Deep Density Destructors (from a biased viewpoint)

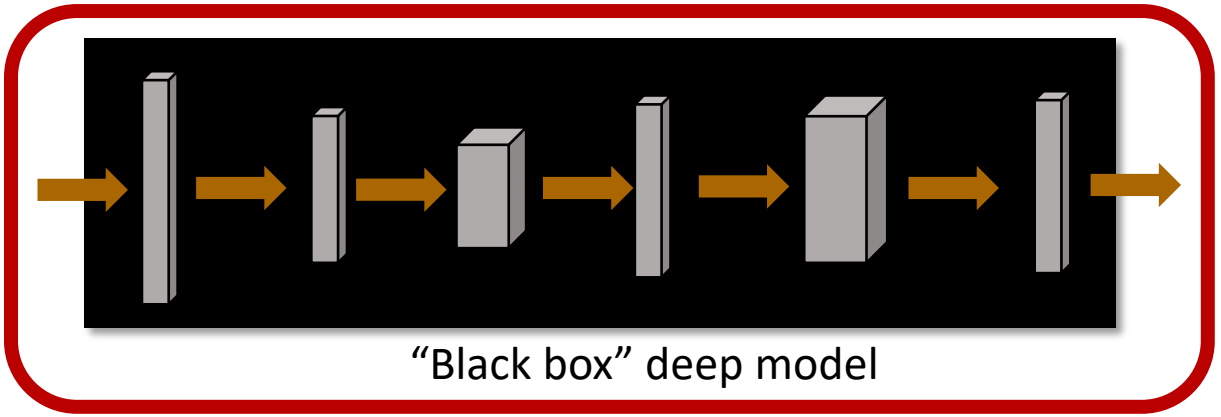
David I. Inouye

Electrical and Computer Engineering

Purdue University

Previous deep normalizing flows are trained end-to-end where all components are optimized simultaneously

End-to-end learning

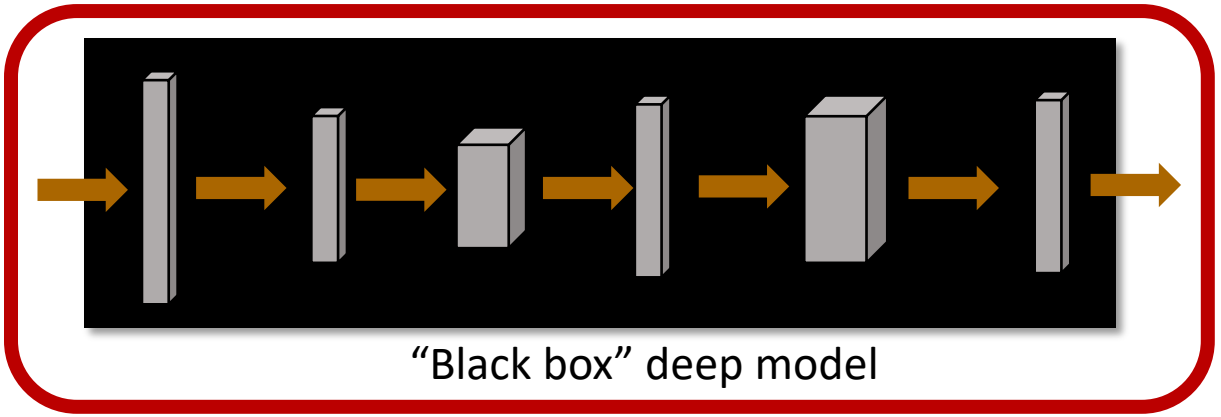


- Real NVP
- MAF
- GLOW
- Etc.

"Gray box" deep model

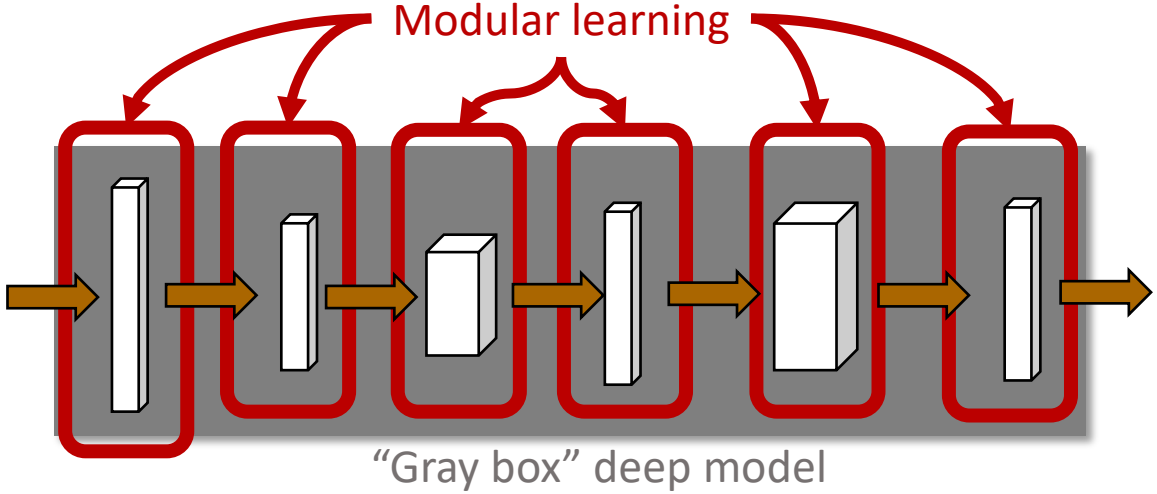
Modular deep learning would allow *local* learning within each component

End-to-end learning



- Real NVP
- MAF
- GLOW
- Etc.

Modular learning



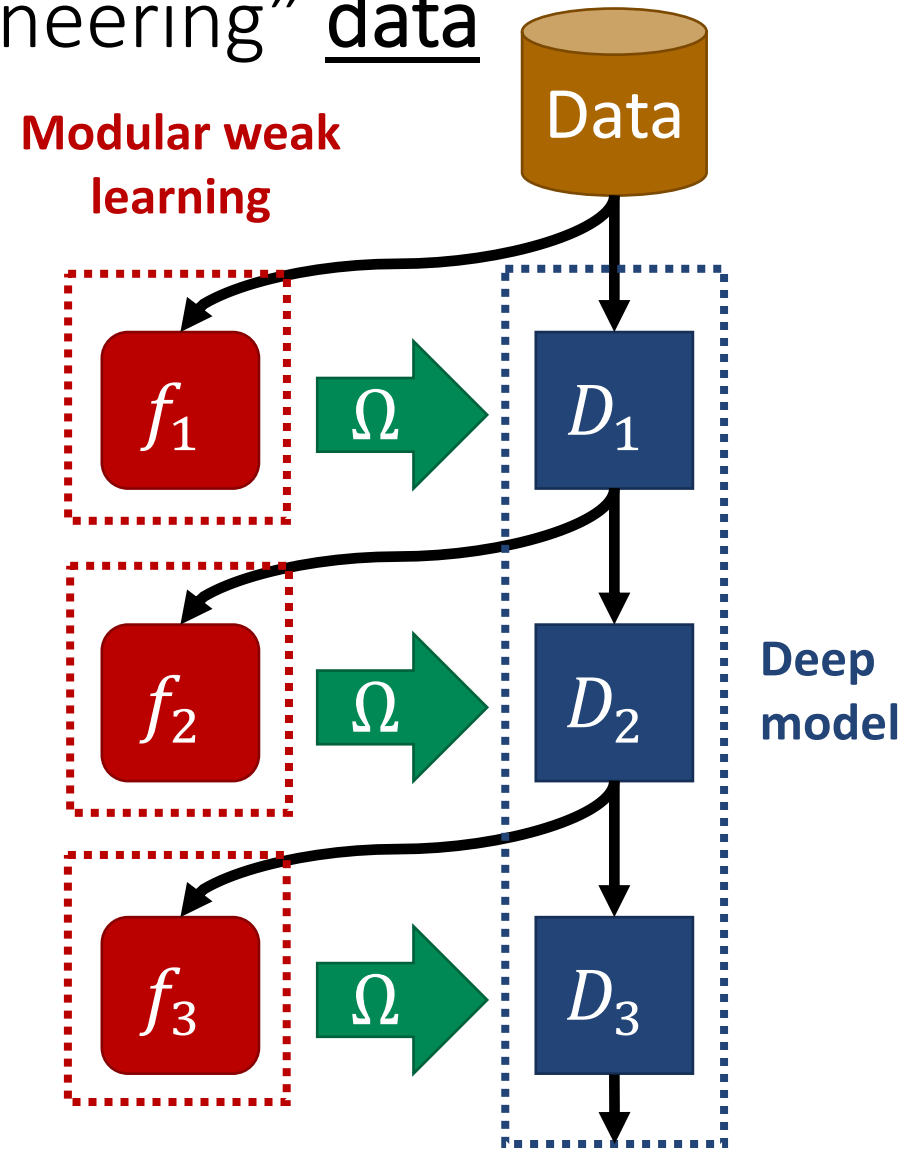
- Density destructors
- Each weak/shallow learning algorithm is independent
- Learning algorithms could be heterogeneous (e.g., SGD and decision trees)

Destructive learning enables modular deep learning via "reverse engineering" **data**

Reverse engineering phone	Reverse engineering data
1. **Find** *part* to take off using **understanding and expertise**	1. **Find** *patterns* in data via **shallow/weak learning**
2. **Determine how** to take off part in a *reversible* way (e.g., unscrewing bolts)	2. **Map model** to destructive but *invertible* transformation
3. **Remove part**	3. **Destroy the patterns** via transformation
4. **Repeat**	4. **Repeat**

Destructive learning enables modular deep learning via "reverse engineering" **data**

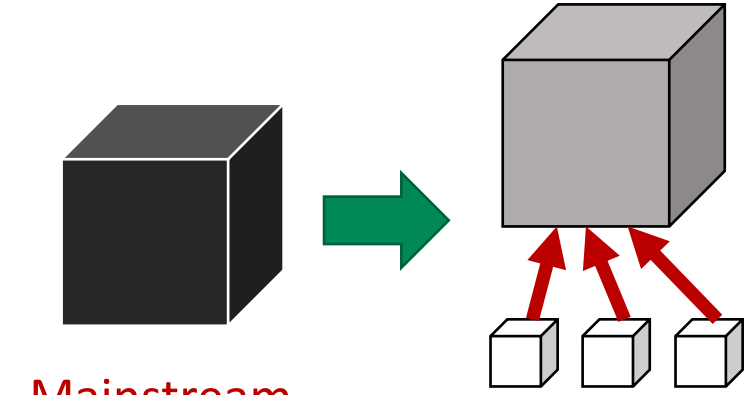
1. **Find** patterns in data via **shallow/weak learning**
2. **Map model** to destructive transformation
3. **Destroy the patterns** via transformation
4. **Repeat**



Why use modular weak learning for deep models?

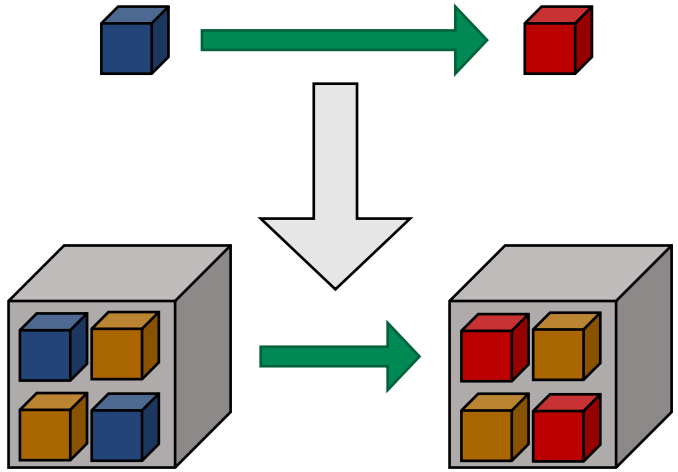
Reuse

The algorithms, insights and intuitions of shallow learning can be lifted into the deep context



Decoupling

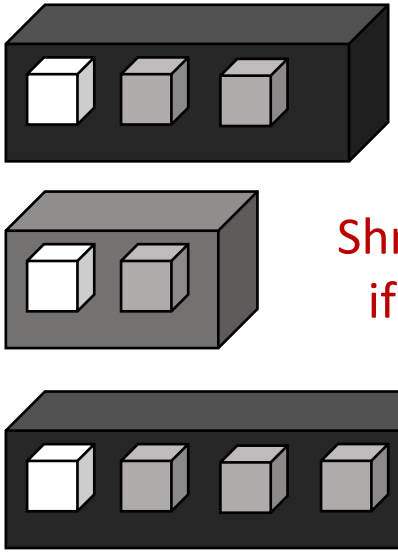
Components can be debugged, tested and improved separate from the system



Why use modular weak learning for deep models?

Algorithmic Interpretability

Increasing or decreasing model complexity is straightforward

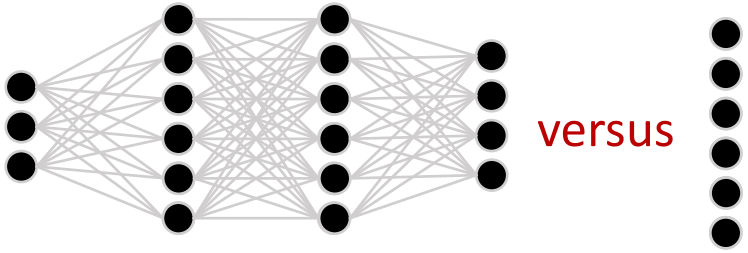


Shrink model if problem

Grow if more data

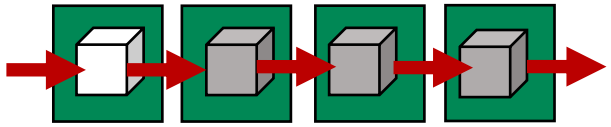
Resource Constraints

Layer-wise training (memory bottleneck)



Pipelined training (computation bottleneck)

Shallow/weak online learners



Distributed on different processors or devices

Overview of iterative destructive learning

Motivation and intuition for modular destructive learning

Density destructors objective function

Modular and greedy deep destructive algorithm

- Simple density destructors
- Deep density destructors

Theory about algorithm: Monotonic decrease of objective

Density destructor results

Limitations and open problems

Iterative alignment and translation

Background for objective: KL equivalence lemma

- KL equivalence lemma: If $z = D(x)$ for invertible D, then
$$KL\big(P_x(x), Q_x(x)\big) = KL\big(P_z(z), Q_z(z)\big)$$
- $KL\big(P_x(x), Q_x(x)\big)$
- $= E_{P_x}\left[\log \frac{P_x(x)}{Q_x(x)}\right]$
- $= E_{P_x}\left[\log \frac{P_z\big(D(x)\big)|J_D(x)|}{Q_z\big(D(x)\big)|J_D(x)|}\right]$ (Change of variables formula)
- $= E_{P_x}\left[\log \frac{P_z\big(D(x)\big)}{Q_z\big(D(x)\big)}\right]$
- $= E_{P_z}\left[\log \frac{P_z\Big(D\Big(D^{-1}(z)\Big)\Big)}{Q_z\Big(D\Big(D^{-1}(z)\Big)\Big)}\right]$ (Expectation change of variables LOTUS)
- $= E_{P_z}\left[\log \frac{P_z(z)}{Q_z(z)}\right]$
- $= KL\big(P_z(z), Q_z(z)\big)$

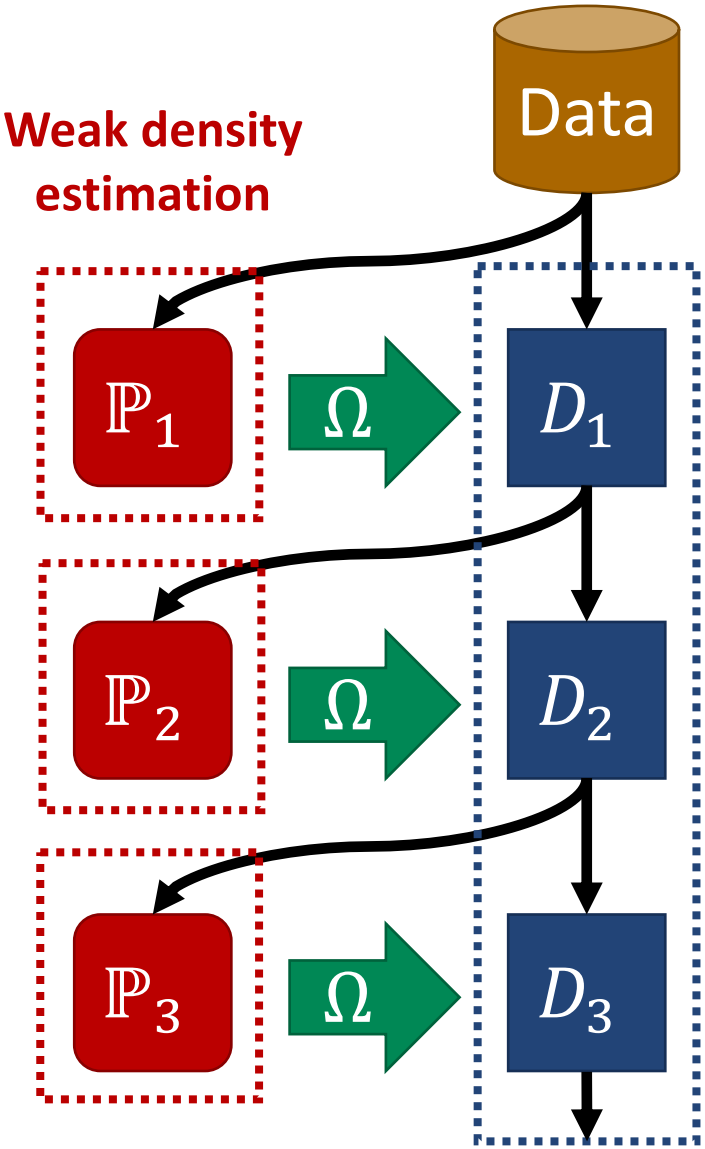
Destructive learning objective is equivalent to MLE

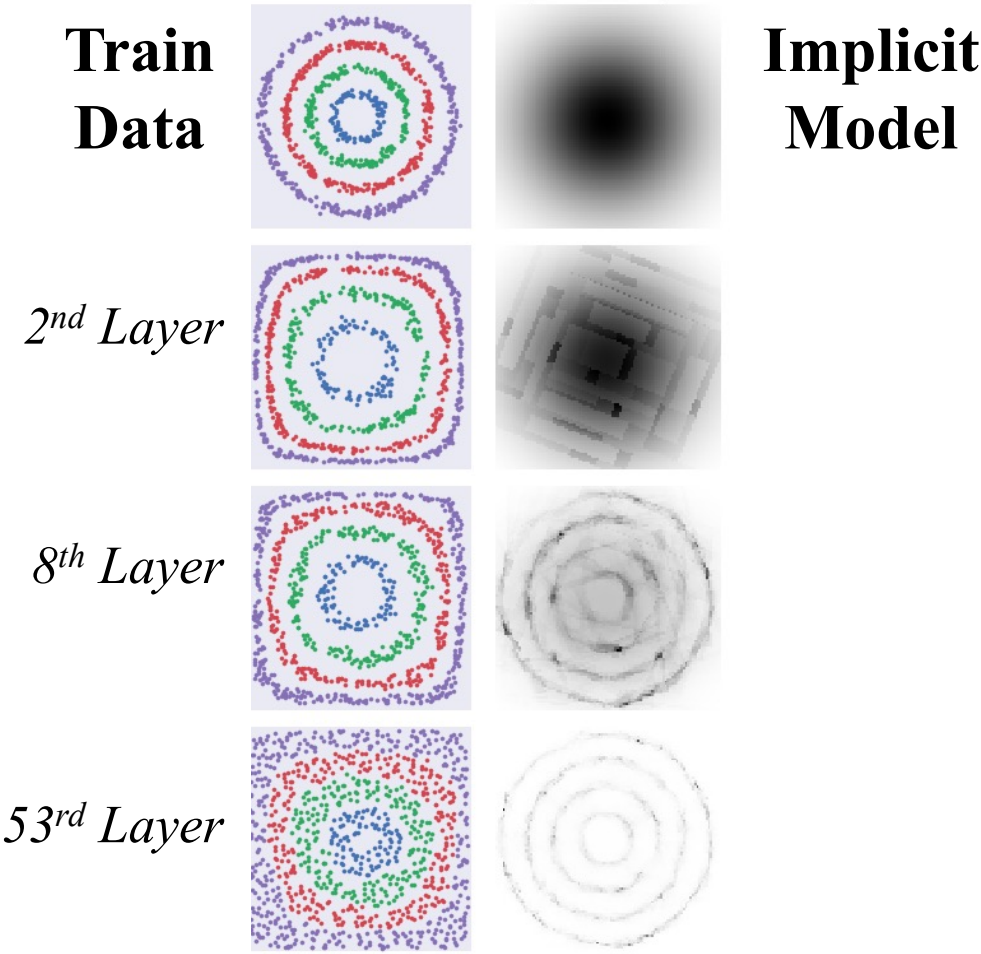
- The destructive learning objective, where $z = D(x)$, and $U_z(z)$ is the uniform density function

$$\arg\min_D KL\left(P_z(z; D), U(z)\right)$$

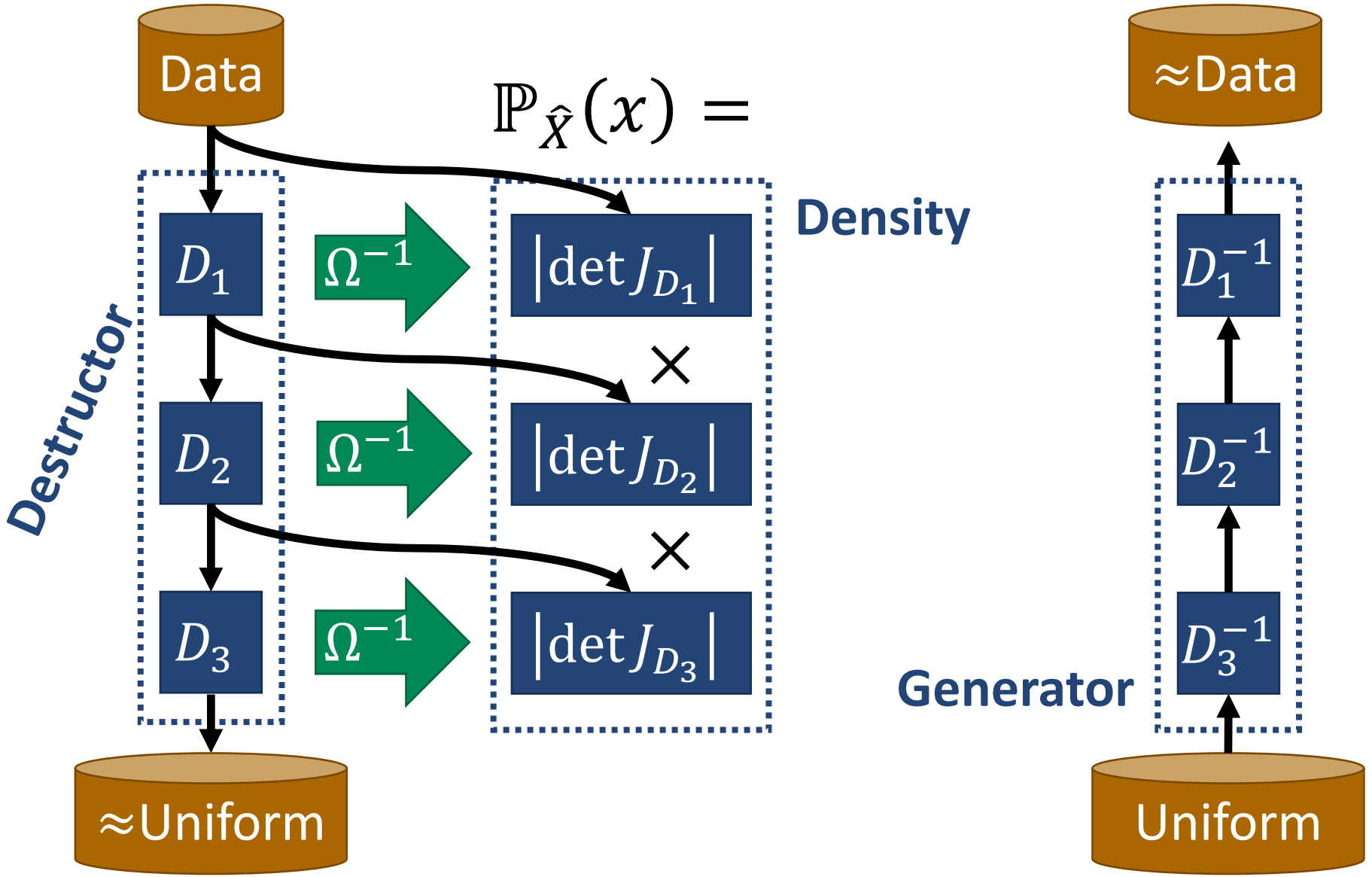
- Simple corollary is that objective above is MLE:
- $KL\left(P_z(z; D), U(z)\right)$
- $= KL\left(P_x(x), Q_x(x; D)\right)$ (KL equivalence, MLE objective)
- $= KL\left(P_x(x), |J_D(x)|U\left(D(x)\right)\right)$ (In terms of D)
- $= KL(P_x(x), |J_D(x)|)$ $(U(z) = 1)$

Algorithm: ***Deep*** density destructors via sequence of *weak* destructors



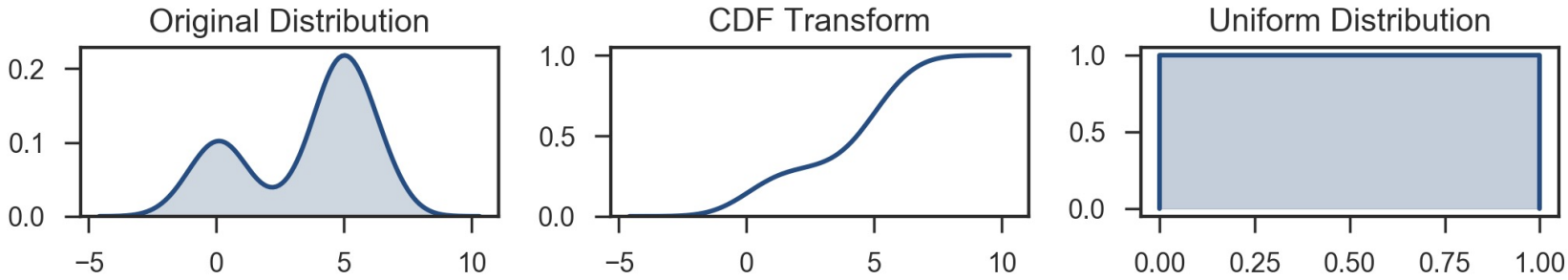


Density computation and sample generation



Definition: *Density destructors* generalize the univariate CDF transformation

- Univariate: CDF transformation



- The map $\Omega(\mathbb{P}) = D$ should:
  1. Encode the density $\mathbb{P}$ into D, i.e. $\exists\ \Omega^{-1}$.
  2. Ensure D destroys all patterns in $\mathbb{P}$ when applied to the random variable, i.e. the distribution of $\mathrm{D_X}(X)$ is maximum entropy.
- A *density destructor* is an invertible transformation such that

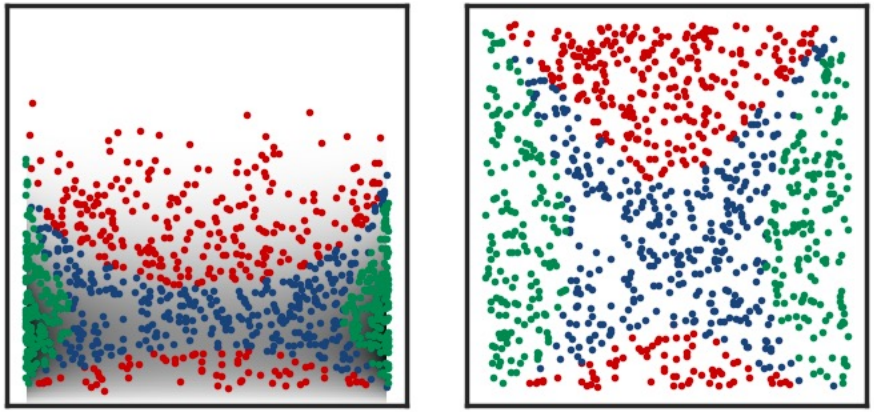
$$X \sim \mathbb{P}_X$$

$$D_X(X) \sim \mathrm{Uniform}\left([0,1]^d\right)$$

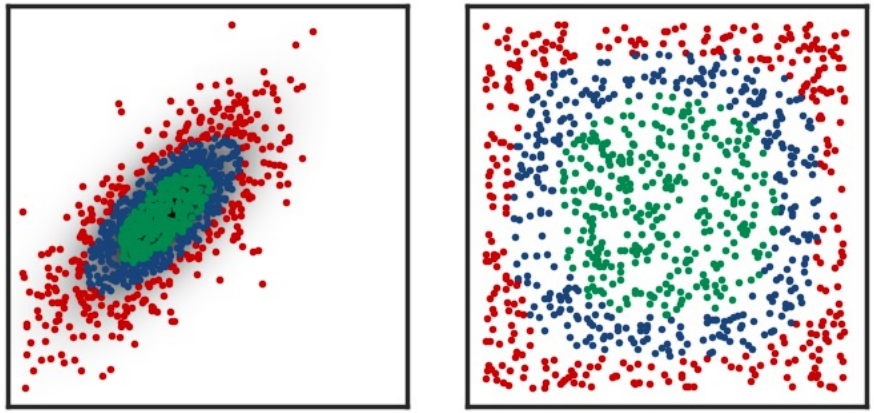
  - $\Omega^{-1}(D_X) = \left|\det J_{D_X}\right| = \mathbb{P}_X$ ← Closed-form density!
  - Different from multivariate CDF function: $F(x)\colon \mathbb{R}^d \to [0,1]$

Many shallow densities can be mapped to destructors

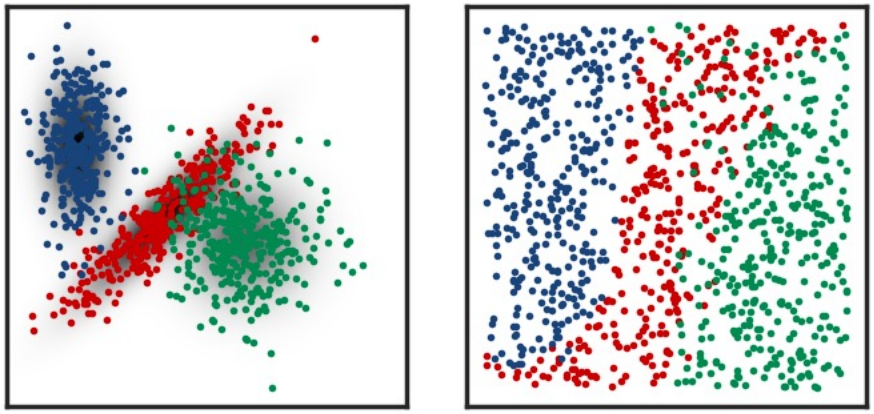
Independent (Beta distributions)



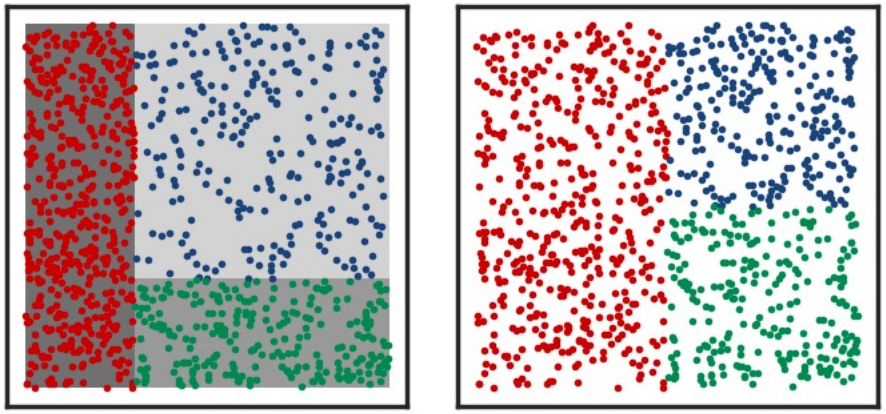
Multivariate Gaussian



Gaussian Mixture



Decision Tree Density



Data before (left) and after (right) transformation via corresponding density destructor.
Note: Color is just to show correspondence between areas before and after transformation.

Examples of simple closed-form destructors

Description	Density	Transformation
Autoregressive Density	$\prod_{s=1}^{d} \mathbb{P}_s(x_s \mid \boldsymbol{x}_{1:s-1})$	$[F_1(x_1), F_2(x_2 \mid x_1), \cdots, F_d(x_d \mid \boldsymbol{x}_{1:s-1})]$
Mixture of Gaussians Conditionals (e.g. MADE, MAF)	$\prod_{s=1}^{d} \left[\sum_{t=1}^{m} \pi_t(\boldsymbol{x}_{1:s-1}) \times \mathbb{P}_{\mathcal{N}}(x_s \mid \mu_{st}(\boldsymbol{x}_{1:s-1}), \sigma_{st}^2(\boldsymbol{x}_{1:s-1})) \right]$	$\left[F_1(x_1), F_2(x_2 \mid x_1), \cdots, F_d(x_d \mid x_1, \cdots, x_{s-1}) \right]$
Block Gaussian Conditionals (e.g. Real NVP, NICE)	$\mathbb{P}_{\mathcal{N}}(\boldsymbol{x}_{1:t} \mid 0, \mathbf{I}) \times \mathbb{P}_{\mathcal{N}}(\boldsymbol{x}_{t+1:d} \mid \boldsymbol{\mu}(\boldsymbol{x}_{1:t}), \boldsymbol{\sigma}^2(\boldsymbol{x}_{1:t}))$	$\left[\Phi(\boldsymbol{x}_{1:t}), \Phi(\frac{x_{t+1} - \mu_{t+1}(\boldsymbol{x}_{1:t})}{\sigma_{t+1}(\boldsymbol{x}_{1:t})}), \cdots, \Phi(\frac{x_d - \mu_d(\boldsymbol{x}_{1:t})}{\sigma_d(\boldsymbol{x}_{1:t})}) \right]$
Linear Projection Density	$\mathbb{P}_{\psi}(W\boldsymbol{x})$	$D_{\theta}(W\boldsymbol{x})$
Independent Components (e.g. Gaussianization via ICA)	$\prod_{s=1}^{d} \mathbb{P}_s(\boldsymbol{w}_s^T \boldsymbol{x})$	$\boldsymbol{F}(W\boldsymbol{x})$
Gaussian (e.g. via PCA)	$\mathbb{P}_{\mathcal{N}}(\boldsymbol{x} \mid \boldsymbol{\mu}, \Sigma)$	$\boldsymbol{\Phi}(\Sigma^{-\frac{1}{2}}(\boldsymbol{x} - \boldsymbol{\mu}))$
Copula-based Density	$\mathbb{P}^{\text{cop}}(\boldsymbol{F}(\boldsymbol{x})) \prod_{s=1}^{d} \mathbb{P}_s(x_s)$	$D_{\theta}(\boldsymbol{F}(\boldsymbol{x}))$
Gaussian Copula	$\mathbb{P}_R^{\mathcal{N}\text{-cop}}(\boldsymbol{F}(\boldsymbol{x})) \prod_{s=1}^{d} \mathbb{P}_s(x_s)$	$\boldsymbol{\Phi}(R^{-\frac{1}{2}} \boldsymbol{\Phi}^{-1}(\boldsymbol{F}(\boldsymbol{x})))$
Gaussian Mixture (note that $F_s(x_s \mid \boldsymbol{x}_{-s})$ is computable)	$\sum_{t=1}^{m} \pi_t \mathbb{P}_{\mathcal{N}}(\boldsymbol{x})$	$[F_1(x_1), F_2(x_2 \mid x_1), \cdots, F_d(x_d \mid x_1, \cdots, x_{s-1})]$
Examples of new destructors enabled by our unified destructor framework		
Piecewise Density (or Tree Density)	$\{\mathbb{P}_{\psi_\ell}(\boldsymbol{x}), \text{ if } \boldsymbol{x} \in \mathcal{L}_\ell\}$, where $\mathcal{L}_\ell$ are the disjoint subspaces of the leaves.	$\{D_{\theta_\ell}(\boldsymbol{x}), \text{ if } \boldsymbol{x} \in \mathcal{L}_\ell\}$
Piecewise Uniform (e.g. DET)	$\{c_\ell, \text{ if } \boldsymbol{x} \in \mathcal{L}_\ell\}$	$\{\text{diag}(\boldsymbol{a}_\ell)\boldsymbol{x} + \boldsymbol{b}_\ell, \text{ if } \boldsymbol{x} \in \mathcal{L}_\ell\}$
Image-Specific Feature Pairs	$\prod_{P \in \mathcal{P}} \mathbb{P}_P(x_{P(1)}, x_{P(2)})$, where feature pairs $\mathcal{P}$ are based on pixel locality.	$\{D_P(x_{P(1)}, x_{P(2)}), \forall P \in \mathcal{P}\}$

Density destructor algorithm performs greedy layer-wise construction of deep destructor

1. Simple density estimation (GMM, Gaussian, tree density, etc.)

$$Q^t \leftarrow \arg\min_{Q \in \mathcal{Q}} KL\left(P(x^{t-1}), Q(x^{t-1})\right)$$

2. Map density to simple destructor layer

$$d^t = \Omega(Q^t)$$

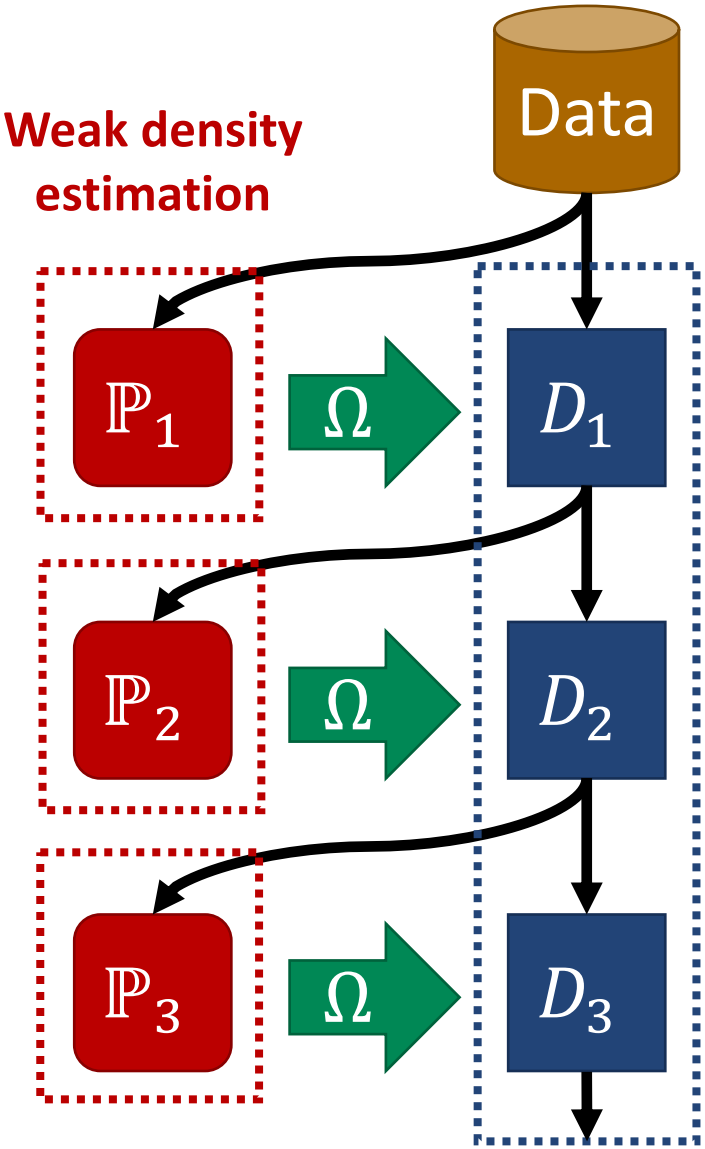
3. Transform data for next layer

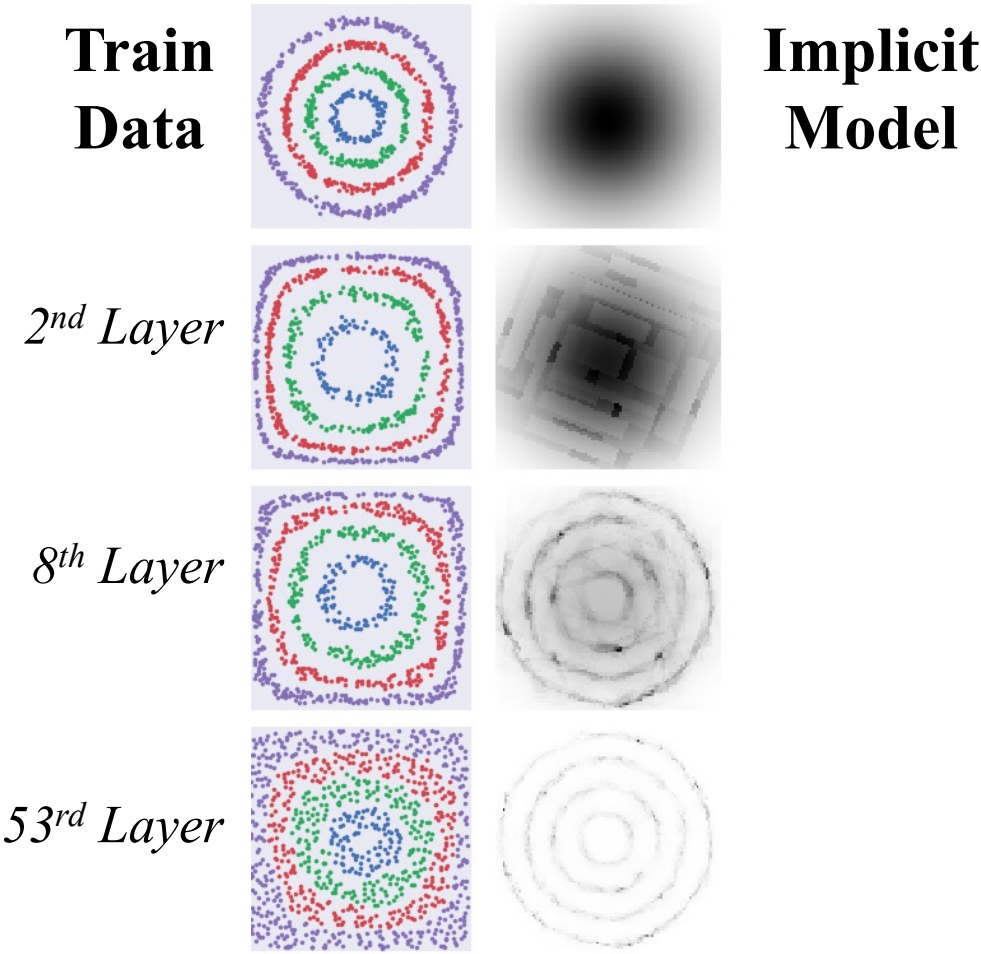
$$x^t = d^t(x^{t-1})$$

4. Update deep destructor

$$D^t = d^t \circ D^{t-1}$$

Deep density destructors via sequence of *weak* destructors





Destructor algorithm can be shown to monotonically decrease the negative log likelihood after every iteration/layer

- The destructive learning objective, where $z = D(x)$, and $U(z)$ is the uniform density function

$$\arg\min_{D} KL\left(P_z(z), U(z)\right)$$

- Want: Every iteration decreases objective:

$$KL\left(P_{d^t \circ D^{t-1}(x)}, U\right) \leq KL\left(P_{D^{t-1}(x)}, U\right)$$

- Let $x = D^{t-1}\left(x^{(0)}\right)$ and $z = d^t(x) = d^t\left(D^{t-1}\left(x^{(0)}\right)\right)$
- $KL\left(P_z(z; D), U(z)\right)$
- $= KL\left(P_x(x), Q_x(x; D)\right)$

  (KL equivalence lemma)
- $\leq KL\left(P_x(x), Q_x(x; D = Id)\right)$

  (minimization is better than one particular)
- $= KL\left(P_x(x), |J_D(x)| U\left(D(x)\right)\right)$ (Expand in terms of *D*)
- $= KL\left(P_x(x), U(x)\right)$

Reuse results: Deep density destructors can be built from simple and well-understood components

- MNIST $d = 784$
- CIFAR-10 $d = 3072$

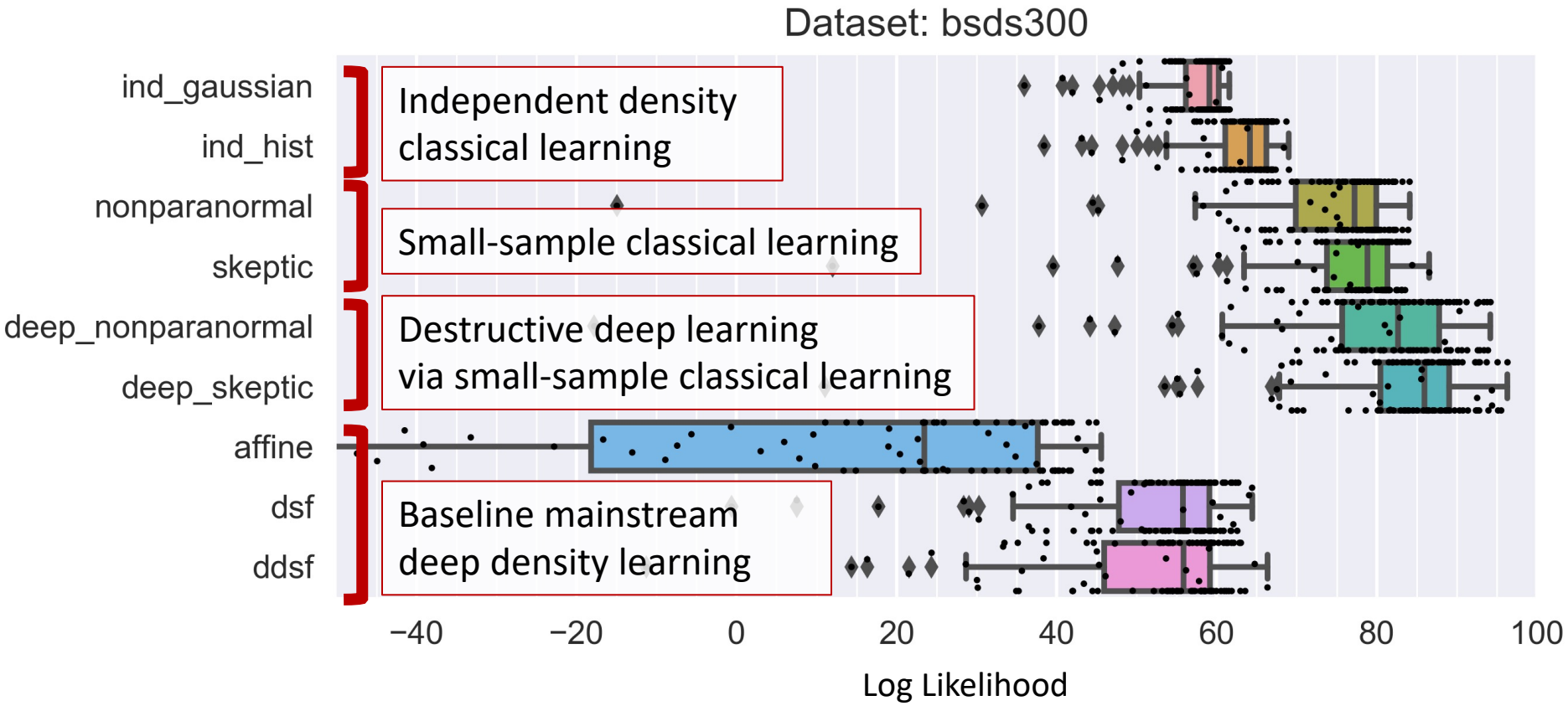
- Autoregressive flow baselines (DNN-based)
  - MADE [Germain et al., 2015]
  - Real NVP [Dinh, et al. 2017]
  - MAF [Papamakarios et al. 2017]

- Our deep copula method
  - PCA + histograms

	MNIST			CIFAR-10		
	LL	D	T	LL	D	T
Models from MAF paper computed on Titan X GPU						
Gaussian	-1367	1	0.0	2367	1	0.0
MADE	-1385	1	0.0	448	1	0.2
MADE MoG	-1042	1	0.1	-53	1	0.3
Real NVP	-1329	5	0.2	2600	5	1.4
Real NVP	-1765	10	0.2	2469	10	1.0
MAF	-1300	5	0.1	2941	5	3.7
MAF	-1314	10	0.2	**3054**	10	7.5
MAF MoG	-1100	5	0.2	2822	5	3.9
Our proposed destructors computed on 10 CPUs						
Copula	-1028	5	0.2	2626	17	10.1

LL = Log Likelihood (higher is better)
D = # of layers, T = Time

Modularity enables classical learning improvements to carry over to deep learning



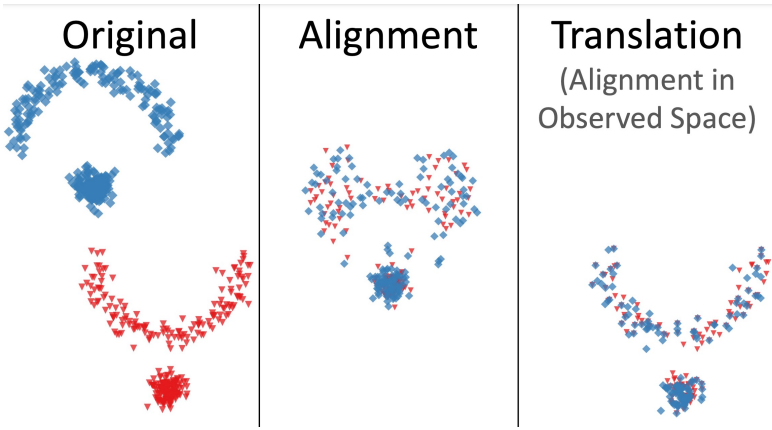
Small-sample experiment where number of dimensions is 63 and number of training samples is 30. Notice how mainstream deep learning fails in this setting.

Limitations of destructive modular learning

- Unlikely to perform as well as joint learning
  - Greedy vs joint optimization
  - Local vs global optimization
- Must create destructor mapping Ω, which can be challenging
- Often requires more layers to achieve similar result because of optimization
- Normalizing flows transform to simple known distribution
  - What about transforming between any two distributions?

Translating or aligning two arbitrary distributions is a more general task

- Flow-based methods for alignment/translation
  - Likelihood-based: AlignFlow[1], LRMF[2]
  - Optimal Transport: Iterative Alignment Flow



- Iterative Alignment Flow: (Greedy) fast and easy alignment of multiple distributions!

[1] Aditya Grover, Christopher Chute, Rui Shu, Zhangjie Cao, Stefano Ermon: AlignFlow: Cycle Consistent Learning from Multiple Domains via Normalizing Flows. AAAI 2020: 4028-4035
[2] Ben Usman, Avneesh Sud, Nick Dufour, Kate Saenko: Log-Likelihood Ratio Minimizing Flows: Towards Robust and Quantifiable Neural Distribution Alignment. NeurIPS 2020

Iterative Alignment Flow: Extending iterative approach to translation/alignment task

- Motivation: Hard to directly align two or more distributions in a high dimensional space
- Proposed solution: Decompose the high dimensional problem into a series of **simple** 1D problems
- Background
  - Optimal transport in 1D
  - Max-sliced Wasserstein Distance (max-SWD)
- Objective and iterative algorithm
  - Min-max optimization (different from GAN)
  - Iterative algorithm

Background: Optimal transport problems in 1D are known in closed-form!

- Computing the Wasserstein distance in 1D is known in **closed-form**

$$W_1(\hat{p}_x, \hat{q}_y) = \sum_{i=1}^{n} |x_i - y_i|$$

  - Where the data is **sorted** $x_1 \leq x_2 \leq \cdots \leq x_n$ and $y_1 \leq y_2 \leq \cdots \leq y_n$
- The optimal Monge map is known in **closed-form**

$$T^*(x) = F_Y^{-1}\left(F_X(x)\right)$$

Background: Max sliced Wasserstein distance simplifies to a "worst case" 1D Wasserstein problem

- Recall Wasserstein-1 Distance

$$W_1(p_X, p_Y) = \begin{pmatrix} \min_T \mathbb{E}_{p_X}[\|x - T(x)\|_1] \\ \text{s.t.} \quad p_{T(X)} = p_Y \end{pmatrix}$$

- Max sliced Wasserstein-1 divergence

$$max\_SW_1(p_X, p_Y) = \max_{\theta:\|\theta\|_2=1} W_1\left(p_{X^T\theta}, p_{Y^T\theta}\right)$$

  - Where θ defines the direction of the projection/slice
  - Note that this finds the **largest** difference along a **1D** projection/slice

- The max-SW is 0 if and only if the distributions are aligned

Iterative alignment flow objective is a different type of min-max optimization

- The objective becomes a min-max optimization:

$$\min_{T_X, T_Y} max\_SW_1\left(p_{T_X(X)}, p_{T_Y(Y)}\right)$$

$$= \min_{T_X, T_Y} \max_{\theta:\|\theta\|_2=1} W_1\left(p_{T_X(X)^T\theta}, p_{T_Y(Y)^T\theta}\right)$$

- The inner optimization tries to find the "worst-case" projection that maximizes the divergence.
- The outer optimization tries to find the transformation that minimizes this "worst-case" divergence.
- Theoretical minimum occurs when distributions are aligned per the property of max-SW

Algorithm: Iteratively solve min and max problems

- Find structure θ by solving max problem

$$\theta^{(i)} = \arg\max_{\theta:\|\theta\|_2=1} W_1\left(p_{X^T\theta}, p_{Y^T\theta}\right)$$

- Fix $\theta^{(i)}$ and solve for simple OT Monge maps via **closed-form solutions** (i.e., Ω)

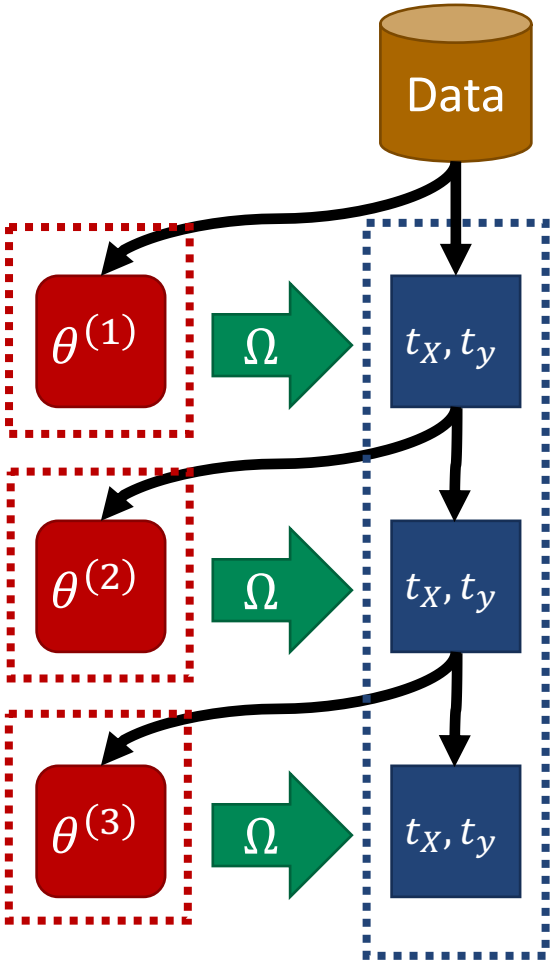
$$t_X^{(i)}, t_Y^{(i)} = \arg\min_{t_X, t_Y} W_1\left(p_{t_X(X)^T\theta^{(i)}}, p_{t_Y(Y)^T\theta^{(i)}}\right)$$

- Transform data

$$X \leftarrow t_X^{(i)}(X), Y \leftarrow t_Y^{(i)}(Y)$$

- Update deep transforms

$$T_X^{(i)} \leftarrow t_x^{(i)} \circ T_X^{(i-1)}$$



Iterative alignment flows can be used to translate between multiple domains via this simple iterative algorithm