Deep Density Destructors (from a biased viewpoint)

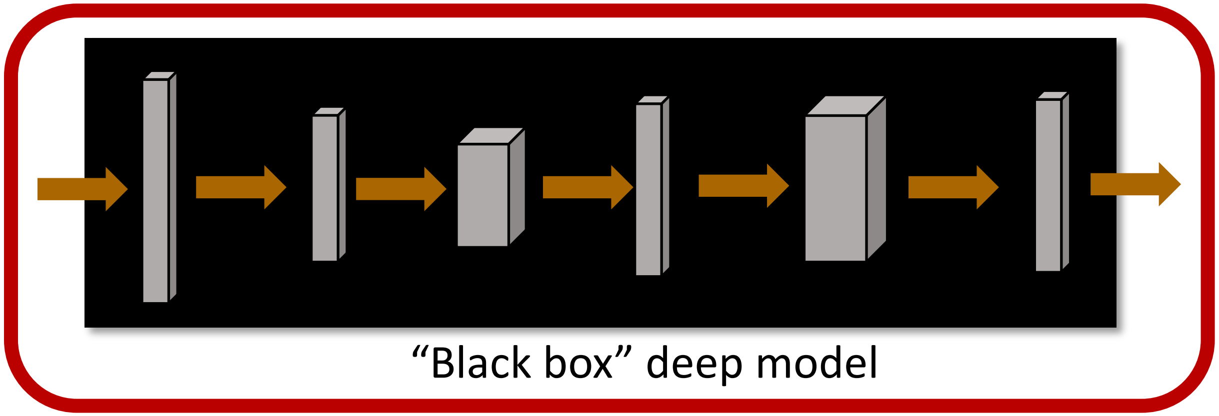
David I. Inouye

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

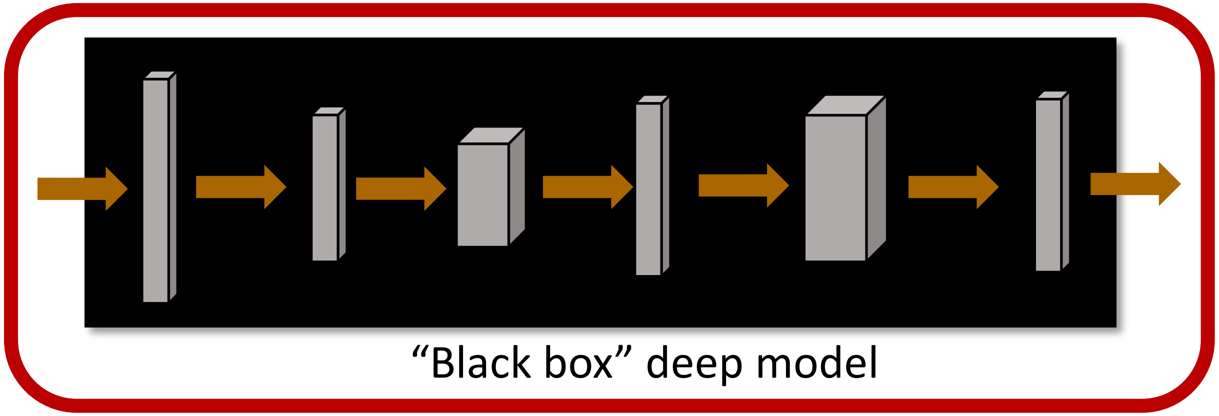


- Real NVP
- MAF
- GLOW
- Etc.

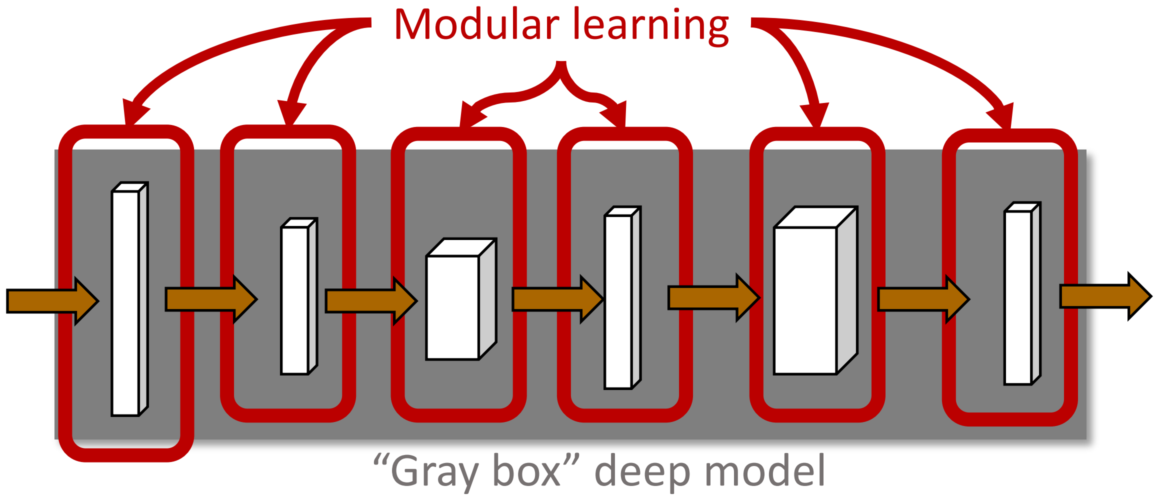
“Gray box” deep model

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

End-to-end learning



- Real NVP
- MAF
- GLOW
- Etc.



- 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

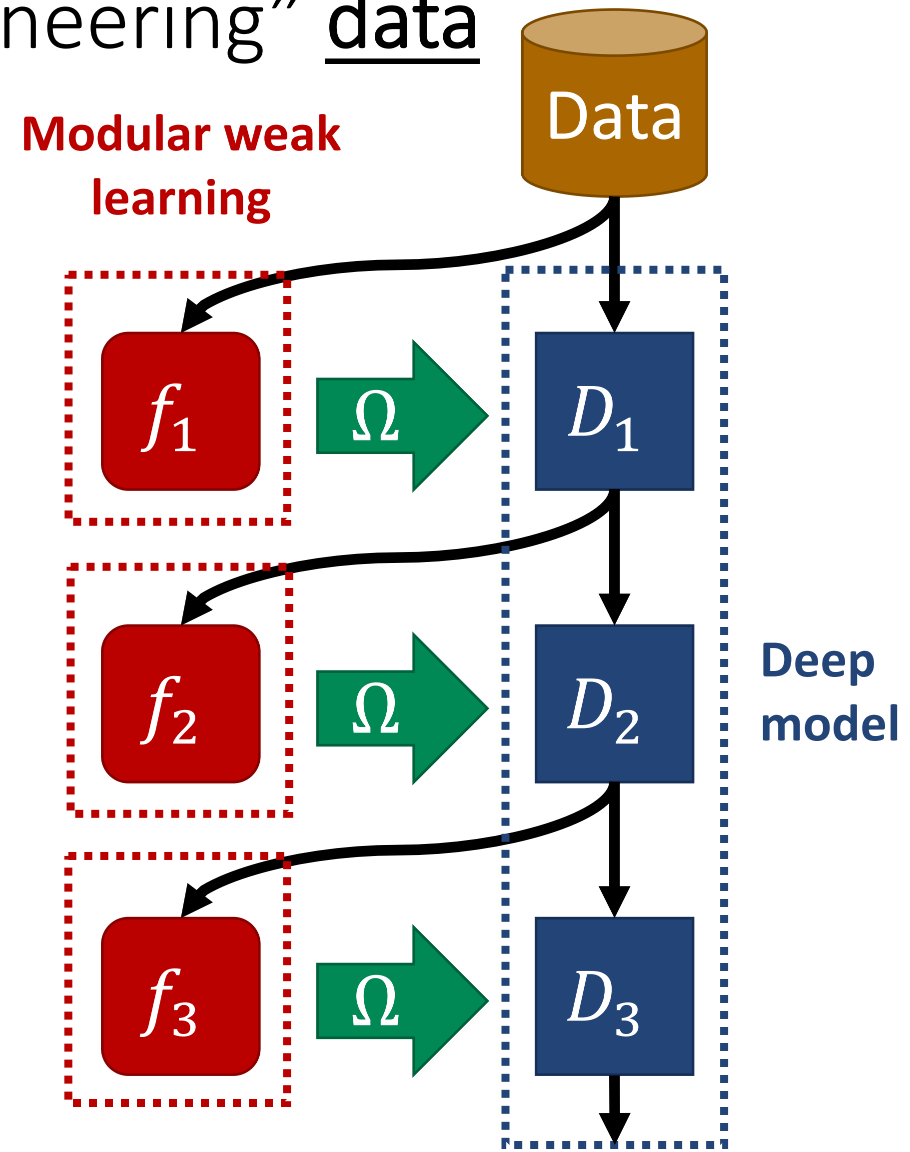
1. **Find** *part* to take off using **understanding and expertise**
2. **Determine how** to take off part in a *reversible* way (e.g., unscrewing bolts)
3. **Remove part**
4. **Repeat**

Reverse engineering data

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



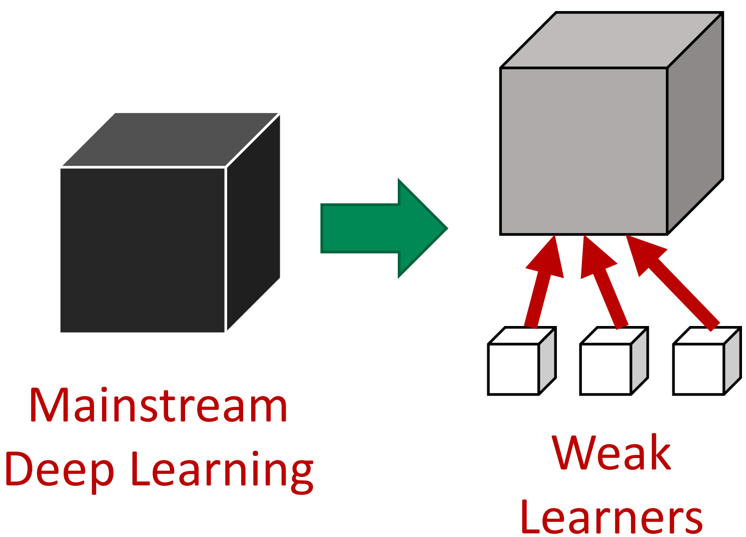
Why use modular weak learning for deep models?

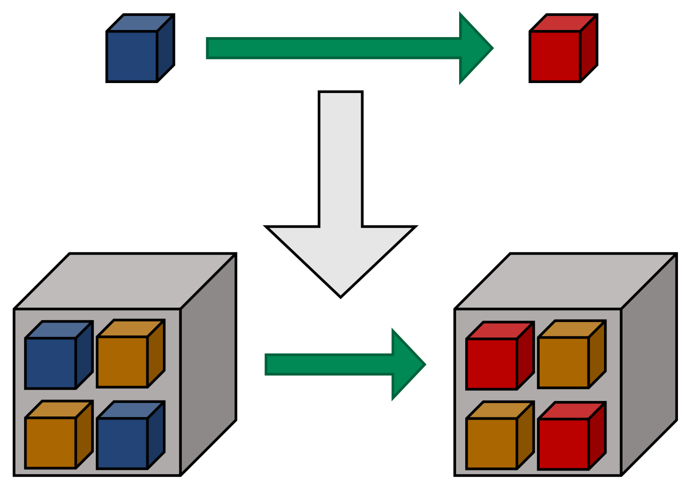
Reuse

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

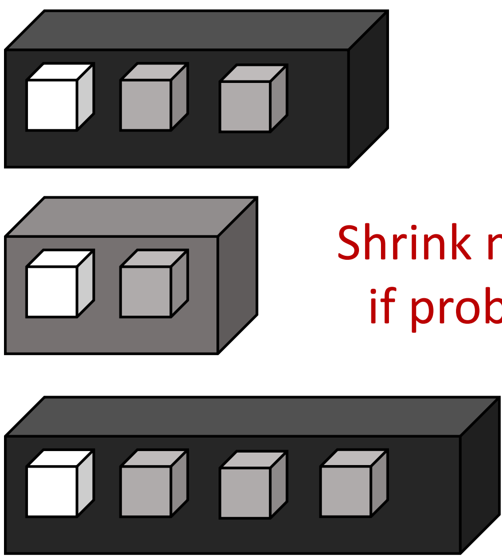




Why use modular weak learning for deep models?

Algorithmic Interpretability

Increasing or decreasing model complexity is straightforward

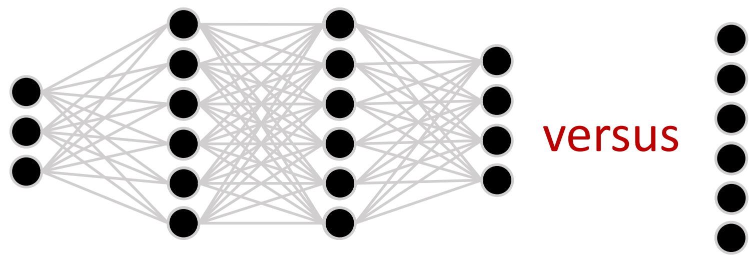


Shrink model if problem

Grow if more data

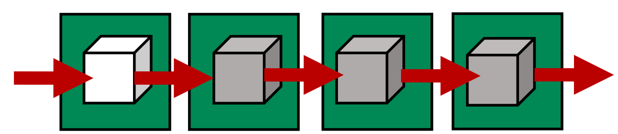
Resource Constraints

Layer-wise training (memory bottleneck)



Pipelined training (computation bottleneck)

Shallow/weak online learners



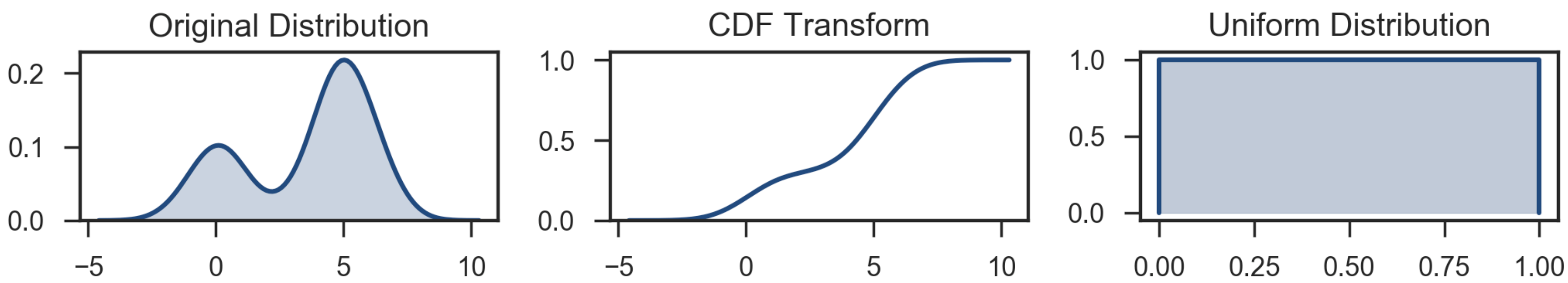
Distributed on different processors or devices

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

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 $D_X(X)$ is maximum entropy.
- A *density destructor* is an invertible transformation such that

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

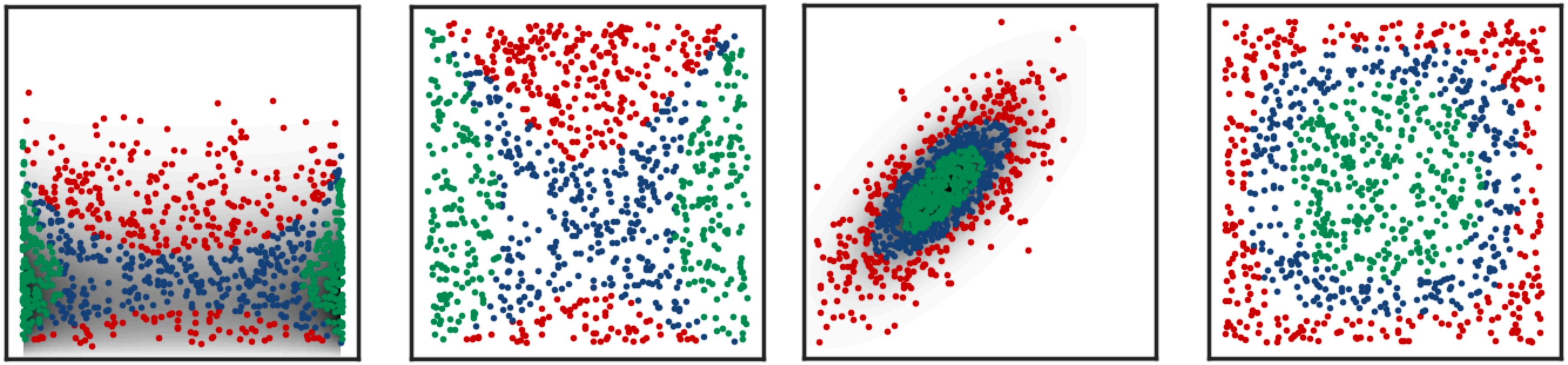
$$D_X(X) \sim \text{Uniform}\big([0,1]^d\big)$$

  - $\Omega^{-1}(D_X) = \left|\det J_{D_X}\right| = \mathbb{P}_X$ ← 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

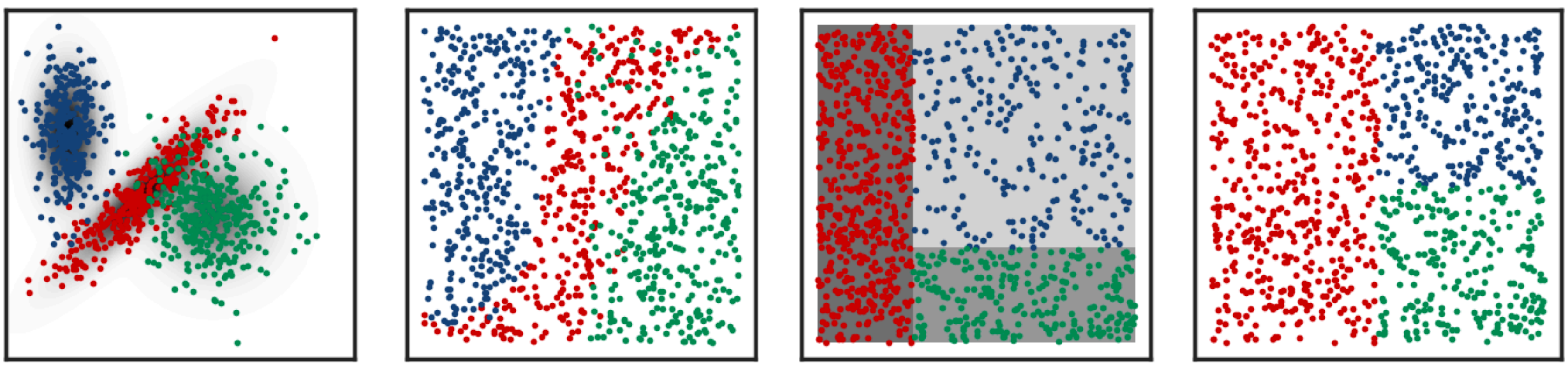
Independent (Beta distributions)

Multivariate Gaussian



Gaussian Mixture

Decision Tree Density

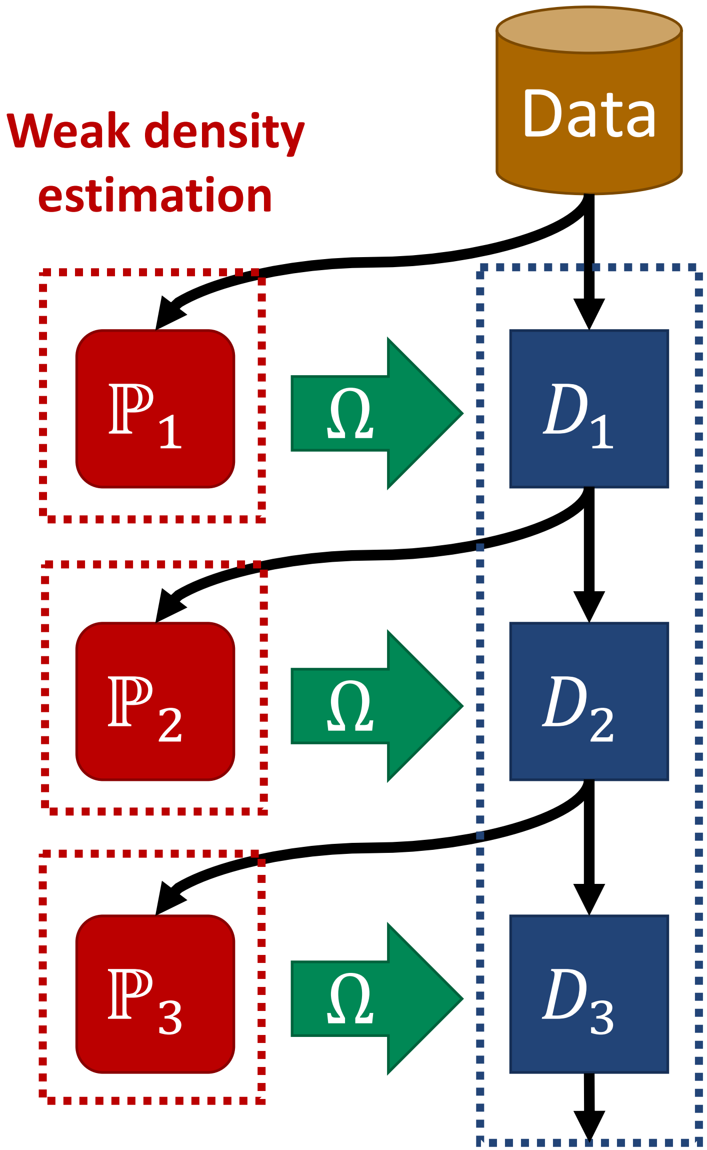


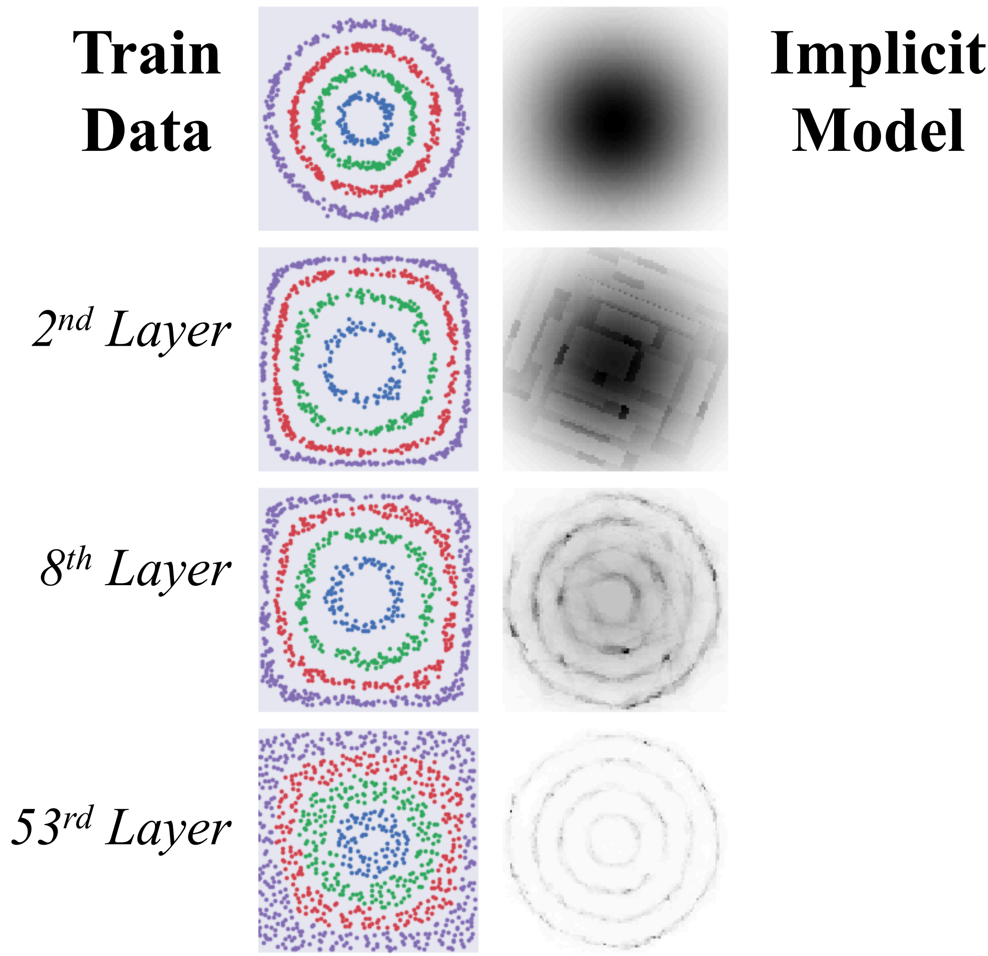
Data before (left) and after (right) transformation via corresponding density destructor.
Note: Color is just to show correspondence between areas before and after transformation.

Example Destructors

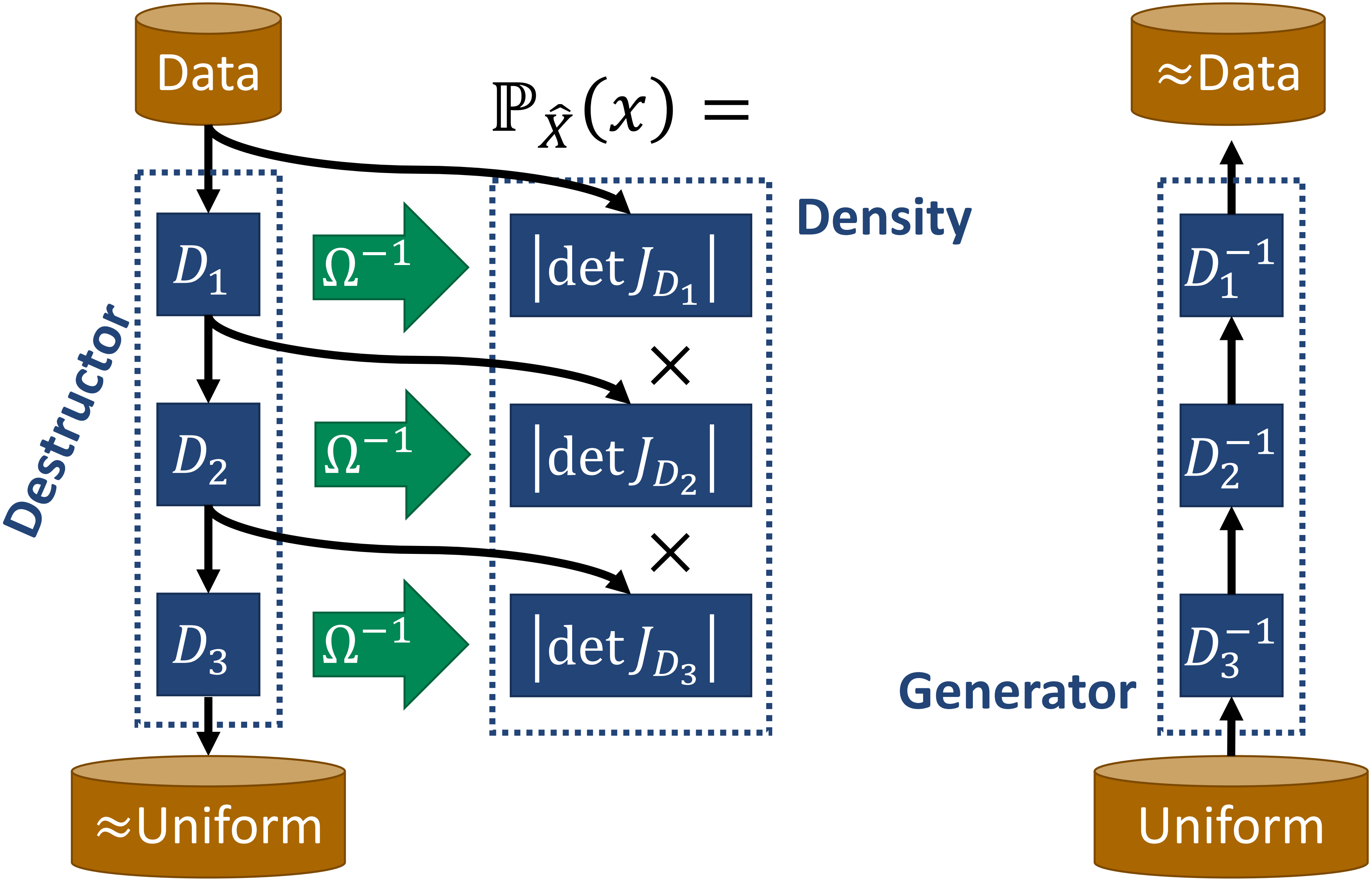
Description	Density	Transformation
Autoregressive Density	$\prod_{s=1}^{d} \mathbb{P}_s(x_s \mid \boldsymbol{x}_{1:s-1})$	$[F_1(x_1), F_2(x_2 \mid x_1), \cdots, F_d(x_d \mid \boldsymbol{x}_{1:s-1})]$
Mixture of Gaussians Conditionals (e.g. MADE, MAF)	$\prod_{s=1}^{d} \left[\sum_{t=1}^{m} \pi_t(\boldsymbol{x}_{1:s-1}) \times \mathbb{P}_{\mathcal{N}}(x_s \mid \mu_{st}(\boldsymbol{x}_{1:s-1}), \sigma_{st}^2(\boldsymbol{x}_{1:s-1})) \right]$	$\left[F_1(x_1), F_2(x_2 \mid x_1), \cdots, F_d(x_d \mid x_1, \cdots, x_{s-1})\right]$
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.	$\{D_P(x_{P(1)}, x_{P(2)}), \forall P \in \mathcal{P}\}$

Deep density destructors via sequence of *weak* destructors





Density computation and sample generation



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

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

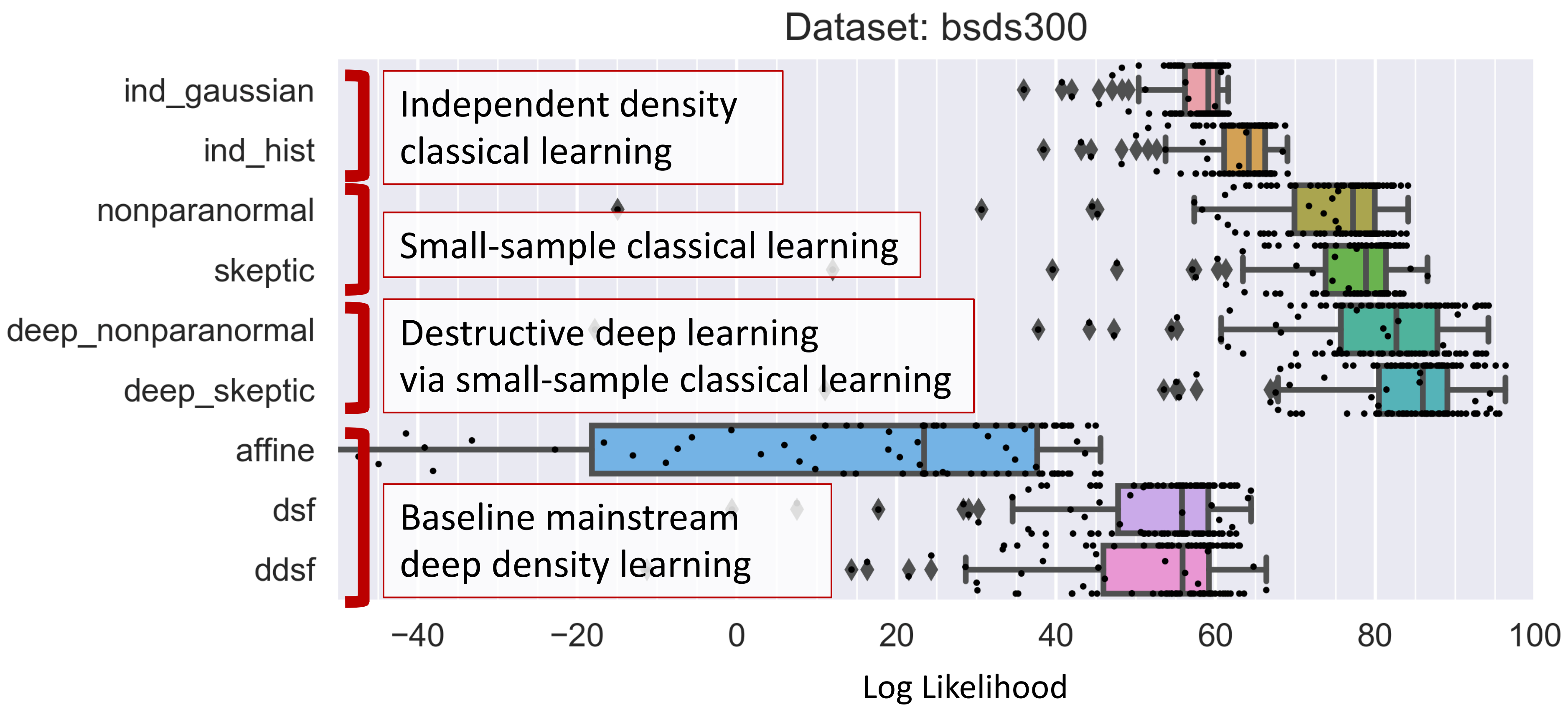
- Autoregressive flow baselines (DNN-based)
  - MADE [Germain et al., 2015]
  - Real NVP [Dinh, et al. 2017]
  - MAF [Papamakarios et al. 2017]

- 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.

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

1. Simple density estimation (GMM, Gaussian, tree density, etc.)
   $$Q^t \leftarrow \arg\min_{Q \in \mathcal{Q}} KL\big(P(x^{t-1}), Q(x^{t-1})\big)$$
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
   $$D^t = d^t \circ D^{t-1}$$

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 $U_z(z)$ is the uniform density function
$$\arg\min_D KL\big(P_z(z), U_z(z)\big)$$
- KL equivalence lemma, let $z = D(x)$
$$KL\big(P_x(x), Q_x(x)\big) = KL\big(P_z(z), Q_z(z)\big)$$
- Simple corollary is that objective above is MLE:
$$\arg\min_D KL\left(P_x(x), \hat{P}_x(x)\right)$$
$$\arg\min_D KL\left(P_x(x), |J_D(x)| U_z(D(x))\right)$$
$$\arg\min_D KL(P_x(x), |J_D(x)|)$$

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 $U_z(z)$ is the uniform density function

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

- Want: Every iteration decreases objective:

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

- Let $Q(z) \equiv U(z)$, $x = D^{t-1}(x)$ and $z = D^t(x)$

$$KL\left(P(z), U(z)\right) = KL\left(P(z), Q(z)\right)$$
$$= KL\left(P(x), Q(x)\right) \leq KL\left(P(x), U(x)\right)$$