Machine Learning
& Inverse Problems

Inverse Problems	Statistical Learning

Inverse Problems

$$y = Af + w$$

$$A^\top y = (A^\top A) f + A^\top w$$

$$\overset{\text{def.}}{=} u \qquad \overset{\text{def.}}{=} C \qquad \overset{\text{def.}}{=} r$$

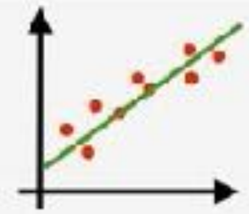
Regularized inversion:

$$\min_f \frac{1}{2}\|Af - y\|^2 + \lambda \|f\|^2$$

$$f_\lambda = (C + \lambda \mathrm{Id}_p)^{-1} u$$

Statistical Learning

$$y = Xf + \varepsilon$$



$$\frac{1}{n} X^\top y = \frac{1}{n}(X^\top X) f + \frac{1}{n} X^\top \varepsilon$$

$$\overset{\text{def.}}{=} u_n \qquad \overset{\text{def.}}{=} C_n \qquad \overset{\text{def.}}{=} r_n$$

$$n \to +\infty \qquad (x_i, y_i)_i \text{ i.i.d.}$$

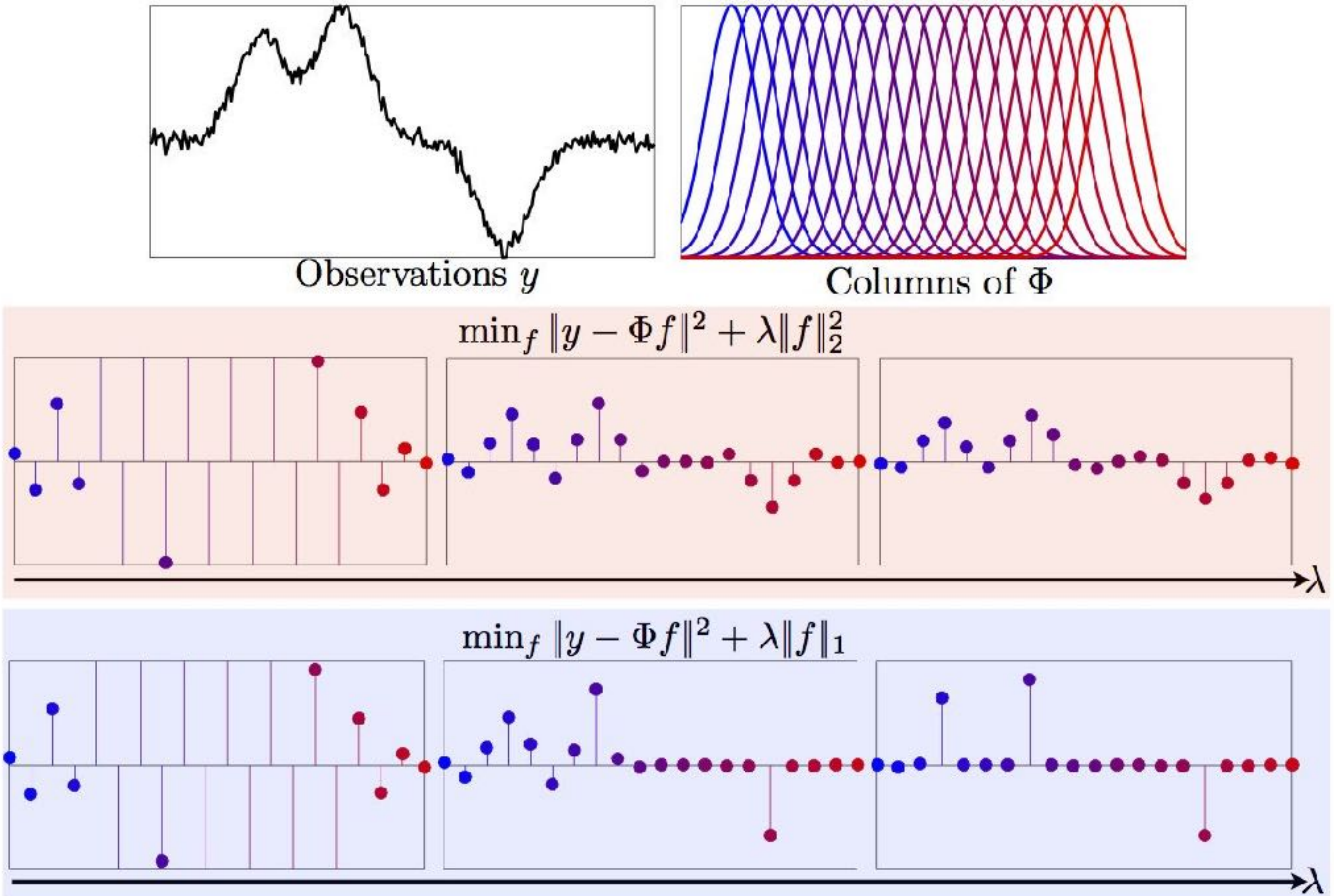
$$u = \mathbb{E}(yx) \qquad C = \mathbb{E}(xx^\top)$$

Empirical risk minimization:

$$\min_f \frac{1}{2n}\|Xf - y\|^2 + \lambda \|f\|^2$$

$$f_{\lambda,n} = (C_n + \lambda \mathrm{Id}_p)^{-1} u_n$$

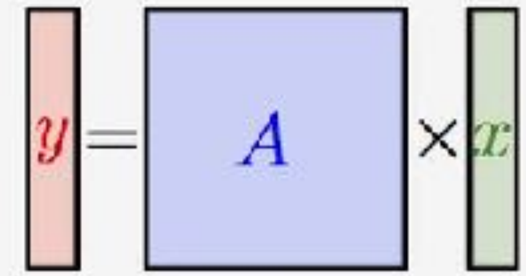
Inverse Problems		Statistical Learning
Exact covariance C	$\longleftrightarrow$	Noisy covariance C_n
Deterministic bounded noise r	$\longleftrightarrow$	Random noise r_n
Noise level $\varepsilon \overset{\text{def.}}{=} \|r\|$	$\longleftrightarrow$	Noise level $\|r_n\| \sim \varepsilon = n^{-\frac{1}{2}}$



Observations y
Columns of Φ
$\min_f \|y - \Phi f\|^2 + \lambda \|f\|_2^2$
λ
$\min_f \|y - \Phi f\|^2 + \lambda \|f\|_1$
λ

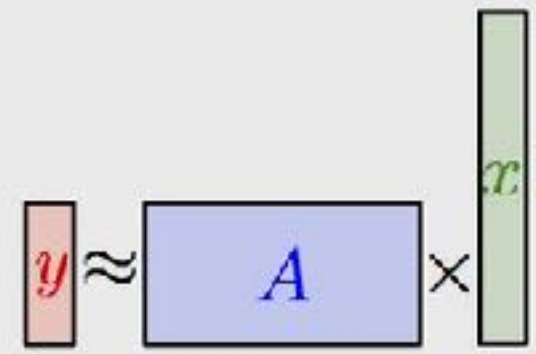
Solving $\quad y \underset{\approx}{=} A\, x \in \mathbb{R}^m \qquad A \in \mathbb{R}^{m \times n}$

Determined $(m = n)$: $\qquad x = A^{-1} y$



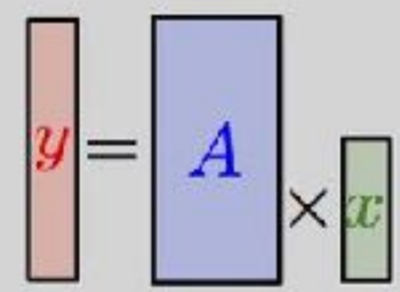
Over-determined $(m > n)$: $\qquad \min_x \|Ax - y\|^2$

$$x = (A^\top A)^{-1} A^\top y \overset{\text{def.}}{=} A^+ y$$



Under-determined $(m < n)$: $\qquad \min_x \{\|x\| \; ; \; Ax = y\}$

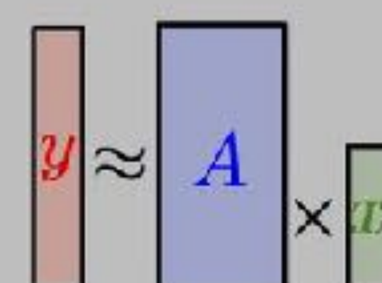
$$x = A^\top (A A^\top)^{-1} y \overset{\text{def.}}{=} A^+ y$$



A ill-posed and/or noise: $\qquad \min_x \|Ax - y\|^2 + \lambda \|x\|^2$

$$x = (A^\top A + \lambda \mathrm{Id}_n)^{-1} A^\top y \xrightarrow{\lambda \to 0} A^+ y$$

$$= A^\top (A A^\top + \lambda \mathrm{Id}_m)^{-1} y \quad \text{(Woodbury identity)}$$



Automatic Differentiation

How to compute $\nabla \ell_{x,y}(\theta)$? $\qquad \ell_{x,y}(\theta) \overset{\text{def.}}{=} L(f(x,\theta),y)$

Chain rule: $\nabla \ell_{x,y}(\theta) = [\partial f(x,\theta)]^\top (\nabla L(f(x,\theta),y))$

Linear $f(x,\theta) = \theta \times x$: $\partial f(x,\theta) = \theta$.

Non-linear $f(x,\theta)$: painful … but $\ell_{x,y}$ it is just a computer program.

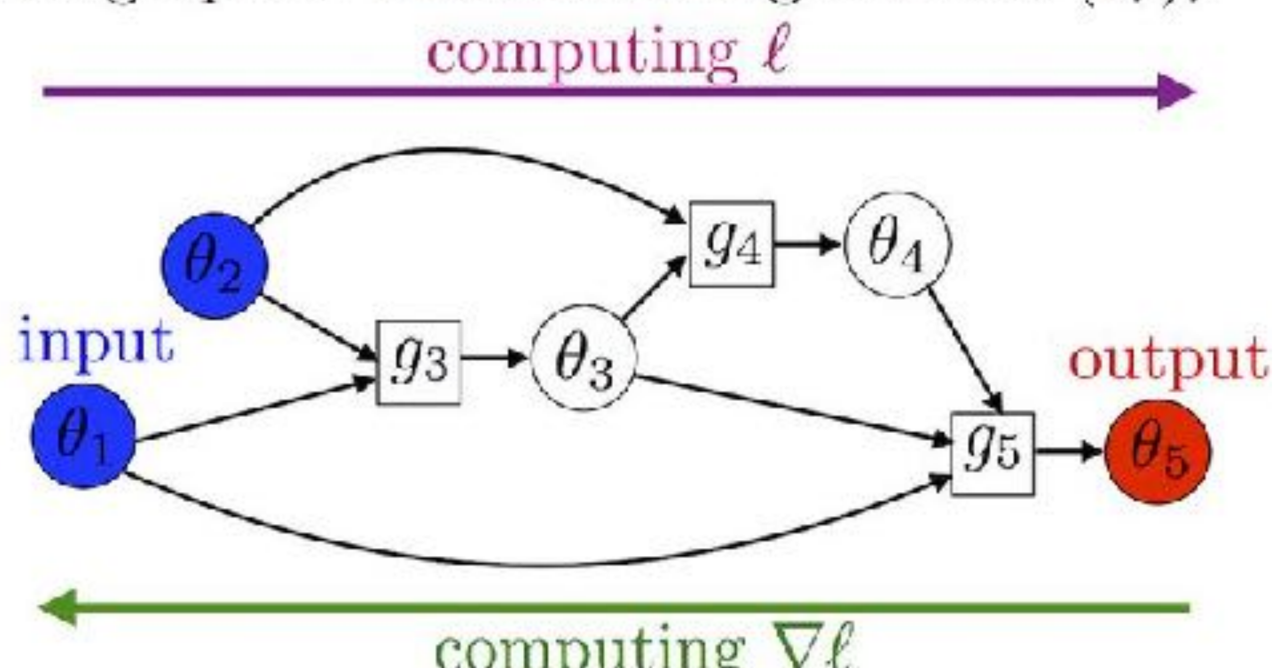
Computer program $\Leftrightarrow$ directed acyclic graph $\Leftrightarrow$ linear ordering of nodes $(\theta_r)_r$

forward

```
function ℓ(θ_1, ..., θ_M)
  for r = M+1, ..., R
  |  θ_r = g_r(θ_Parents(r))
  return θ_R
```

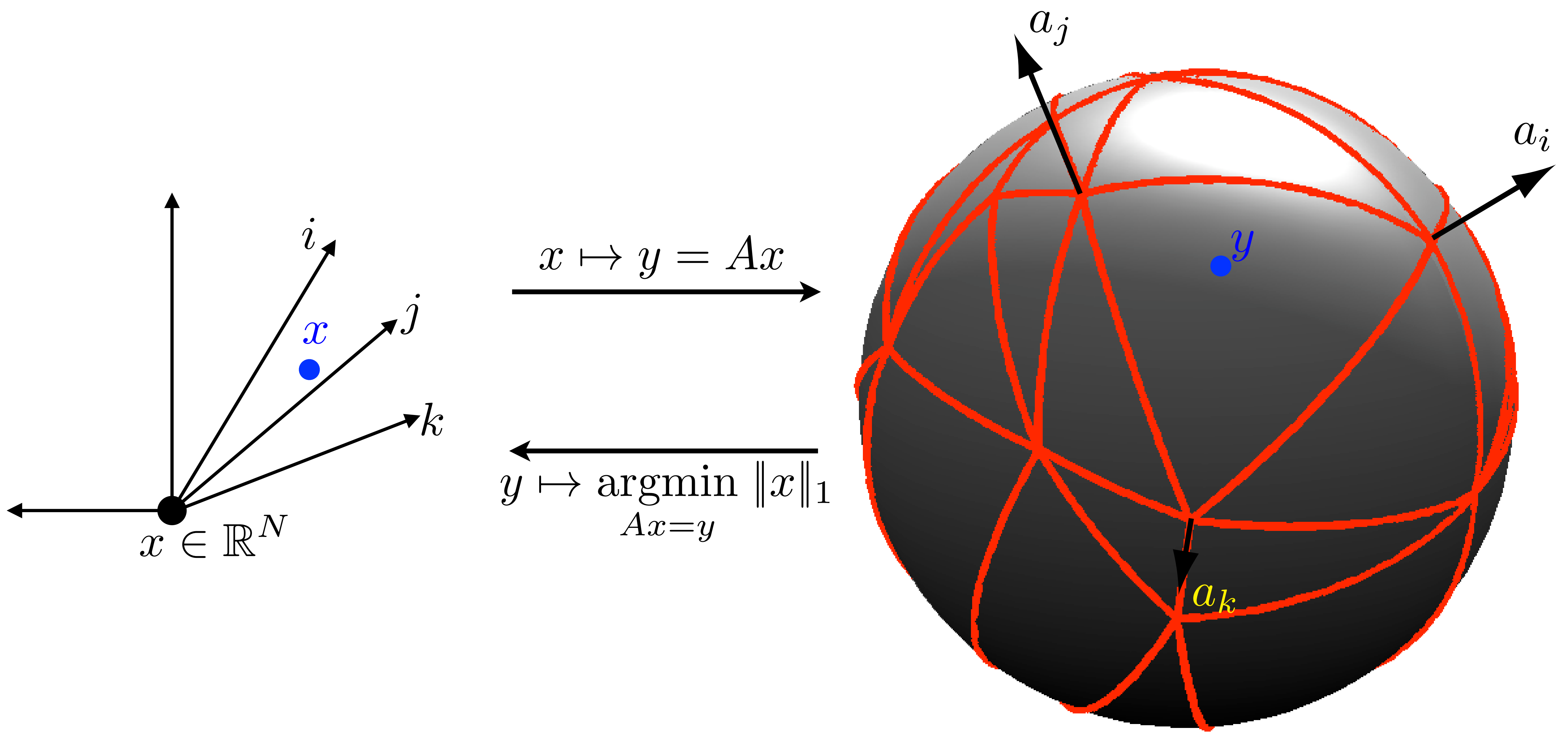
backward

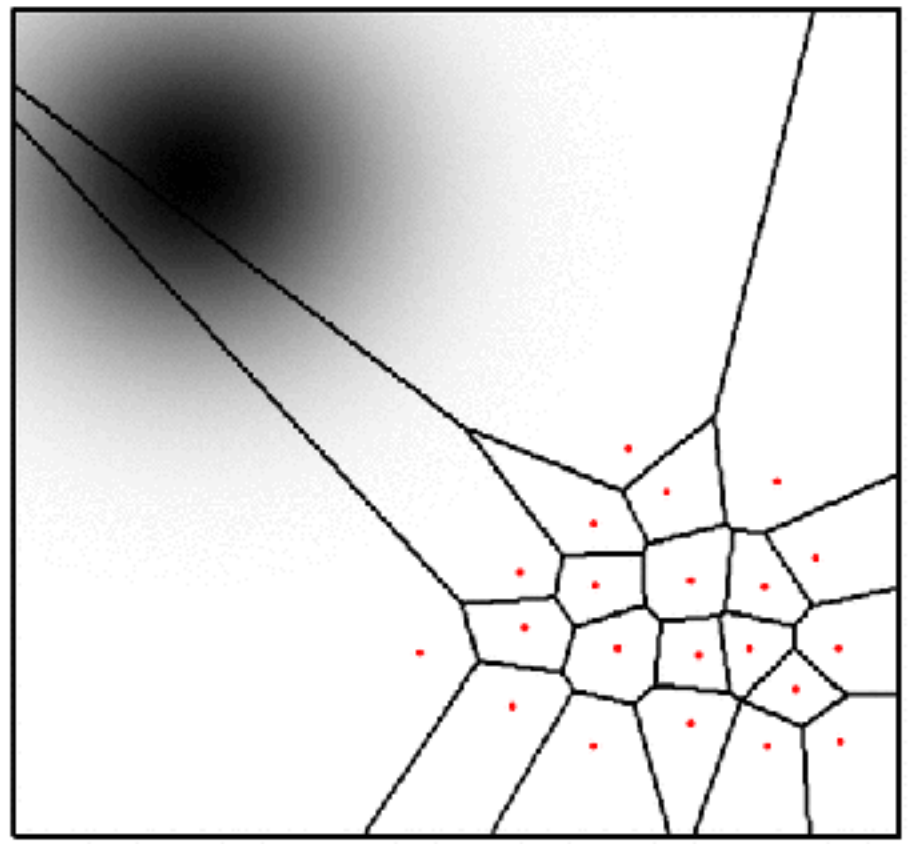
```
function ∇ℓ(θ_1, ..., θ_M)
  ∇_R ℓ = 1
  for r = R-1, ..., 1
  |  ∇_r ℓ = Σ_{s∈Child(r)} ∂_r g_s(θ) ∇_s ℓ
  return (∇_1 ℓ, ..., ∇_M ℓ)
```



"$\dfrac{\partial \ell}{\partial \theta_r} = \sum_{s \in \text{Child}(r)} \dfrac{\partial \ell}{\partial \theta_s} \dfrac{\partial \theta_s}{\partial \theta_r}$"

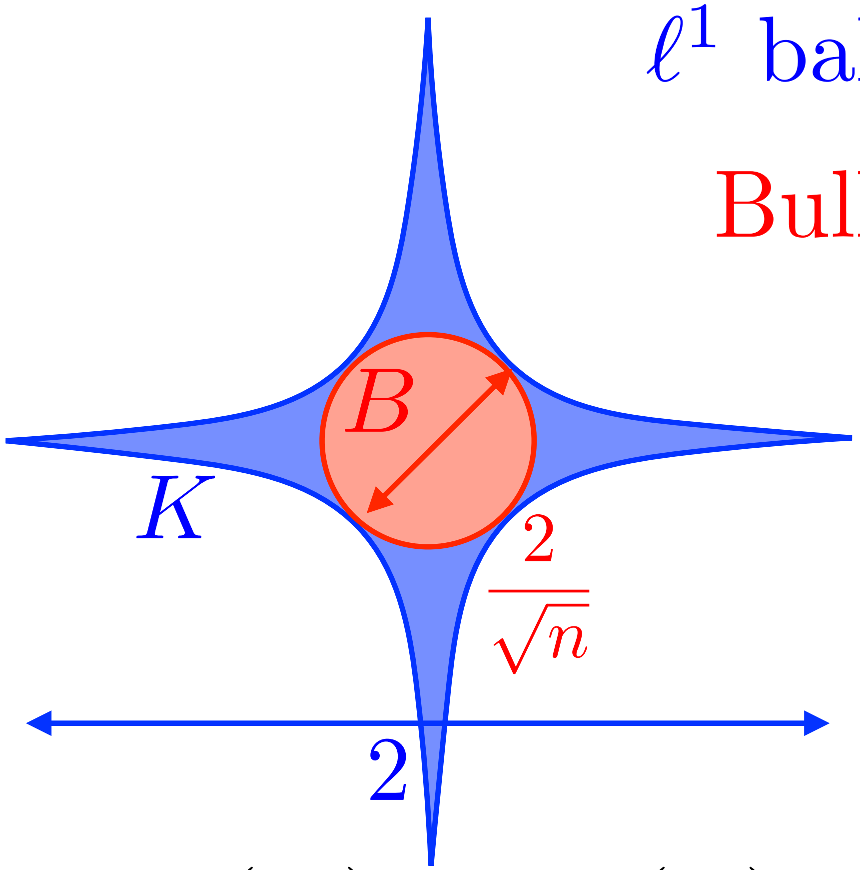
$\nabla_r \ell(\theta)$ $\quad \nabla_s \ell(\theta)$ $\quad \partial_r g_s(\theta)$


i
j
k
x
$x \in \mathbb{R}^N$
$x \mapsto y = Ax$
$y \mapsto \underset{Ax=y}{\operatorname{argmin}} \|x\|_1$
a_j
a_i
y
a_k



ℓ^1 ball: $K \stackrel{\text{def.}}{=} \{x \in \mathbb{R}^n \; ; \; \sum_{i=1}^n |x_i| \leqslant 1\}$

Bulk: $B \stackrel{\text{def.}}{=} \{x \in \mathbb{R}^n \; ; \; \sum_{i=1}^n |x_i|^2 \leqslant n^{-1}\}$



$\mathrm{Vol}(K) \sim \mathrm{Vol}(B)$

small n

large n

Setup: $\mathcal{E} : \mathbb{R}^n \to \mathbb{R}$ computable in K operations.

```
def ForwardNN(A,b,Z):
    X = []
    X.append(Z)
    for r in arange(0,R):
        X.append( rhoF( A[r].dot(X[r]) + tile(b[r],[1,Z.shape[1]]) ) )
    return X
```

Hypothesis: elementary operations $(a \times b, \log(a), \sqrt{a} \ldots)$ and their derivatives cost $O(1)$.

Question: What is the complexity of computing $\nabla\mathcal{E} : \mathbb{R}^n \to \mathbb{R}^n$?

Finite differences:

$$\nabla\mathcal{E}(\theta) \approx \frac{1}{\varepsilon}(\mathcal{E}(\theta + \varepsilon\delta_1) - \mathcal{E}(\theta), \ldots \mathcal{E}(\theta + \varepsilon\delta_1) - \mathcal{E}(\theta))$$

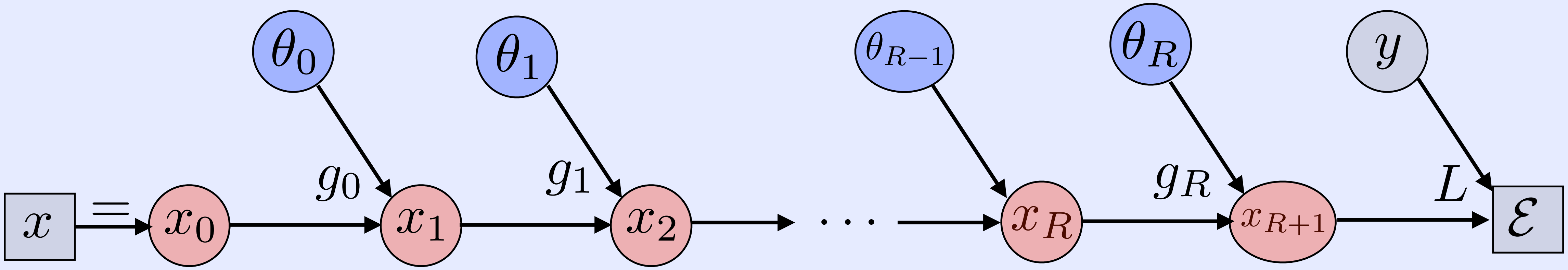
$K(n+1)$ operations, intractable for large n.

Theorem: there is an algorithm to compute $\nabla\mathcal{E}$ in $O(K)$ operations. [Seppo Linnainmaa, 1970]

This algorithm is reverse mode automatic differentiation

```
def BackwardNN(A,b,X):
    gx = lossG(X[R],Y) # initialize the gradient
    for r in arange(R-1,-1,-1):
        M = rhoG( A[r].dot(X[r]) + tile(b[r],[1,n]) ) * gx
        gx = A[r].transpose().dot(M)
        gA[r] = M.dot(X[r].transpose())
        gb[r] = MakeCol(M.sum(axis=1))
    return [gA,gb]
```

$$x_{r+1} = g_r(x_r, \theta_r) \qquad \mathcal{E}(x) = L(x_{R+1}, y)$$



Proposition: $\forall r = R, \dots, 0,$

$$\nabla_{x_r}\mathcal{E} = [\partial_{x_r} g_R(x_r, \theta_r)]^\top (\nabla_{x_{r+1}}\mathcal{E})$$

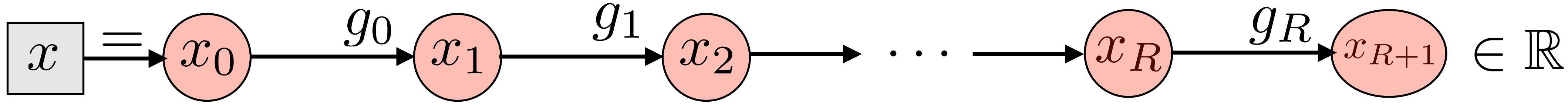
$$\nabla_{\theta_r}\mathcal{E} = [\partial_{\theta_r} g_R(x_r, \theta_r)]^\top (\nabla_{x_{r+1}}\mathcal{E})$$

Example: deep neural network $x_{r+1} = \rho(A_r x_r + b_r)$

$$\forall r = R, \dots, 0, \qquad \begin{aligned} \nabla_{x_r}\mathcal{E} &= A_r^\top M_r \\ \nabla_{A_r}\mathcal{E} &= M_r x_r^\top \\ \nabla_{b_r}\mathcal{E} &= M_r \mathbb{1} \end{aligned} \qquad M_r \overset{\text{def.}}{=} \rho'(A_r x_r + b_r) \odot \nabla_{x_{r+1}}\mathcal{E}$$

```
def ForwardNN(A,b,Z):
    X = []
    X.append(Z)
    for r in arange(0,R):
        X.append( rhoF( A[r].dot(X[r]) + tile(b[r],[1,Z.shape[1]]) ) )
    return X
```

```
def BackwardNN(A,b,X):
    gx = lossG(X[R],Y) # initialize the gradient
    for r in arange(R-1,-1,-1):
        M = rhoG( A[r].dot(X[r]) + tile(b[r],[1,n]) ) * gx
        gx = A[r].transpose().dot(M)
        gA[r] = M.dot(X[r].transpose())
        gb[r] = MakeCol(M.sum(axis=1))
    return [gA,gb]
```

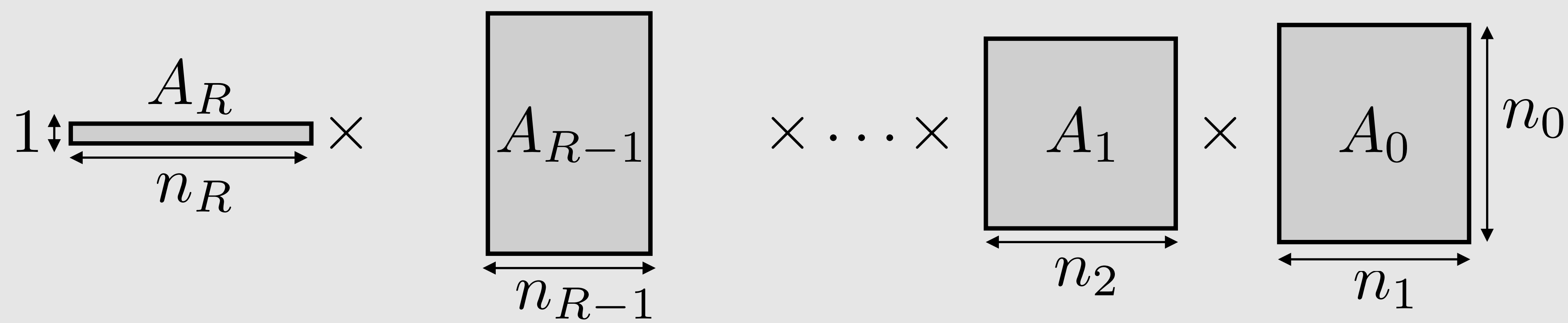


$$\partial g_r(x_r) \in \mathbb{R}^{n_{r+1} \times n_r}$$

$$x_{r+1} = g_r(x_r) \qquad g_r : \mathbb{R}^{n_r} \to \mathbb{R}^{n_{r+1}} \qquad \nabla g_R(x_R) = [\partial g_R(x_R)]^\top \in \mathbb{R}^{n_R \times 1}$$

Chain rule:

$$\partial g(x) = \partial g_R(x_R) \times \partial g_{R-1}(x_{R-1}) \times \ldots \times \partial g_1(x_1) \times \partial g_0(x_0)$$



Forward

$O(n^3)$

$$\partial g(x) = ((\ldots((\underbrace{A_0 \times A_1}_{n_0 n_1 n_2}) \times A_2) \ldots \times \underbrace{A_{R-2}) \times A_{R-1}}_{n_{R-2} n_{R-1} n_R}) \times A_R$$

$$\underbrace{}_{n_1 n_2 n_3} \qquad \underbrace{\phantom{A_{R-1}) \times A_R}}_{n_{R-1} n_R}$$

Complexity: (if $n_r = 1$ for $r = 0, \ldots, R-1$) $(R-1)n^3 + n^2$

Backward

$O(n^2)$

$$\partial g(x) = A_0 \times (\underbrace{A_1 \times (A_2}_{n_1 n_2} \times \ldots \times (A_{R-2} \times (\underbrace{A_{R-1} \times A_R}_{n_{R-1} n_R})) \ldots))$$

$$\underbrace{}_{n_0 n_1} \qquad \underbrace{\phantom{(A_{R-2} \times (A_{R-1}}}_{n_{R-2} n_{R-1}}$$

Complexity: Rn^2

Log-sum-exp: $\mathrm{LSE}_\varepsilon(x) \overset{\text{def.}}{=} \varepsilon \log \sum_i e^{x_i/\varepsilon}$

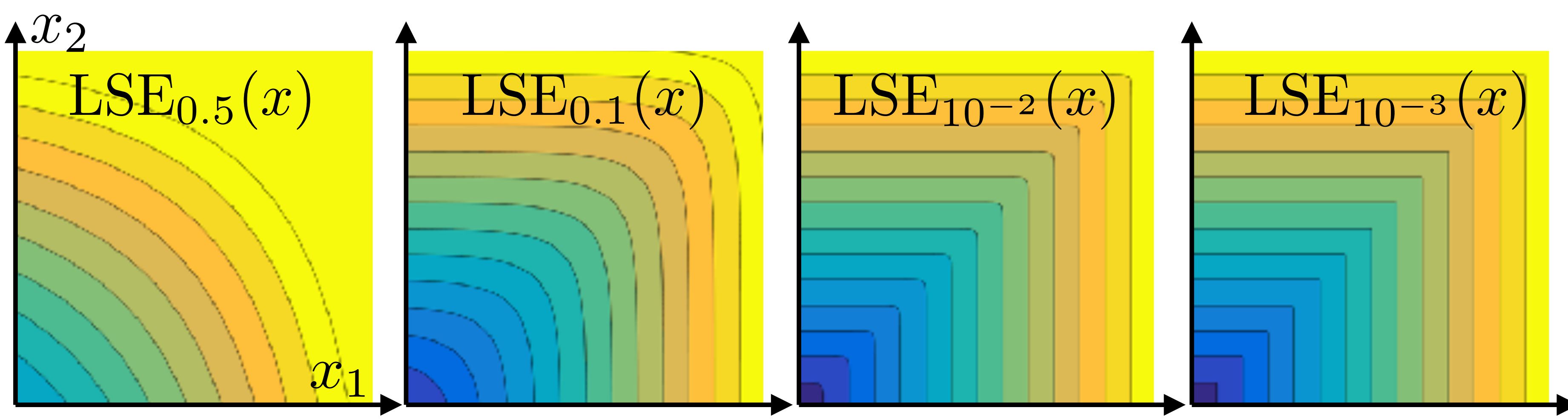
Soft-max: $\mathrm{SM}_\varepsilon(x) \overset{\text{def.}}{=} \nabla_x \mathrm{LSE}(x) = \dfrac{1}{\sum_i e^{x_i/\varepsilon}} \left(e^{x_i/\varepsilon}\right)_i$

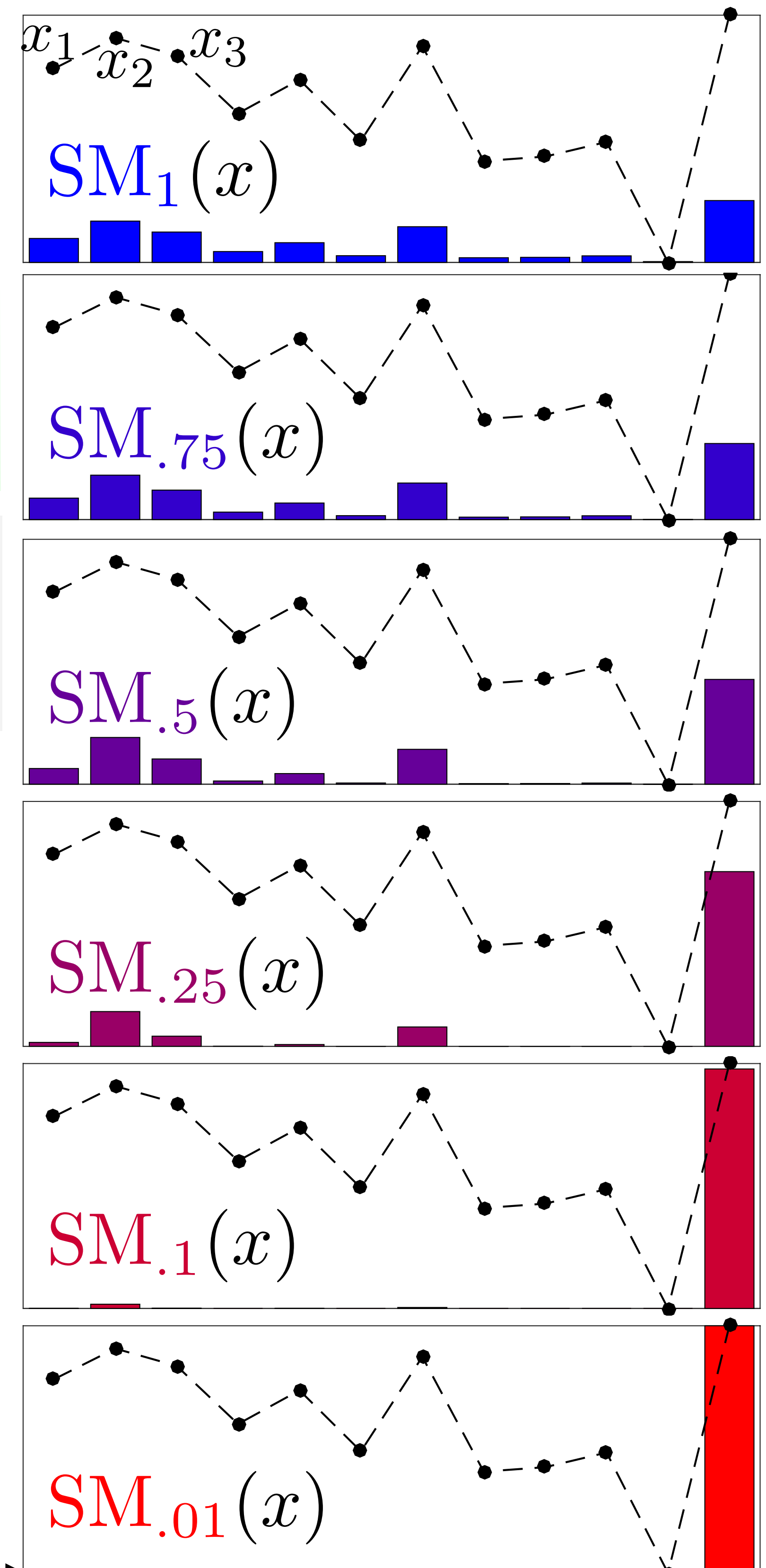
Prop. $\mathrm{LSE}_\varepsilon(x) \xrightarrow{\varepsilon\to 0} \max(x)$ $\quad \mathrm{SM}_\varepsilon(x) \xrightarrow{\varepsilon\to 0} \delta_{\mathrm{argmax}(x)}$

LSE trick:

$$\mathrm{LSE}_\varepsilon(x) = \mathrm{LSE}_\varepsilon(x - \max(x)) + \max(x)$$

Unstable Stable





Soft-max: $\mathrm{SM}(u)_k \overset{\text{def.}}{=} \frac{e^{u_k}}{\sum_\ell e^{u_\ell}}$

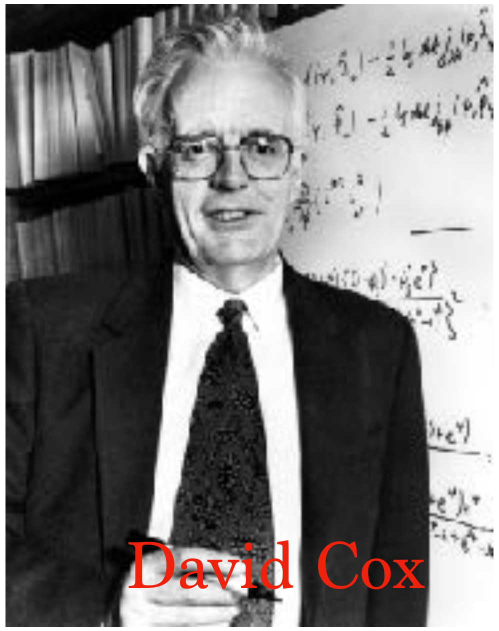
Log-sum-exp: $\mathrm{LSE}(u) \overset{\text{def.}}{=} \log \sum_k e^{u_k}$

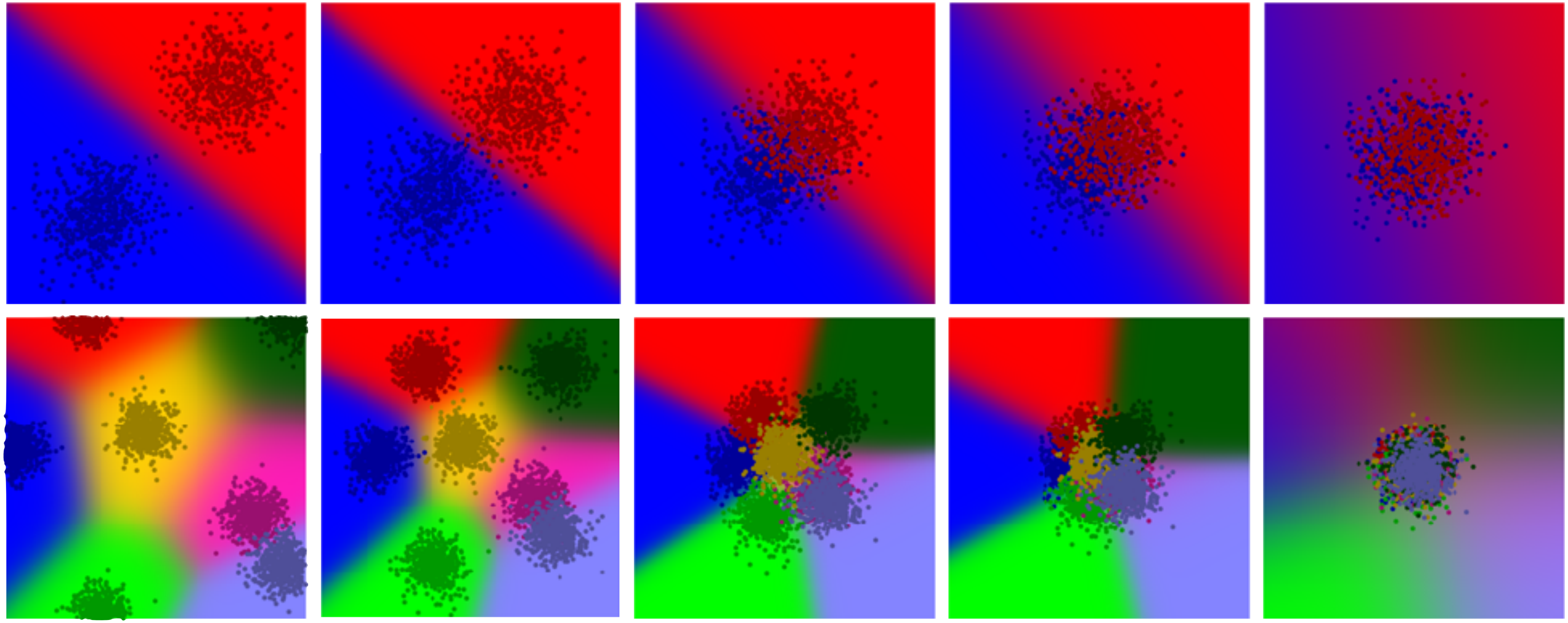
Logit model: $\mathbb{P}(\mathrm{Class}(x) = k) \overset{\text{def.}}{=} \mathrm{SM}((\langle x,\, w_\ell \rangle)_\ell)_k$

Training data: $(x_i, y_i)_i \qquad y_{i,k} = \mathbb{P}(\mathrm{Class}(x_i) = k)$

Logistic classification:

$$\min_{(w_k)_k} \sum_i \mathrm{LSE}((\langle x_i,\, w_k \rangle)_k) - \sum_{i,k} y_{i,k} \langle x_i,\, w_k \rangle$$





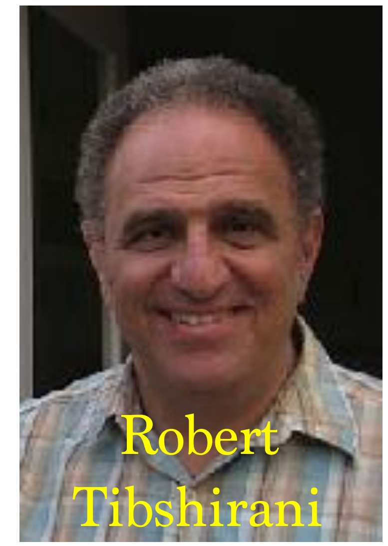
ℓ^p "norms" $\quad \|x\|_p^p \stackrel{\text{def.}}{=} \sum_i |x_i|^p$

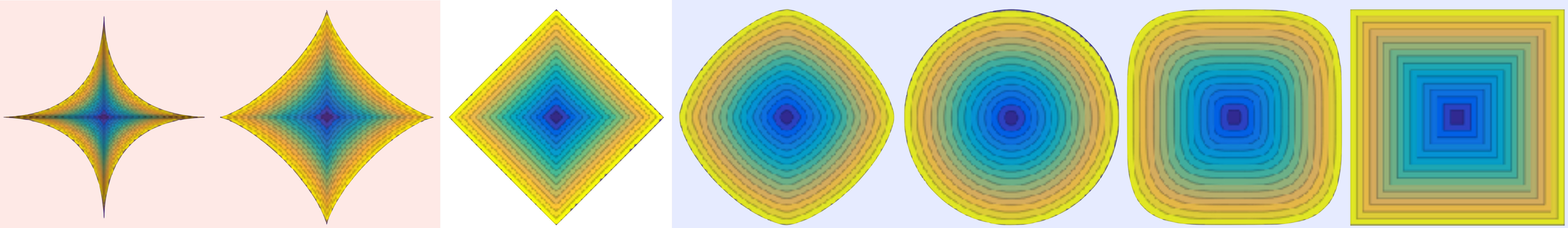
Lasso / Basis-Pursuit:

$$\min_x \|x\|_1 + \frac{1}{2\lambda}\|Ax - y\|^2$$

$$\min_{Ax=y} \|x\|_1$$

$\lambda \to 0$





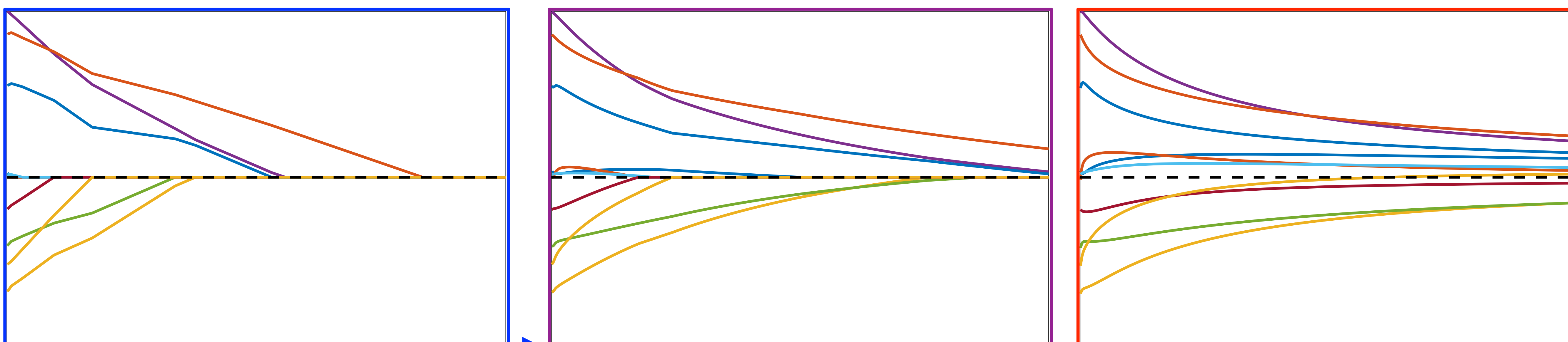
$p = 0.5$ $\quad p = 0.75$ $\quad p = 1$ $\quad p = 1.5$ $\quad p = 2$ $\quad p = 4$ $\quad p = \infty$

Non-convex

Non-sparse

$$\text{Elastic net:} \quad x_\lambda \in \underset{x}{\operatorname{argmin}} \frac{1}{2\lambda}\|Ax - y\|^2 + (1-\theta)\|x\|_1 + \frac{\theta}{2}\|x\|_2^2$$

$$\text{Regularization path:} \quad \lambda \longmapsto x_\lambda$$



Lasso $\theta = 0$ λ Ellastic net $\theta = 1/2$ Ridge $\theta = 1$

Supervised learning: Observations: $(a_i, y_i)_i$, parametric model: $g(x, a)$

Regression:	$y_i \approx g(x, a_i)$	$\ell(y, y') = \|y - y'\|^2$
Classification:	$y_i \approx \theta(g(x, a_i))$ $\theta(u) = (1 + e^u)^{-1}$	$\ell(y, y') = \log(1 + e^{-yy'})$

Empirical risk minimization: $\min_x f(x) = \frac{1}{n} \sum_i \ell(g(x, a_i), y_i)$

$$\min_x f(x)$$

$$f(x) \overset{\text{def.}}{=} \frac{1}{n} \sum_{i=1}^n f_i(x) \quad \underset{n \to +\infty}{\overset{\text{sampling}}{\leftrightarrows}} \quad f(x) \overset{\text{def.}}{=} \mathbb{E}_{\mathbf{z}}(f(x, \mathbf{z}))$$

finite sum / empirical — integral / expectation