Modeling tasks & losses

1) Choosing physically-meaningful metrics

ex: regression $\mathcal{D} = \{(x_i, y_i)\}$

$$\sum_{i \in \mathcal{D}} \| \hat{y}_i - y_i \|^2$$

($\hat{y}_i$: prediction; y_i: desired output)

- classification: $\hat{y}_i = \begin{pmatrix} \hat{p}_1 \\ \hat{p}_i \\ \vdots \\ \hat{p}_C \end{pmatrix}$ proba distribut° over classes

$\| \hat{y}_i - y_i \|^2$: right optimum but bad landscape

Better compare $-\log \hat{p}_i \Leftrightarrow -\log p_i$

⇒ Kullback-Leibler divergence; comes information theory

$$KL(p \| q) = \int p \log \frac{p}{q} = \sum_c p_c \log \frac{p_c}{q_c}$$

↳ not symmetric! $KL(p\|q) \neq KL(q\|p)$

$$KL(p\|q) = \mathop{\mathbb{E}}_{p}\left[\log \frac{p}{q}\right] = \mathop{\mathbb{E}}_{p}\left[(-\log q - (-\log p))^2\right]$$

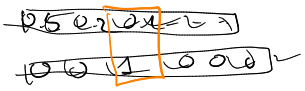
(p: $\hat{p}$)

For a given sample x

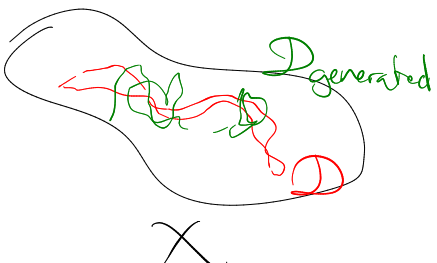
cross-entropy: optimize w.r.t $q = \hat{p}$: $\mathop{\mathbb{E}}_{p}[-\log q]$

$= -\log q_{(\text{right class})}$

in practice: $p = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix}$

$q = \hat{p}$ 

Compare distributions: eg - generative models

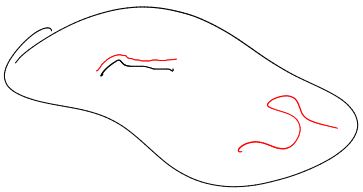


$KL(\mathcal{D} \| \mathcal{D}_\theta)$

$KL(\mathcal{D}_\theta \| \mathcal{D})$

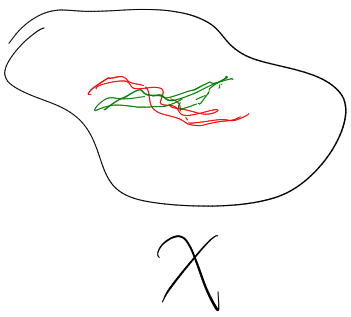
⚠ not symmetric

- if $\exists x$ $p(x) = 0$ or $\hat{p}(x) \neq 0$ then $KL = +\infty$

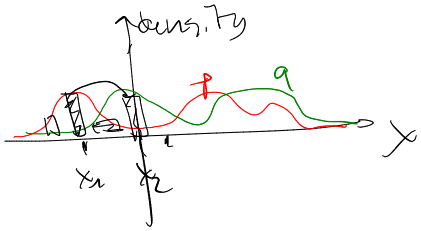


⚠ if discretized space into bins, then issue with empty bins

Optimal transport aka. Earth Mover Distance



$\mathcal{X}$



Transport: $m(x_1) \rightarrow x_2$

$m \times \|x_1 - x_2\|$ ← choose the metric

total cost: $\min_m \iint_{x_1 x_2} m_{x_1,x_2} \|x_1 - x_2\| \, dx_1 dx_2$

$m \geq 0$

$\int_{x_1} m(x_1, x_2) \, dx_1 = q(x_2)$

+ constraints w.r.t p, q : $\int_{x_2} m(x_1, x_2) \, dx_2 = p(x_1)$

- in 1D: close-form solution computable in linear time
- in higher dims: very difficult ⇒ approximation: "Sinkhorn" algorithm [Marco Cuturi, 2013]

↳ regularize:

$\min_m$ total cost $+ \varepsilon H(m)$

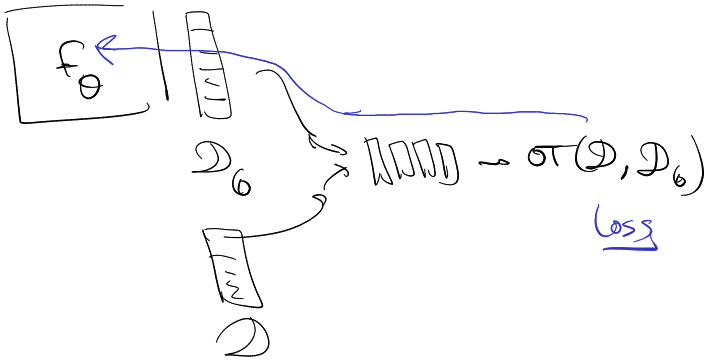
approximate solution ↳ $m = \begin{pmatrix} u \\ \ddots \\ & u \end{pmatrix} \underbrace{e^{-C/\varepsilon}}_{K \text{ fixed}} \begin{pmatrix} v \\ & \ddots \\ & & v \end{pmatrix}$

iterative algorithm:

init: $u, v = (1, \dots, 1)$

$C \begin{cases} u \leftarrow Ku \\ v \leftarrow Ku \end{cases}$

→ in Pytorch: for a # of iterations



⚠ this relies on distances between samples

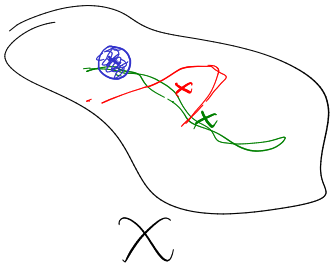
⇒ in high dim, distances don't make sense anymore

Minimum Mean Discrepancy (MMD)

→ to compare statistics of p & q

→ means: $\mu_p = \mathop{\mathbb{E}}_{x \sim p}[x]$

$\mu_q = \mathop{\mathbb{E}}_{x \sim q}[x]$



↳ $\|\mu_p - \mu_q\|$: minimize

→ design lots of statistics: check that p & q lead to the same values for each statistics

eg: density around point x_i

⇒ sufficient statistics if sample x_i everywhere

φ = list of possibly transforms

eg-

⇒ kernel trick: don't compute φ but use a kernel [Gretton]

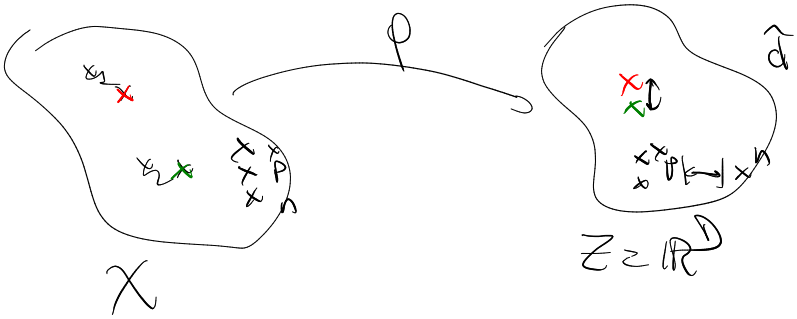
$k(x_1, x_2) = \varphi(x_1) \cdot \varphi(x_2)$

similarity

$k(x_1, x_2) = e^{-\frac{\|x_1 - x_2\|^2}{2\sigma^2}} \times \longrightarrow \varphi$

$\mathop{\mathbb{E}}_{x}[\varphi]$ → expressible with k

2) Metric Learning [Siamese Networks, GE]



Euclidean distance on Z:

$\hat{d}(x_1, x_2) = \|\varphi(x_1) - \varphi(x_2)\|$

Dataset: triplets (x, p, n)

↑ "positive" sample: "close" to x

↑ "negative" sample: farther from x

↳ desire: $d(x, p) \leq d(x, n) - m$

↑ predefined margin $\in \mathbb{R}$

Criterion: triplet loss $\sum_{(x,p,n)\sim\mathcal{D}} \max(d(x,p) - d(x,n) + m, 0)$ — better ≤ 0

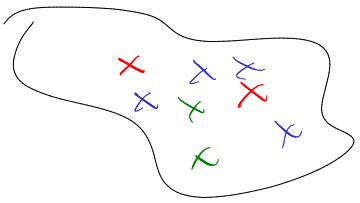
Generalization & Robustness

I Generalization

1) Ensemble methods

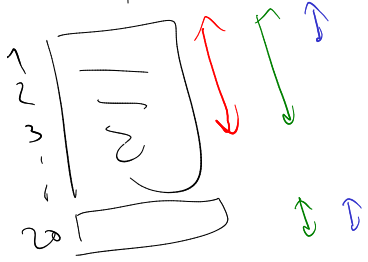
- train N predictors
  - consider the average: $f(x) = \frac{1}{N}\sum_n f_n(x)$ or other aggregation method

⇒ known to be robust



works if mistakes happen at ϵ locations

Leaderboard



- Issue: complexity when applying to a new point (test time)
  ex: $N = 100$

- "teacher-student", "distillation" [Distilling the knowledge in a NN]
  - hard task
    - train a "small" network: doesn't work (size (not huge))
    - train a huge ... : works → optimization reasons
  - train a small network to imitate the huge one: works

A ↑ more feedback: ImageNet 1000 feedbacks/sample

A' $\|A' - A\|$

B) Generalizº without regularizº

"double gradient descent" / NTK: "neural tangent kernel"

$O(1/\sqrt{\#\text{neurons}})$ ⇒ robustness / stability of the estimated Fº obtained after training for a random initº w.r.t. the initº.

C) Know when you don't know

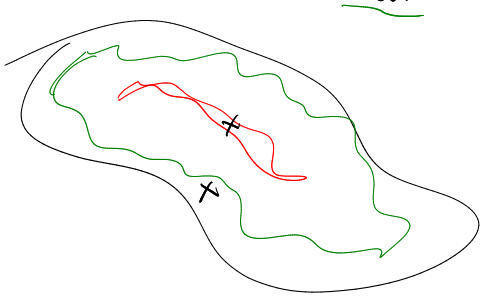
[Matthias Hein]

2 distributions → $\mathcal{D}$: real dataset CIFAR

→ $\mathcal{D}_{out}$: to model samples outside of $\mathcal{D}$ ImageNet

train: classifº

- show samples in $\mathcal{D}$: $\begin{pmatrix}0\\1\\\vdots\\8\end{pmatrix}$ true class
- show samples in $\mathcal{D}_{out}$: $\begin{pmatrix}1/c\\1/c\\1/c\\\vdots\\1/c\end{pmatrix}$ uniform distrib° over classes



II Learning from noisy data

Denoising auto-encoder [ICML 2008]

- originally : x (noisy) → [ ] → $\tilde{x}$ denoised : auto-encoder
- learn to be robust to a certain noise:

$x \xrightarrow[\text{noise}]{\text{add}} x'' \to [\text{DAE}] \to x' \qquad \|x' - x\|$

true input

⇒ learn features robust to the noise

Classification with noisy labels [D.L. is robust to massive label noise, 2017]

- true distribº (x, y)
- noisy : $(x, \tilde{y})$ → sometimes true label / " a random one

- if some samples have wrong labels → still ok → not much change of accuracy

90%. - - - → 99%.

↳ still possible to get reasonable results
provided data is available in large quantity
↳ what matters: ~~# of~~ samples with true labels

noise: 80% → ×10
99% → ×100

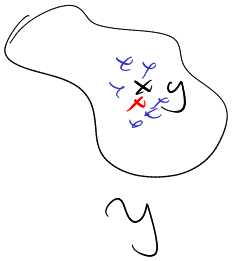
Regression with noisy labels

- dataset = $(x_i, y+\varepsilon_i)$ ← noise: i.i.d. centered (variance σ)

- simplistic case:
always same x, varied $y+\varepsilon_i$

$f(x) \rightarrow$ constant $\hat{y}$

[Noise2Noise]



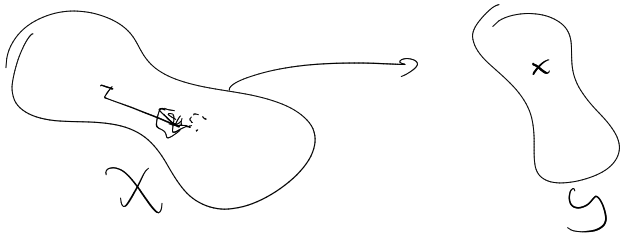
$$\min_{\hat{y}} \sum_i \|\hat{y} - (y+\varepsilon_i)\|^2$$

↳ solution: $\hat{y} = \frac{1}{N}\sum_i (y+\varepsilon_i)$

$= y + \underbrace{\frac{1}{N}\sum_i \varepsilon_i}$

$\sim O(1/\sqrt{N})$

- real case: x varies
→ possible to define similarity between samples x



→ compute the factor in noise reduction

$\varepsilon \sim \mathcal{N}(0,\sigma)$ → $\hat{y} : \pm \frac{\sigma}{\sqrt{N}}$ ← depends on x

1/100 factor σ

⇒ "influence functions"

II. Formal proofs

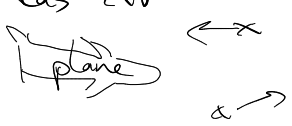
- to prove (formally) that NN (trained) will never mistake
- express property to check:

$$P: \forall \text{input samples } x \in X, \quad \|\hat{y}(x) - y(x)\| < \varepsilon$$

$\mathbb{R}^d$
$[0,1]^d$

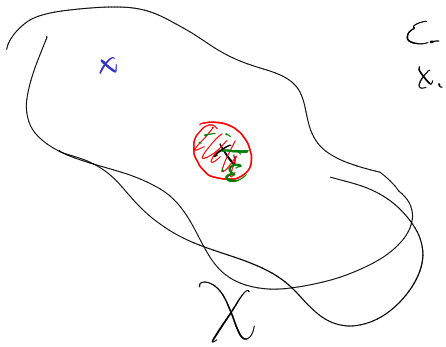
$y(x)$: requires specification

- Acas-Xu



complexity: $\sim O(2^{\#\text{neurons}})$

- local properties ≈ robustness to adversarial attacks



ϵ

x

$\forall x' \in B(x, \varepsilon), \quad |f(x) - f(x')| < \eta$

0.1