, y is a switch variable that determines whether we will generate x from the model or from the noise distribution
prob. classifier: $q_\theta=\frac{\alpha\tilde{p}_\theta}{\alpha\tilde{p}_\theta+p_n}$, $\alpha>0$, p_n: constrastive distr.
Bayes optimal if $\alpha\tilde{p}_\theta=p$; does not work with lower bound; estimator for θ consistent as long as p_n is dominating p; Generally not statistically efficient; much worse than Cramer-Rao bound if p_n very different from p.

Generative Adversarial Models (GAN)
Generator: gen. samples that are indistinguishable from real data. Train by minimizing logistic likelihood:
$l^*(\theta):=\mathbf{E}_{\tilde{p}_\theta}[y\ln q_\theta(\mathbf{x})+(1-y)\ln(1-q_\theta(\mathbf{x}))]$
Classification model: $q_\phi:\mathbf{x}\to[0;1],\phi\in\mathbf{\Phi}$
$l^*(\theta)\geq\sup_{\phi\in\Phi}l(\theta,\phi)$
$l(\theta,\phi):=\mathbf{E}_{\tilde{p}_\theta}[y\ln q_\phi(\mathbf{x})+(1-y)\ln(1-q_\phi(\mathbf{x}))]$
Optimizing GANs: saddle-point problem:
$\theta^*:=\arg\min_{\theta\in\Theta}\{\sup_{\phi\in\Phi}l(\theta,\phi)\}$
explicitly performing inner sup is impractical, iteratively update θ,ϕ with SGD, but may diverge. $\max_D\min_G V(G,D)$ with $V(G,D)=\mathbf{E}_{p_{data}(\mathbf{x})}\log D(\mathbf{x})+\mathbf{E}_{p_g(\mathbf{x})}\log(1-D(\mathbf{x}))$
$D^*(x)=\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$, G is optimal when $p_g(x)=p_{data}(x)$, equivalent to optimal discriminator producing 0.5 for all samples drawn from x. G is optimal when the discriminator is maximally confused and cannot distinguish real samples from fake ones.

$\mathbf{v}^\top\mathbf{w}=\sum_i v_i w_i=\mathrm{Tr}(\mathbf{v}\mathbf{w}^\top)$, $\mathrm{Tr}(\mathbf{A}+\mathbf{B})=\mathrm{Tr}(\mathbf{A})+\mathrm{Tr}(\mathbf{B})$, $\mathbf{E}\,\mathrm{Tr}(\mathbf{X})=\mathrm{Tr}\,\mathbf{E}(\mathbf{X})$, $\nabla_\mathbf{A}\mathrm{Tr}(\mathbf{A}\mathbf{A}^\top)=2\mathbf{A}$, $\nabla_\mathbf{A}\mathrm{Tr}(\mathbf{A}\mathbf{B})=\mathbf{B}^\top$, $\nabla_\mathbf{A}\mathrm{Tr}(\mathbf{S}\mathbf{A}^{-1})=-\mathbf{A}^{-1}\mathbf{S}\mathbf{A}^{-1}$, $\nabla_\mathbf{A}\log\det\mathbf{A}=\mathbf{A}^{-1}$
$\mathbf{v}^\top\mathbf{w}=\sum_i v_i w_i=\mathrm{Tr}(\mathbf{v}\mathbf{w}^\top)$, $\mathbf{E}\,\mathrm{Tr}(\mathbf{X})=\mathrm{Tr}\,\mathbf{E}(\mathbf{X})$, $\mathrm{Tr}(\mathbf{A})=\mathrm{Tr}(\mathbf{A}^\top)$, $\mathrm{Tr}(\mathbf{ABC})=\mathrm{Tr}(\mathbf{CAB})=\mathrm{Tr}(\mathbf{BCA})$

Differentiation rules

power: $\frac{d}{dx}x^n=nx^{n-1}$
product: $\frac{d}{dx}[f(x)\cdot g(x)]=f(x)\cdot\frac{d}{dx}g(x)+g(x)\cdot\frac{d}{dx}f(x)$
quotient: $\frac{d}{dx}\frac{f(x)}{g(x)}=\frac{g(x)\frac{d}{dx}f(x)-f(x)\frac{d}{dx}g(x)}{(g(x))^2}$
chain: $(f\circ g)'=(f'\circ g)\cdot g'$ or $\frac{dz}{dx}=\frac{dz}{dy}\frac{dy}{dx}=f'(y)g'(x)=f'(g(x))g'(x)$ or $\frac{d}{dx}[f(g(x))]=\frac{d}{dx}f(g(x))\cdot\frac{d}{dx}g(x)$
Schwarz-Theorem: $\frac{\partial^2 f(x,y)}{\partial x\partial y}=\frac{\partial^2 f(x,y)}{\partial y\partial x}$
Leibniz integral rule: $\frac{d}{dx}(\int_a^b f(x,t)dt)=\int_a^b\frac{\partial}{\partial x}f(x,t)dt$
$\frac{\partial\mathbf{x}^\top\mathbf{a}}{\partial\mathbf{x}}=\frac{\partial\mathbf{a}^\top\mathbf{x}}{\partial\mathbf{x}}=\mathbf{a}$, $\frac{\partial\mathbf{a}^\top\mathbf{X}\mathbf{b}}{\partial\mathbf{X}}=\mathbf{a}\mathbf{b}^\top$, $\frac{\partial\mathbf{a}^\top\mathbf{X}^\top\mathbf{b}}{\partial\mathbf{X}}=\mathbf{b}\mathbf{a}^\top$
$\frac{\partial\mathbf{b}^\top\mathbf{X}^\top\mathbf{X}\mathbf{c}}{\partial\mathbf{X}}=\mathbf{X}(\mathbf{b}\mathbf{c}^\top+\mathbf{c}\mathbf{b}^\top)$
$\frac{\partial(\mathbf{B}\mathbf{x}+\mathbf{b})^\top\mathbf{C}(\mathbf{D}\mathbf{x}+\mathbf{d})}{\partial\mathbf{x}}=\mathbf{B}^\top\mathbf{C}(\mathbf{D}\mathbf{x}+\mathbf{d})+\mathbf{D}^\top\mathbf{C}^\top(\mathbf{B}\mathbf{x}+\mathbf{b})$
$\frac{\partial\mathbf{x}^\top\mathbf{B}\mathbf{x}}{\partial\mathbf{x}}=(\mathbf{B}+\mathbf{B}^\top)\mathbf{x}$, $\frac{\mathbf{b}^\top\mathbf{X}^\top\mathbf{D}\mathbf{X}\mathbf{c}}{\partial\mathbf{X}}=\mathbf{D}^\top\mathbf{X}\mathbf{b}\mathbf{c}^\top+\mathbf{D}\mathbf{X}\mathbf{c}\mathbf{b}^\top$

Important functions $\sigma(x)=\frac{1}{1+e^{-x}}$, $\tanh(x)=\frac{e^x-e^{-x}}{e^x+e^{-x}}$, $\mathrm{softmax}(x)_i=\frac{e^{x_i}}{\sum_k e^{x_k}}$, Softplus: $\zeta(x)=\log(1+\exp(x))$, $\sigma'(x)=\sigma(x)(1-\sigma(x))$

Differences $f'(x)=\lim_{h\to0}\frac{f(x+h)-f(x)}{h}$
Finite differences: $\nabla f(w)=\frac{f(w+\epsilon)-f(w)}{\epsilon}+\mathcal{O}(\epsilon)$
Finite differences (2nd): $f''(x)=\frac{f(x+2h)-2f(x+h)+f(x)}{h^2}$
Central differences: $\nabla f(w)=\frac{f(w+\epsilon)-f(w-\epsilon)}{2\epsilon}$
Central differences (2nd): $f''(x)=\frac{f(x+h)-2f(x)+f(x-h)}{h^2}$
Taylor series: $f(x+h)=f(x)+f'(x)h+\frac{f''(x)}{2!}h^2+\frac{f'''(x)}{3!}h^3$..
T. exp.: $f(x)\approx f(a)+(x-a)^\top\nabla f(a)+\frac{1}{2!}(x-a)^\top\nabla^2 f(a)(x-a)$

Norms $\|x\|_p=\sqrt{\sum_i x_i^p}$, $\langle x,y\rangle=y^\top x=\sum x_i y_i$, $\|v\|=\sqrt{\langle v,v\rangle}$, $\|\mathbf{A}\|_F:=\sqrt{\sum_{i=1}^m\sum_{j=1}^n|a_{ij}|^2}=\sqrt{\mathrm{Tr}\,(\mathbf{A}\mathbf{A}^\top)}$

Probabilities $p(\mathbf{z}|\mathbf{x})=\frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})}$, $p(x)=\sum_z p(x|z)p(z)$, $p(b,c)=p(b|c)p(c)$; $\mu=\mathbf{E}(X)$, $Var(X)=\mathbf{E}[(X-\mu)^2=\mathbf{E}[X^2]-\mathbf{E}[X]^2$, $\mu=\int xf(x)dx$, $Var(X)=\int(x-\mu)^2 f(x)dx$
Entropy: $H(X)=-\sum_i P(x_i)\log_2 P(x_i)$

$H(X|Y)=-\sum p(x,y)\log\frac{p(x,y)}{p(y)}=\sum p(x,y)\log\left(\frac{p(x)}{p(x,y)}\right)=-\sum p(x,y)\log(p(x,y))+\sum p(x,y)\log(p(x))=H(X,Y)+\sum p(x)\log(p(x))=H(X,Y)-H(X)$
$H(Y|X)=0$ iff Y is completely determined by X.
$H(Y|X)=H(Y)$ iff Y and X are indep. RVs.
Bayes rule: $H(Y|X)=H(X|Y)-H(X)+H(Y)$
Proof: $H(Y|X)=H(X,Y)-H(X)$ and $H(X|Y)=H(Y,X)-H(Y)$, symmetry implies: $H(X,Y)=H(Y,X)$

$H(p,q)=E_p[-\log q]=H(p)+D_{KL}(p\|q)$
Discrete p,q: $H(p,q)=-\sum_x p(x)\log q(x)=\sum_x p(x)\log\frac{1}{q(x)}$
logistic function: $g(z)=1/(1+e^{-z})$
$q_{y=1}=\hat{y}=g(\mathbf{w}\cdot\mathbf{x})=1/(1+e^{-\mathbf{w}\cdot\mathbf{x}})$
$q_{y=0}=1-\hat{y}$
$H(p,q)=-\sum_i p_i\log q_i=-y\log\hat{y}-(1-y)\log(1-\hat{y})$

$D_{KL}=0$: expect same/similar behavior of two distributions.
$D_{KL}=-\sum_i P(i)\log\frac{Q(i)}{P(i)}=\sum_i P(i)\log\frac{P(i)}{Q(i)}$
$=-\sum p(x)\log q(x)+\sum p(x)\log p(x)=H(P,Q)-H(P)$
$D_{\mathrm{KL}}(q\|p)=\int q(z)\log\frac{q(z)}{p(z)}dz$

$\mathrm{JSD}(P\|Q)=\frac{1}{2}D_{KL}(P\|M)+\frac{1}{2}D_{KL}(Q\|M)$, $M=\frac{1}{2}(P+Q)$

ELBO: $\log p(x)=\log\int_z p(x,z)=\log\int_z p(x,z)\frac{q(z)}{q(z)}=\log\left(\mathbf{E}_q\left[\frac{p(x,z)}{q(z)}\right]\right)\geq\mathbf{E}_q\left[\log\frac{p(x,z)}{q(z)}\right]=\mathbf{E}_q[\log p(x,z)]+H(z)$

Loc. Min $\Rightarrow\nabla f(x)=0,\nabla^2 f$ psd. Assume $\nabla f(x^*)\neq0,\zeta=-\nabla f(x^*)/\|\nabla f(x^*)\|^2\Rightarrow f(x^*+\lambda\zeta)=f(x^*)+\lambda\nabla f(x^*)^\top\zeta+o(\lambda)=f(x^*)-\lambda+o(\lambda)\Rightarrow\exists\lambda^*>0$ st $f(x^*+\lambda\zeta)<f(x^*)$
Not psd $\Rightarrow\exists z:z^\top\nabla^2 f(x^*)z<0\Rightarrow f(x)\approx f(x^*)+(x-x^*)^\top\nabla f(x^*)+0.5(x-x^*)\nabla^2 f(x^*)(x-x^*)$ with $\nabla f(x^*)=0$, $\epsilon z=x-x^*\Rightarrow x=\epsilon z+x^*$ st $z^\top\nabla^2 f(x^*)z<0,\epsilon>0\Rightarrow f(x)\approx f(x^*)+0.5\epsilon^2 z^\top\nabla^2 f(x^*)z\Rightarrow f(x)<f(x^*)$