Modeling tasks & losses

1) Choosing physically meaningfull metrics

Task : - dataset
- Functional space (architecture)
- optimization criterion ← modeling the errors

regression → $\| \hat{y} - y \|^2_{L^2}$

↑ prediction $\in \mathbb{R}^d$ ↑ ground truth $\in \mathbb{R}^d$

classification ≃ regression on probabilities

$\hat{y} = \begin{pmatrix} \hat{p}_1 \\ \hat{p}_2 \\ \vdots \end{pmatrix}$ $\quad y = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \end{pmatrix}$ $\quad \sum_i \hat{p}_i = 1$

→ L^2 norm ? meaningful

ex : $p_1 = 0,73 \pm 0,01$ 73% 72-74%

$p_1' = 0,001 \pm 0,01$ 0,1% 0% - 1,1% 1 chance over 1000 not possible — ~1 chance over 100

$|\hat{p}_1 - p_1| = 0,01$

↳ information theory ⇒ $-\log p \geq 0$ ← output of the network ↓ softmax

$|-\log \hat{p}_1 - (-\log p_1)| = |\log \frac{p_1}{\hat{p}_1}|$

↙ criteria to compare probabilities

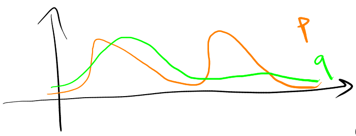
Kullback-Leibler : $KL(p\|q) = \int p \log \frac{p}{q} \geq 0$

$= \mathbb{E}_{x\sim p}\left[\log \frac{p(x)}{q(x)}\right]$

$= 0$ only if $p = q$

not symmetric : $KL(p\|q) \neq KL(q\|p)$

$KL(q\|p) = \int q \log \frac{q}{p} = \mathbb{E}_{x\sim q}\left[-\log \frac{p}{q}\right]$



↳ mode-drop : $KL(q\|p)$

$KL(p\|q) = \int p \log \frac{p}{q} = \int p \log p - \int p \log q$

$\int p \log p$: $-H(p)$ (entropy of p) ↓ constant (does not depend on θ)

$-\int p \log q$: cross-entropy — easy to implement! → $\sum_{i \in \text{minibatch}}$ - network output before softmax (x_i)

p : target distribution

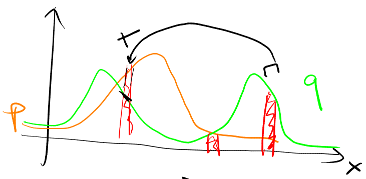
q_θ : output "

$\inf_\theta KL(p\|q_\theta)$

$\theta_{t+1} = \theta_t - \varepsilon \nabla_\theta KL(p\|q_\theta)$

↙ other issue :
if at some point x,
$p(x) = 0$ and $q(x) \neq 0$
⇒ $KL(q\|p)$ is too
↳ careful about bin discretization

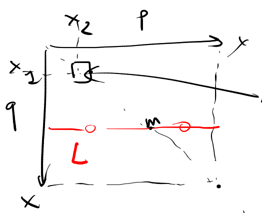
Optimal transport aka Earth Mover Distance



$\int p = 1$

$\int q = 1$

$\inf_M$ Total Labor (M, c) → optimal transport



Transportation Matrix

quantity of earth moved from $q(x_1)$ to $p(x_2)$

M $\begin{cases} \text{proj } M_{/q\text{ axis}} = q \\ \text{proj } M_{/p\text{ axis}} = p \end{cases}$ $\quad M \geq 0$

Cost : $c(x_1, x_2)$

ex : $= \|x_1 - x_2\|$

Total Labor (M, c) $= \int_{x_1, x_2} M_{x_1, x_2} \, c(x_1, x_2) \, dx_1 dx_2$

↑ how much earth I move from location x_1 to location x_2

↑ cost of this move (length of transport)

- in dim 1 : exact solution in linear complexity (match cumulative distrib°)
- in dim > 1 : hard problem
  ⇒ approximate solution : Sinkhorn algorithm [Marco Cuturi, 2013]
  + Gabriel Peyré 2018 ...

bistochastic (set of samples with same mass)

$\text{Total cost} + \varepsilon H(M)$ — ε: small value; H: entropy

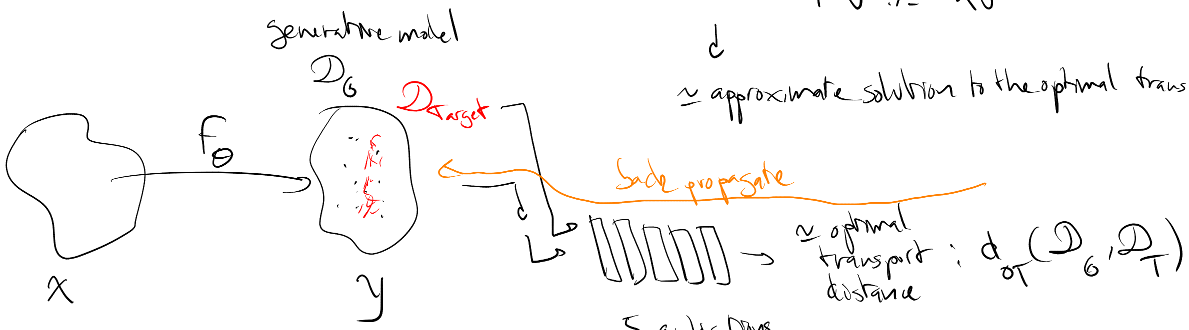
⇒ optimal solution of the form : $M = \begin{pmatrix} u \end{pmatrix} \boxed{e^{-C/\varepsilon}} \begin{pmatrix} v \end{pmatrix}$ ($K = e^{-C/\varepsilon}$, fixed)

initialize u, v with $(1, 1, \dots, 1)$

iterative scheme
$$\begin{cases} u \;./\!= Kv \\ v \;./\!= Ku \end{cases}$$

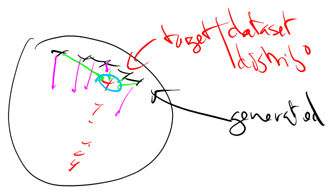
↓

≃ approximate solution to the optimal transport pb



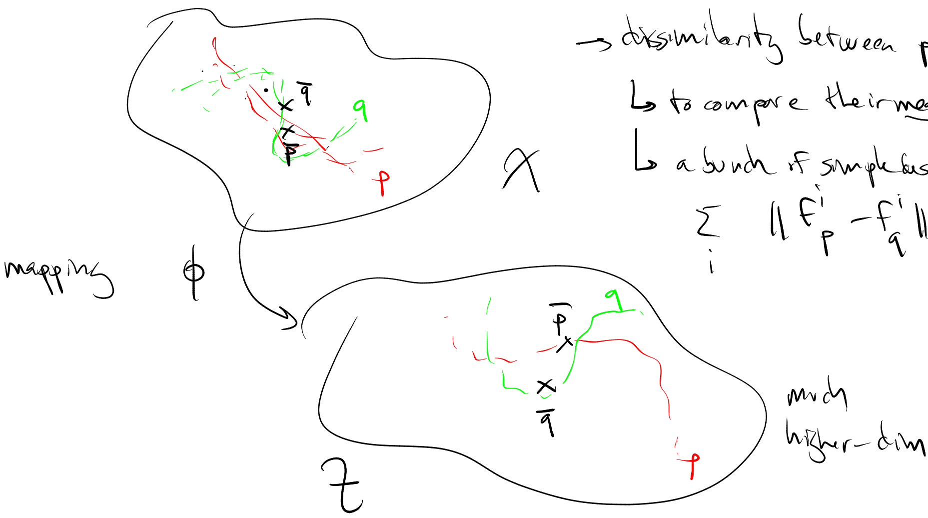
5 applications of the process
⟷
neural network with fixed weights
differentiable (& simple)

use case :
- if classification : cross-entropy is fine
- general distribution to be matched to a target one (continuous space) & metric in that space
  → O.T.



Minimum Mean Discrepancy (MMD)

to compare 2 distributions p, q



→ dissimilarity between p & q ?
  ↳ to compare their means : $\|\bar{p} - \bar{q}\|$
  ↳ a bunch of simple descriptors of distrib° : f^i
  $$\sum_i \|f_p^i - f_q^i\|^2$$

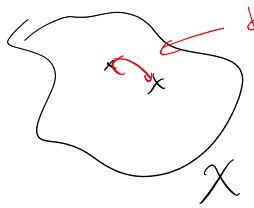
$f^i = \phi^i$

means matching ⇒ close match

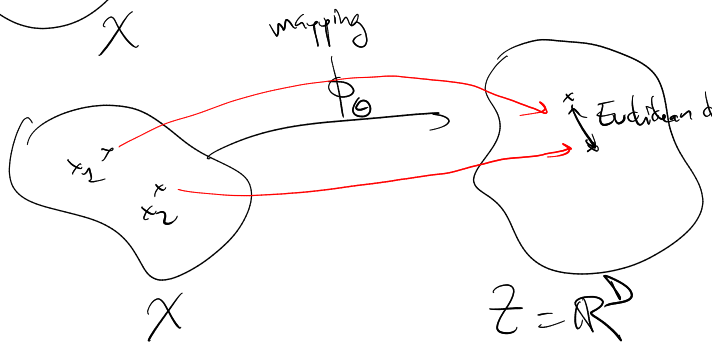
ϕ^i = density around point x_i

do not compute all ϕ^i explicitly
↳ use a kernel $k(p, q)$
[Gretton]

2) Modeling the task: metric learning

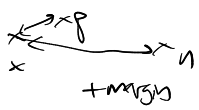


[Siamese networks, NIPS 1994]



Dataset: triplets (x, p, n)
- x: example of interest
- p: "positive": sample "close to" x
- n: "negative": sample "far away" from x

desired property: $d(x,p) \leq d(x,n) - m$ (predefined margin $\in \mathbb{R}^+$)



Criterion: $\sum_{(x,p,n) \in \text{dataset}} \max(d(x,p) - d(x,n) + m, 0)$ "triplet loss"
- → if negative: do nothing
- → if ≥ 0: do something

- Curriculum learning: from easy task to harder task [Bengio et al, 2009]
  - hard ML task
    ⇒ series of tasks, T_1 = easy
    T_2 = harder
    ⋮
    T_N: really hard (real task)

ex: classification objects

complexifying a bit at each step

  - change: the distribution of samples
    - real: $p(x)$
    - at time t: $q_t(x) = w_t(x)\, p(x)$

easy → less easy → ... → difficult

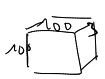
at $t=0$: $w_0(x) = \begin{cases} 0 & \text{if difficult } x \\ 1 & \text{if easy} \end{cases}$

with time: $w_t(x)$ ↗

at final step $t=N$: $w_t(x) = 1$

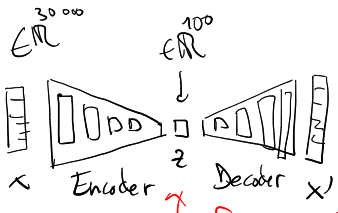
Entropy $H(q_t)$ ↗ increasing with time

Generative Models



→ 30 000 dim

1) Auto-encoders



↳ information bottleneck

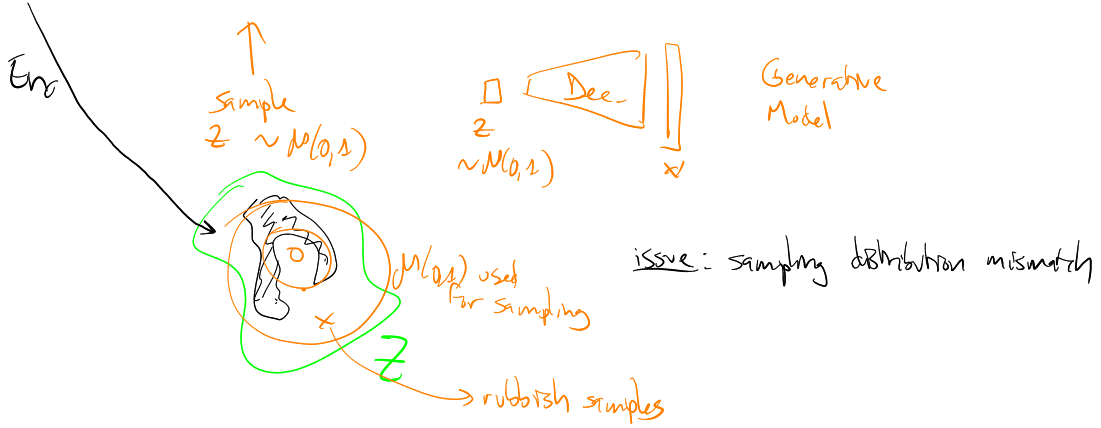
Optimized criterion: $\|x' - x\|$ reconstruction error → issue: which metric?

↳ best: identity

(for VGG activities: perceptual loss)

$x \to$ VGG

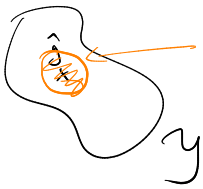
$x' \to$ VGG



issue: sampling distribution mismatch

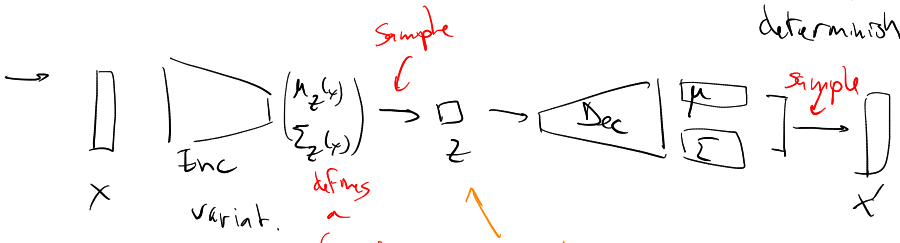
Variational Auto-Encoders (VAE)

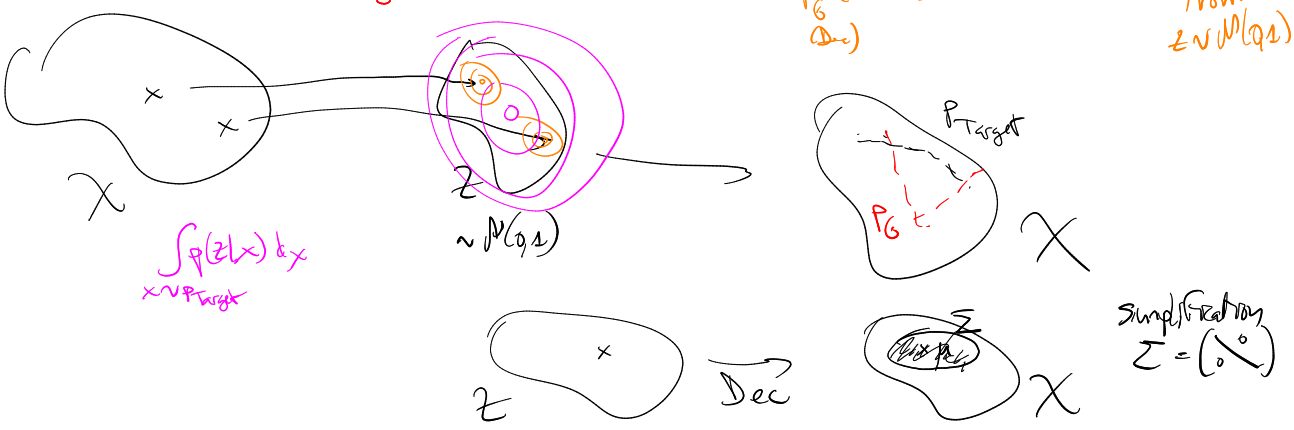
→ variational set-up:
- networks not deterministic, stochastic
→ output: distribution



$$f(x) = \begin{pmatrix} \mu(x) \\ \Sigma(x) \end{pmatrix} \xrightarrow[\text{from } \mathcal{N}(\mu(x), \Sigma(x))]{\text{sample}} \hat{y}$$

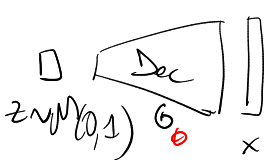
deterministic — stochastic





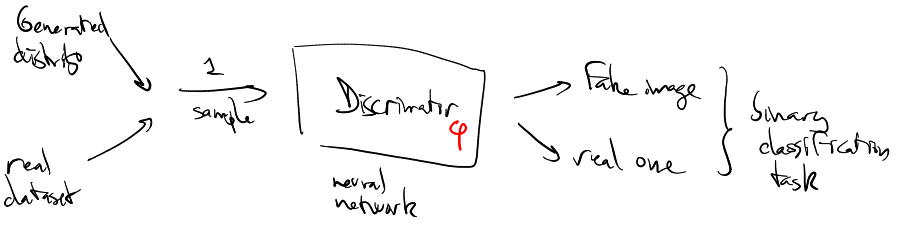
2) Adversarial setting: GAN: Generative Adversial Networks

↳ DANN: Domain-adversarial NN



→ quantify how good outputs are

head? eyes, mouth? colors typical?] modeling by hand



Goal: generate samples (with G_θ) that no one can distinguish from real samples

Loop: G ⇄ D

Same loss for G_θ & D_φ but with opposite sign → no convergence guarantee

Yet if F converges, Nash equilibrium ⇒ $p_G = p_T$

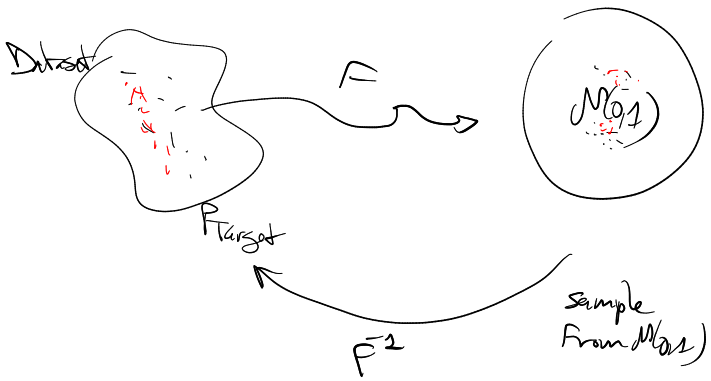
↳ Kolmogorov test to distinguish 2 distributions

Extensions: Wasserstein GAN (optimal transport)

⇒ Same setup, with one constraint on Discr.

$$OTD(p,q) = \sup_{\substack{\text{functions } f \\ \|f\| \leq 1 \\ \text{Lipschitz}}} \mathbb{E}_{x\sim p}[f(x)] - \mathbb{E}_{x\sim q}[f(x)]$$

3) Normalizing Flows



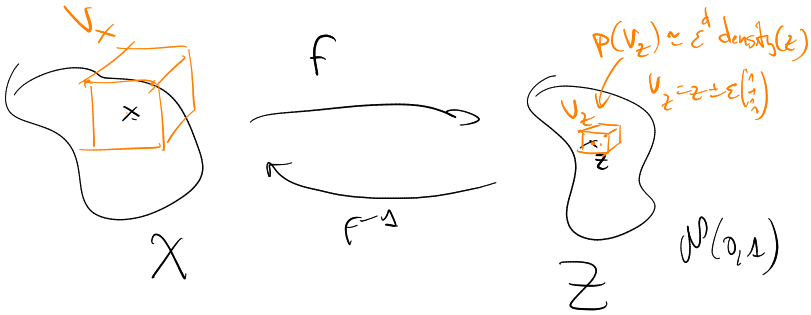
transforming p_{Target} into $\mathcal{N}(0,1)$ normal distrib°

Constraints:

- F needs to be invertible (and computationally easily)
- need to compute fast $\det J$ J: Jacobian of $J_{(F)} = \frac{\partial F}{\partial x}$ ← huge matrix
- same dimensions in input & output space

if: $F: \mathbb{R}^d \to \square \to \square \to \square \to \mathbb{R}^d$ (feedforward stack of layers)

⇒ Same number of neurons in every layer = dim input space



$KL(p_{Target} \| p_{Generated})$

$KL(p\|q) = \int p \log p/q$

$q(z)dz = p(V_z)$

$p_\theta(x) = p_{z_0}(z) \times$ volume ratio $\rightarrow \det \left| \frac{\partial F^{-1}}{\partial z} \right| = \det \left(\frac{\partial F}{\partial x} \right)^{-1} = \frac{1}{\det \frac{\partial F}{\partial x}}$ ($J_{(F)}$)

$\log p_\theta(x) = \log p_{z_0}(z) - \log \det J_{(F)}$

Example of architectures:

- 

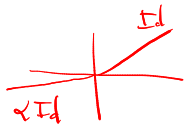
⚠ not expressive at all

$y^i = \lambda^i x^i \quad \lambda^i \neq 0$

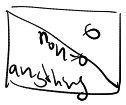
$y = \Lambda x \quad \Lambda^{-1} = \begin{pmatrix} 1/\lambda_i & & 0 \\ & \ddots & \\ 0 & & \end{pmatrix}$

$\frac{dy}{dx} = \Lambda \quad \det J = \det \Lambda = \prod_i \lambda_i$

⚠ activation function: invertible! Leaky ReLU



- Triangular matrices:



"auto-regressive"

$y = \triangle x$

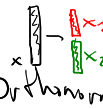
$\det J = \prod \lambda_i$

Inverse by Gauss method

$\begin{cases} y_1 = \lambda_1 x_1 \Rightarrow x_1 = 1/\lambda_1 \, y_1 \\ y_2 = \lambda_2 x_2 + m_{12} x_1 \Rightarrow x_2 = \frac{1}{\lambda_2}(y_2 - m_{12} x_1) \end{cases}$

↳ iterable process → $O(N^2)$

- Real-NVP: variation on this



↳ triangular Jacobian

↳ split x in 2: $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$

$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_1 \\ x_2 \cdot \exp(s(x_1)) + t(x_1) \end{pmatrix}$

coefficient-wise multiplication

easy to inverse + triangular Jacobian

- Orthonormal matrices

$\Rightarrow \det = \pm 1 \Rightarrow$ how to train?

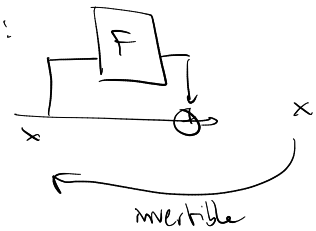
Q becomes $Q = S_1 S_2 S_3 \cdots S_N \quad N \leq d$

$S_i = Id - 2 \frac{\vec{v_i} \cdot \vec{x}}{\|v_i\|^2}$ (Householder decomposition)

- any matrix can be decomposed as Orthonormal $\times \triangle$ $\in \mathbb{R}$

$\Rightarrow$ full expressivity

- ResNet: each block:



$y = x + F(x)$

constraint $\|A\| < \varepsilon$

$Id +$ small is invertible

- KL loss: $KL(p_{Target} \,||\, p_\theta)$ ← generated distrib° (over χ)

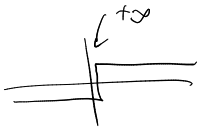
$$= \int_{x \in \chi} p_{Target}(x) \log \frac{p_{Target}(x)}{p_\theta(x)} dx$$

$$= C^{st} - \underbrace{\int_{x \in \chi} p_{Target}(x) \log p_\theta(x) dx}_{\mathbb{E}_{x \sim p_{Target}}[\log p_\theta(x)]}$$

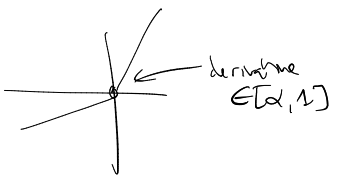
→ Sample mini-batch of real examples → approximate $\mathbb{E}_{x \sim p_{Target}}$ by $\frac{1}{N} \sum_{i \in \text{mini-batch}}$

$$- \frac{1}{N} \sum_i \log p_\theta(x_i)$$

- Heaviside:



ReLU (Leaky)



- Stacking layers: if $F = f_3 \circ f_2 \circ f_1$:

  and associated Jacobians $J_3 = \frac{df_3(z)}{dz}$ $J_2 = \frac{df_2(z)}{dz}$ ...

  $$J_F = J_3 \times J_2 \times J_1$$

  $$\det J_F = \det J_3 \times \det J_2 \times \det J_1$$

  $$\log\det J_F = \sum_{\text{layers } l} \log\det J_l$$

  ⚠ do not forget activation functions

- ⚠ Check the expressivity of your architecture

  ex: stacking only lower-triangular matrices: ◺ × ◺ × ◺ is a lower-triangular matrix ◺ → no full expressivity

  But stacking lower-/upper-triangular matrices alternately:

  ◺ × ◹ × ◺ × ◹ mixes better the variables → ▦ full matrix

  Theorem: full expressivity of PLU decomposition: can represent any matrix

  P (permutation) ◺ ◹

  but: P not optimizable by gradient descent

  ⇒ better use QR decomposition (and stack), and add non-linear layers (leaky ReLU, e.g.)

  * Full expressivity proven for some non-linear blocks (real-NVP, iResNet ...) upon large hidden layers / many blocks enough assumption

  leaky ReLU: diagonal (= coefficient-wise) function

  $$\det \text{Jacobian}_{\text{LeakyReLU}} = \prod_i \begin{cases} \alpha & \text{if } x_i < 0 \\ 1 & \text{if } x_i \geq 0 \end{cases}$$

  $$= \alpha^{\#\text{activities} < 0} \times 1^{\#\text{activities} \geq 0}$$

  $$\log\det J_{\text{LeakyReLU layer}} = (\#\text{activities} < 0) \times \log\alpha$$