Path to Neural Networks II

Image courtesy Vogel et al. [2018], Gharbi et al. [2019]

Philipp Slusallek   *Karol Myszkowski*   ***Gurprit Singh***

Today's Menu

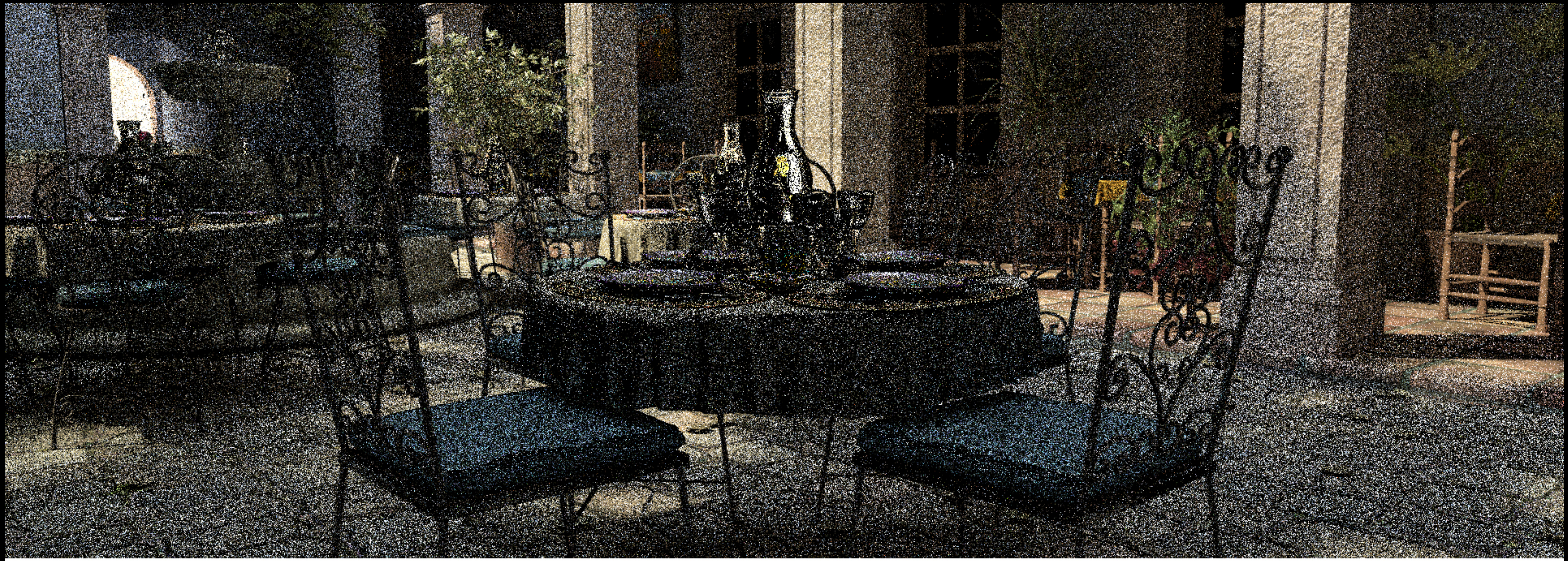
Sample-based denoising

CNN-based approach to generate blue-noise samples

Normalizing Flows

Path guiding using Normalizing Flows

Recap



input Monte Carlo (8 samples/pixel)

after RPF (8 samples/pixel)

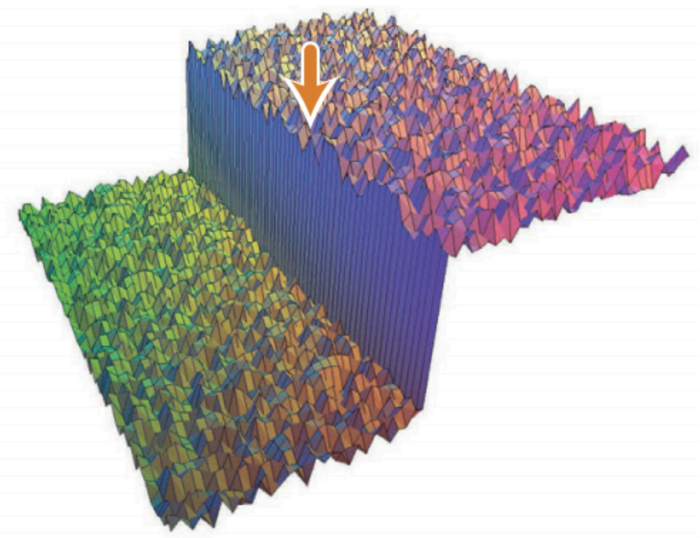
(a) Input MC (8 spp) **(b)** Dependency on (u, v) **(c)** Our approach (RPF)

Bilateral Filtering

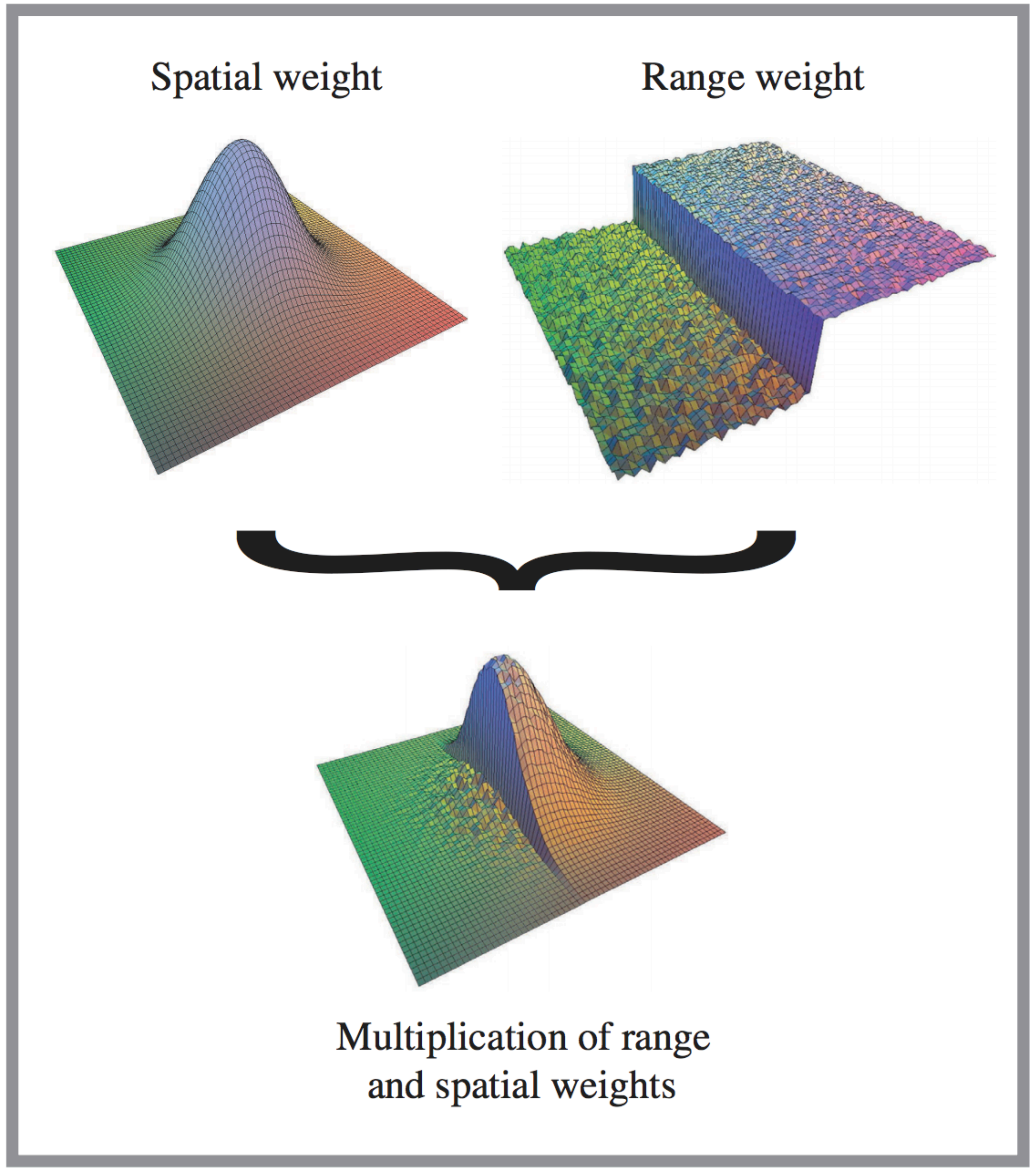
$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_{\mathrm{s}}}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_{\mathrm{r}}}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

$$W_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_{\mathrm{s}}}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_{\mathrm{r}}}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)$$

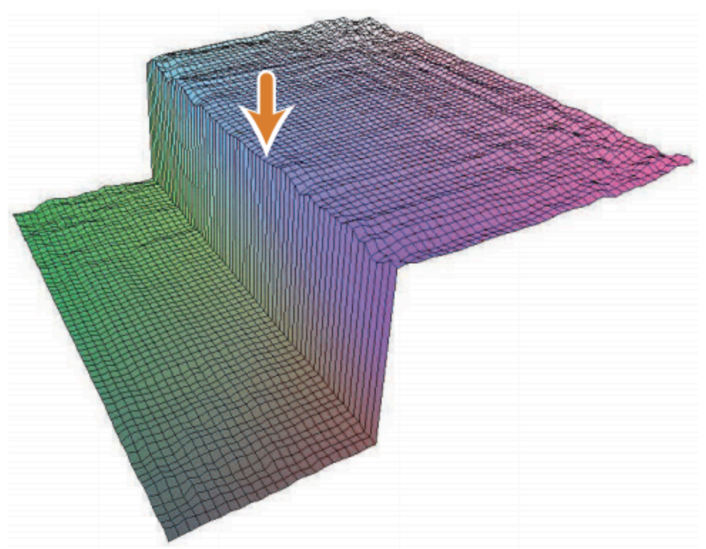
Input

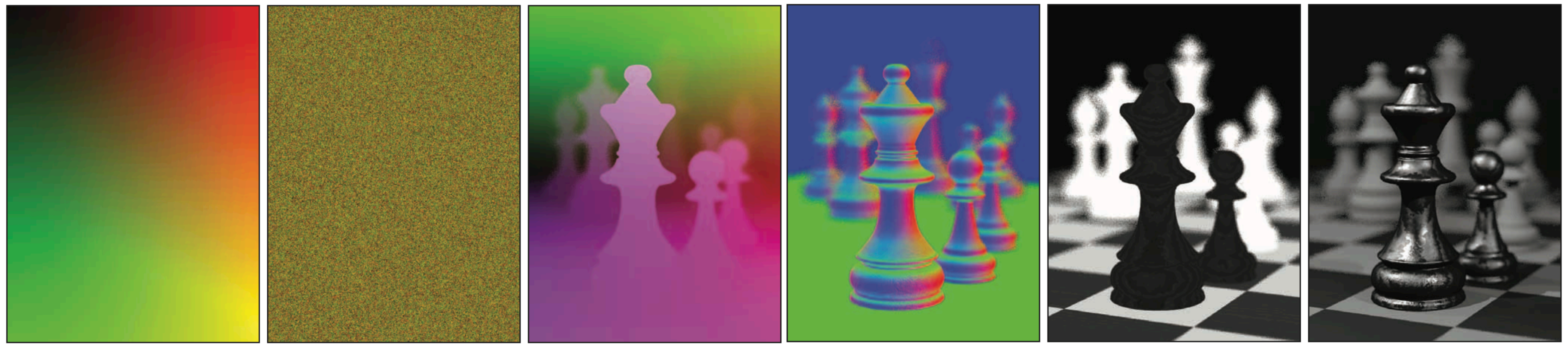


Bilateral filter weights at the central pixel



Result





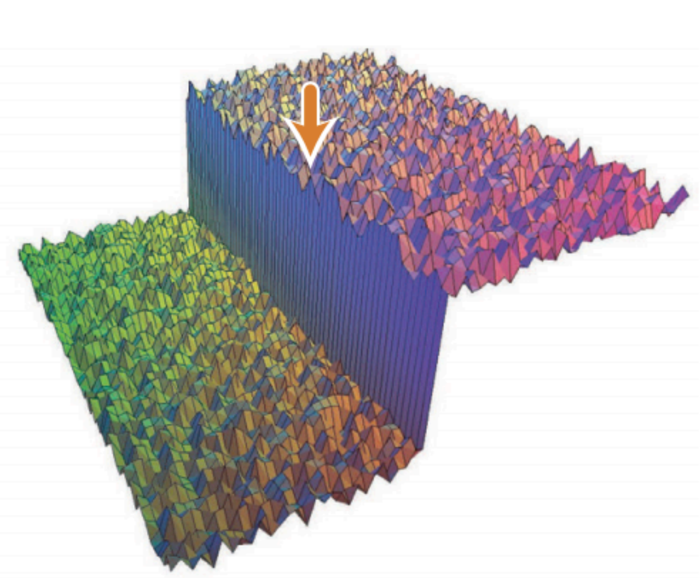
(a) Screen position **(b)** Random parameters **(c)** World space coords. **(d)** Surface normals **(e)** Texture value **(f)** Sample color

Bilateral Filtering

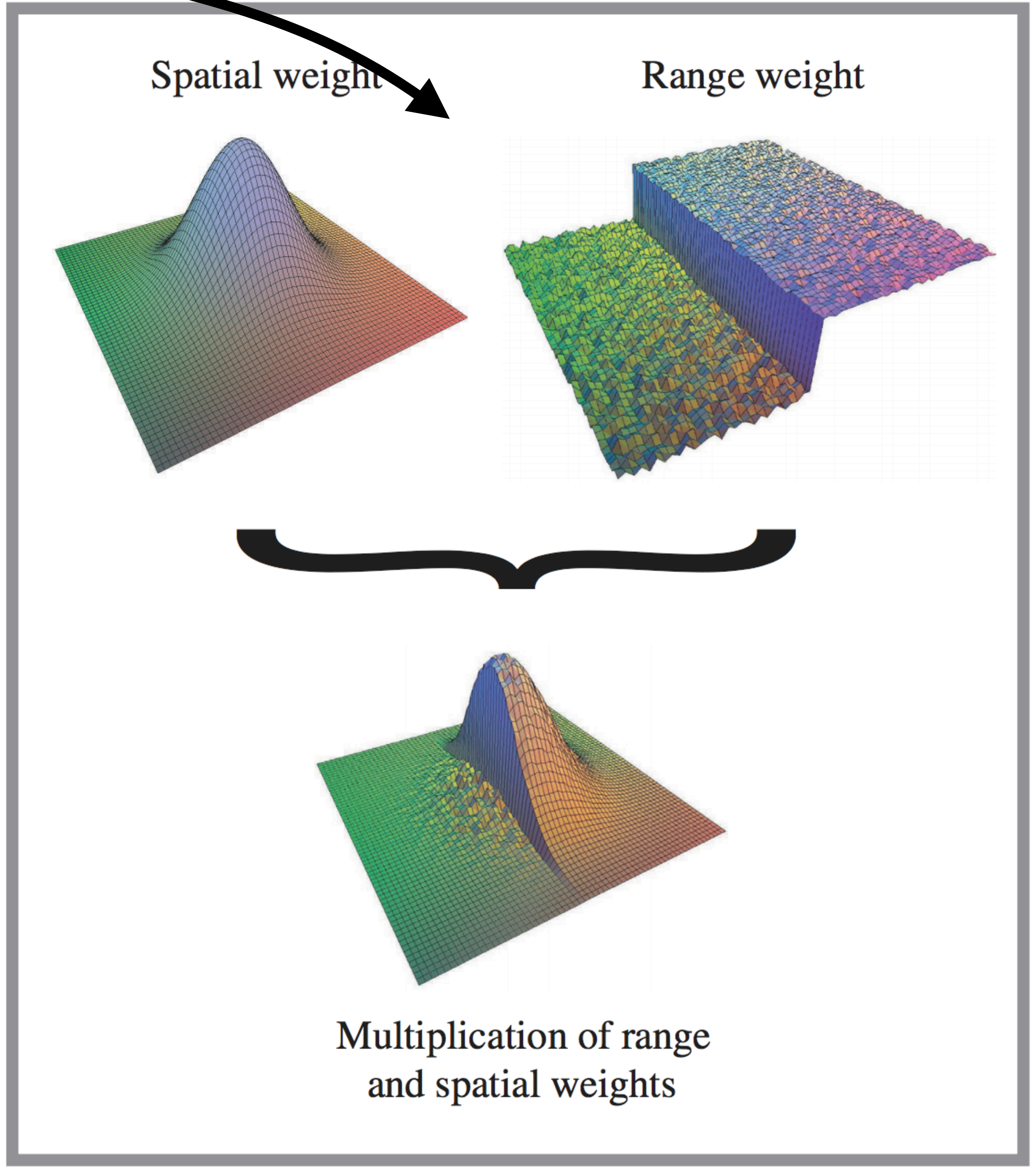
$$BF[I]_{\mathbf{p}} = \frac{1}{\boxed{W_{\mathbf{p}}}} \sum_{\mathbf{q}\in\mathcal{S}} \boxed{G_{\sigma_{\mathrm{s}}}(\|\mathbf{p}-\mathbf{q}\|)}\ \boxed{G_{\sigma_{\mathrm{r}}}(|I_{\mathbf{p}}-I_{\mathbf{q}}|)}\ I_{\mathbf{q}}$$

$$\boxed{W_{\mathbf{p}} = \sum_{\mathbf{q}\in\mathcal{S}} G_{\sigma_{\mathrm{s}}}(\|\mathbf{p}-\mathbf{q}\|)\ G_{\sigma_{\mathrm{r}}}(|I_{\mathbf{p}}-I_{\mathbf{q}}|)}$$

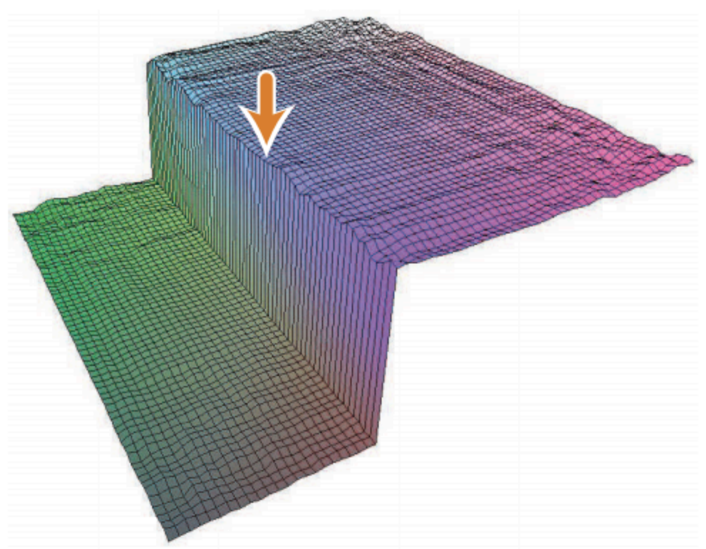
Input

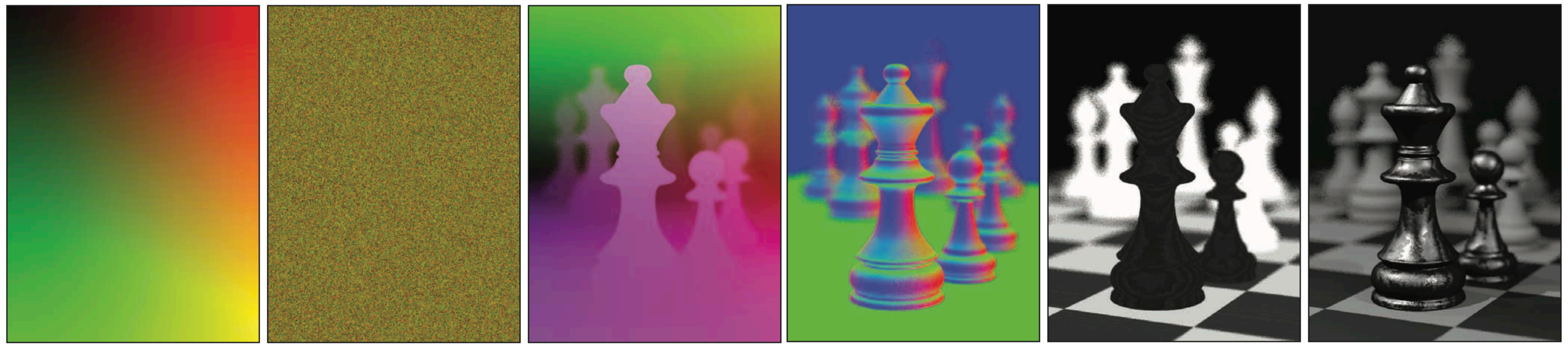


Bilateral filter weights at the central pixel



Result





(a) Screen position **(b)** Random parameters **(c)** World space coords. **(d)** Surface normals **(e)** Texture value **(f)** Sample color

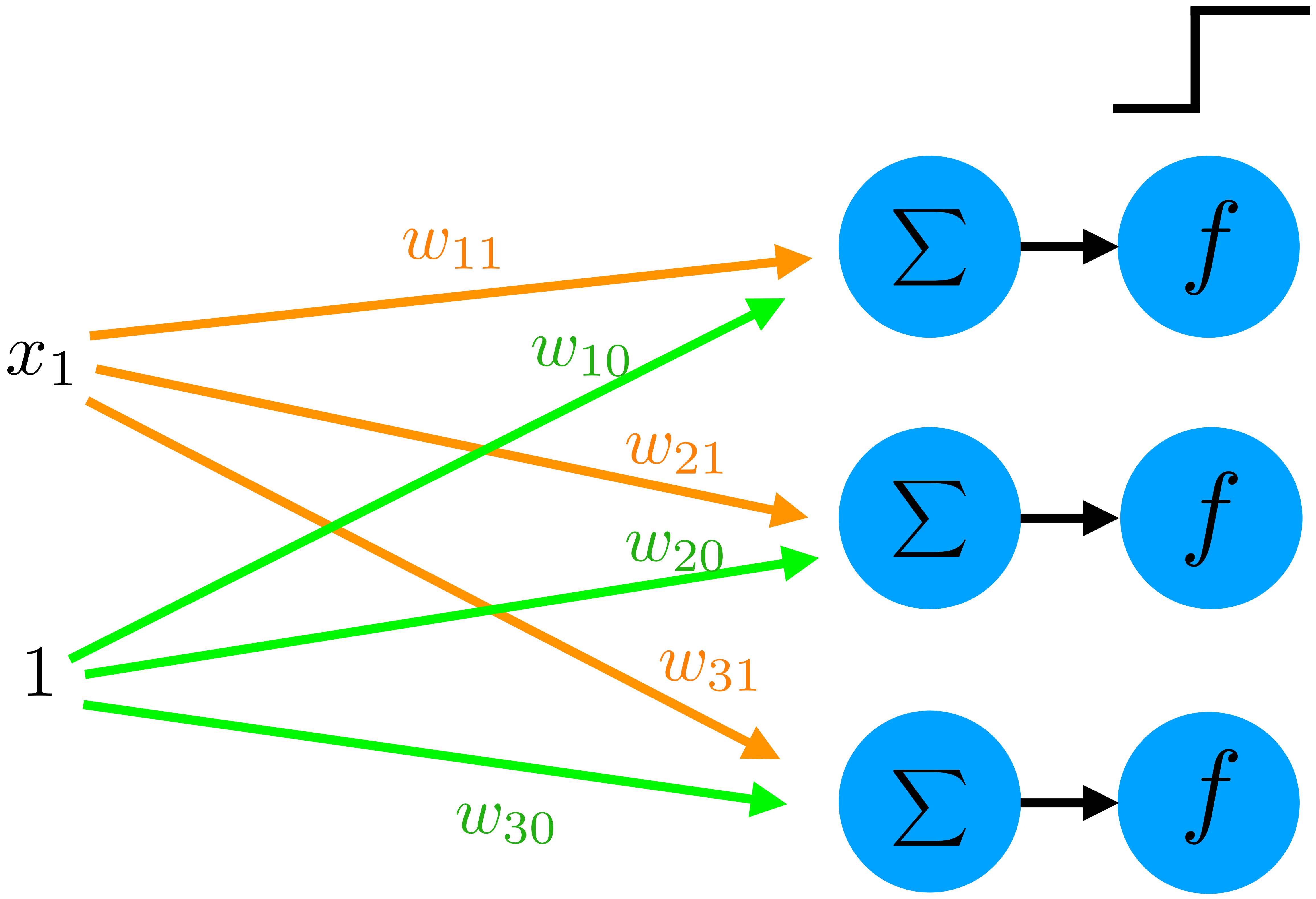
Bilateral Filtering of Features

$$w_{ij} = \exp\Big[-\frac{1}{2\sigma_{\mathbf{p}}^2}\sum_{1\leq k\leq 2}(\bar{\mathbf{p}}_{i,k}-\bar{\mathbf{p}}_{j,k})^2\Big]\times$$

$$\exp\Big[-\frac{1}{2\sigma_{\mathbf{c}}^2}\sum_{1\leq k\leq 3}\boxed{\alpha_k}(\bar{\mathbf{c}}_{i,k}-\bar{\mathbf{c}}_{j,k})^2\Big]\times$$

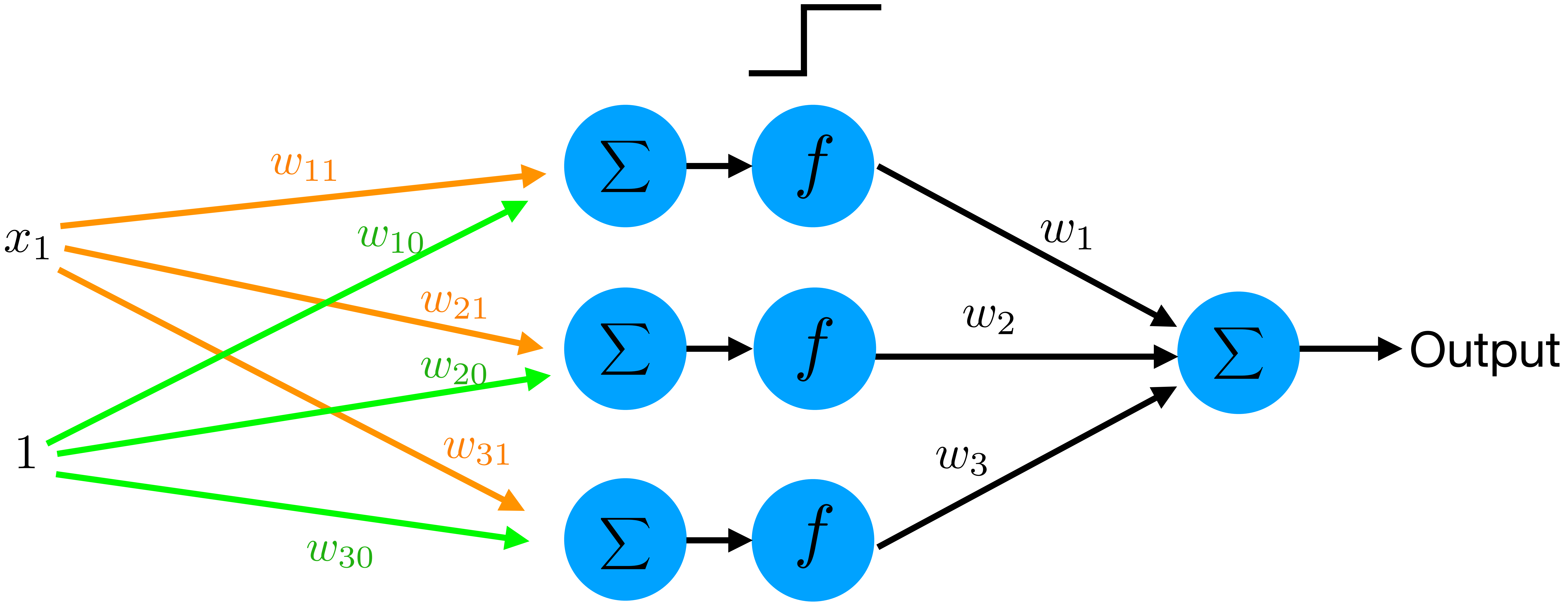
$$\exp\Big[-\frac{1}{2\sigma_{\mathbf{f}}^2}\sum_{1\leq k\leq m}\boxed{\beta_k}(\bar{\mathbf{f}}_{i,k}-\bar{\mathbf{f}}_{j,k})^2\Big],$$

Multi-layer Perceptron



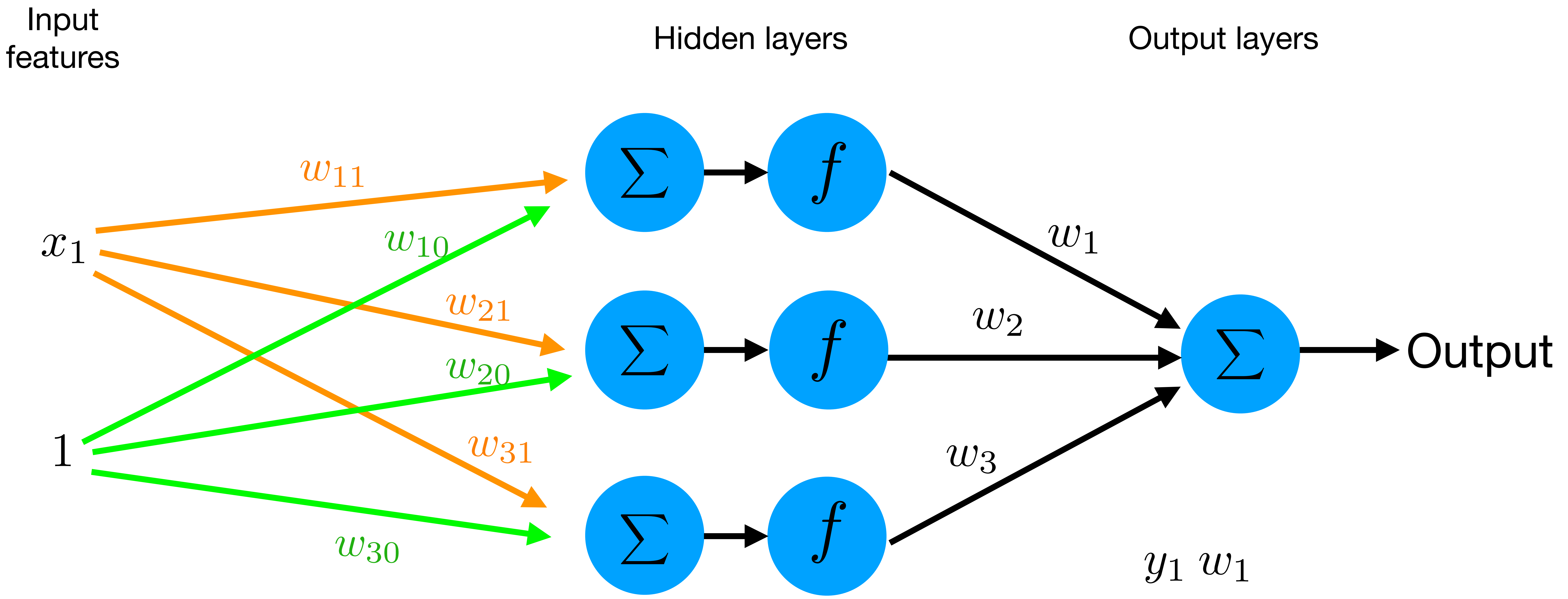
$$y_1 = f(x_1 w_{11} + w_{10})$$
$$y_2 = f(x_1 w_{21} + w_{20})$$
$$y_3 = f(x_1 w_{31} + w_{30})$$

Multi-layer Perceptron



$$
\begin{aligned}
y_1 &= f(x_1 w_{11} + w_{10}) \\
y_2 &= f(x_1 w_{21} + w_{20}) \\
y_3 &= f(x_1 w_{31} + w_{30})
\end{aligned}
$$

Multi-layer Perceptron



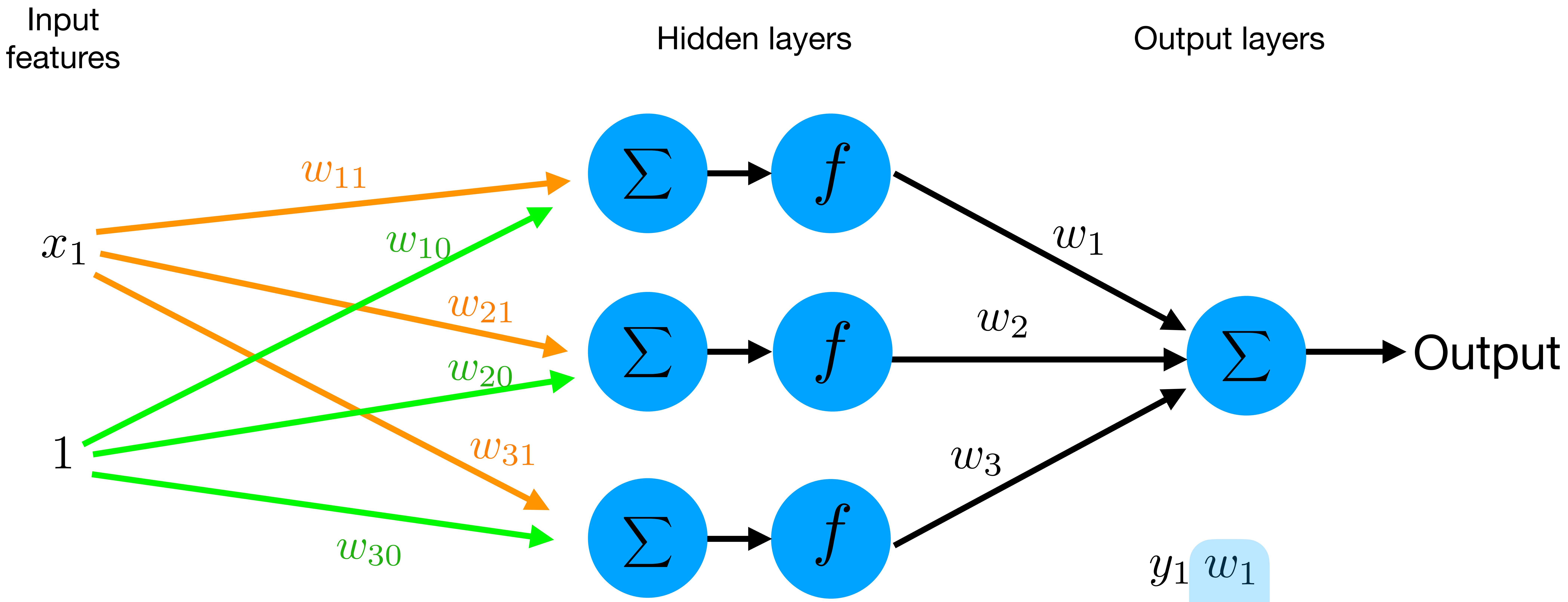
$$y_1 = f(x_1 w_{11} + w_{10})$$
$$y_2 = f(x_1 w_{21} + w_{20})$$
$$y_3 = f(x_1 w_{31} + w_{30})$$

$y_1\ w_1$

$y_2\ w_2$

$y_3\ w_3$

Multi-layer Perceptron



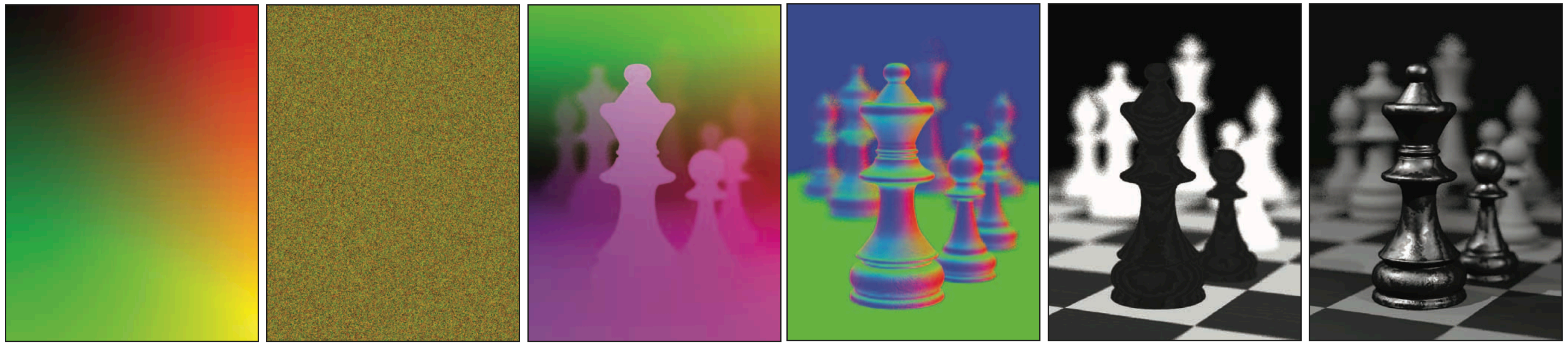
$$y_1 = f(x_1 w_{11} + w_{10})$$
$$y_2 = f(x_1 w_{21} + w_{20})$$
$$y_3 = f(x_1 w_{31} + w_{30})$$

$y_1\ w_1$
$y_2\ w_2$
$y_3\ w_3$

Filter weights

For cross Bilateral filters:

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right]$$
$$\times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$



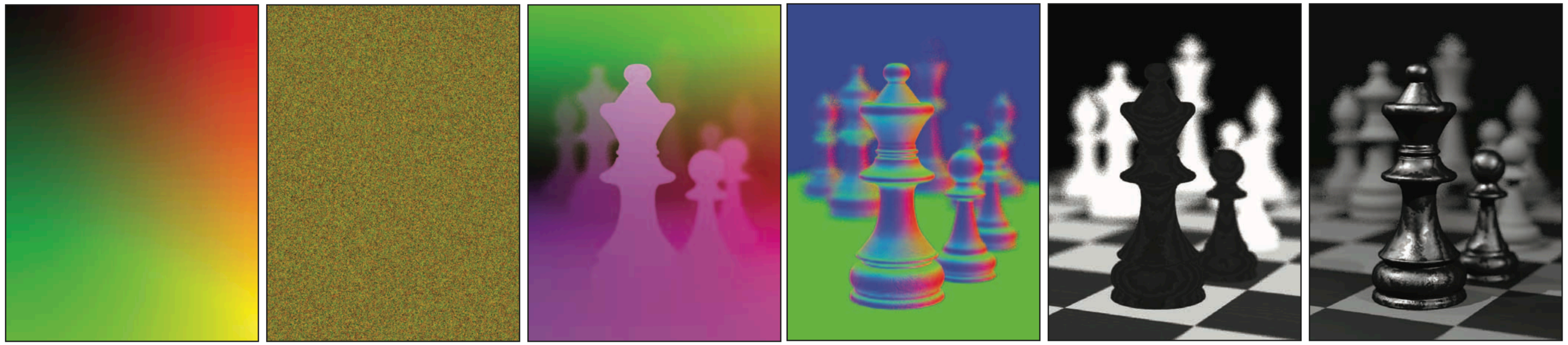
(a) Screen position **(b)** Random parameters **(c)** World space coords. **(d)** Surface normals **(e)** Texture value **(f)** Sample color

Filter weights

For cross Bilateral filters:

Pixel screen coordinates

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$



(a) Screen position **(b)** Random parameters **(c)** World space coords. **(d)** Surface normals **(e)** Texture value **(f)** Sample color

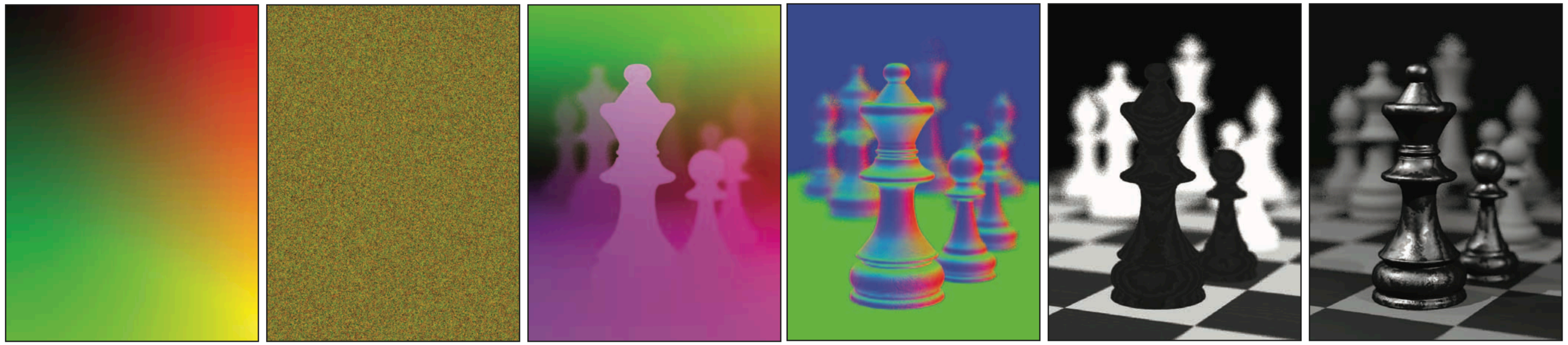
Filter weights

For cross Bilateral filters:

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$

Pixel screen coordinates

Mean sample color value



(a) Screen position **(b)** Random parameters **(c)** World space coords. **(d)** Surface normals **(e)** Texture value **(f)** Sample color

Filter weights

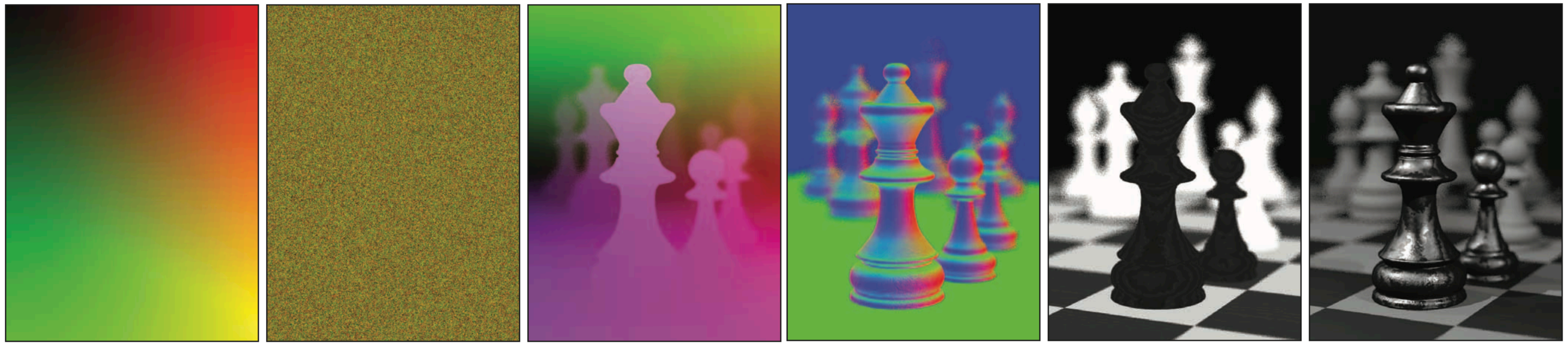
For cross Bilateral filters:

$$d_{i,j} = \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right],$$

Pixel screen coordinates

Mean sample color value

Scene features



(a) Screen position **(b)** Random parameters **(c)** World space coords. **(d)** Surface normals **(e)** Texture value **(f)** Sample color

Our result with a cross-bilateral filter (4 spp)

Overview on Convolutional Neural Networks (CNNs)

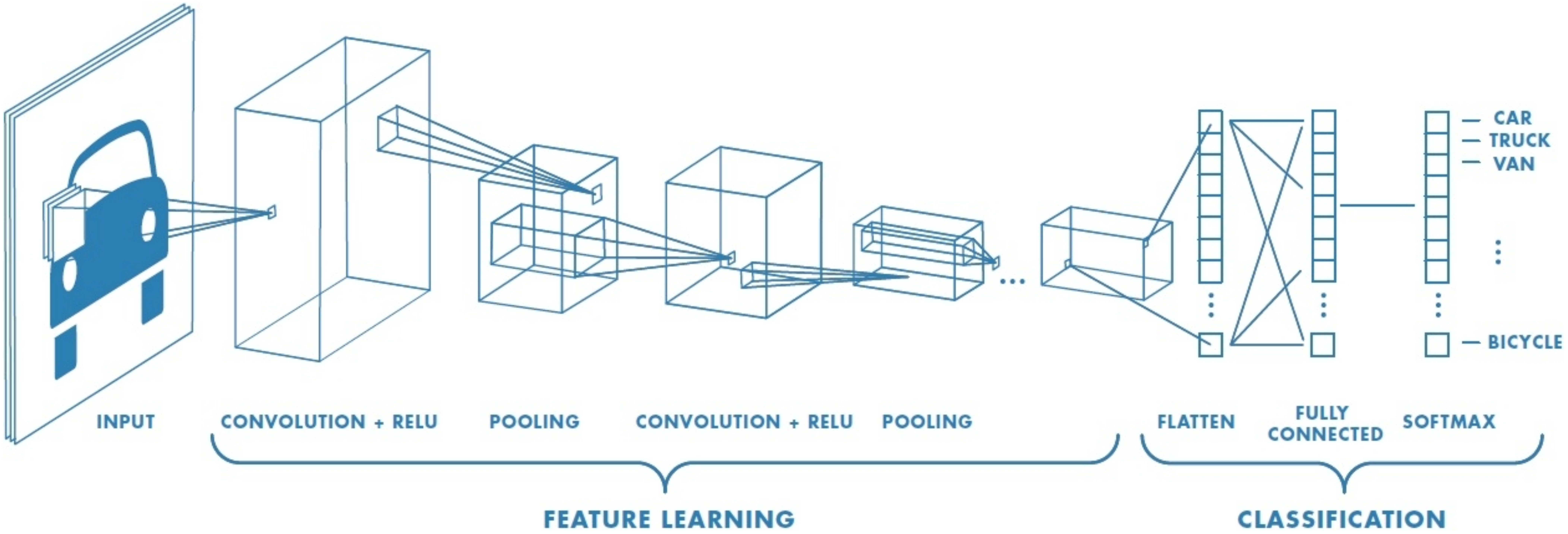
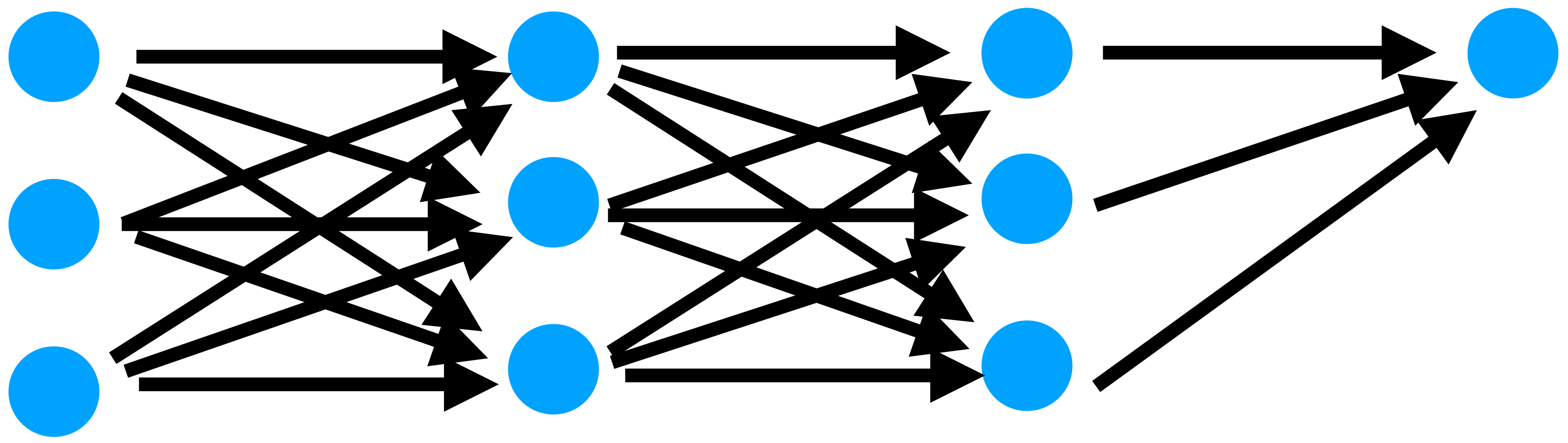
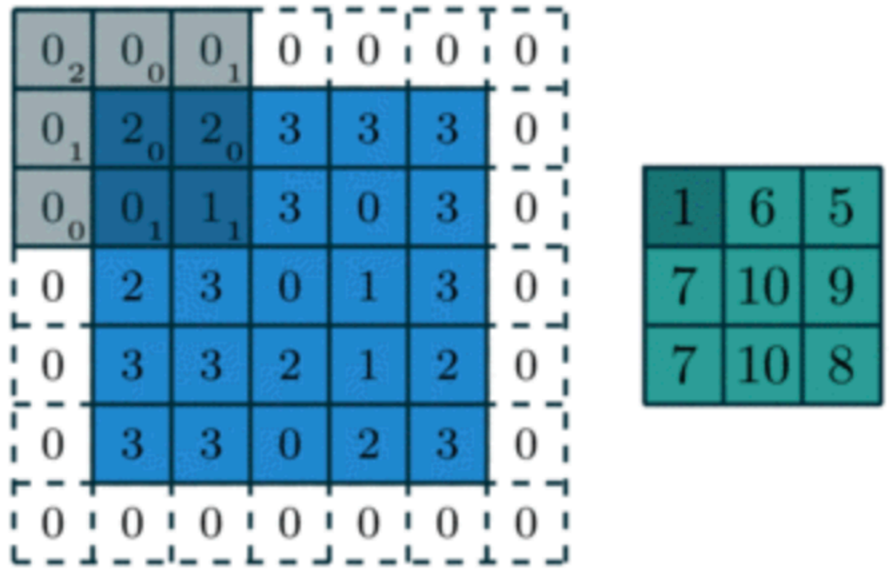
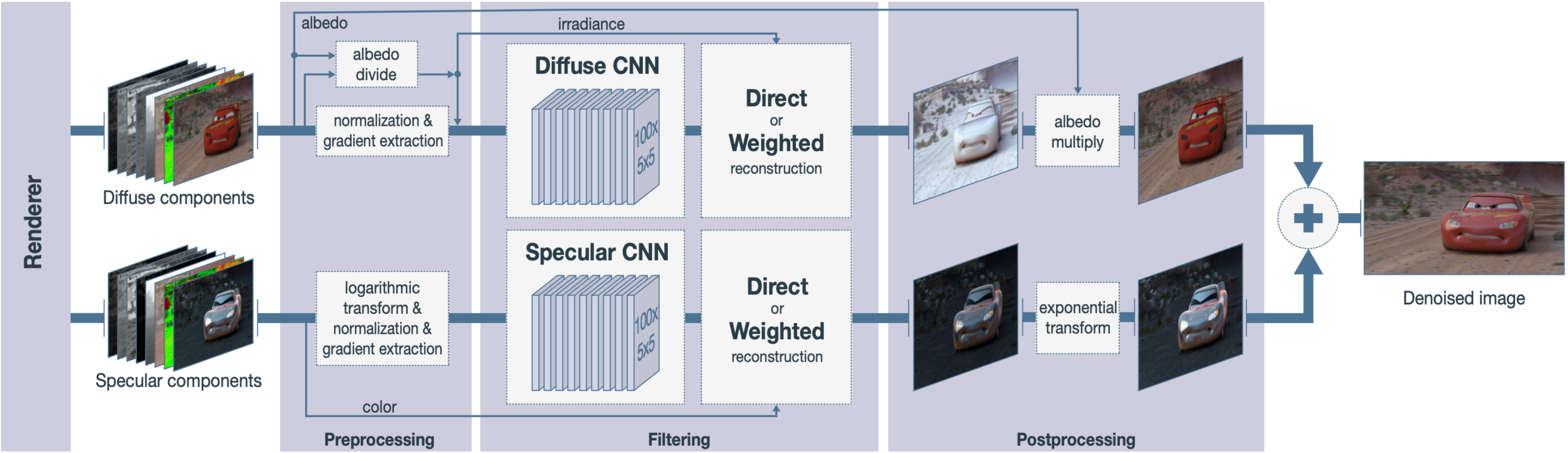


Image Courtesy: Mathworks (online tutorial)

Multi-layer Perceptron vs. CNNs

Multi-layer perceptron	CNNs
	
All nodes are fully connected in all layers	Weights are shared across layers
In theory, should be able to achieve good quality results in small number of layers.	Requires significant number of layers to capture all the features (e.g. Deep CNNs)
Number of weights to be learnt are very high	Relatively small number of weights required

Kernel-Predicting Networks for Denoising Monte-Carlo Renderings



Recurrent AutoEncoder for Interactive Reconstruction

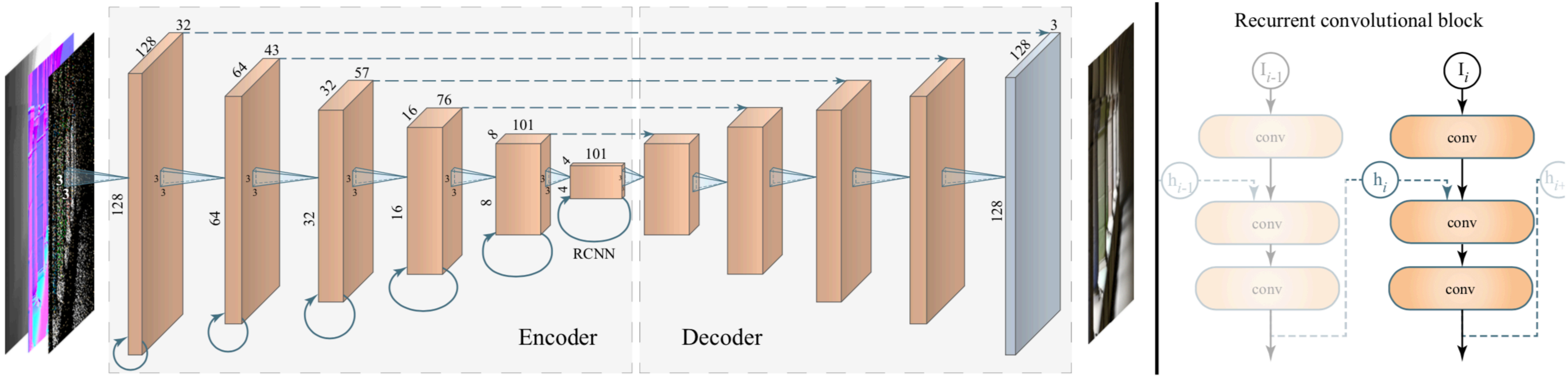
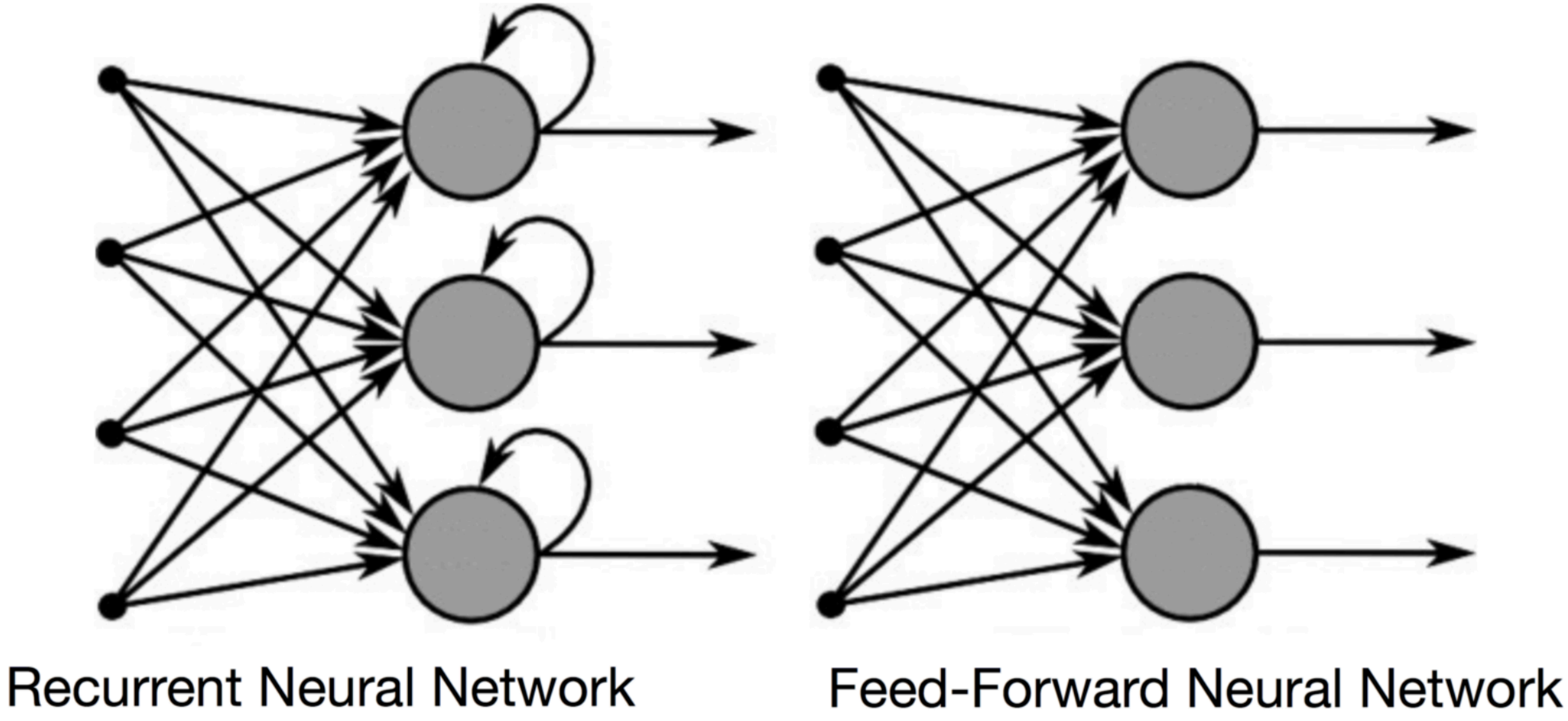


Fig. 2. Architecture of our recurrent autoencoder. The input is 7 scalar values per pixel (noisy RGB, normal vector, depth, roughness). Each encoder stage has a convolution and 2×2 max pooling. A decoder stage applies a 2×2 nearest neighbor upsampling, concatenates the per-pixel feature maps from a skip connection (the spatial resolutions agree), and applies two sets of convolution and pooling. All convolutions have a 3×3-pixel spatial support. On the right we visualize the internal structure of the recurrent RCNN connections. I is the new input and h refers to the hidden, recurrent state that persists between animation frames.

Recurrent Neural Networks vs. Simple Feed-Forward NN

Source link



Loss Functions

Spatial Loss to emphasize more the dark regions

$$L_s = \frac{1}{N}\sum_{i}^{N}|P_i - T_i|$$

Temporal loss

$$L_t = \frac{1}{N}\sum_{i}^{N}\left(\left|\frac{\partial P_i}{\partial t} - \frac{\partial T_i}{\partial t}\right|\right)$$

High frequency error norm loss for stable edges

$$L_g = \frac{1}{N}\sum_{i}^{N}|\nabla P_i - \nabla T_i|$$

Final Loss is a weighted averaged of above losses

$$L = w_s L_s + w_g L_g + w_t L_t$$

Pixel-space
Kernel Predicting
Denoising

#Learnable Parameters?

How to compute "learnable" parameters?

Pixel-space
Kernel Predicting
Denoising

#Learnable
Parameters?

How to compute "learnable" parameters?

Sample-based
MC Denoising

How to compute "learnable" parameters?

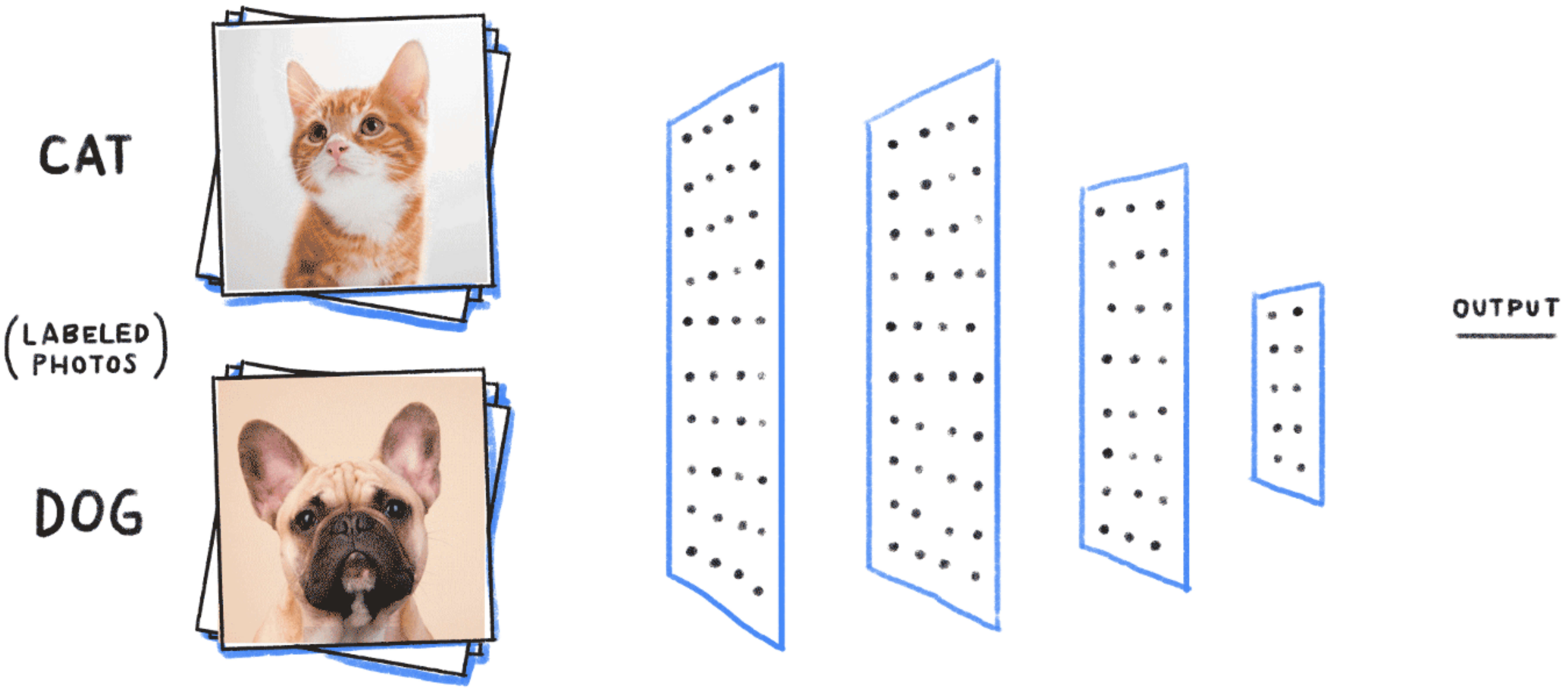


Image Source: Google

How to compute "learnable" parameters?

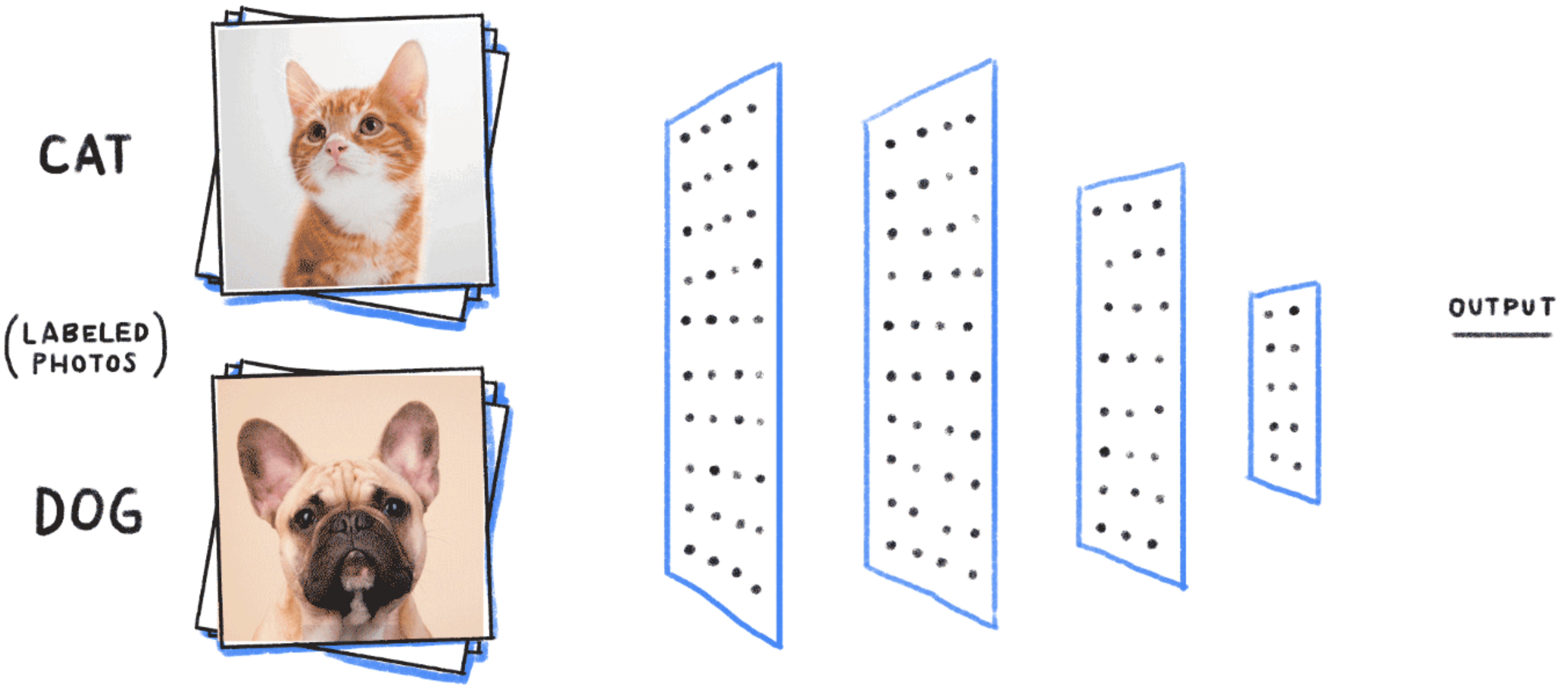


Image Source: Google

How to compute "learnable" parameters?

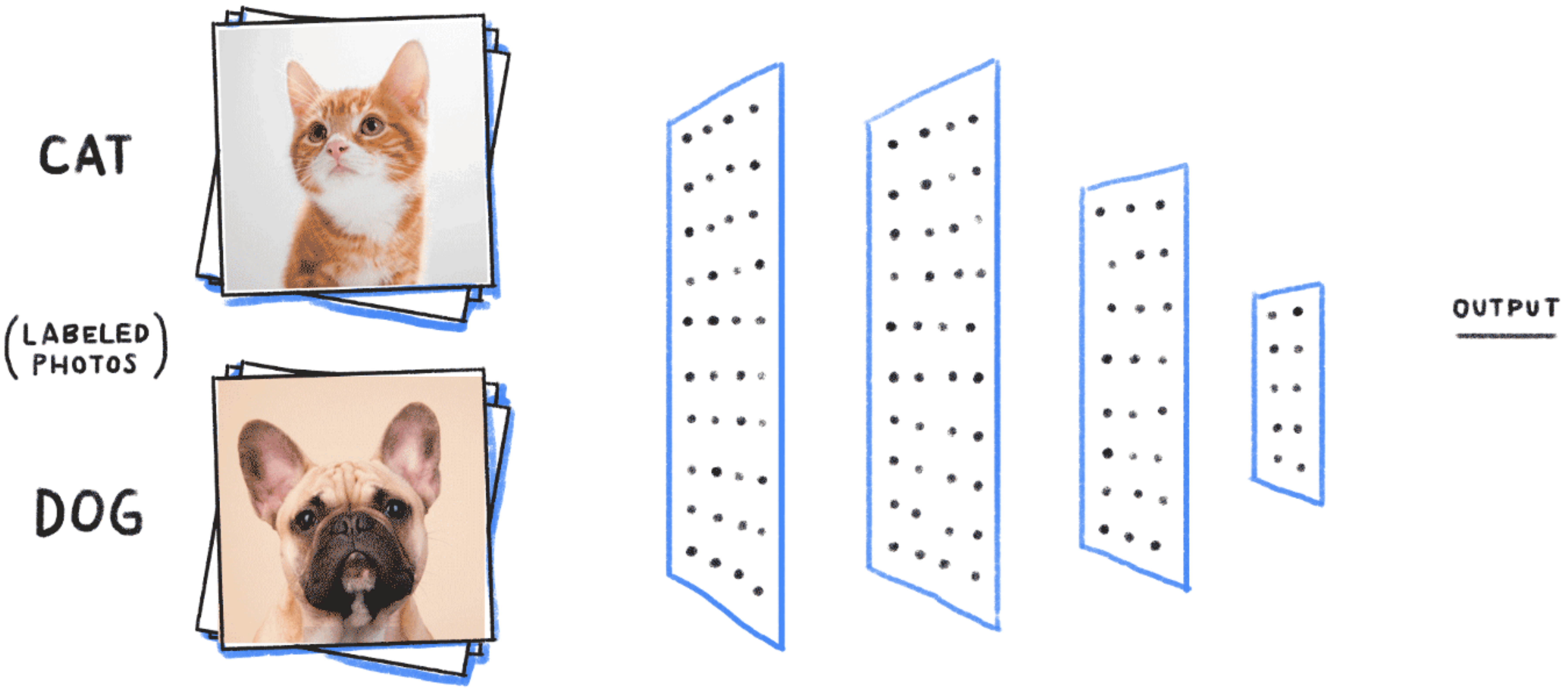


Image Source: Google

How to compute "learnable" parameters?

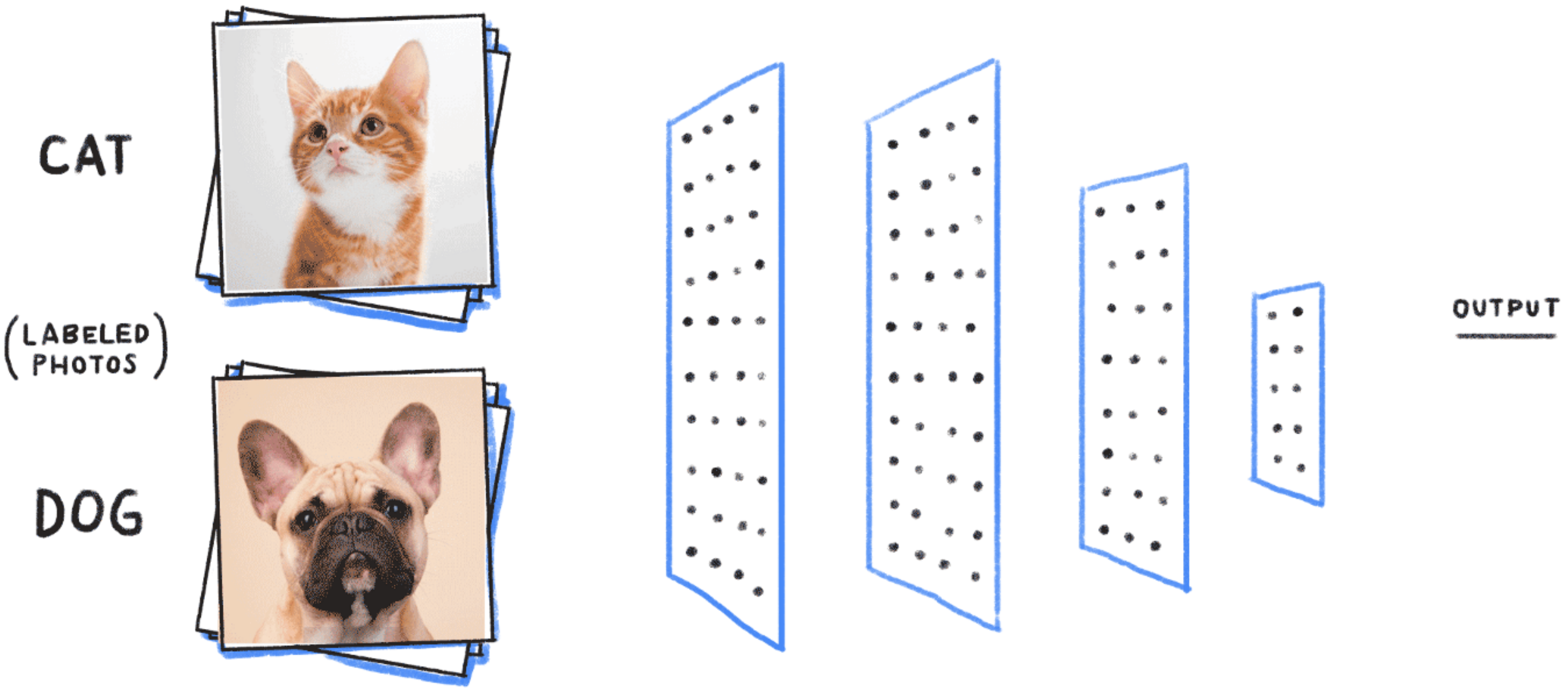


Image Source: Google

Feed-Forward Neural Network

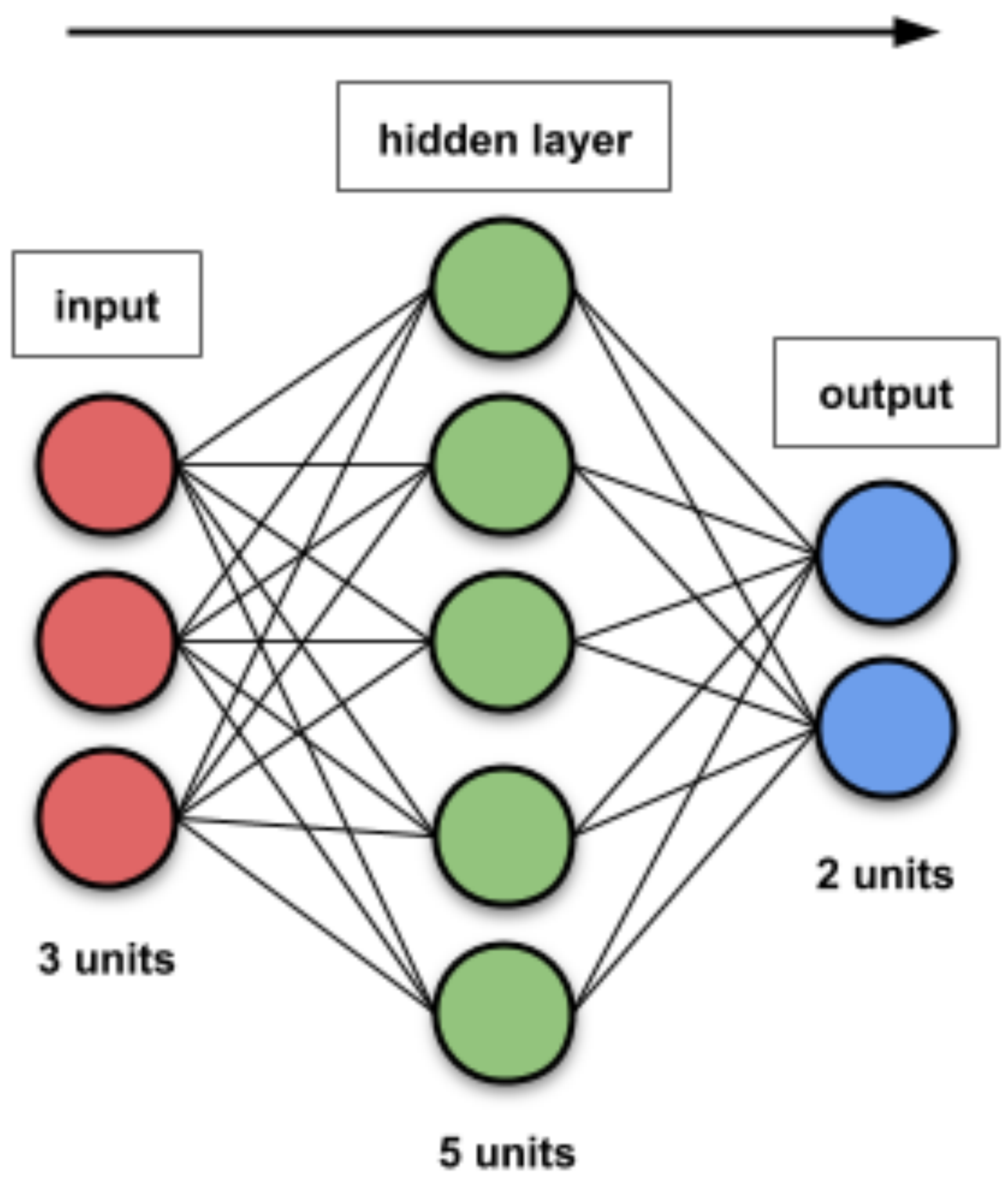
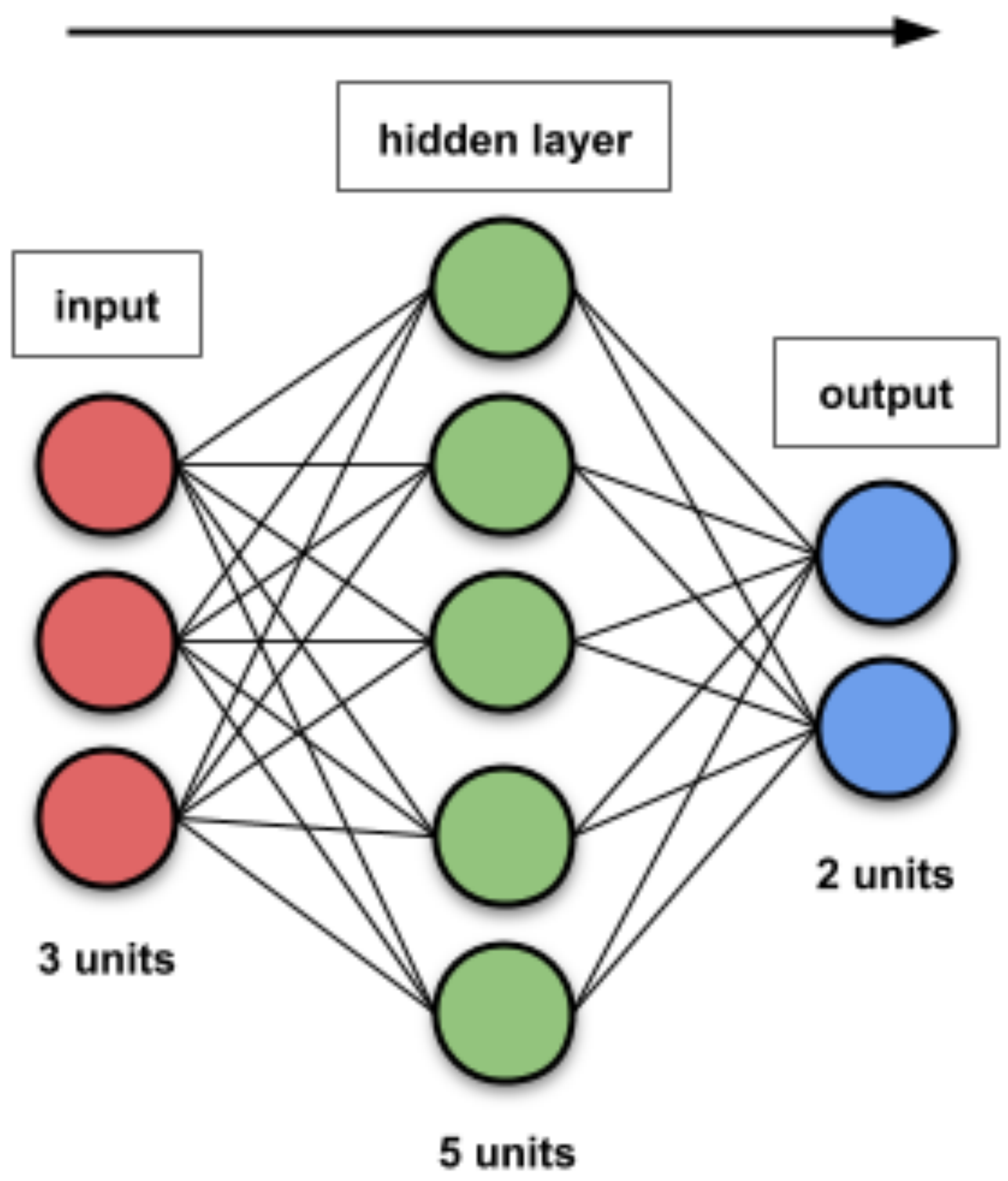


Image Source: towards-data-science

Feed-Forward Neural Network



$(3 \times 5) + (5 \times 2) + (5 + 2) = 17$ parameters

weights biases

<u>**Image Source: towards-data-science**</u>

Feed-Forward Neural Network

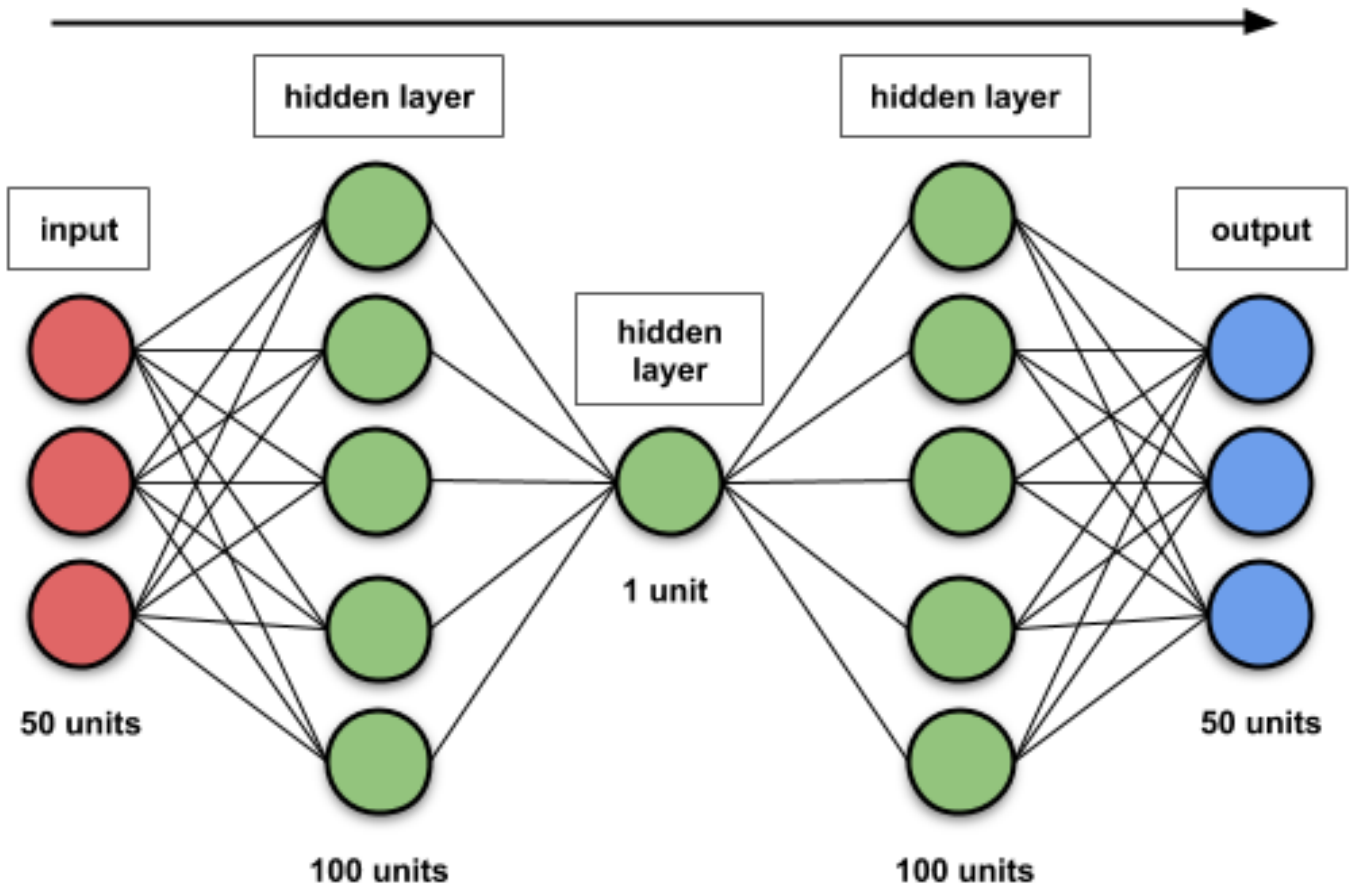


Image Source: towards-data-science

Pixel-space
Kernel Predicting
Denoising

#Learnable
Parameters?

Pixel-space
Kernel Predicting
Denoising

#Learnable
Parameters?

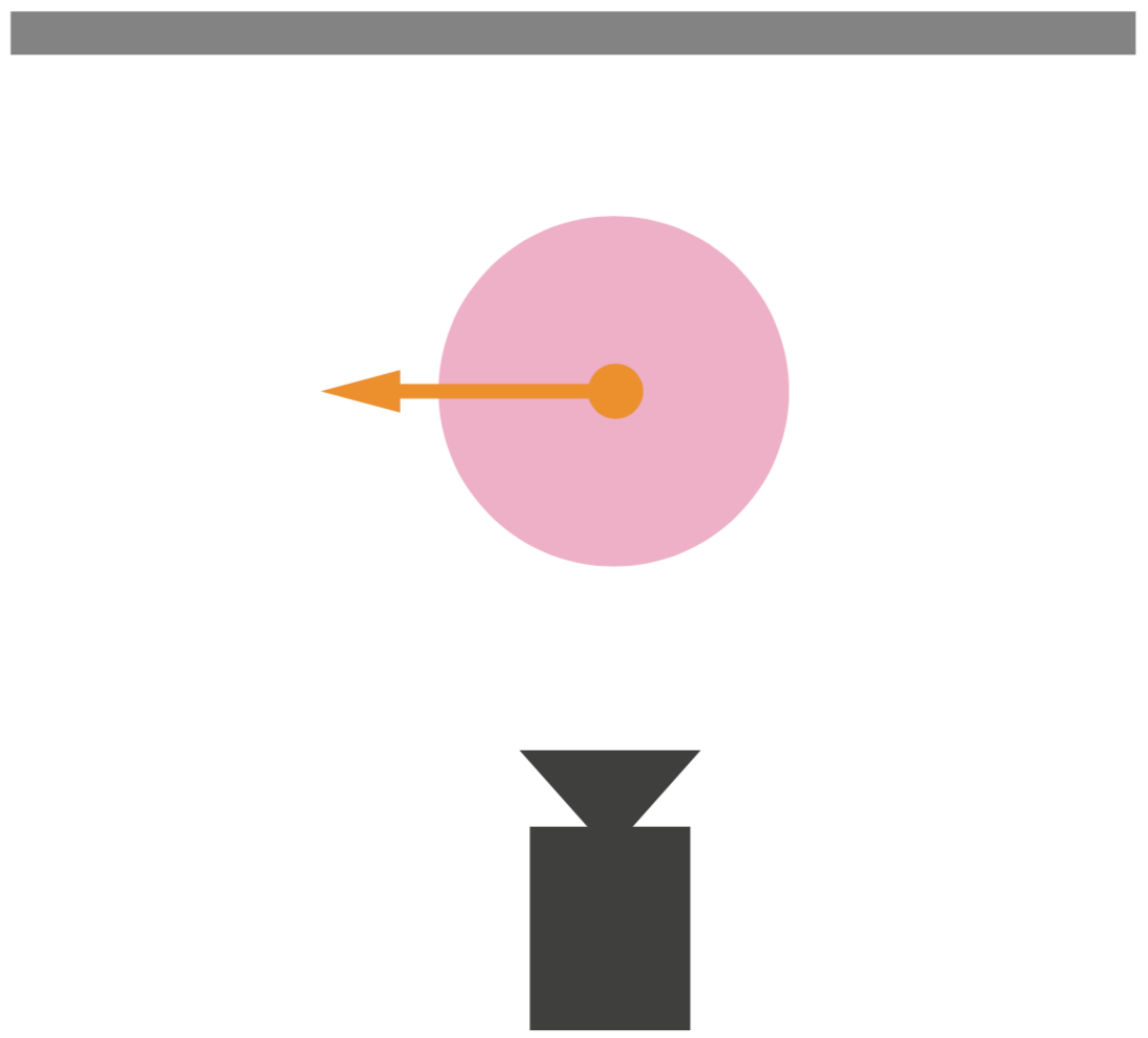
Sample-based
MC Denoising

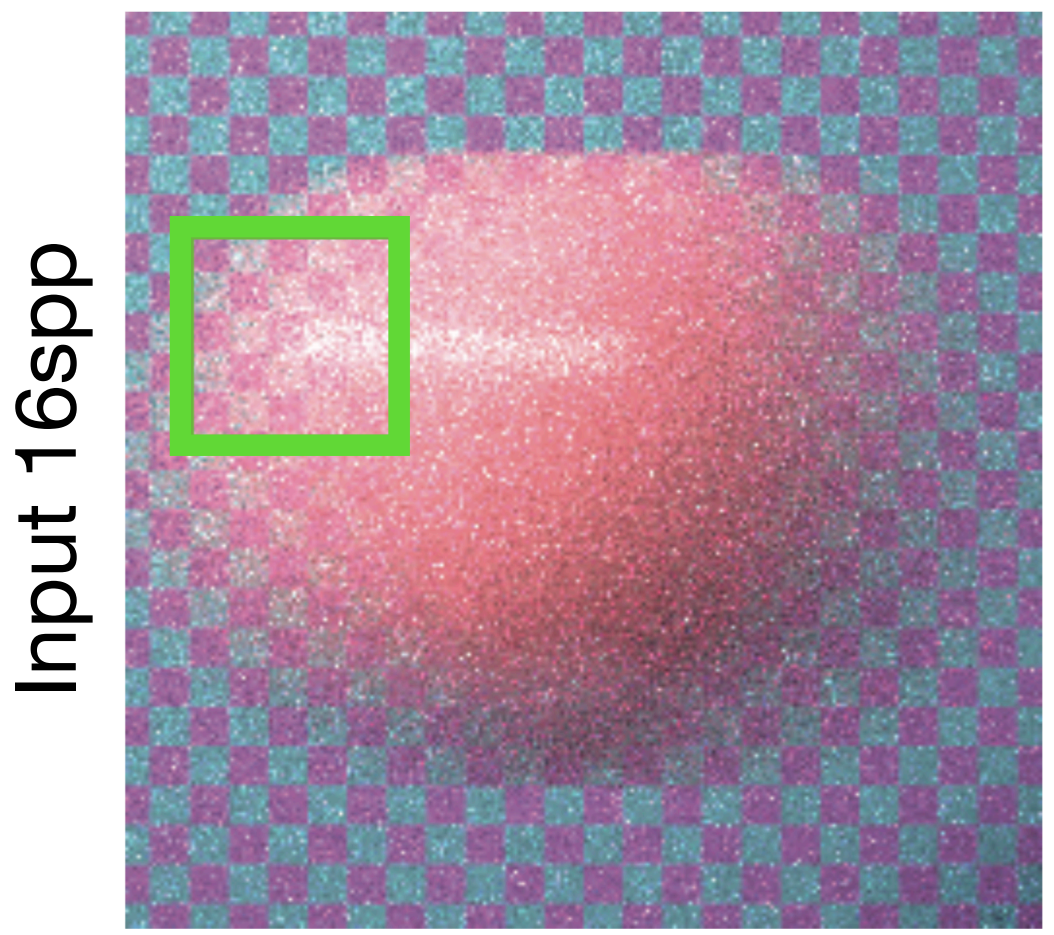
Sample-based Denoising Network

Michael Gharbi, Tzu-Mao Li, Miika Aittala, Jakko Lehtinen, Fredo Durand

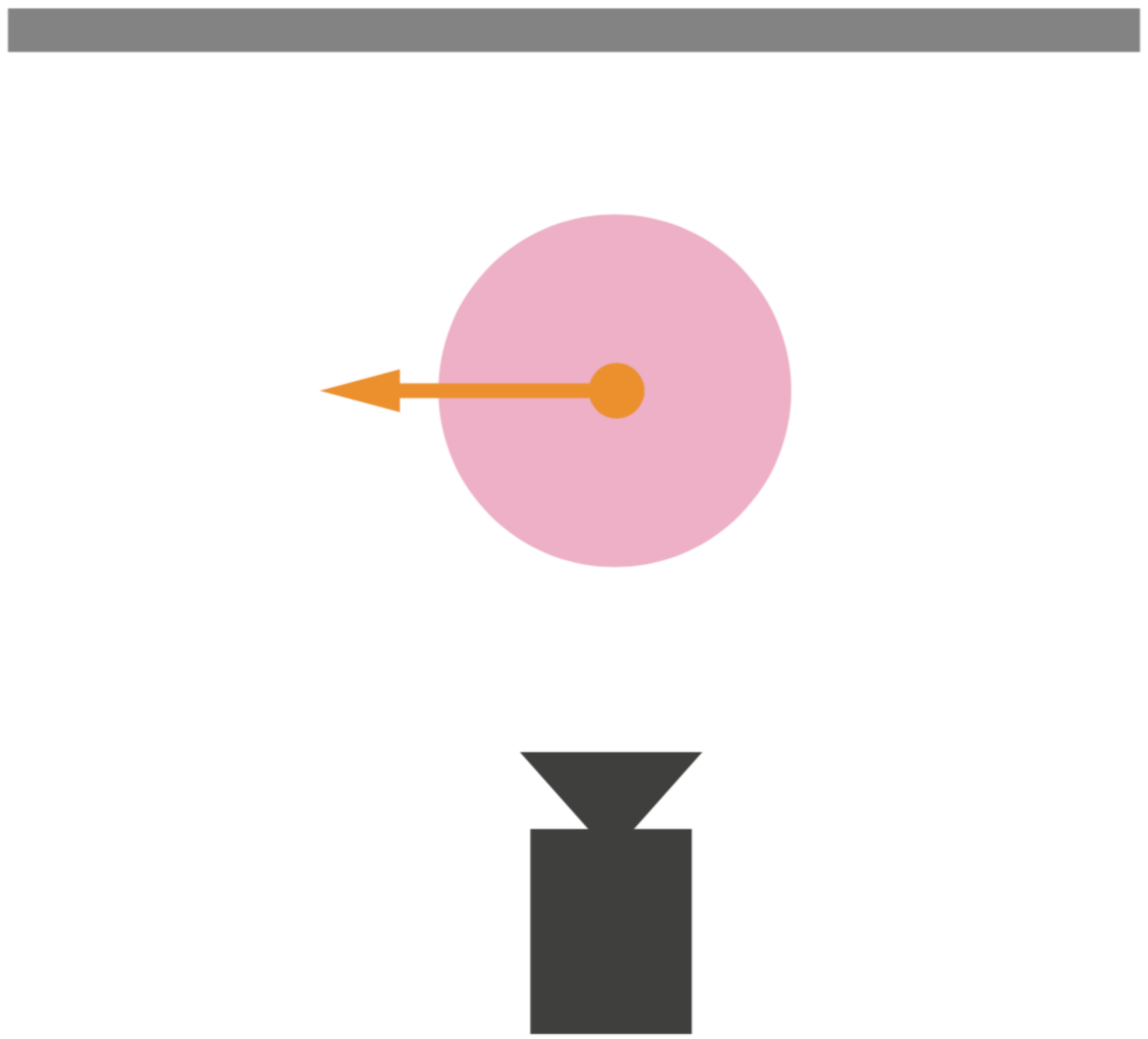
SIGGRAPH 2019

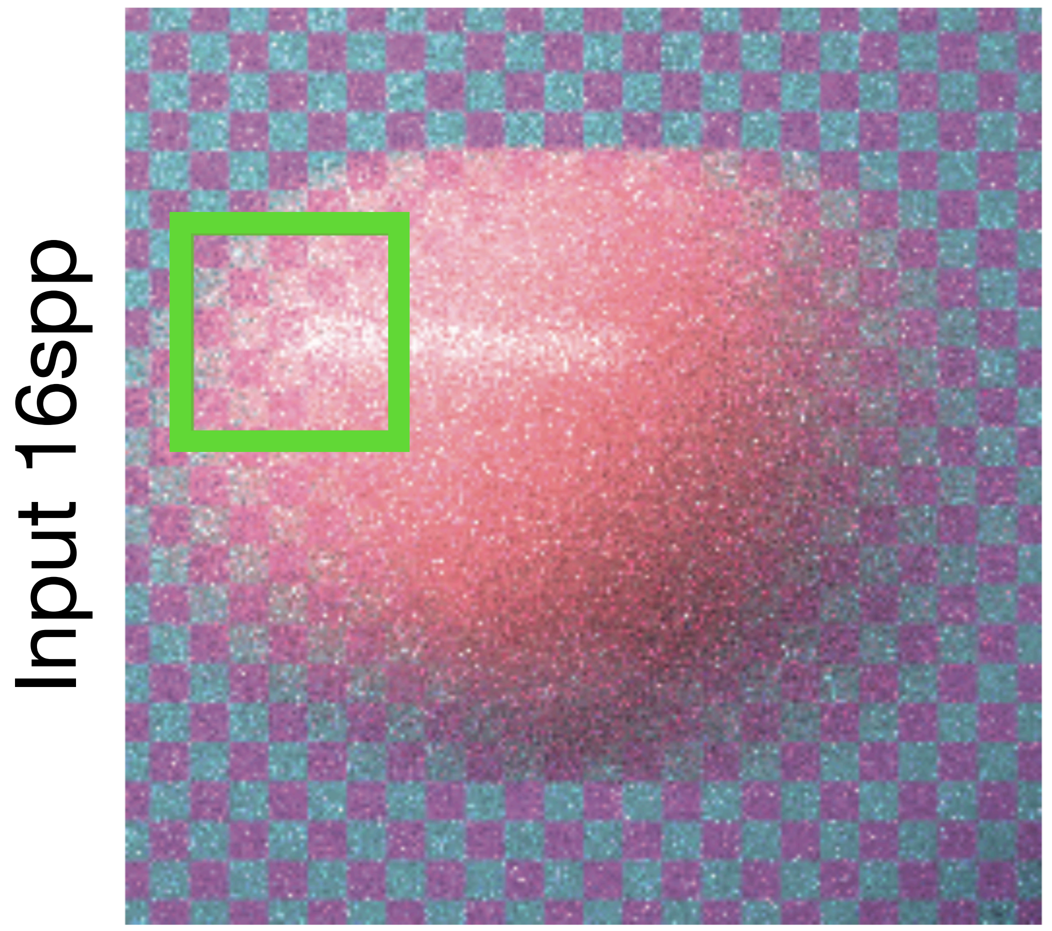
Multimodal distribution of sample features

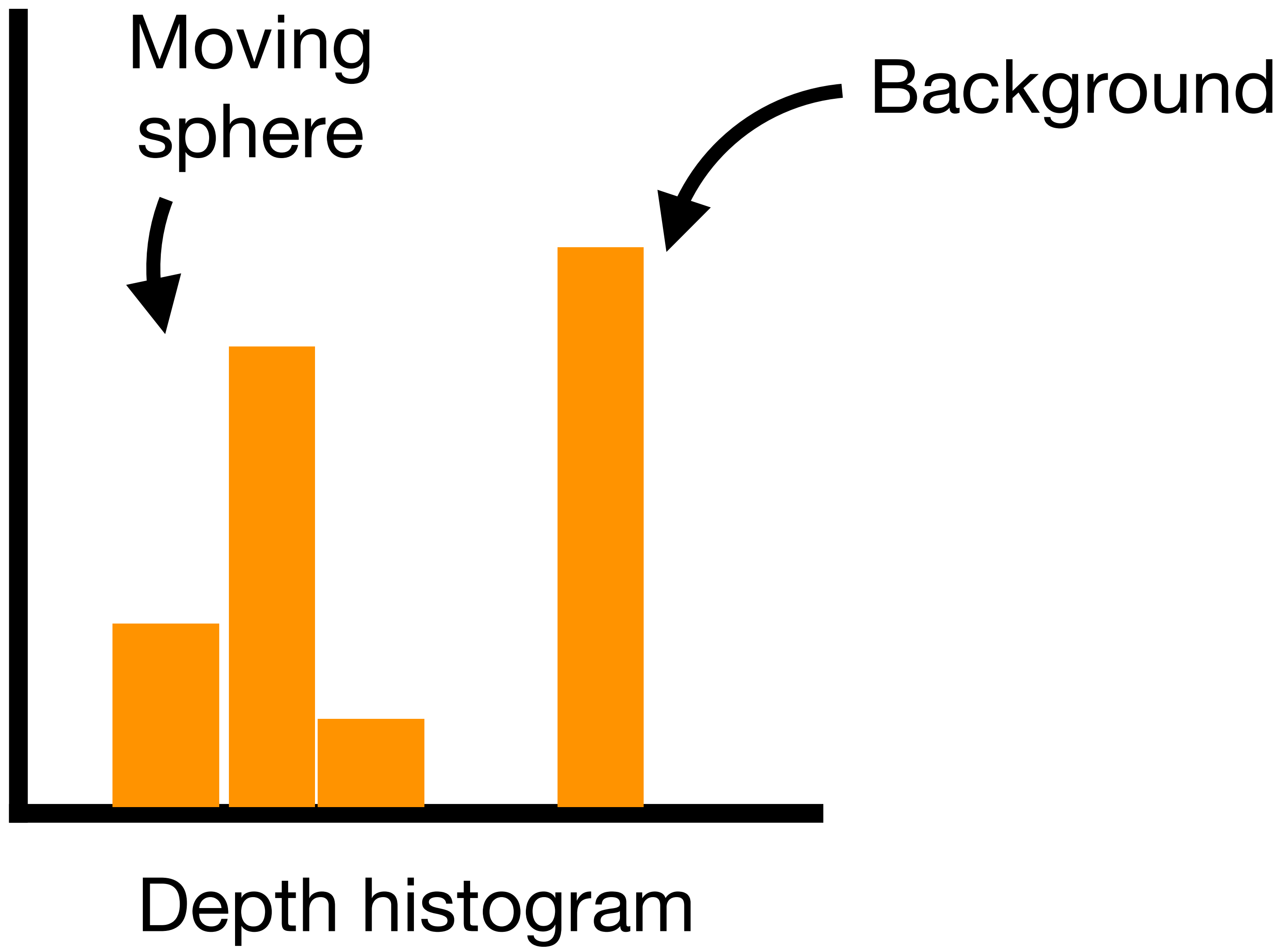




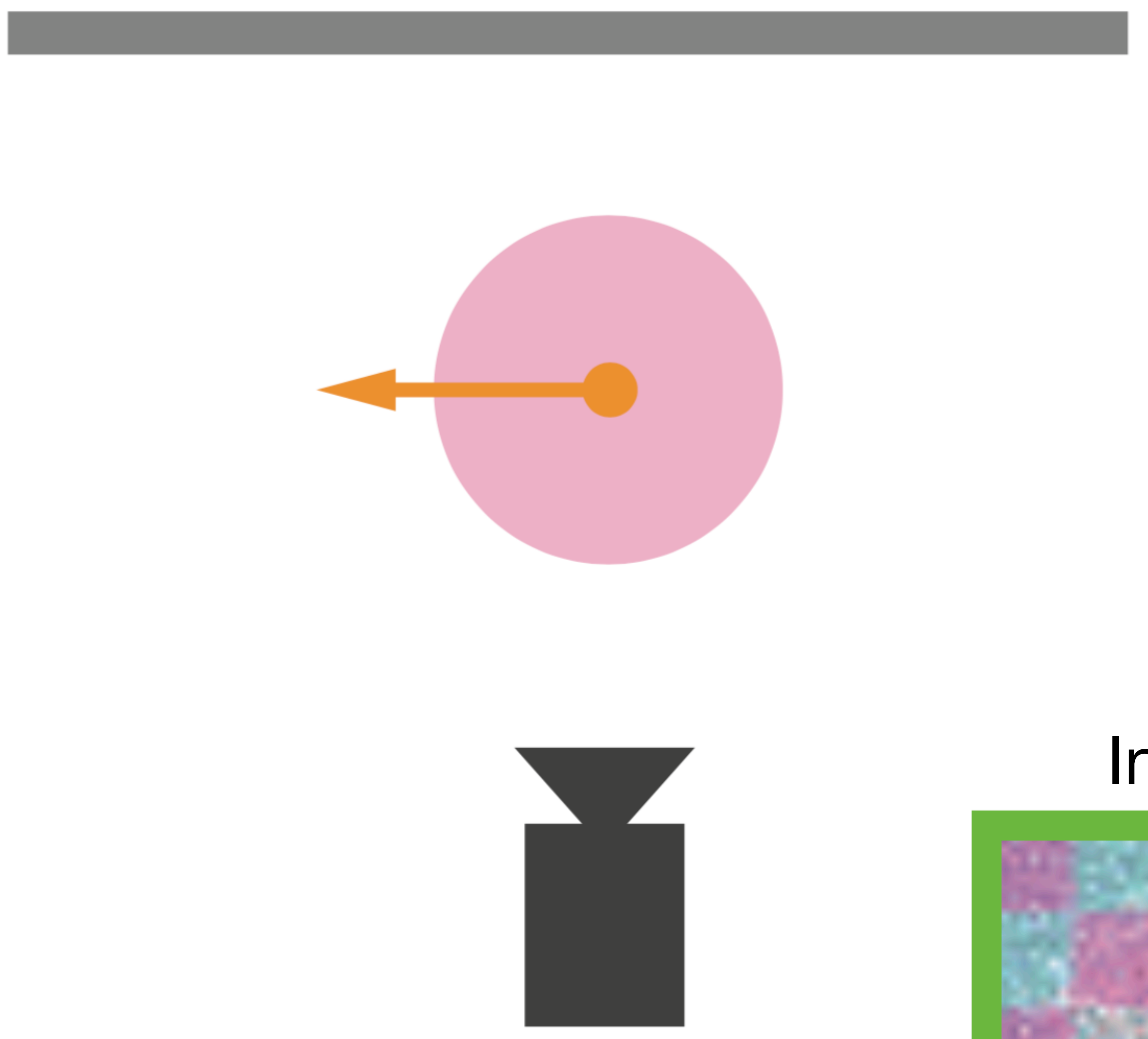
Multimodal distribution of sample features



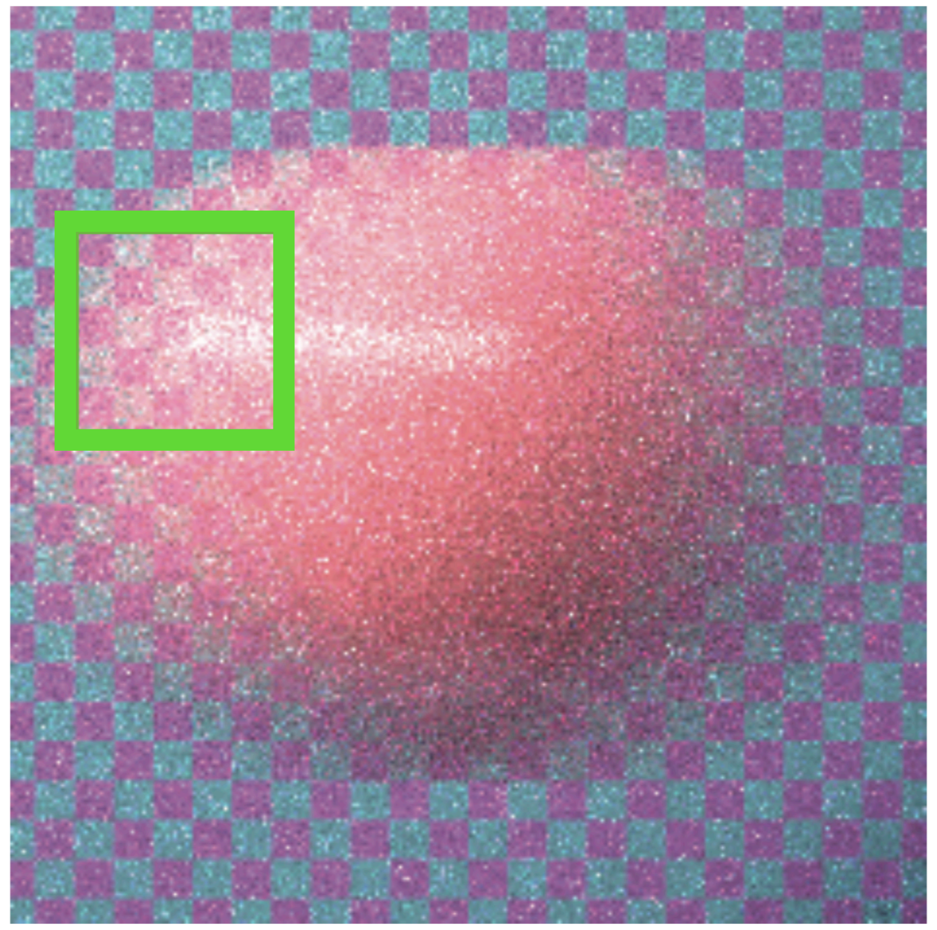


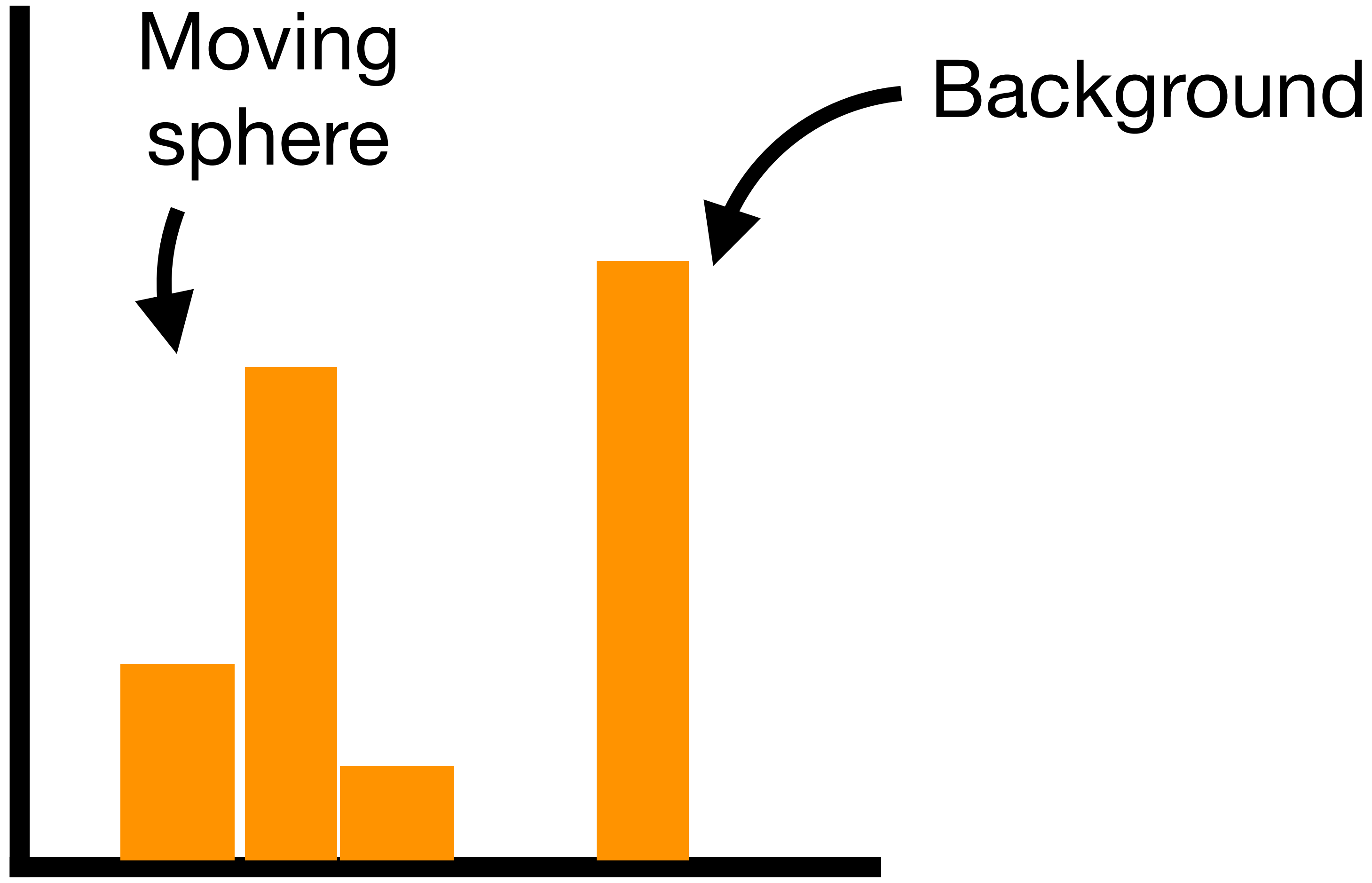


Multimodal distribution of sample features



Input 16spp

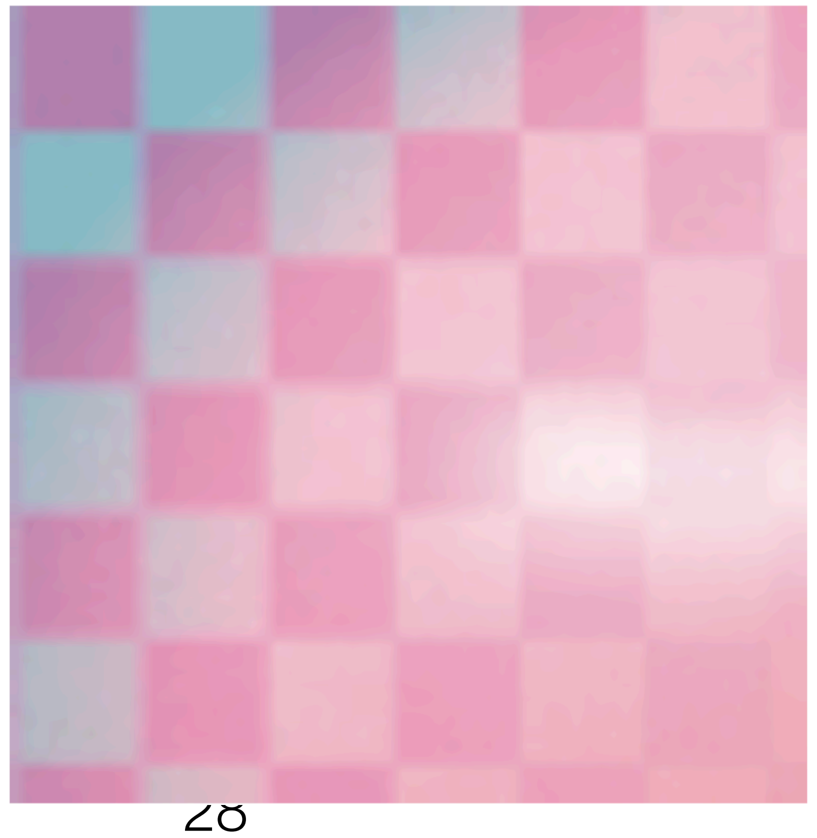




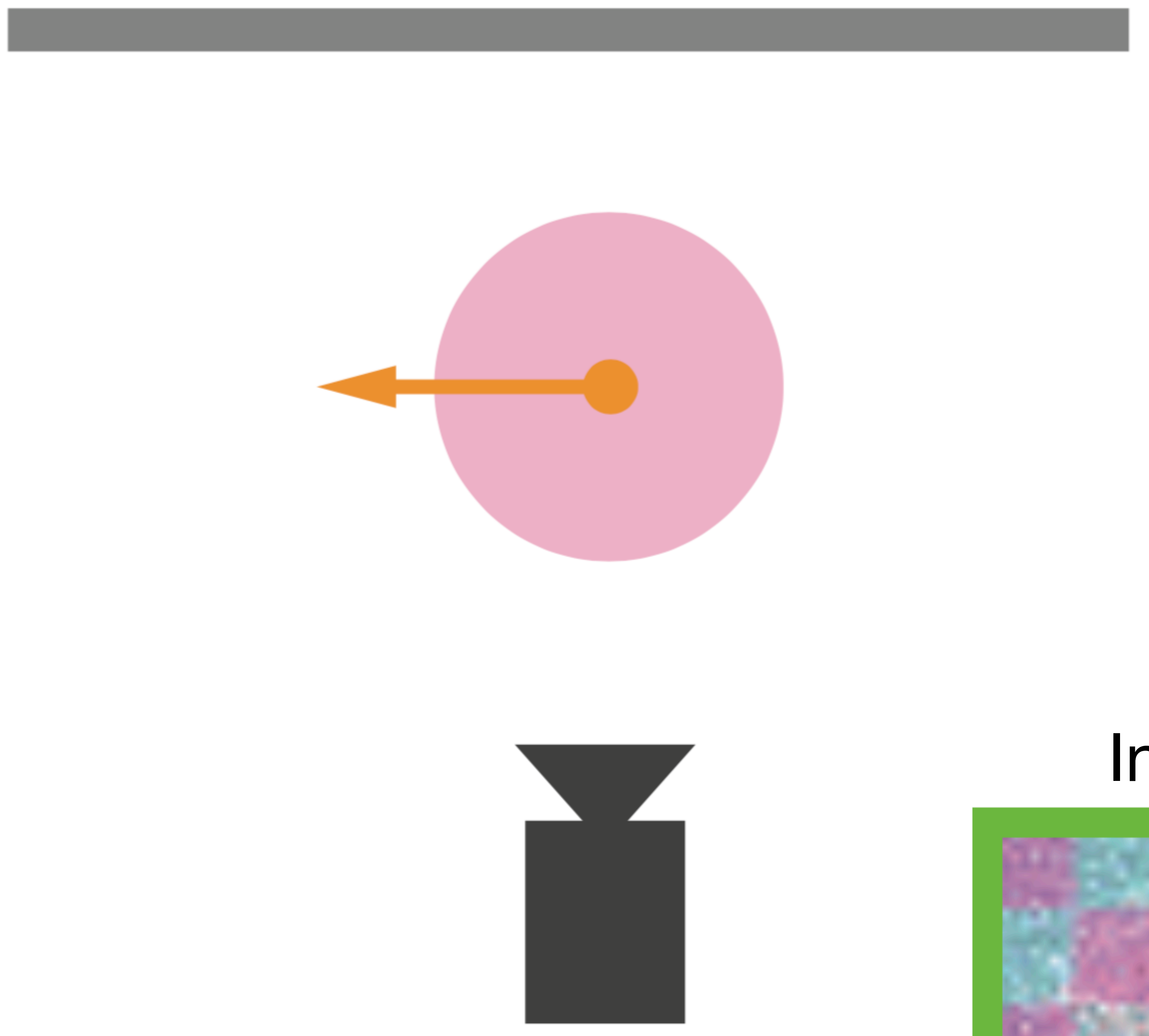
Depth histogram

Inset 16spp

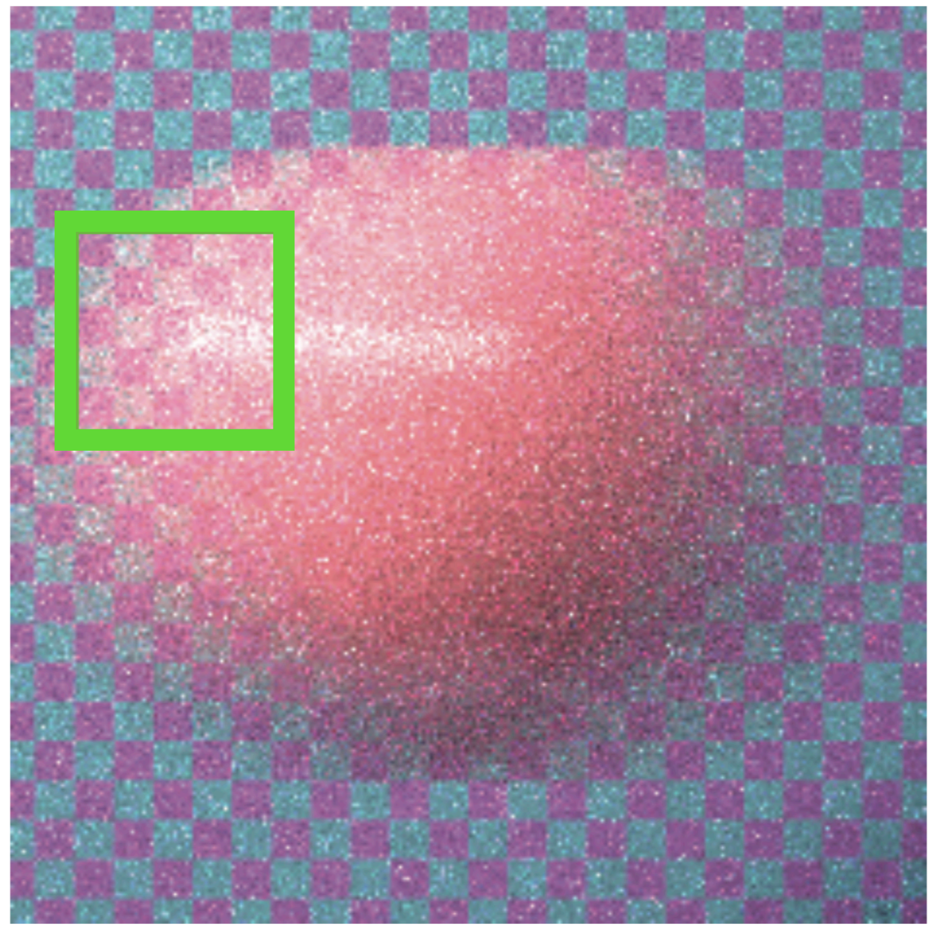
Reference

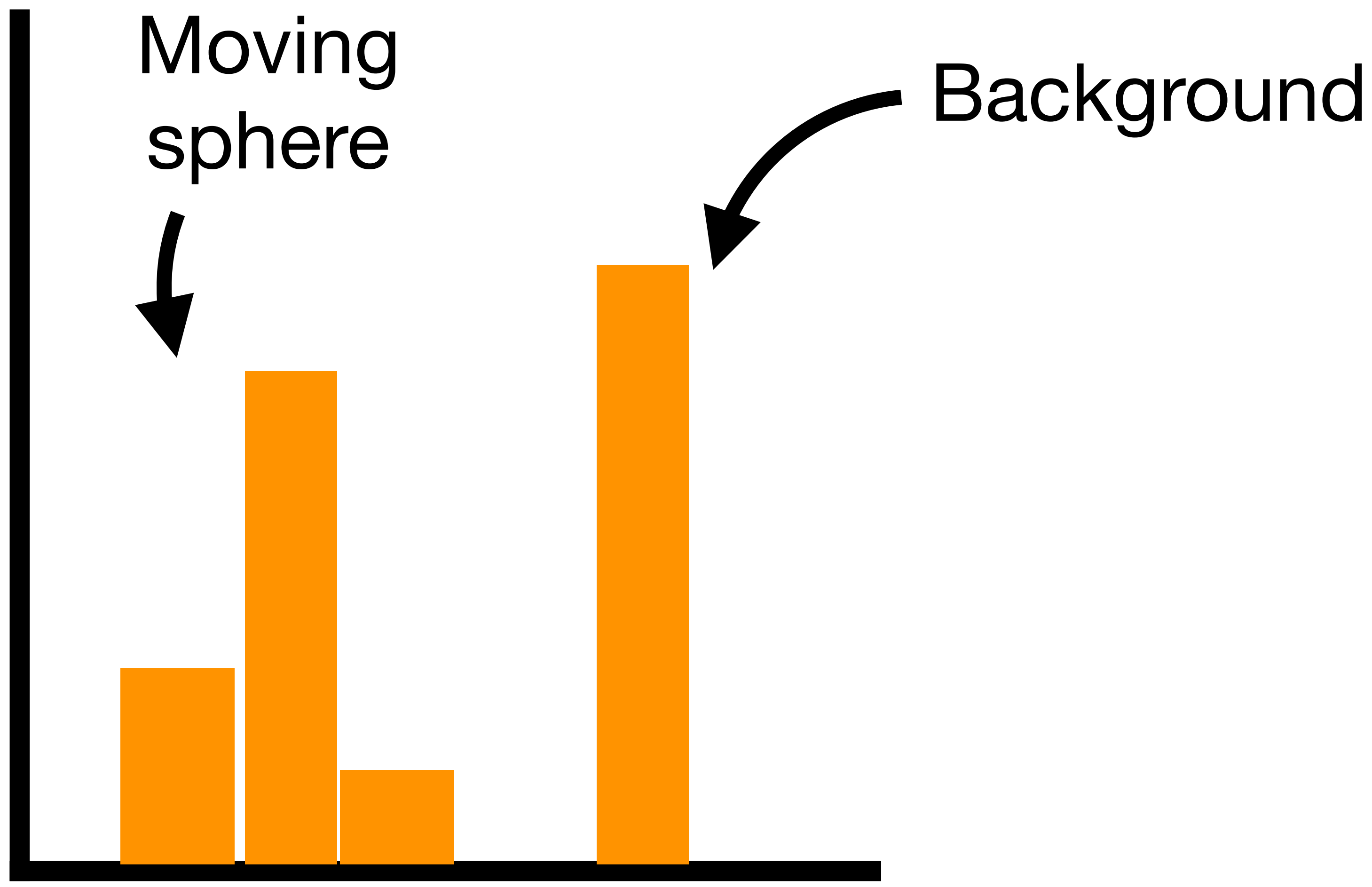


Multimodal distribution of sample features



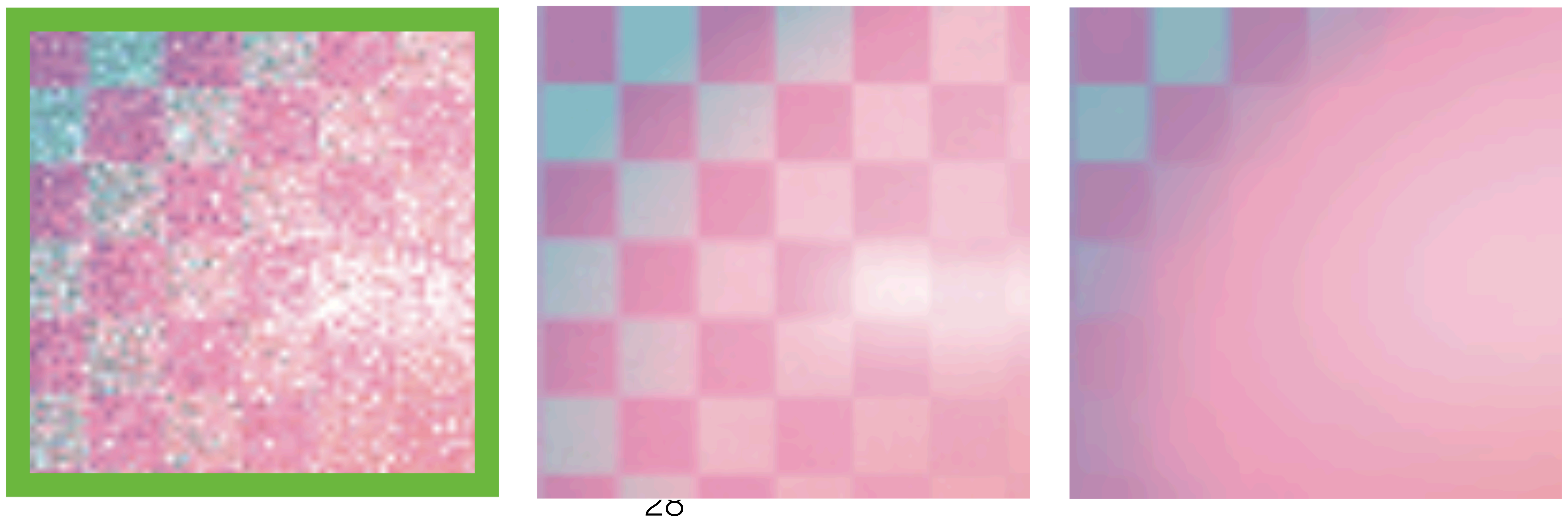
Input 16spp



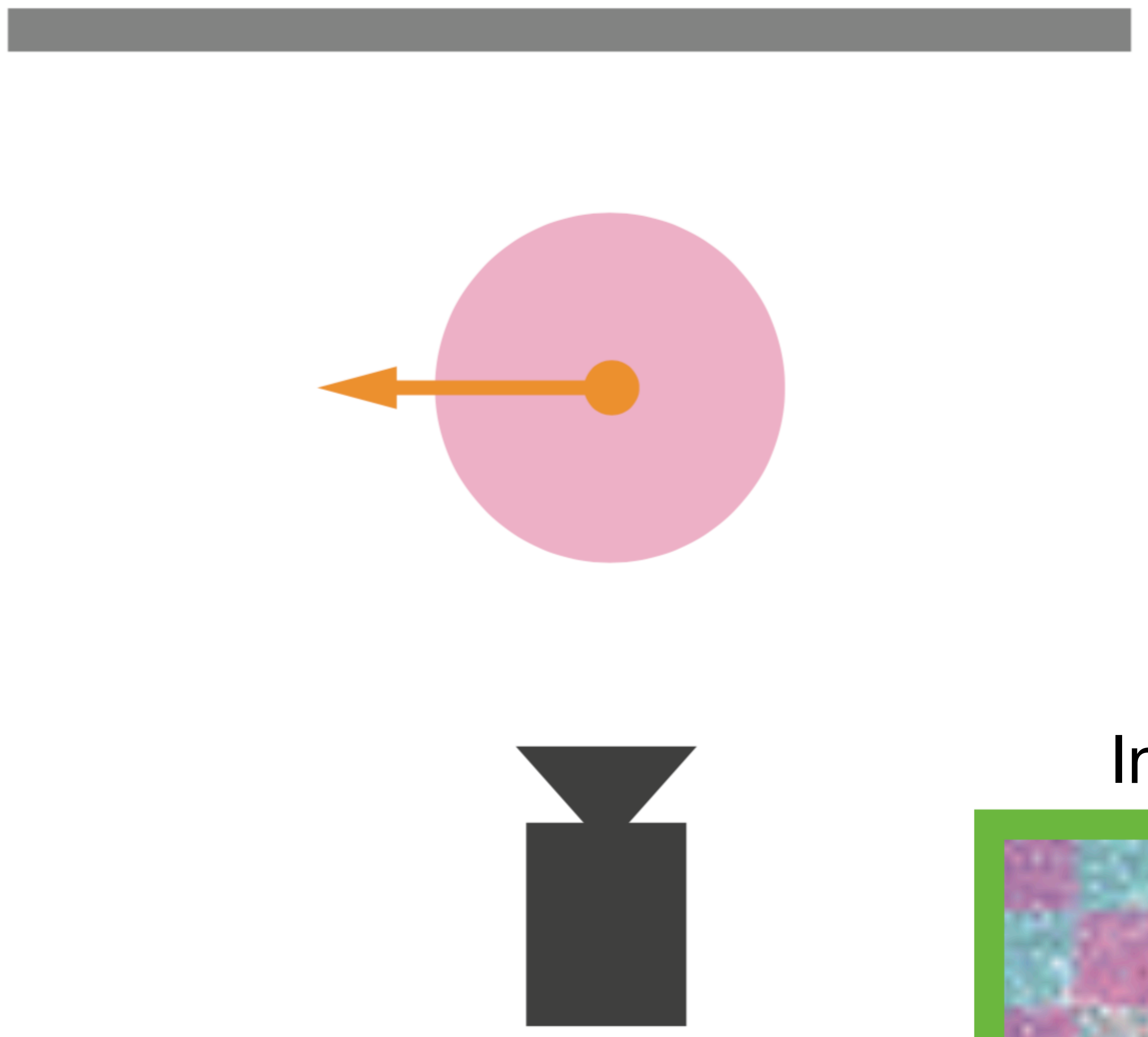


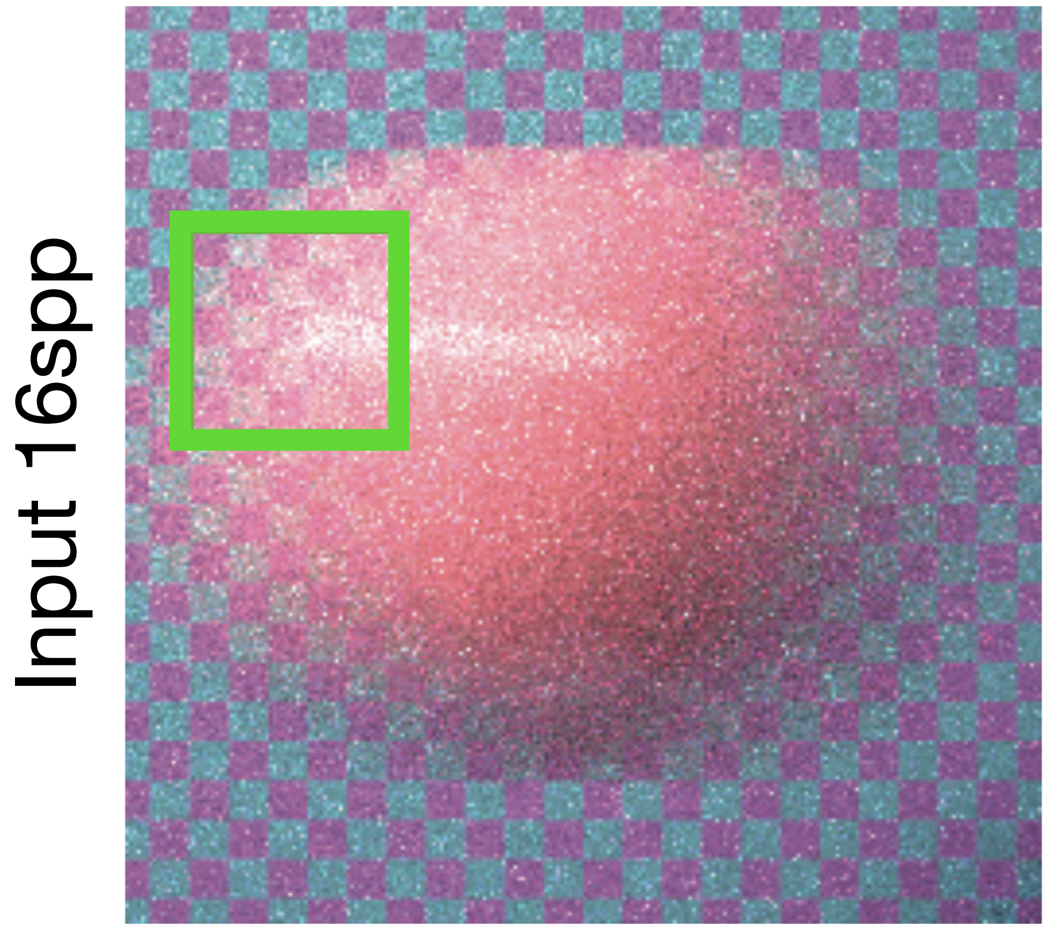
Depth histogram

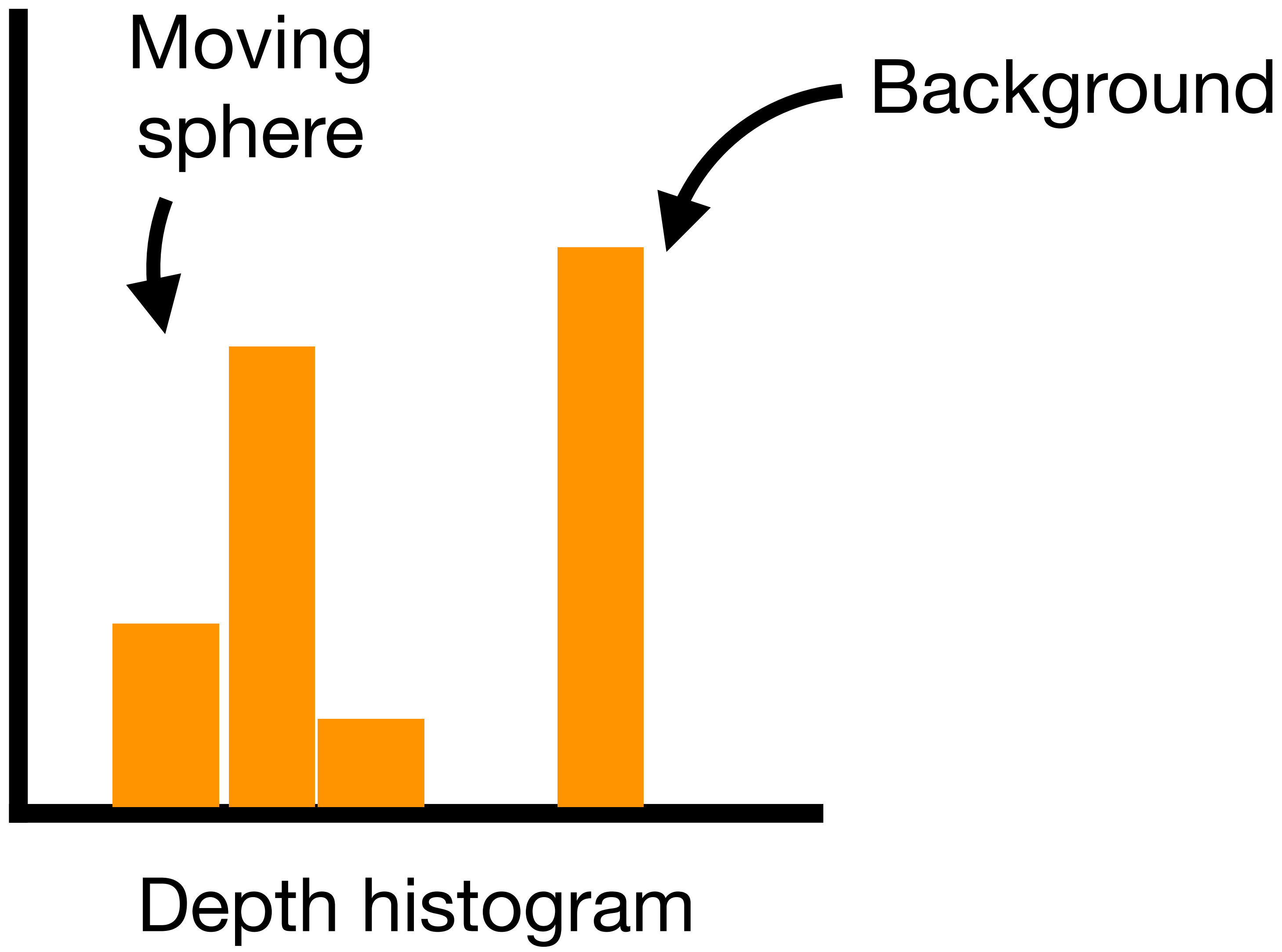
Inset 16spp | Reference | Sen [2012]

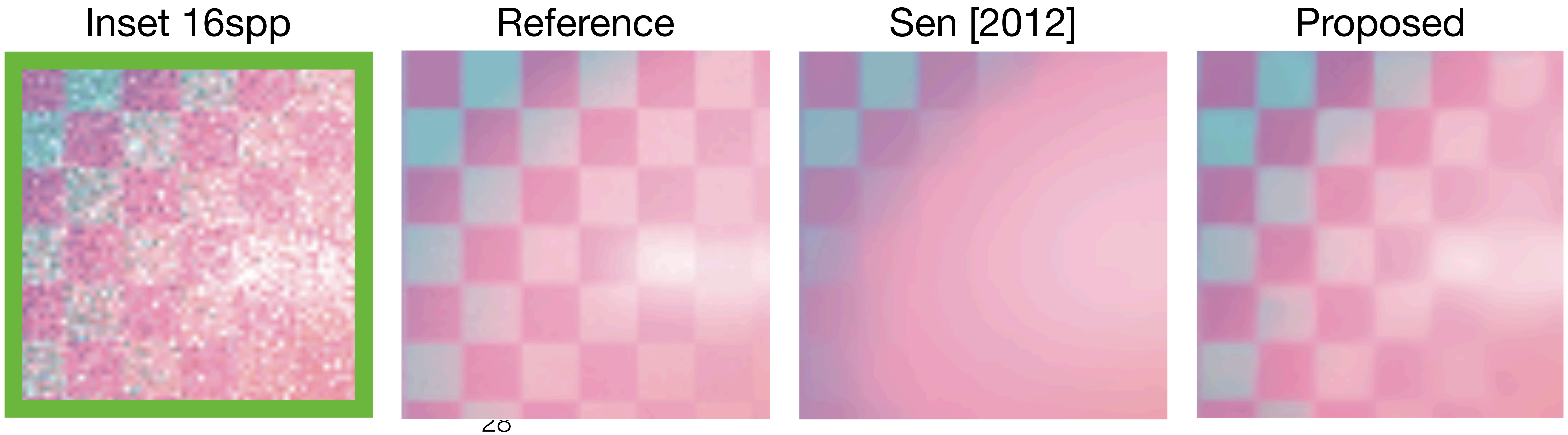


Multimodal distribution of sample features

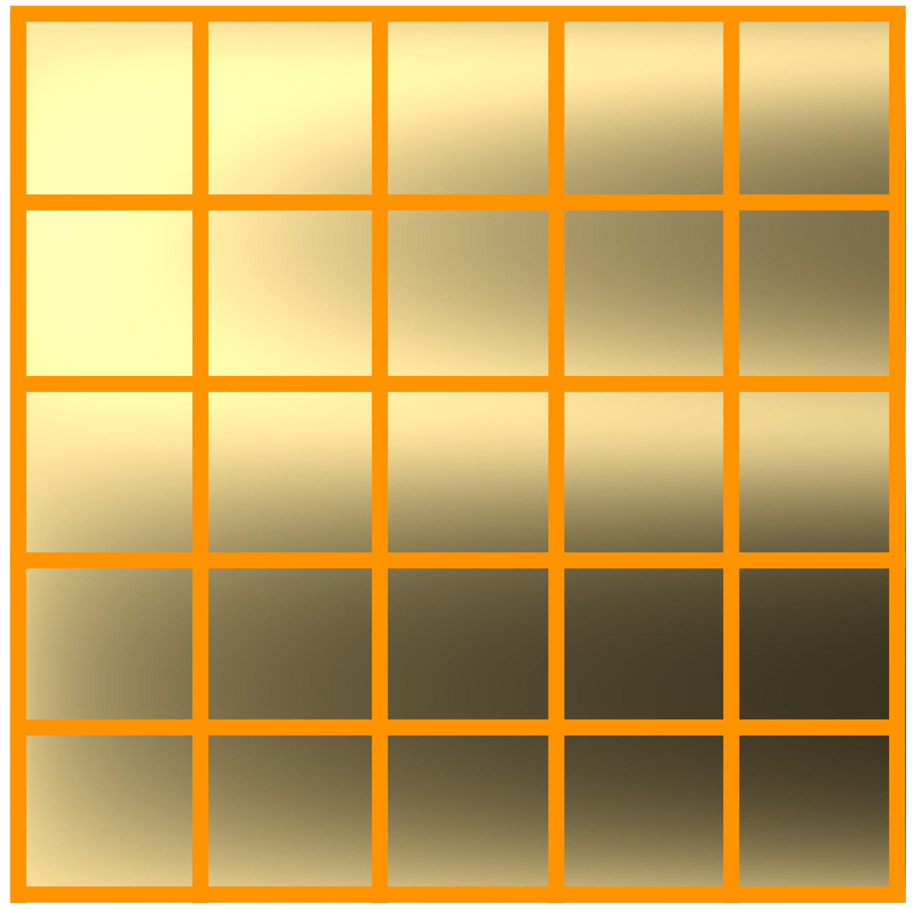




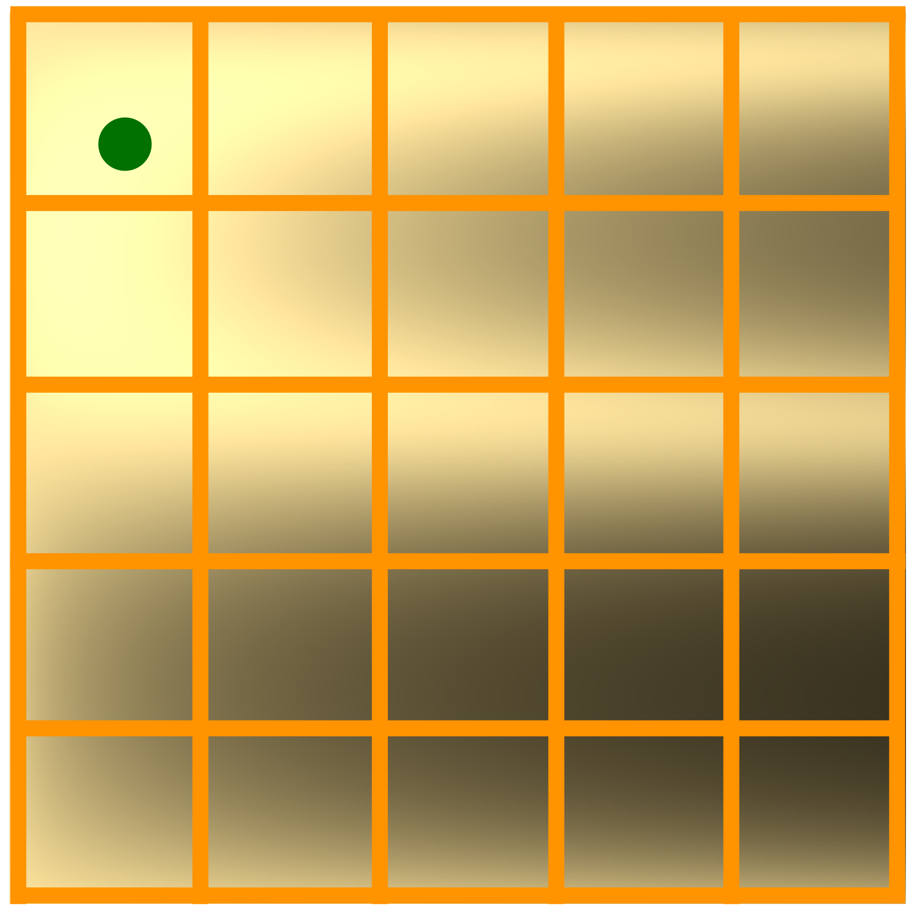




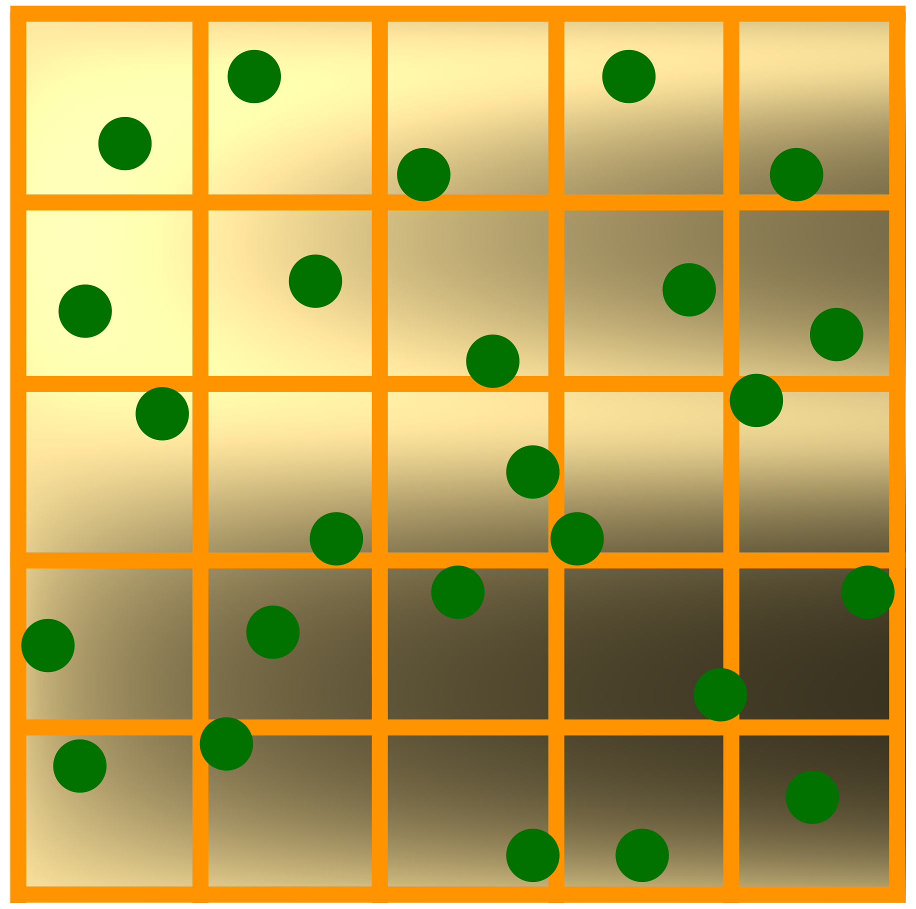
Reconstruction: Kernel Gather



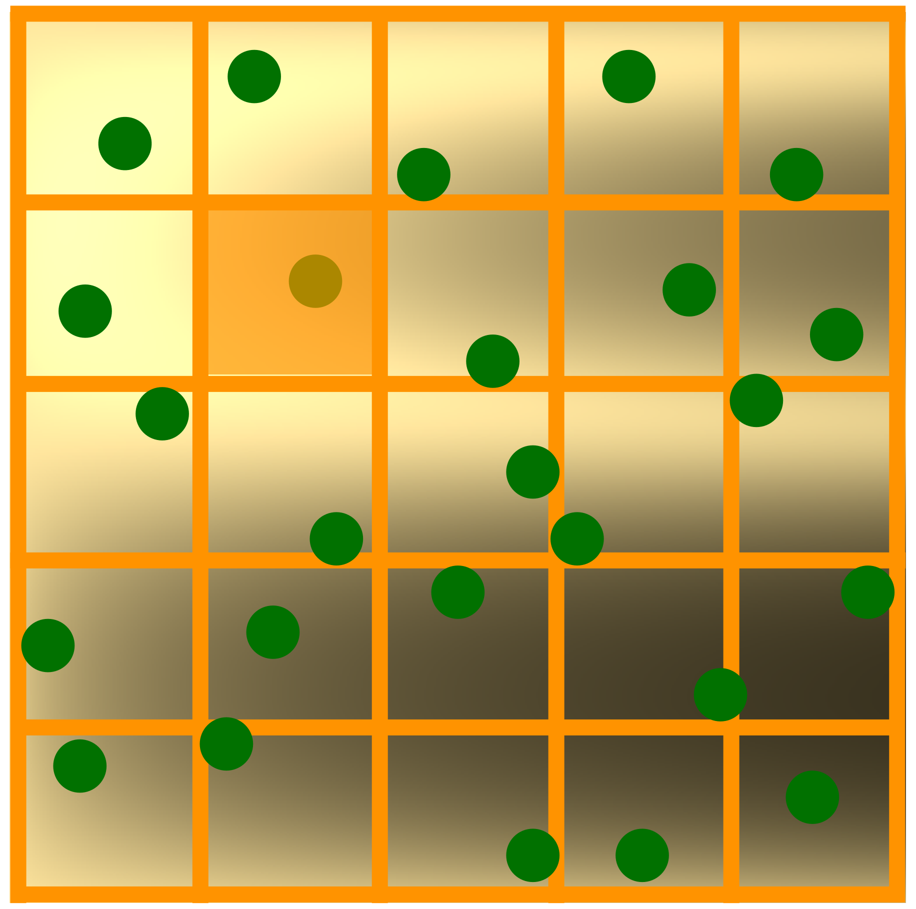
Reconstruction: Kernel Gather



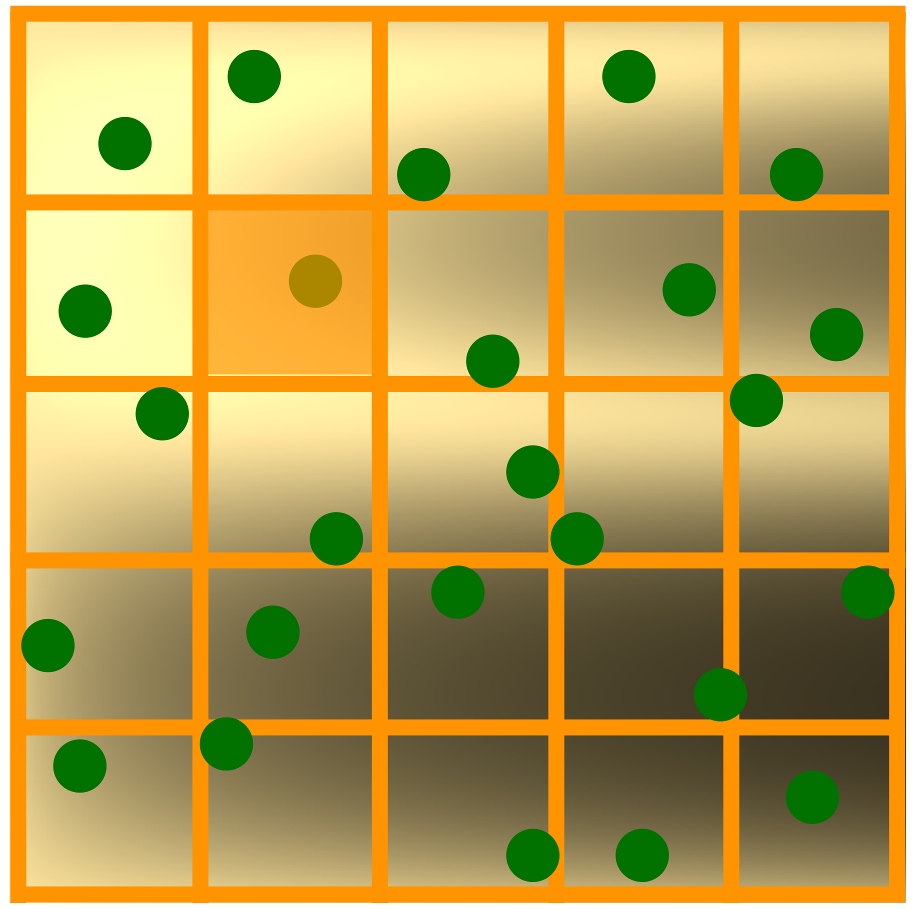
Reconstruction: Kernel Gather



Reconstruction: Kernel Gather



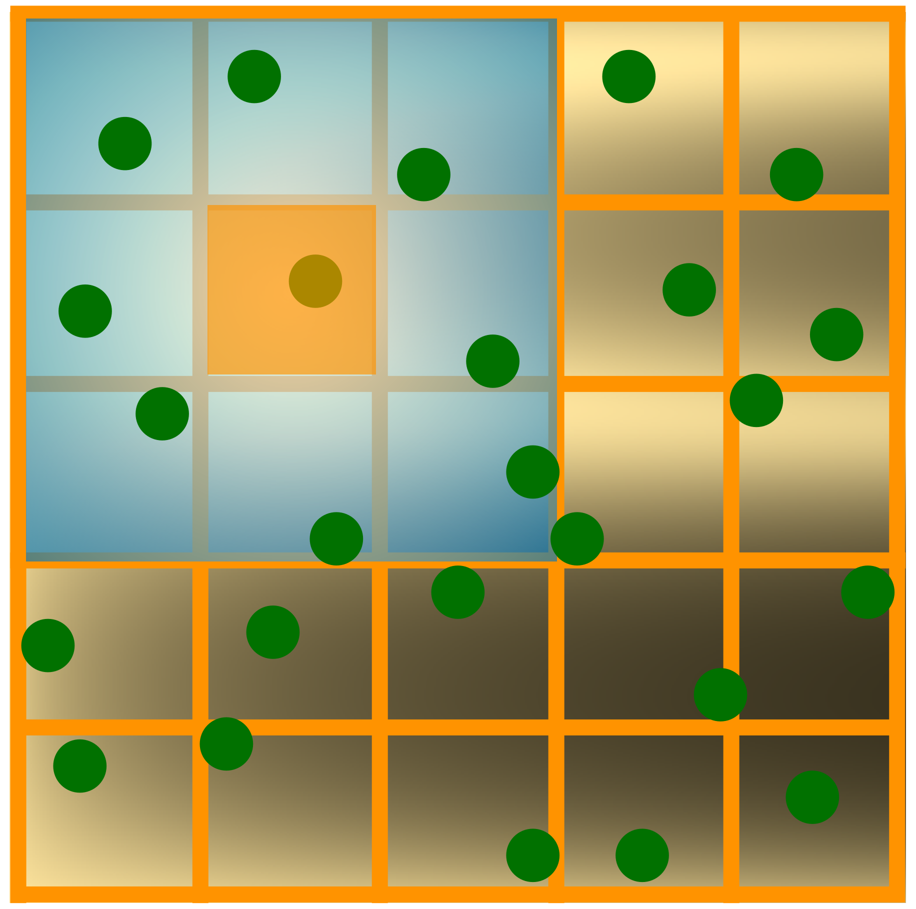
Reconstruction: Kernel Gather



Kernel

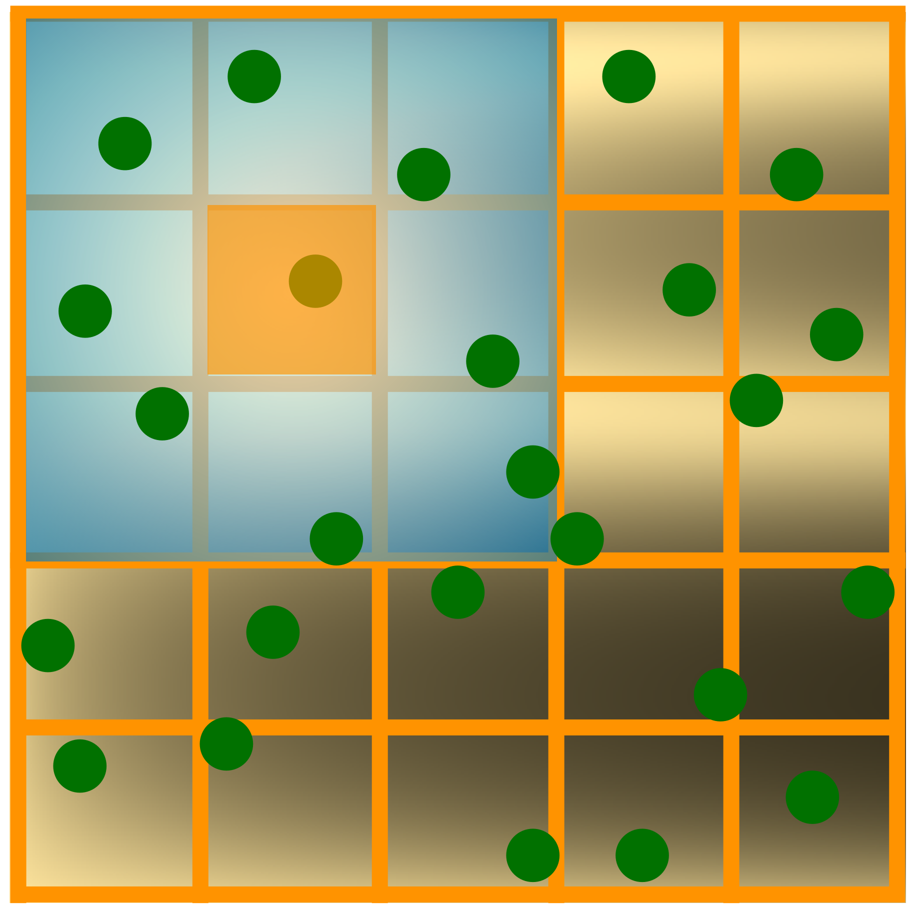
Reconstruction: Kernel Gather

Kernel



Reconstruction: Kernel Gather

Kernel gather

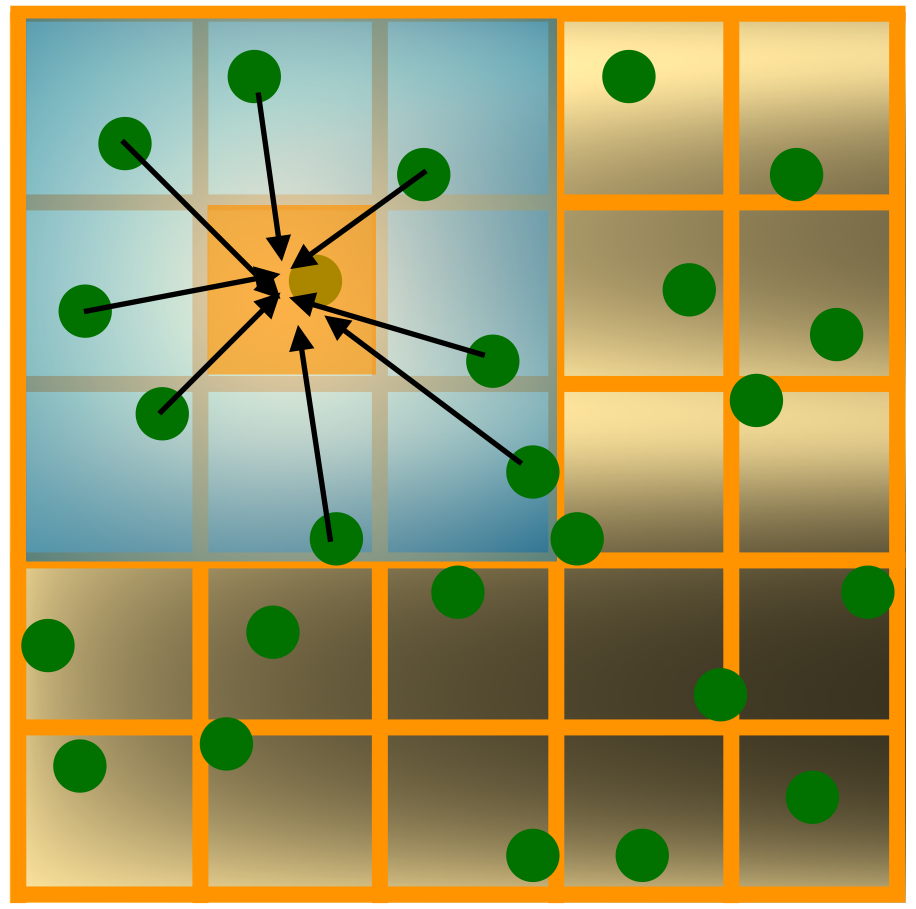


2D example

How should nearby samples influence me?

Reconstruction: Kernel Gather

Kernel gather

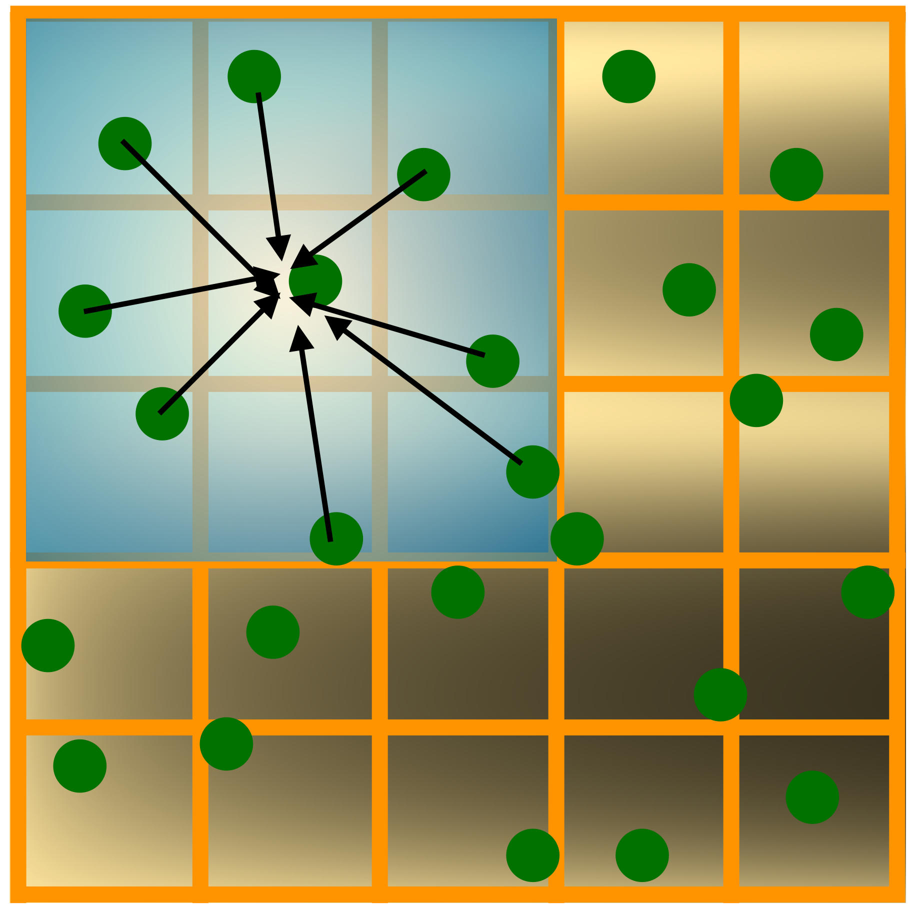


2D example

How should nearby samples influence me?

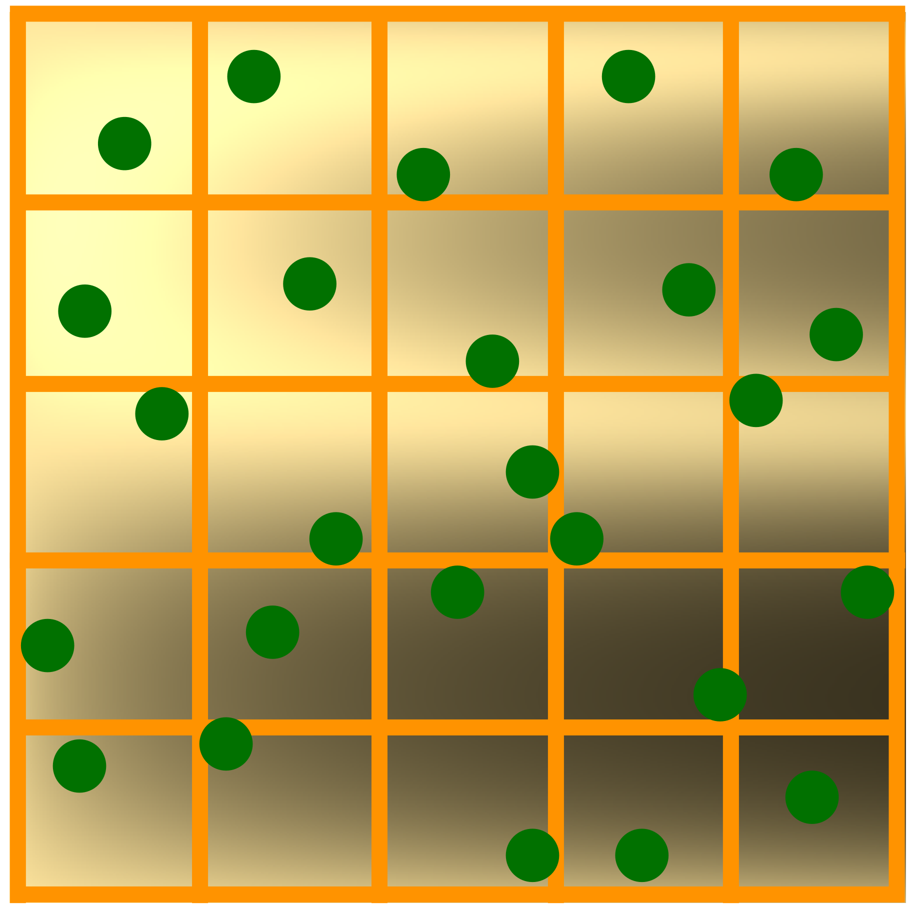
Reconstruction: Kernel Splatting

Kernel gather



2D example

Kernel Splatting

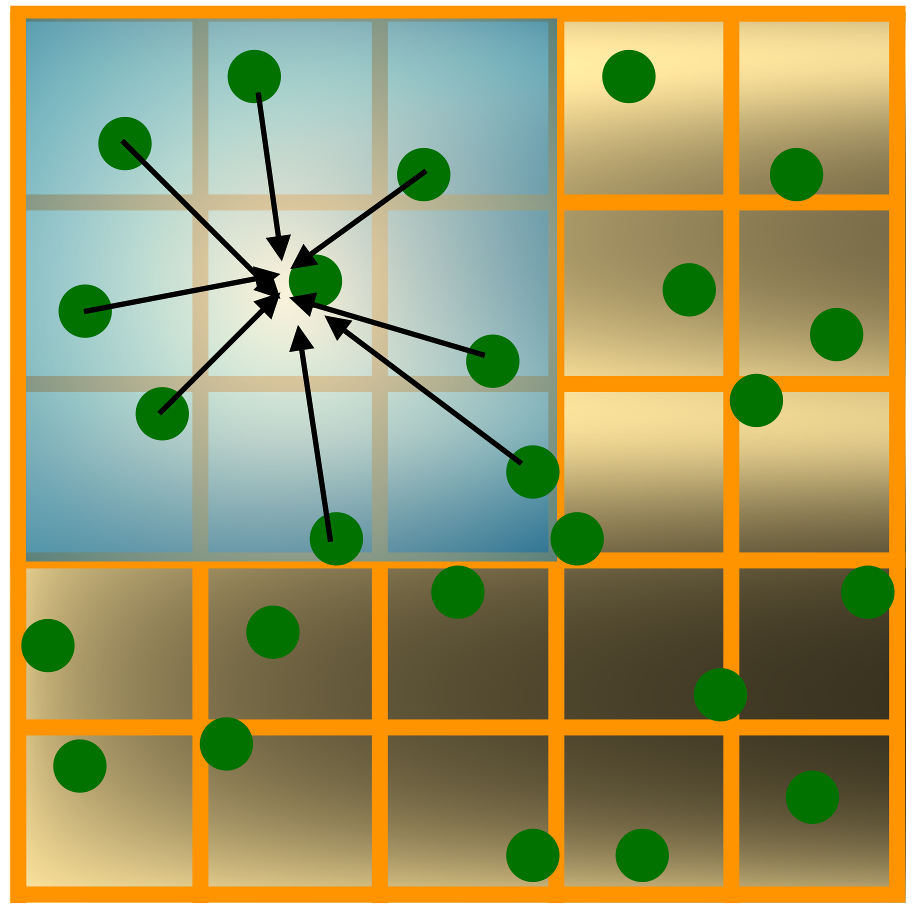


2D example

How should nearby samples influence me?

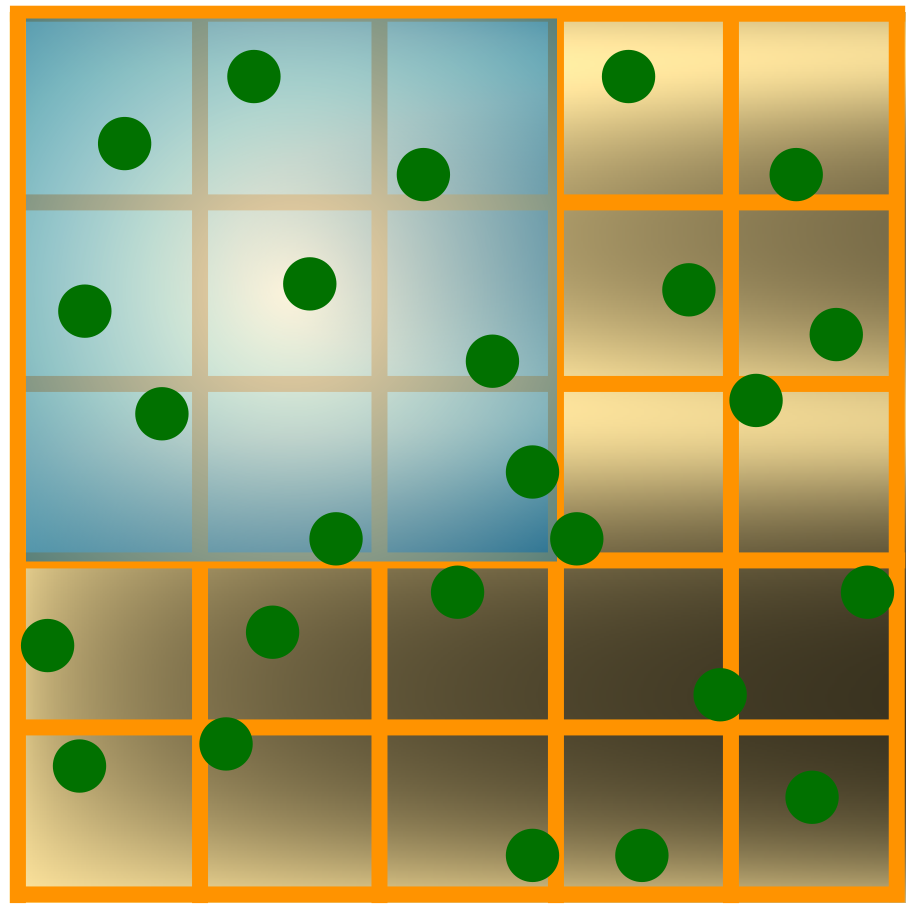
Reconstruction: Kernel Splatting

Kernel gather



2D example

Kernel Splatting

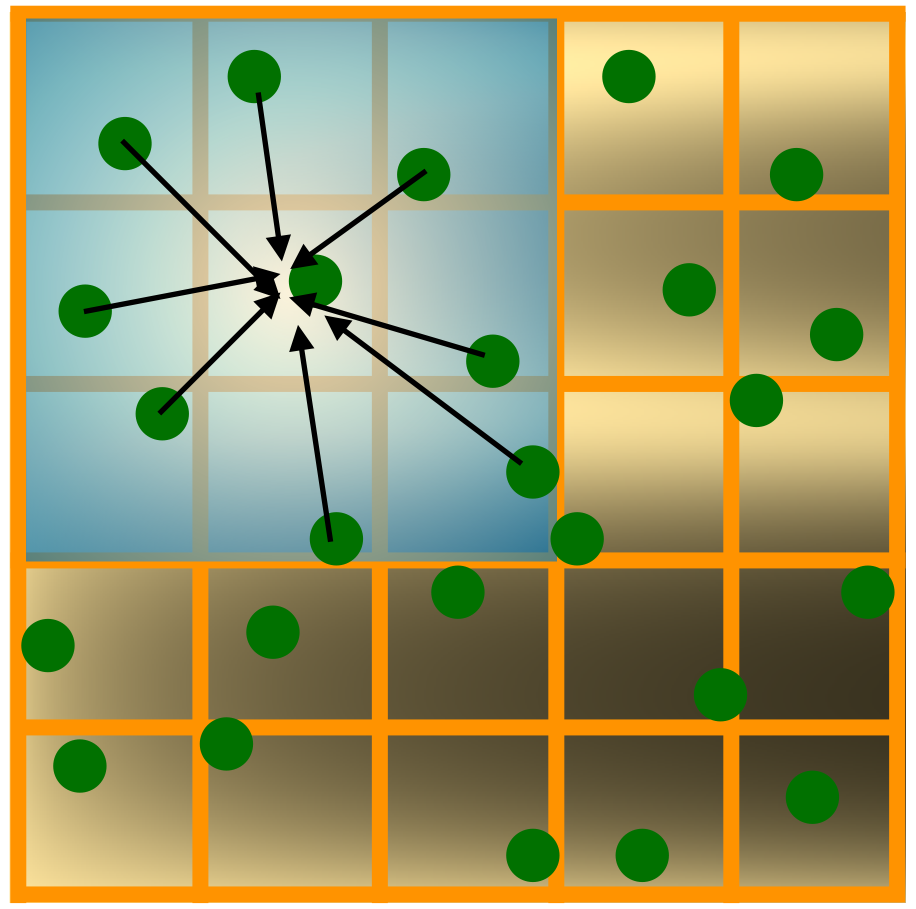


2D example

How should nearby samples influence me?

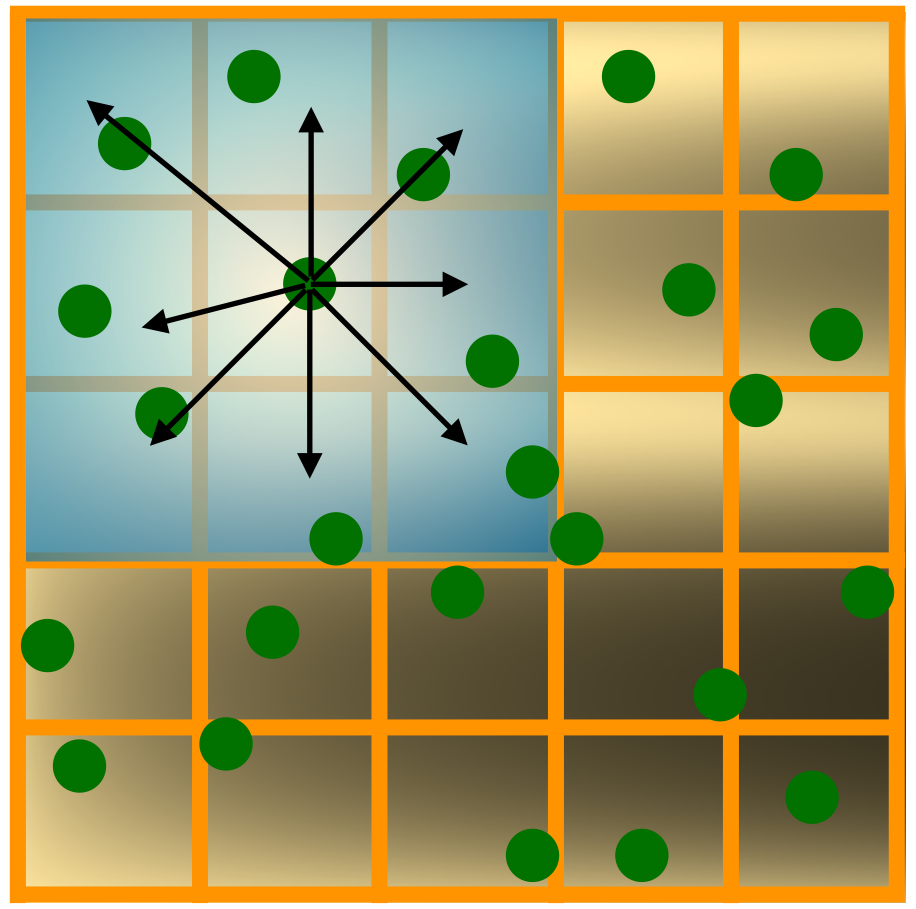
Reconstruction: Kernel Splatting

Kernel gather



2D example

Kernel Splatting

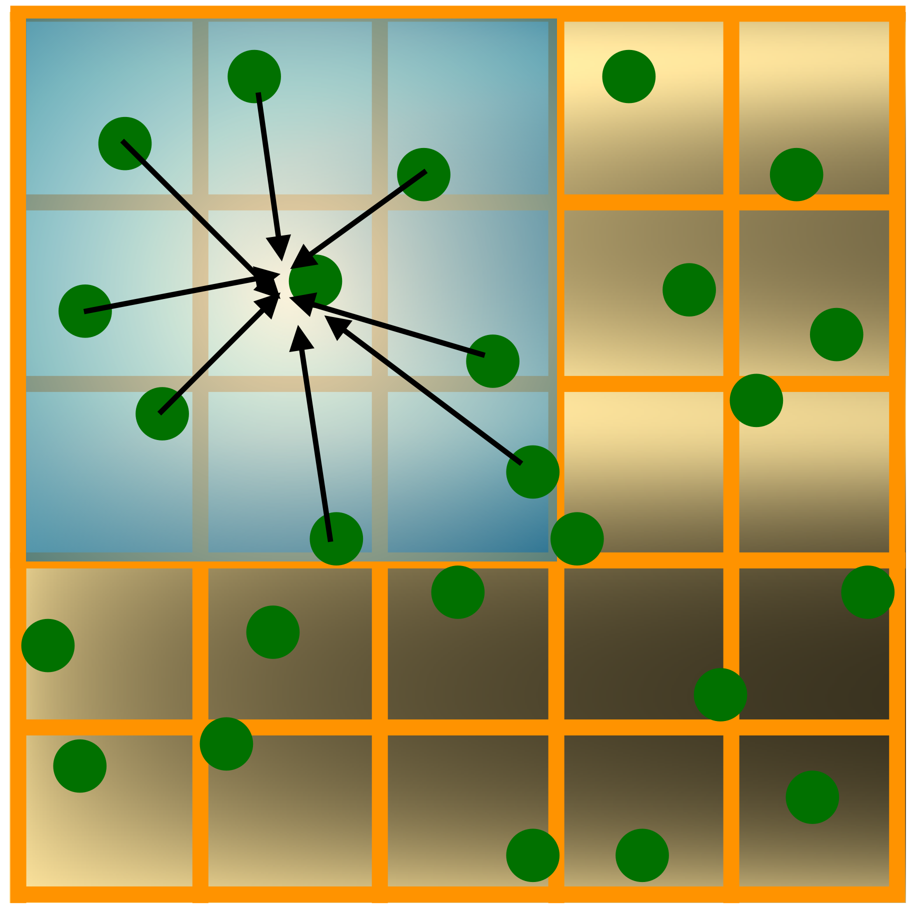


2D example

How should nearby samples influence me?

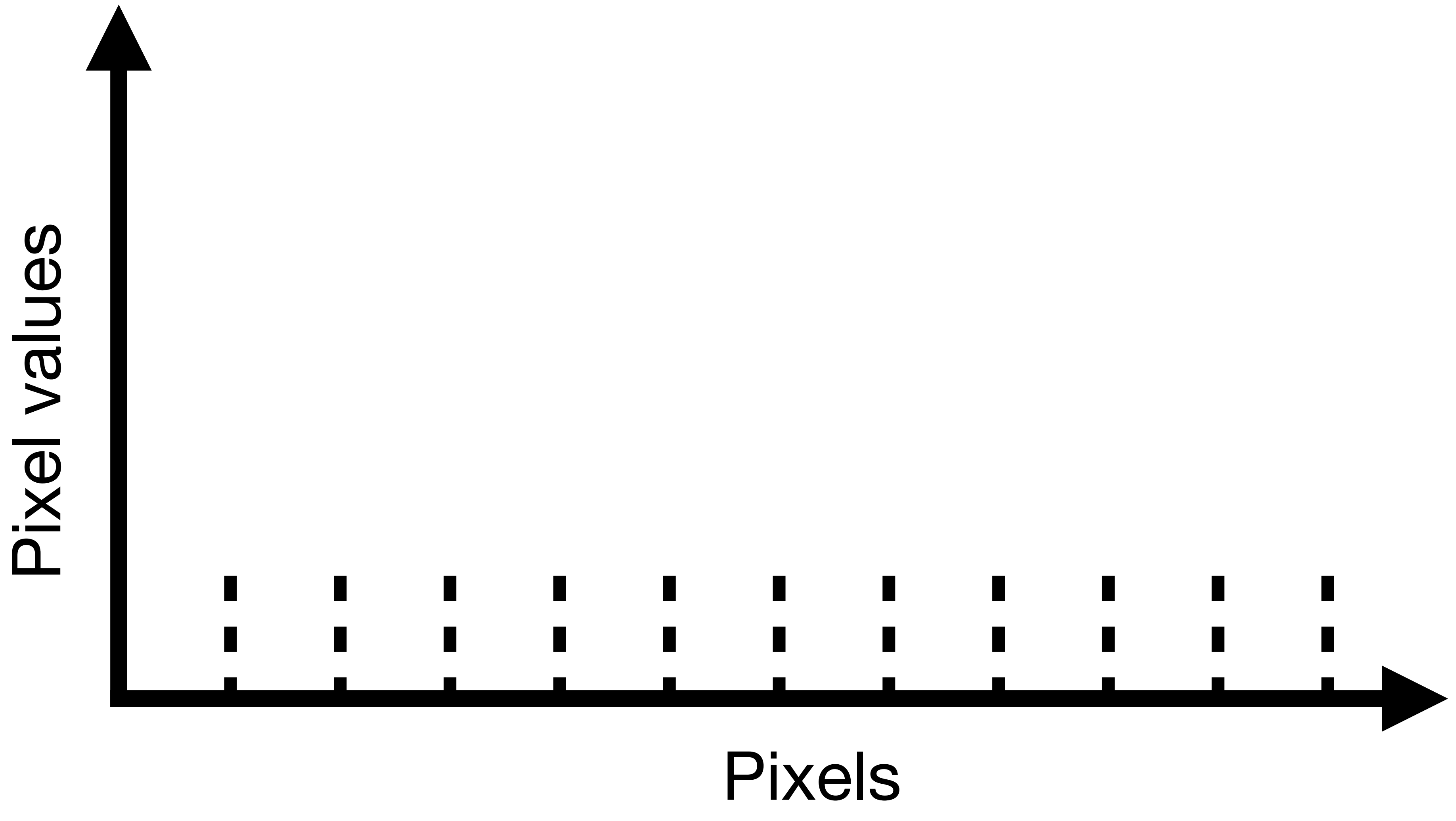
Reconstruction: Kernel Splatting

Kernel gather



2D example

Kernel Splatting

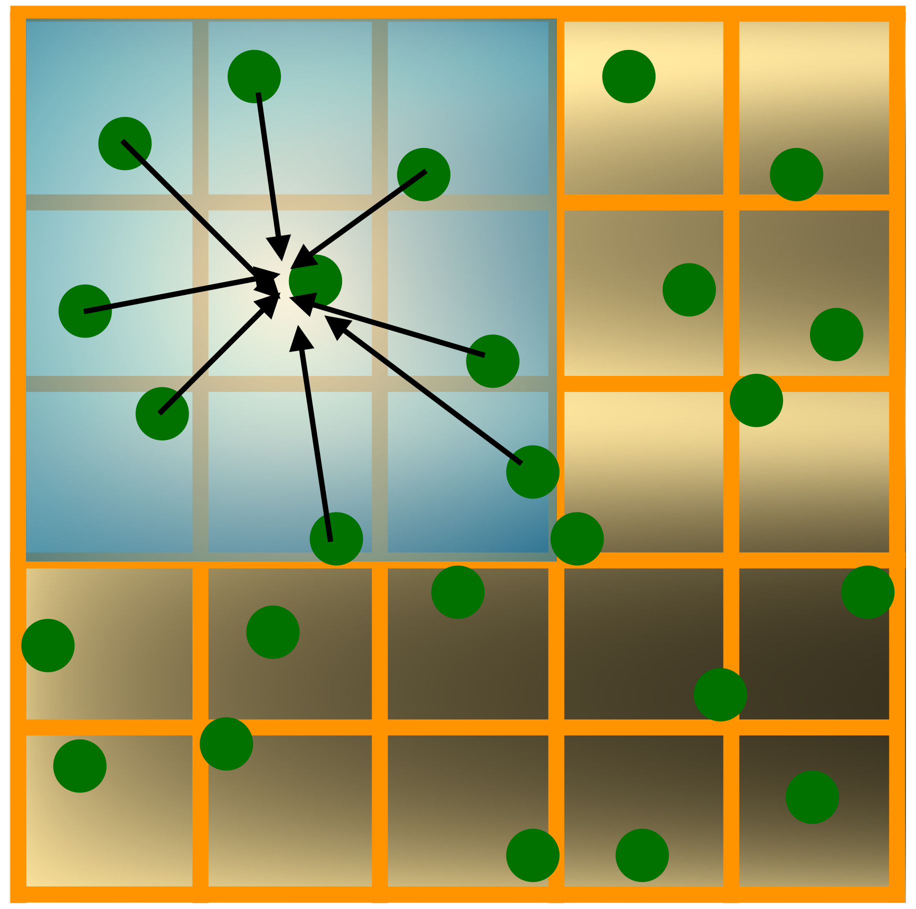


1D example

How should nearby samples influence me?

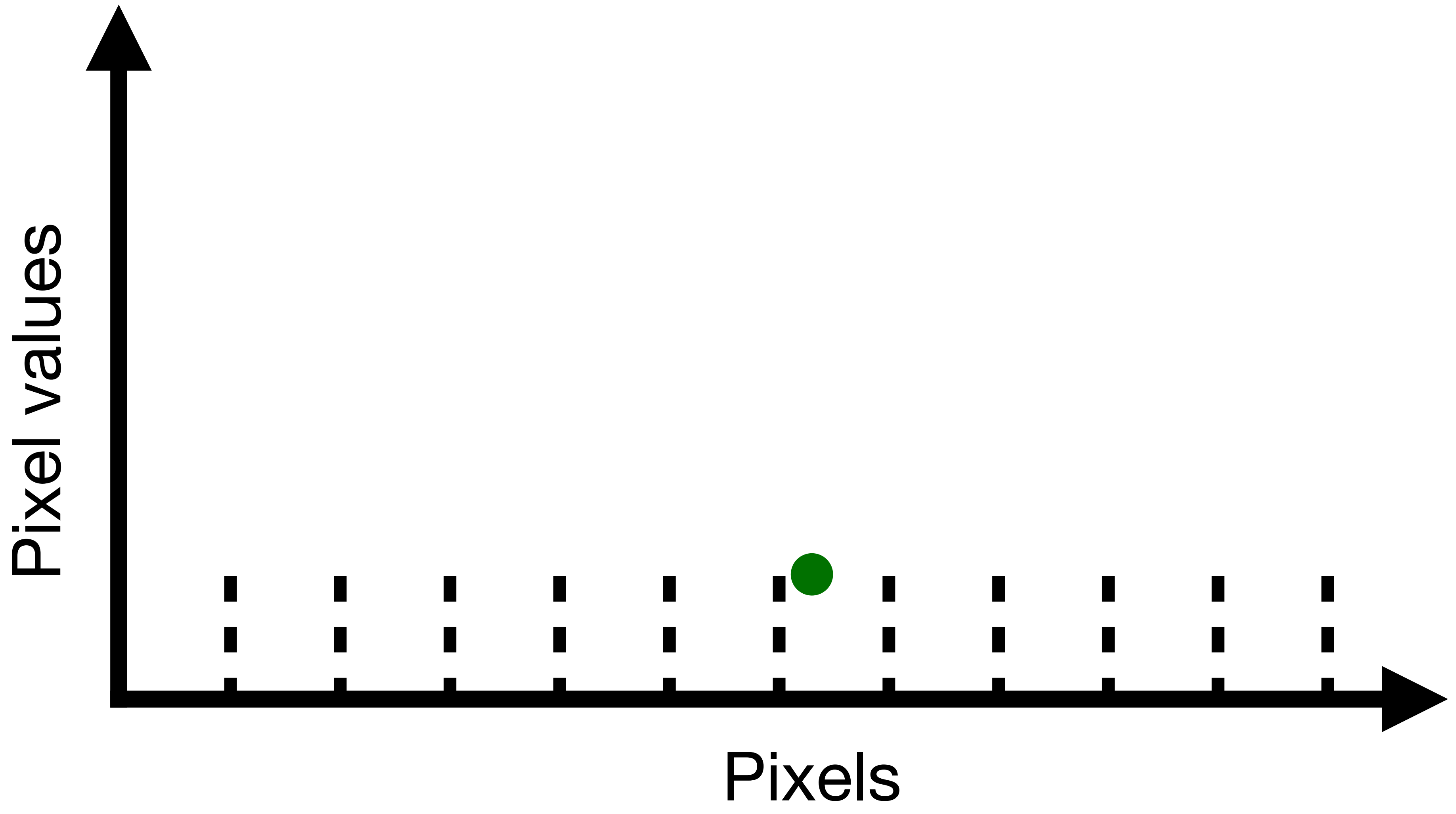
Reconstruction: Kernel Splatting

Kernel gather



2D example

Kernel Splatting

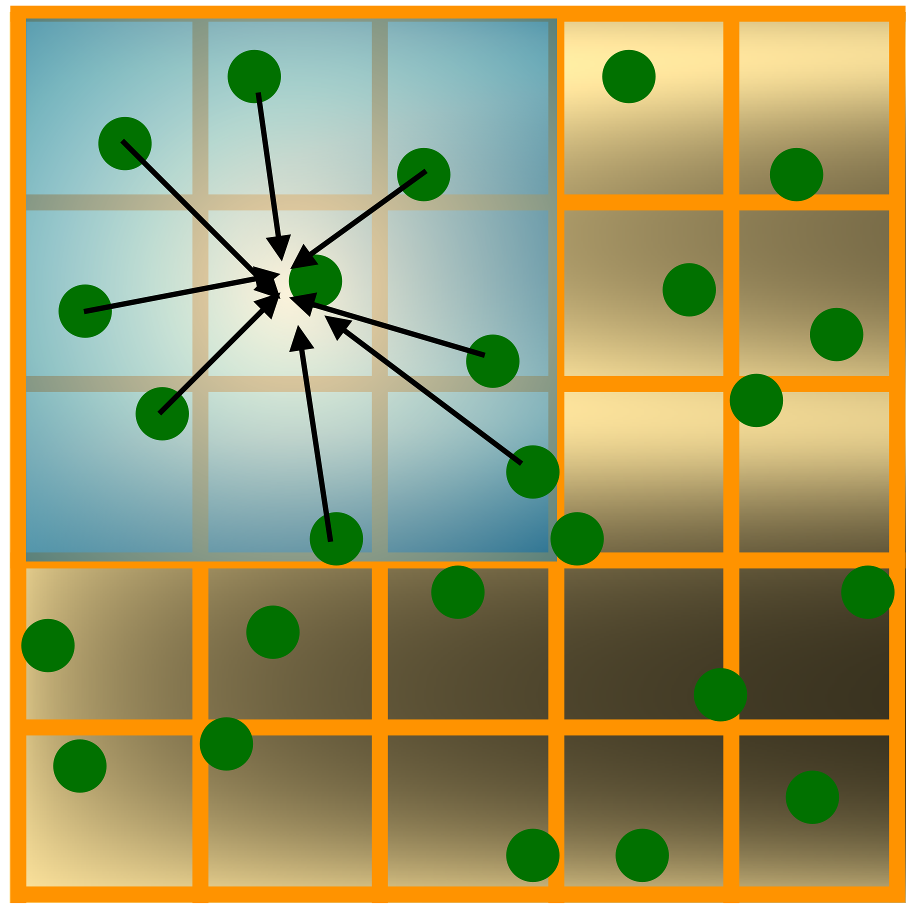


1D example

How should nearby samples influence me?

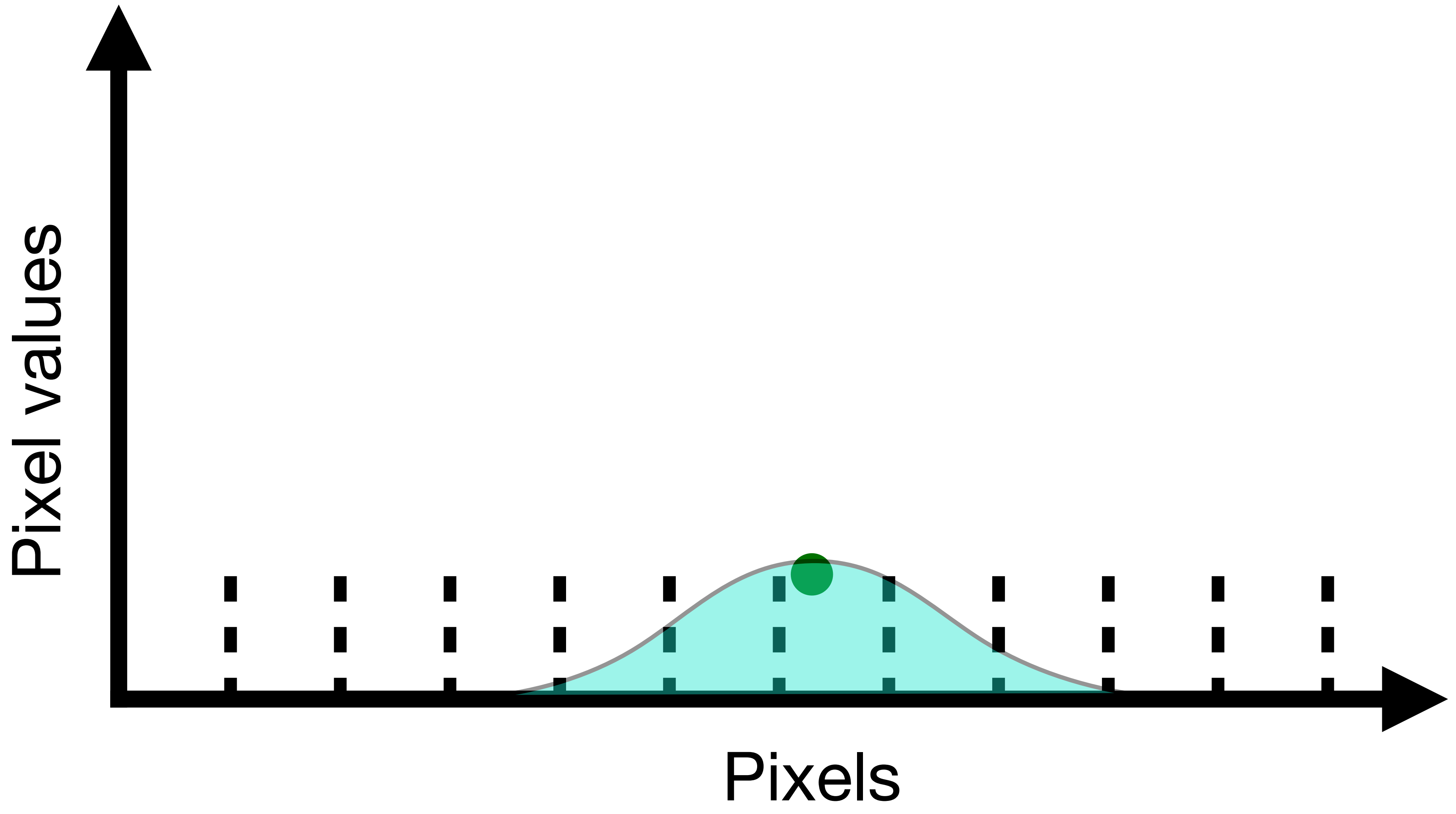
Reconstruction: Kernel Splatting

Kernel gather



2D example

Kernel Splatting

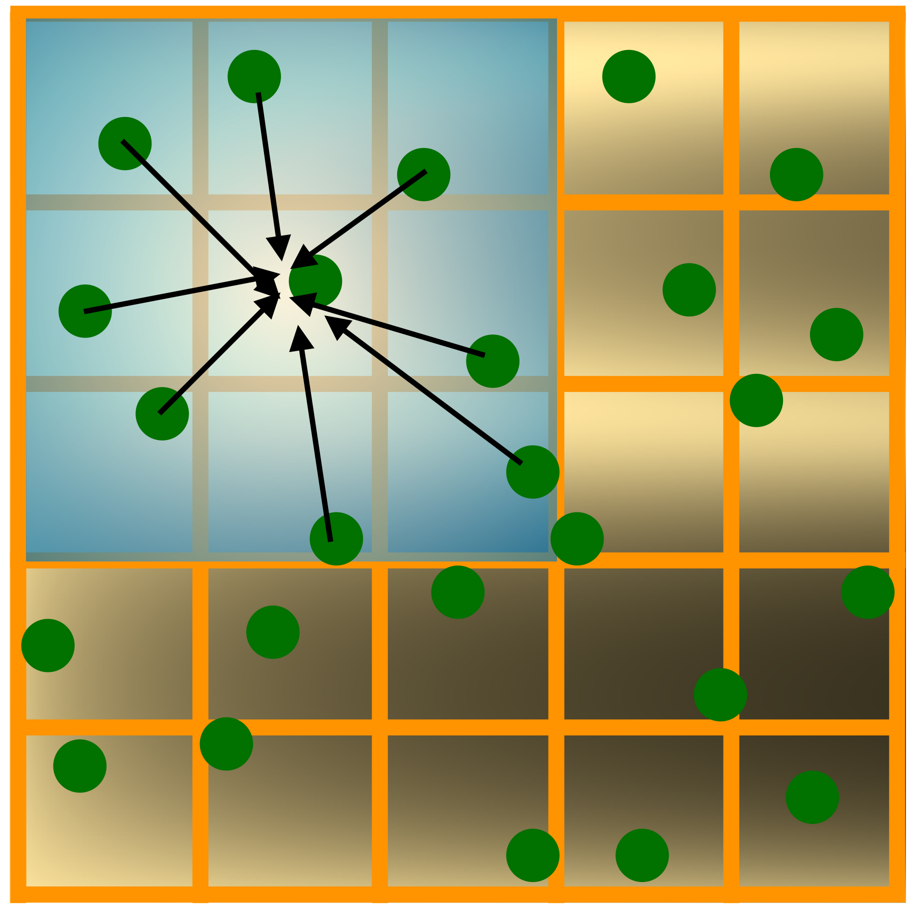


1D example

How should nearby samples influence me?

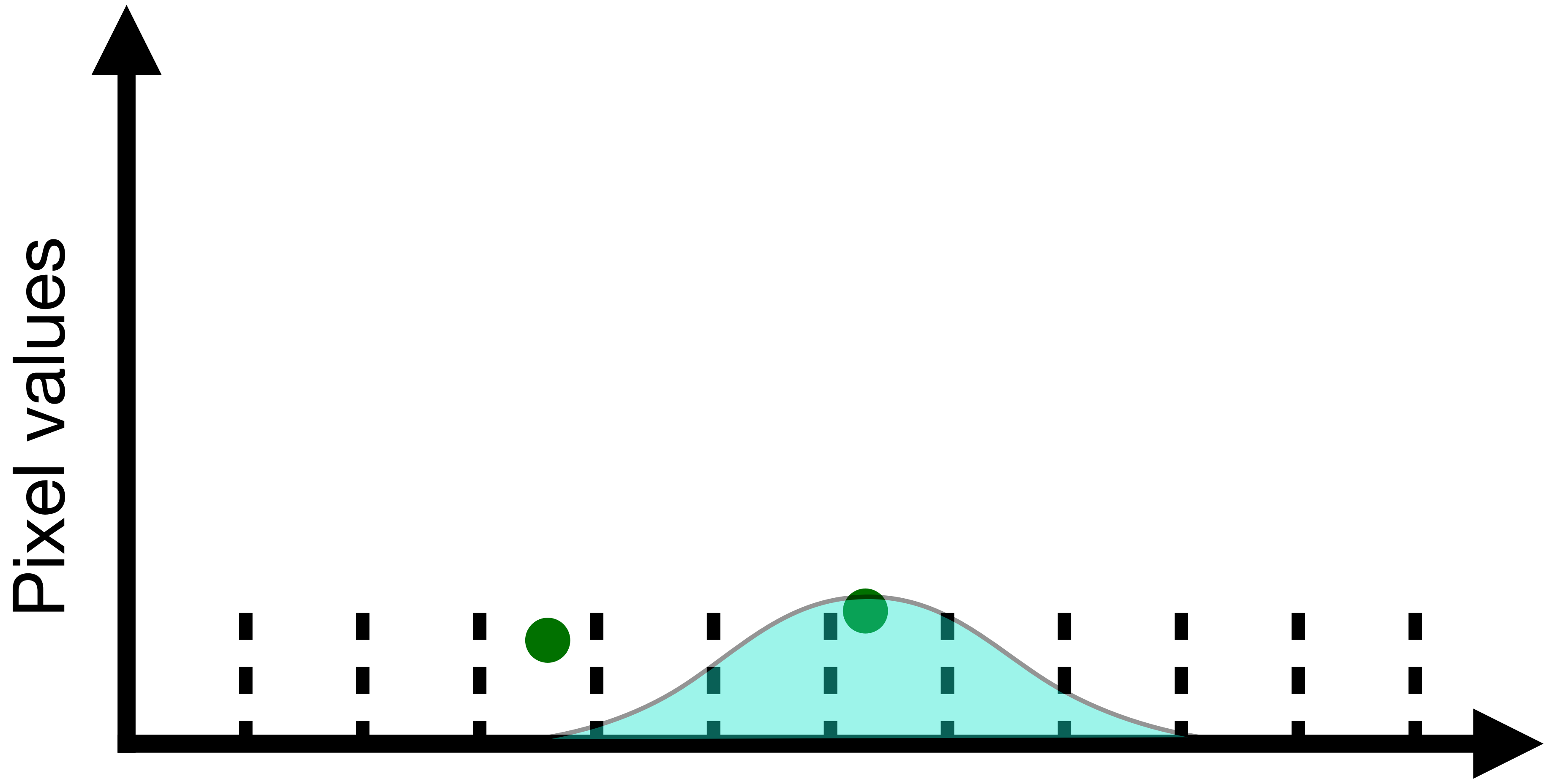
Reconstruction: Kernel Splatting

Kernel gather



2D example

Kernel Splatting

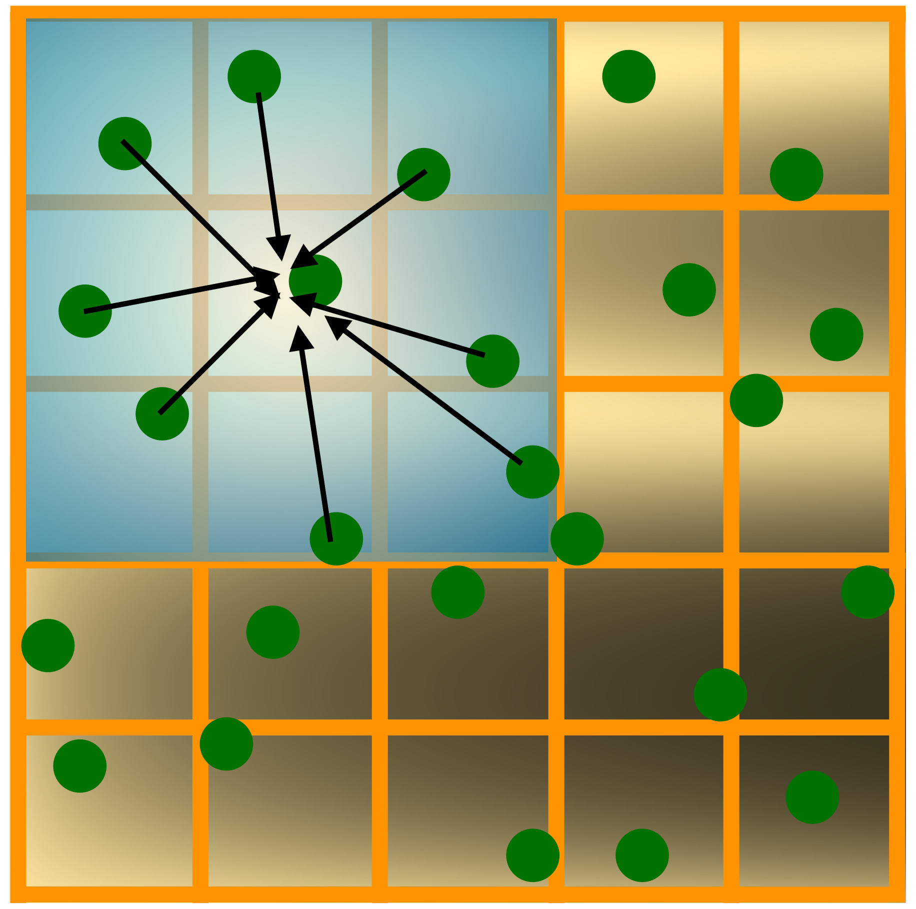


1D example

How should nearby samples influence me?

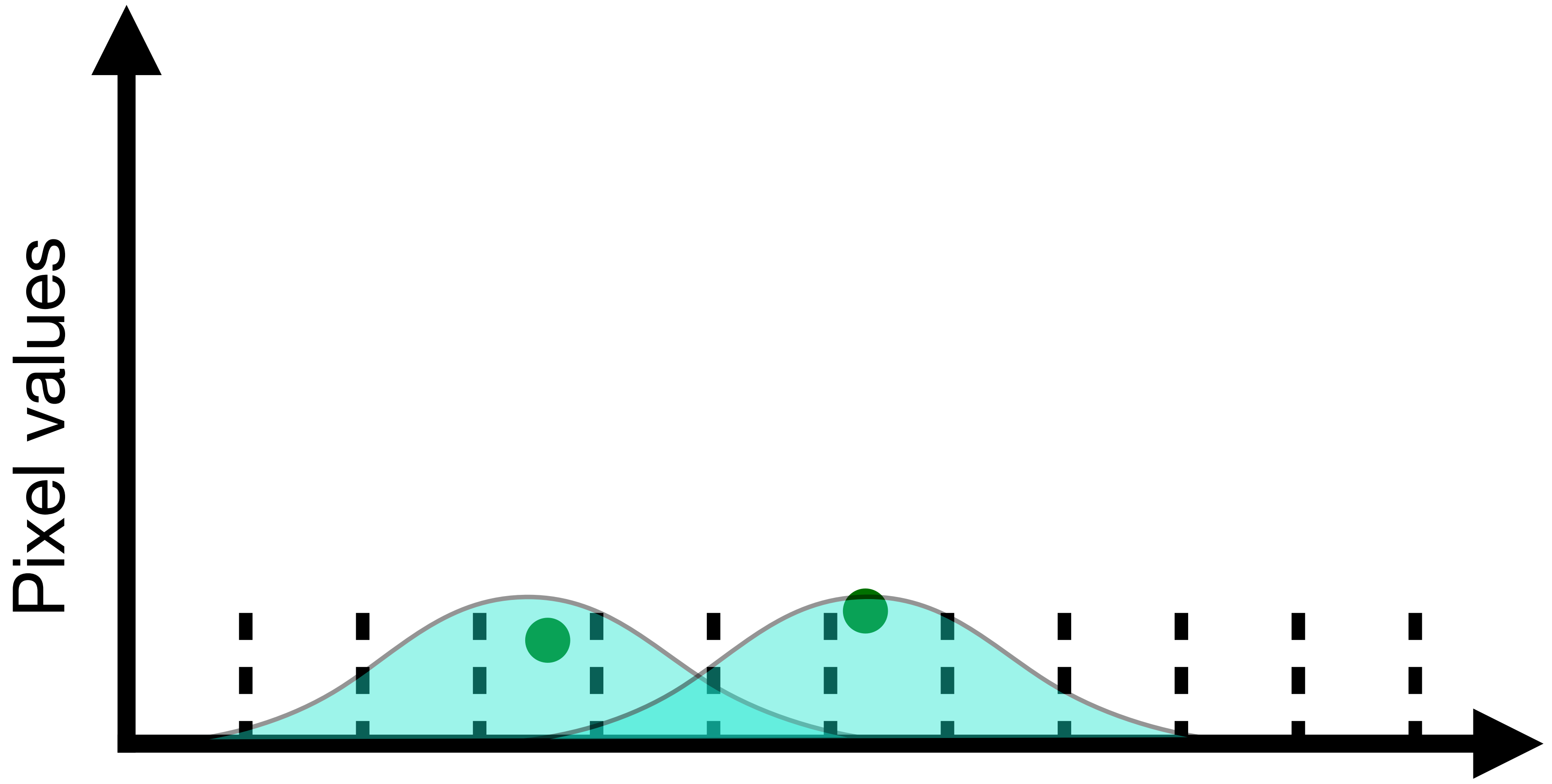
Reconstruction: Kernel Splatting

Kernel gather



2D example

Kernel Splatting

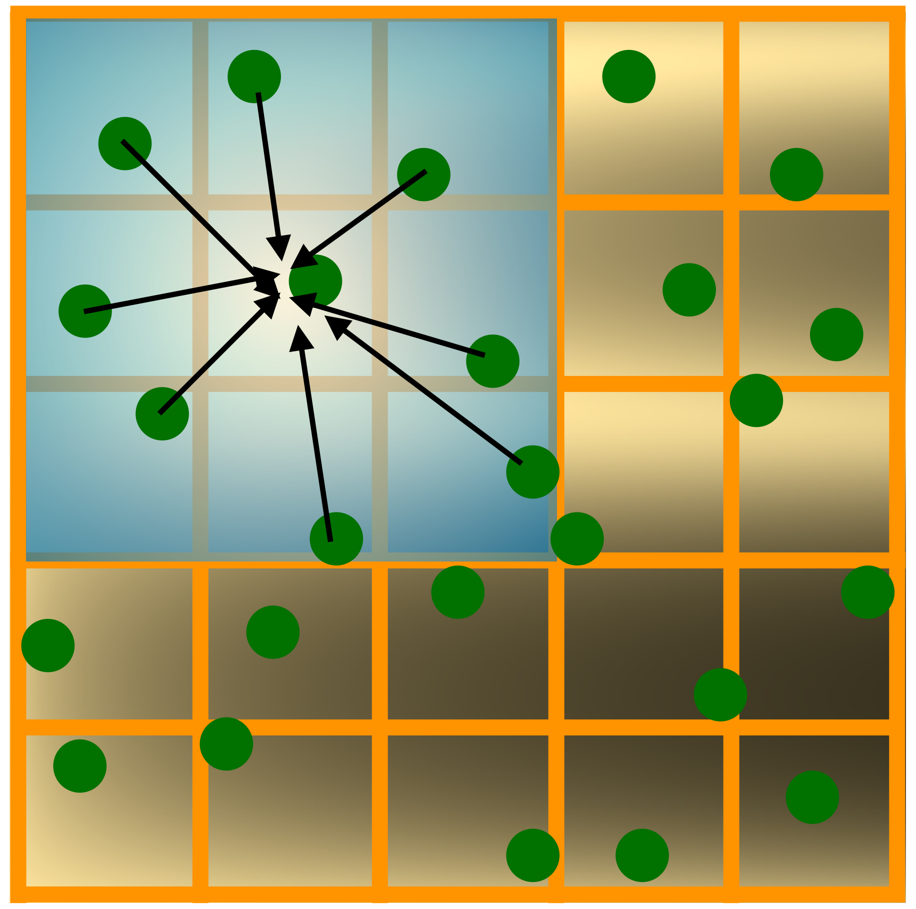


1D example

How should nearby samples influence me?

Reconstruction: Kernel Splatting

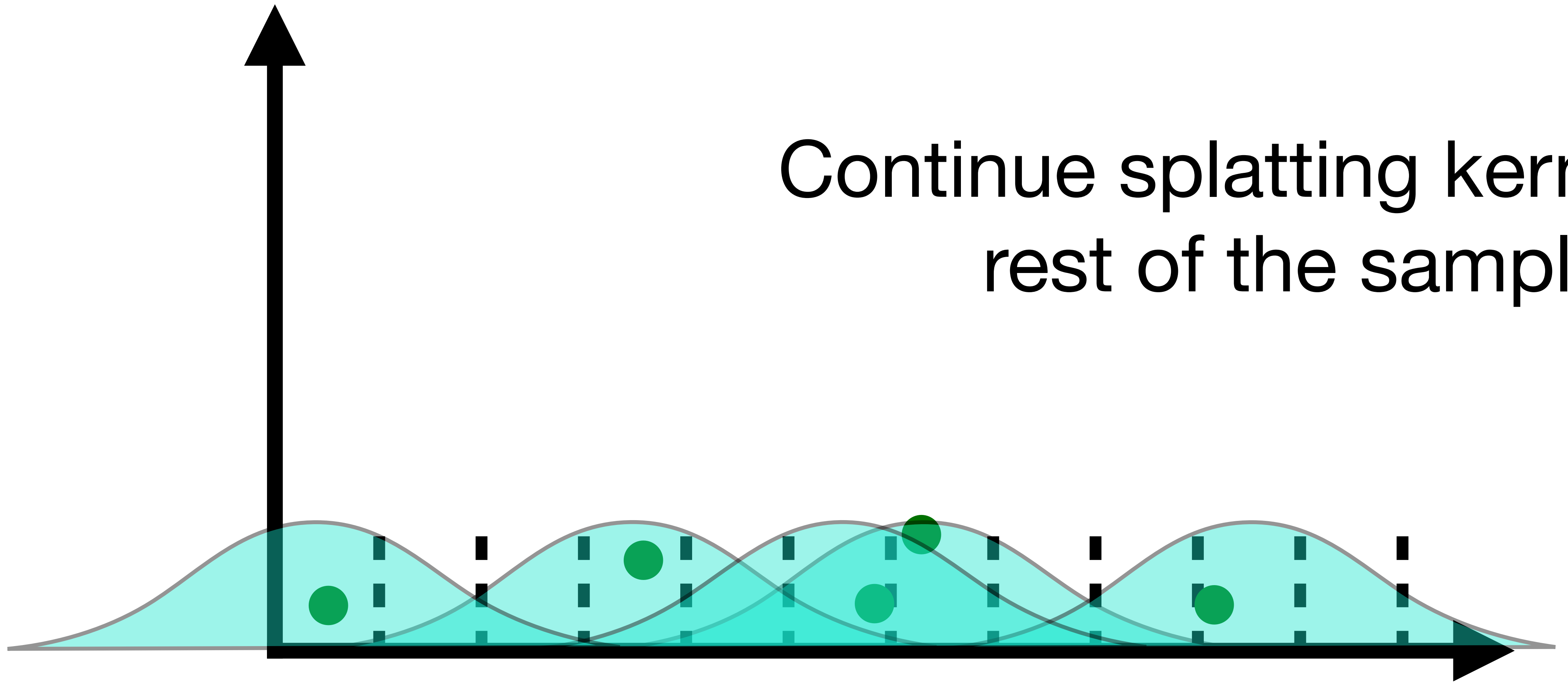
Kernel gather



2D example

How should nearby samples influence me?

Kernel Splatting

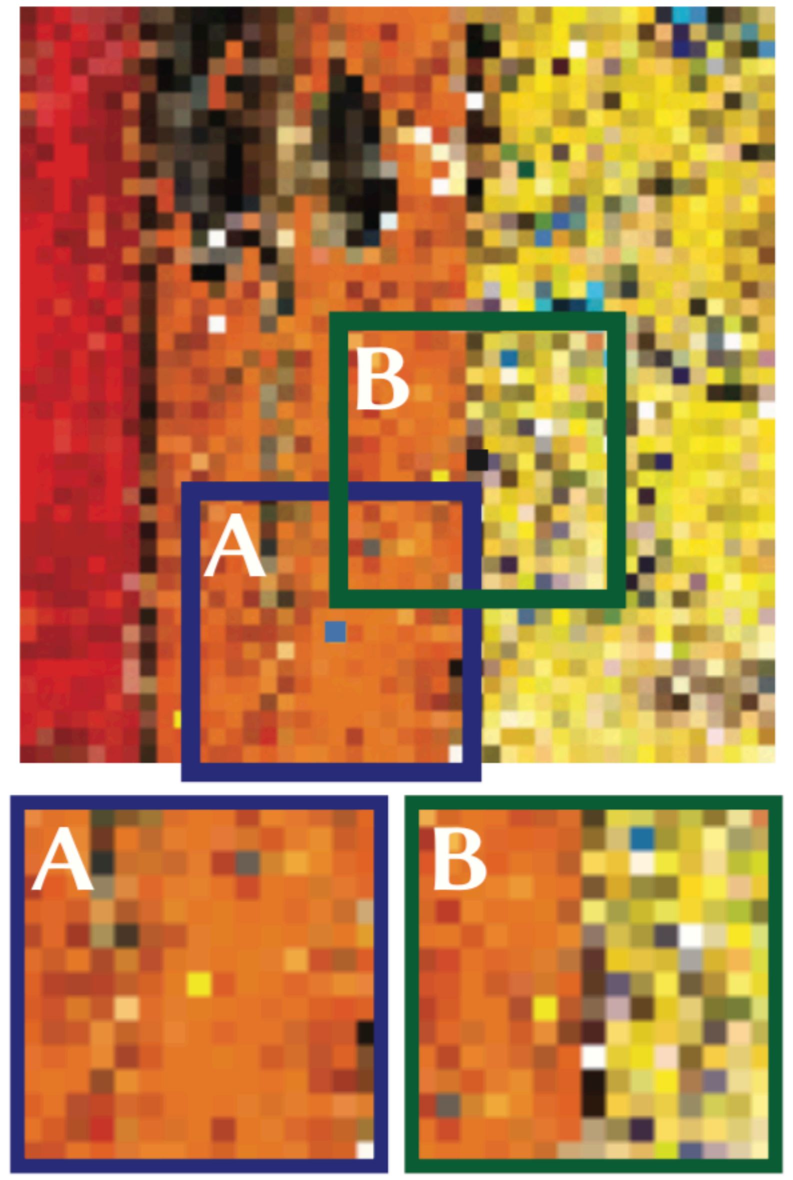


Continue splatting kernels for the rest of the samples....

How do I contribute to nearby pixels, given all the samples around me?

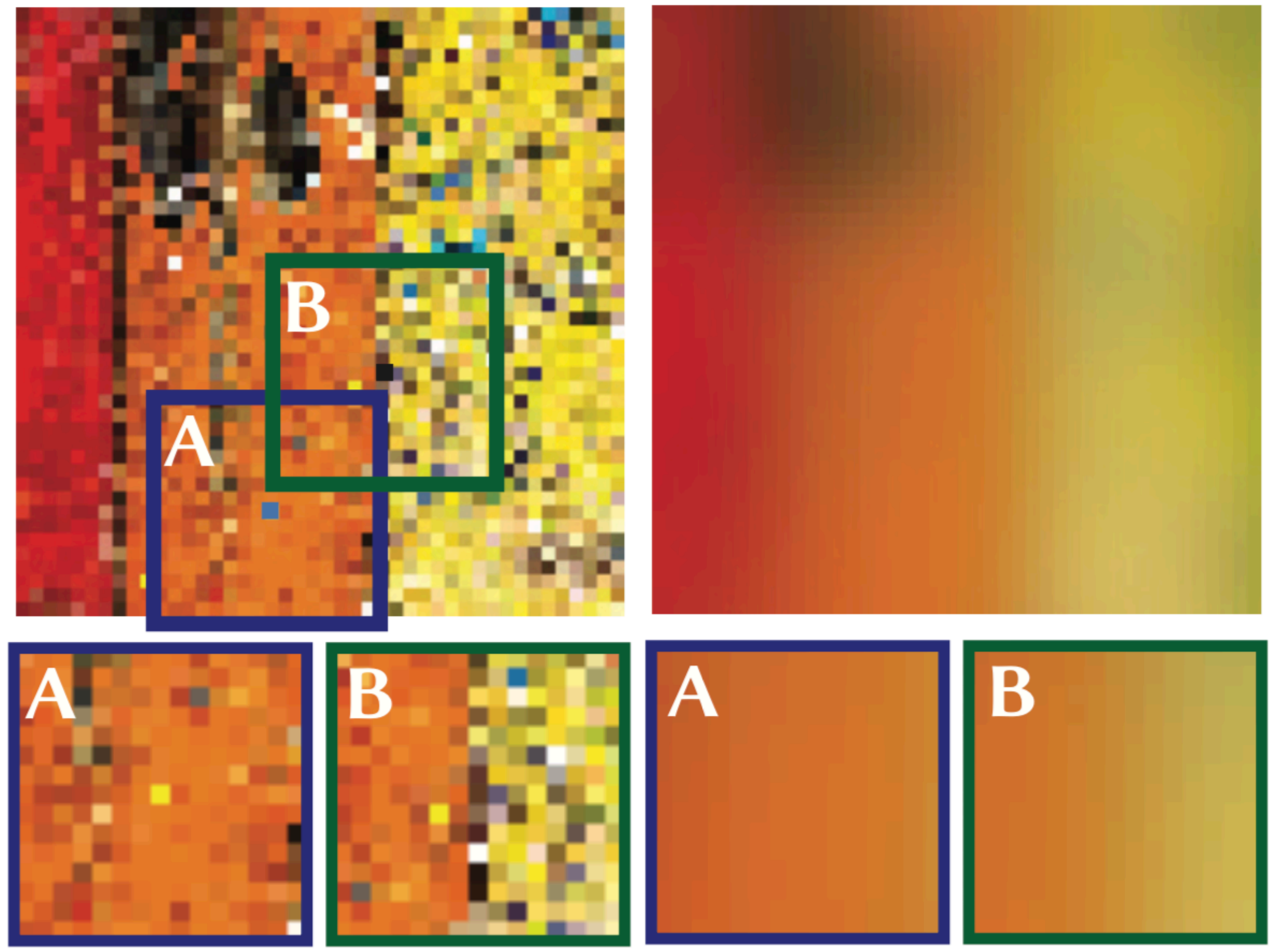
Network: Kernel Gather vs Splatting

intput

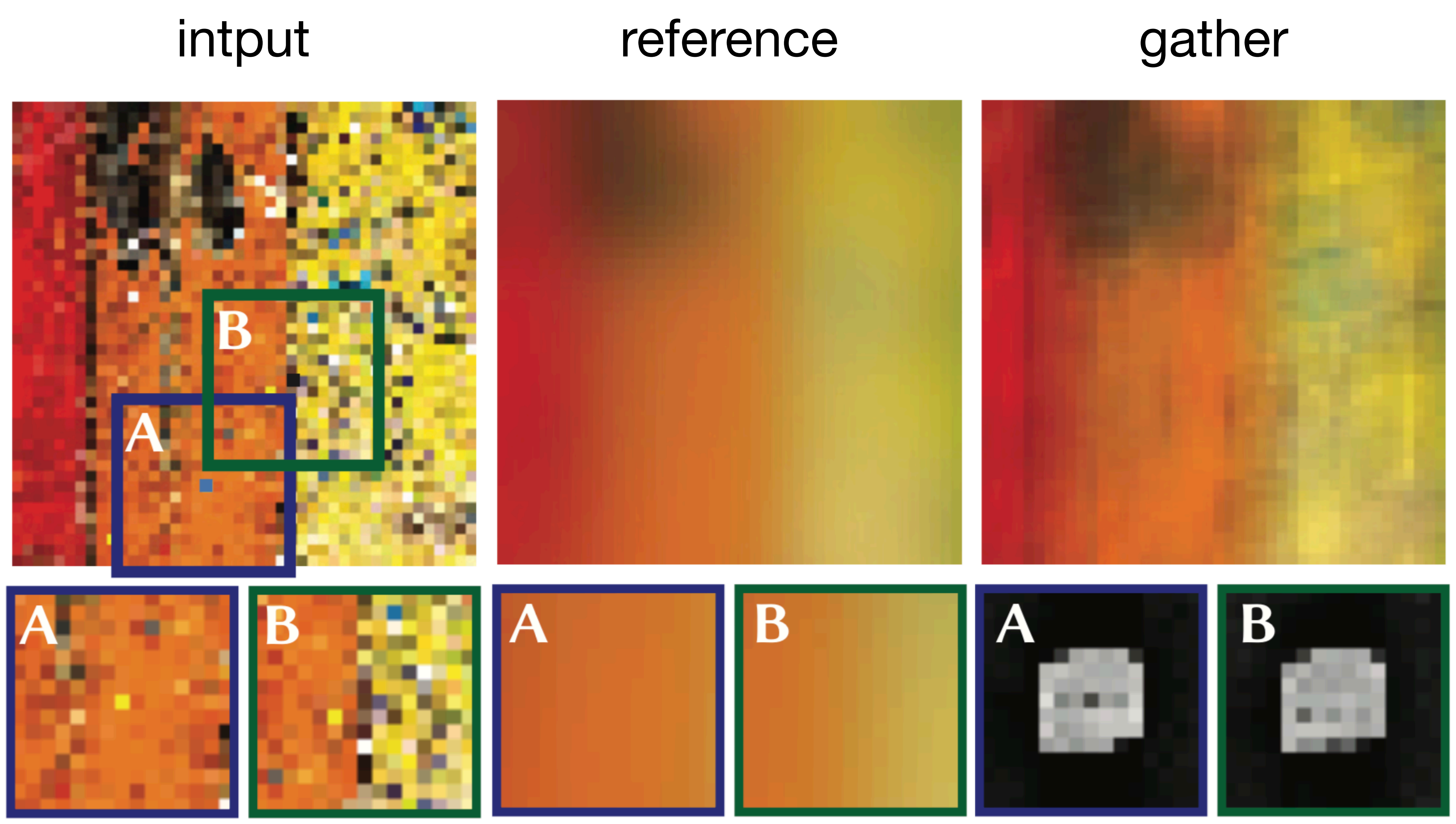


Network: Kernel Gather vs Splatting

intput reference

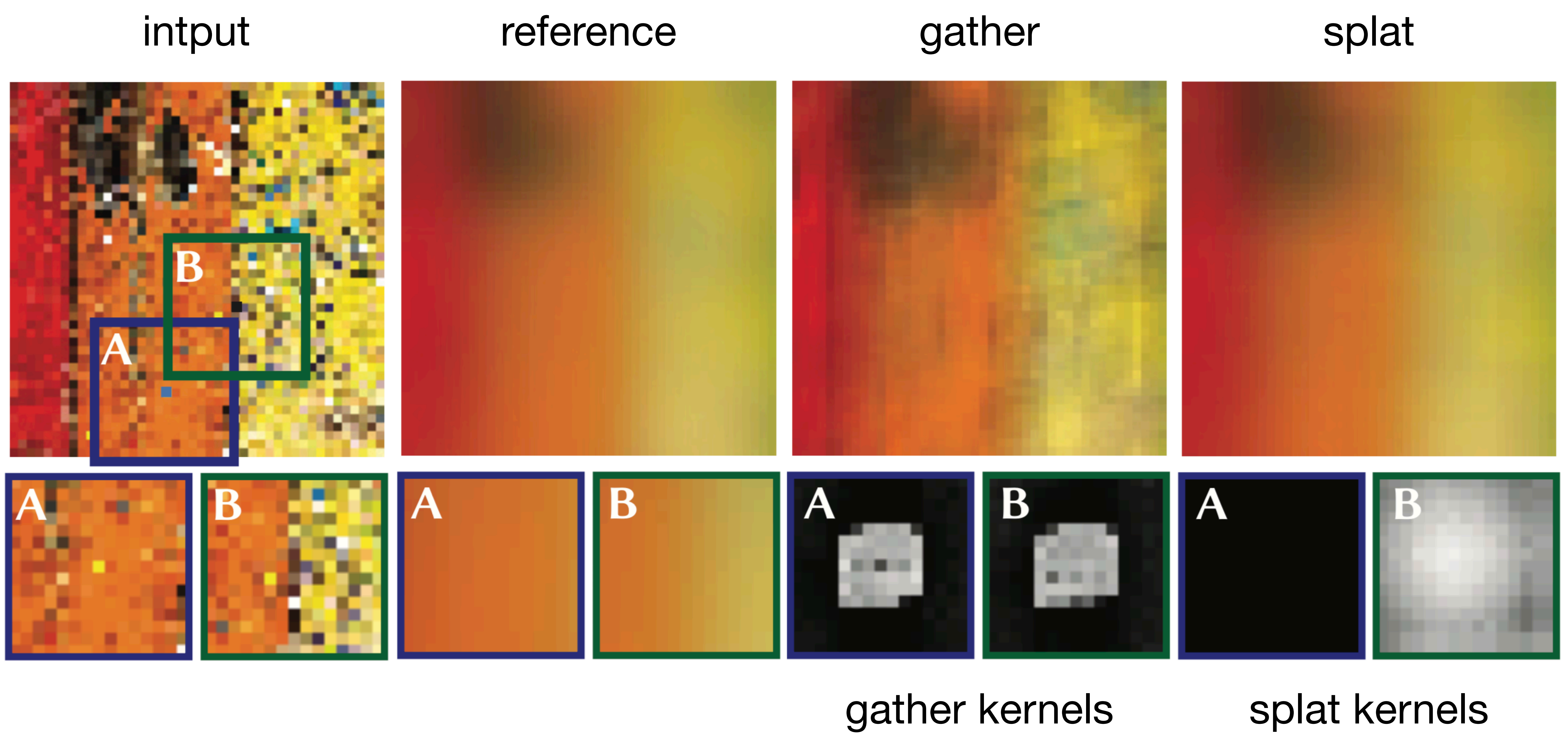


Network: Kernel Gather vs Splatting

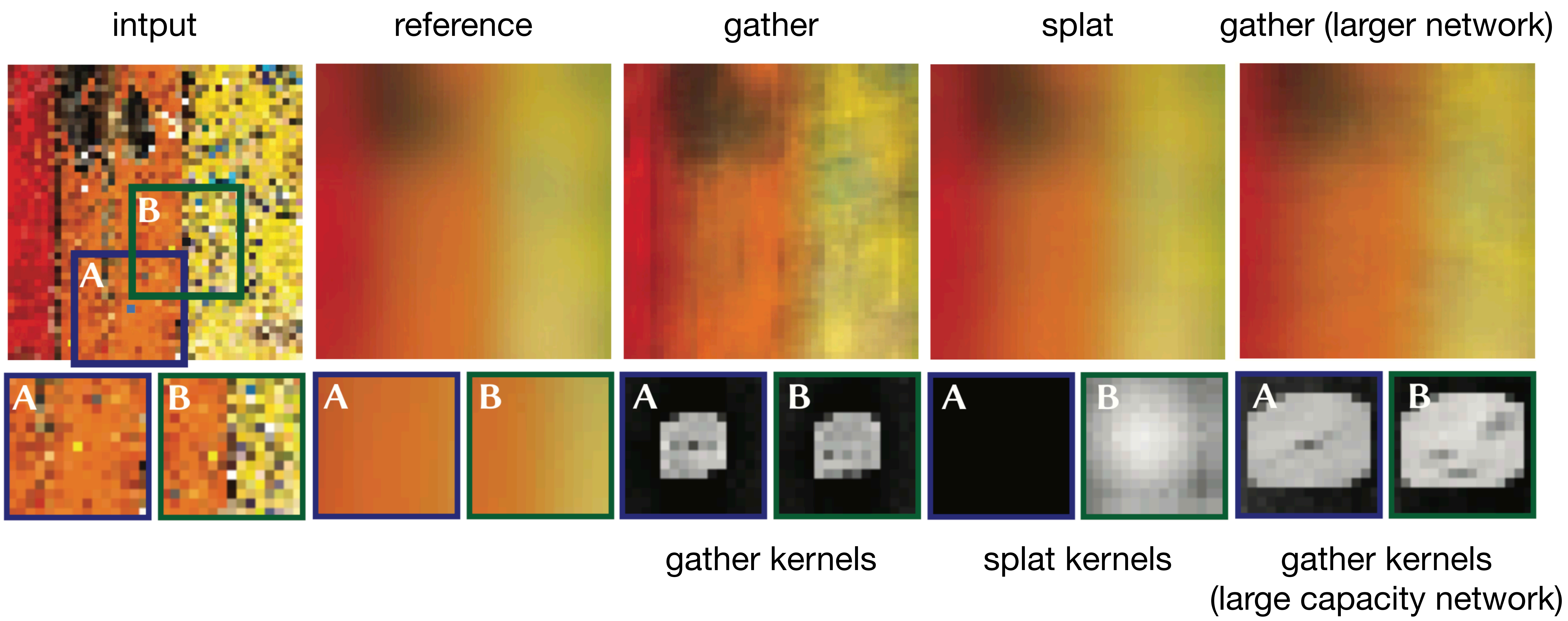


gather kernels

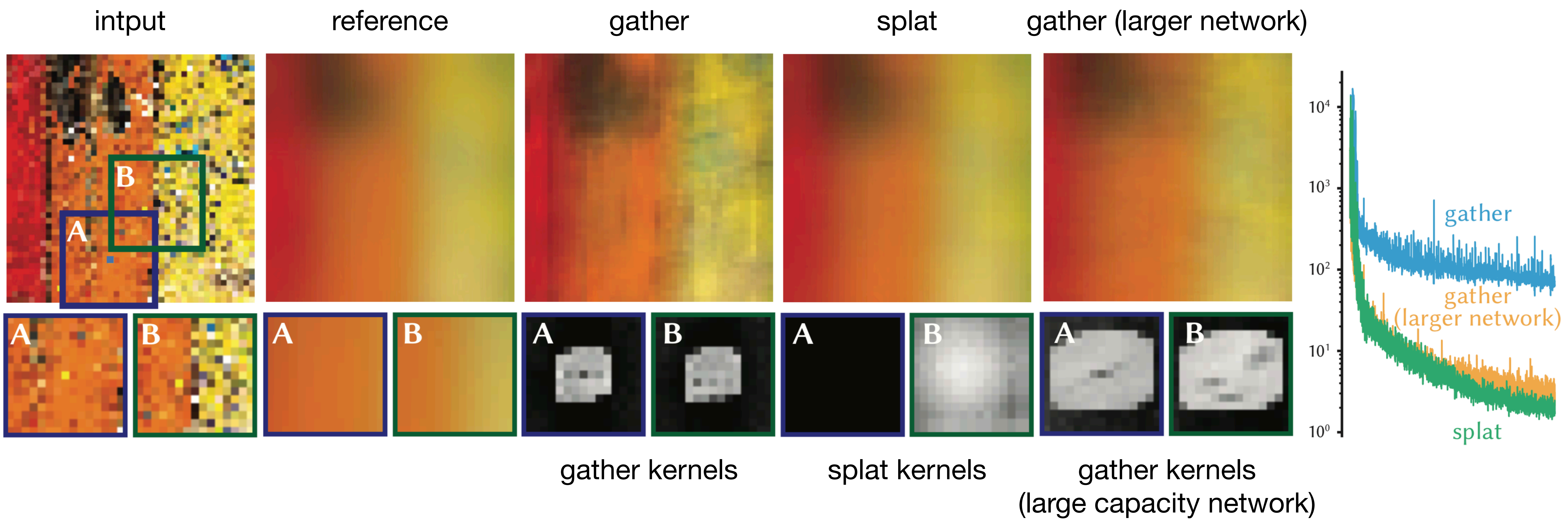
Network: Kernel Gather vs Splatting



Network: Kernel Gather vs Splatting



Network: Kernel Gather vs Splatting



Permutation Invariance

Permutation Invariance

A model that produces the same output regardless of the order of elements in the input vector

Permutation Invariance: Example

 *  =

Permutation Invariance: Example

*

=

Permutation Invariance: Example

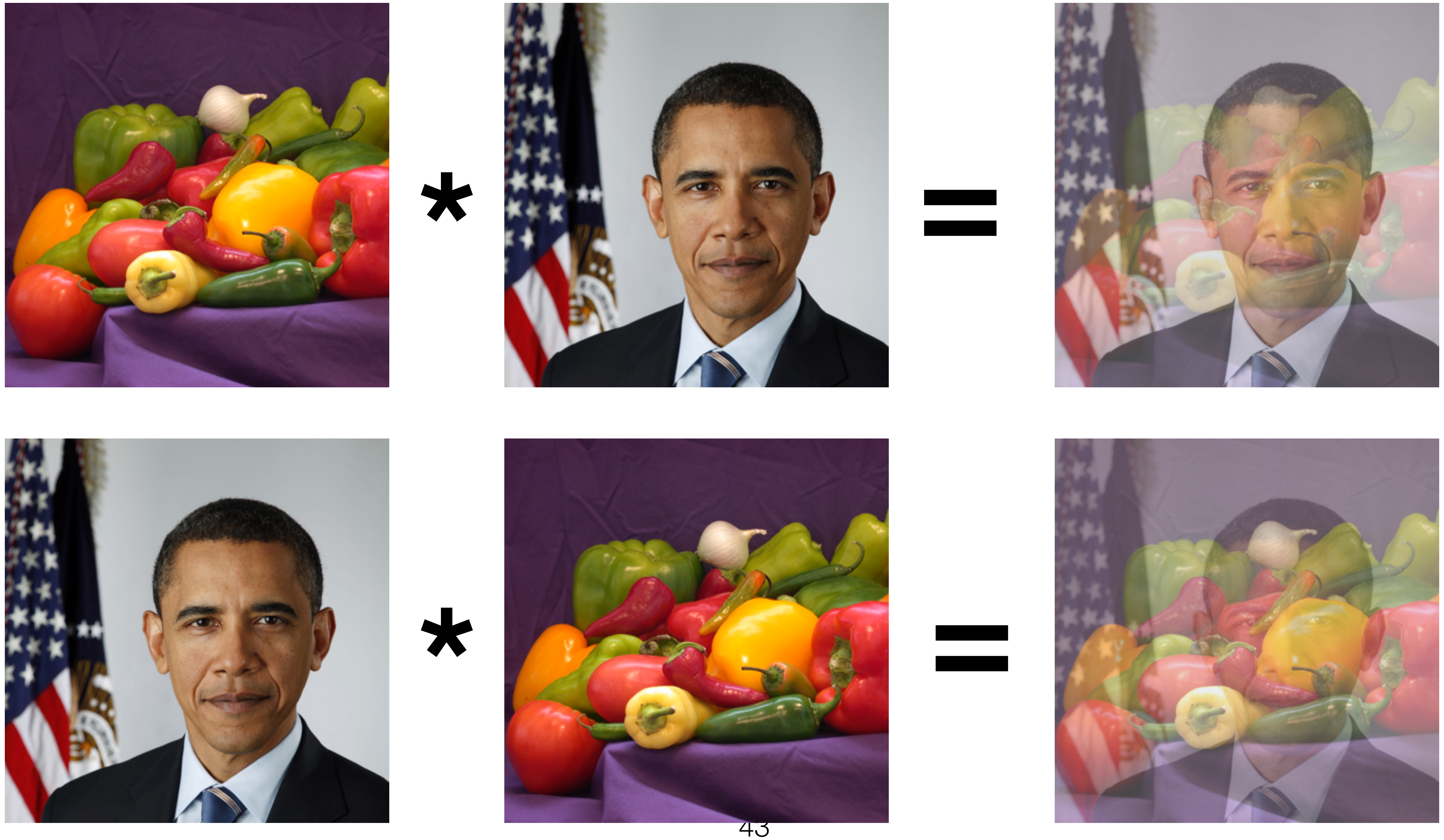
Not Permutation Invariance

 *  = 

 *  =

Permutation Invariance: Example

Not Permutation Invariance



Permutation Invariance: Architectures

Permutation Invariance: Architectures

- A standard feedforward neural net such as multilayer perceptron (MLP) is insensitive to order of elements in input vector - so it is inherently permutation insensitive

Permutation Invariance: Architectures

- A standard feedforward neural net such as multilayer perceptron (MLP) is insensitive to order of elements in input vector - so it is inherently permutation insensitive

Permutation Invariance: Architectures

- A standard feedforward neural net such as multilayer perceptron (MLP) is insensitive to order of elements in input vector - so it is inherently permutation insensitive

- However, both a Convnet and RNNs for instance make full use of input ordering - they are permutation sensitive.

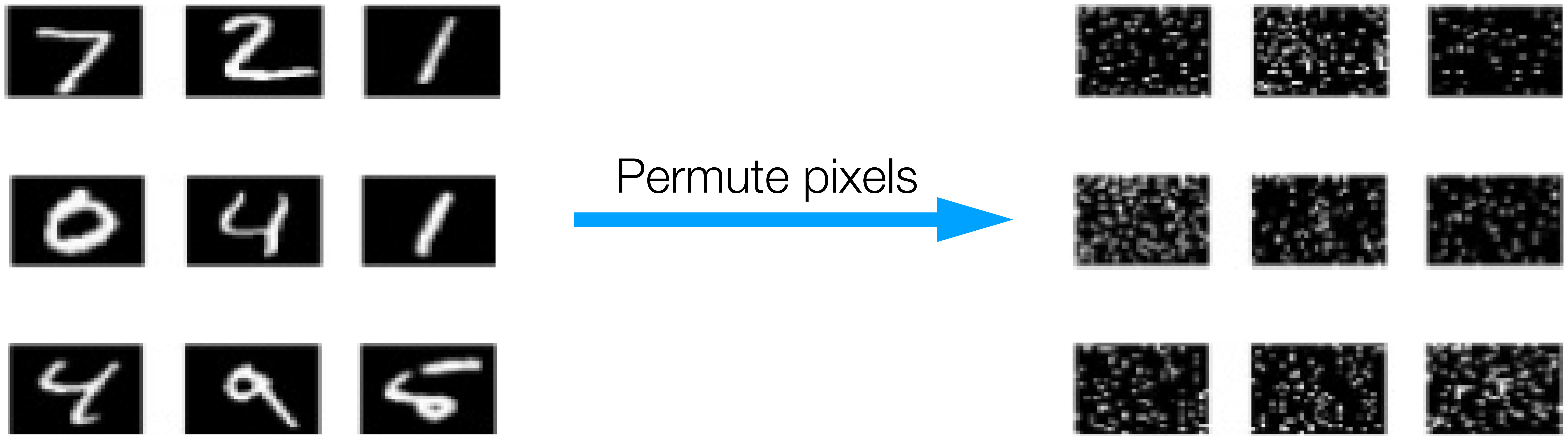
Permutation Invariance: Architectures

- A standard feedforward neural net such as multilayer perceptron (MLP) is insensitive to order of elements in input vector - so it is inherently permutation insensitive

- However, both a Convnet and RNNs for instance make full use of input ordering - they are permutation sensitive.

Permutation Invariance: Example

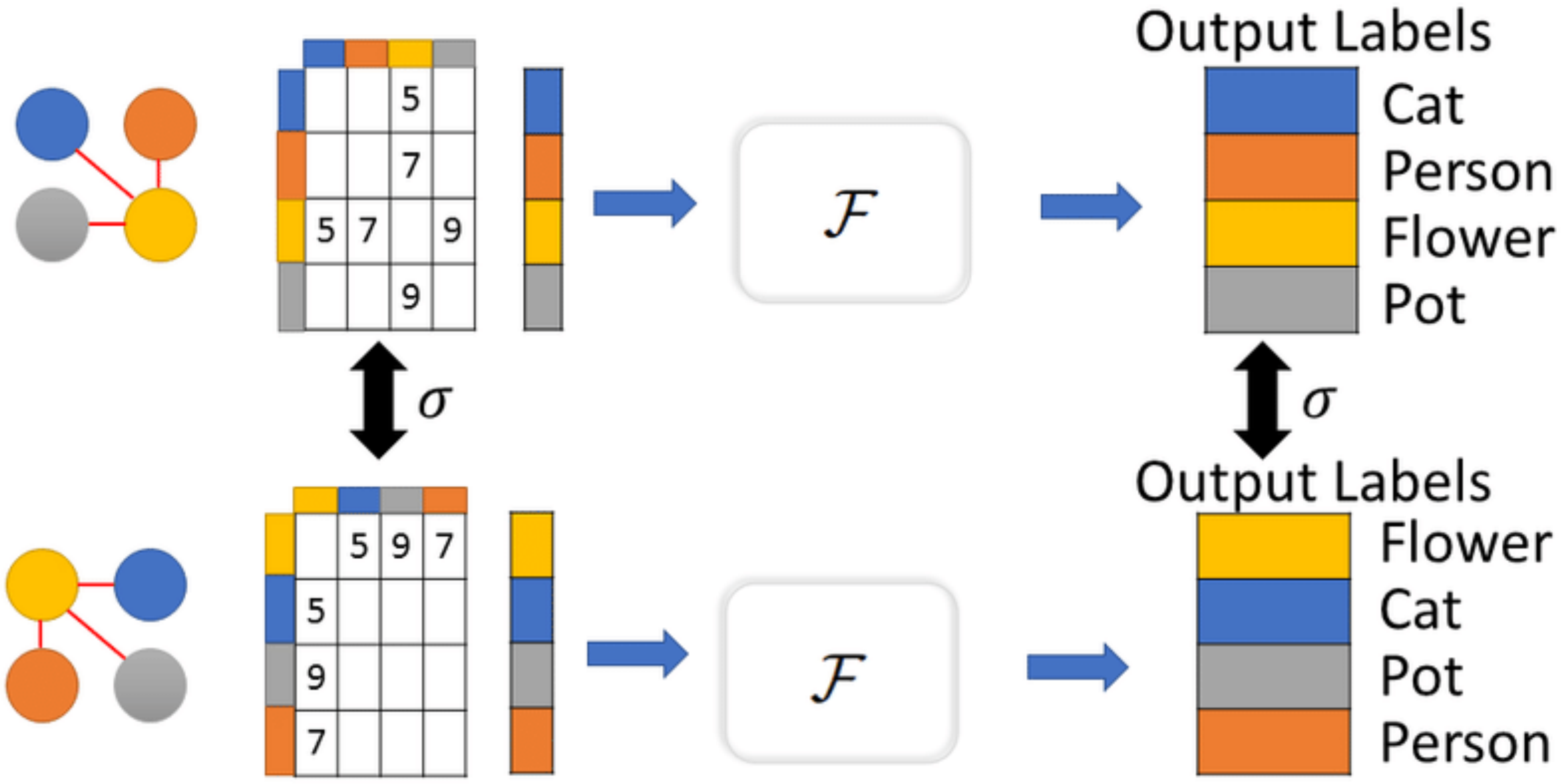
MNIST Dataset



Permuting pixels makes it difficult for humans to understand the images.

However, permutation invariant networks like MLP can detect digits irrespective of the order of pixels

Permutation Invariance: Example

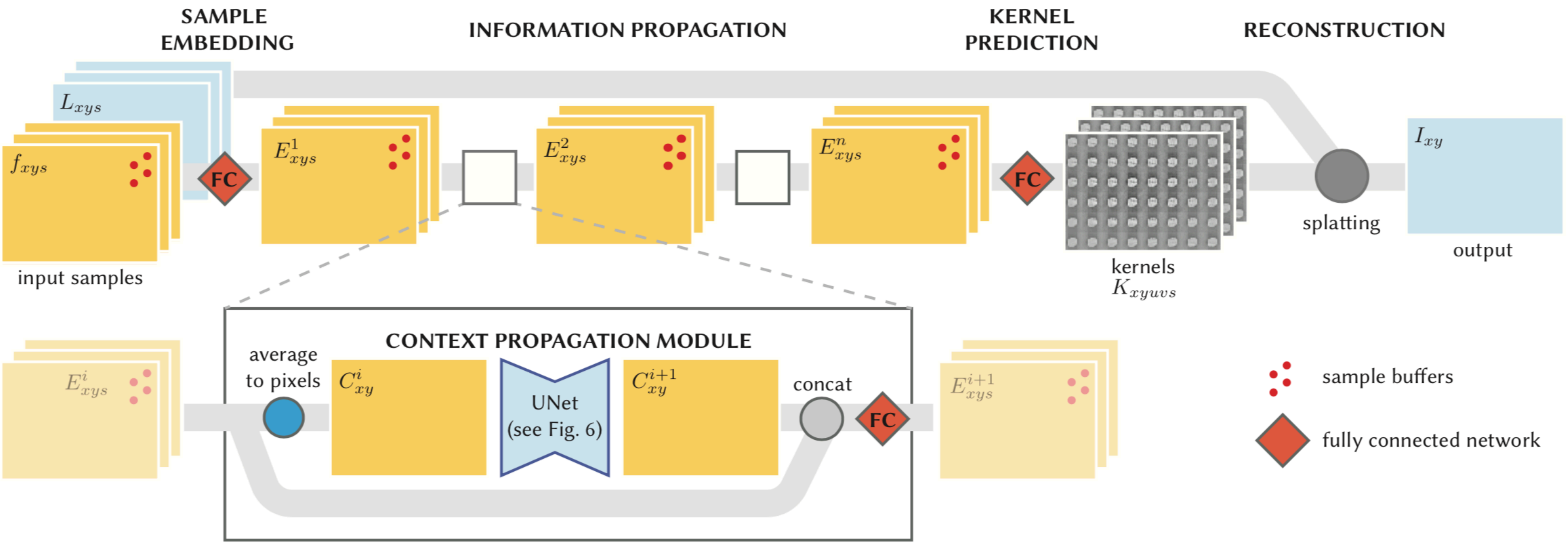


A graph labeling function F is graph permutation invariant (GPI) if permuting the names of nodes maintains the output. Herzig et al.[2018]

Permutation Invariance

- In MLPs, since each component is connected to each other, the order does not matter
- In structured convolutions, the order matters and therefore, it is not permutation invariant.

Proposed Network Architecture



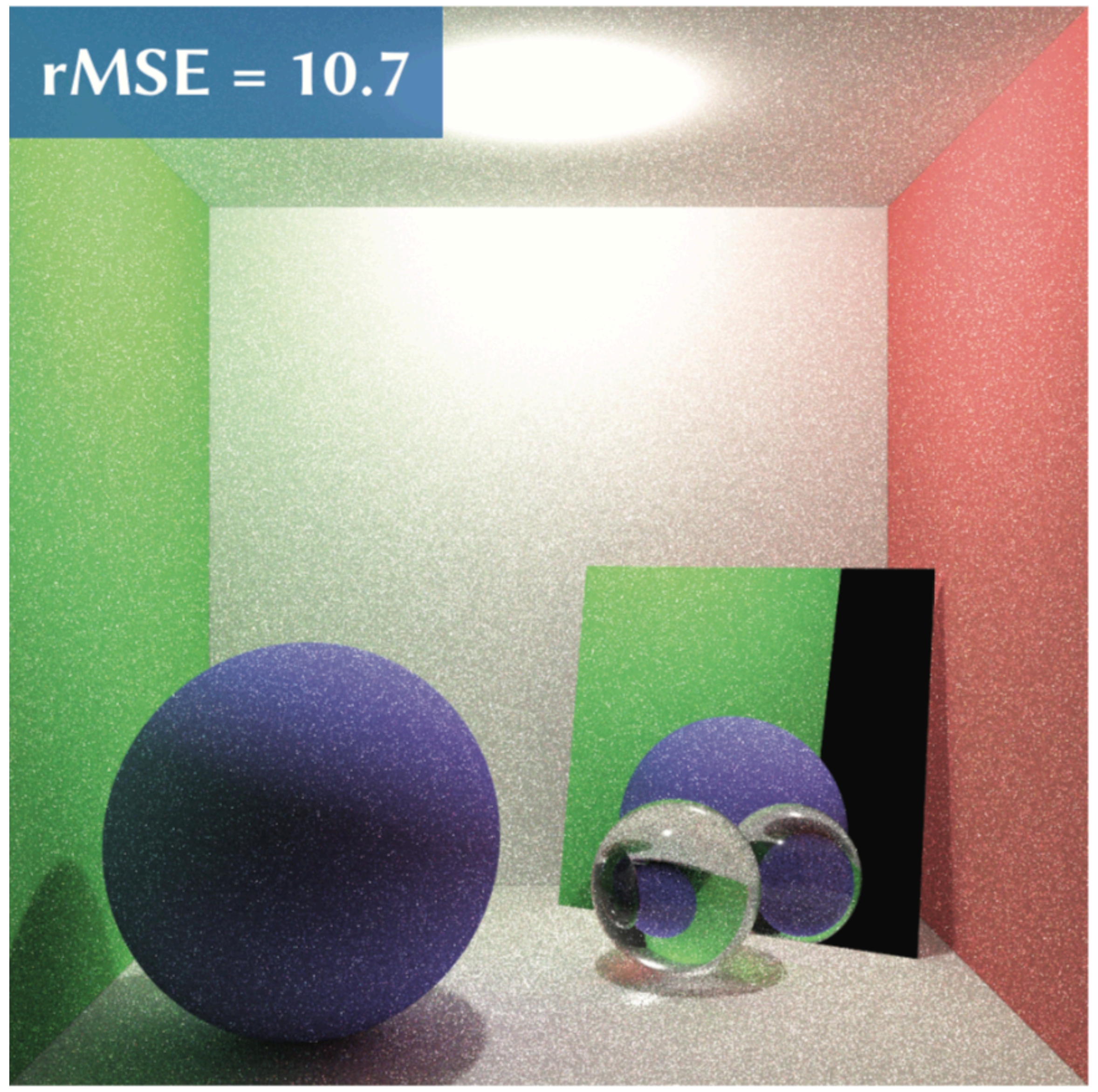
Dataset and Training Procedure

Procedurely generated dataset: 300,000 renderings with 128x128 resolution

Also generated input buffer (4, 32 spp), but this time also maintained auxiliary features

Reference was generated for 4096 samples

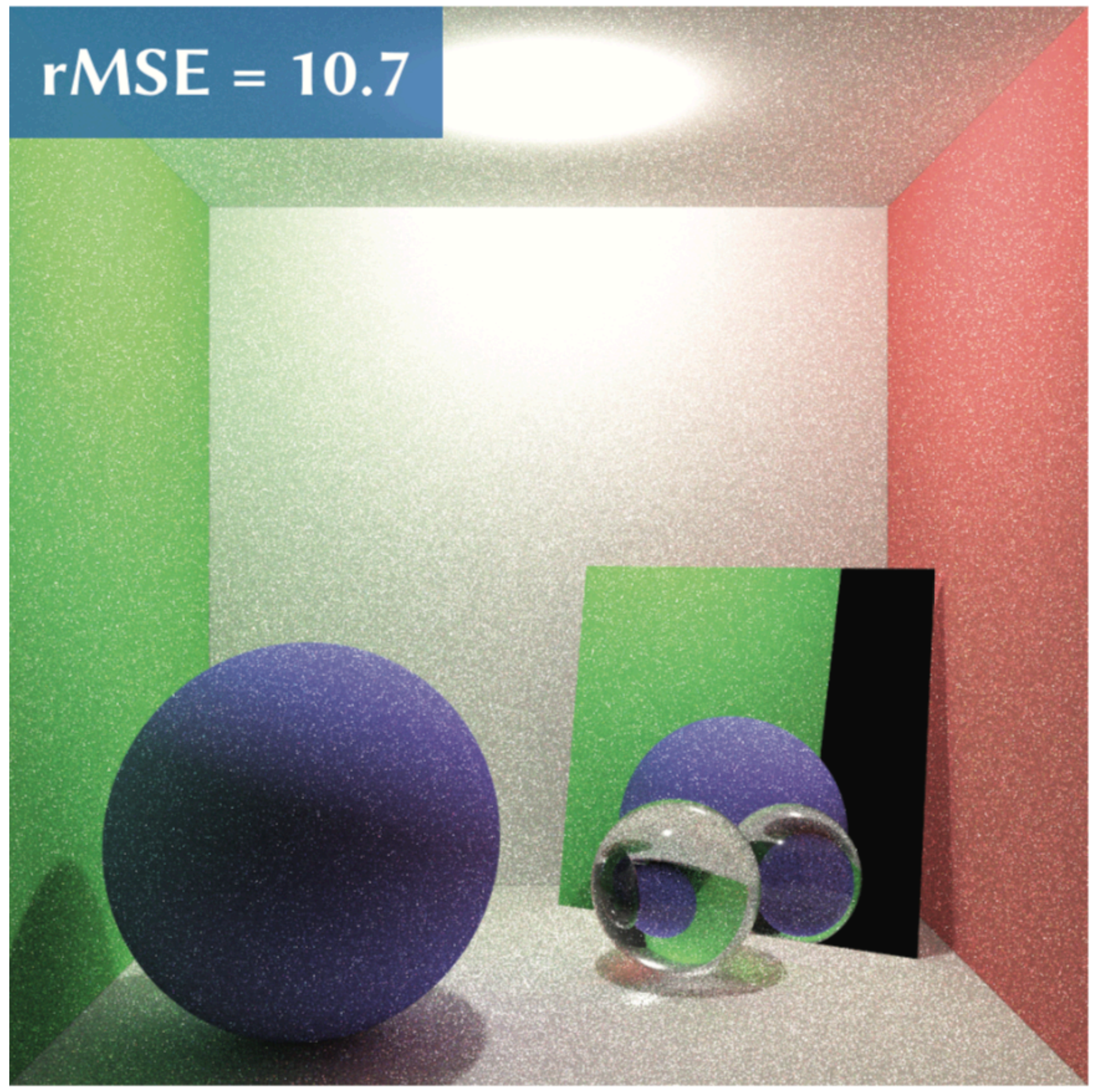
Splat vs Gather



Input

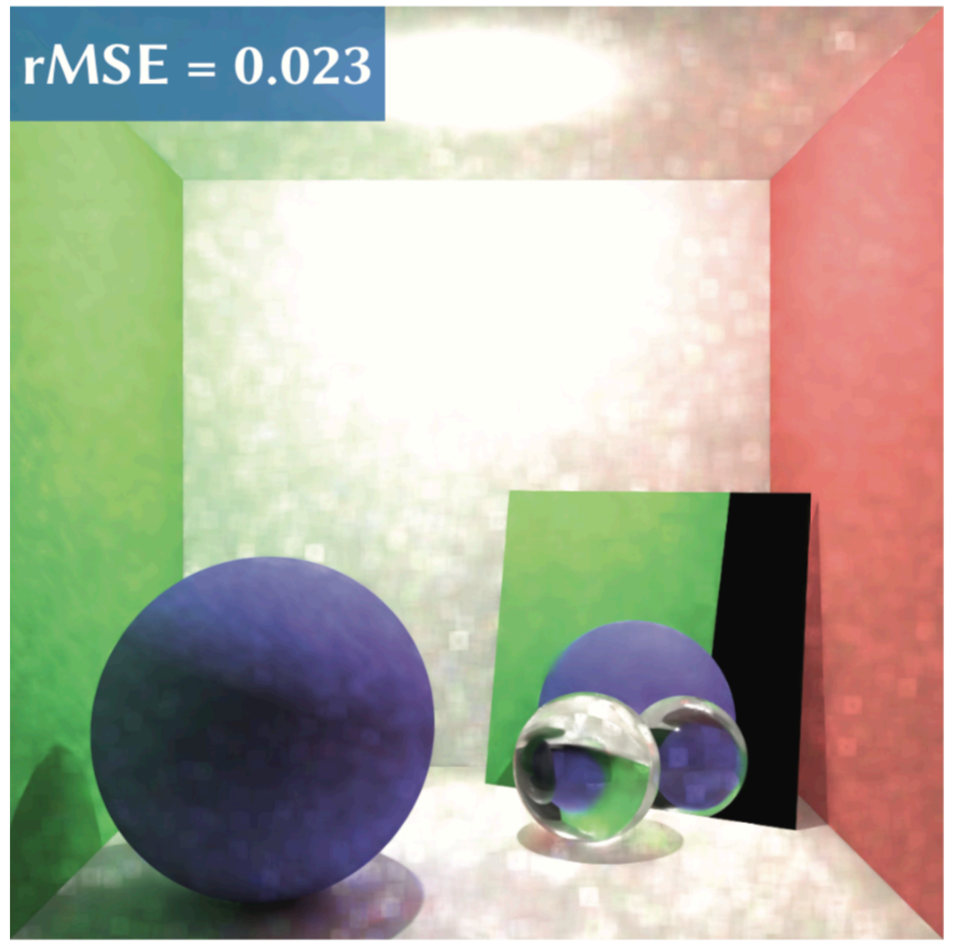
Reference

Splat vs Gather



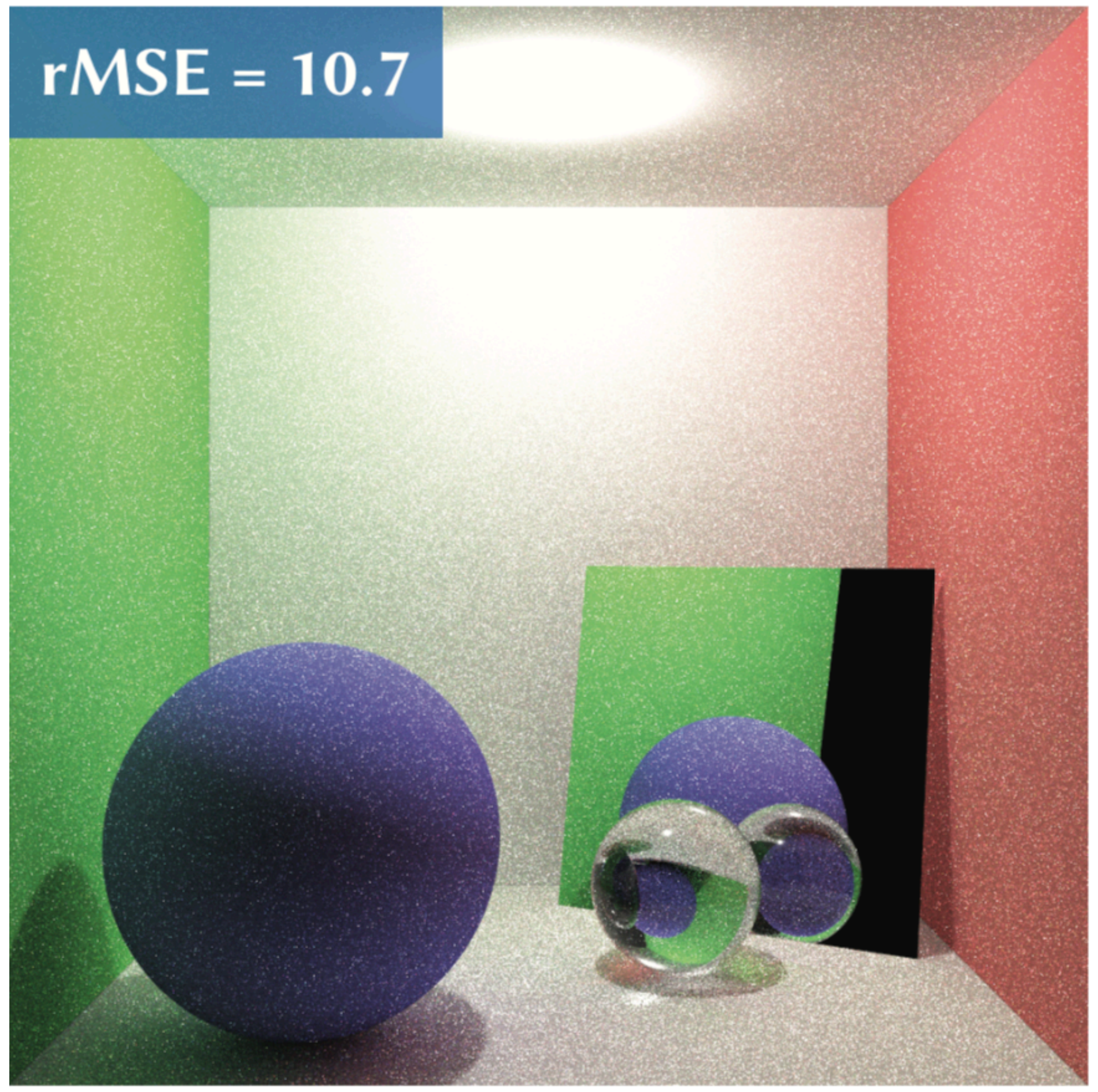
Input

per sample gather



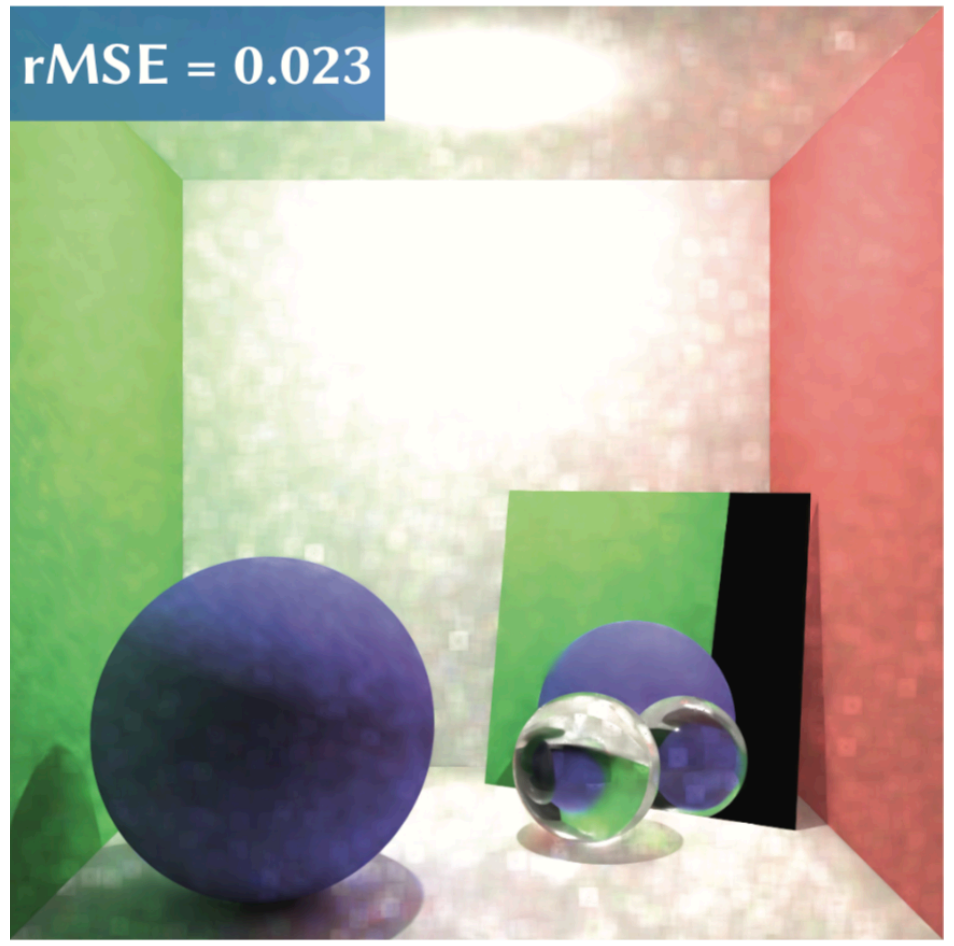
Reference

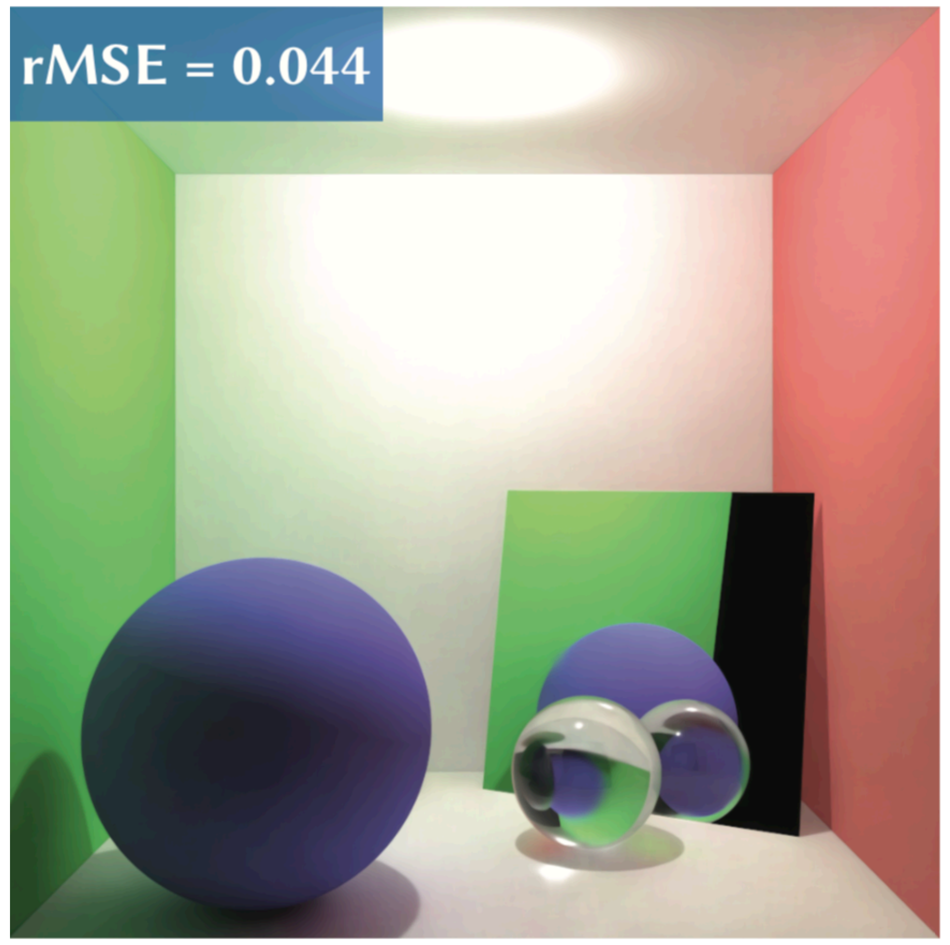
Splat vs Gather



Input

per sample gather

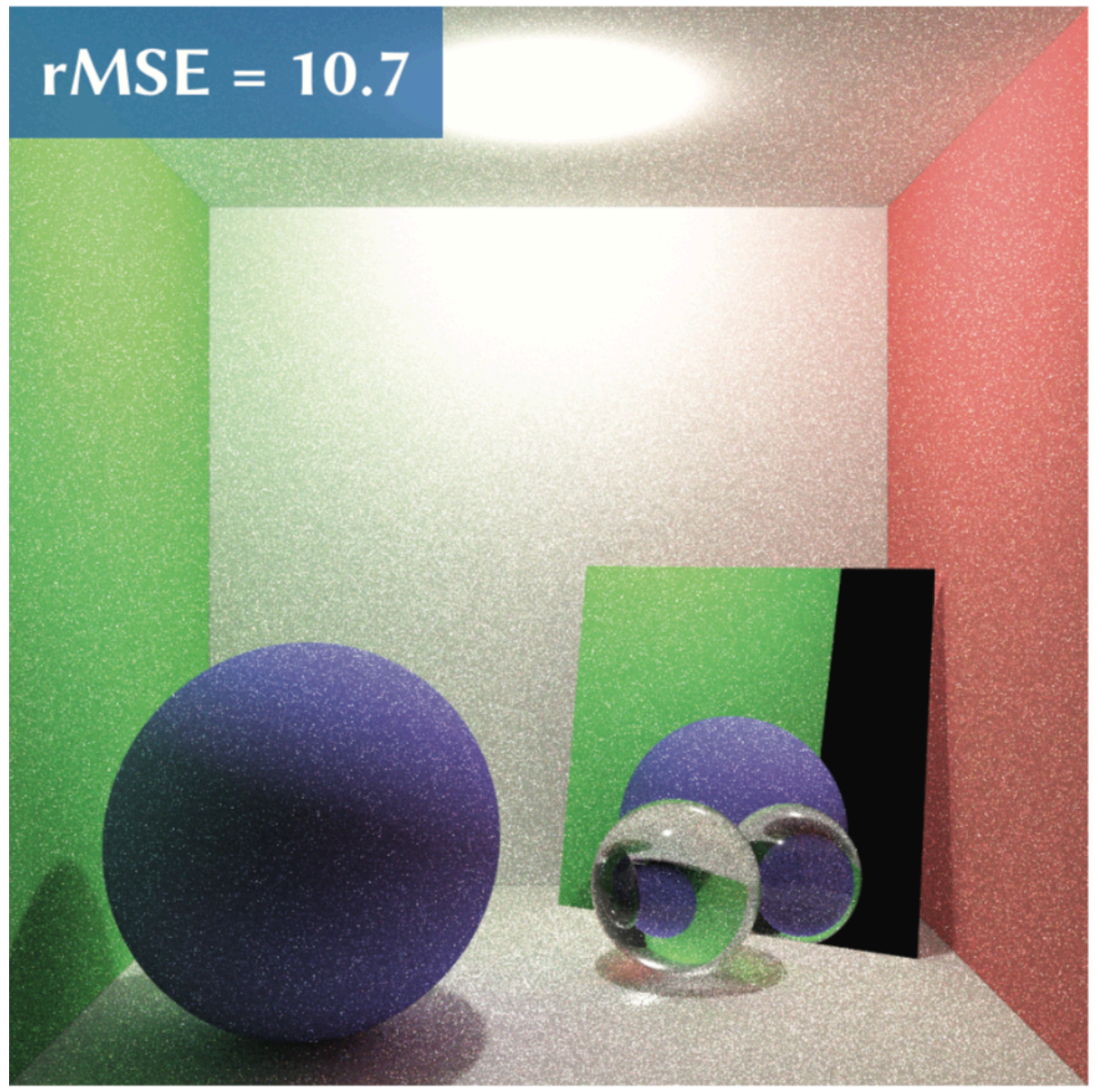




per sample splat

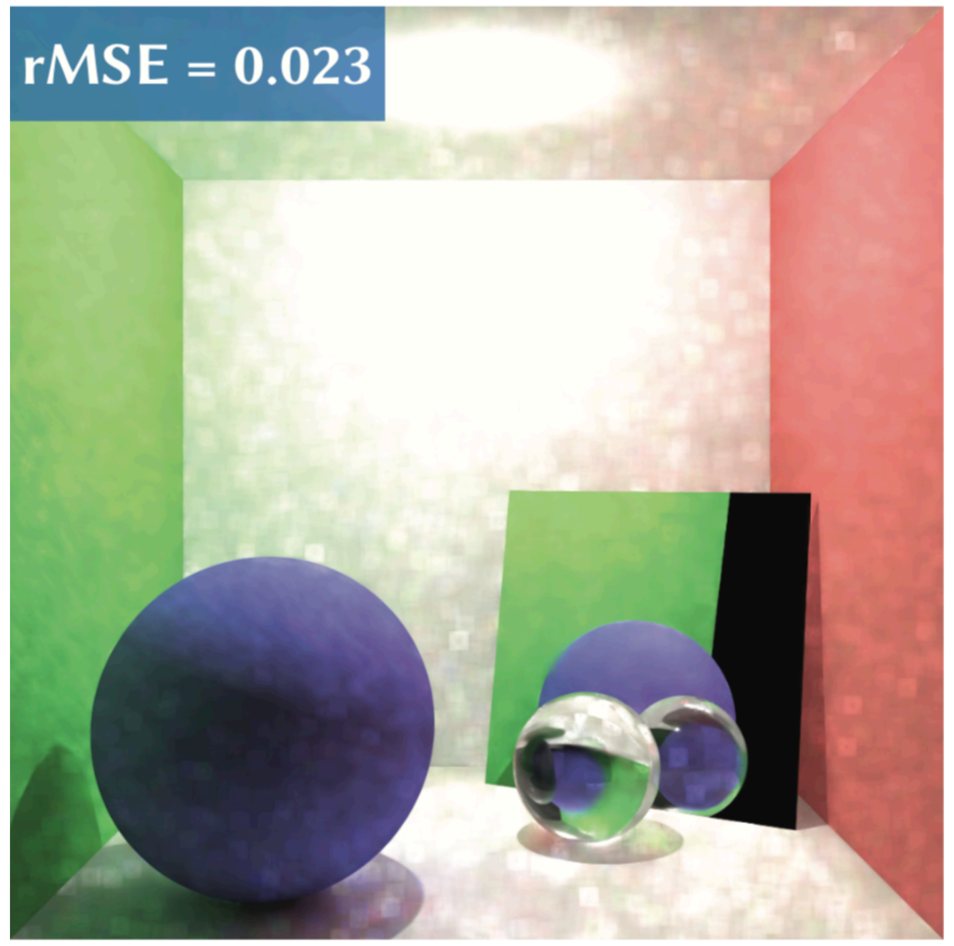
Reference

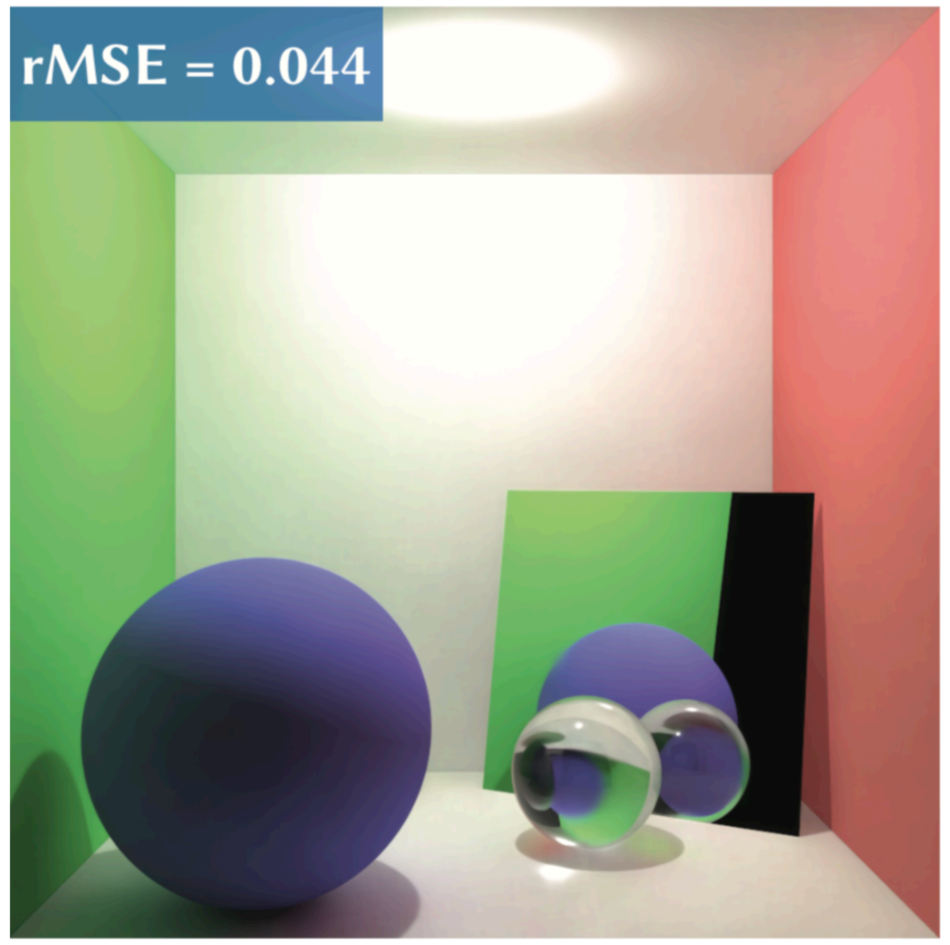
Splat vs Gather



Input

per sample gather

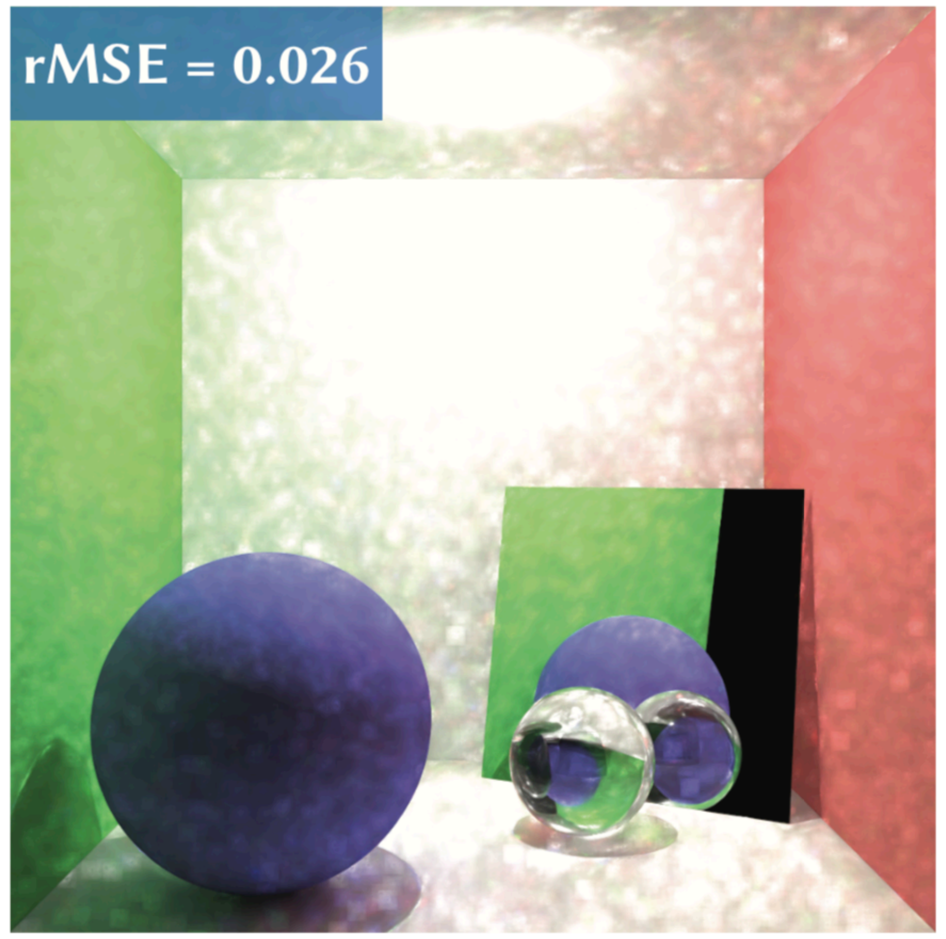




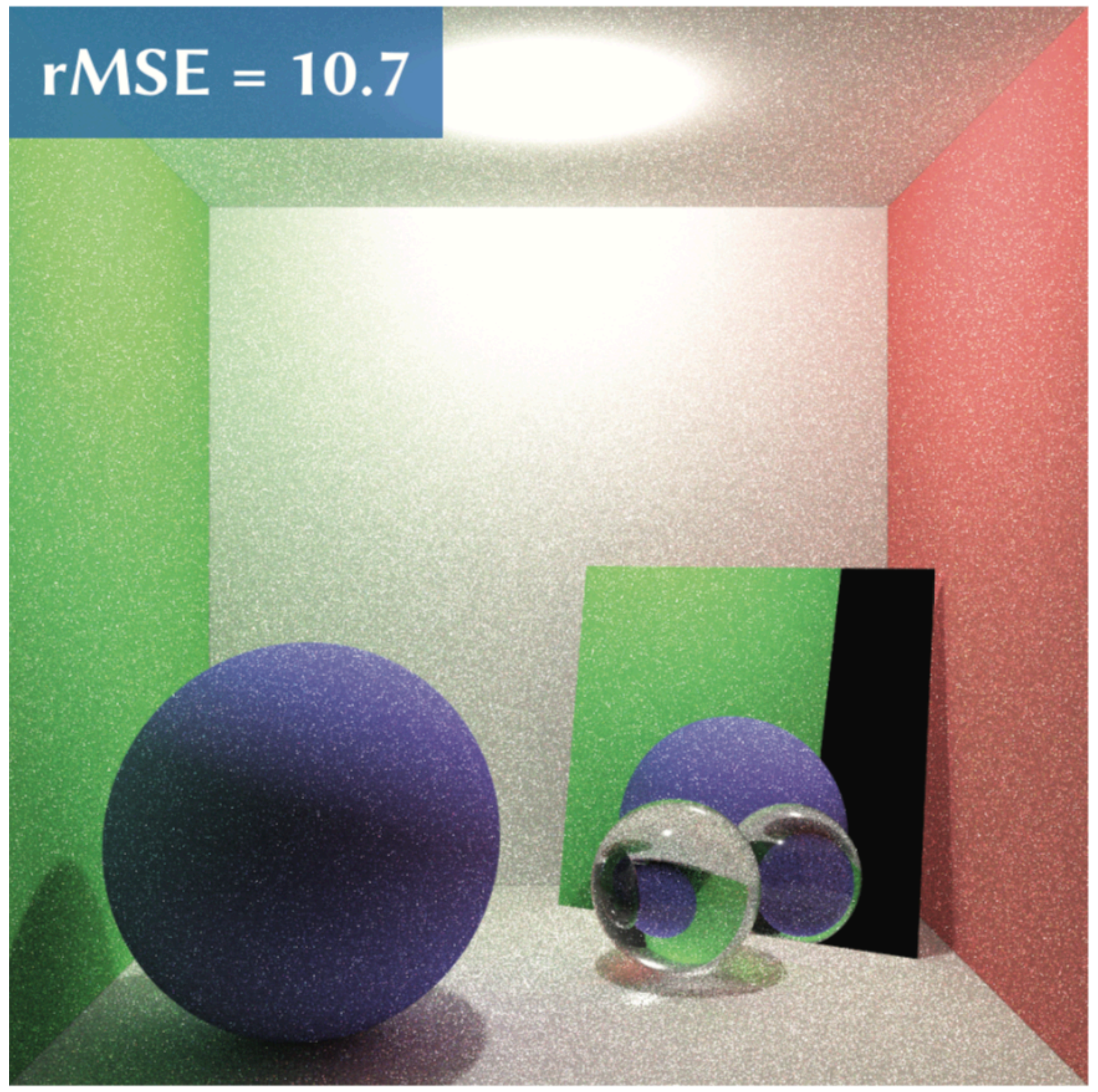
per sample splat

Reference

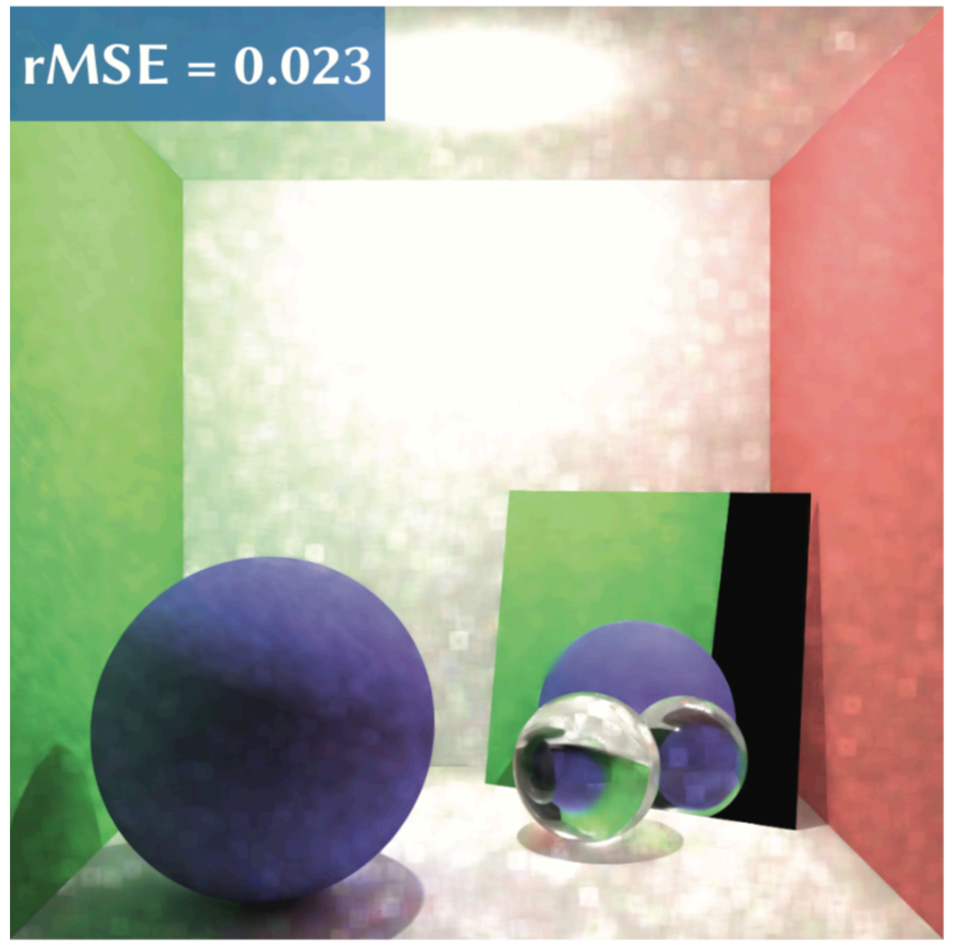
per pixel gather



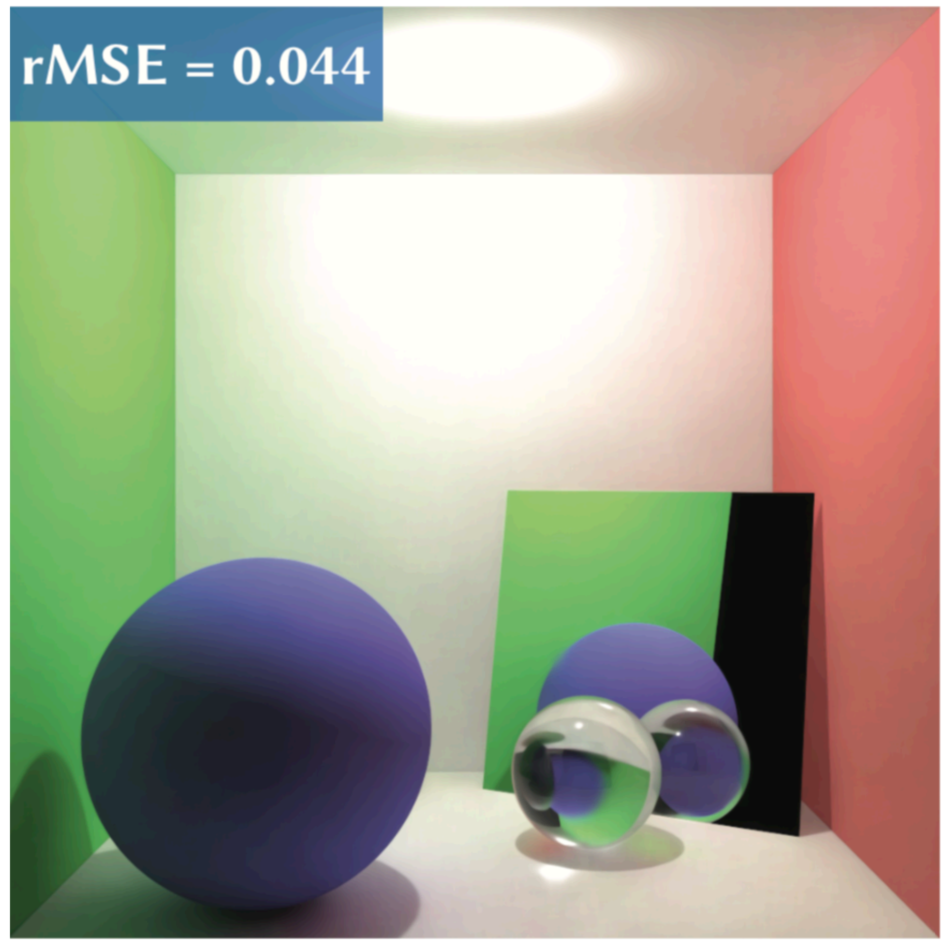
Splat vs Gather



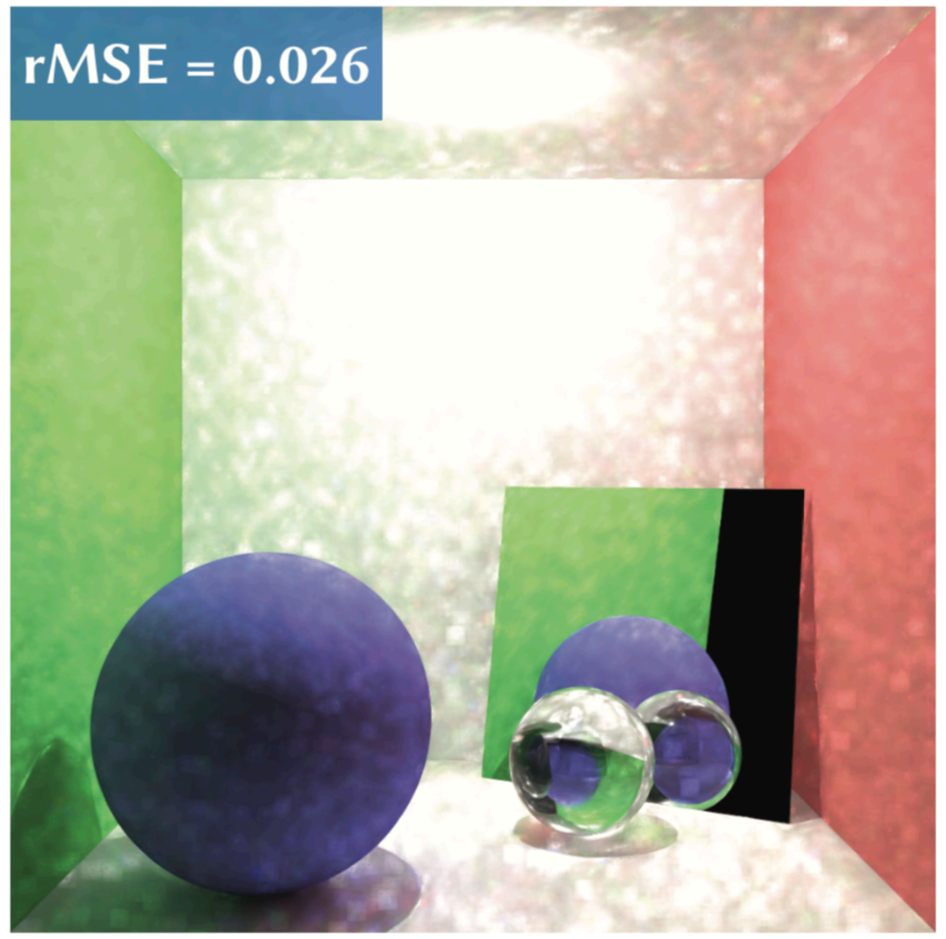
Input



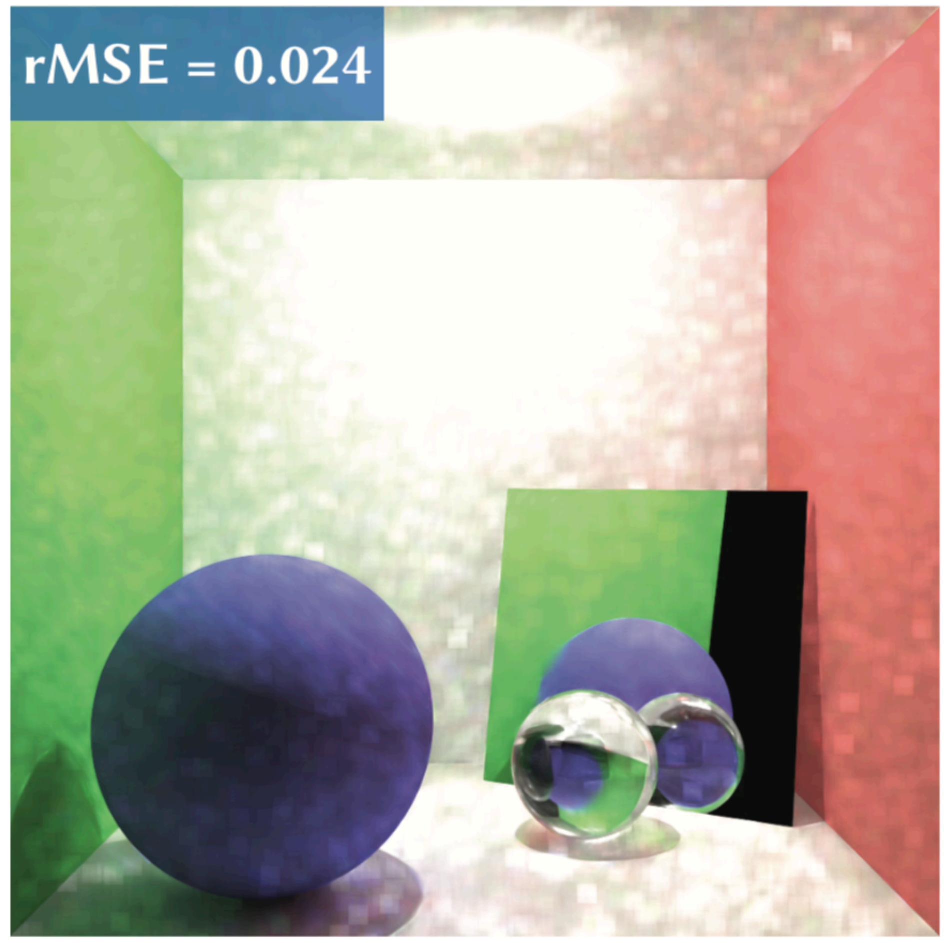
per sample gather



per sample splat



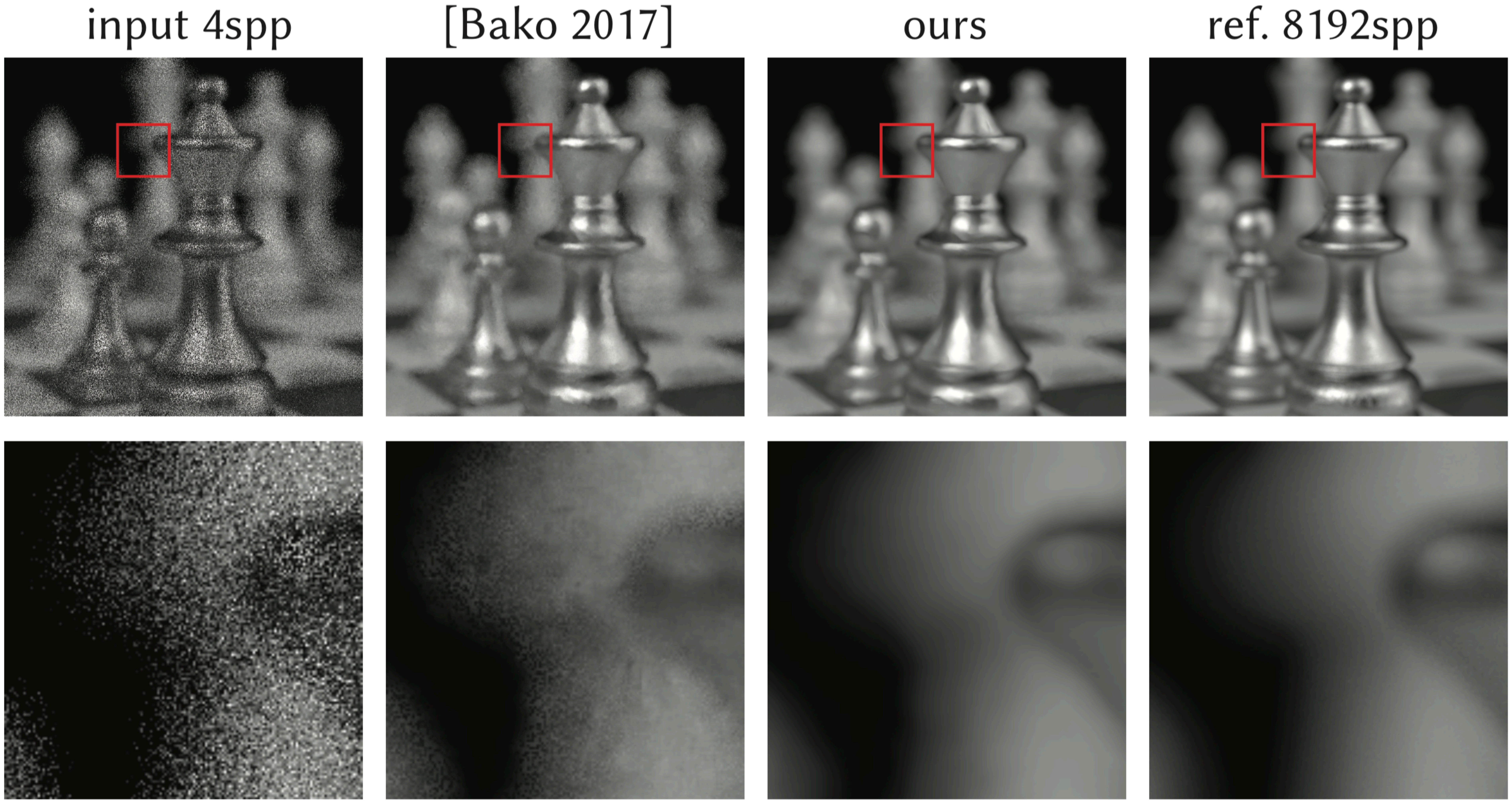
per pixel gather



per pixel splat

Reference

Results

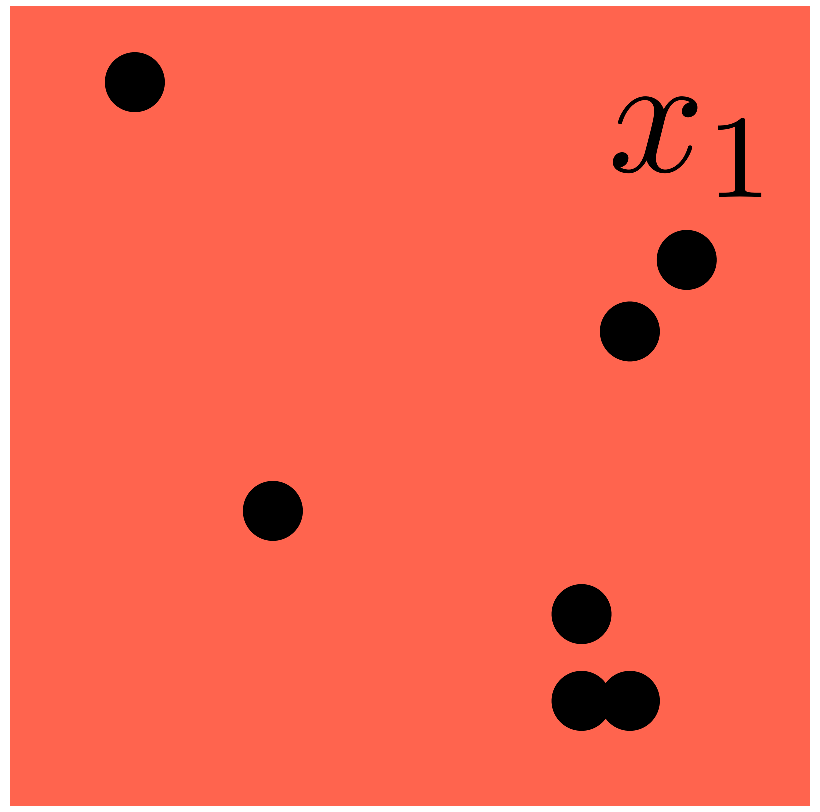


Network Architecture Comparisons

(Deep) Convolutional Neural Networks

Based on Convolutional Neural Networks

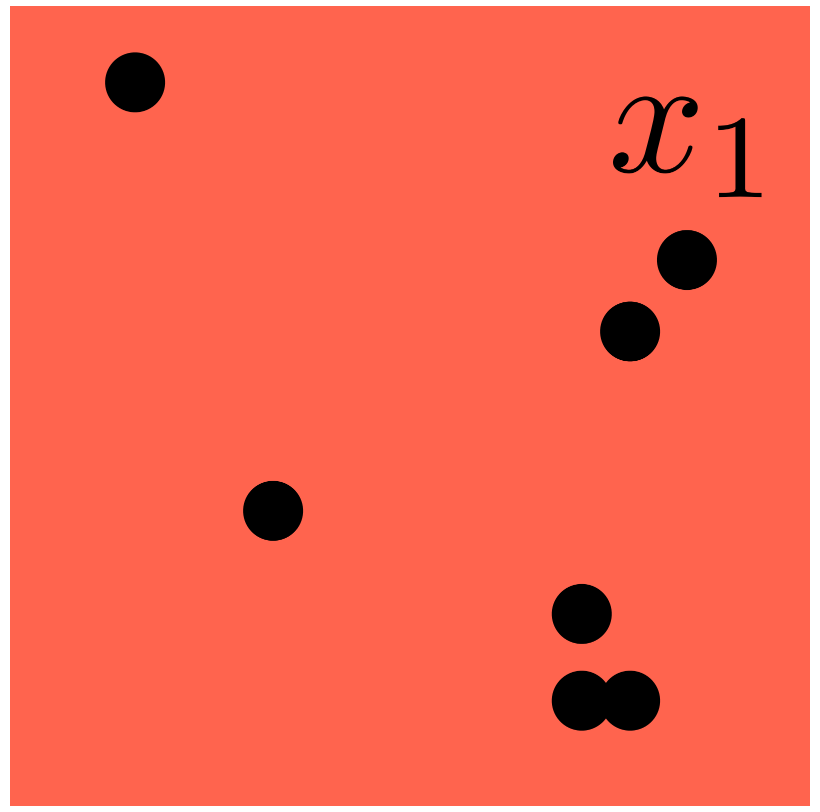
Unstructured data



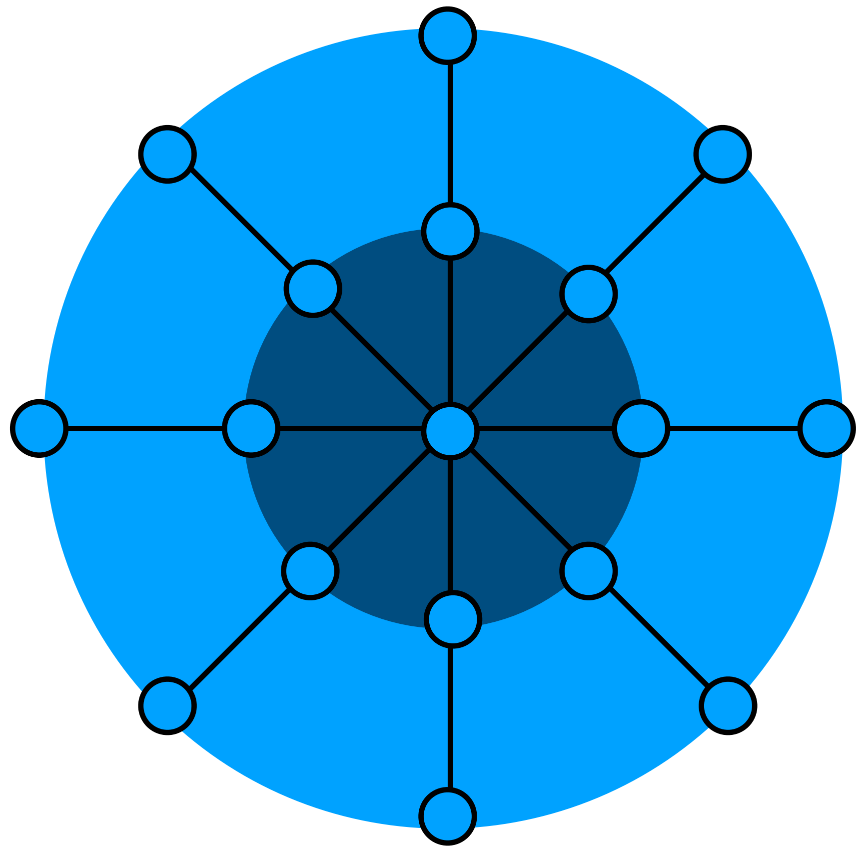
N number of point samples

Based on Convolutional Neural Networks

Unstructured data

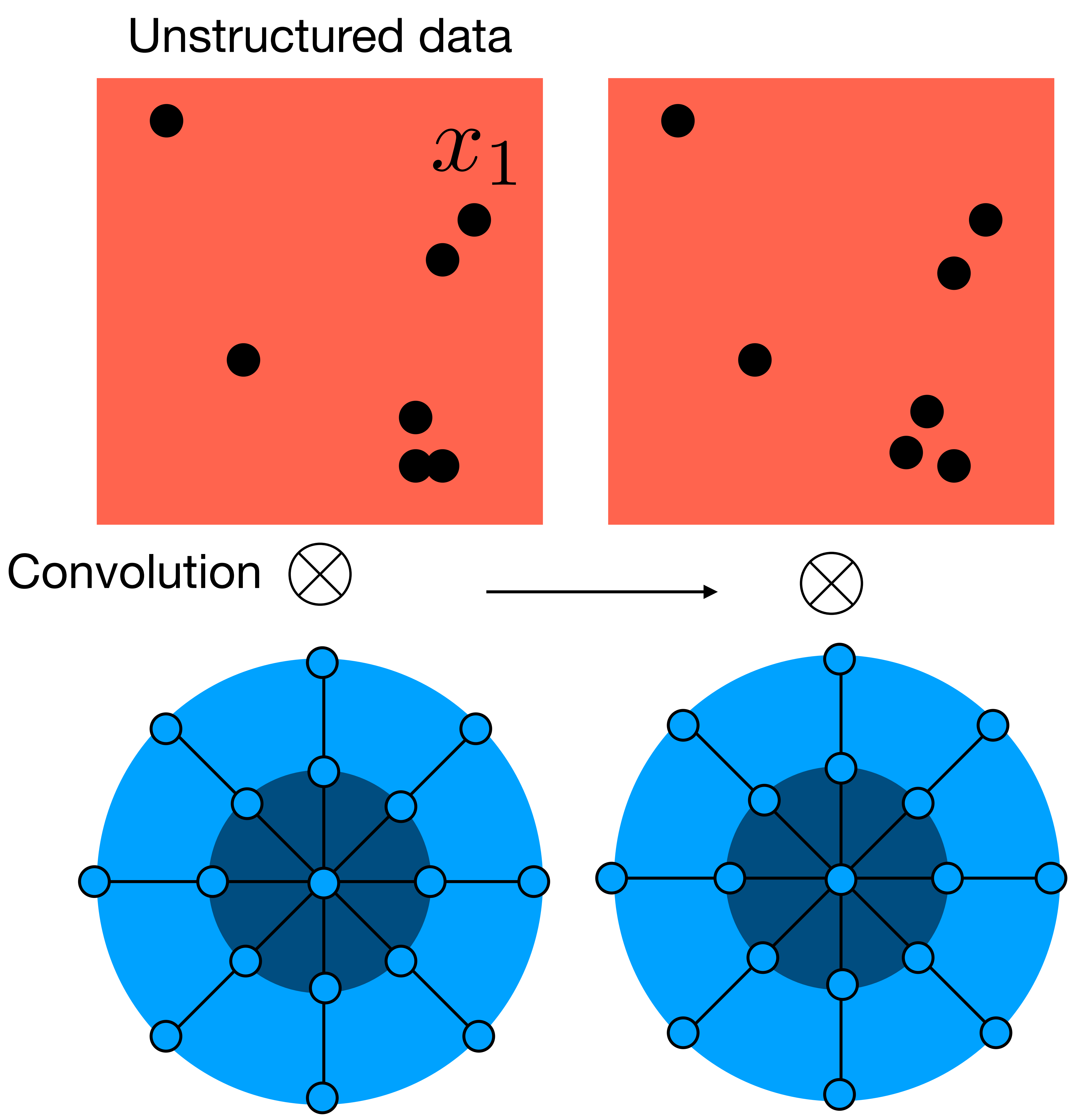


Convolution $\otimes$



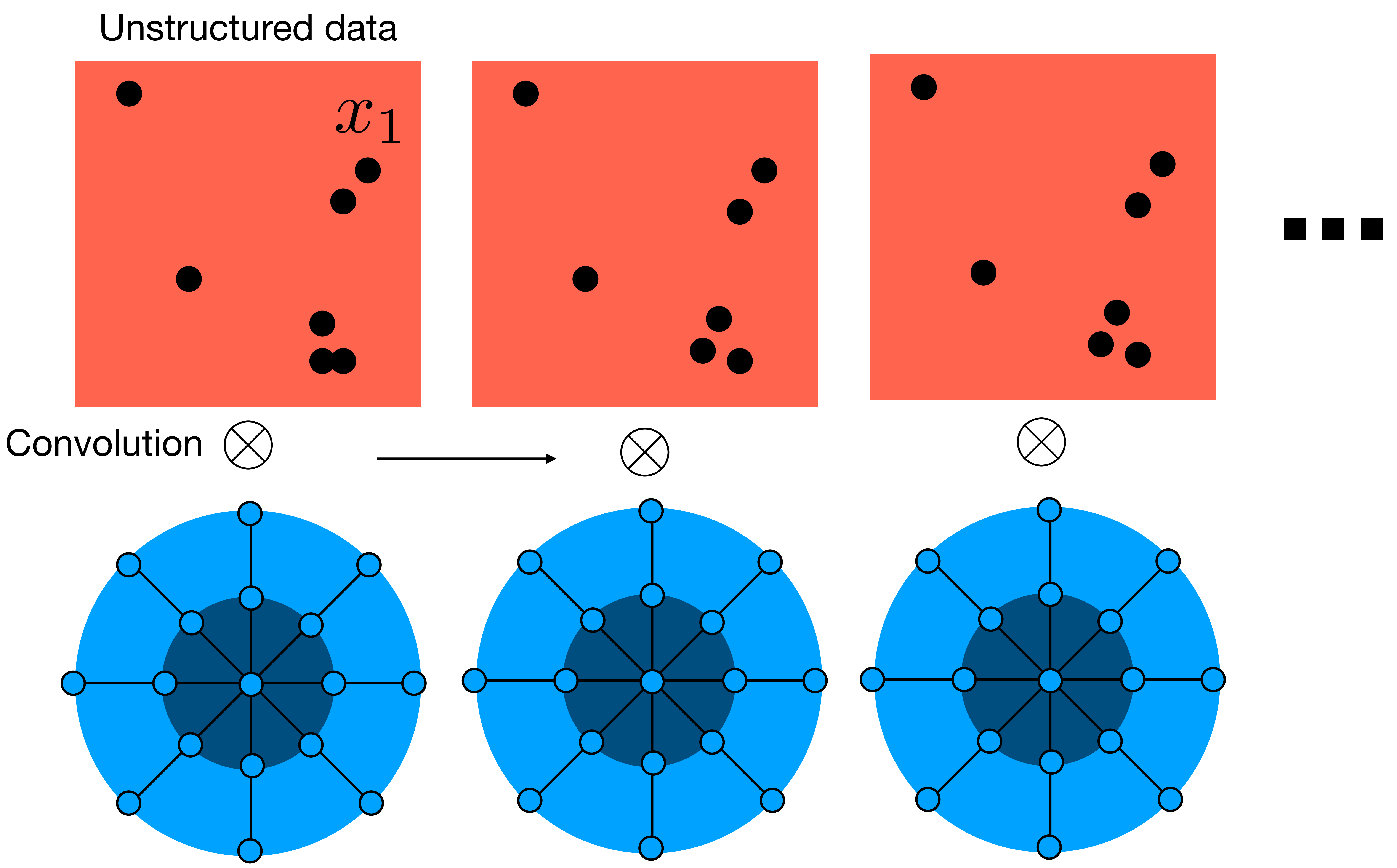
N number of point samples

Based on Convolutional Neural Networks



N number of point samples

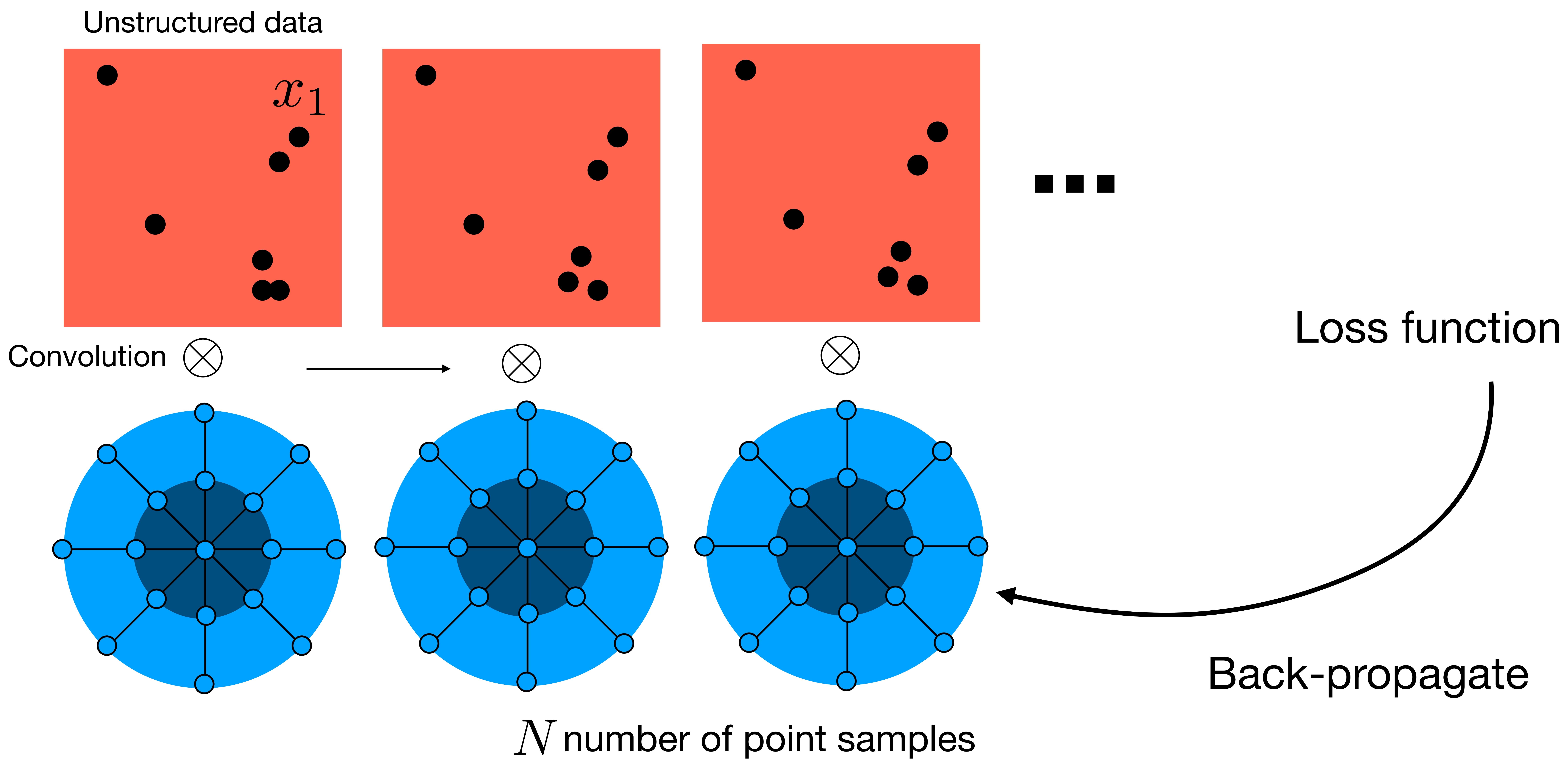
Based on Convolutional Neural Networks



Loss function

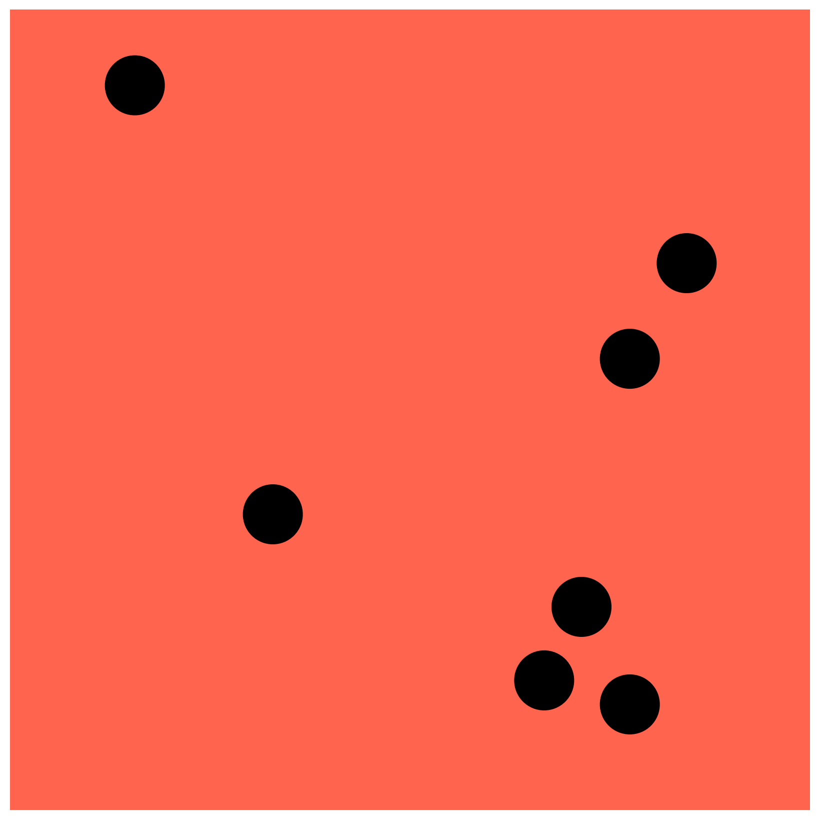
N number of point samples

Based on Convolutional Neural Networks

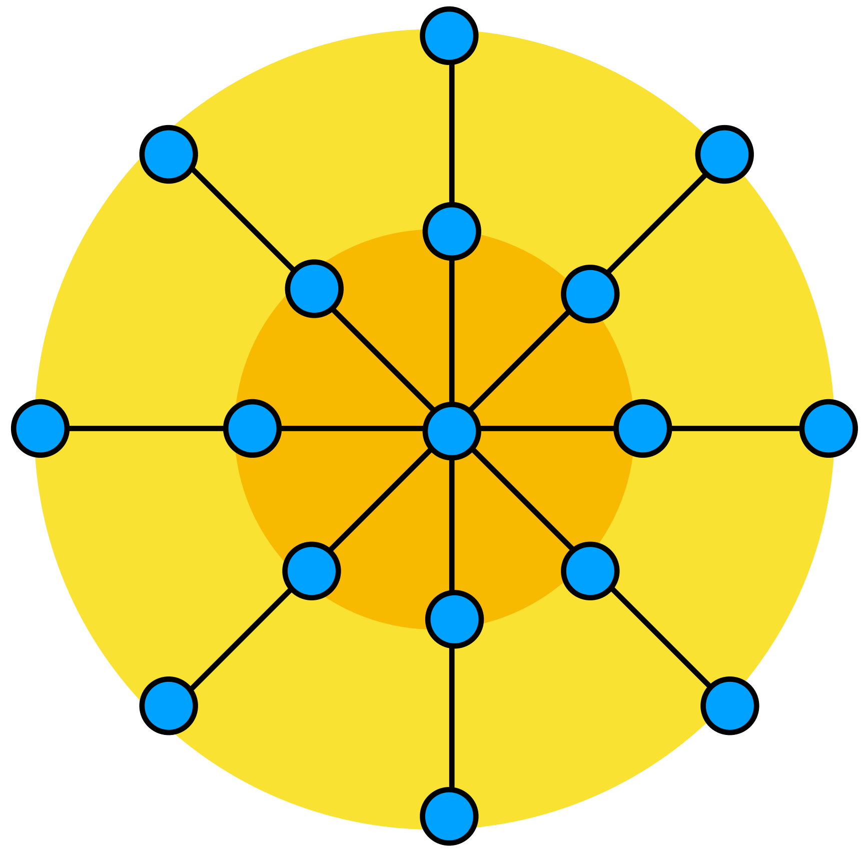


Based on Convolutional Neural Networks

Unstructured data

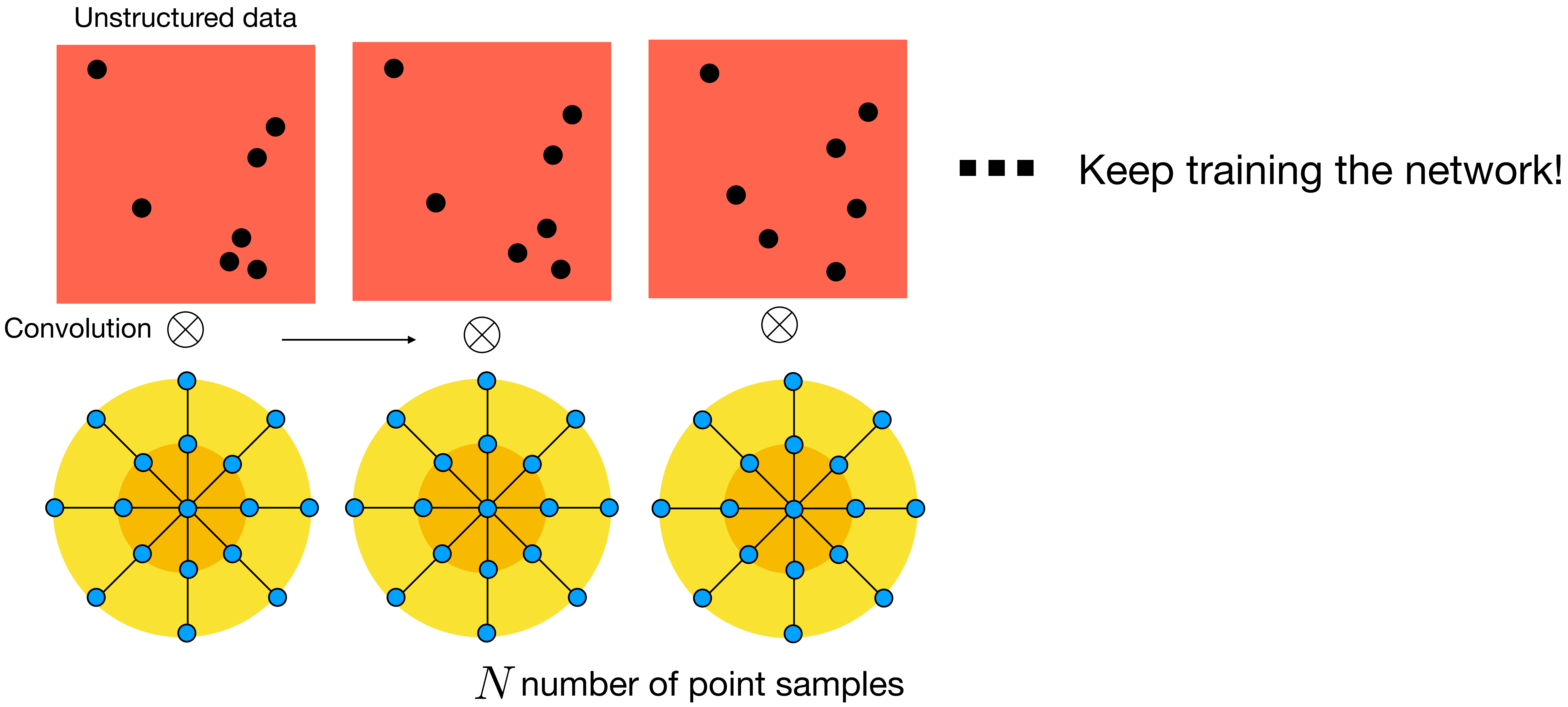


Convolution $\otimes$ $\longrightarrow$

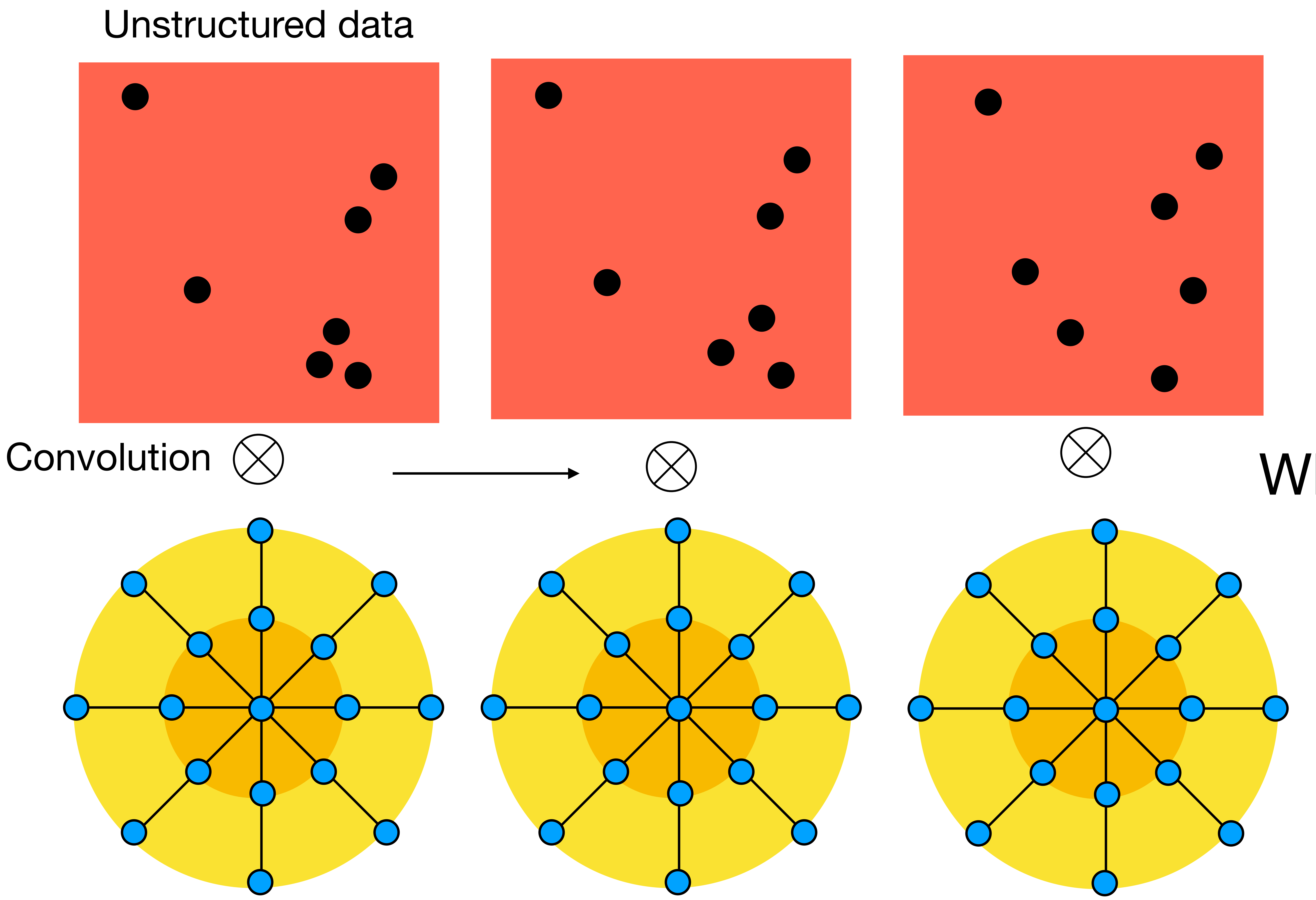


N number of point samples

Based on Convolutional Neural Networks



Based on Convolutional Neural Networks



■ ■ ■ Keep training the network!

Which Loss function can we use?

N number of point samples

Spectral Loss Function

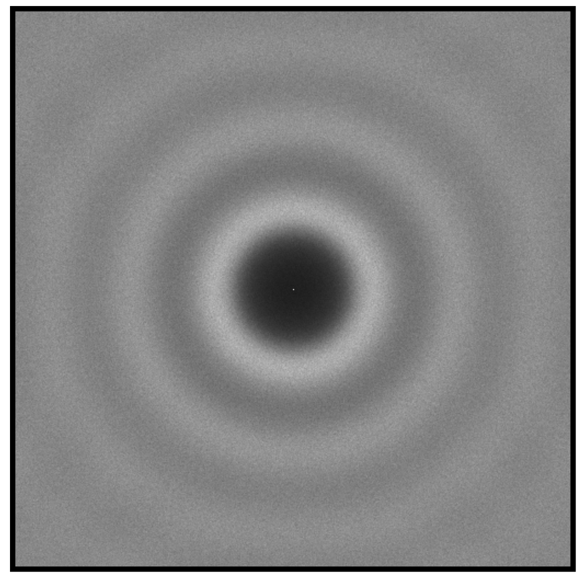
Spectral Loss at i-th training iteration

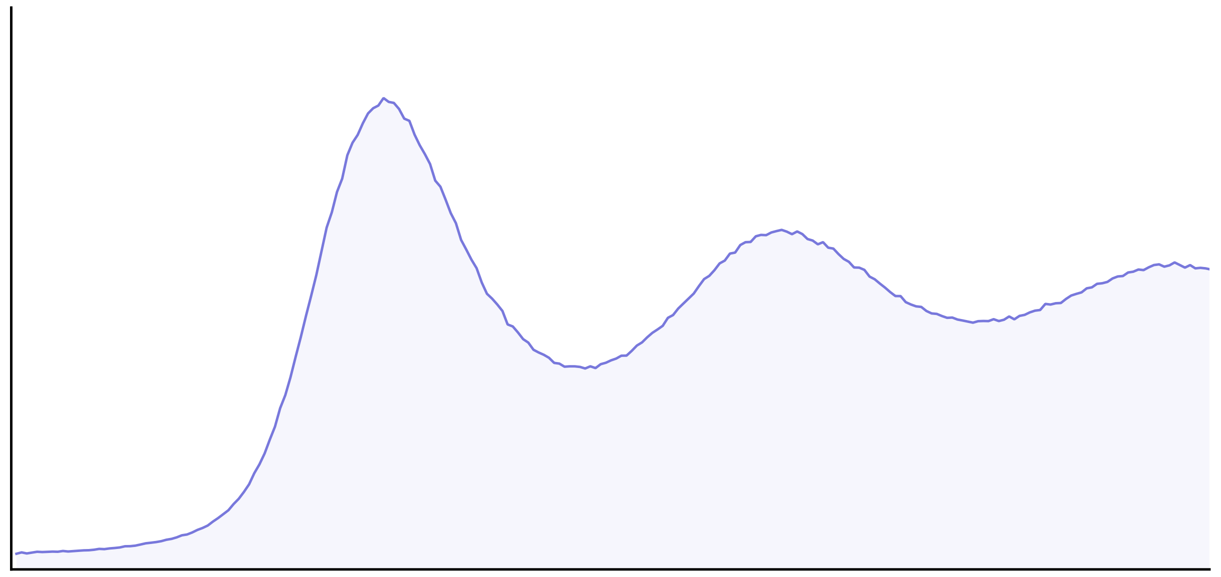
$$L_{\mathrm{spectral}} = ||\langle \mathcal{P}_i(\nu) \rangle - \langle \mathcal{P}(\nu) \rangle||^2$$

Spectral Loss Function

Spectral Loss at i-th training iteration

$$L_{\text{spectral}} = ||\langle \mathcal{P}_i(\nu) \rangle - \langle \mathcal{P}(\nu) \rangle||^2$$



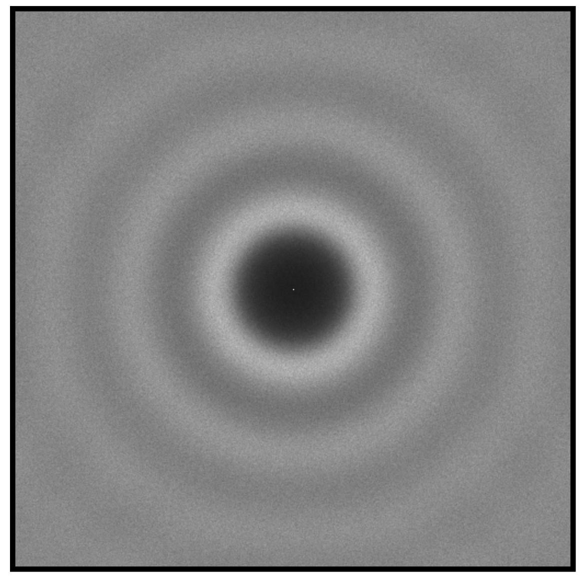


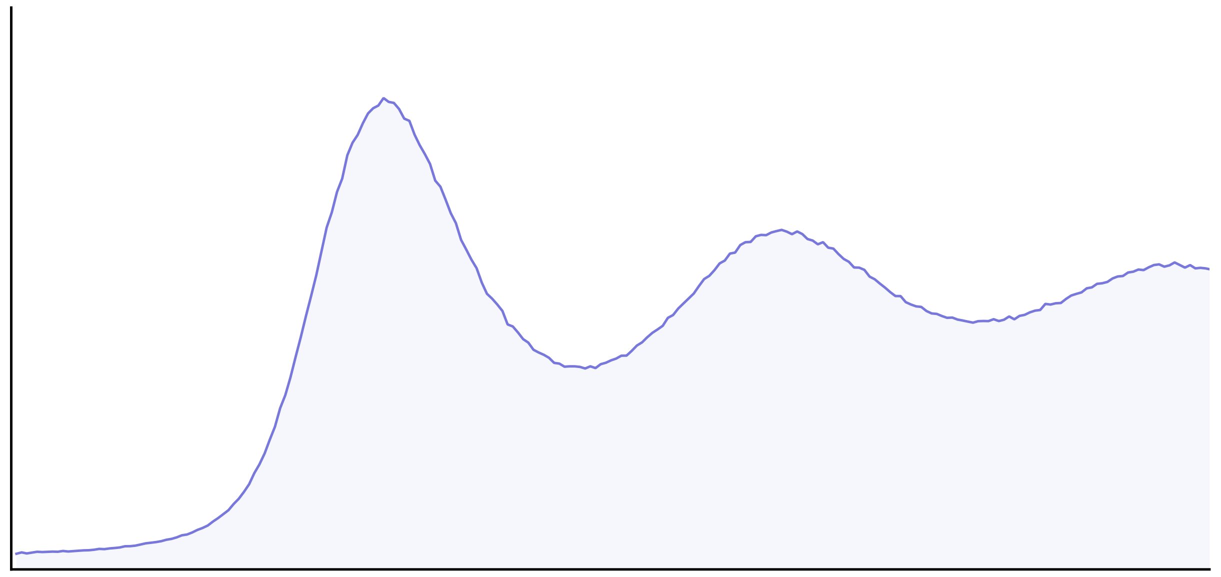
Radially averaged power

Spectral Loss Function

Spectral Loss at $i\text{-th}$ training iteration

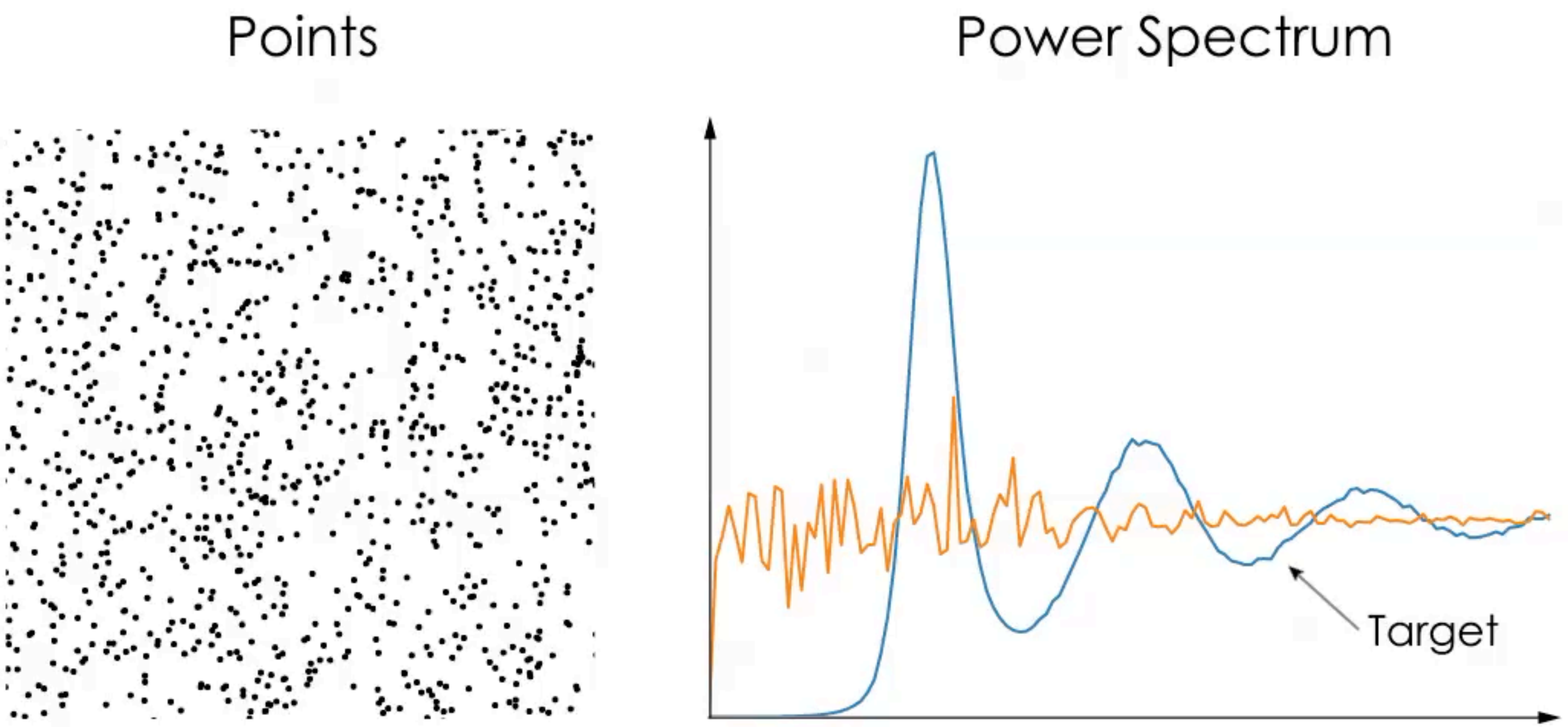
$$L_{\text{spectral}} = ||\underline{\langle \mathcal{P}_i(\nu) \rangle} - \langle \mathcal{P}(\nu) \rangle||^2$$





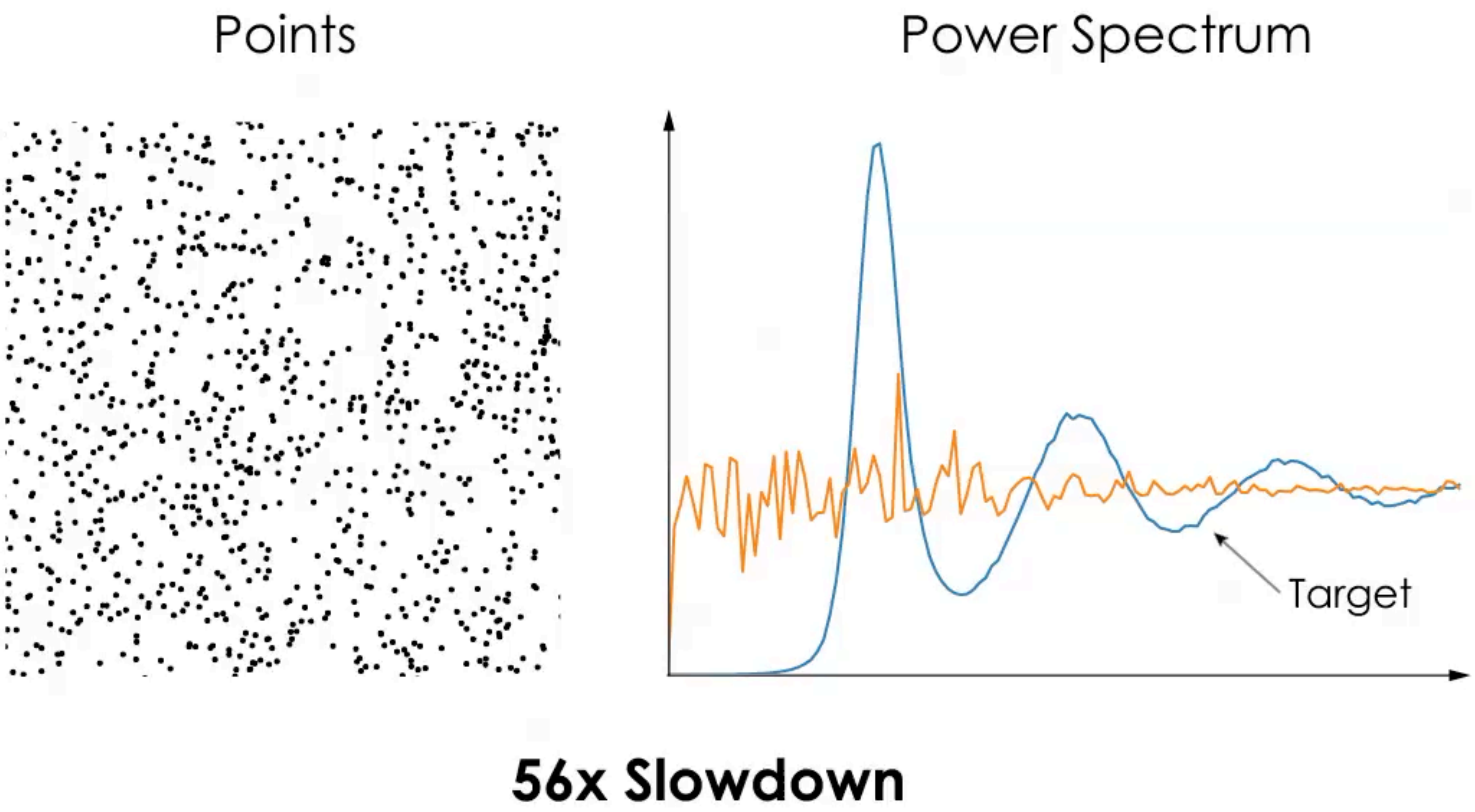
Radially averaged power

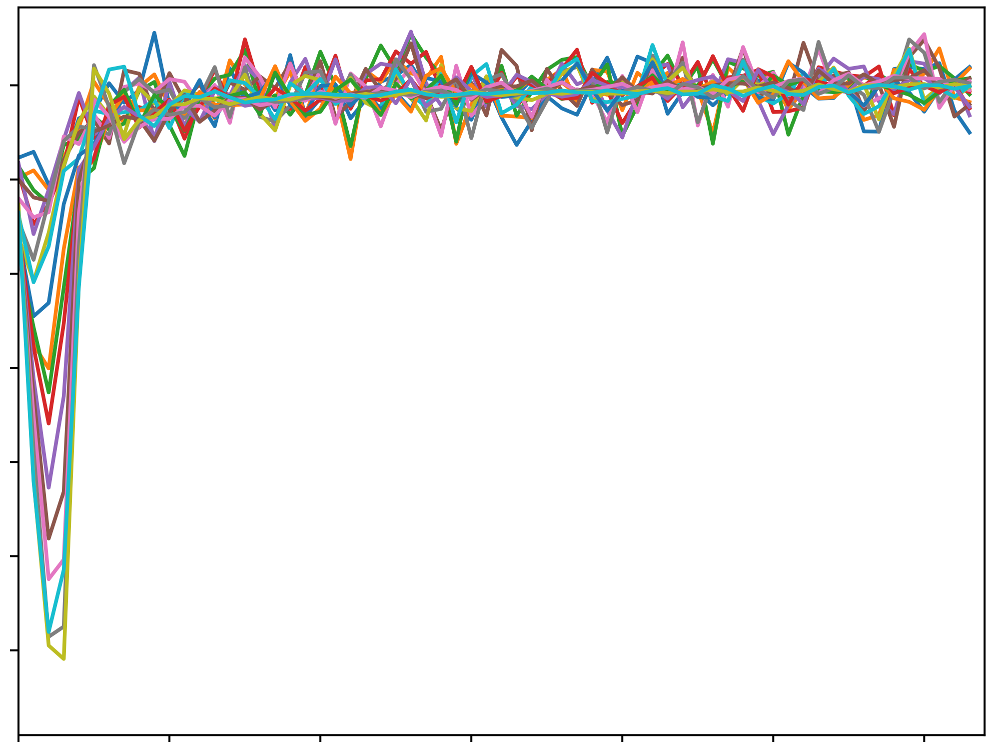
Training Process



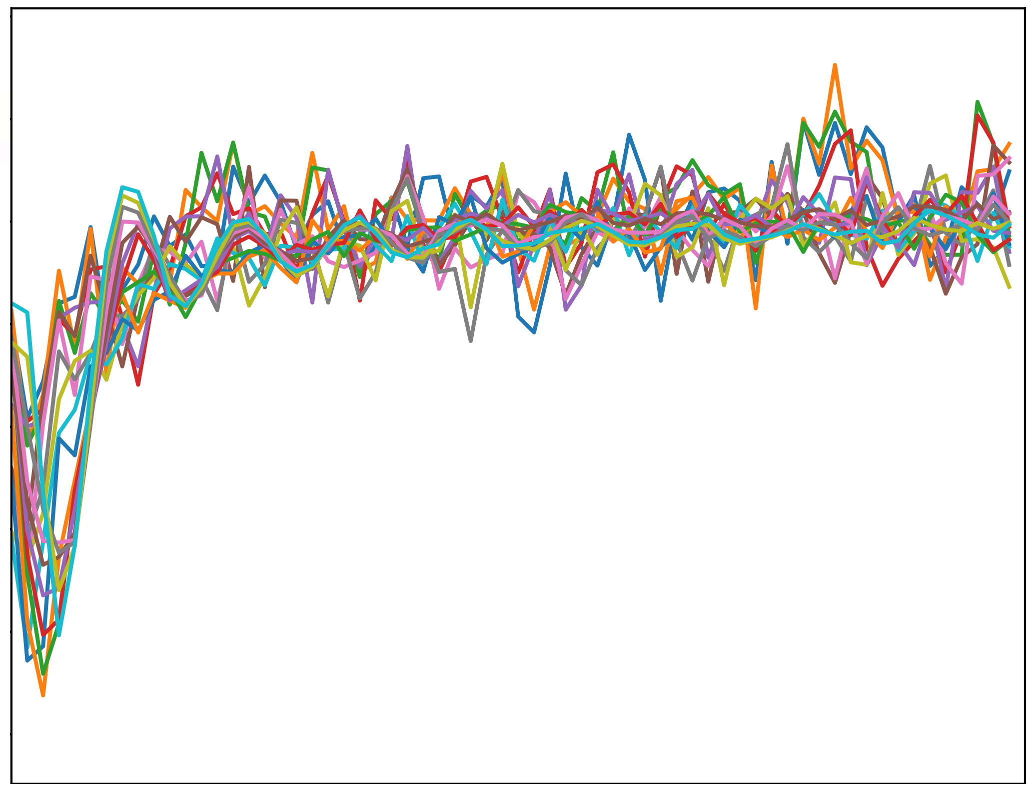
56x Slowdown

Training Process





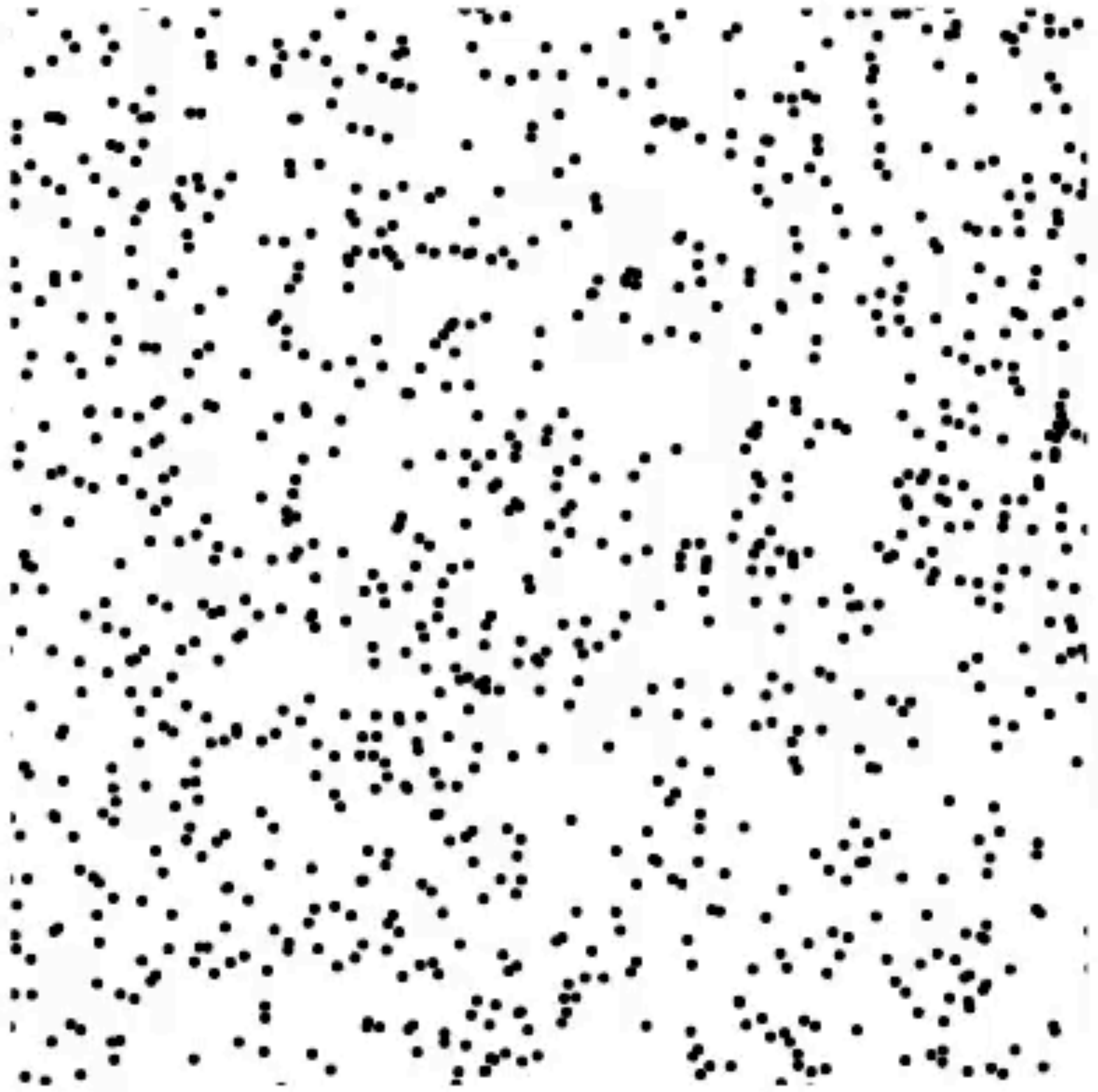
Kernels for BNOT
(de Goes et al.[2012])



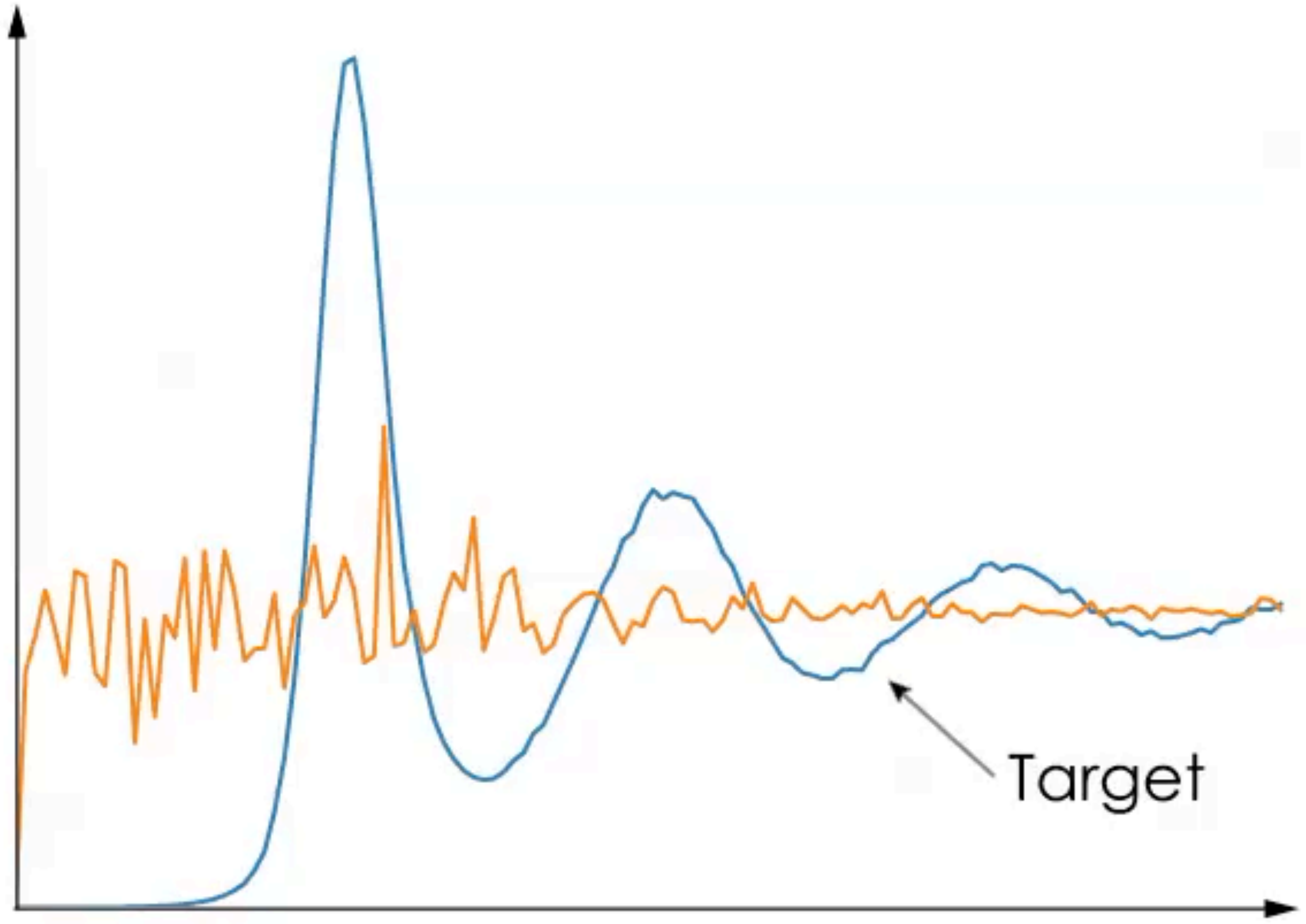
Kernels for Step
(de Heck et al.[2013])

Training Process

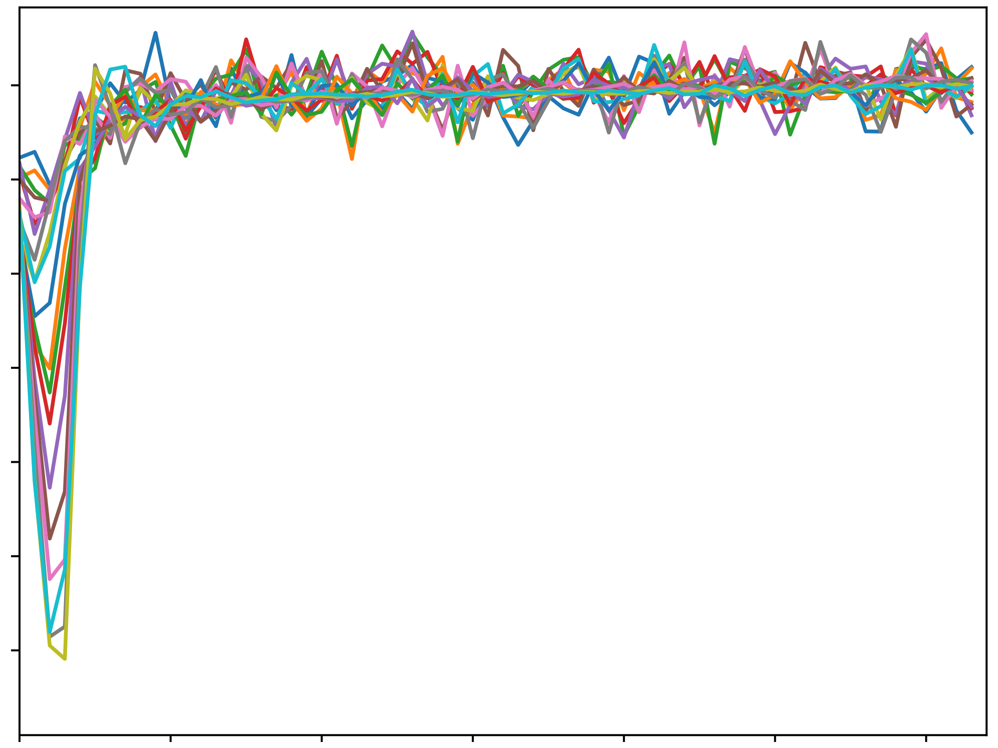
Points



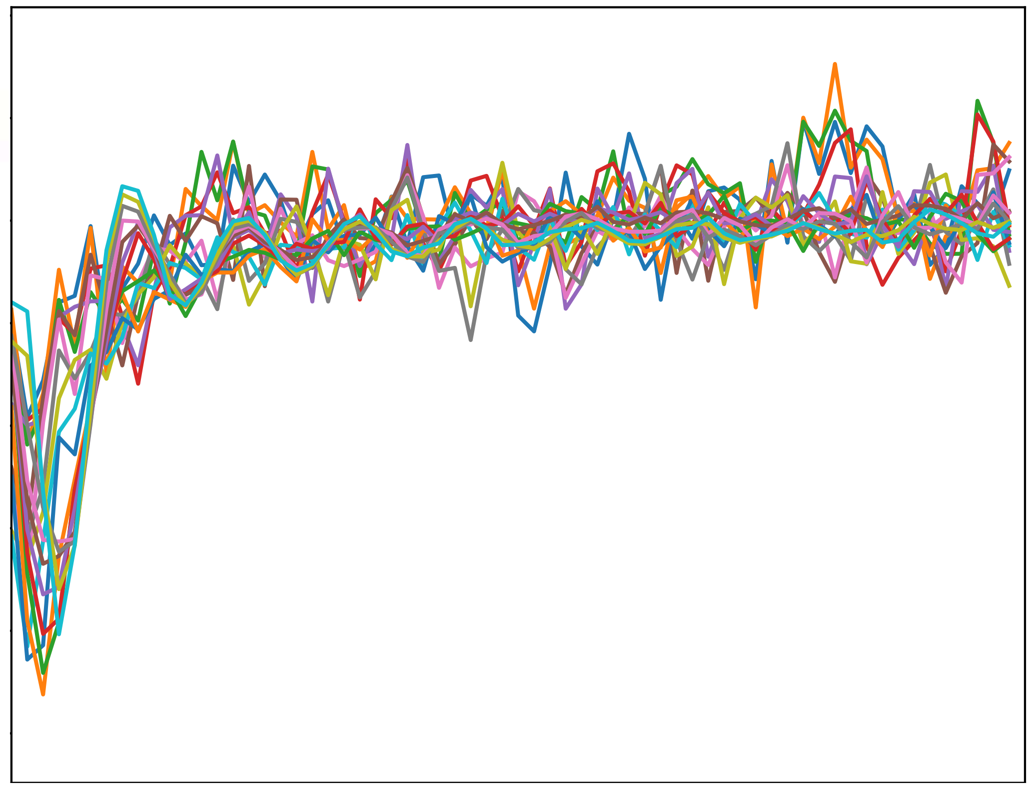
Power Spectrum



56x Slowdown



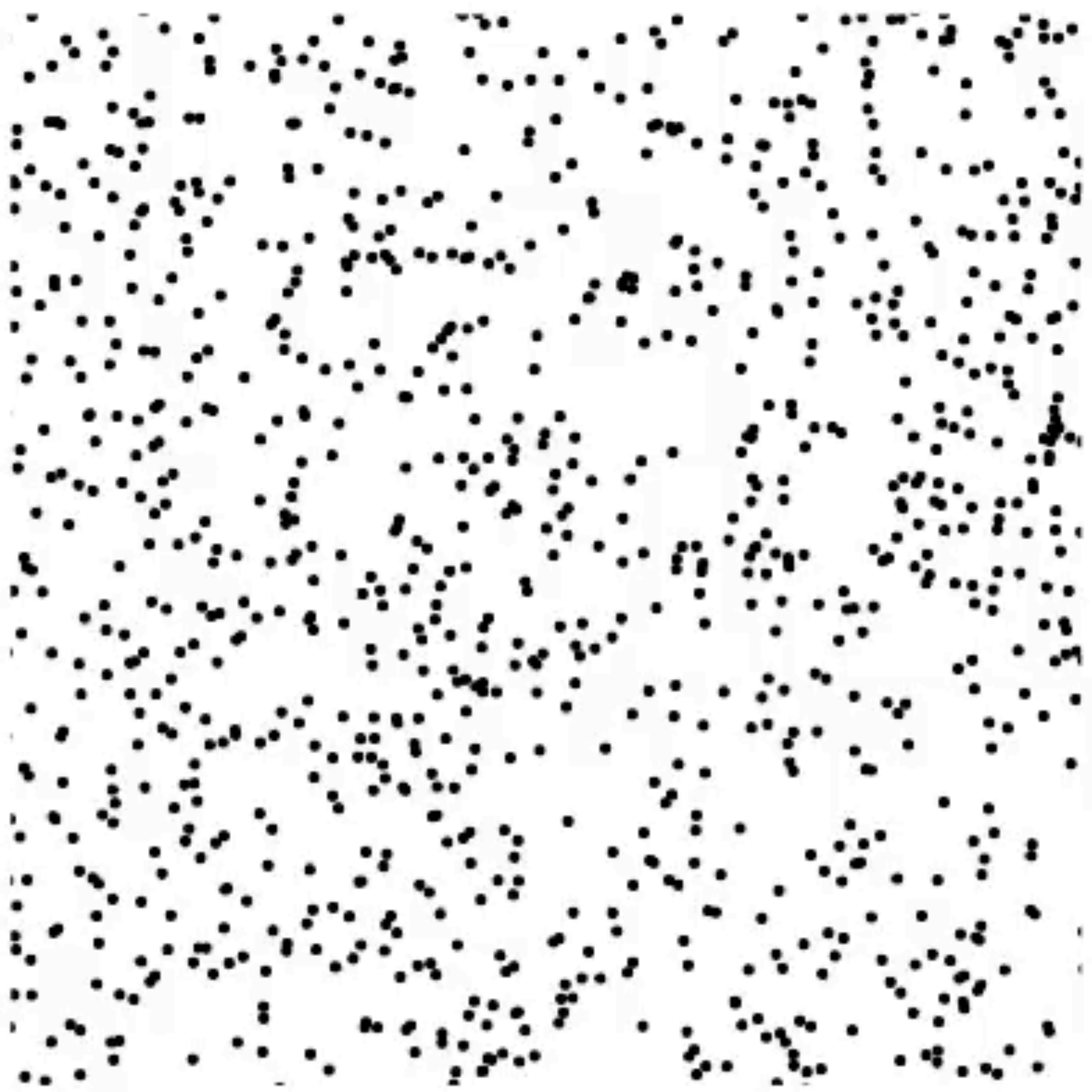
Kernels for BNOT
(de Goes et al.[2012])



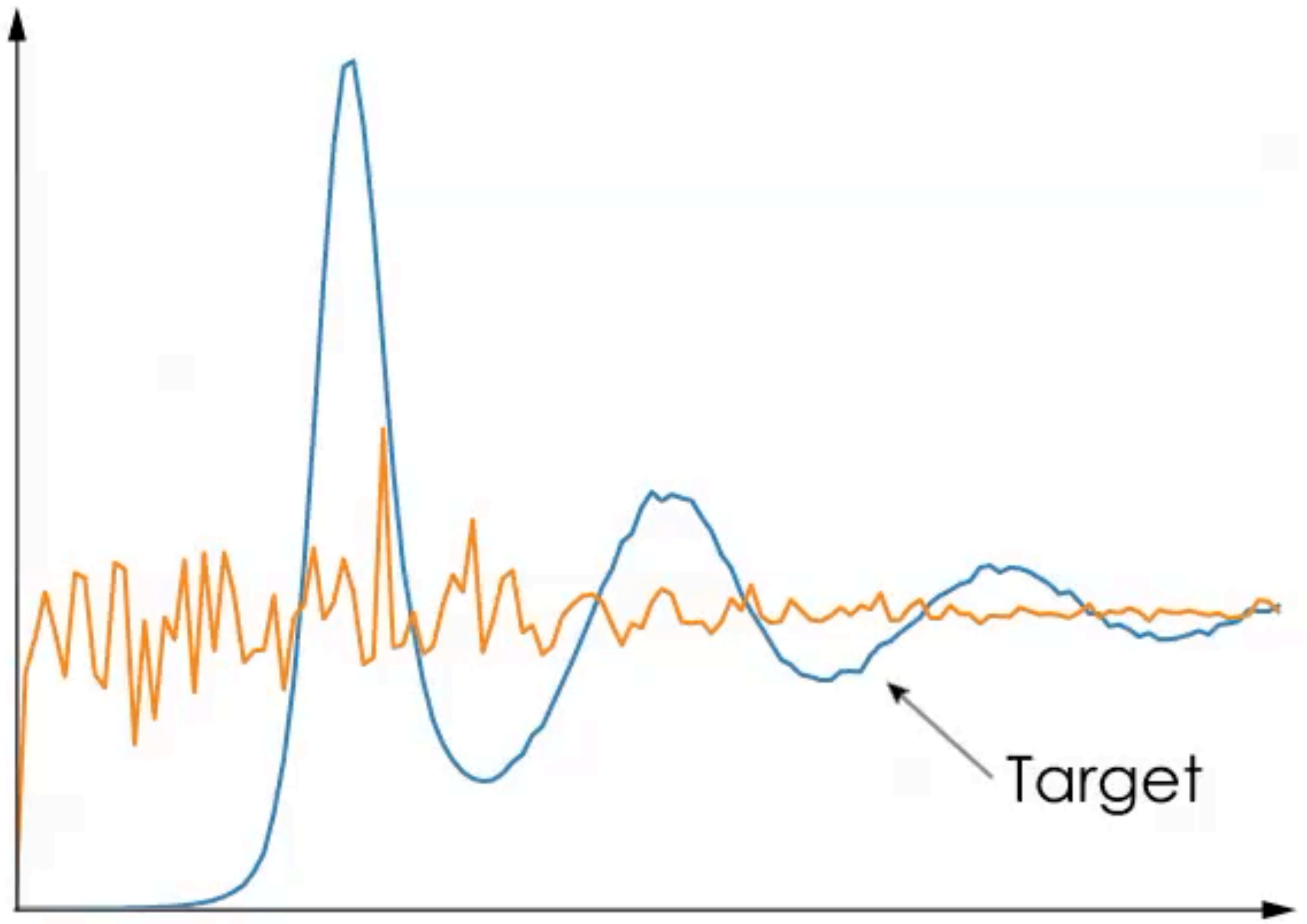
Kernels for Step
(de Heck et al.[2013])

Training Process

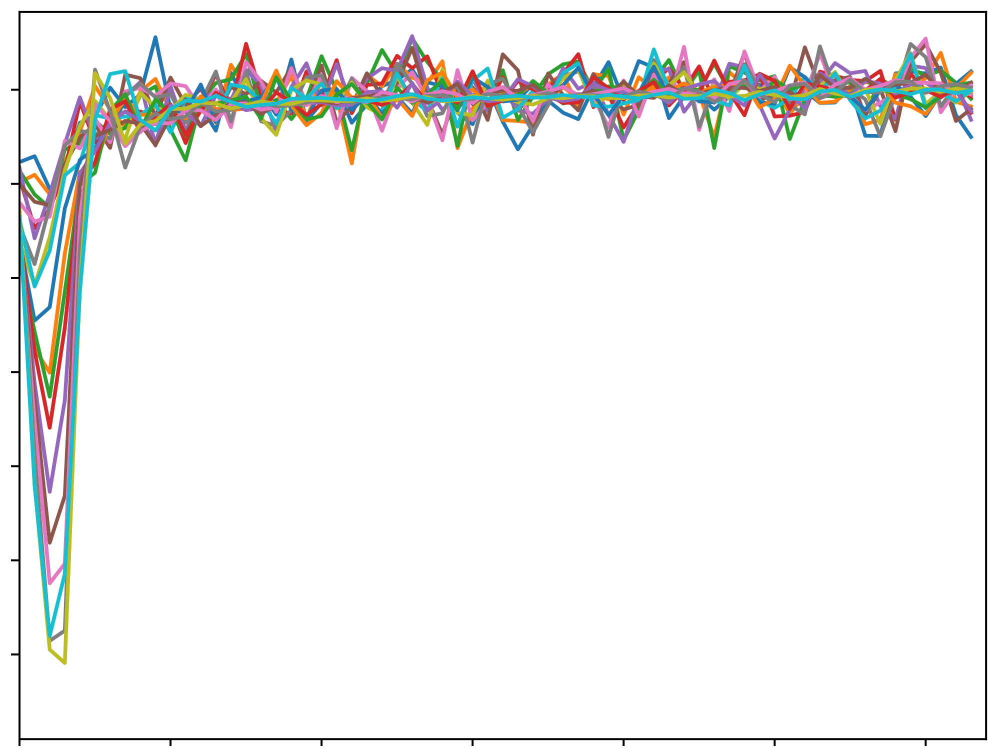
Points



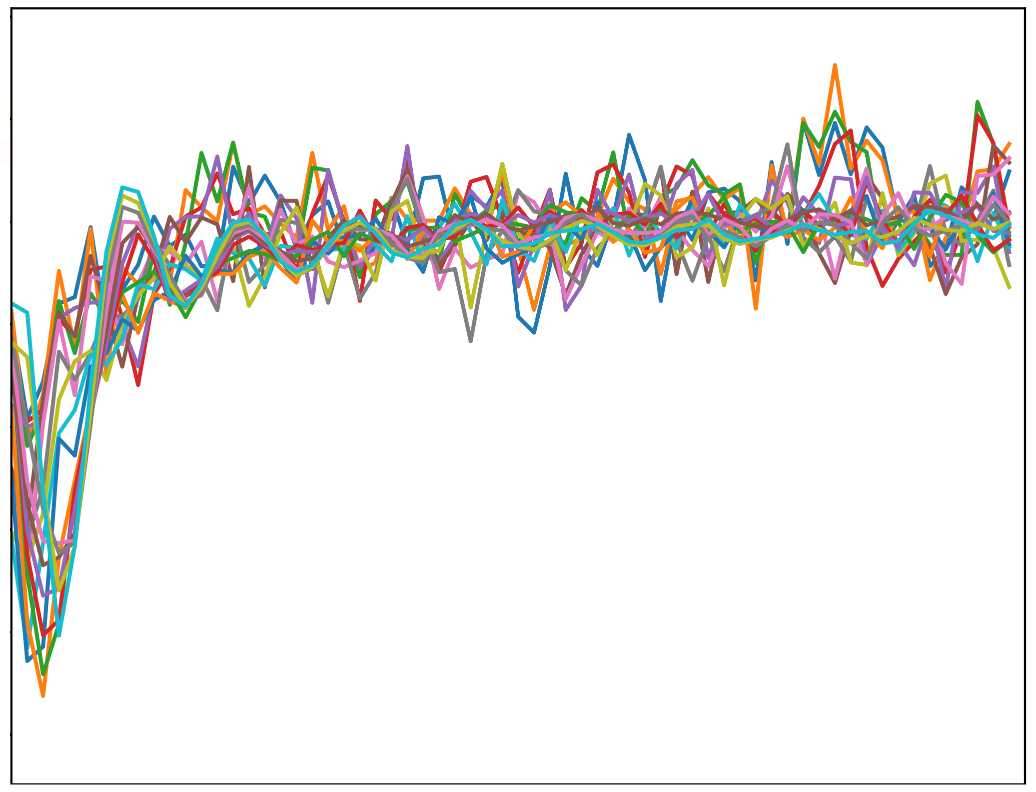
Power Spectrum



56x Slowdown



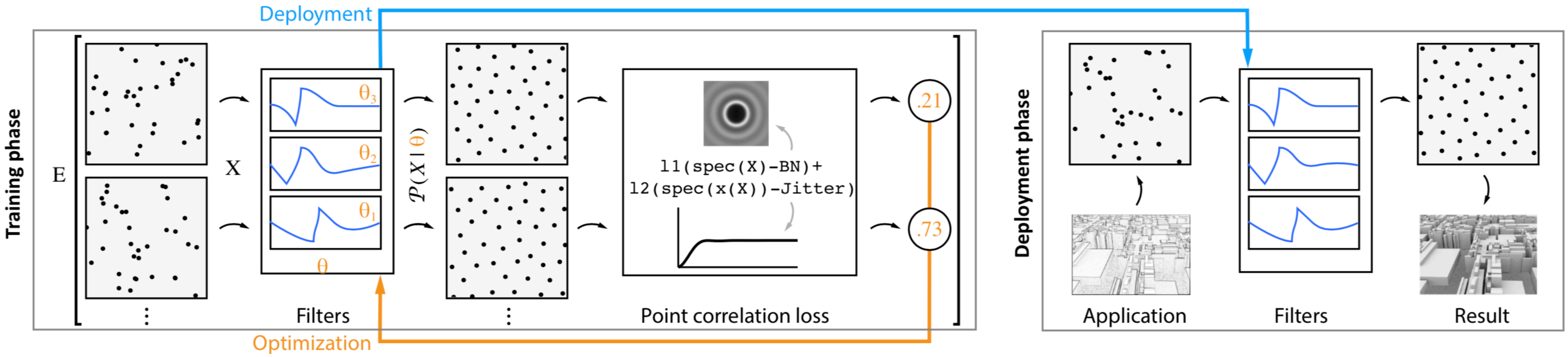
Kernels for BNOT
(de Goes et al.[2012])



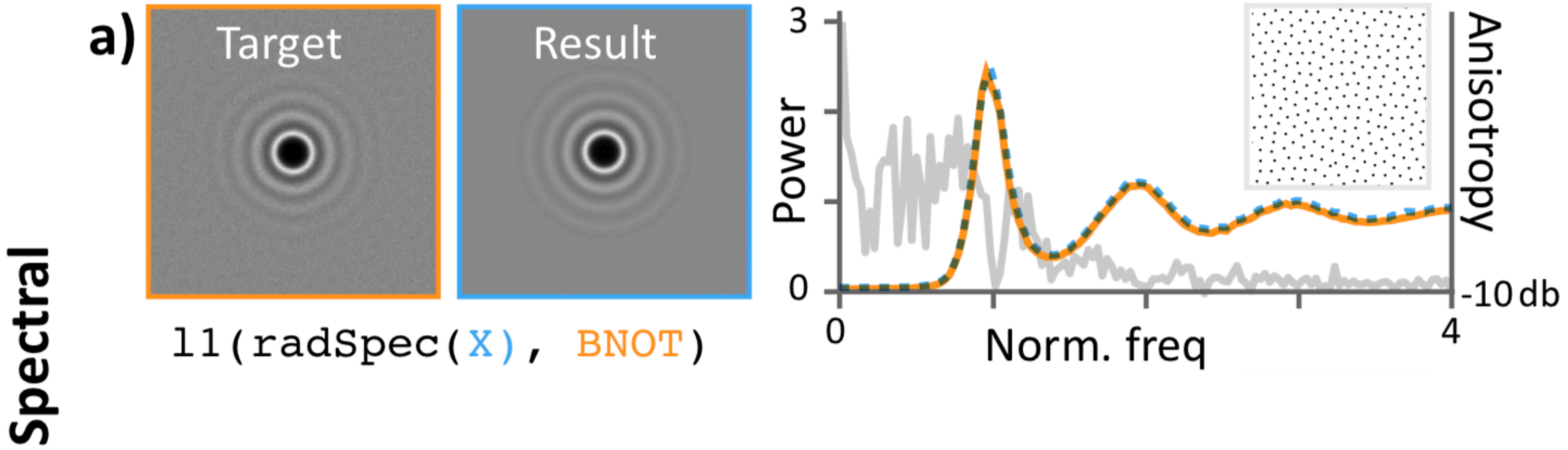
Kernels for Step
(de Heck et al.[2013])

Architecture: Full pipeline

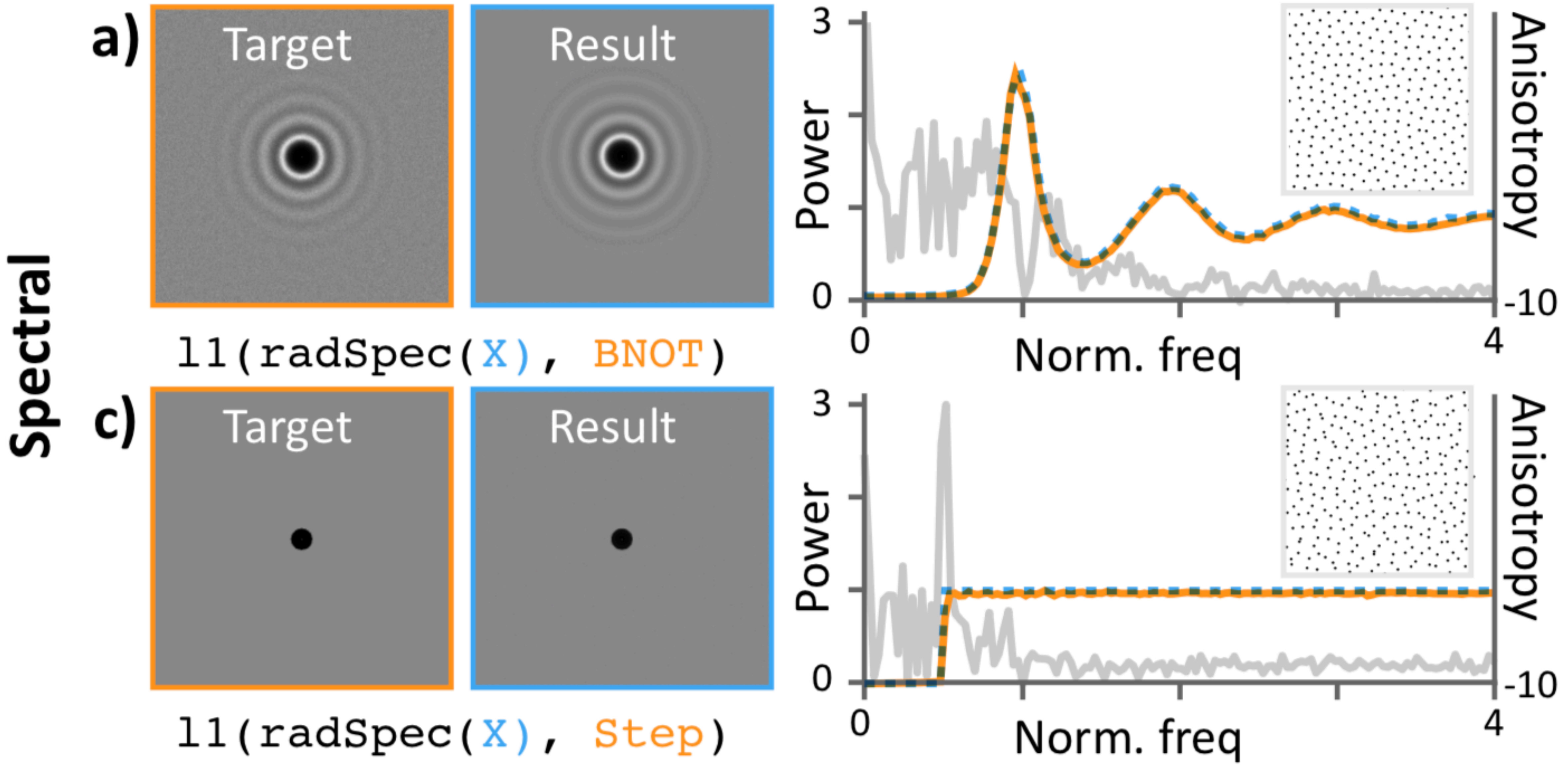
Architecture: Full pipeline



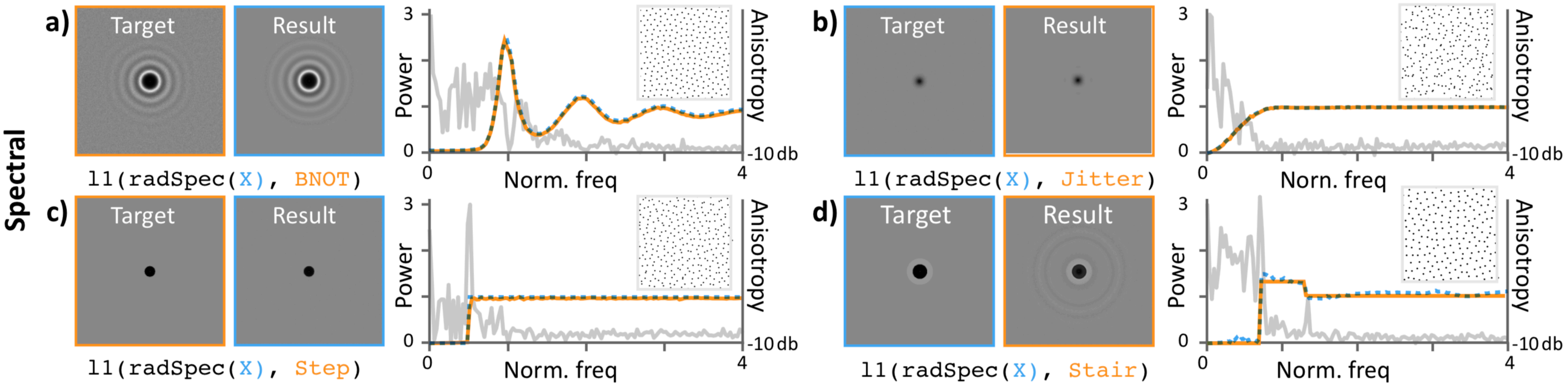
Results: Spectral Target Spectra



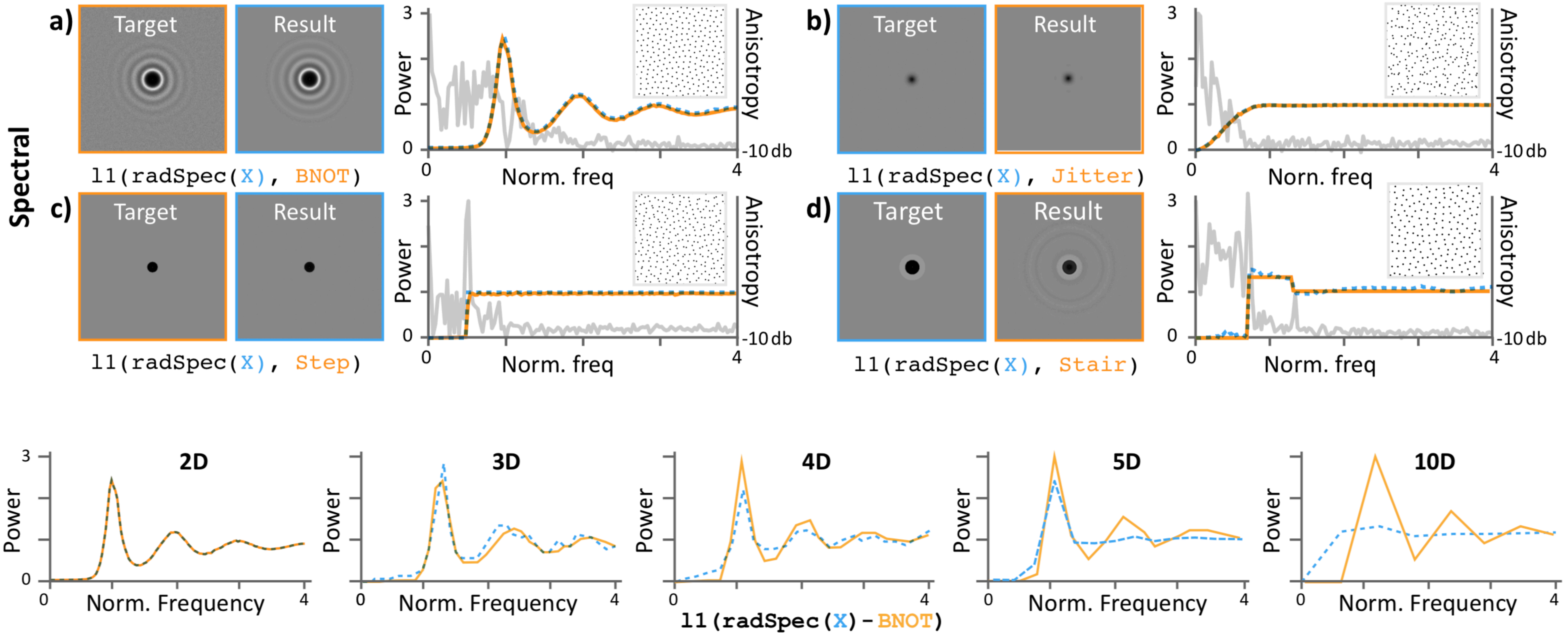
Results: Spectral Target Spectra



Results: Spectral Target Spectra



Results: Spectral Target Spectra

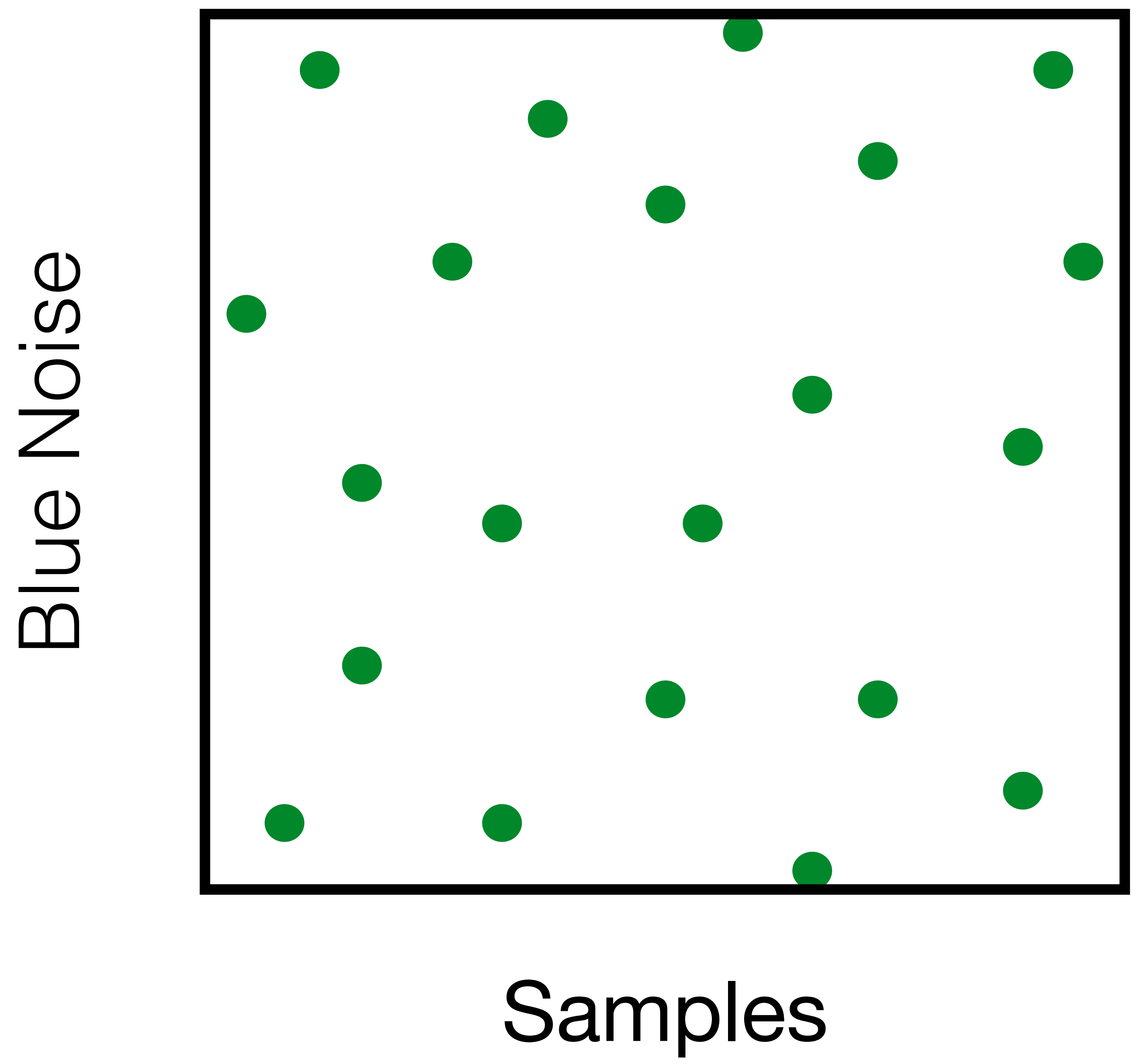


Spatial Loss Function

PCF Loss at $i\text{-th}$ training iteration

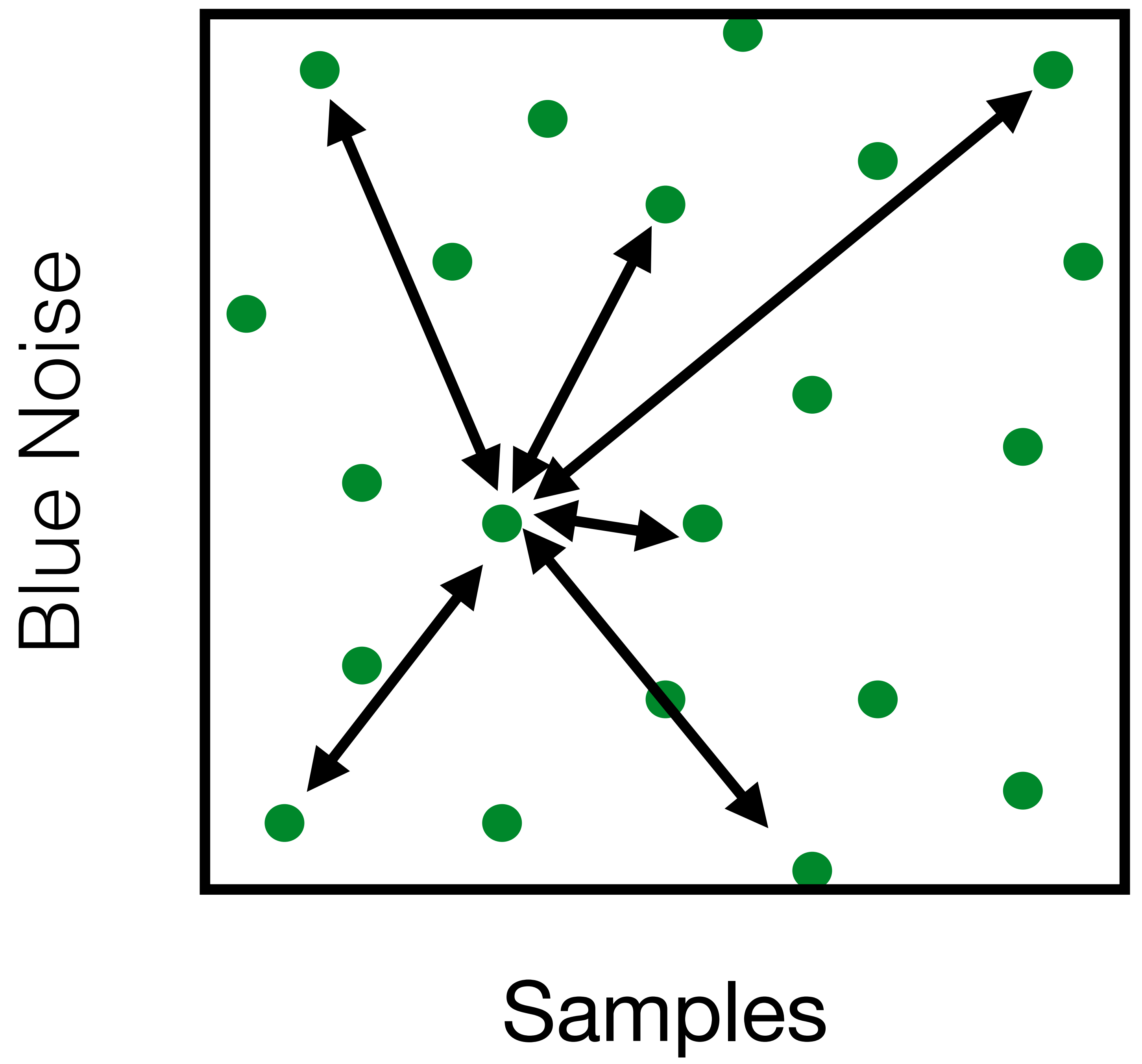
Pair Correlation Function

Spatial Domain



Pair Correlation Function

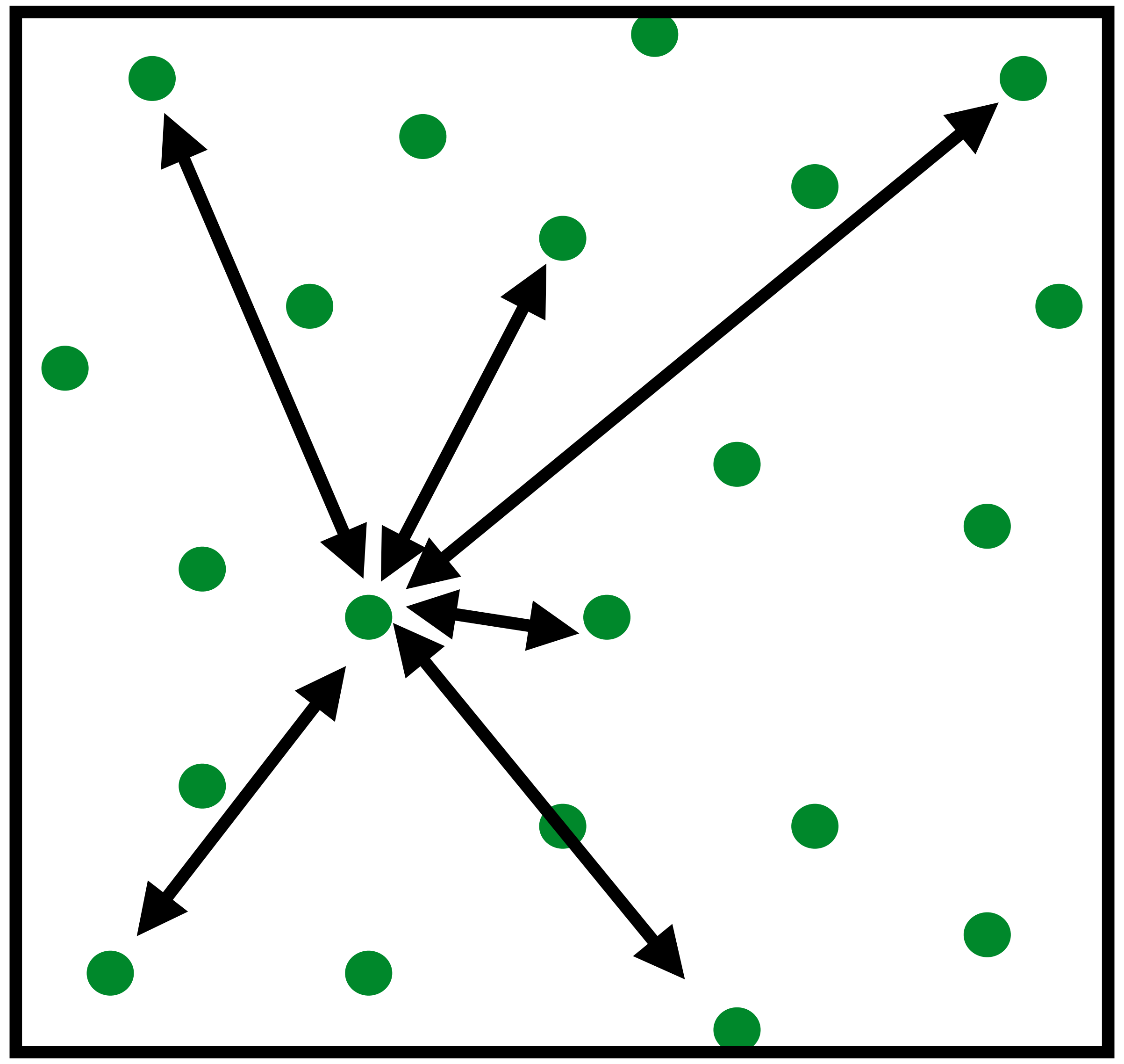
Spatial Domain



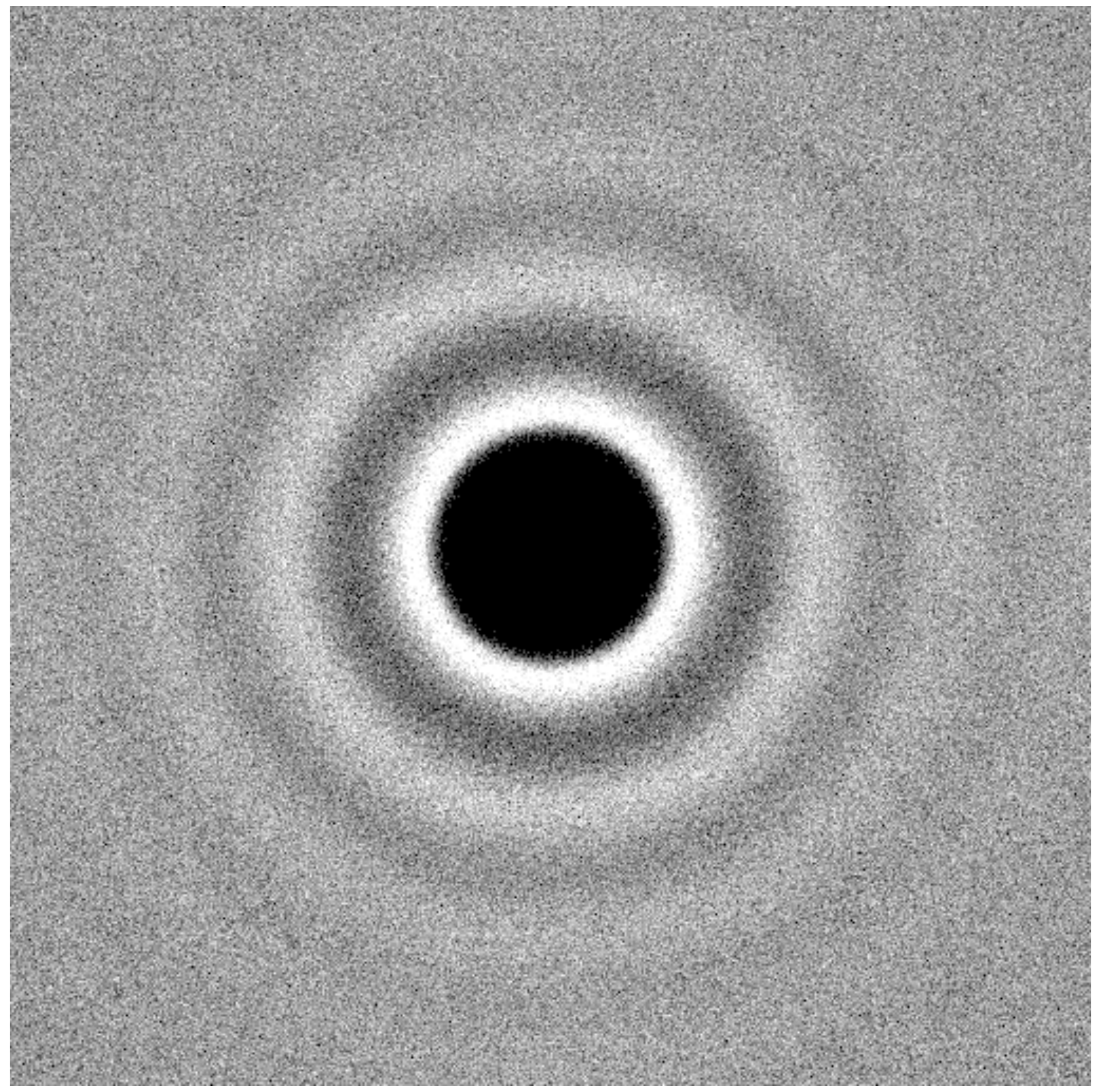
Pair Correlation Function

Spatial Domain

Blue Noise



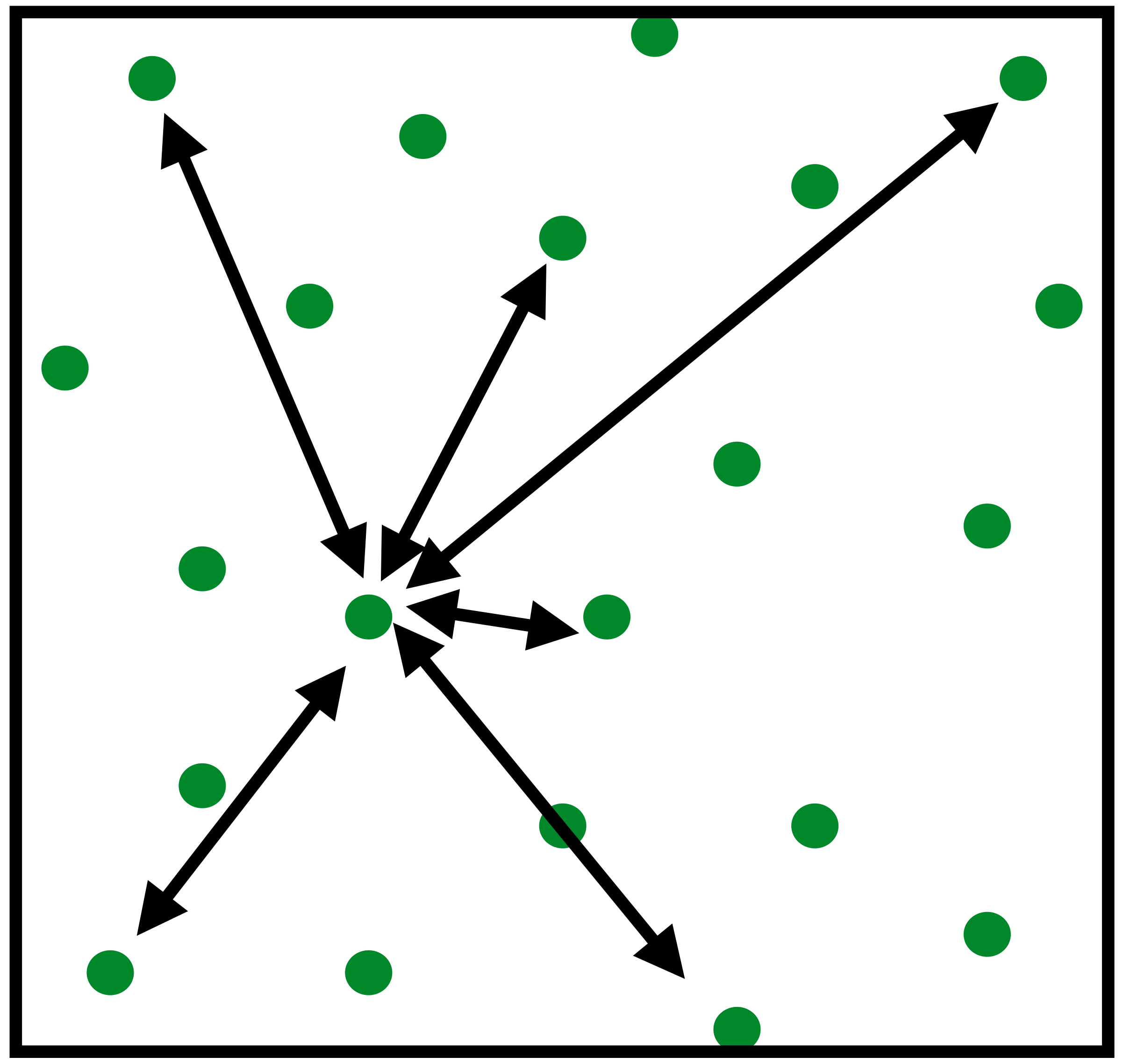
Samples

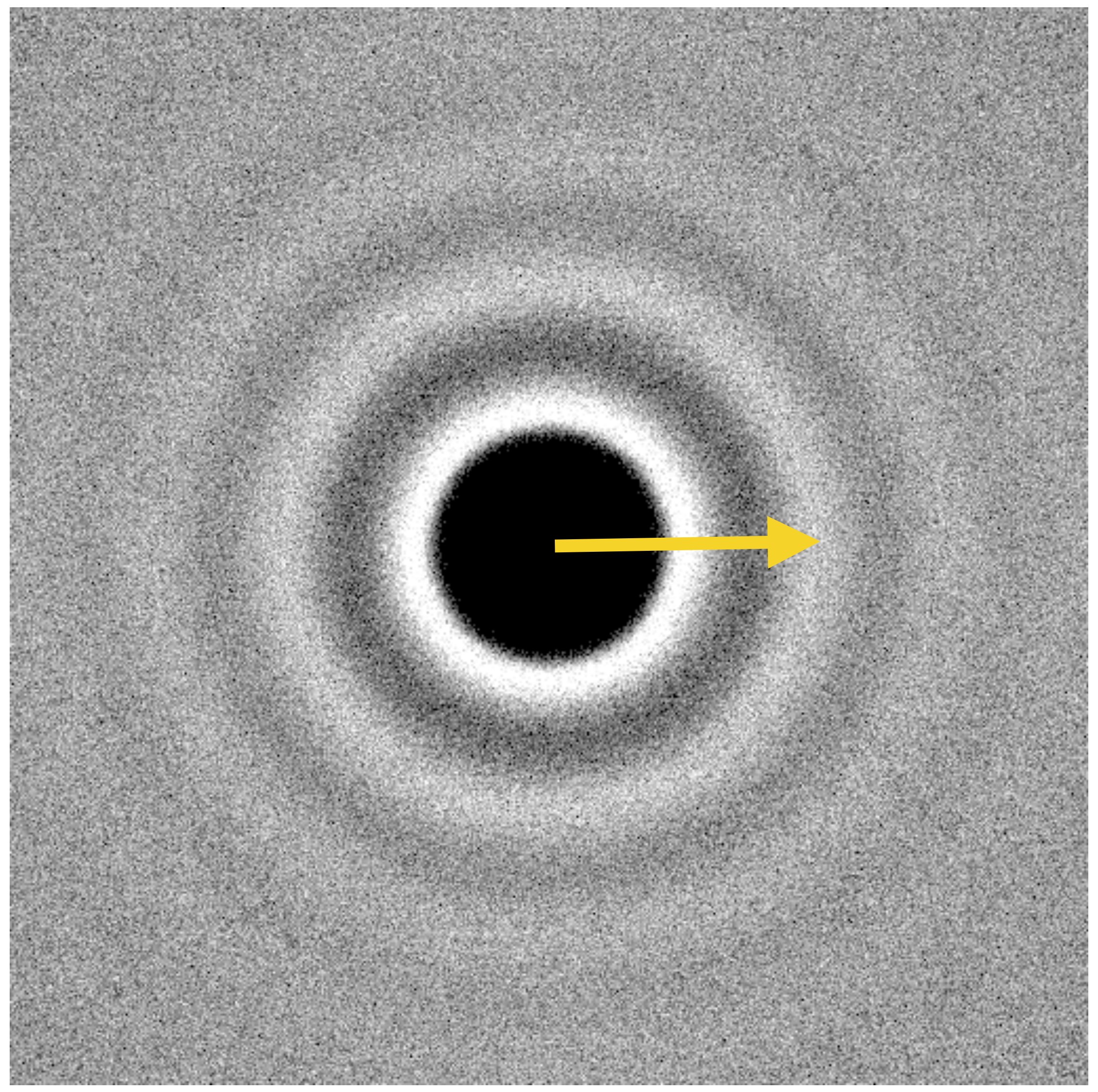


Pair Correlation Function

Spatial Domain

Blue Noise

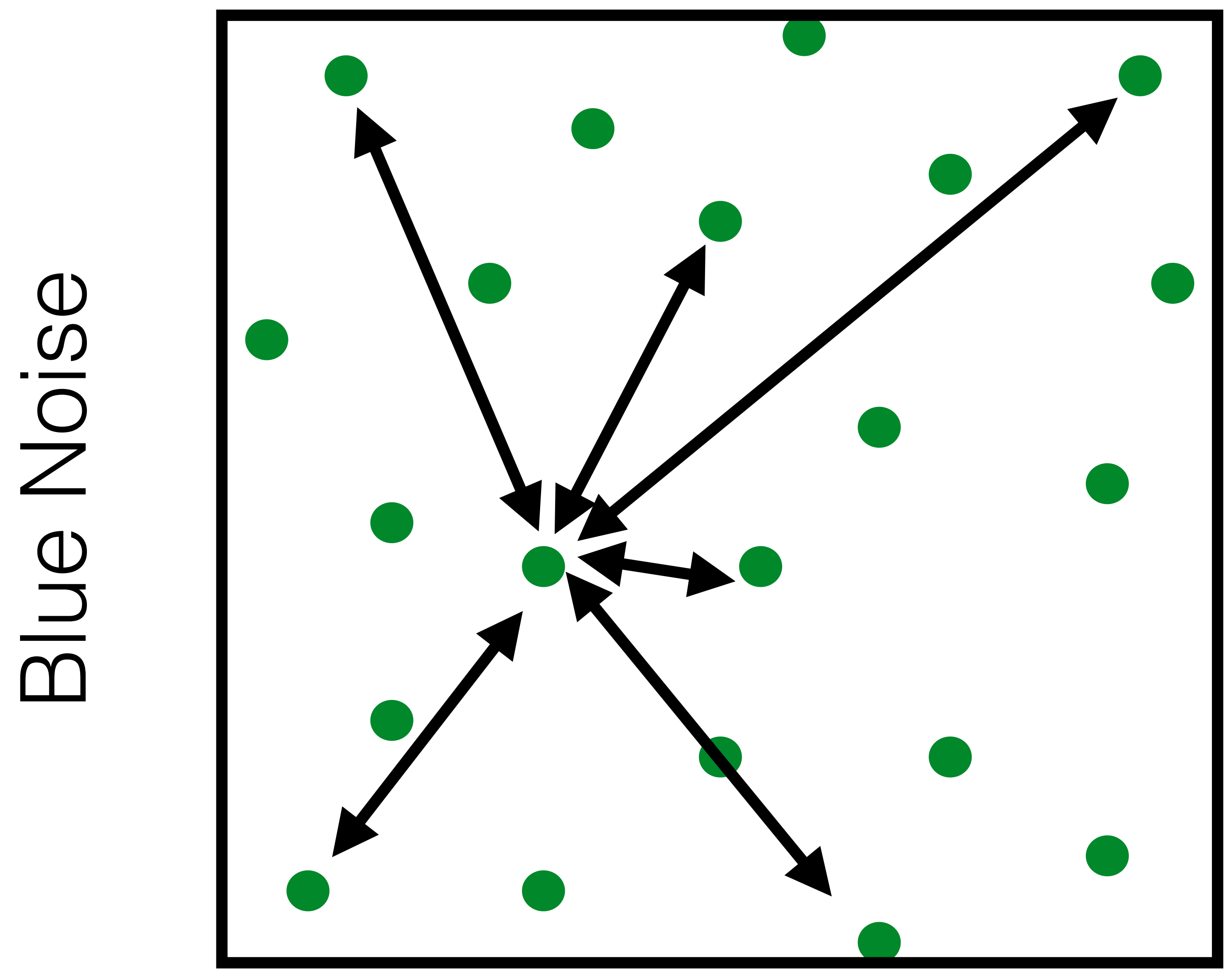




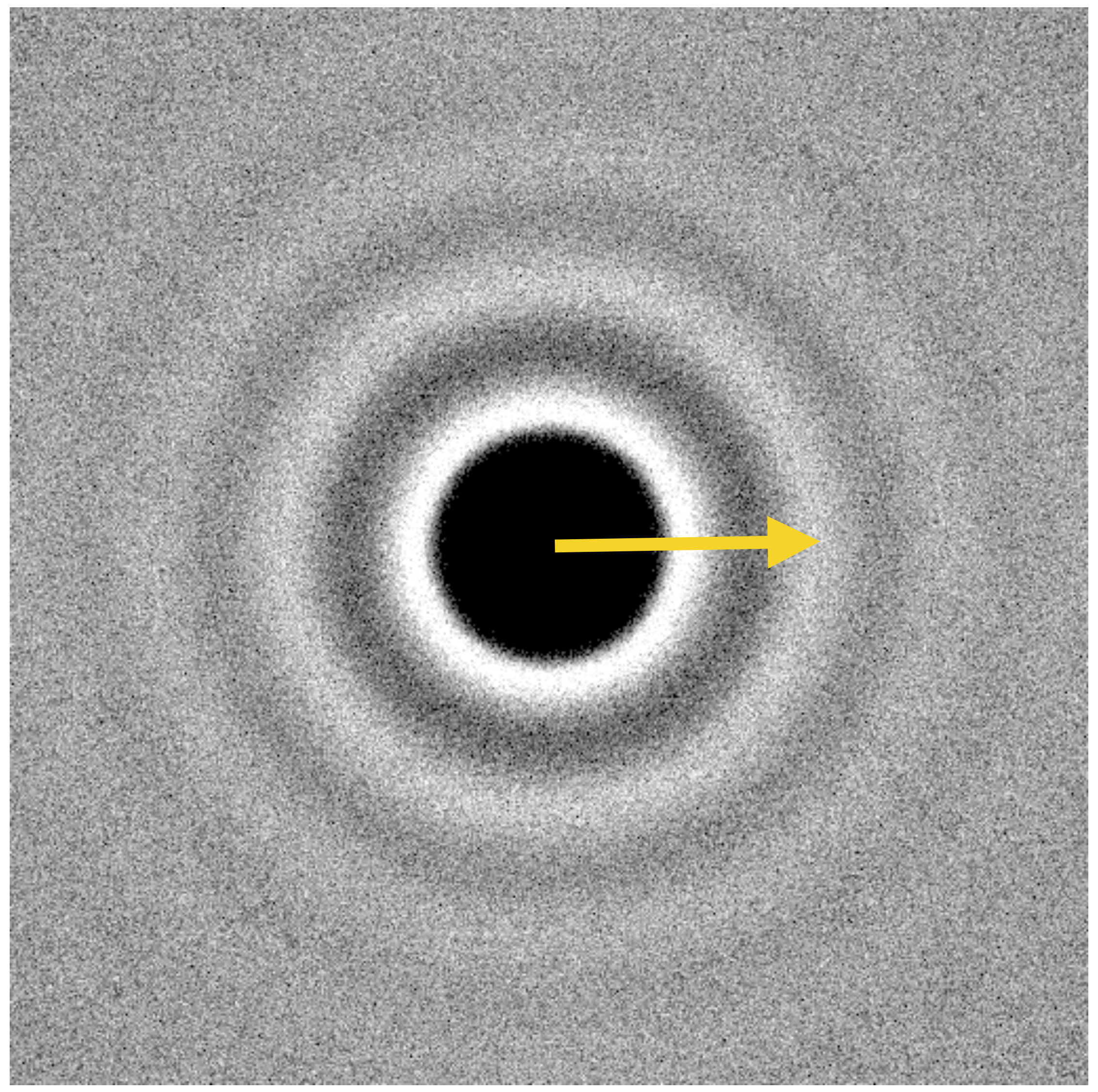
Samples

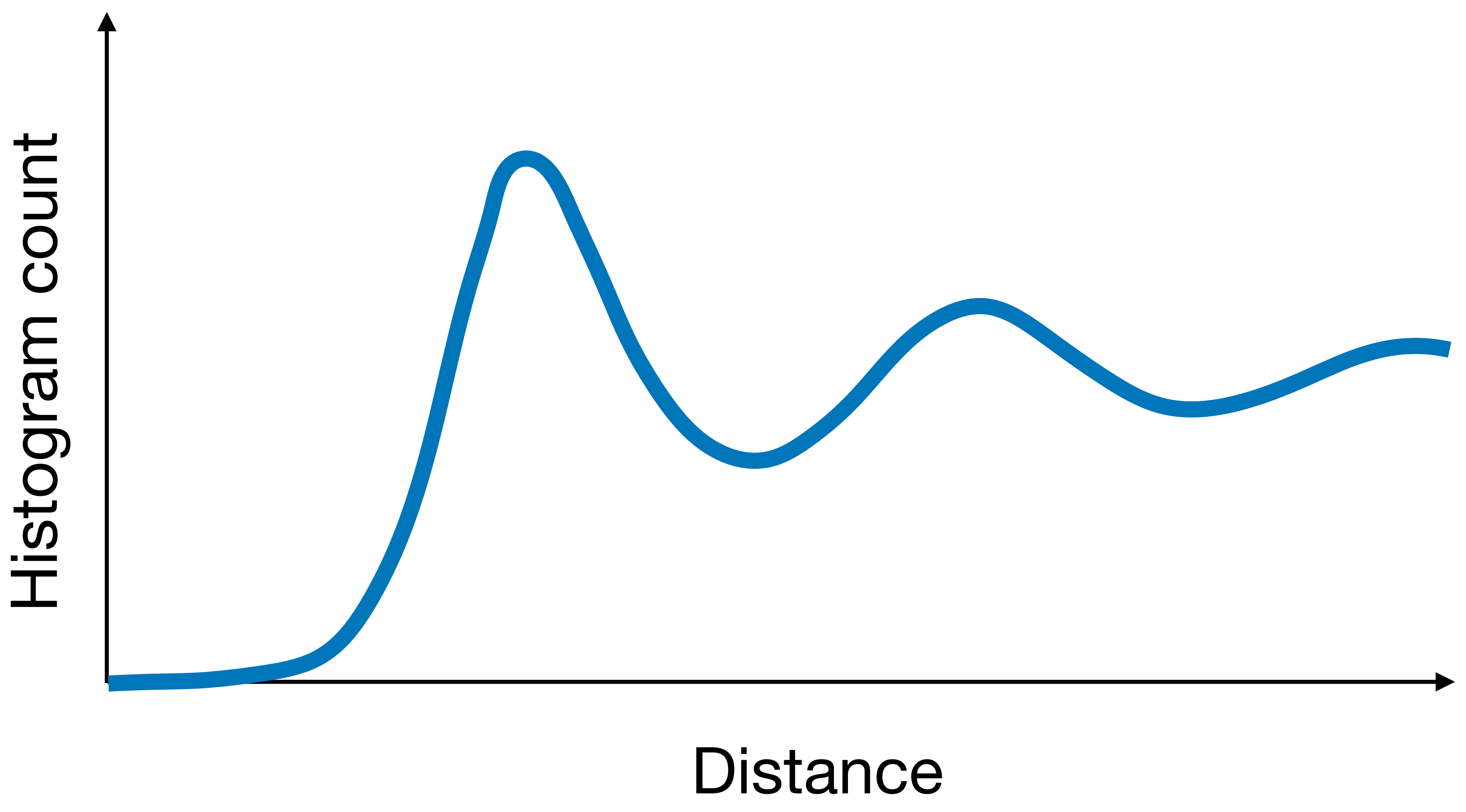
Pair Correlation Function

Spatial Domain



Samples





Loss Functions

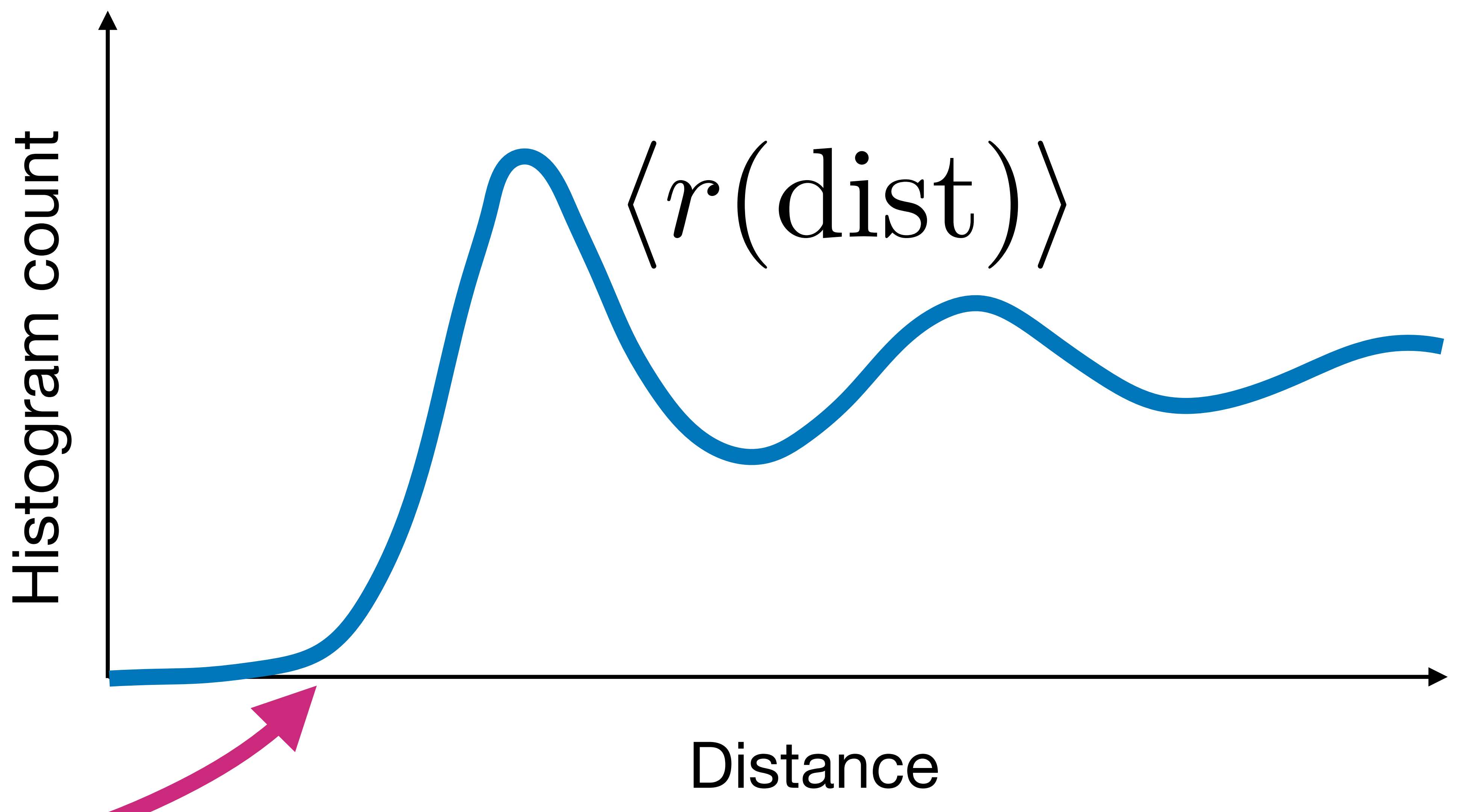
PCF Loss at $i\text{-th}$ training iteration

$$L_{\mathrm{PCF}} = ||\langle r_i(\mathrm{dist})\rangle - \langle r(\mathrm{dist})\rangle||^2$$

Loss Functions

PCF Loss at $i\text{-th}$ training iteration

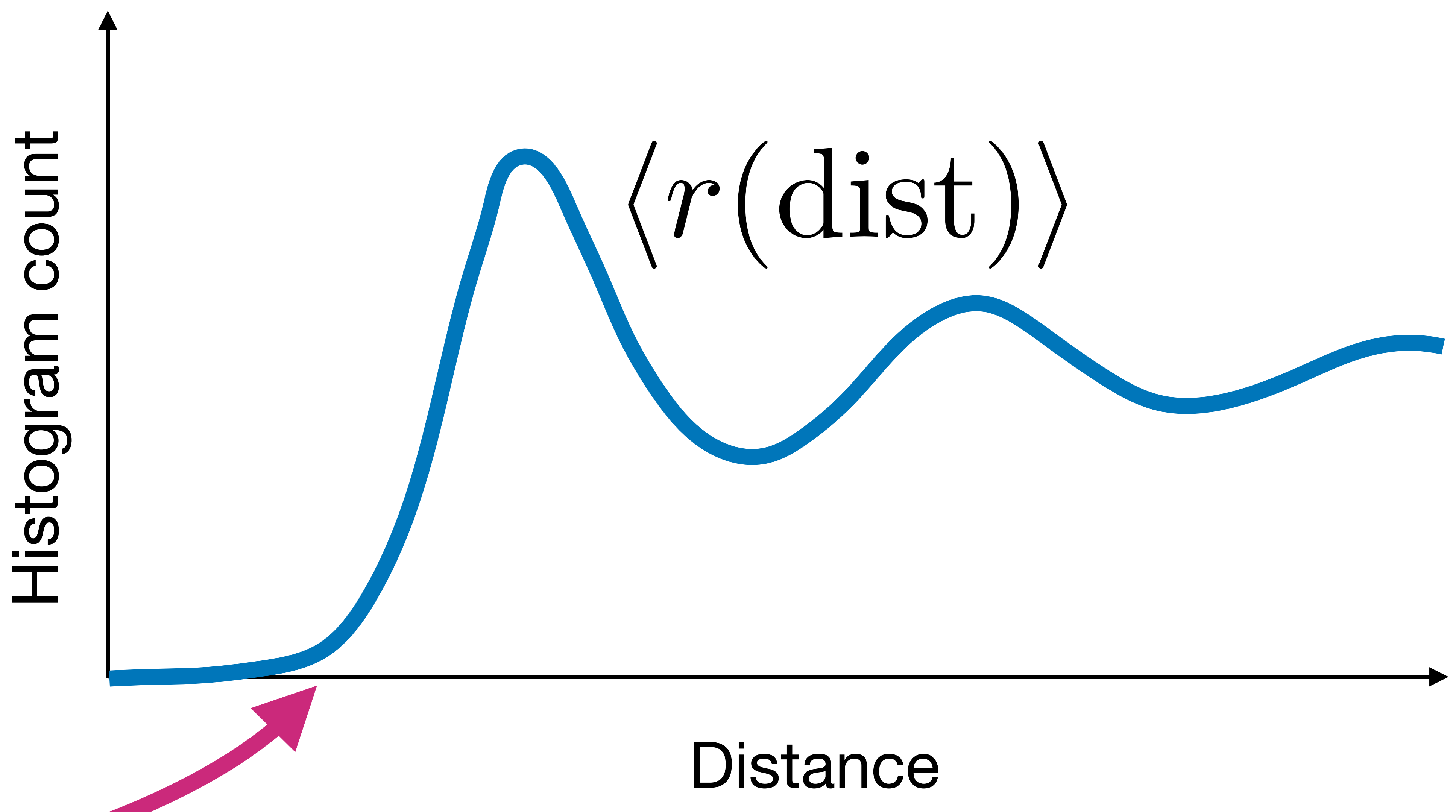
$$L_{\mathrm{PCF}} = ||\langle r_i(\mathrm{dist})\rangle - \langle r(\mathrm{dist})\rangle||^2$$



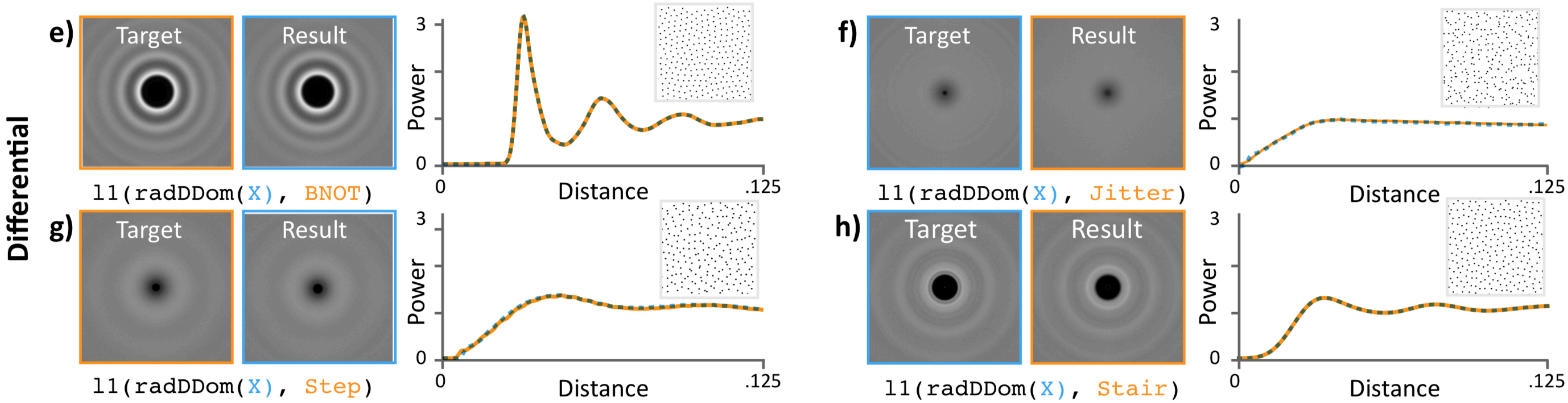
Loss Functions

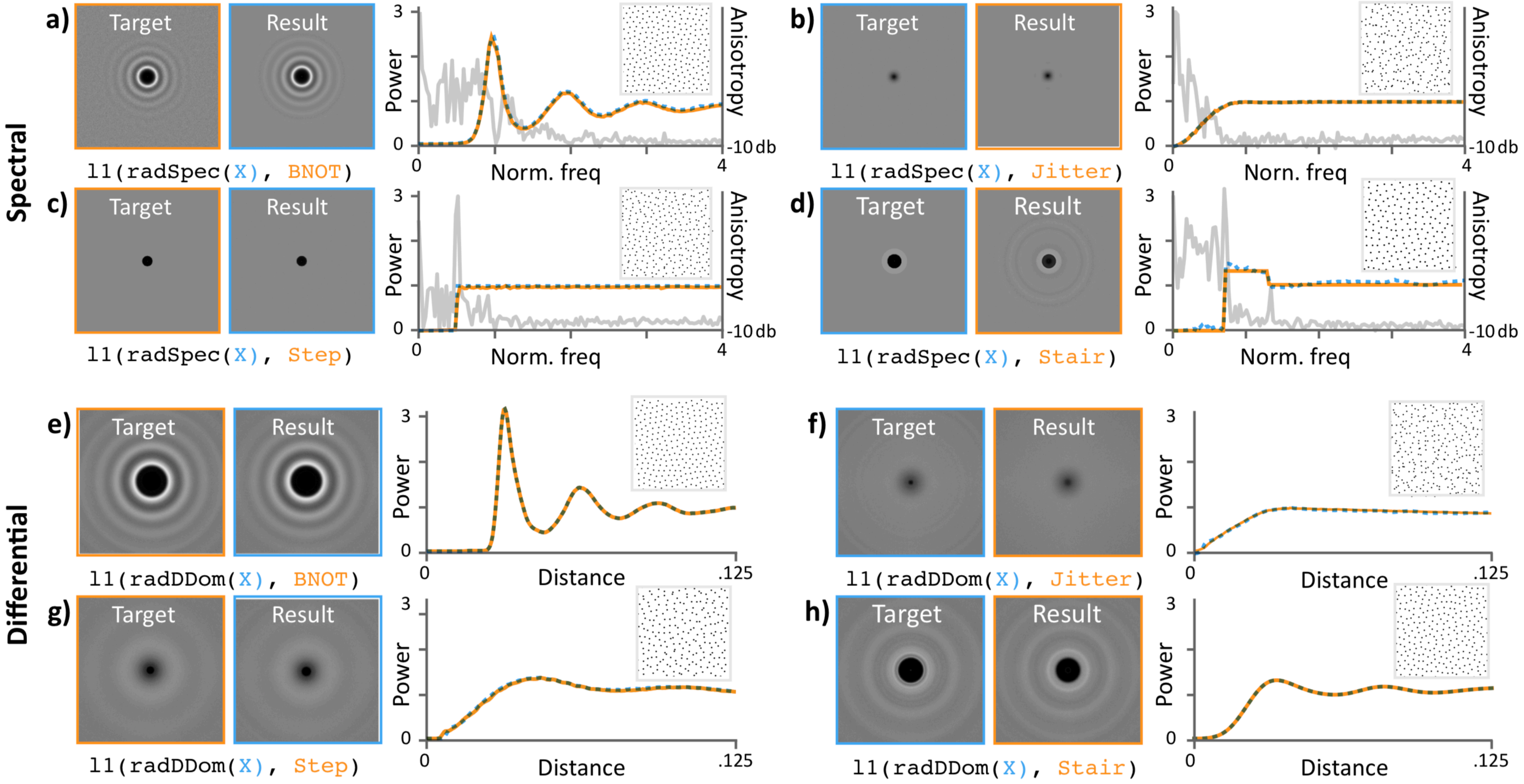
PCF Loss at i-th training iteration

$$L_{\mathrm{PCF}} = ||\langle r_i(\mathrm{dist})\rangle - \langle r(\mathrm{dist})\rangle||^2$$

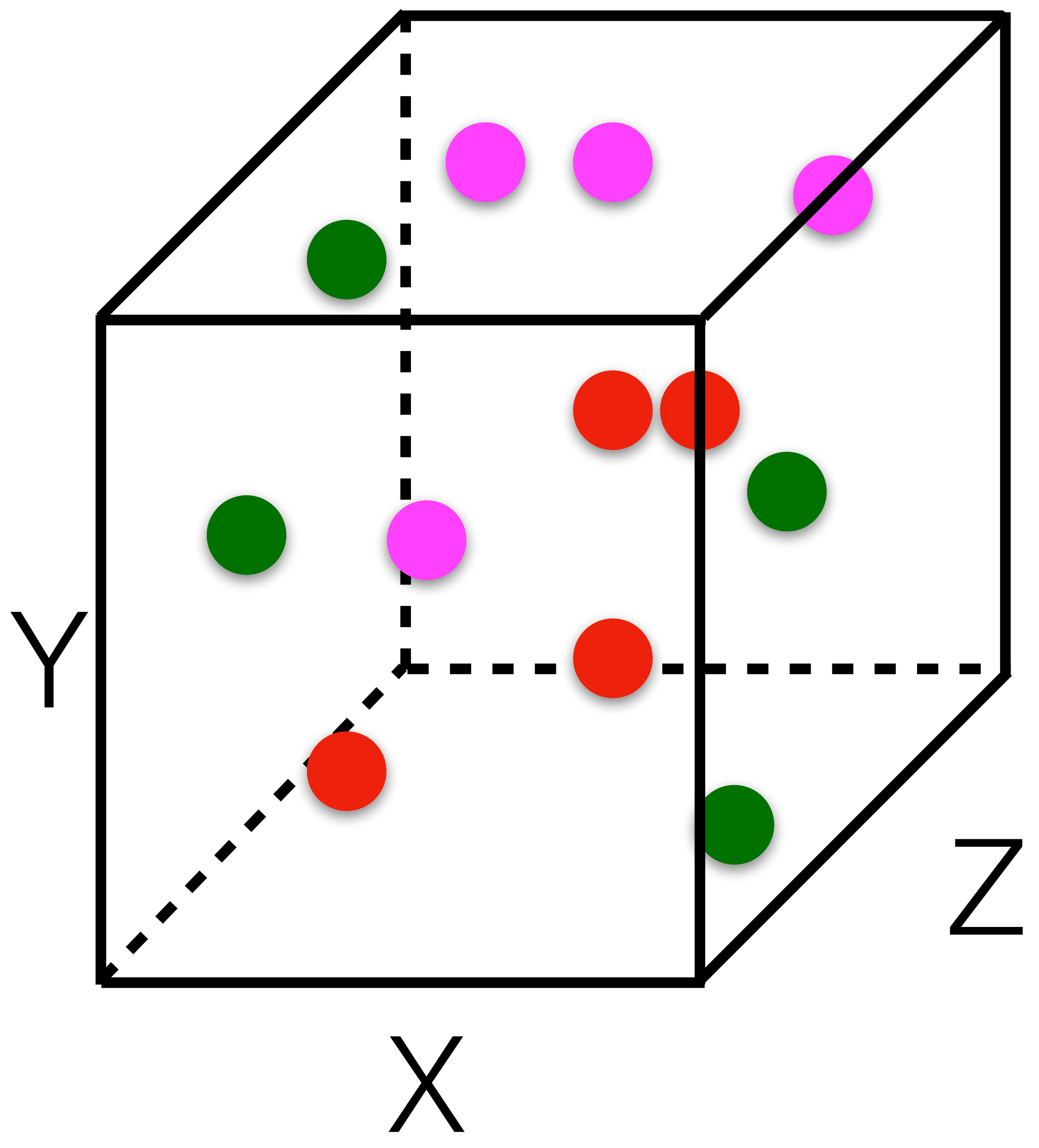


Spatial Target PCFs

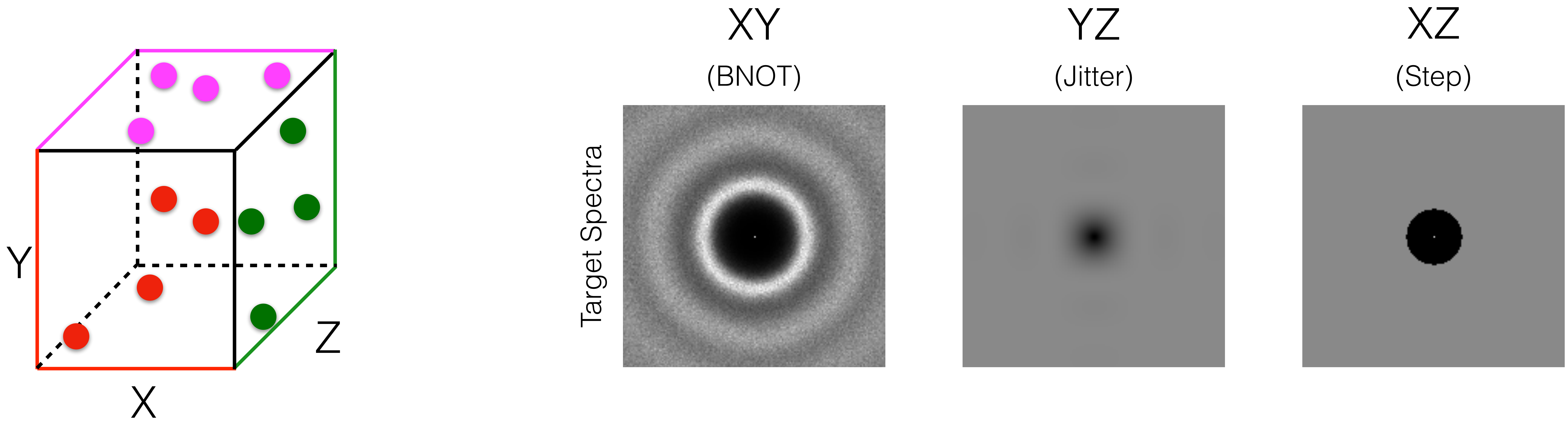



Spectral
Differential
a)
Target
Result
l1(radSpec(X), BNOT)
Power
Norm. freq
Anisotropy
-10db
b)
Target
Result
l1(radSpec(X), Jitter)
Norn. freq
c)
l1(radSpec(X), Step)
d)
l1(radSpec(X), Stair)
e)
l1(radDDom(X), BNOT)
Distance
.125
f)
l1(radDDom(X), Jitter)
g)
l1(radDDom(X), Step)
h)
l1(radDDom(X), Stair)

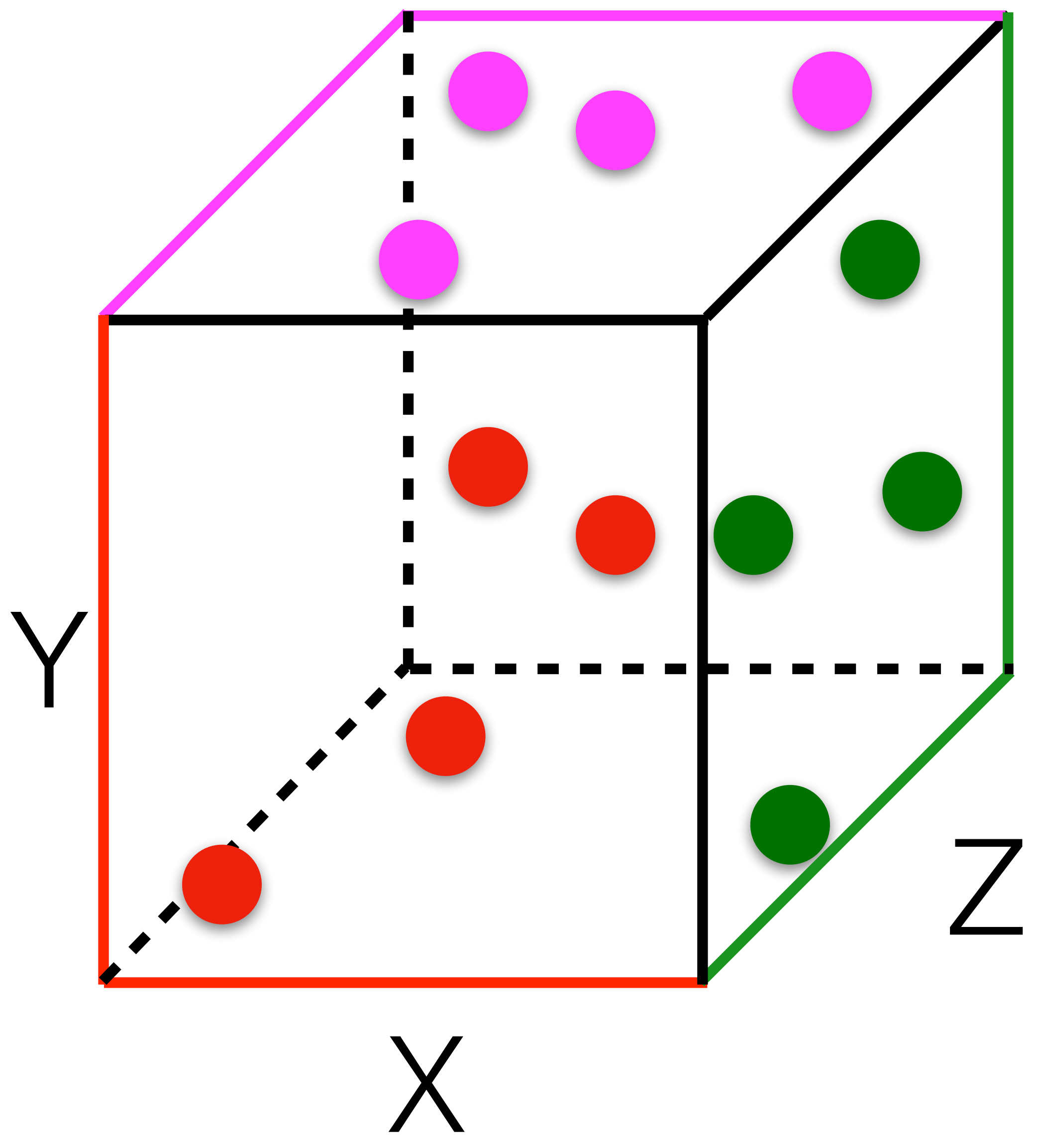
3D Point Samples (Different Projection Targets)

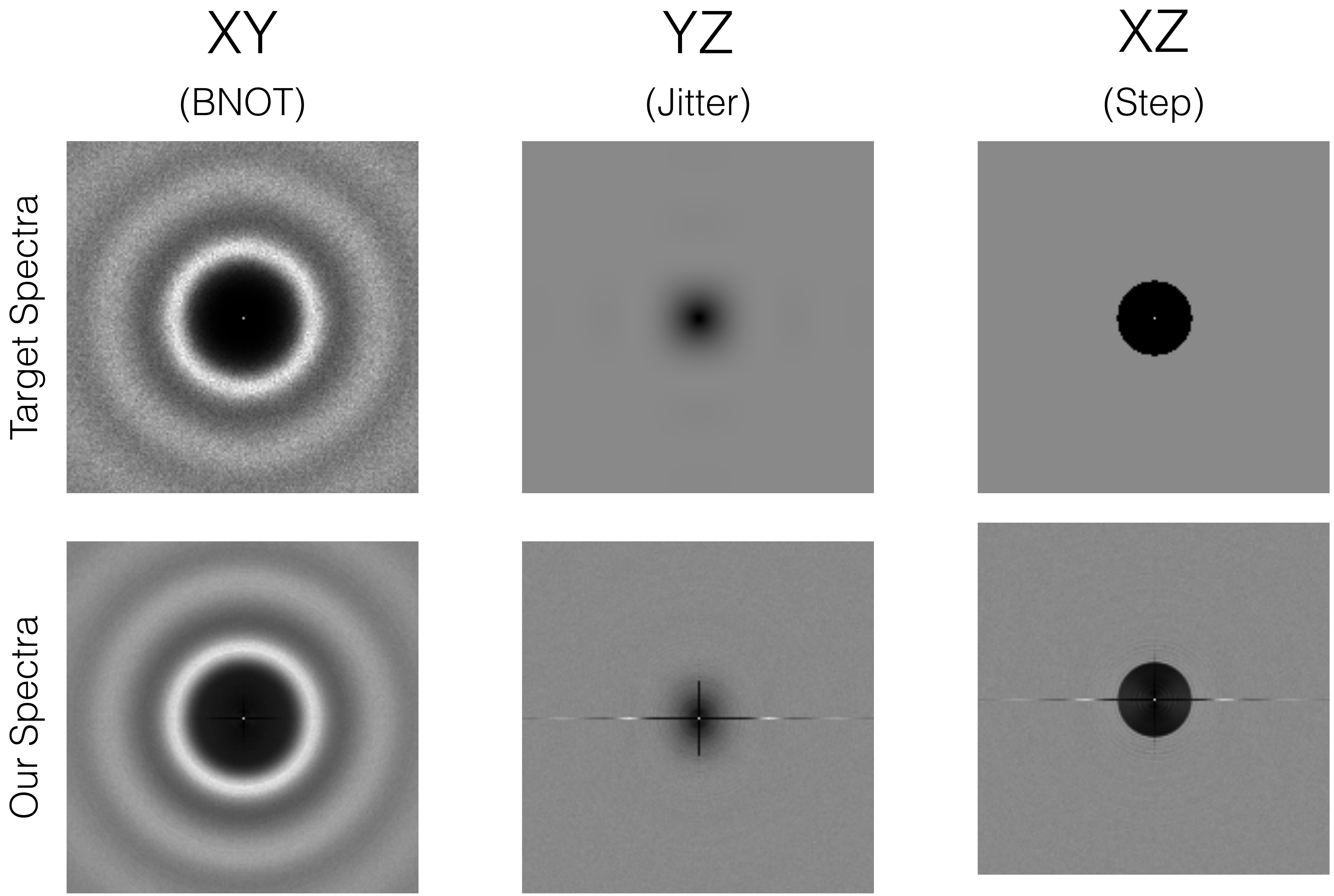


3D Point Samples (Different Projection Targets)

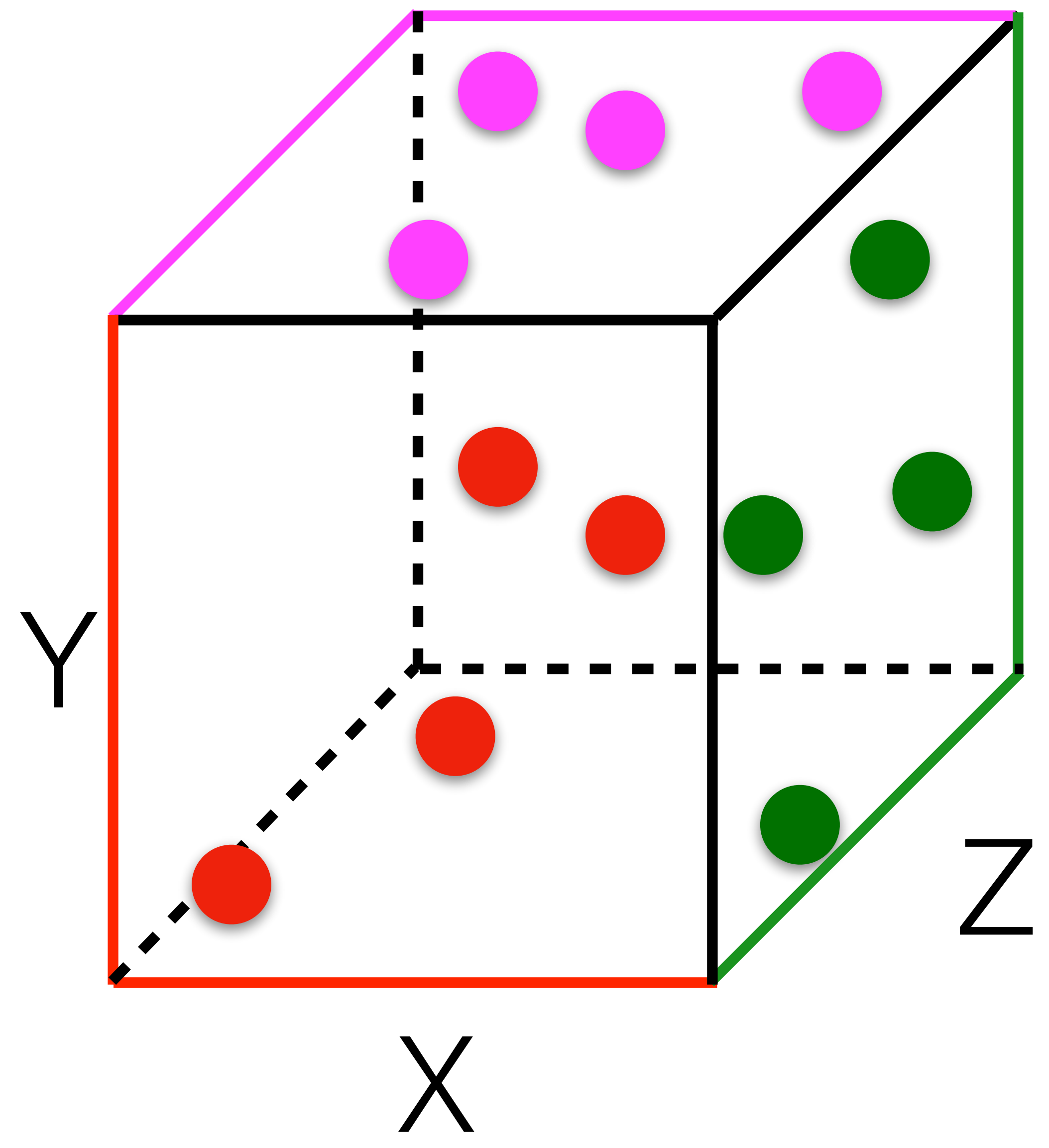


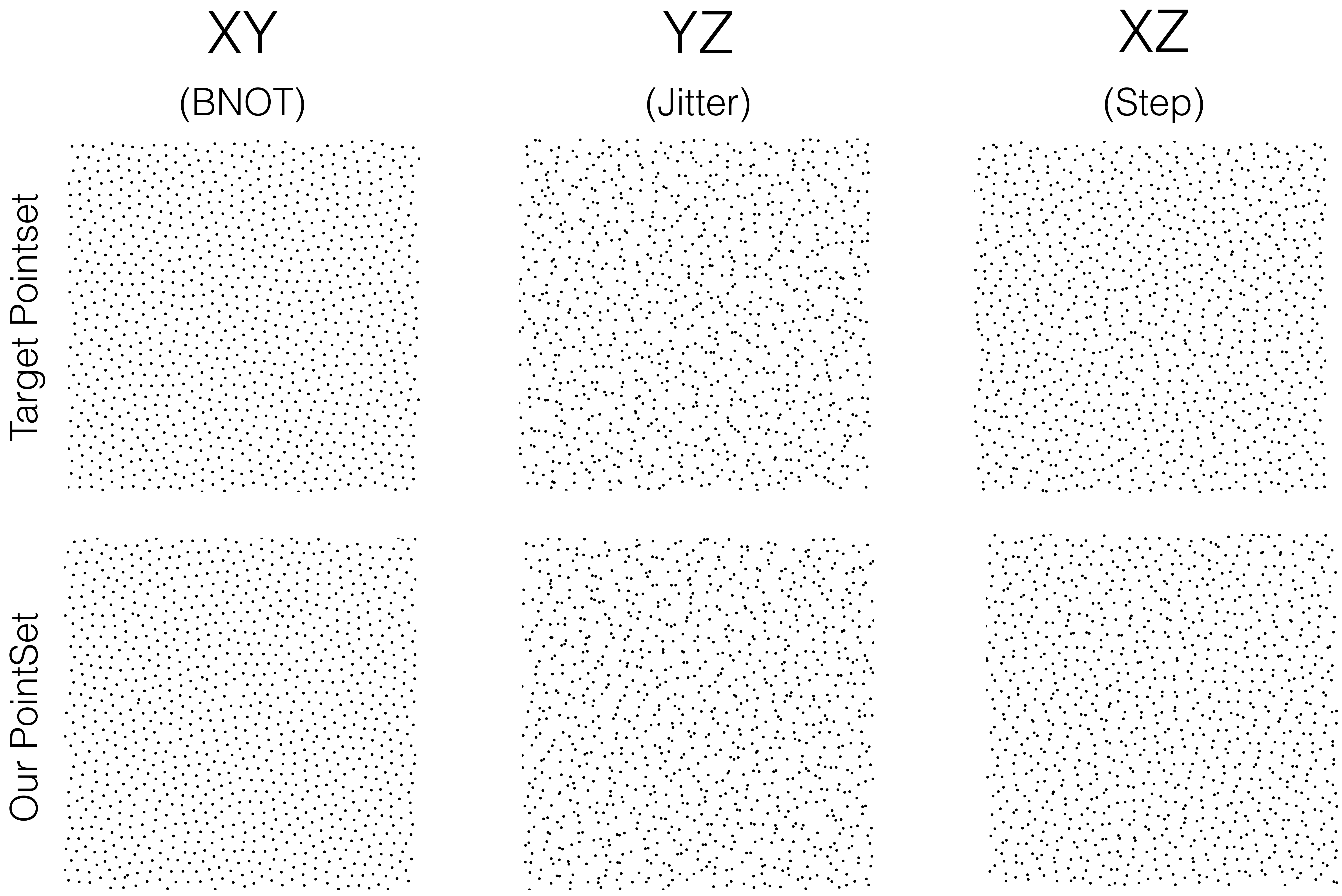
3D Point Samples (Different Projection Targets)





Point set: Projections are Preserved

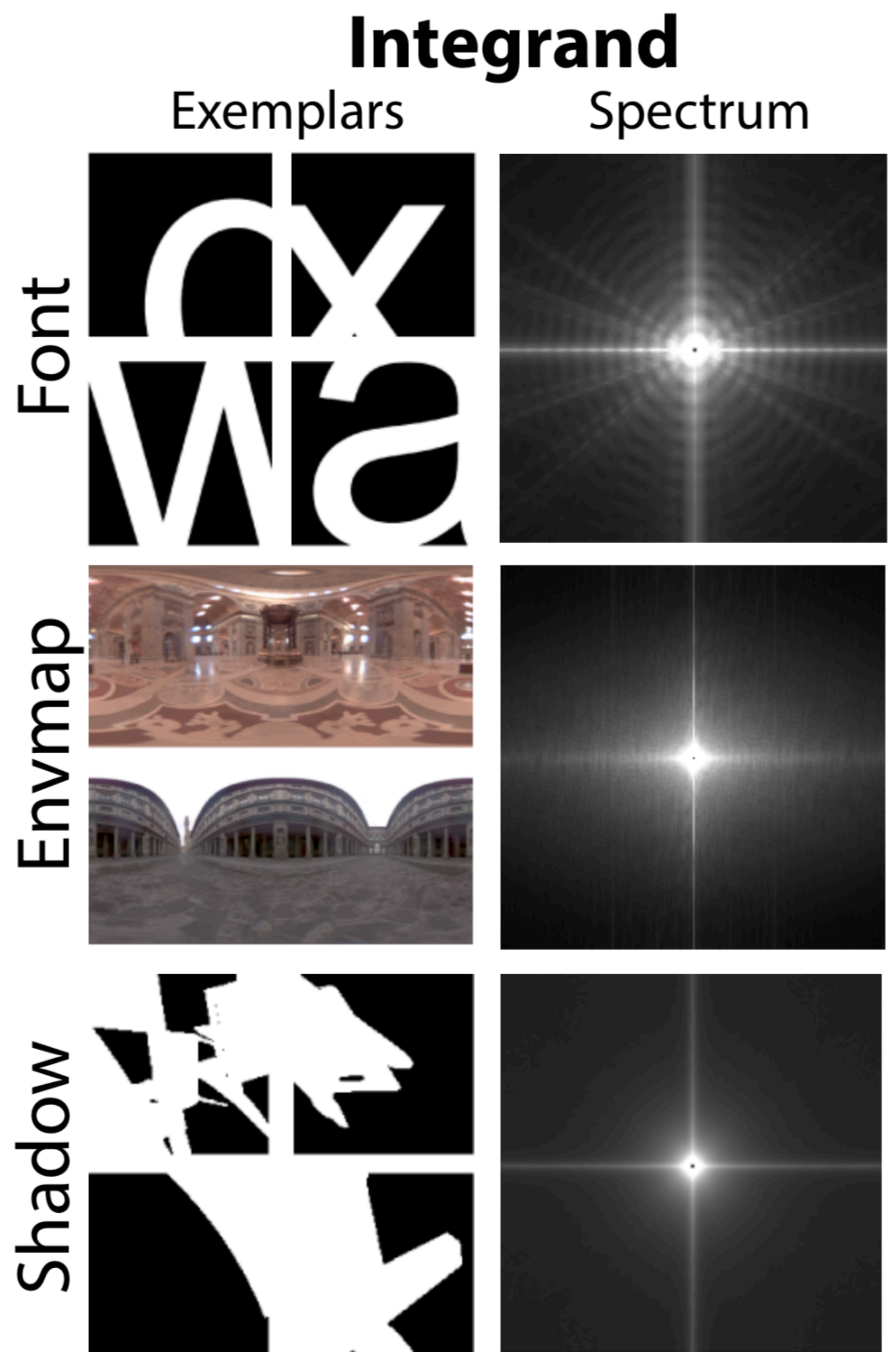




Novel Sampling Patterns

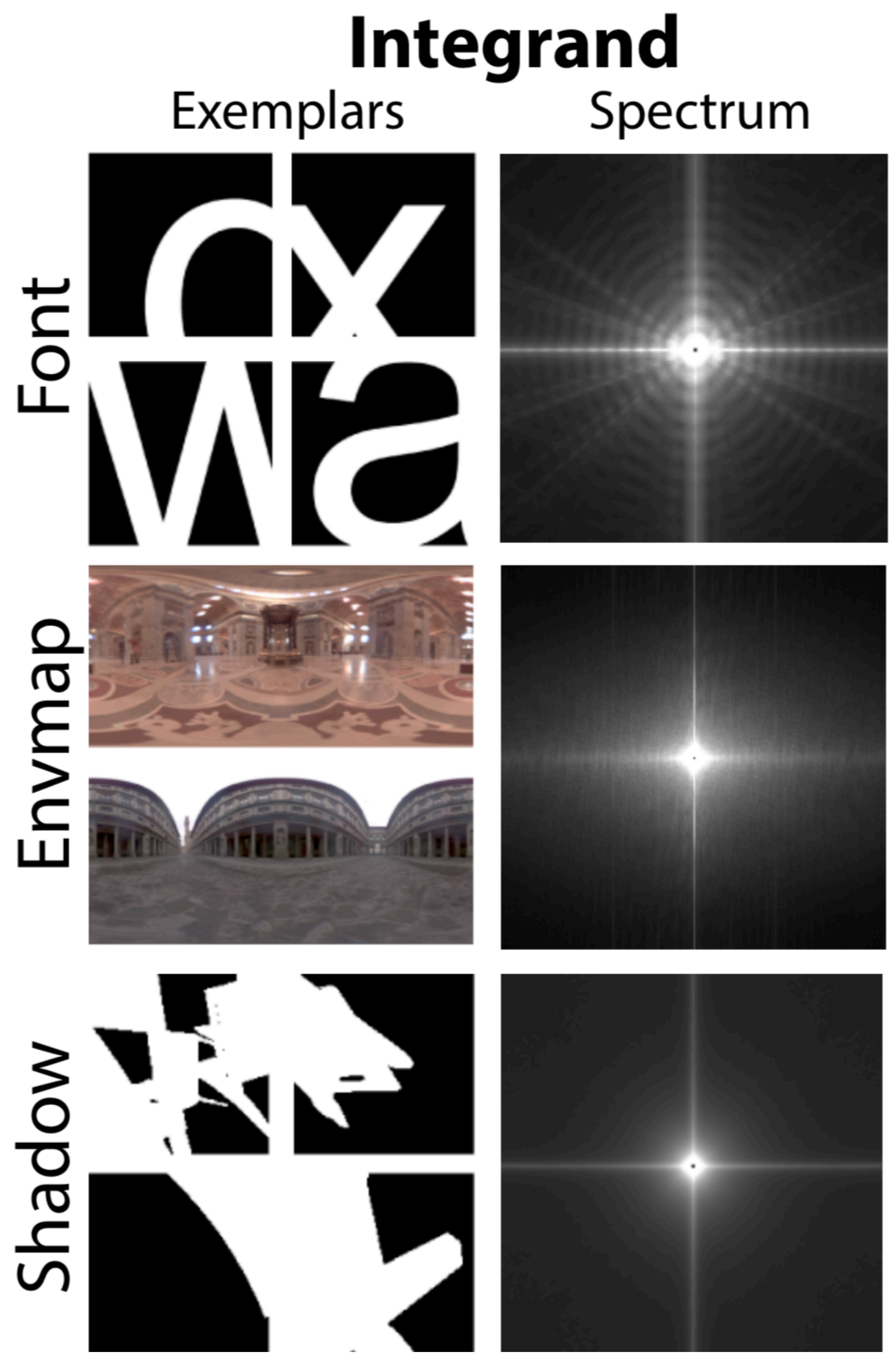
Leimkuhler et al. [SIGGRAPH Asia 2019]

Novel Sampling Patterns

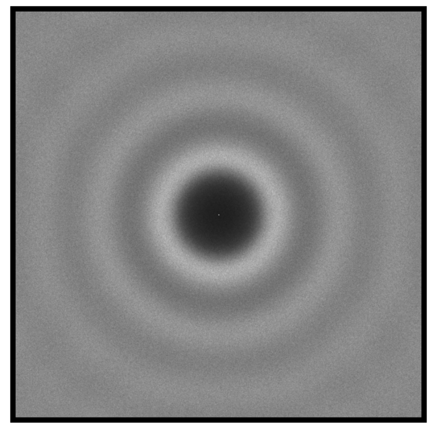
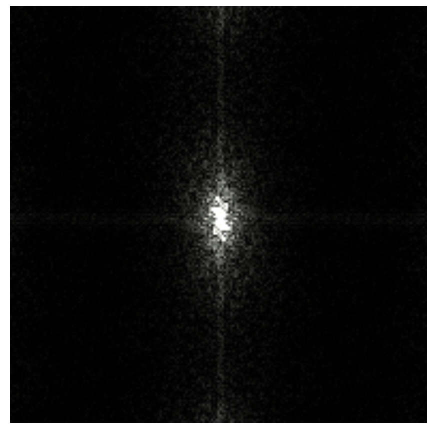


Leimkuhler et al. [SIGGRAPH Asia 2019]

Novel Sampling Patterns



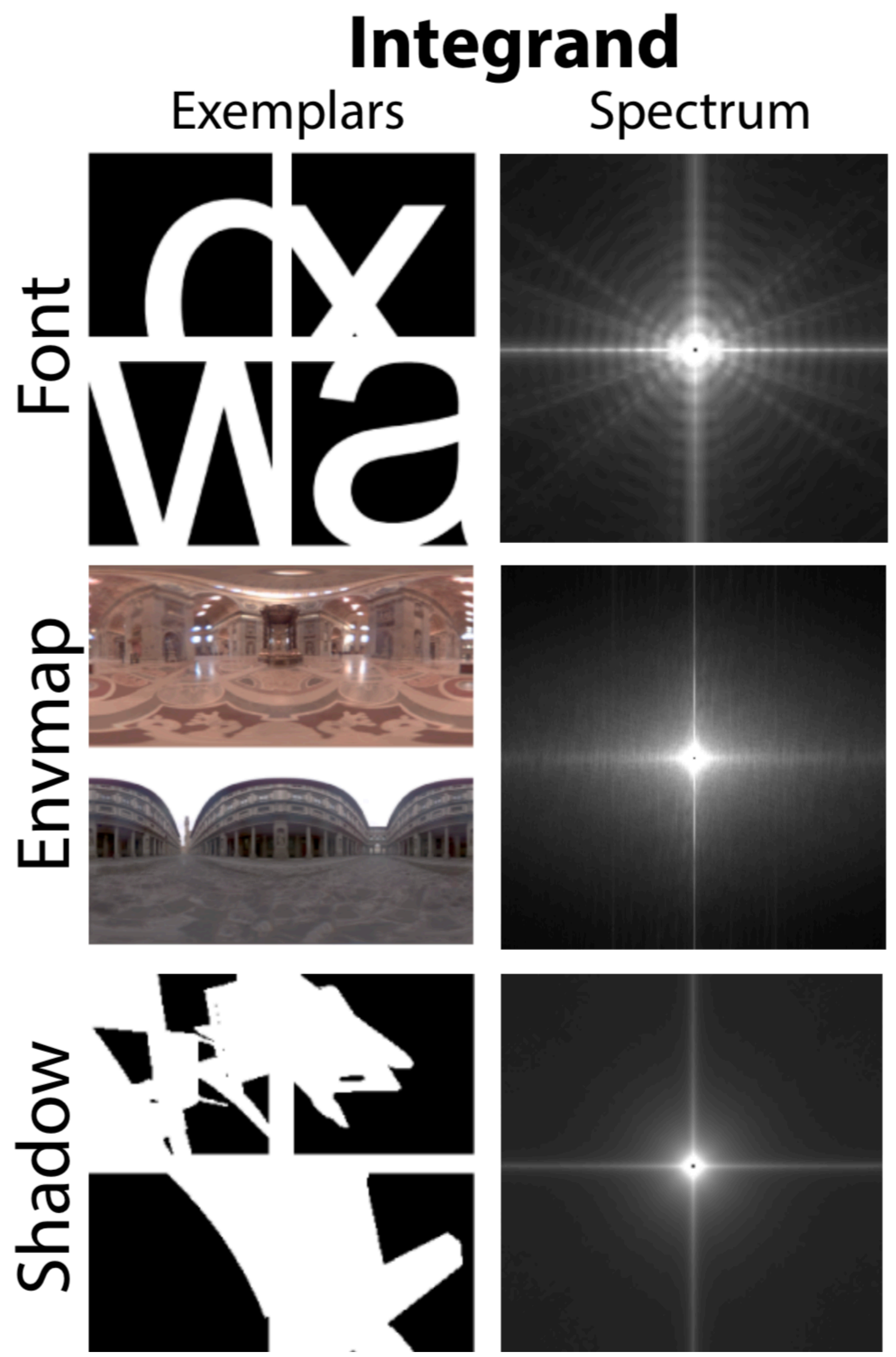
Integrand
Power spectrum

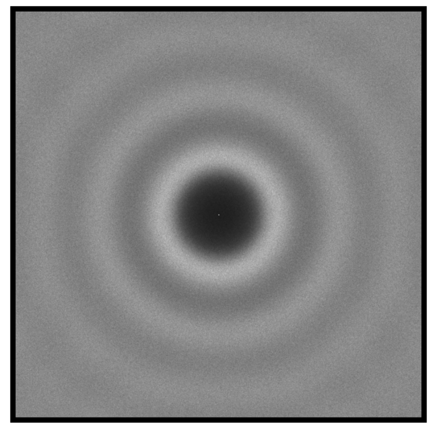
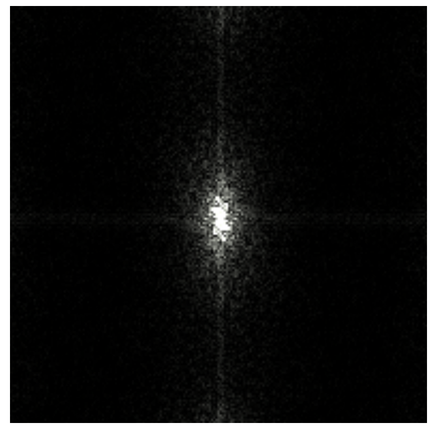
$$\mathrm{Var}(I_N) = \sum_{\Omega}$$  $\times$ 

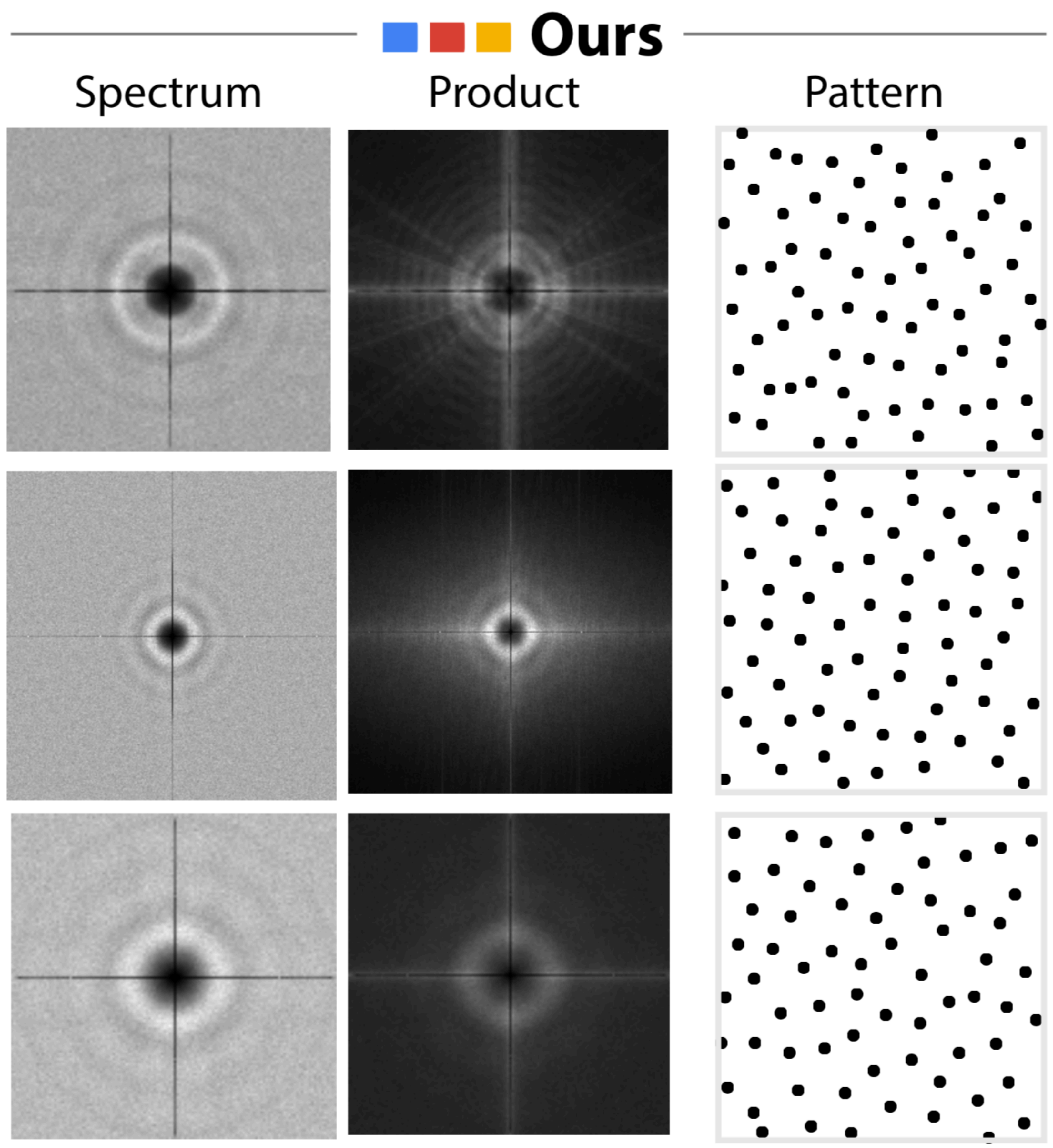
Sampling
Power spectrum

Leimkuhler et al. [SIGGRAPH Asia 2019]

Novel Sampling Patterns

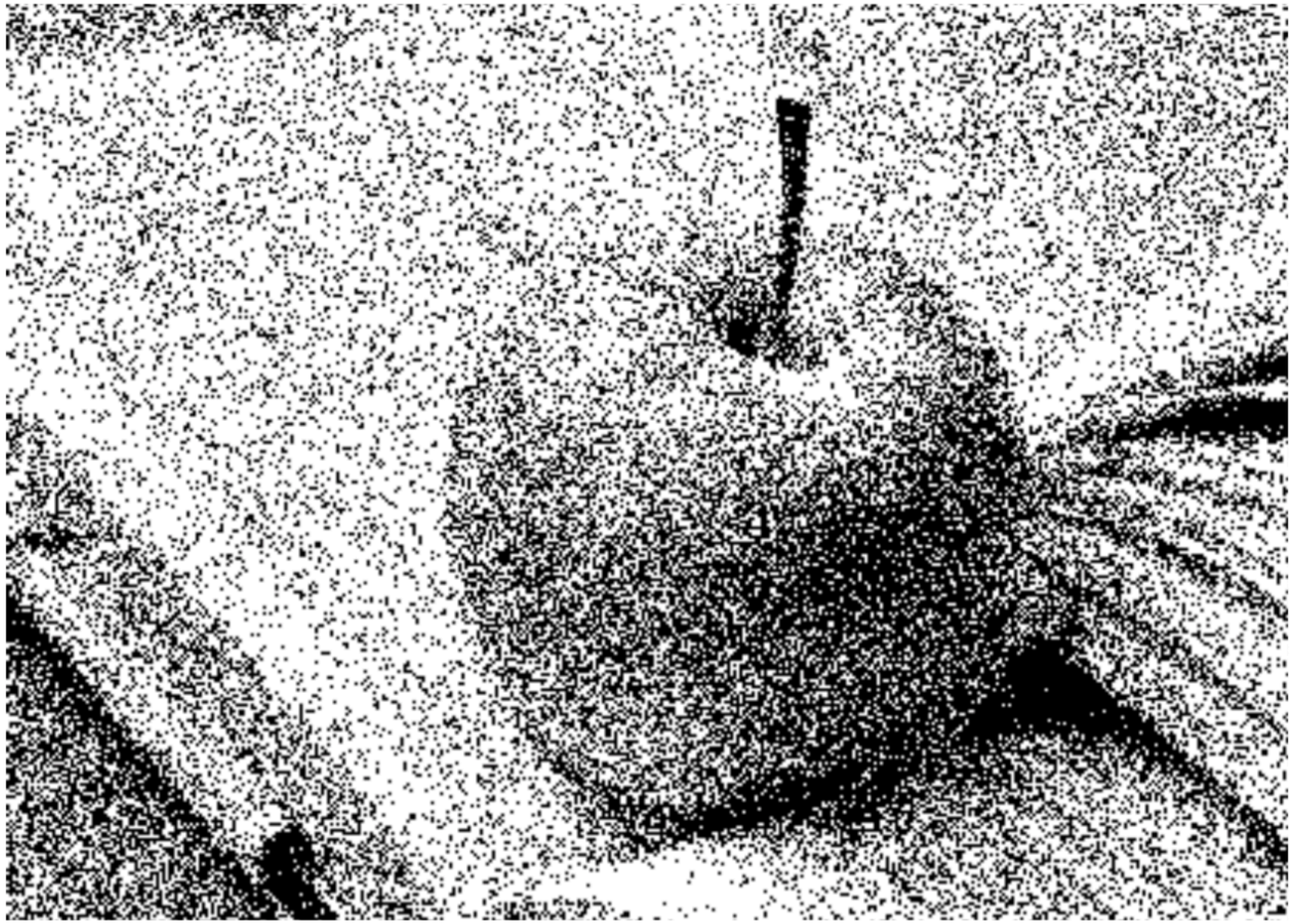


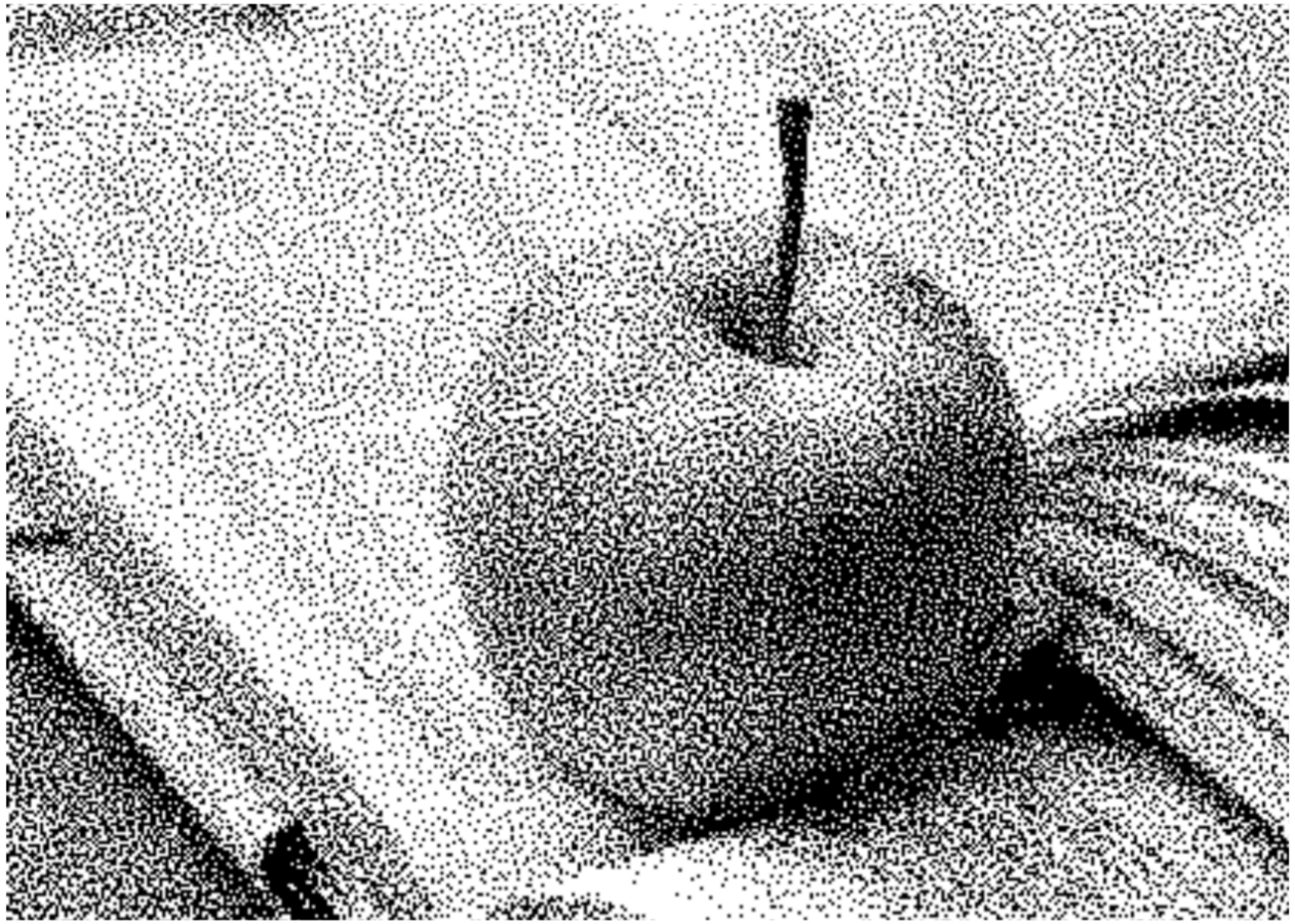
$$\mathrm{Var}(I_N) = \sum_{\Omega}$$  $\times$ 



Leimkuhler et al. [SIGGRAPH Asia 2019]

Blue Noise Dithering

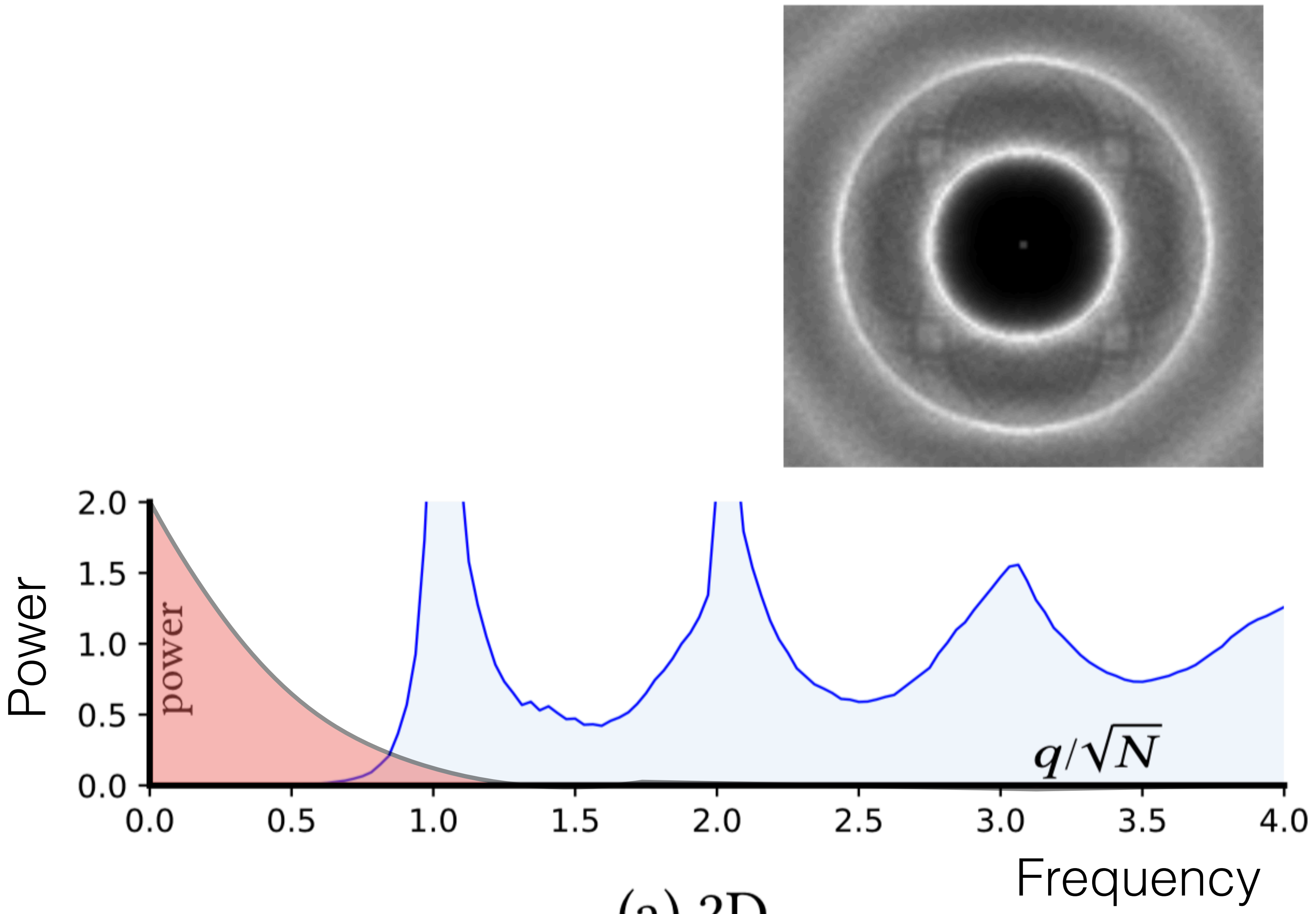




Leimkuhler et al. [SIGGRAPH Asia 2019]

Novel Sampling Patterns

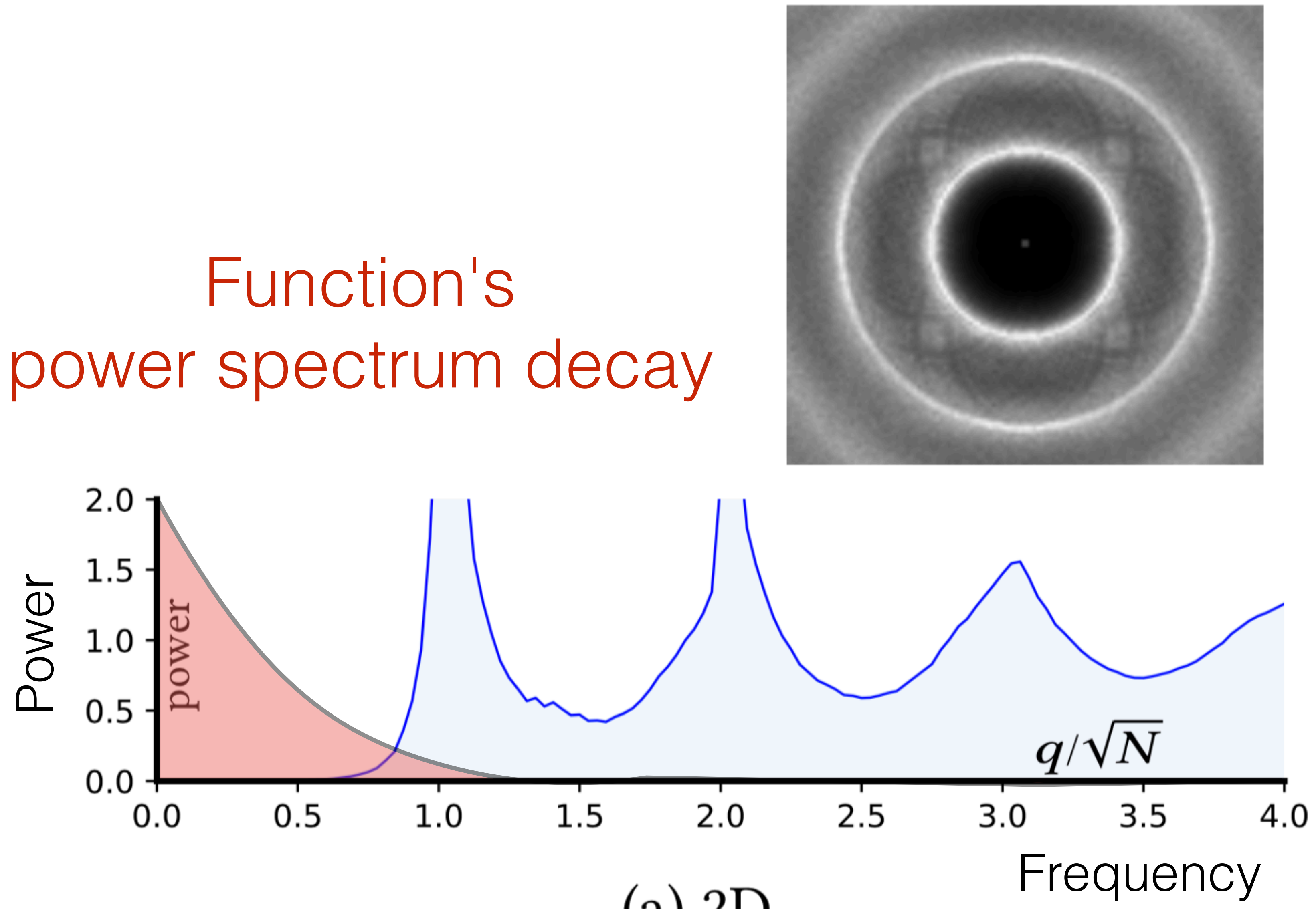
using radially averaged loss



(a) 2D

Novel Sampling Patterns

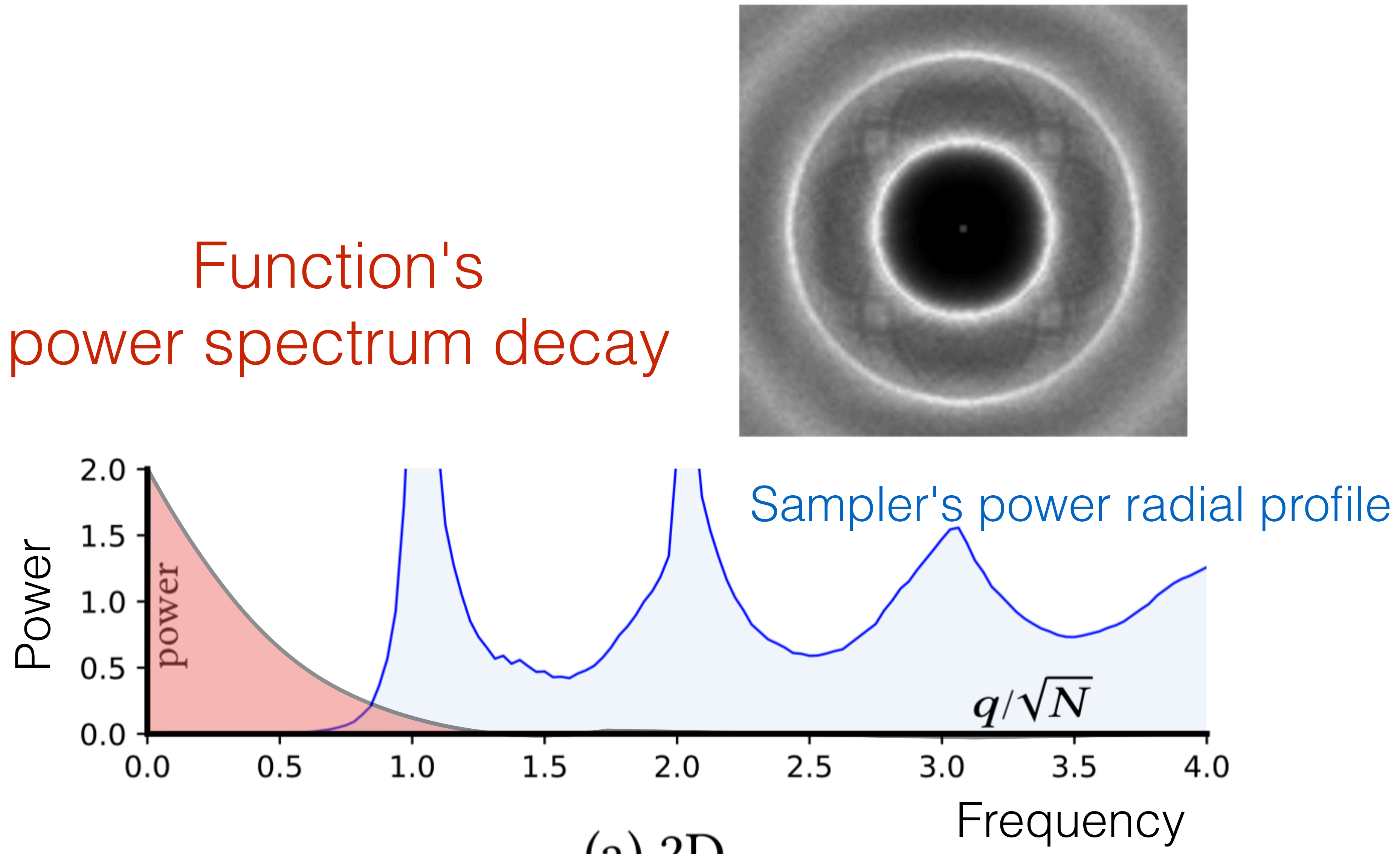
using radially averaged loss



(a) 2D

Novel Sampling Patterns

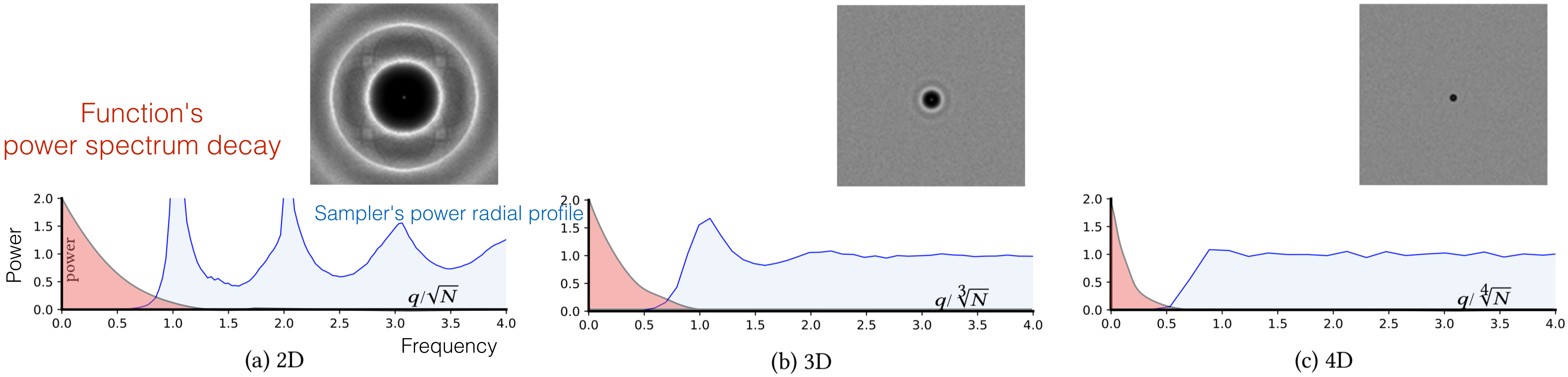
using radially averaged loss



(a) 2D

Novel Sampling Patterns

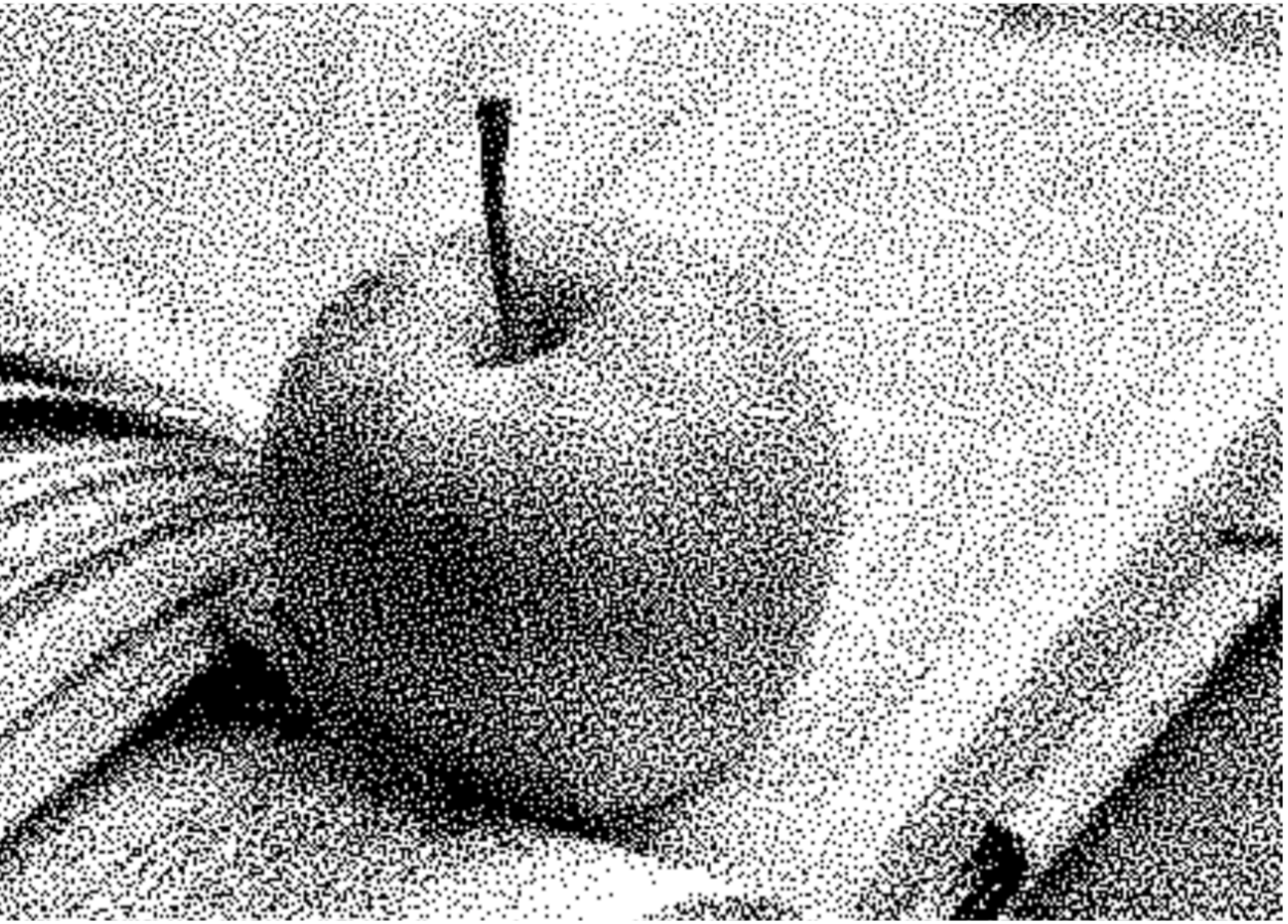
using radially averaged loss

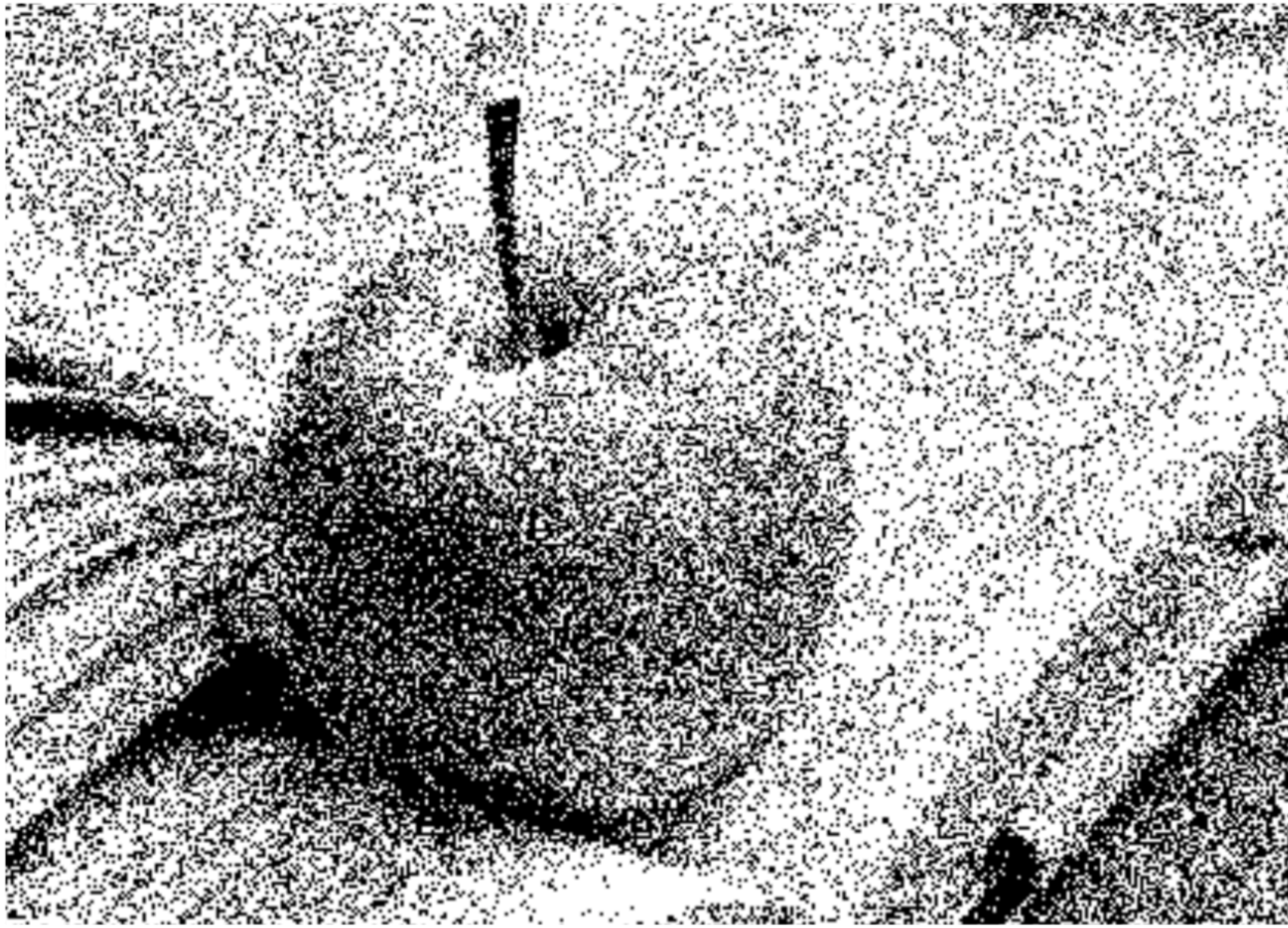


Object Placement

Object Placement

Blue Noise Dithering



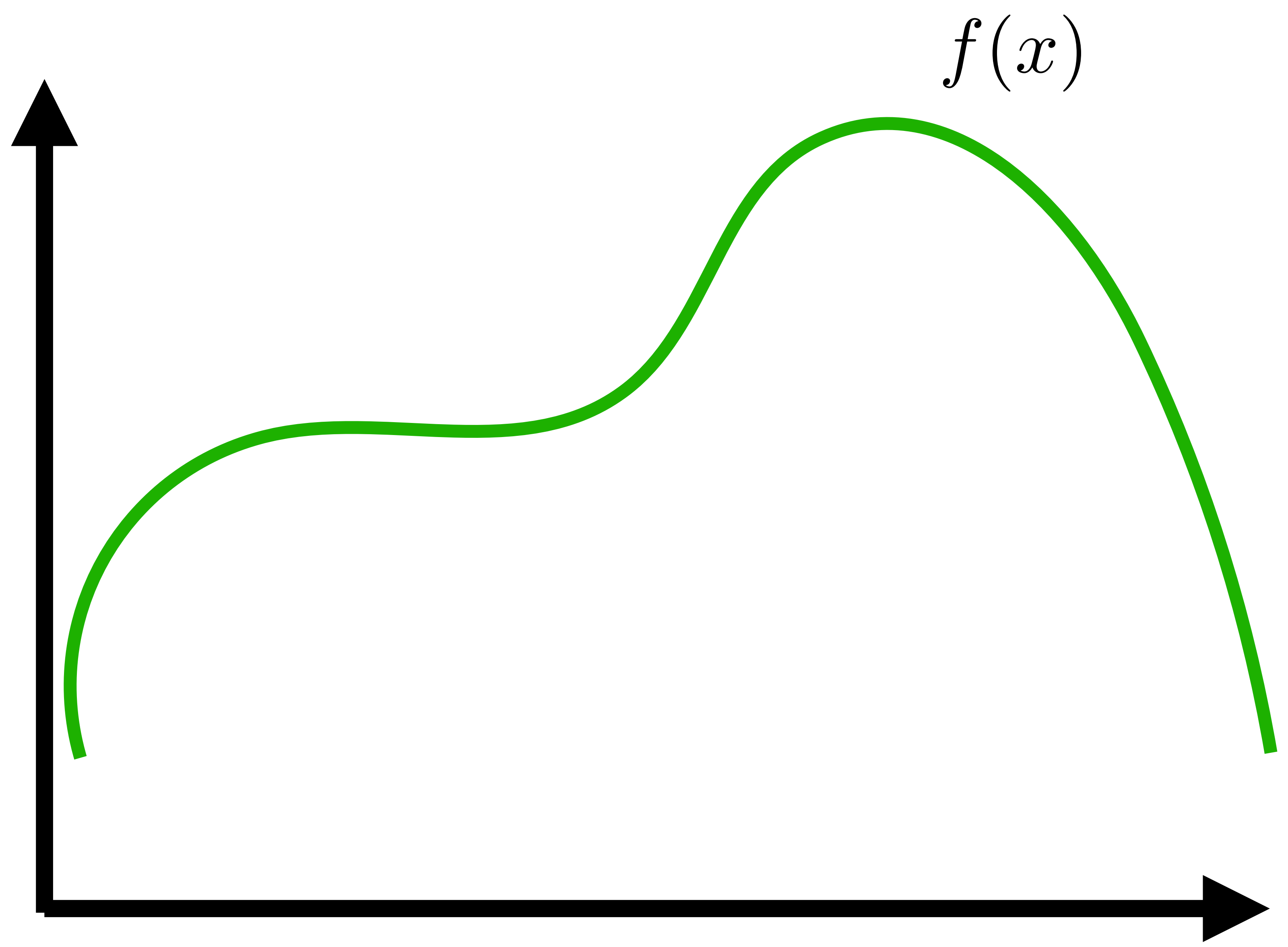


Normalizing Flows

Importance Sampling

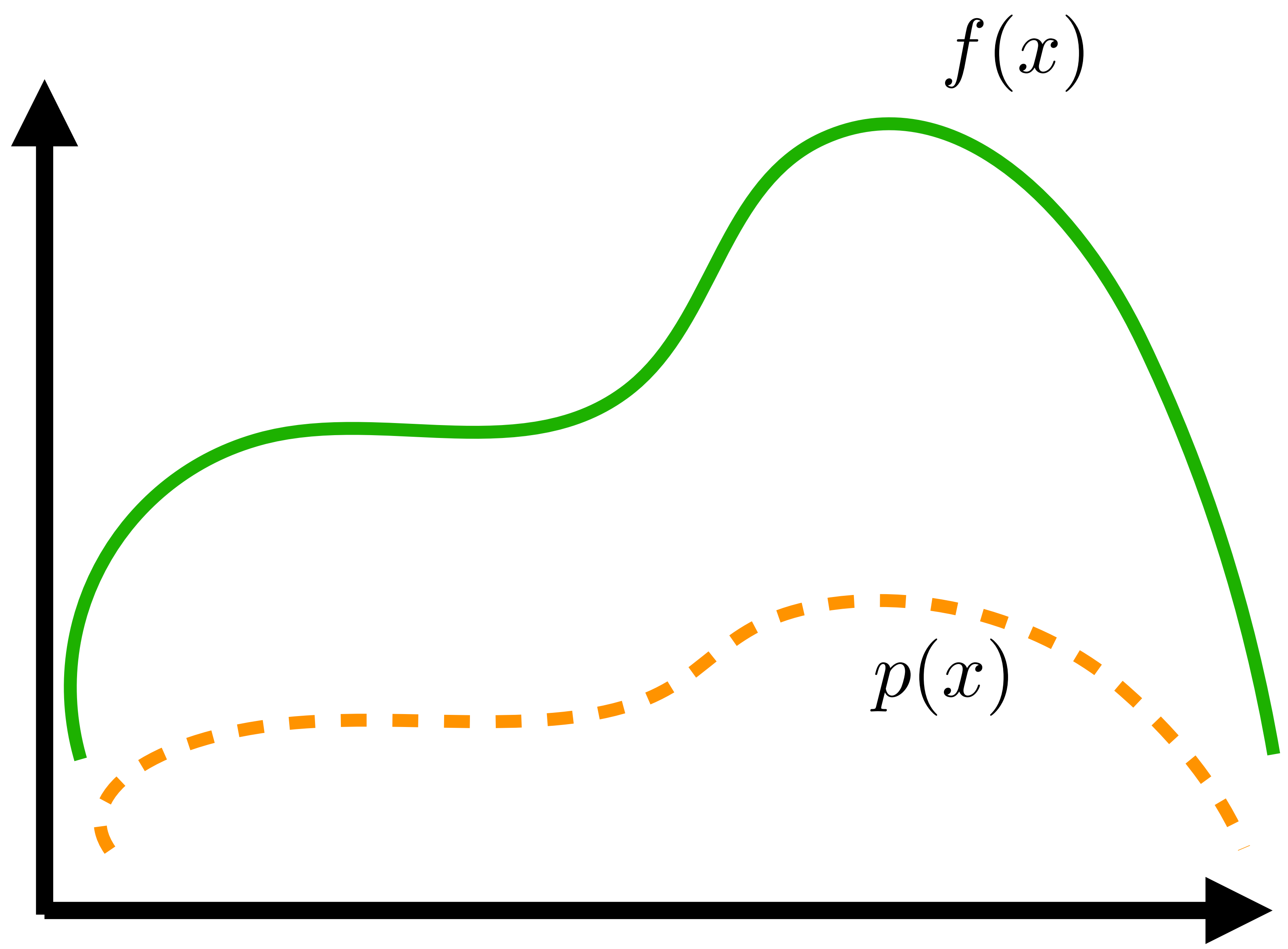
$$I_N = \frac{1}{N}\sum_{k=1}^{N}\frac{f(x)}{p(x)}$$

Importance Sampling



$$I_N = \frac{1}{N} \sum_{k=1}^{N} \frac{f(x)}{p(x)}$$

Importance Sampling



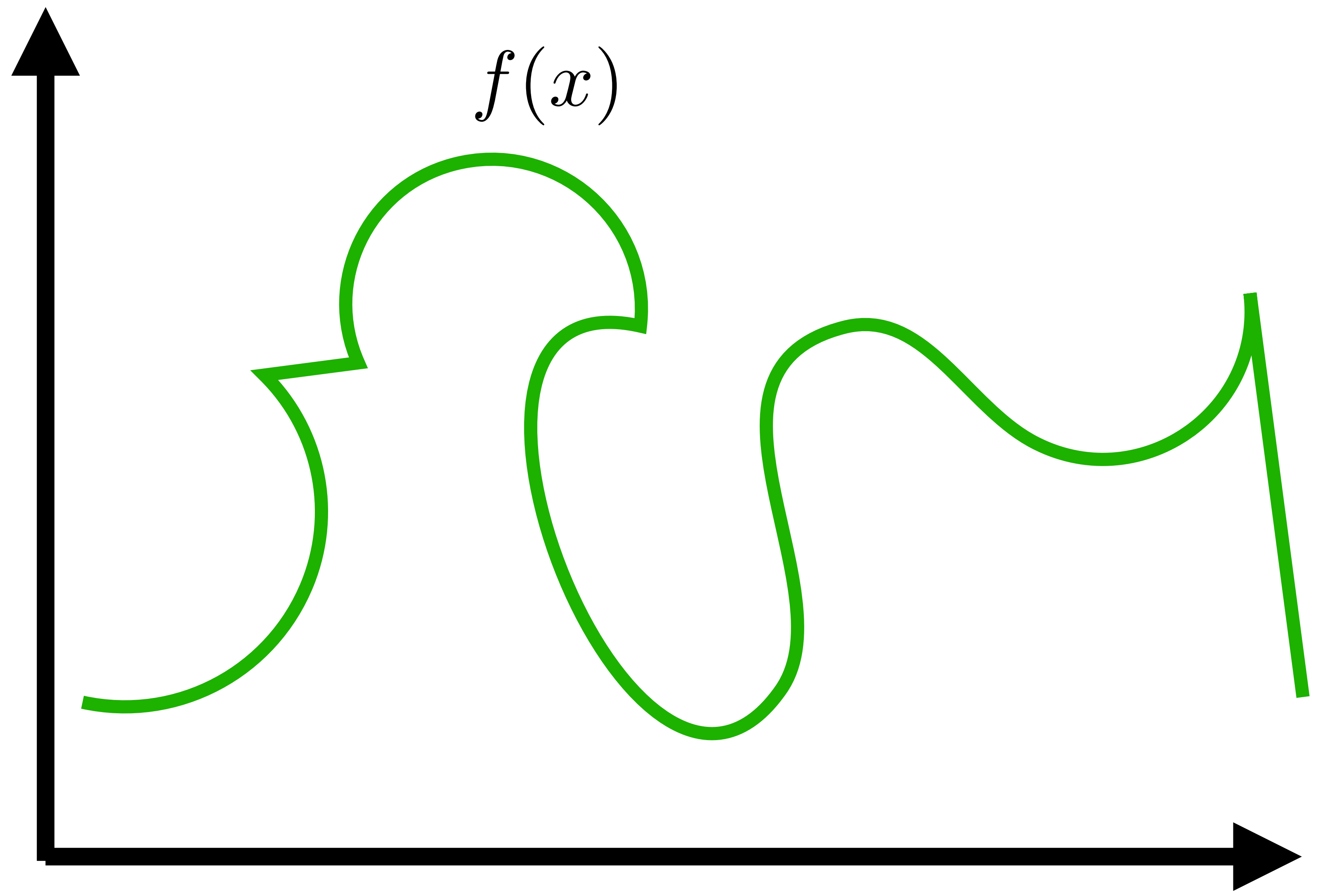
$$I_N = \frac{1}{N} \sum_{k=1}^{N} \frac{f(x)}{p(x)}$$

Importance Sampling

$$I_N = \frac{1}{N}\sum_{k=1}^{N}\frac{f(x)}{p(x)}$$

$$p(x) = ???$$

Importance Sampling



$$I_N = \frac{1}{N} \sum_{k=1}^{N} \frac{f(x)}{p(x)}$$

$$p(x) = ???$$

Normalizing Flows

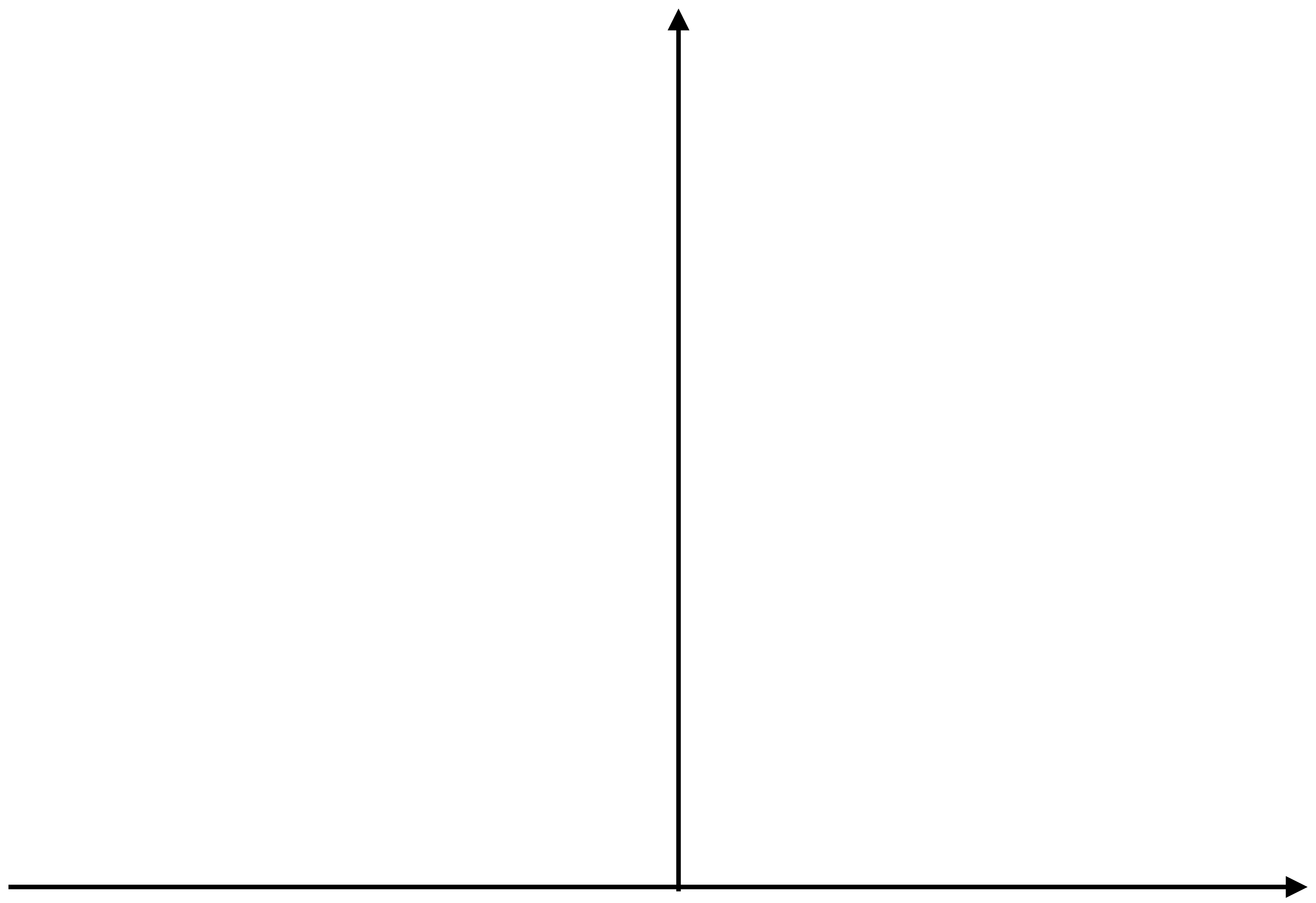
Normalizing Flows

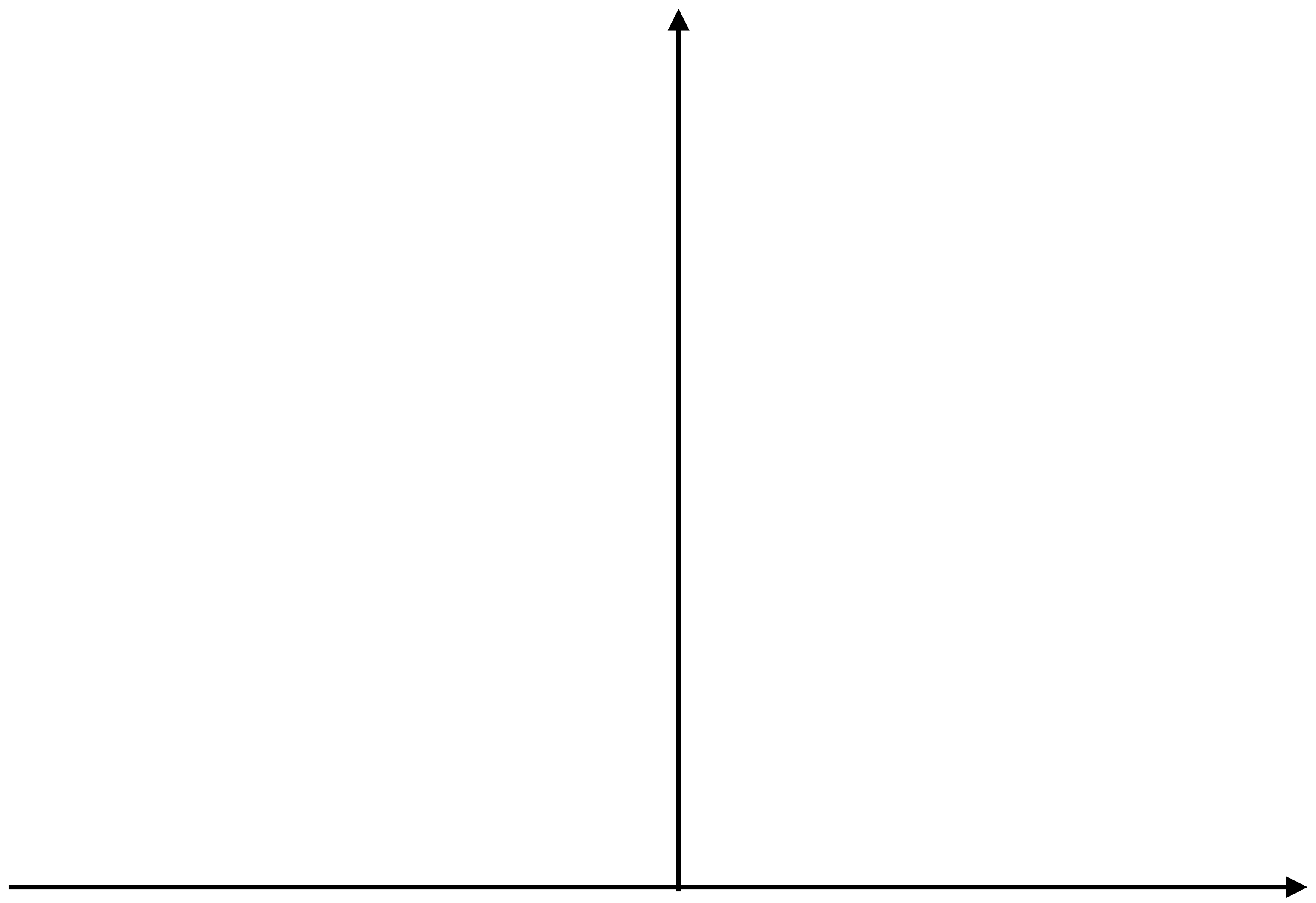
Technique used in Machine learning to build complex probability distributions by transforming simple ones

Used in the context of generative modeling

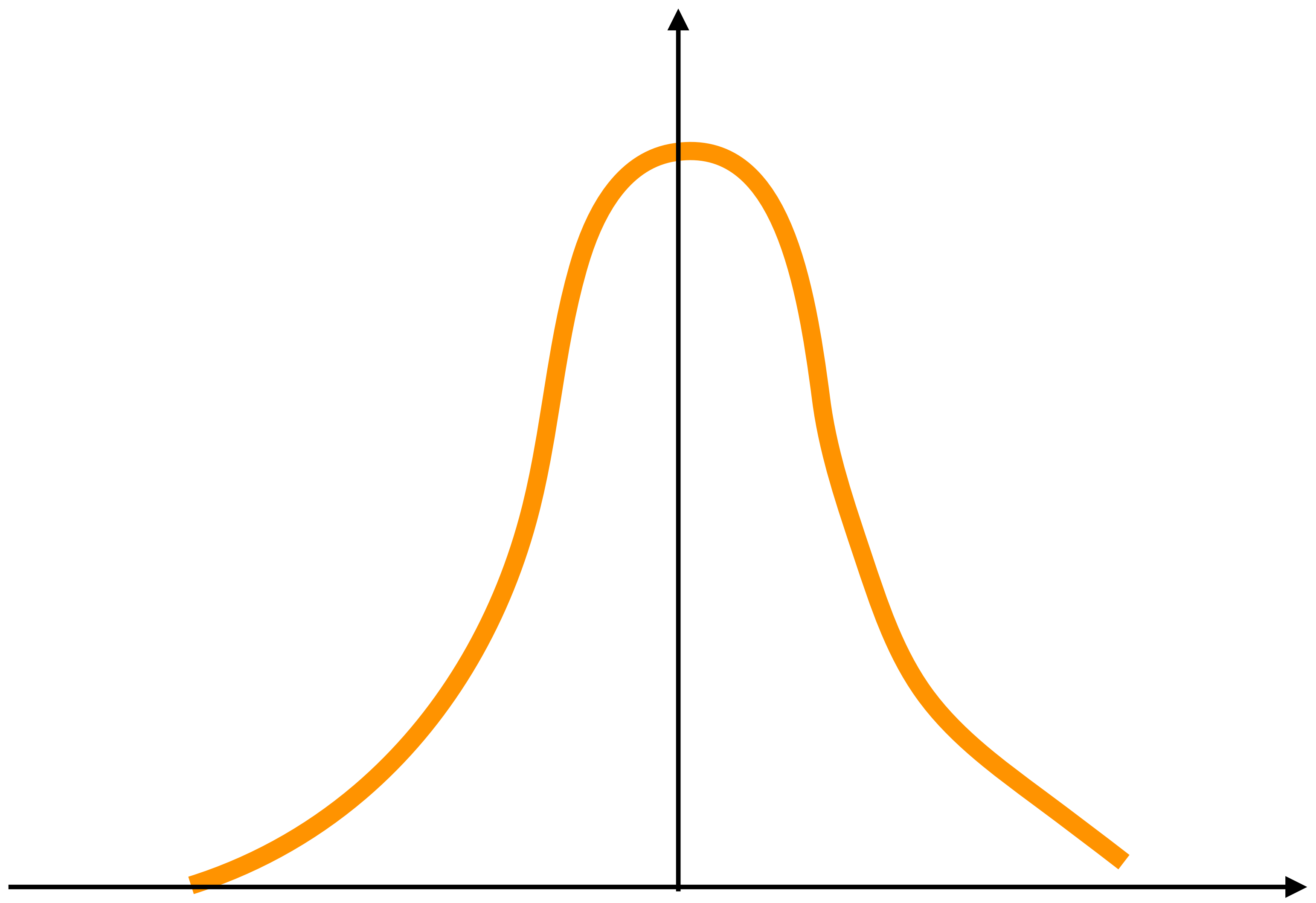
Generative modeling: learning without any target (unsupervised)

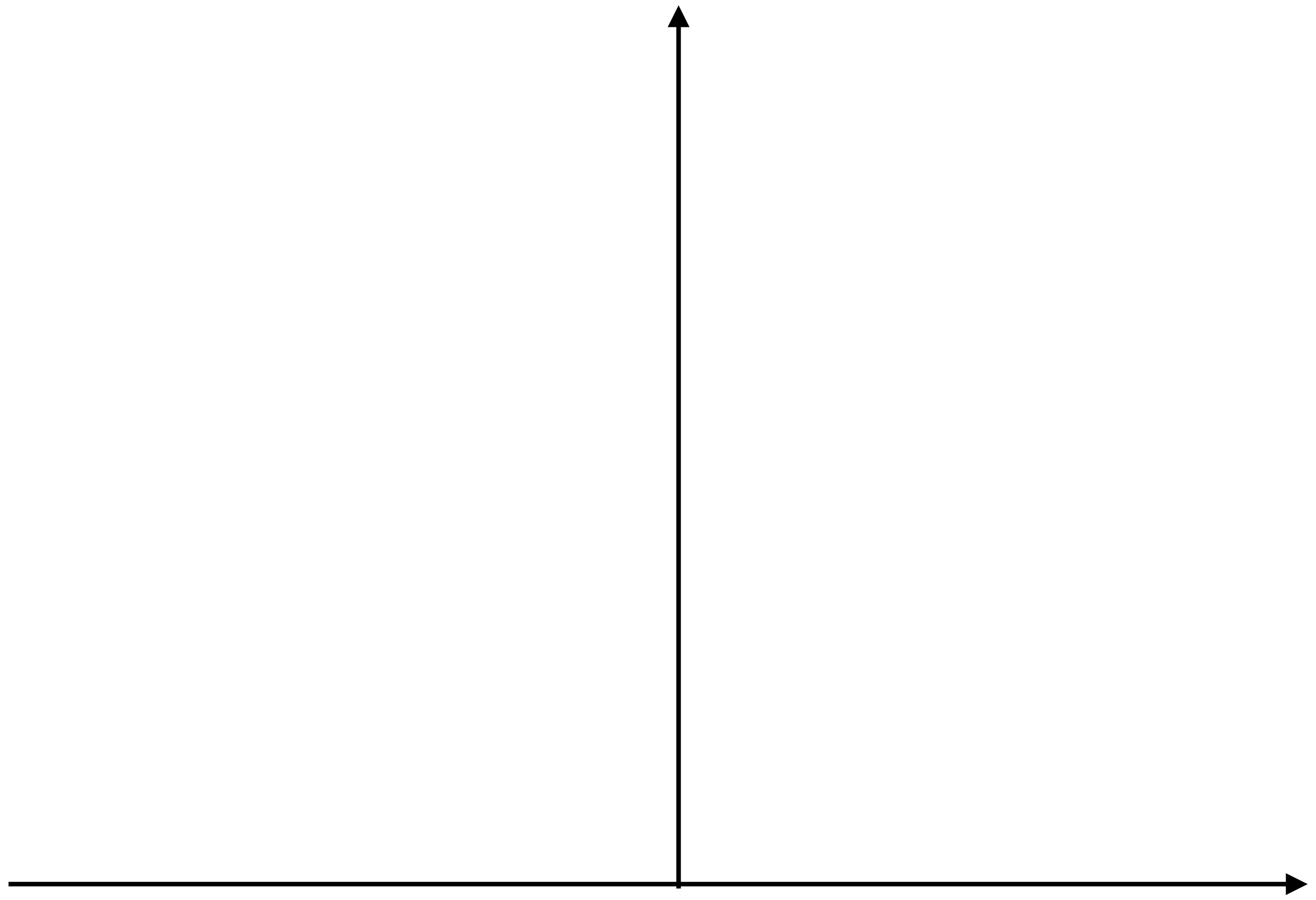
Complex Probability distributions from simple ones



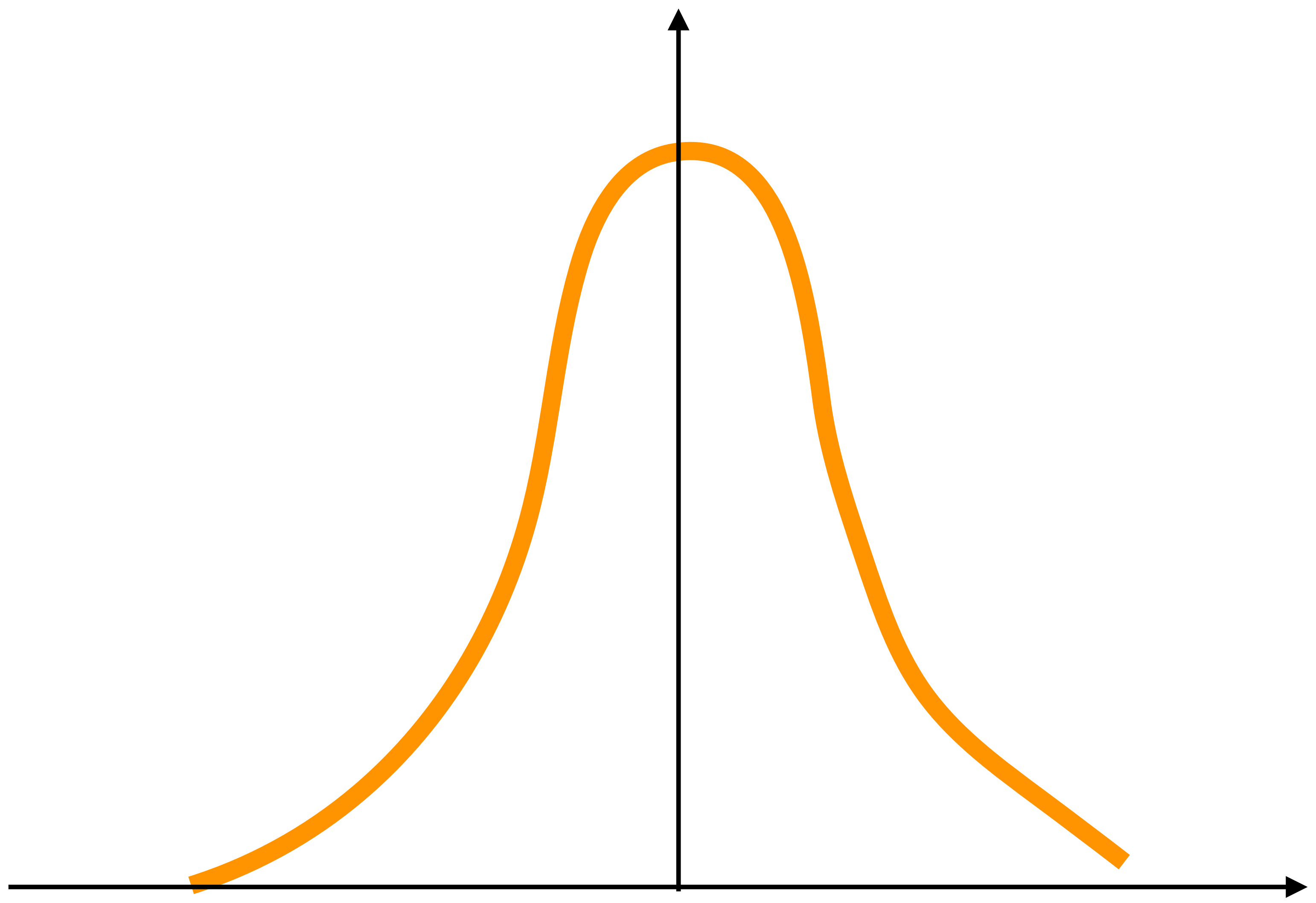


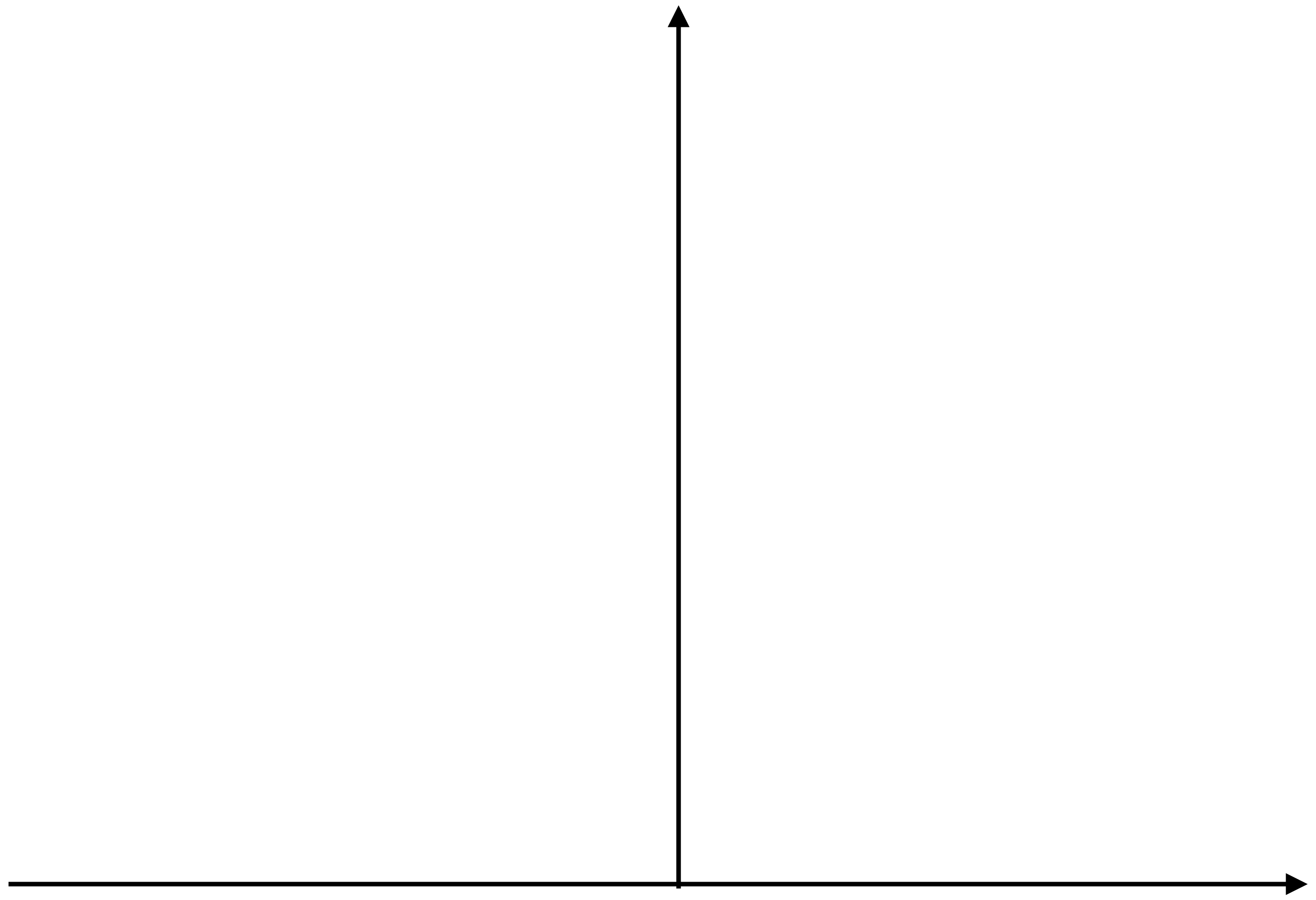
Complex Probability distributions from simple ones



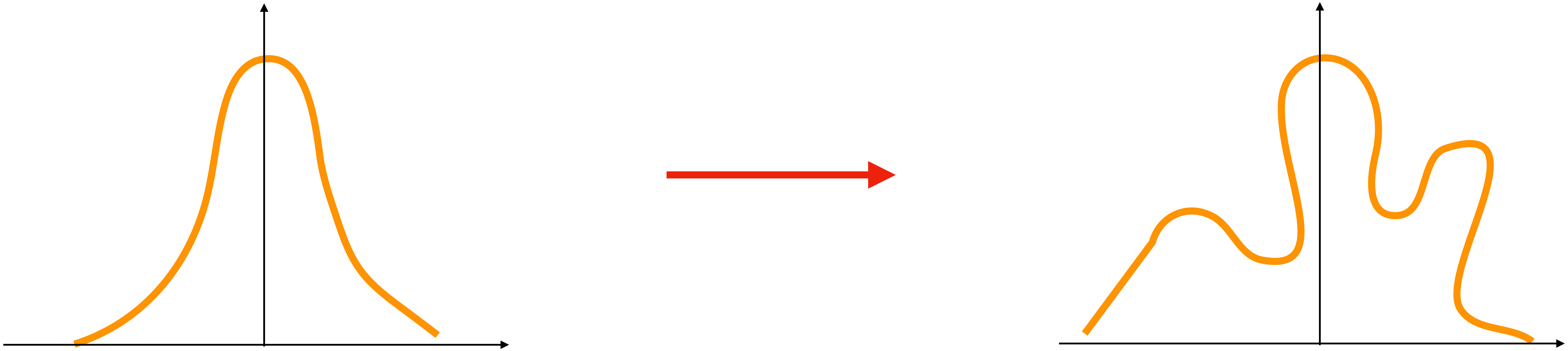


Complex Probability distributions from simple ones

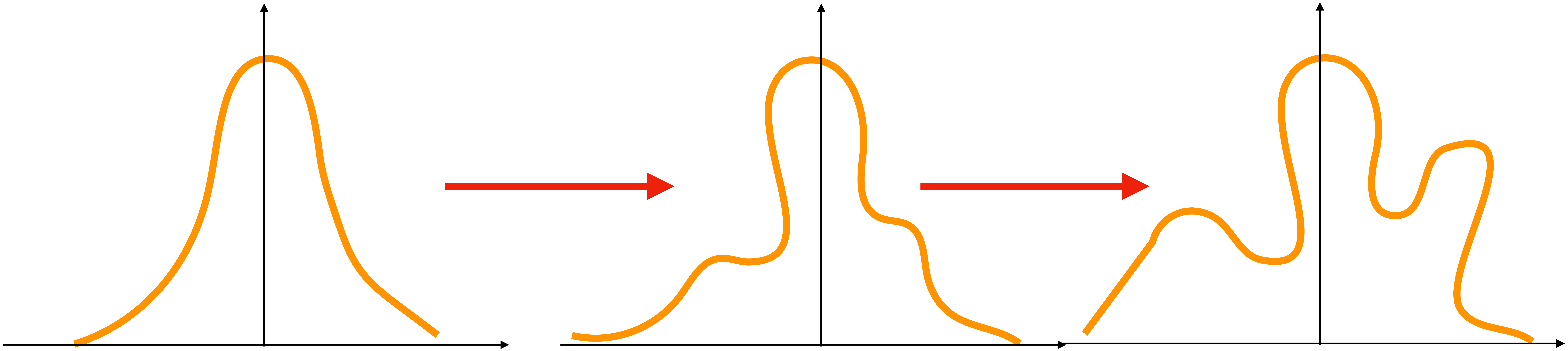




Complex Probability distributions from simple ones



Complex Probability distributions from simple ones



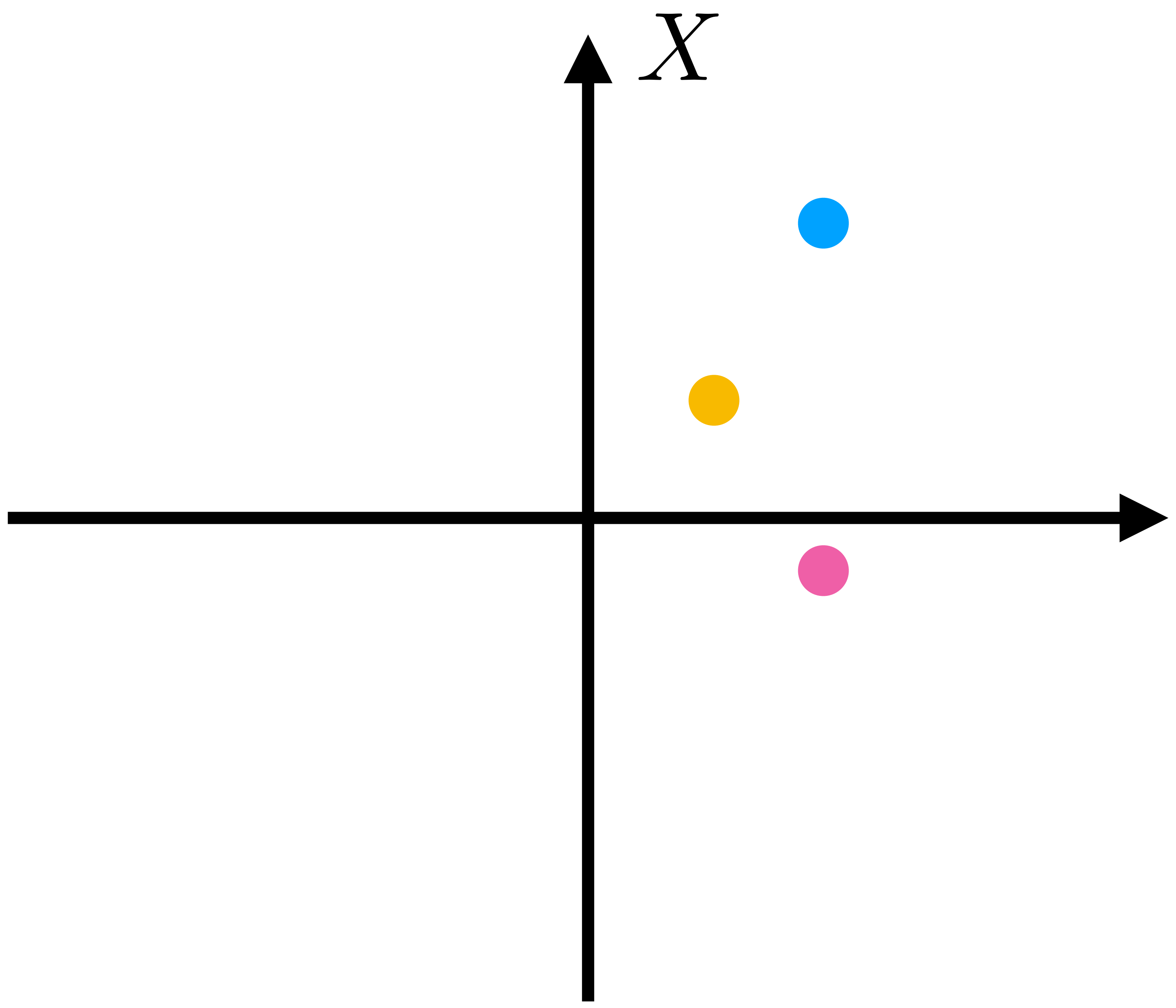
Normalizing Flows:
Basic mathematical framework

$z \sim p_\theta(z)$ Given a continuous variable with a distribution

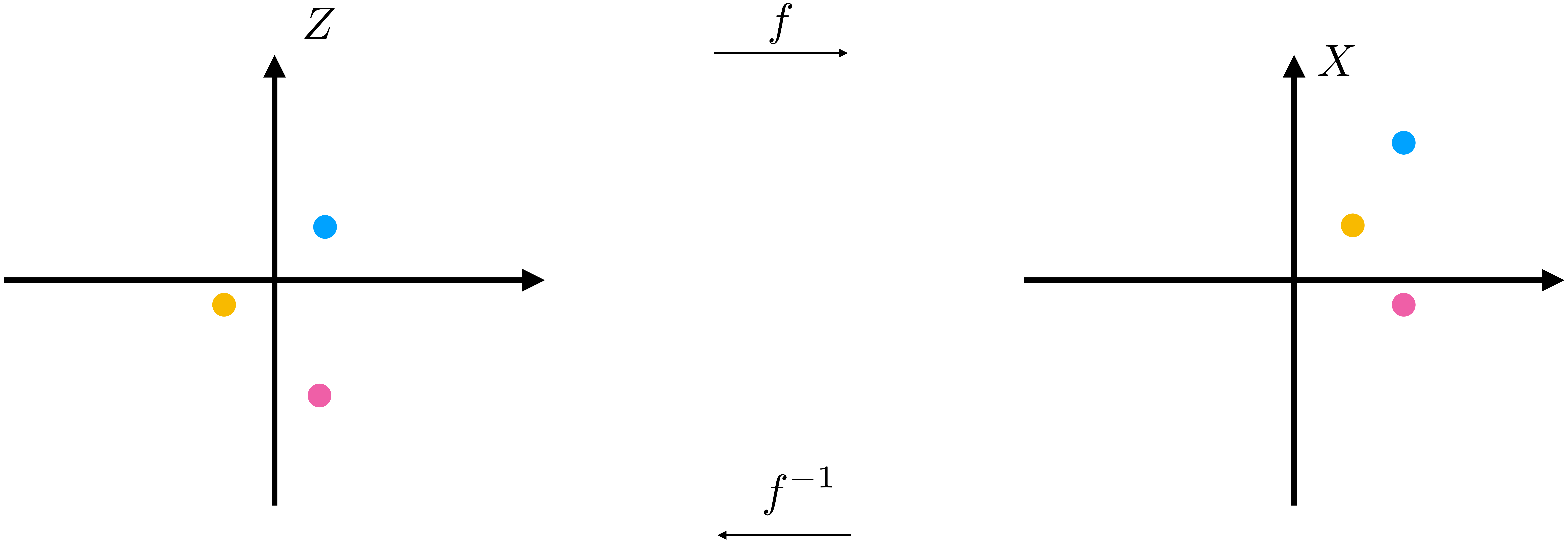
$x = f_\theta(z) = f_k \circ \circ \circ f_2 \circ f_1(z)$ New distribution obtained

each f_i is invertible (bijective)

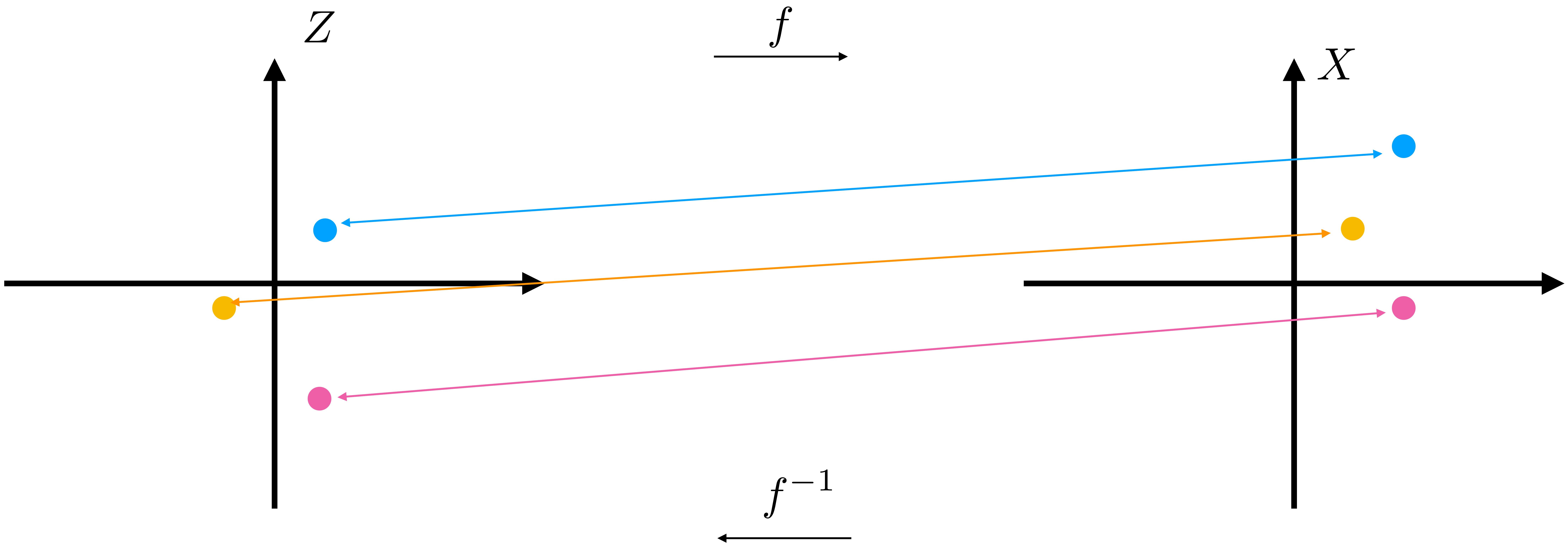
Distributions



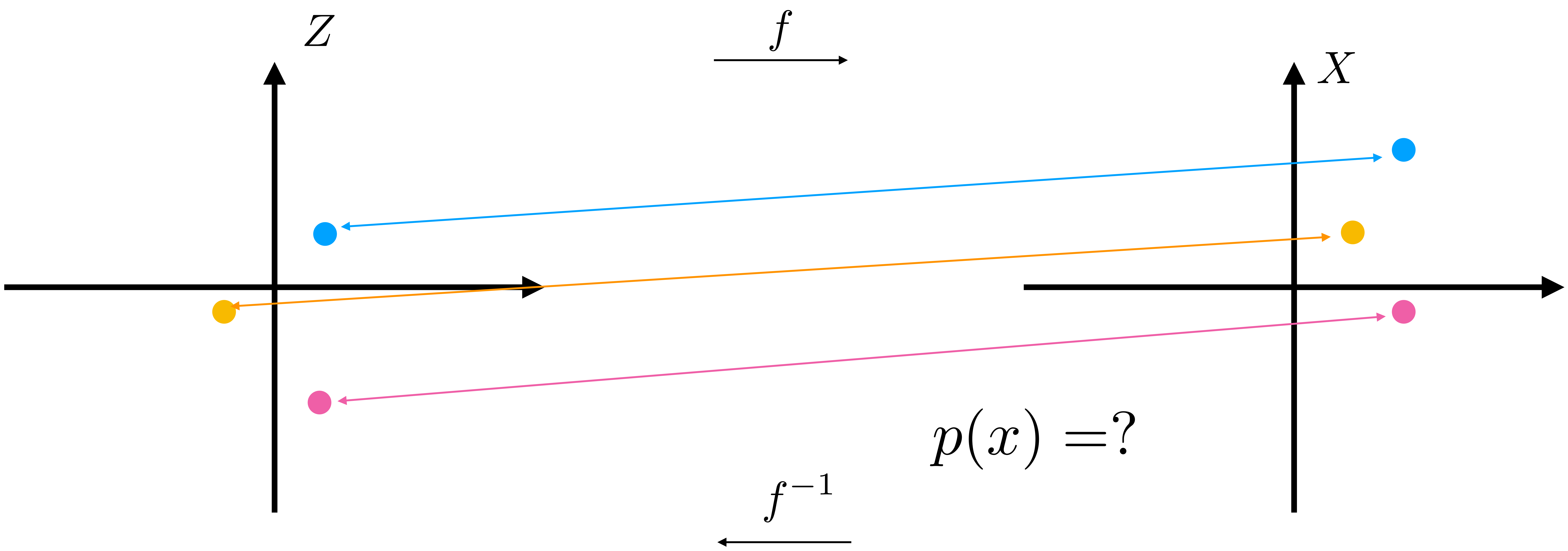
Distributions



Distributions



Distributions



Distributions

$z \sim p_\theta(z)$ Given a continuous variable with a distribution

$$x = f_\theta(z) = f_k \circ \circ \circ f_2 \circ f_1(z)$$

each f_i is invertible (bijective)

$$p(x) = p(f^{-1}(x))$$

Distributions

$z \sim p_\theta(z)$ Given a continuous variable with a distribution

$$x = f_\theta(z) = f_k \circ \circ \circ f_2 \circ f_1(z)$$

$$\text{each } f_i \text{ is invertible (bijective)}$$

$$p(x) \cancel{=} p(f^{-1}(x))$$

Change of Variables

$$f : Z \to X, f \text{ is invertible}$$
$$p(z) \text{ defined over } z \in Z$$

Change of Variables

$$f : Z \to X, f \text{ is invertible}$$
$$p(z) \text{ defined over } z \in Z$$

Change of variable formula says that:

Change of Variables

$$f : Z \to X, f \text{ is invertible}$$
$$p(z) \text{ defined over } z \in Z$$

Change of variable formula says that:

$$p(x) = p(f^{-1}(x)) \left| det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right|$$

Change of Variables

$f : Z \rightarrow X, f$ is invertible

$p(z)$ defined over $z \in Z$

$$p(x) = p(f^{-1}(x)) \left| det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right|$$

$$p(x) = p(z) \left| det \left(\frac{\partial z}{\partial x} \right) \right|$$

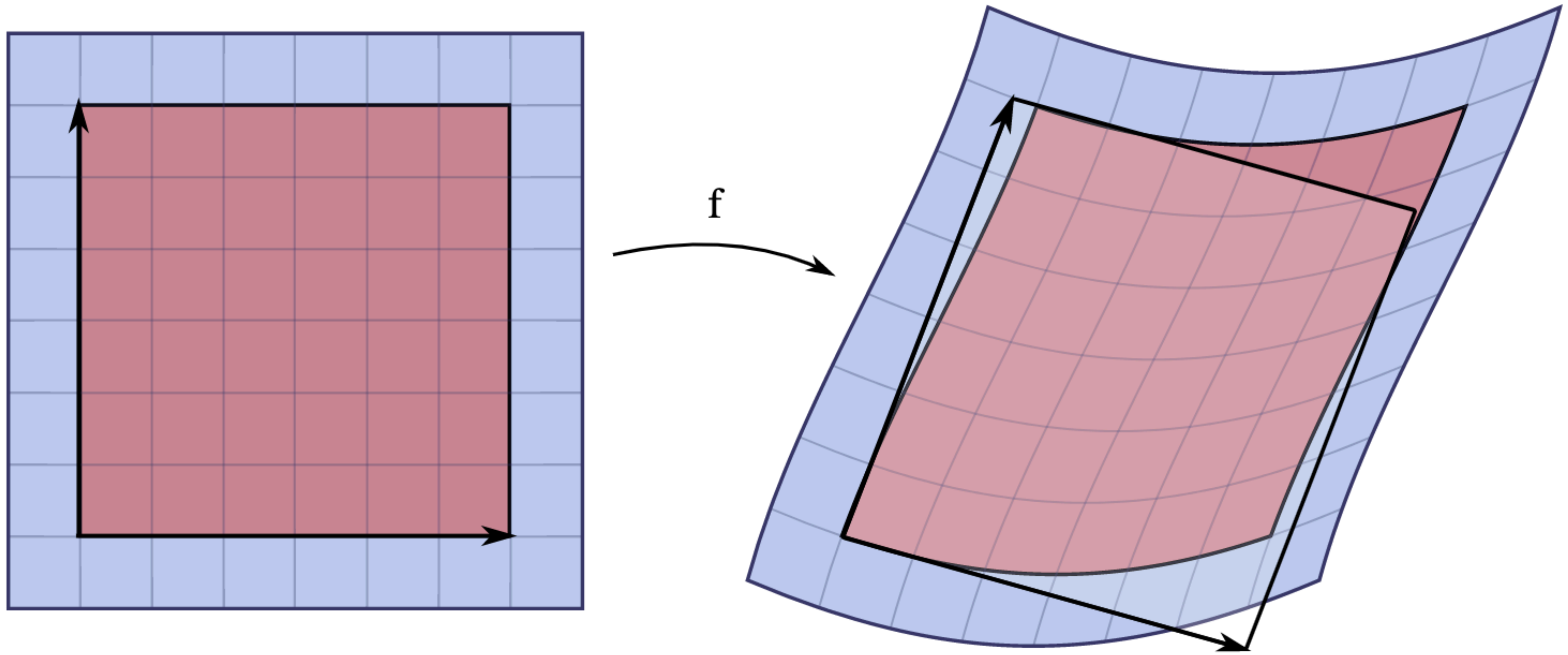
Jacobian Matrix

$$\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$$

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Jacobian Matrix

$$\mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^2$$



Jacobian determinant gives the ratio of the area of the approximating parallelogram to that of the original square.

Jacobian Matrix

$f : Z \to X, f$ is invertible

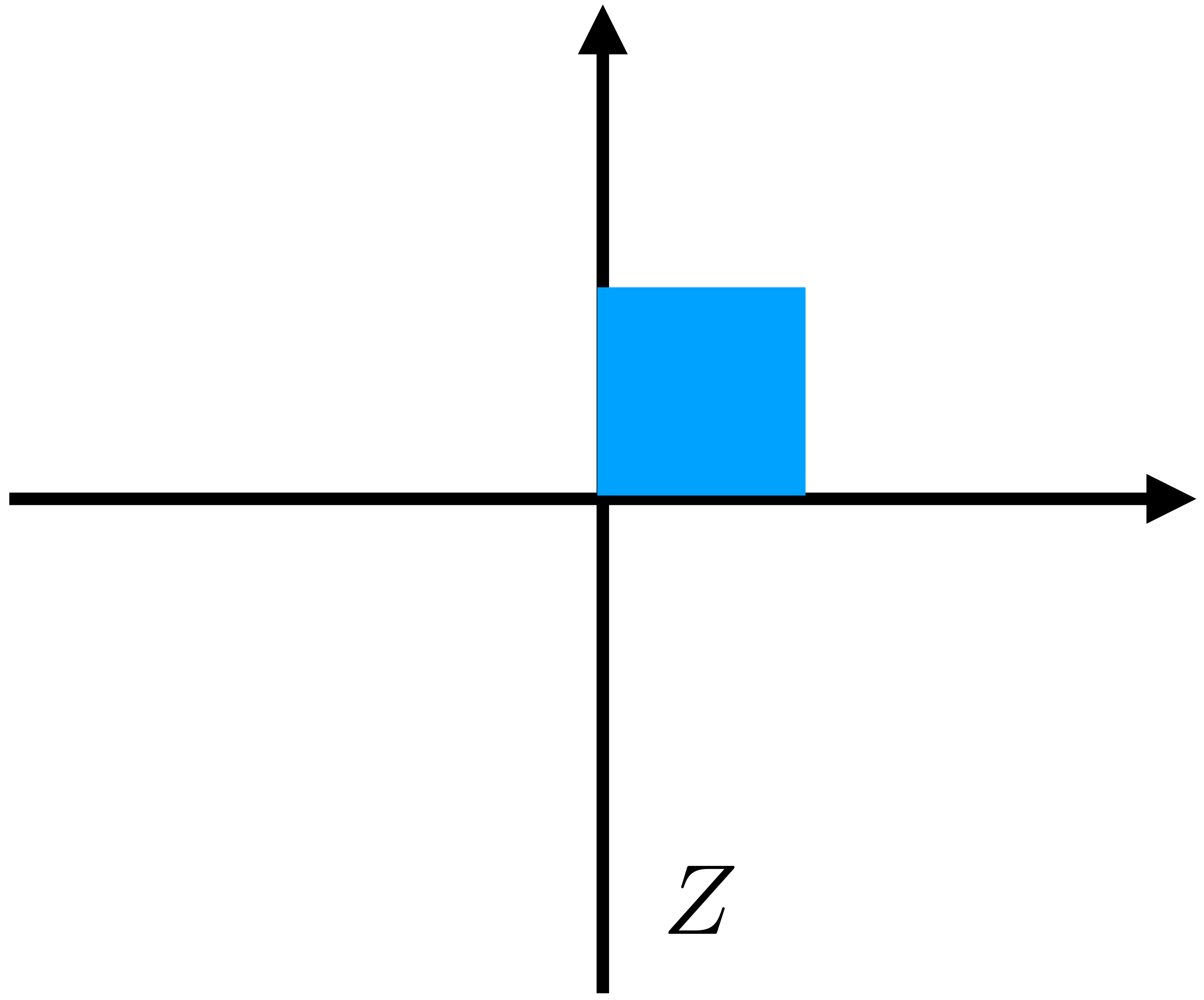
$p(z)$ defined over $z \in Z$

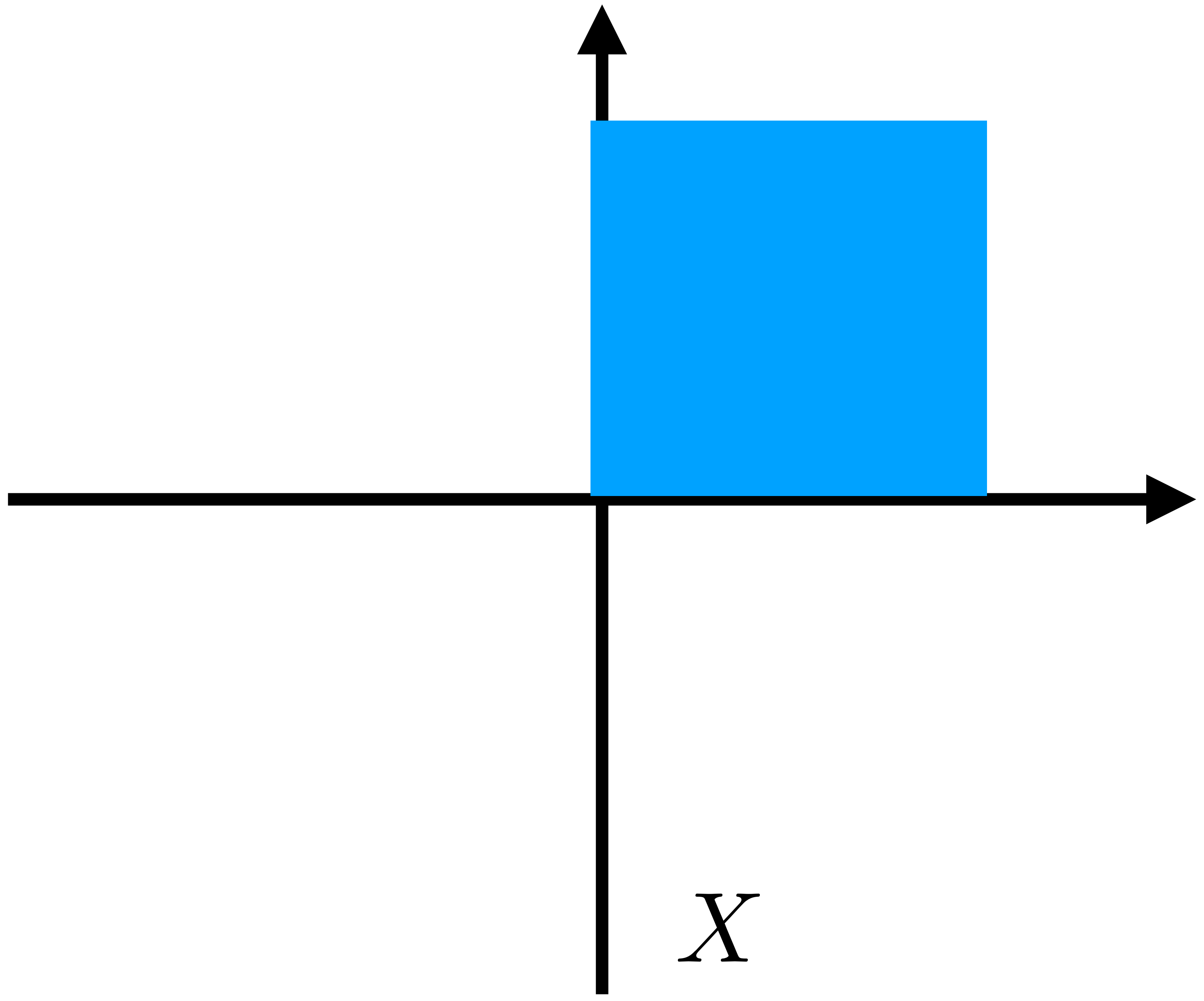
$$p(x) = p(f^{-1}(x)) \left| det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right|$$

$$p(x) = p(z) \left| det \left(\frac{\partial z}{\partial x} \right) \right|$$

$f : Z \to X, f$ is invertible

$p(z)$ defined over $z \in Z$

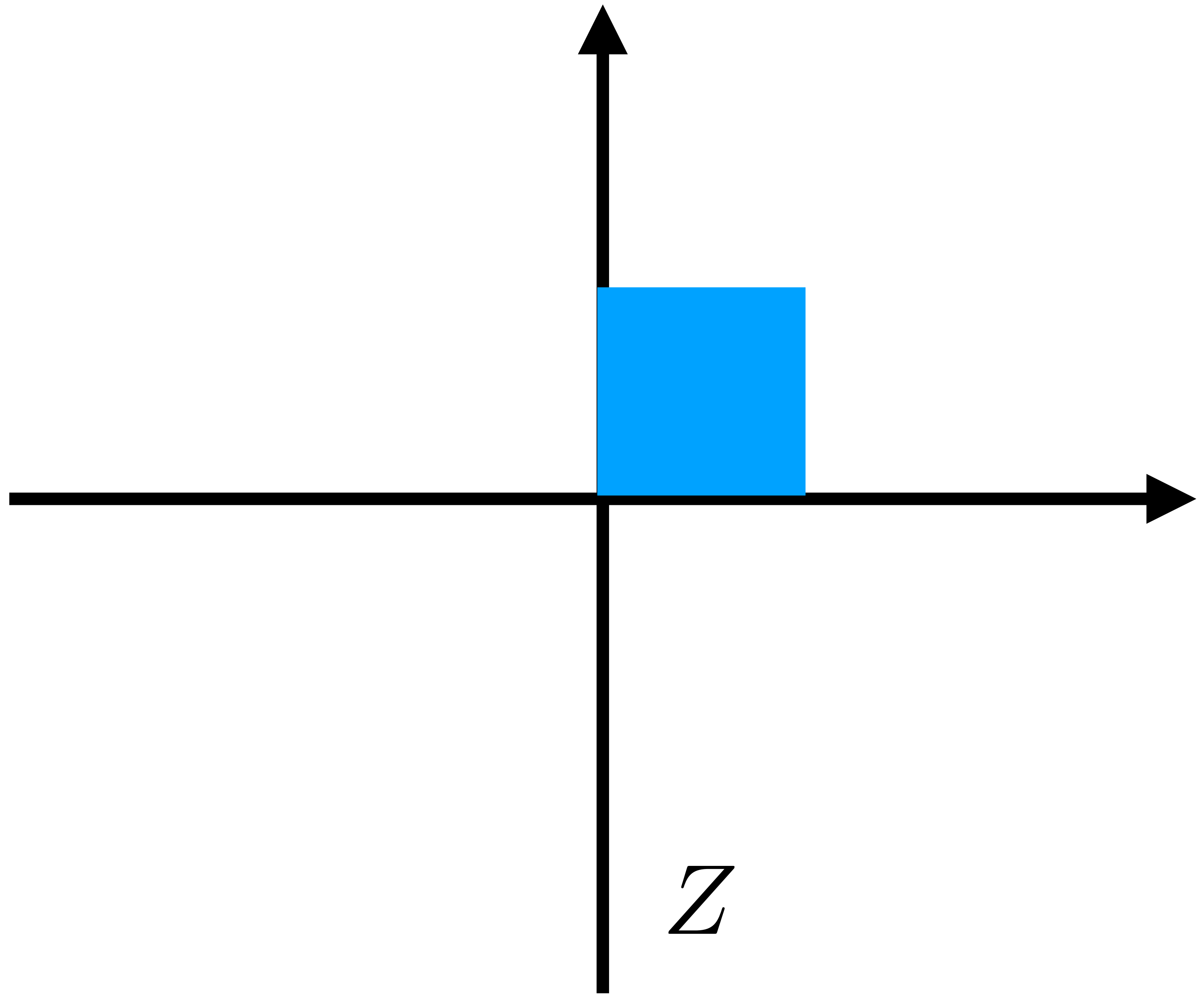


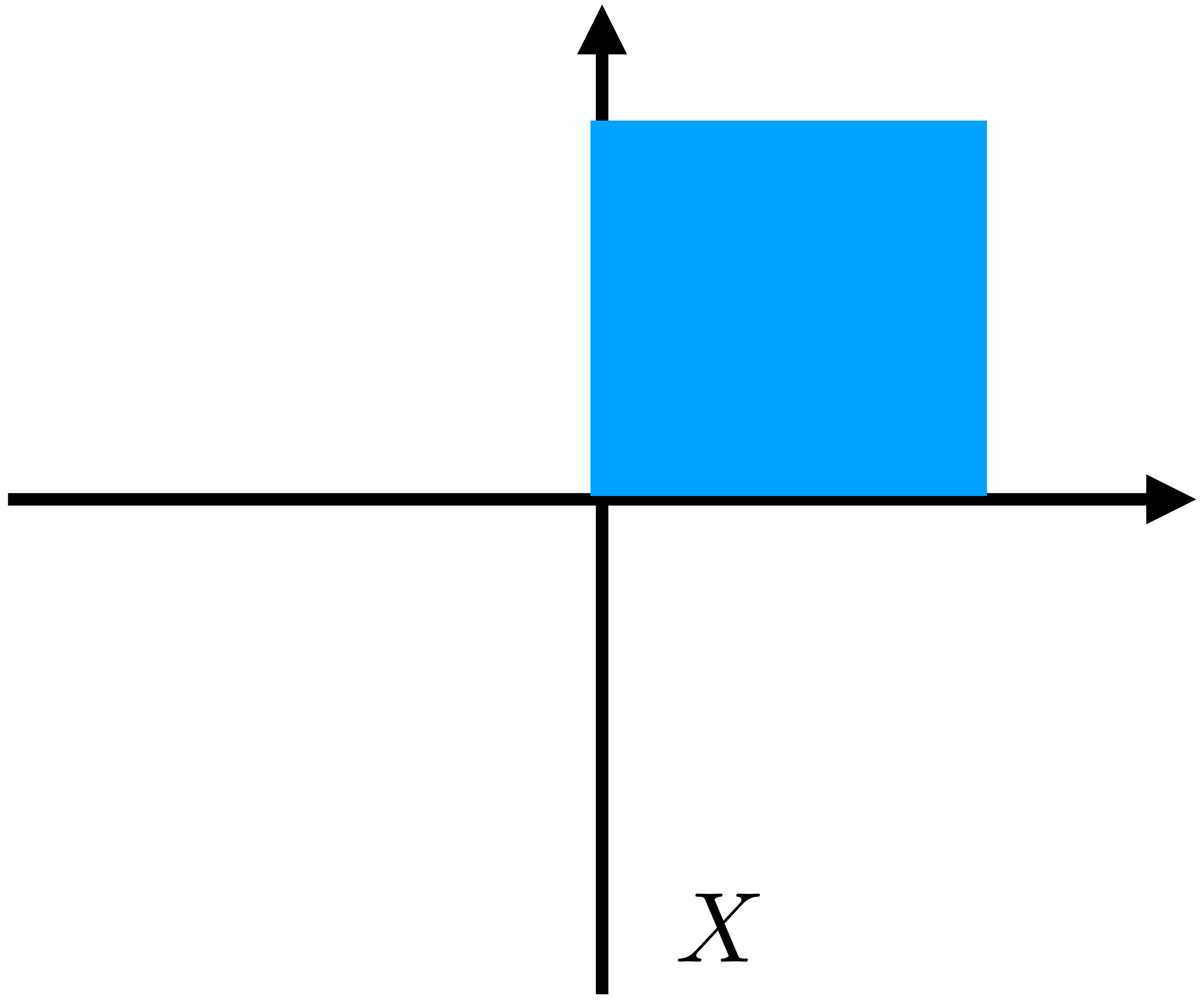


$f : Z \to X, f$ is invertible

$p(z)$ defined over $z \in Z$

$$det\left(\frac{\partial x}{\partial z}\right) = 4$$

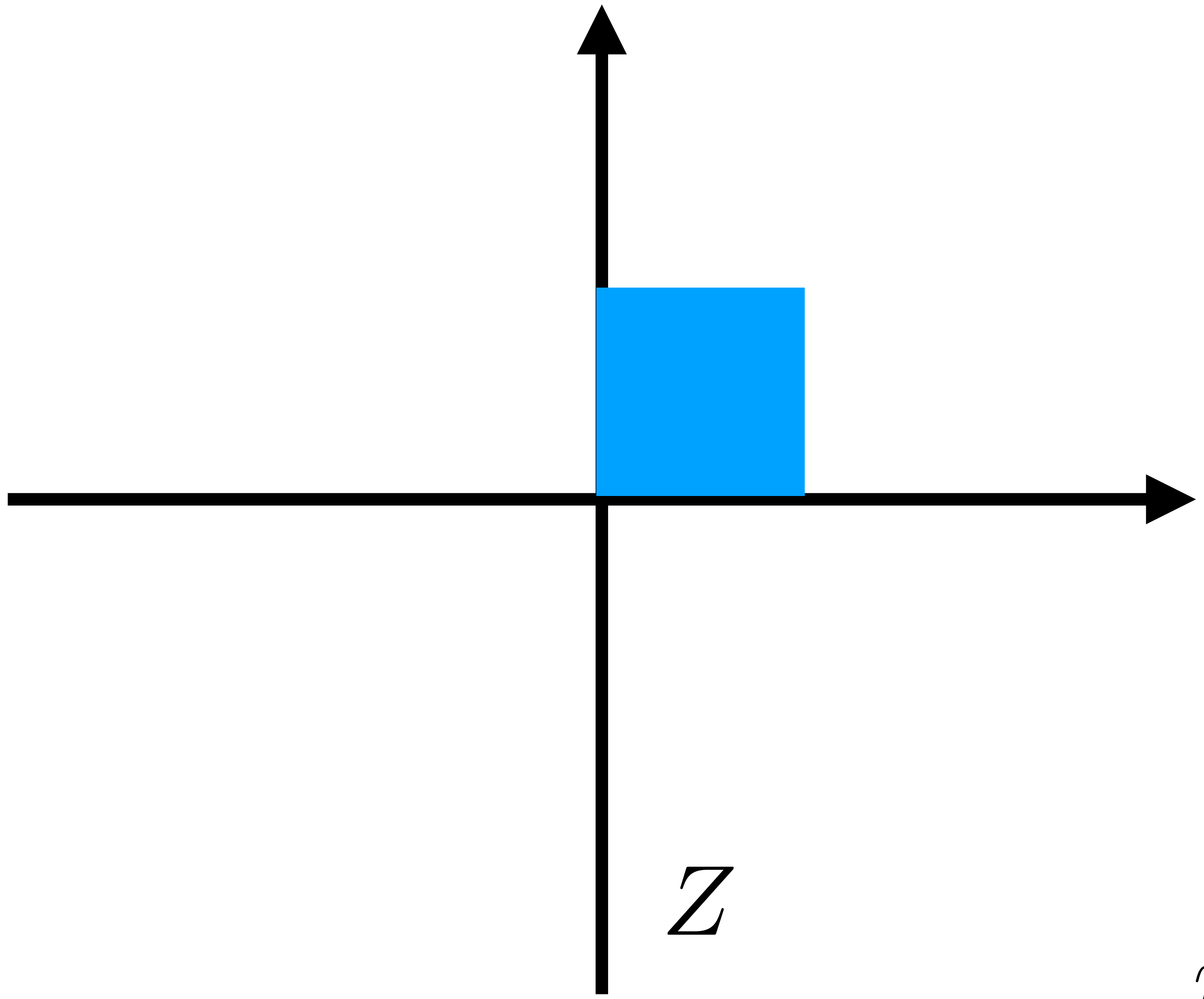


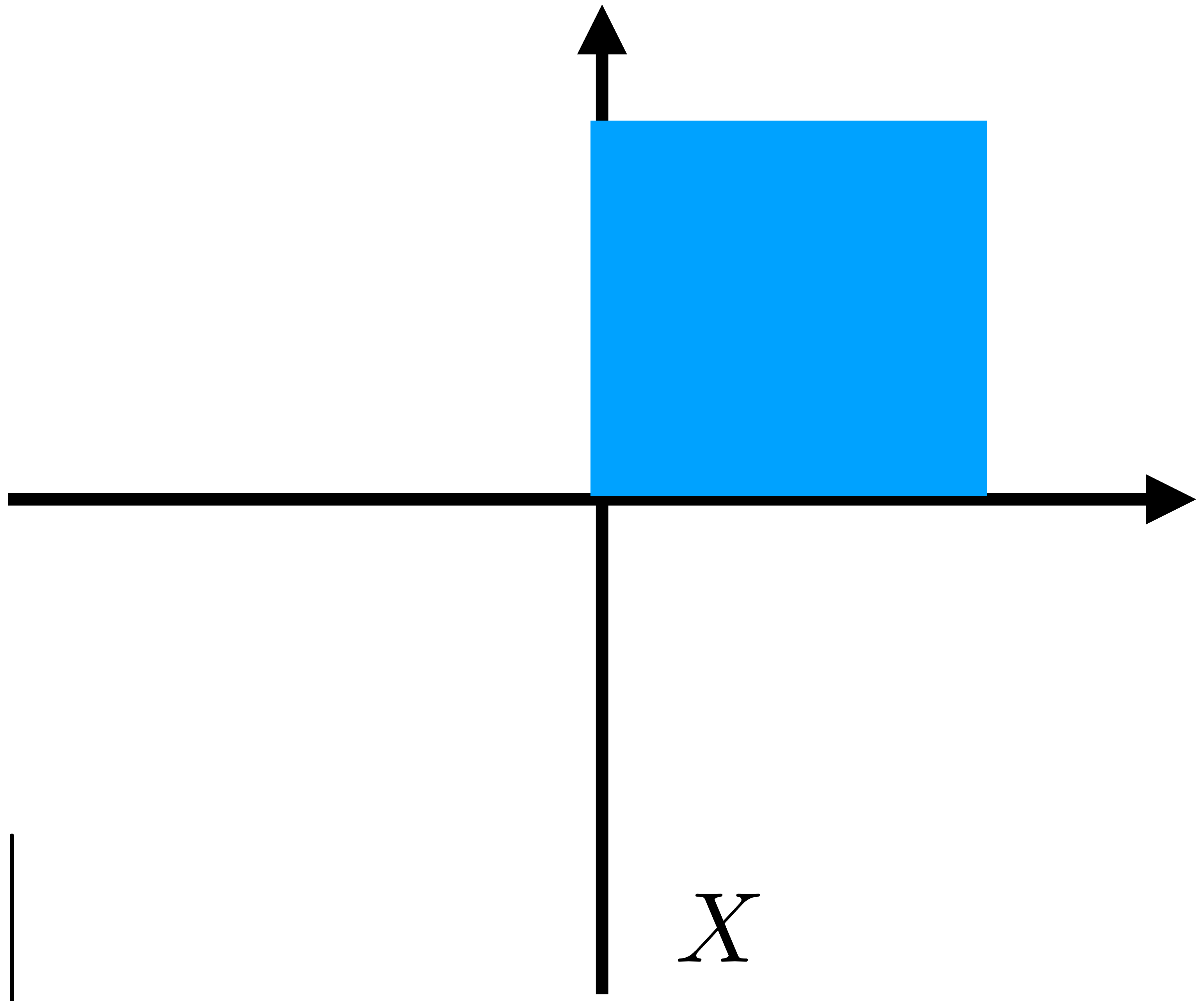


$f : Z \to X, f$ is invertible

$p(z)$ defined over $z \in Z$

$$det\left(\frac{\partial x}{\partial z}\right) = 4$$





$$p(x) = p(z) = \left|\frac{1}{det\left(\frac{\partial x}{\partial z}\right)}\right|$$

Maximize Log-likelihood

$f : Z \rightarrow X, f$ is invertible

$p(z)$ defined over $z \in Z$

$$\log p(x) = \log p(z) + \log \left| det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right|$$

Maximize Log-likelihood

$f : Z \to X, f$ is invertible

$p(z)$ defined over $z \in Z$

$$\log p(x) = \log p(z) + \log \left| det\left(\frac{\partial f^{-1}(x)}{\partial x}\right)\right|$$

$$\log p(x) = \log p(z) + \sum_{i=1}^{K} \log \left| det\left(\frac{\partial f^{-1}(x)}{\partial x}\right)\right|$$

Jacobian: Lower Triangular Matrix

$f : Z \to X, f$ is invertible
$p(z)$ defined over $z \in Z$

$$\mathbf{J} = \begin{bmatrix} \ell_{1,1} & & & & 0 \\ \ell_{2,1} & \ell_{2,2} & & & \\ \ell_{3,1} & \ell_{3,2} & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ \ell_{n,1} & \ell_{n,2} & \dots & \ell_{n,n-1} & \ell_{n,n} \end{bmatrix}$$

Jacobian: Lower Triangular Matrix

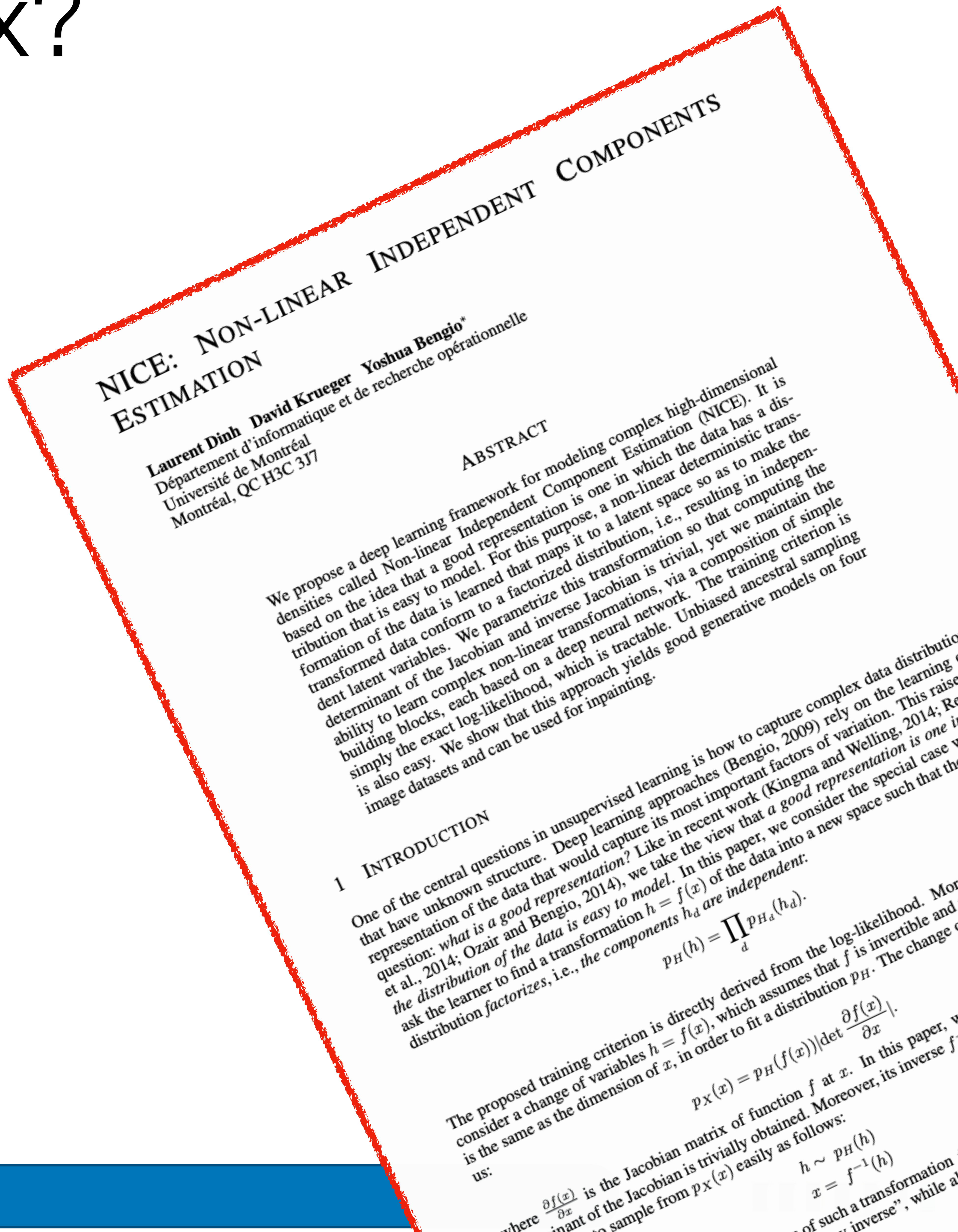
$f : Z \to X, f$ is invertible

$p(z)$ defined over $z \in Z$

$$\mathbf{J} = \begin{bmatrix} \ell_{1,1} & & & & 0 \\ \ell_{2,1} & \ell_{2,2} & & & \\ \ell_{3,1} & \ell_{3,2} & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ \ell_{n,1} & \ell_{n,2} & \cdots & \ell_{n,n-1} & \ell_{n,n} \end{bmatrix}$$

How to ensure lower-triangular Jacobian matrix?

$$z \in \mathbb{R}^D$$


NICE: Non-linear Independent Components Estimation

Laurent Dinh David Krueger Yoshua Bengio*
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

We propose a deep learning framework for modeling complex high-dimensional densities called Non-linear Independent Component Estimation (NICE). It is based on the idea that a good representation is one in which the data has a distribution that is easy to model. For this purpose, a non-linear deterministic transformation of the data is learned that maps it to a latent space so as to make the transformed data conform to a factorized distribution, i.e., resulting in independent latent variables. We parametrize this transformation so that computing the determinant of the Jacobian and inverse Jacobian is trivial, yet we maintain the ability to learn complex non-linear transformations, via a composition of simple building blocks, each based on a deep neural network. The training criterion is simply the exact log-likelihood, which is tractable. Unbiased ancestral sampling is also easy. We show that this approach yields good generative models on four image datasets and can be used for inpainting.

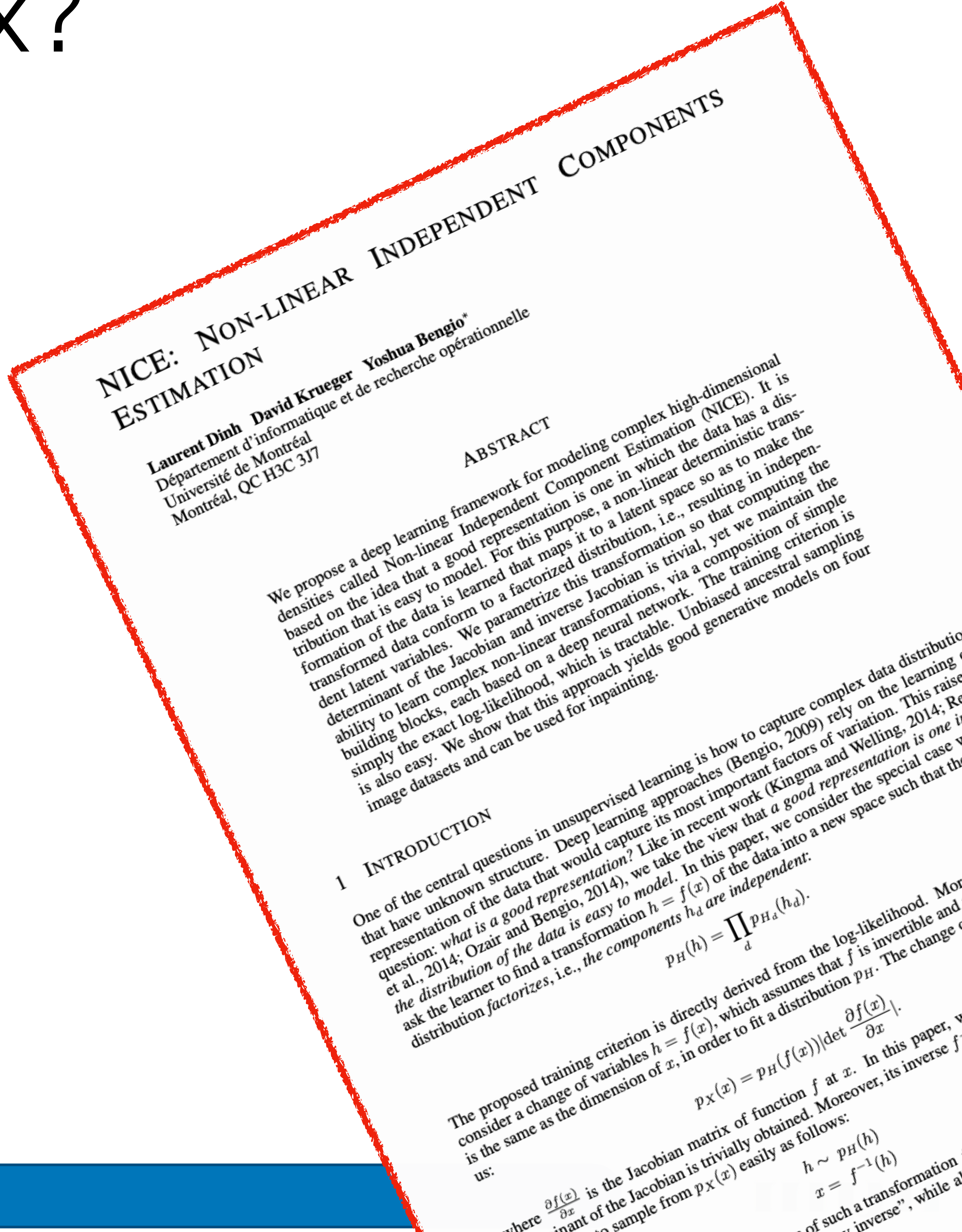
1 Introduction

How to ensure lower-triangular Jacobian matrix?

$$z \in \mathbb{R}^D$$

$$z_{1:d}$$

$$z_{d+1:D}$$



NICE: NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION

Laurent Dinh David Krueger Yoshua Bengio*
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

ABSTRACT

We propose a deep learning framework for modeling complex high-dimensional densities called Non-linear Independent Component Estimation (NICE). It is based on the idea that a good representation is one in which the data has a distribution that is easy to model. For this purpose, a non-linear deterministic transformation of the data is learned that maps it to a latent space so as to make the transformed data conform to a factorized distribution, i.e., resulting in independent latent variables. We parametrize this transformation so that computing the determinant of the Jacobian and inverse Jacobian is trivial, yet we maintain the ability to learn complex non-linear transformations, via a composition of simple building blocks, each based on a deep neural network. The training criterion is simply the exact log-likelihood, which is tractable. Unbiased ancestral sampling is also easy. We show that this approach yields good generative models on four image datasets and can be used for inpainting.

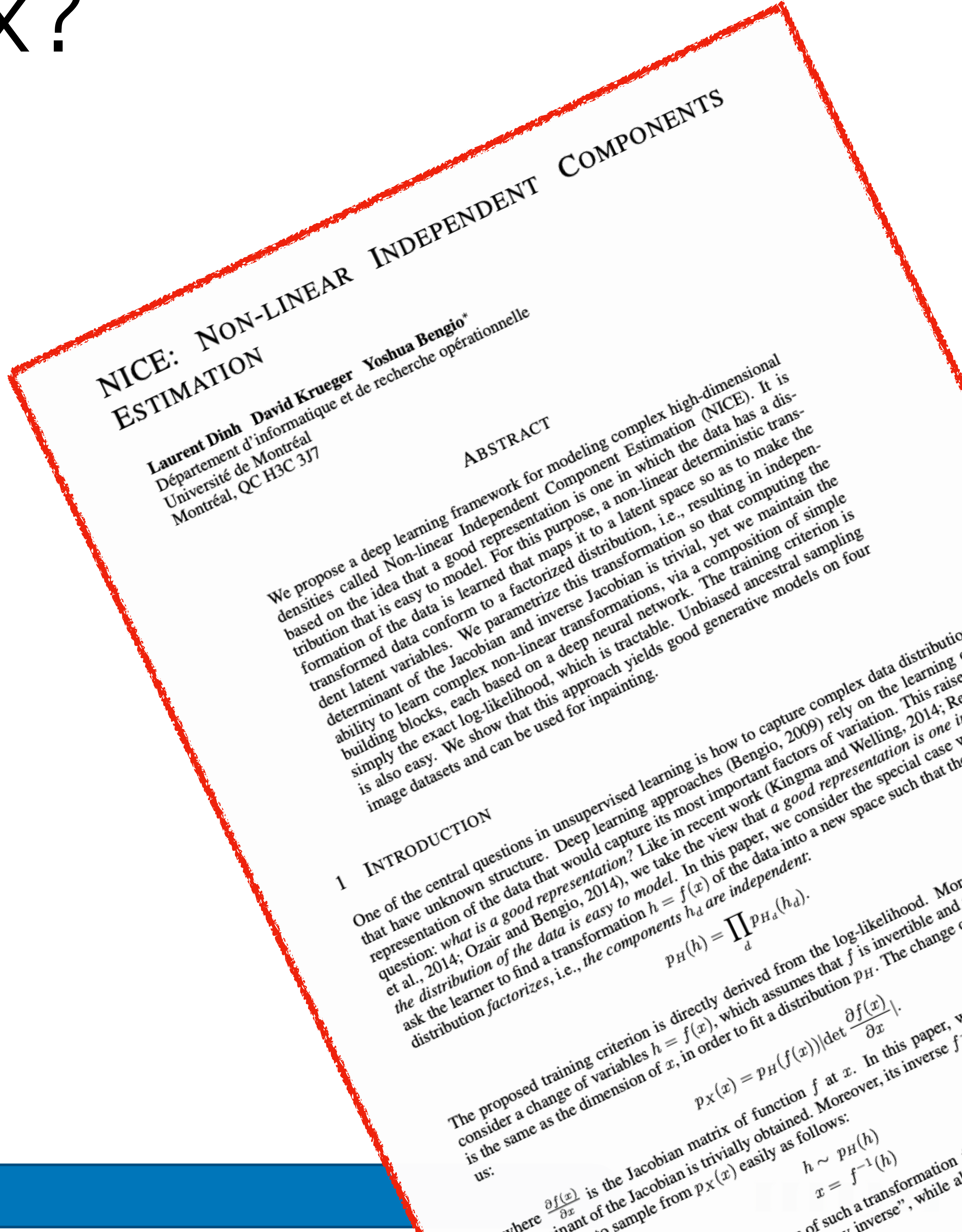
1 INTRODUCTION

How to ensure lower-triangular Jacobian matrix?

Coupling layer

$$z \in \mathbb{R}^D$$

$z_{1:d}$ $z_{d+1:D}$


NICE: NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION

Laurent Dinh David Krueger Yoshua Bengio*
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

ABSTRACT

We propose a deep learning framework for modeling complex high-dimensional densities called Non-linear Independent Component Estimation (NICE). It is based on the idea that a good representation is one in which the data has a distribution that is easy to model. For this purpose, a non-linear deterministic transformation of the data is learned that maps it to a latent space so as to make the transformed data conform to a factorized distribution, i.e., resulting in independent latent variables. We parametrize this transformation so that computing the determinant of the Jacobian and inverse Jacobian is trivial, yet we maintain the ability to learn complex non-linear transformations, via a composition of simple building blocks, each based on a deep neural network. The training criterion is simply the exact log-likelihood, which is tractable. Unbiased ancestral sampling is also easy. We show that this approach yields good generative models on four image datasets and can be used for inpainting.

1 INTRODUCTION

One of the central questions in unsupervised learning is how to capture complex data distributio
that have unknown structure. Deep learning approaches (Bengio, 2009) rely on the learning o
representation of the data that would capture its most important factors of variation. This raise
question: *what is a good representation?* Like in recent work (Kingma and Welling, 2014; Re
et al., 2014; Ozair and Bengio, 2014), we take the view that *a good representation is one i*
the distribution of the data is easy to model. In this paper, we consider the special case w
ask the learner to find a transformation $h = f(x)$ of the data into a new space such that th
distribution factorizes, i.e., *the components h_d are independent*:

$$p_H(h) = \prod_d p_{H_d}(h_d).$$

The proposed training criterion is directly derived from the log-likelihood. Mo
consider a change of variables $h = f(x)$, which assumes that f is invertible and
is the same as the dimension of x, in order to fit a distribution p_H. The change
us:

$$p_X(x) = p_H(f(x))|\det \frac{\partial f(x)}{\partial x}|.$$

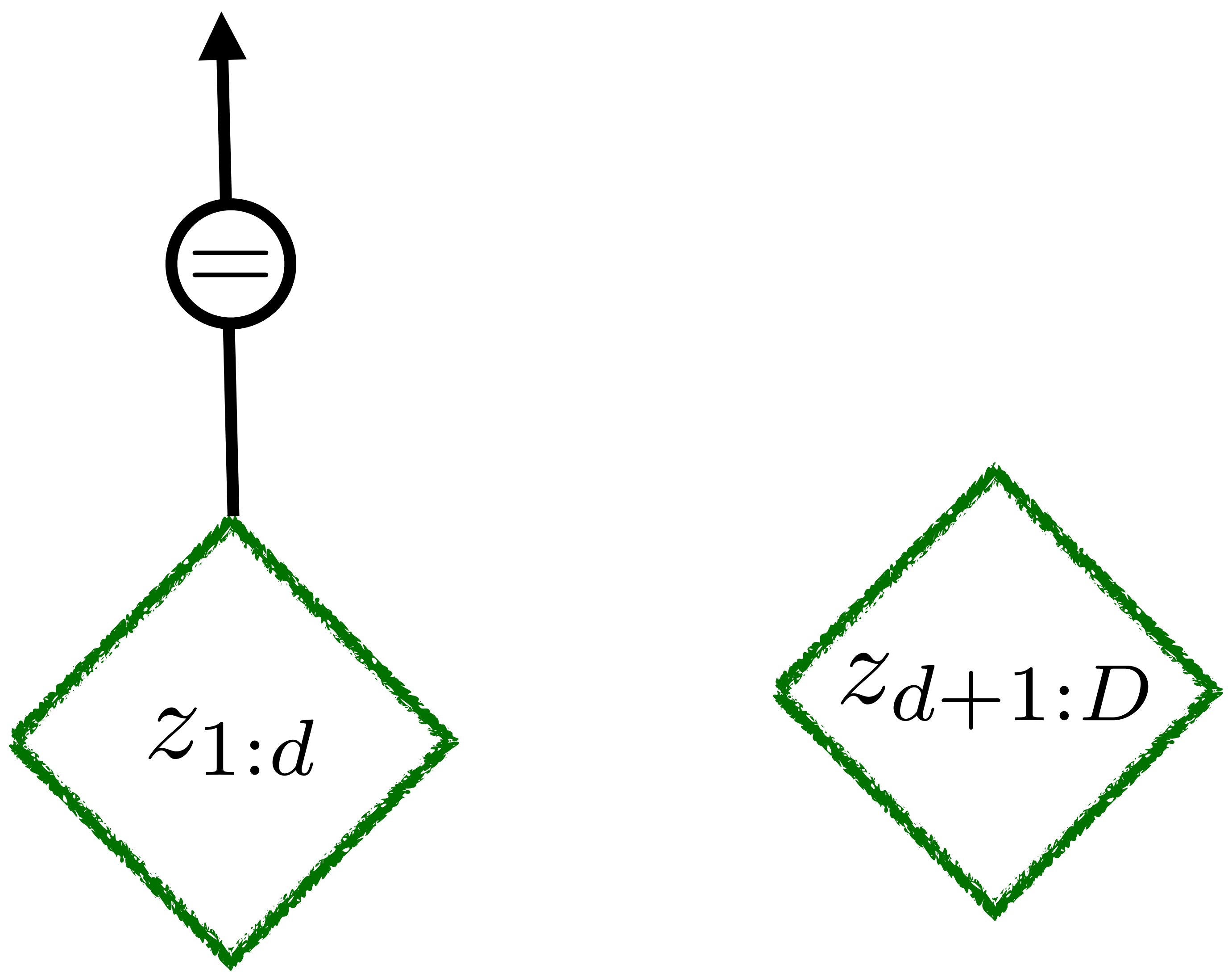
where $\frac{\partial f(x)}{\partial x}$ is the Jacobian matrix of function f at x. In this paper, w
...inant of the Jacobian is trivially obtained. Moreover, its inverse f
...to sample from $p_X(x)$ easily as follows:

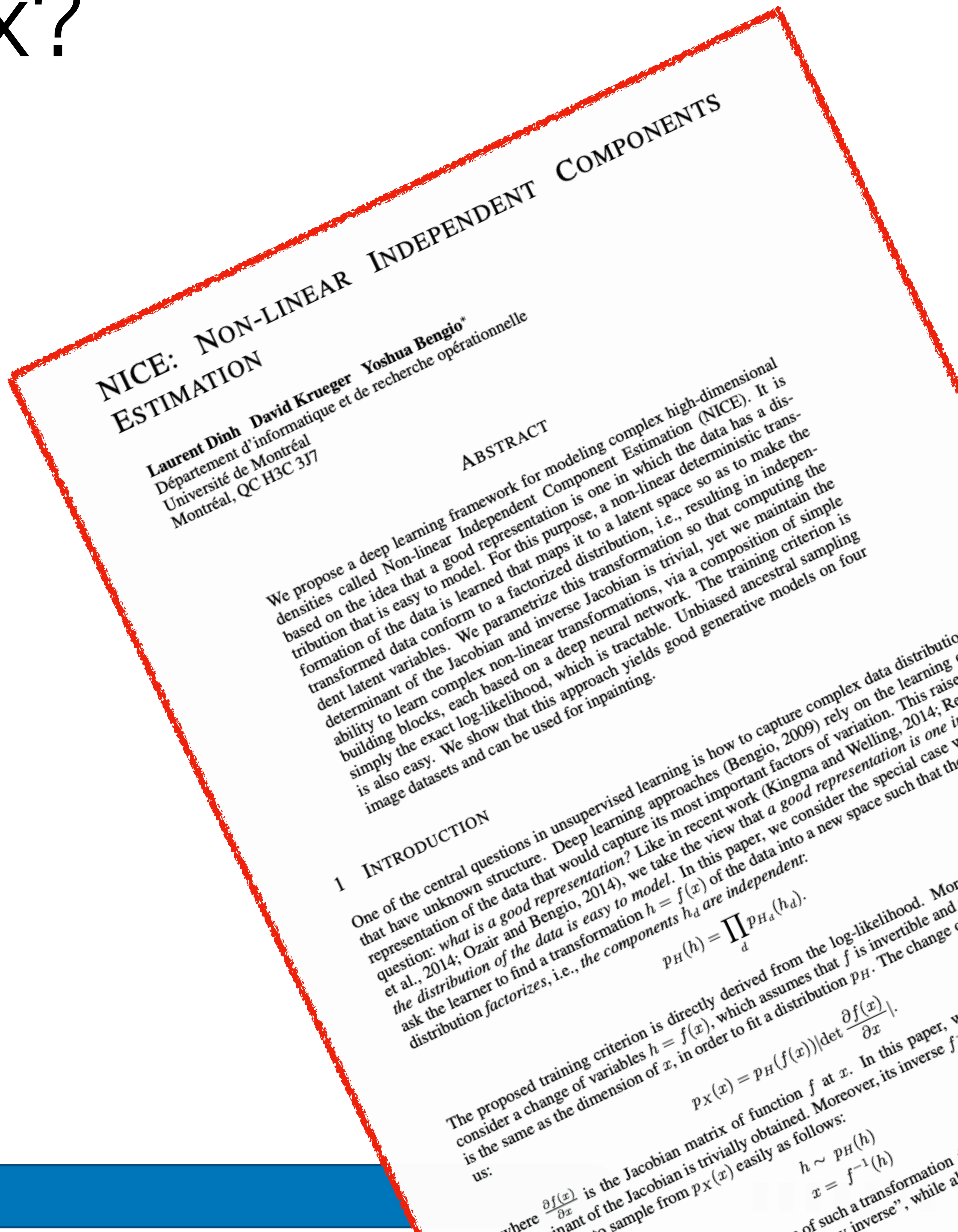
$$h \sim p_H(h)$$
$$x = f^{-1}(h)$$

How to ensure lower-triangular Jacobian matrix?

Coupling layer

$$z \in \mathbb{R}^D$$





NICE: NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION

Laurent Dinh David Krueger Yoshua Bengio*
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

ABSTRACT

We propose a deep learning framework for modeling complex high-dimensional densities called Non-linear Independent Component Estimation (NICE). It is based on the idea that a good representation is one in which the data has a distribution that is easy to model. For this purpose, a non-linear deterministic transformation of the data is learned that maps it to a latent space so as to make the transformed data conform to a factorized distribution, i.e., resulting in independent latent variables. We parametrize this transformation so that computing the determinant of the Jacobian and inverse Jacobian is trivial, yet we maintain the ability to learn complex non-linear transformations, via a composition of simple building blocks, each based on a deep neural network. The training criterion is simply the exact log-likelihood, which is tractable. Unbiased ancestral sampling is also easy. We show that this approach yields good generative models on four image datasets and can be used for inpainting.

1 INTRODUCTION

One of the central questions in unsupervised learning is how to capture complex data distributio
that have unknown structure. Deep learning approaches (Bengio, 2009) rely on the learning
representation of the data that would capture its most important factors of variation. This raise
question: *what is a good representation?* Like in recent work (Kingma and Welling, 2014; Re
et al., 2014; Ozair and Bengio, 2014), we take the view that *a good representation is one i*
the distribution of the data is easy to model. In this paper, we consider the special case w
ask the learner to find a transformation $h = f(x)$ of the data into a new space such that th
distribution *factorizes*, i.e., *the components h_d are independent*:

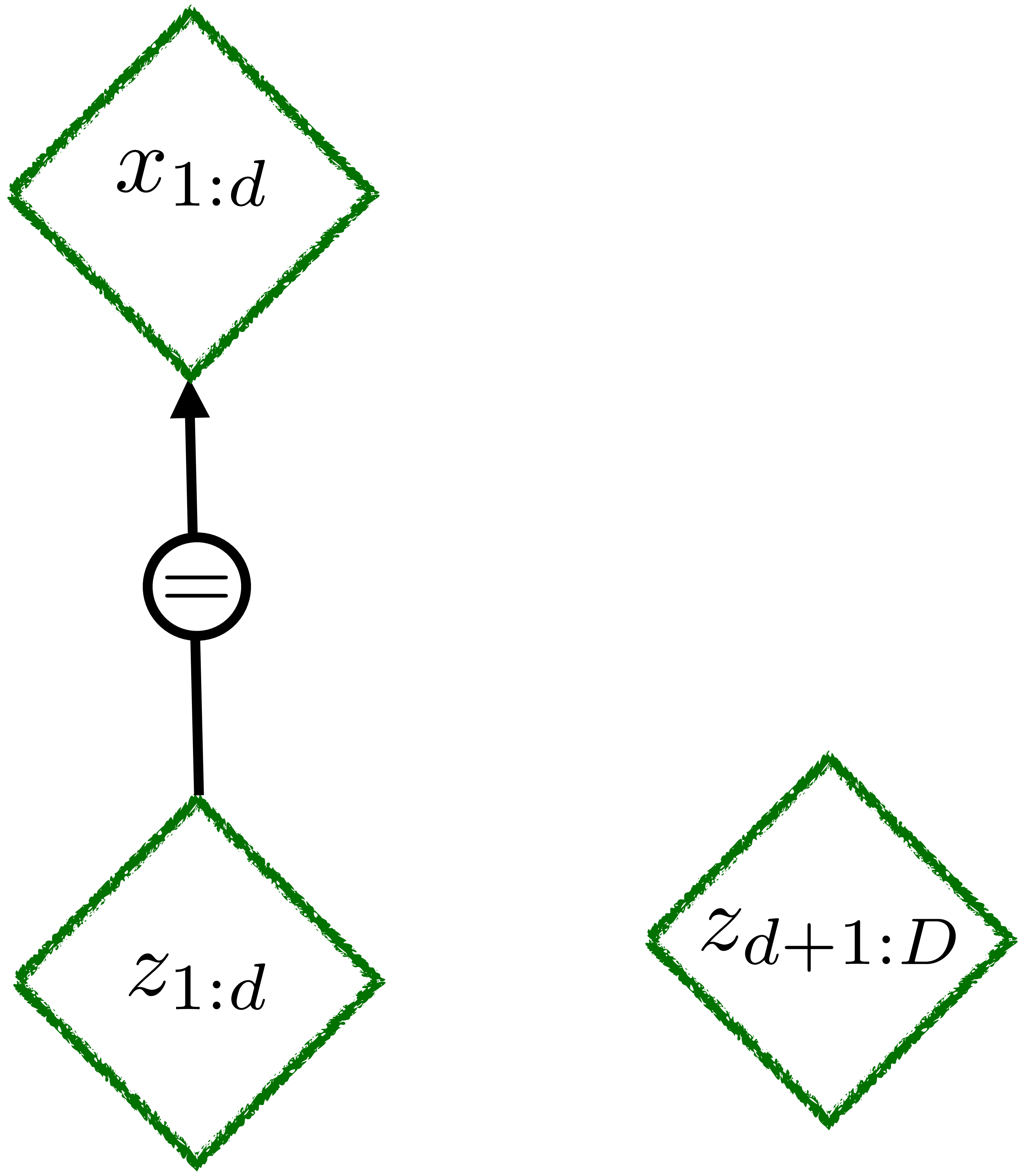
$$p_H(h) = \prod_d p_{H_d}(h_d).$$

The proposed training criterion is directly derived from the log-likelihood. Mor
consider a change of variables $h = f(x)$, which assumes that f is invertible and
is the same as the dimension of x, in order to fit a distribution p_H. The change

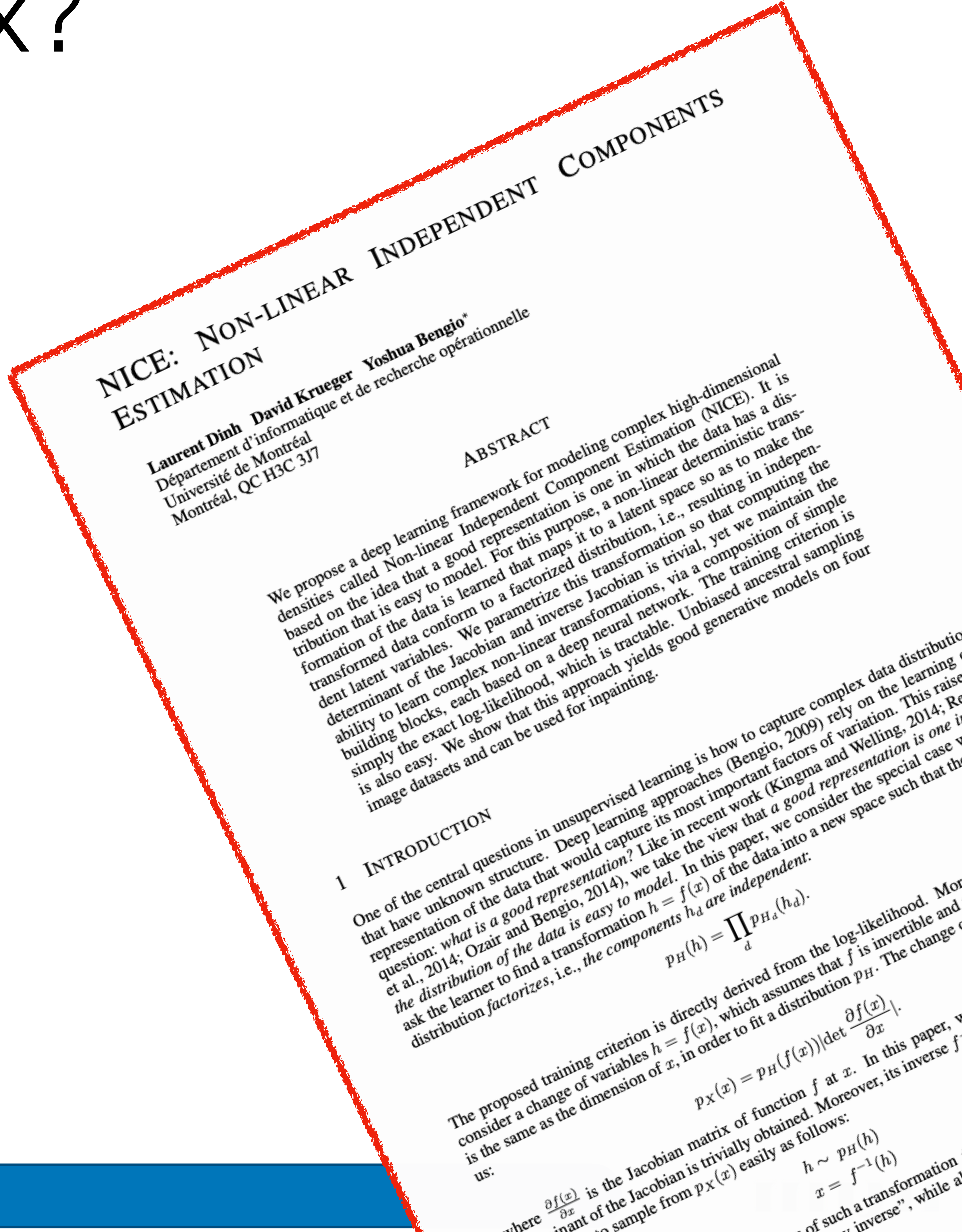
$$p_X(x) = p_H(f(x))|\det \frac{\partial f(x)}{\partial x}|.$$

How to ensure lower-triangular Jacobian matrix?

Coupling layer



$$z \in \mathbb{R}^D$$


NICE: NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION

Laurent Dinh David Krueger Yoshua Bengio*
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

ABSTRACT

We propose a deep learning framework for modeling complex high-dimensional densities called Non-linear Independent Component Estimation (NICE). It is based on the idea that a good representation is one in which the data has a distribution that is easy to model. For this purpose, a non-linear deterministic transformation of the data is learned that maps it to a latent space so as to make the transformed data conform to a factorized distribution, i.e., resulting in independent latent variables. We parametrize this transformation so that computing the determinant of the Jacobian and inverse Jacobian is trivial, yet we maintain the ability to learn complex non-linear transformations, via a composition of simple building blocks, each based on a deep neural network. The training criterion is simply the exact log-likelihood, which is tractable. Unbiased ancestral sampling is also easy. We show that this approach yields good generative models on four image datasets and can be used for inpainting.

1 INTRODUCTION

One of the central questions in unsupervised learning is how to capture complex data distributio… that have unknown structure. Deep learning approaches (Bengio, 2009) rely on the learning o… representation of the data that would capture its most important factors of variation. This raise… question: *what is a good representation?* Like in recent work (Kingma and Welling, 2014; Re… et al., 2014; Ozair and Bengio, 2014), we take the view that *a good representation is one i… the distribution of the data is easy to model*. In this paper, we consider the special case w… ask the learner to find a transformation $h = f(x)$ of the data into a new space such that th… distribution factorizes, i.e., *the components h_d are independent*:

$$p_H(h) = \prod_d p_{H_d}(h_d).$$

The proposed training criterion is directly derived from the log-likelihood. Mo… consider a change of variables $h = f(x)$, which assumes that f is invertible and… is the same as the dimension of x, in order to fit a distribution p_H. The change o… us:

$$p_X(x) = p_H(f(x))|\det \frac{\partial f(x)}{\partial x}|.$$

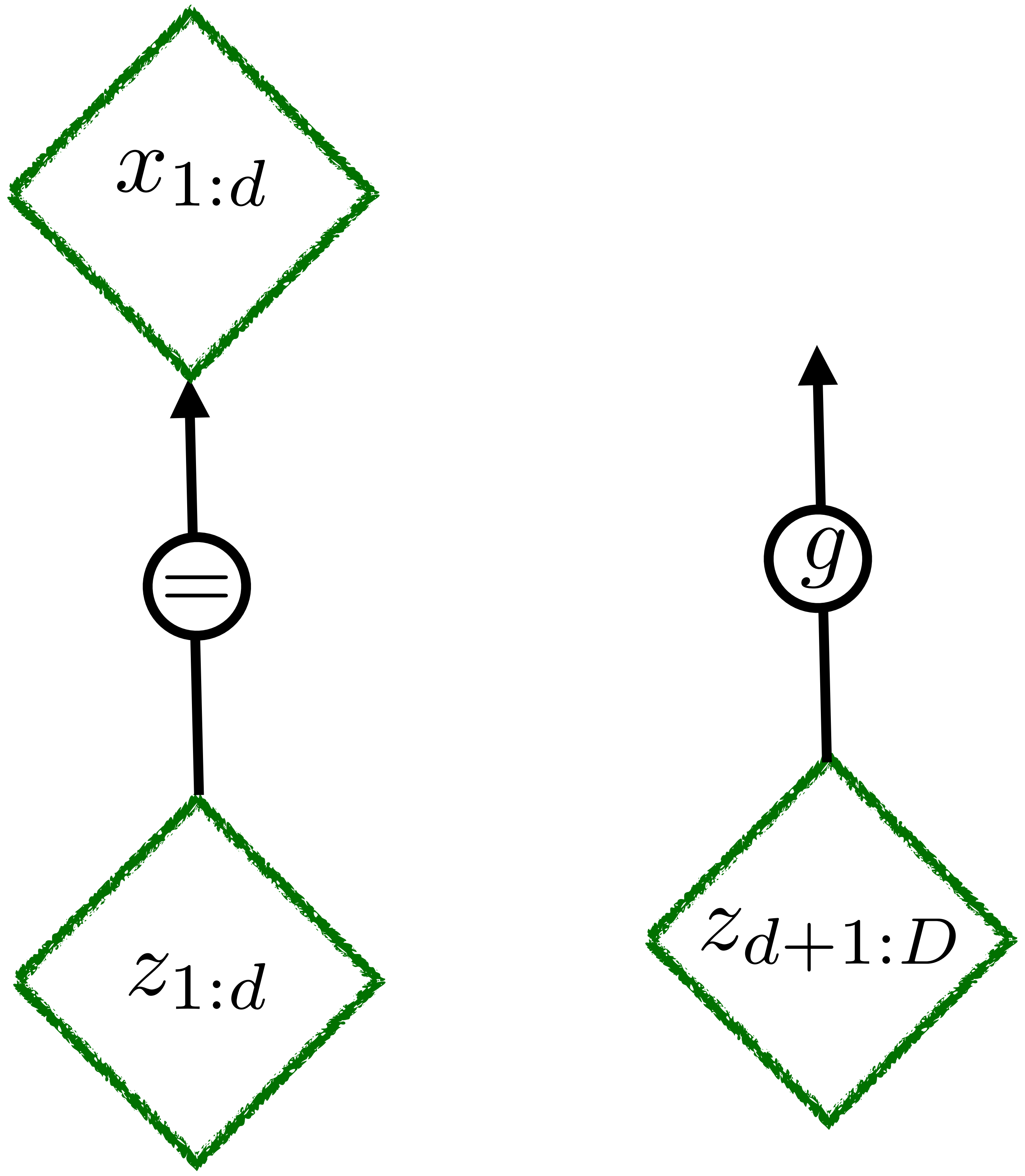
where $\frac{\partial f(x)}{\partial x}$ is the Jacobian matrix of function f at x. In this paper, w… …inant of the Jacobian is trivially obtained. Moreover, its inverse f… …to sample from $p_X(x)$ easily as follows:

$$h \sim p_H(h)$$
$$x = f^{-1}(h)$$

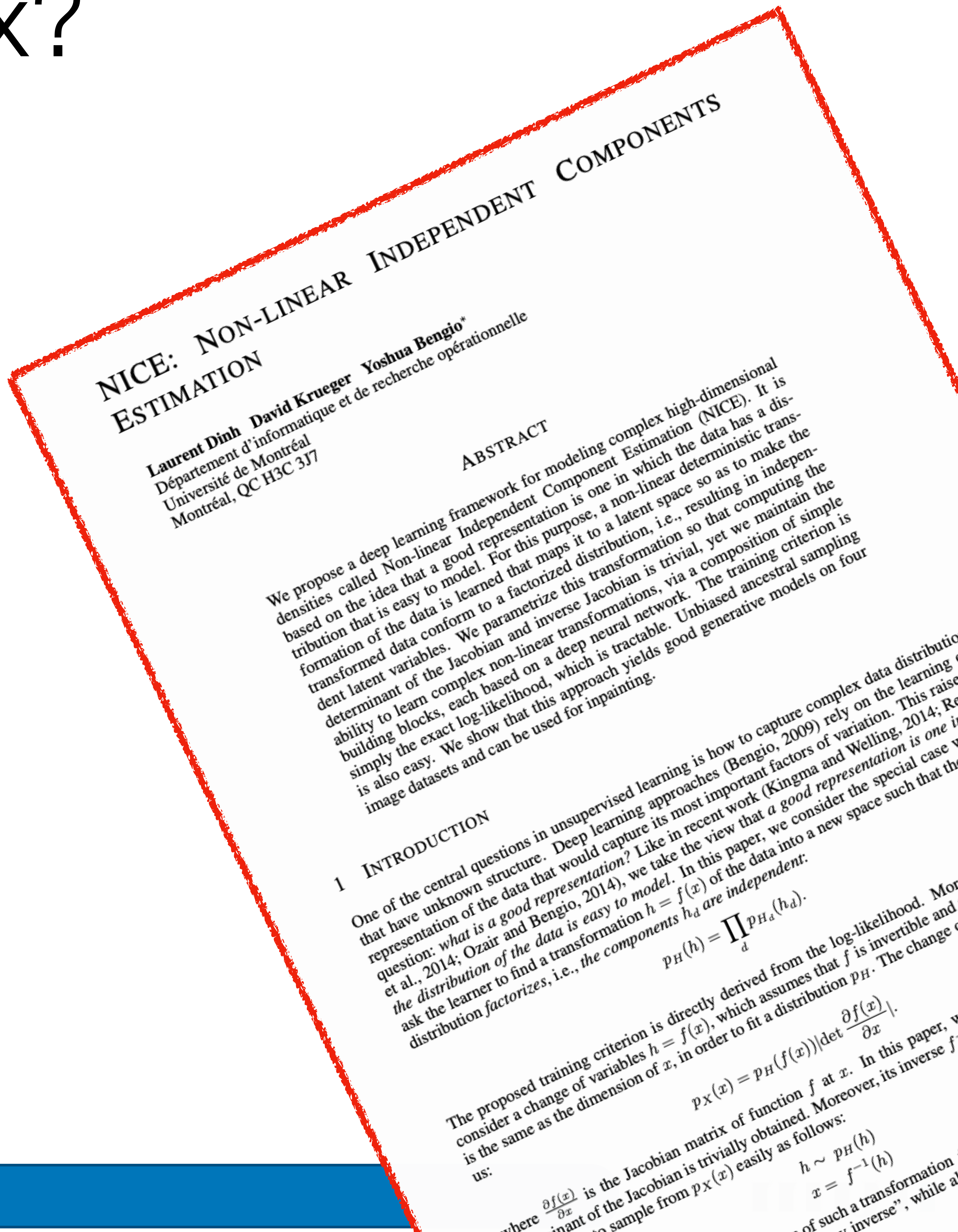
…of such a transformation… …inverse", while al…

How to ensure lower-triangular Jacobian matrix?

Coupling layer



$$z \in \mathbb{R}^D$$



NICE: NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION

Laurent Dinh David Krueger Yoshua Bengio*
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

ABSTRACT

We propose a deep learning framework for modeling complex high-dimensional densities called Non-linear Independent Component Estimation (NICE). It is based on the idea that a good representation is one in which the data has a distribution that is easy to model. For this purpose, a non-linear deterministic transformation of the data is learned that maps it to a latent space so as to make the transformed data conform to a factorized distribution, i.e., resulting in independent latent variables. We parametrize this transformation so that computing the determinant of the Jacobian and inverse Jacobian is trivial, yet we maintain the ability to learn complex non-linear transformations, via a composition of simple building blocks, each based on a deep neural network. The training criterion is simply the exact log-likelihood, which is tractable. Unbiased ancestral sampling is also easy. We show that this approach yields good generative models on four image datasets and can be used for inpainting.

1 INTRODUCTION

One of the central questions in unsupervised learning is how to capture complex data distributio that have unknown structure. Deep learning approaches (Bengio, 2009) rely on the learning o representation of the data that would capture its most important factors of variation. This raise question: *what is a good representation?* Like in recent work (Kingma and Welling, 2014; Re et al., 2014; Ozair and Bengio, 2014), we take the view that *a good representation is one i the distribution of the data is easy to model*. In this paper, we consider the special case w ask the learner to find a transformation $h = f(x)$ of the data into a new space such that th *distribution factorizes*, i.e., *the components h_d are independent*:

$$p_H(h) = \prod_d p_{H_d}(h_d).$$

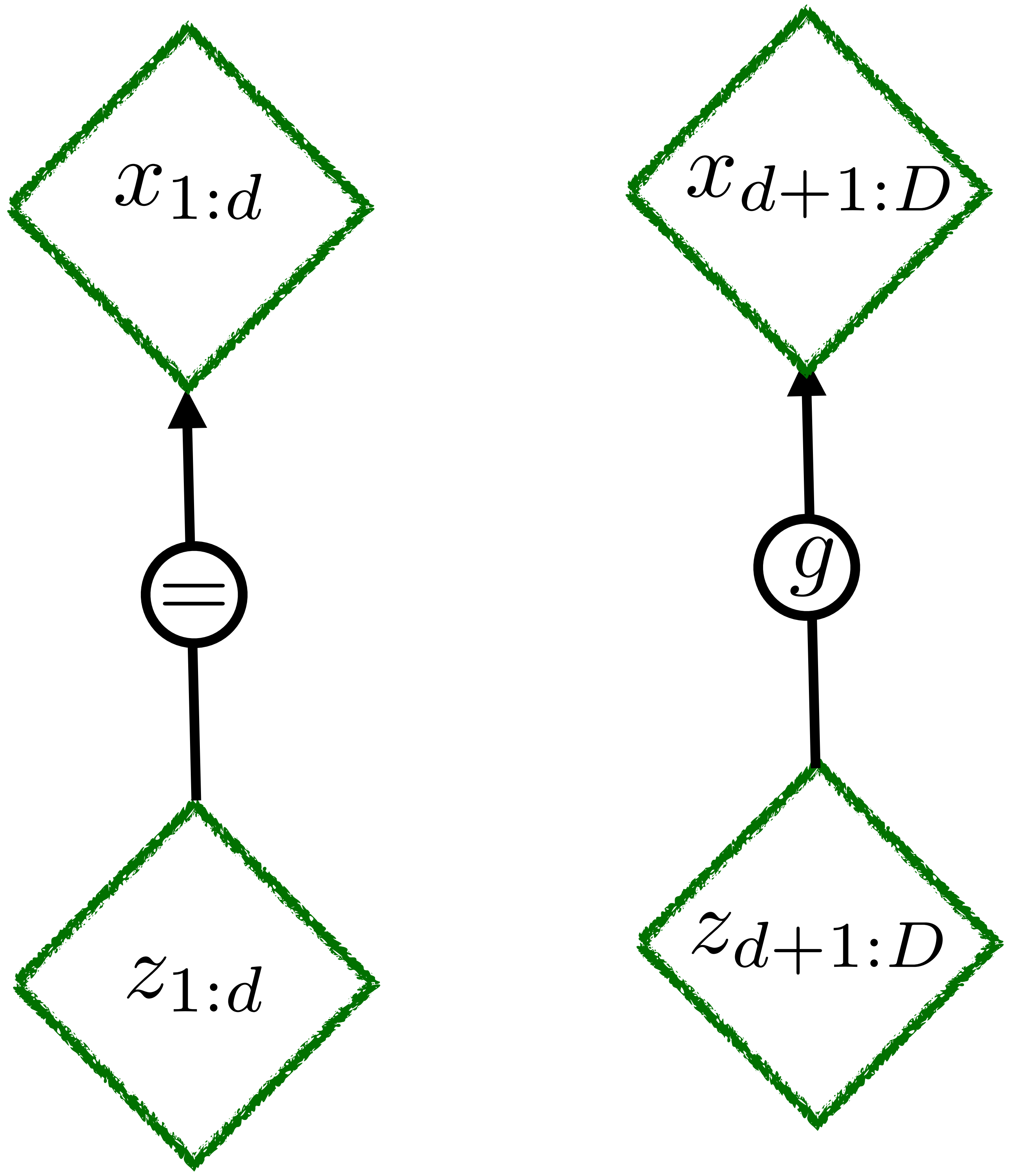
The proposed training criterion is directly derived from the log-likelihood. Mo consider a change of variables $h = f(x)$, which assumes that f is invertible and is the same as the dimension of x, in order to fit a distribution p_H. The change o us:

$$p_X(x) = p_H(f(x))|\det \frac{\partial f(x)}{\partial x}|.$$

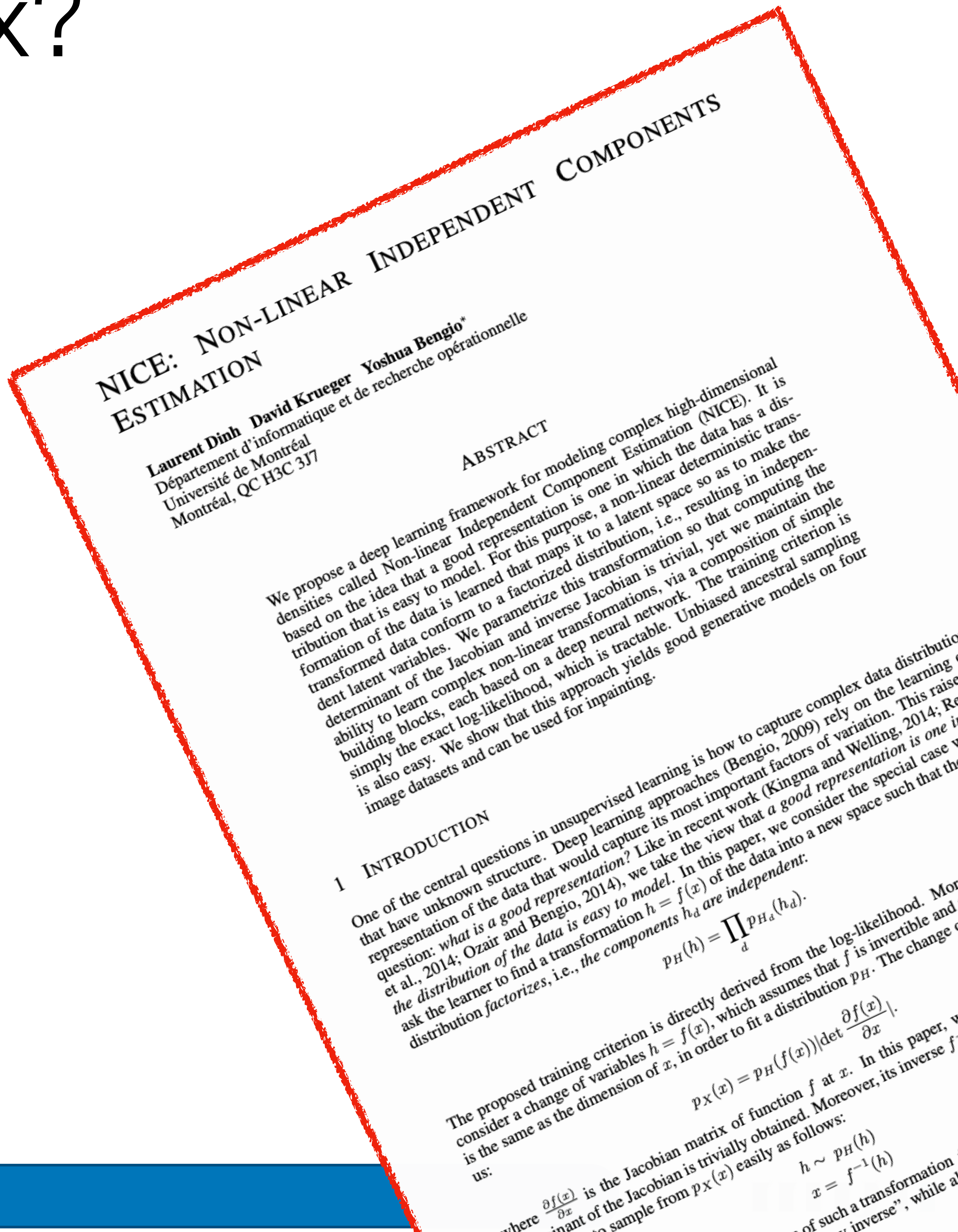
where $\frac{\partial f(x)}{\partial x}$ is the Jacobian matrix of function f at x. In this paper, w

How to ensure lower-triangular Jacobian matrix?

Coupling layer



$z \in \mathbb{R}^D$



NICE: NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION

Laurent Dinh David Krueger Yoshua Bengio*
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

ABSTRACT

We propose a deep learning framework for modeling complex high-dimensional densities called Non-linear Independent Component Estimation (NICE). It is based on the idea that a good representation is one in which the data has a distribution that is easy to model. For this purpose, a non-linear deterministic transformation of the data is learned that maps it to a latent space so as to make the transformed data conform to a factorized distribution, i.e., resulting in independent latent variables. We parametrize this transformation so that computing the determinant of the Jacobian and inverse Jacobian is trivial, yet we maintain the ability to learn complex non-linear transformations, via a composition of simple building blocks, each based on a deep neural network. The training criterion is simply the exact log-likelihood, which is tractable. Unbiased ancestral sampling is also easy. We show that this approach yields good generative models on four image datasets and can be used for inpainting.

1 INTRODUCTION

One of the central questions in unsupervised learning is how to capture complex data distributio that have unknown structure. Deep learning approaches (Bengio, 2009) rely on the learning representation of the data that would capture its most important factors of variation. This raise question: *what is a good representation?* Like in recent work (Kingma and Welling, 2014; Re et al., 2014; Ozair and Bengio, 2014), we take the view that *a good representation is one i the distribution of the data is easy to model*. In this paper, we consider the special case w ask the learner to find a transformation $h = f(x)$ of the data into a new space such that th distribution factorizes, i.e., *the components h_d are independent*:

$$p_H(h) = \prod_d p_{H_d}(h_d).$$

The proposed training criterion is directly derived from the log-likelihood. Mo consider a change of variables $h = f(x)$, which assumes that f is invertible and is the same as the dimension of x, in order to fit a distribution p_H. The change o us:

$$p_X(x) = p_H(f(x))|\det \frac{\partial f(x)}{\partial x}|.$$

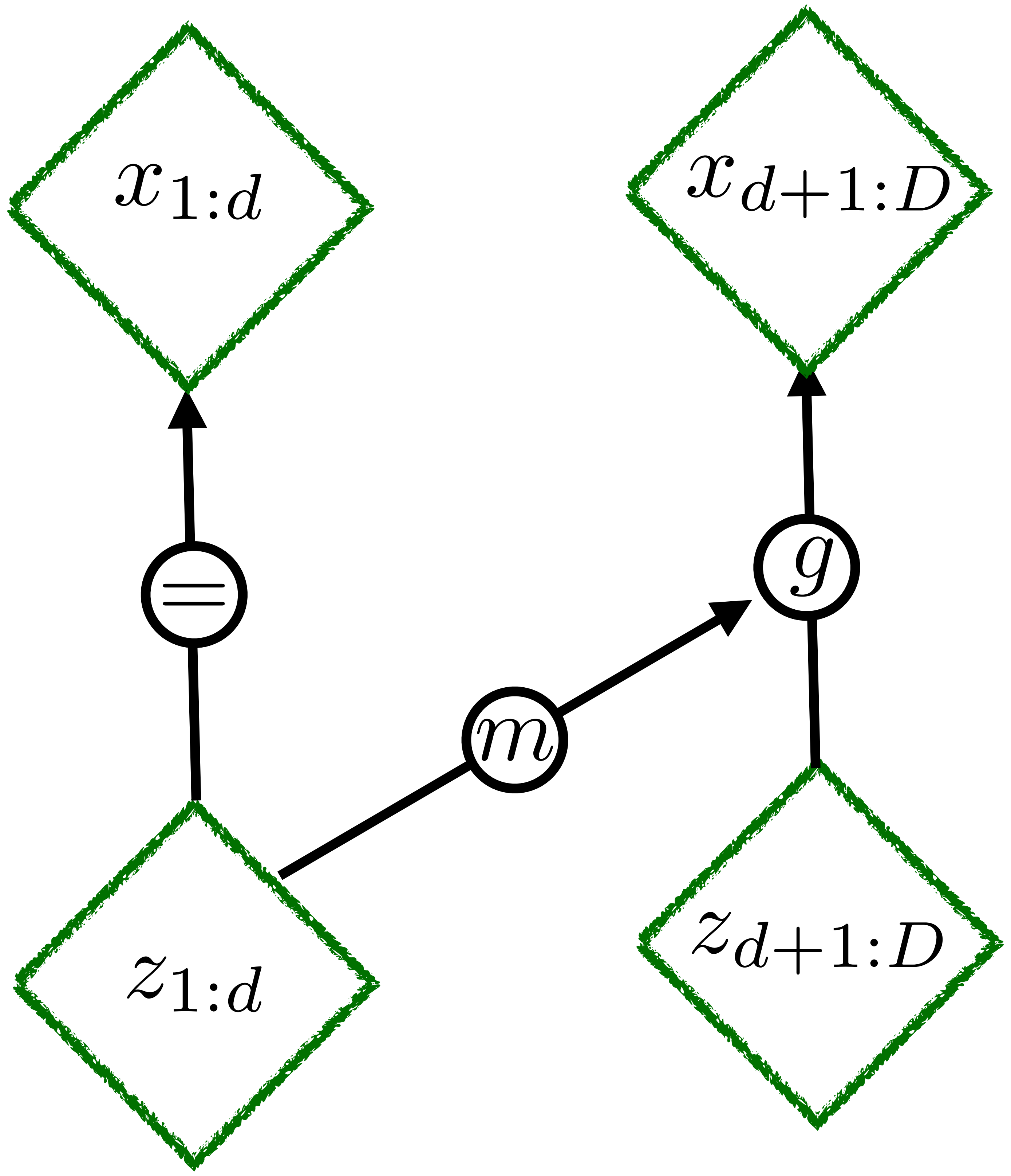
where $\frac{\partial f(x)}{\partial x}$ is the Jacobian matrix of function f at x. In this paper, w inant of the Jacobian is trivially obtained. Moreover, its inverse f to sample from $p_X(x)$ easily as follows:

$$h \sim p_H(h)$$
$$x = f^{-1}(h)$$

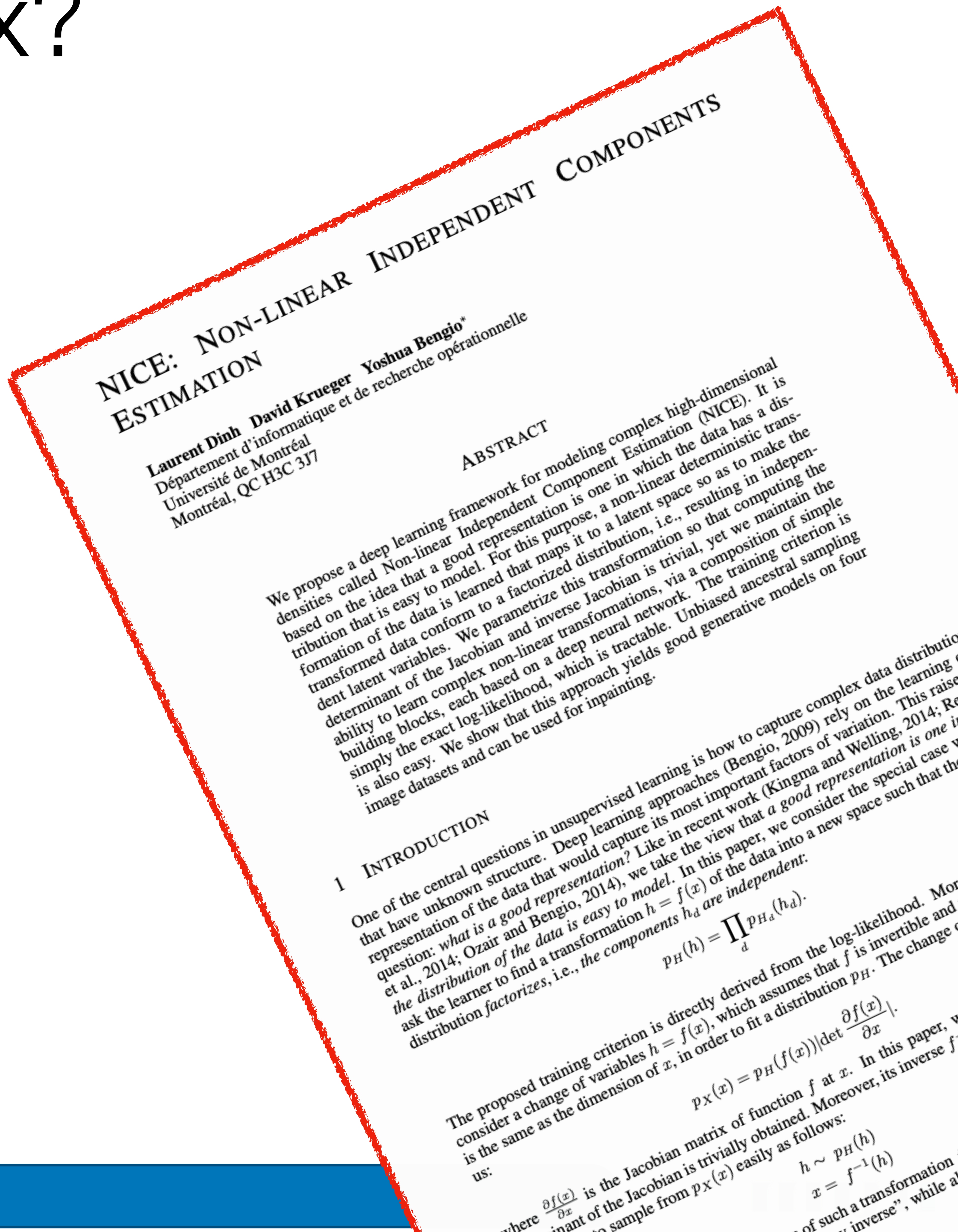
of such a transformation "inverse", while al

How to ensure lower-triangular Jacobian matrix?

Coupling layer



$$z \in \mathbb{R}^D$$



NICE: NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION

Laurent Dinh David Krueger Yoshua Bengio*
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

ABSTRACT

We propose a deep learning framework for modeling complex high-dimensional densities called Non-linear Independent Component Estimation (NICE). It is based on the idea that a good representation is one in which the data has a distribution that is easy to model. For this purpose, a non-linear deterministic transformation of the data is learned that maps it to a latent space so as to make the transformed data conform to a factorized distribution, i.e., resulting in independent latent variables. We parametrize this transformation so that computing the determinant of the Jacobian and inverse Jacobian is trivial, yet we maintain the ability to learn complex non-linear transformations, via a composition of simple building blocks, each based on a deep neural network. The training criterion is simply the exact log-likelihood, which is tractable. Unbiased ancestral sampling is also easy. We show that this approach yields good generative models on four image datasets and can be used for inpainting.

1 INTRODUCTION

One of the central questions in unsupervised learning is how to capture complex data distributio that have unknown structure. Deep learning approaches (Bengio, 2009) rely on the learning o representation of the data that would capture its most important factors of variation. This raise question: *what is a good representation?* Like in recent work (Kingma and Welling, 2014; Re et al., 2014; Ozair and Bengio, 2014), we take the view that *a good representation is one i the distribution of the data is easy to model.* In this paper, we consider the special case w ask the learner to find a transformation $h = f(x)$ of the data into a new space such that th distribution factorizes, i.e., the components h_d are independent:

$$p_H(h) = \prod_d p_{H_d}(h_d).$$

The proposed training criterion is directly derived from the log-likelihood. Mo consider a change of variables $h = f(x)$, which assumes that f is invertible and is the same as the dimension of x, in order to fit a distribution p_H. The change o us:

$$p_X(x) = p_H(f(x))|\det \frac{\partial f(x)}{\partial x}|.$$

where $\frac{\partial f(x)}{\partial x}$ is the Jacobian matrix of function f at x. In this paper, w

Neural Importance Sampling

Thomas Müller
Brian McWilliams
Fabrice Rousselle
Markus Gross
Jan Novák

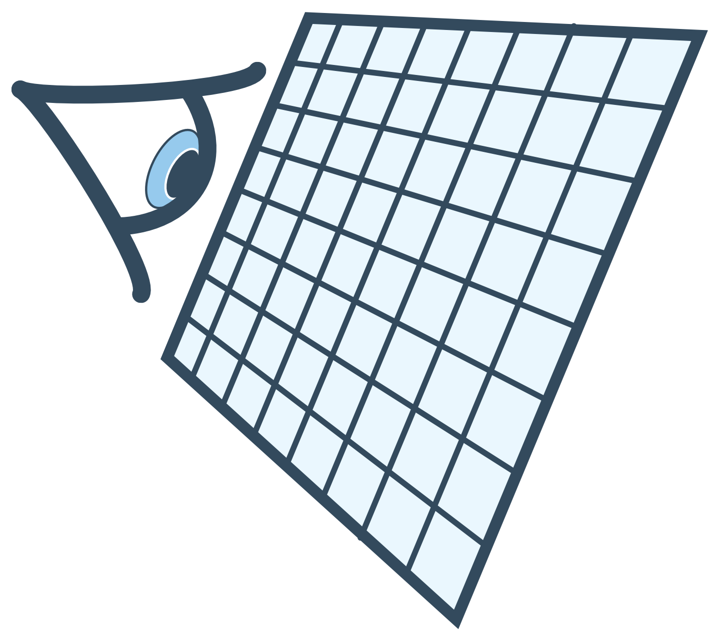
Affiliation: NVIDIA

Work done while at: ETH zürich, Disney Research, cgl computer graphics laboratory


Render time: sometimes >100 cpu-hours


512 paths per pixel
Path tracing
Neural path guiding

Path tracing: BSDF sampling

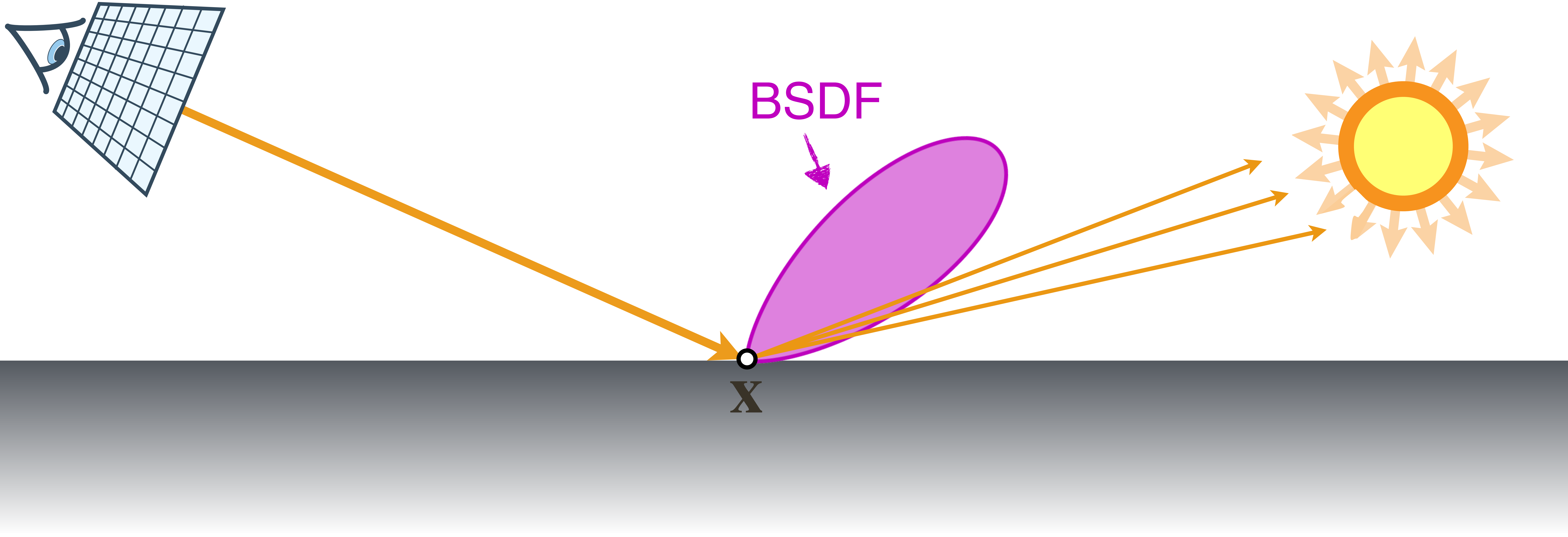


Path tracing: BSDF sampling

Path tracing: BSDF sampling

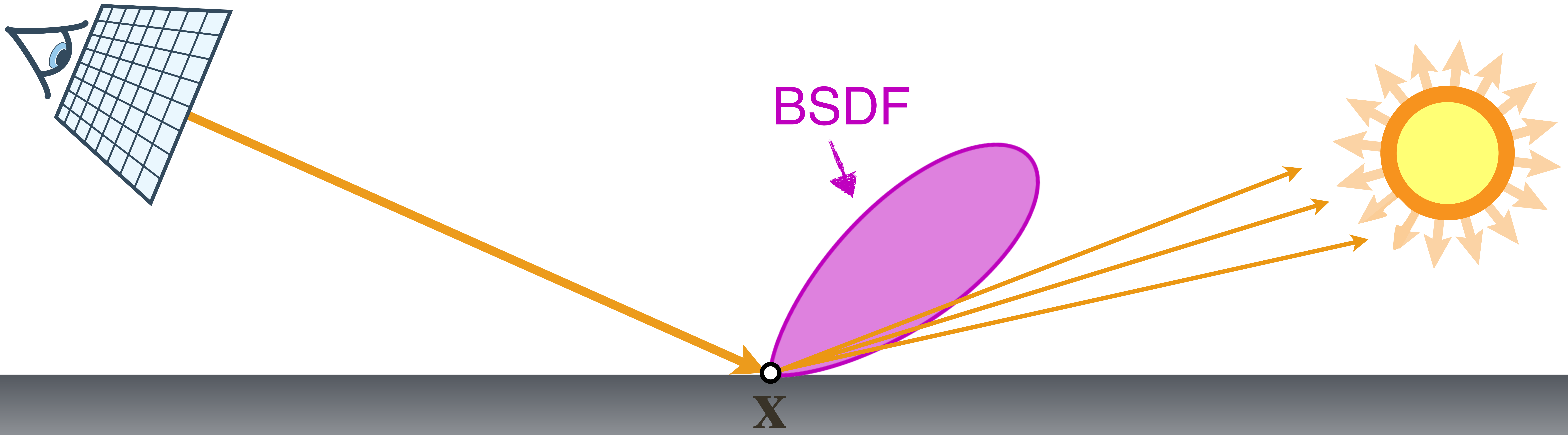
Path tracing: direct-illumination sampling

Path tracing: direct-illumination sampling



Path tracing: direct-illumination sampling

Multiple Importance Sampling
[Veach and Guibas 1995]



Where is path guiding useful?

Where is path guiding useful?

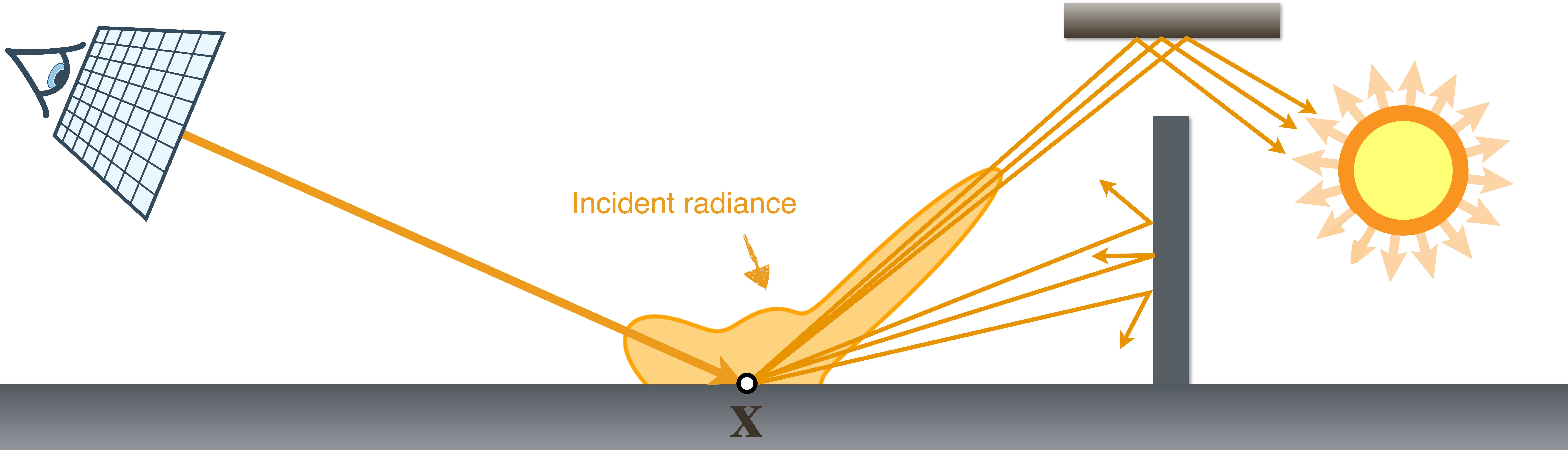
Where is path guiding useful?

Where is path guiding useful?

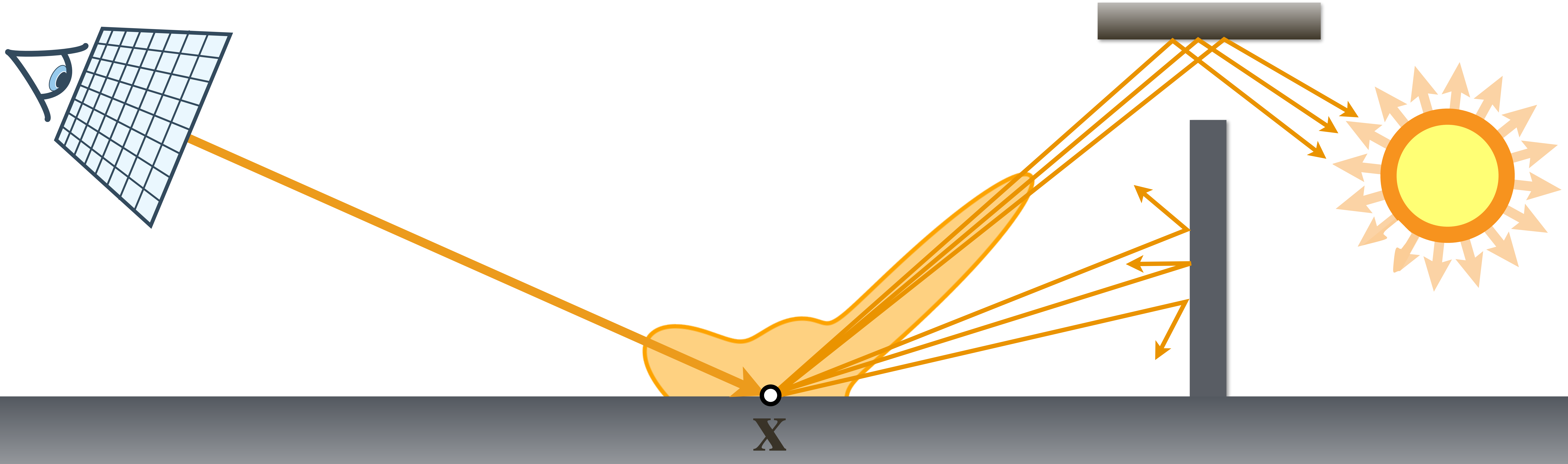
Where is path guiding useful?

Where is path guiding useful?

Goal: Sample proportional to incident radiance.

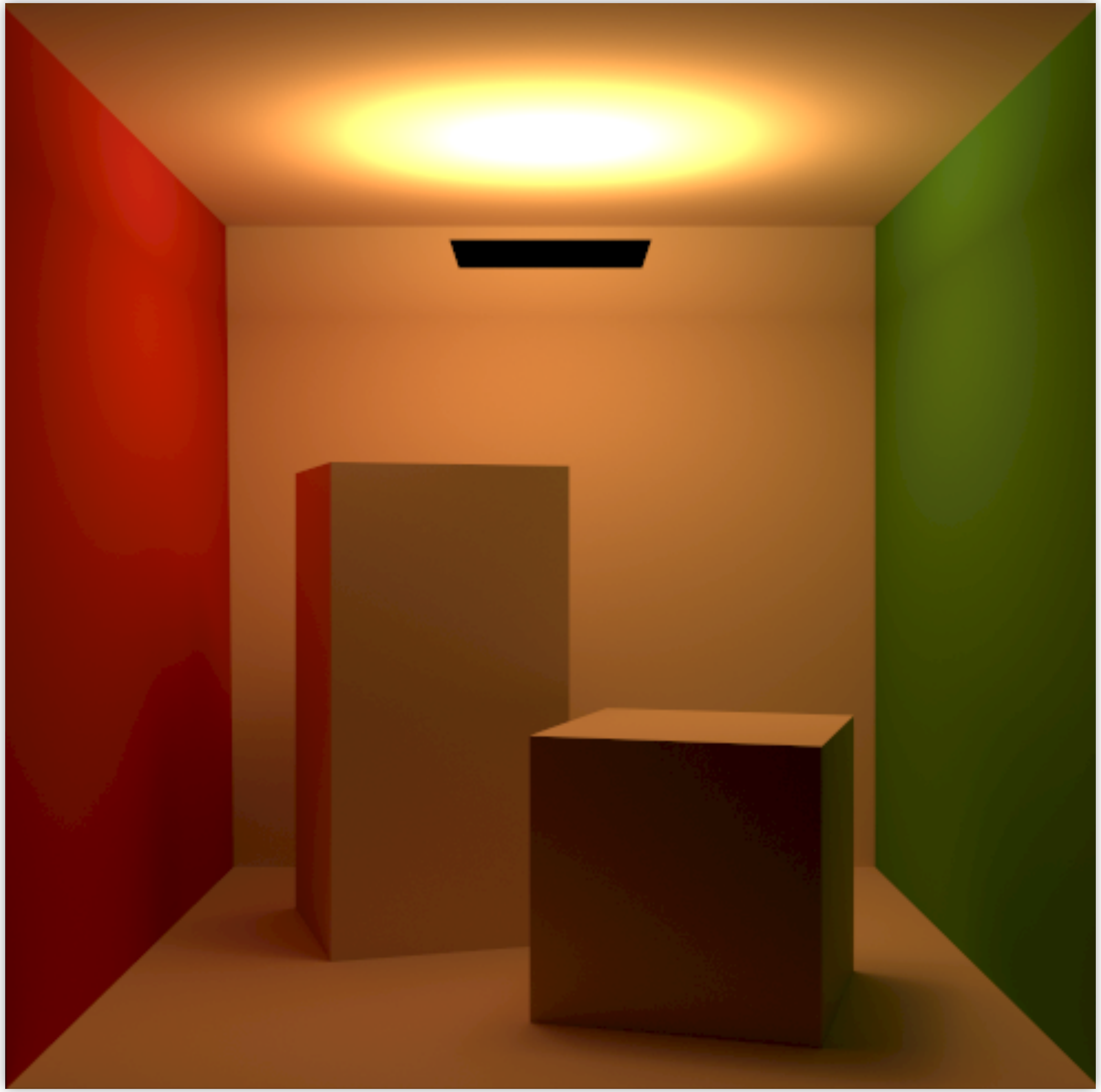


Where is path guiding useful?

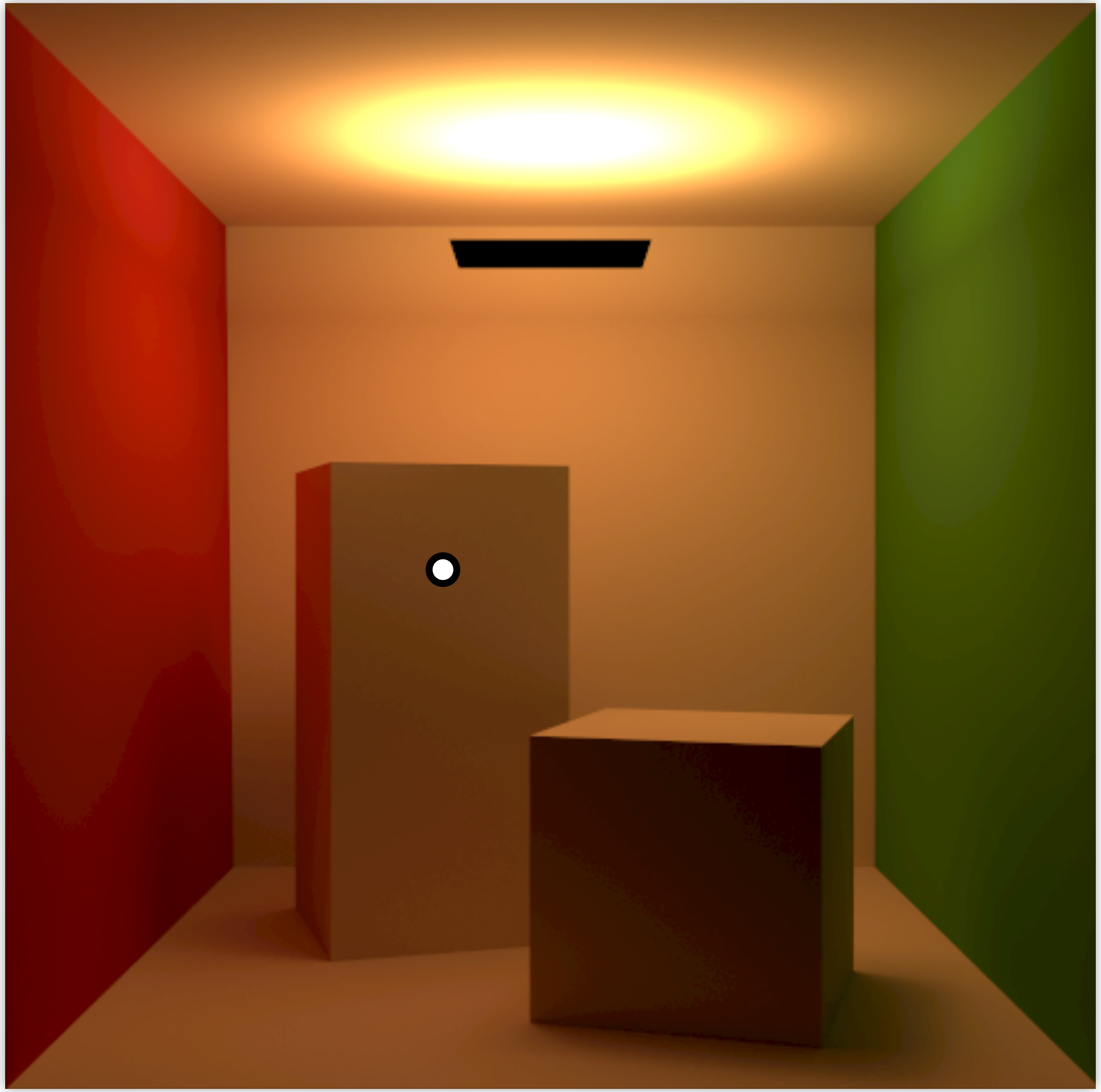


Where is path guiding useful?

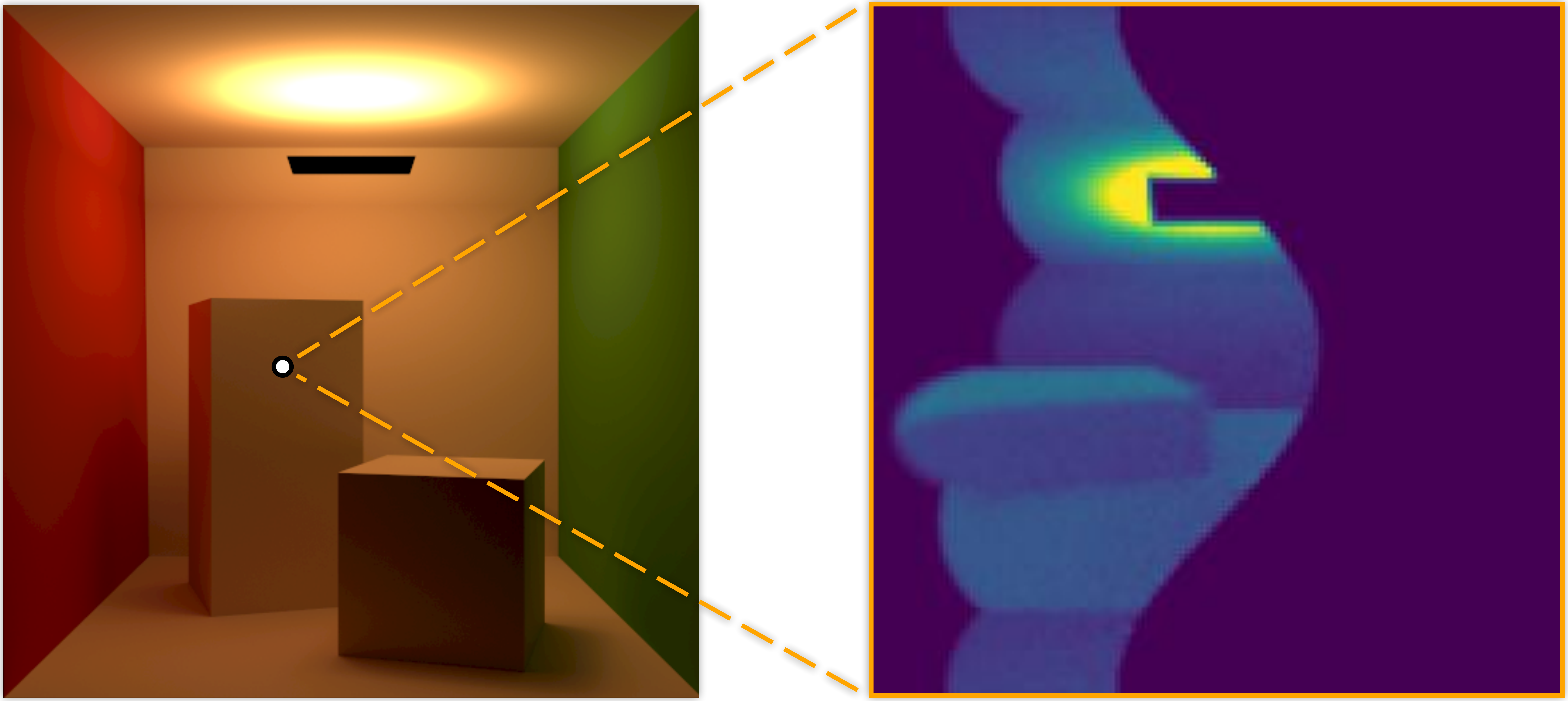
Learning incident radiance in a Cornell box



Learning incident radiance in a Cornell box

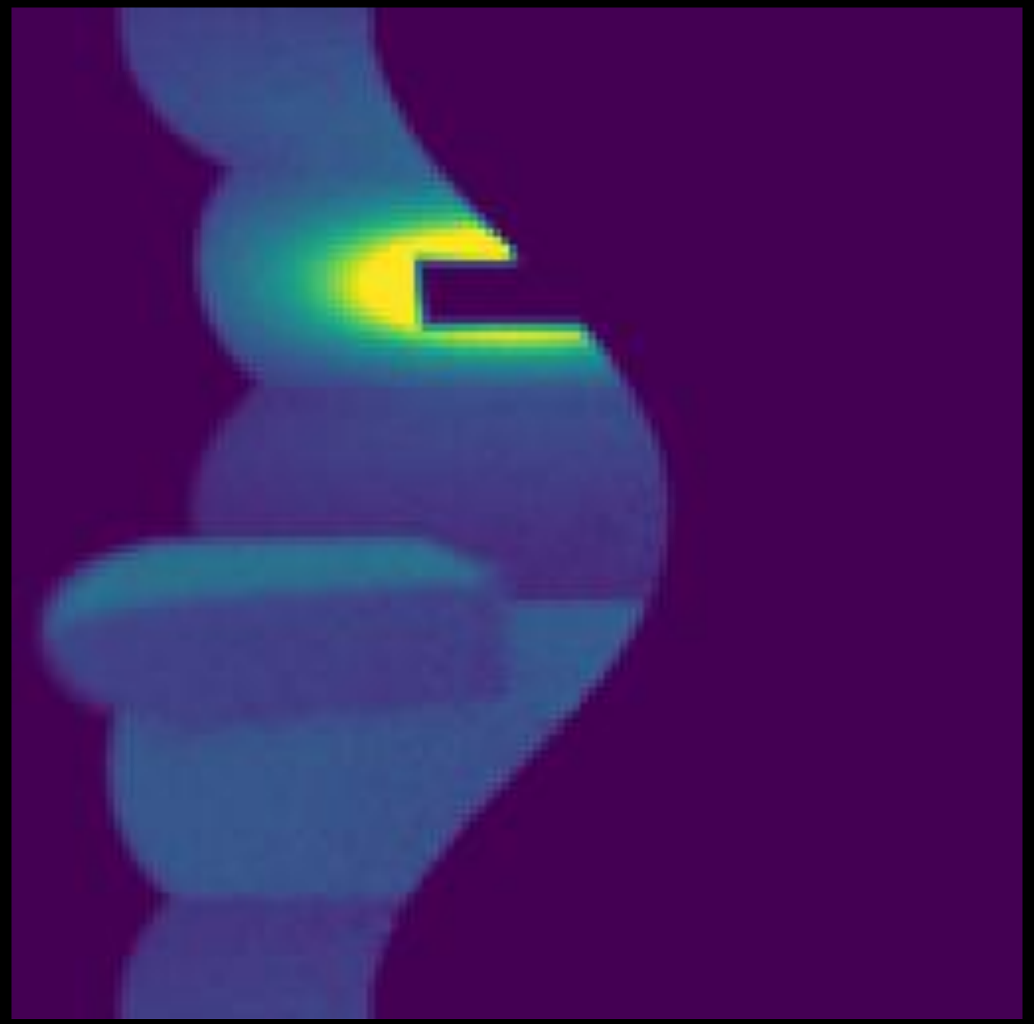


Learning incident radiance in a Cornell box



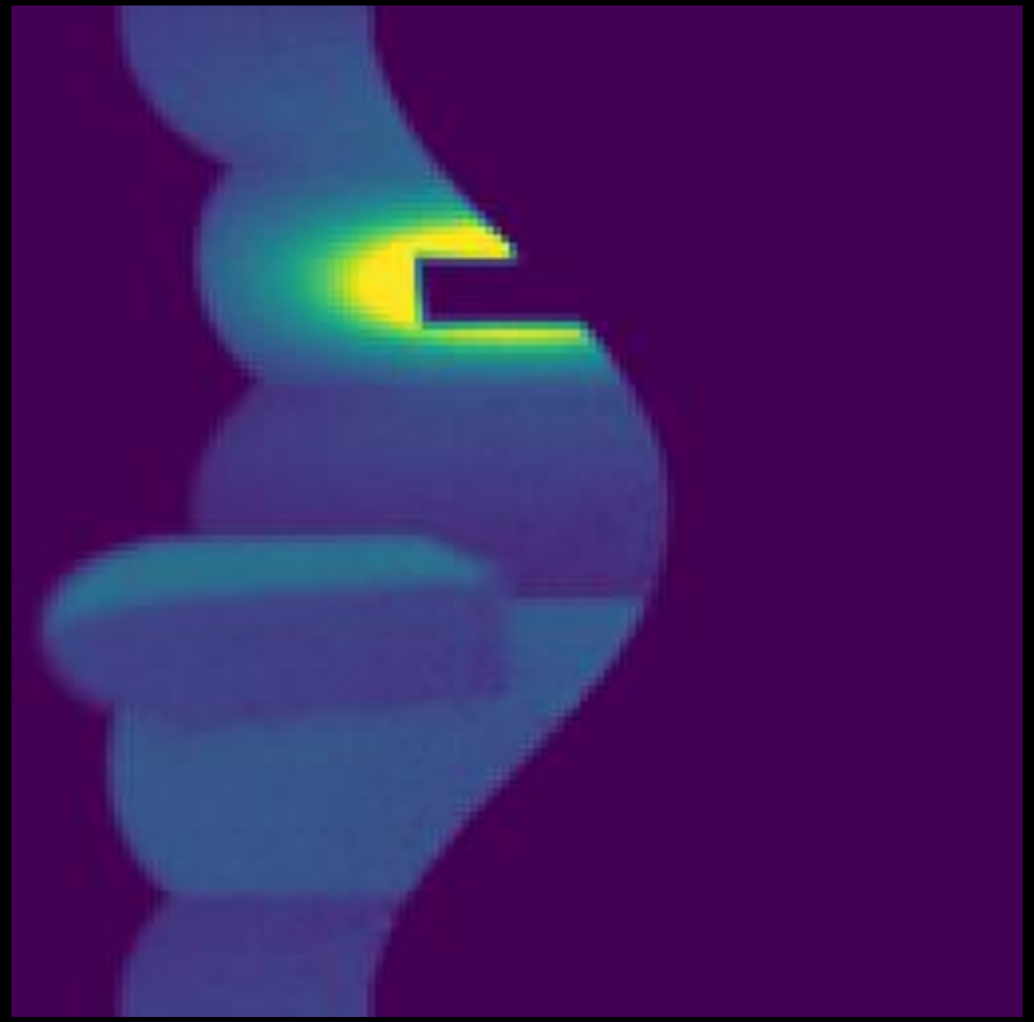
Neural networks as function approximators

Reference

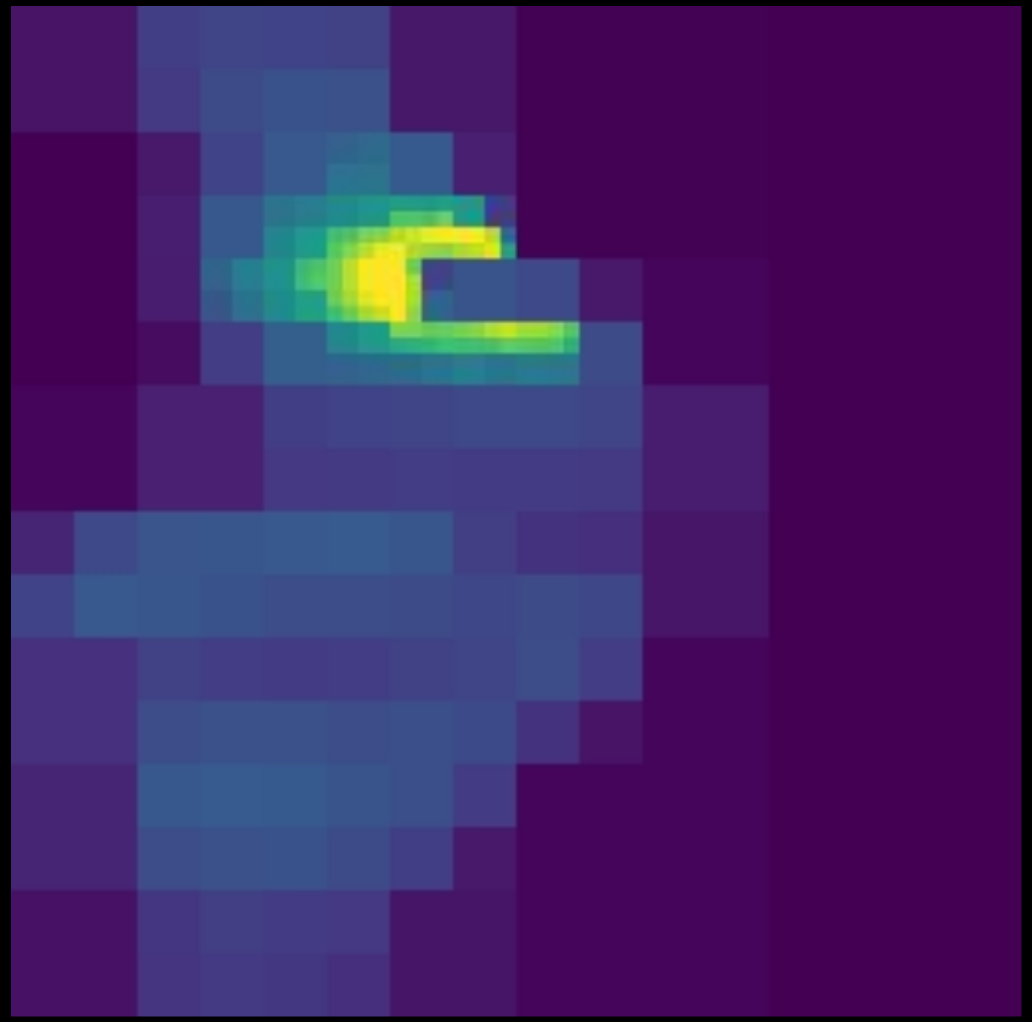


Neural networks as function approximators

Reference

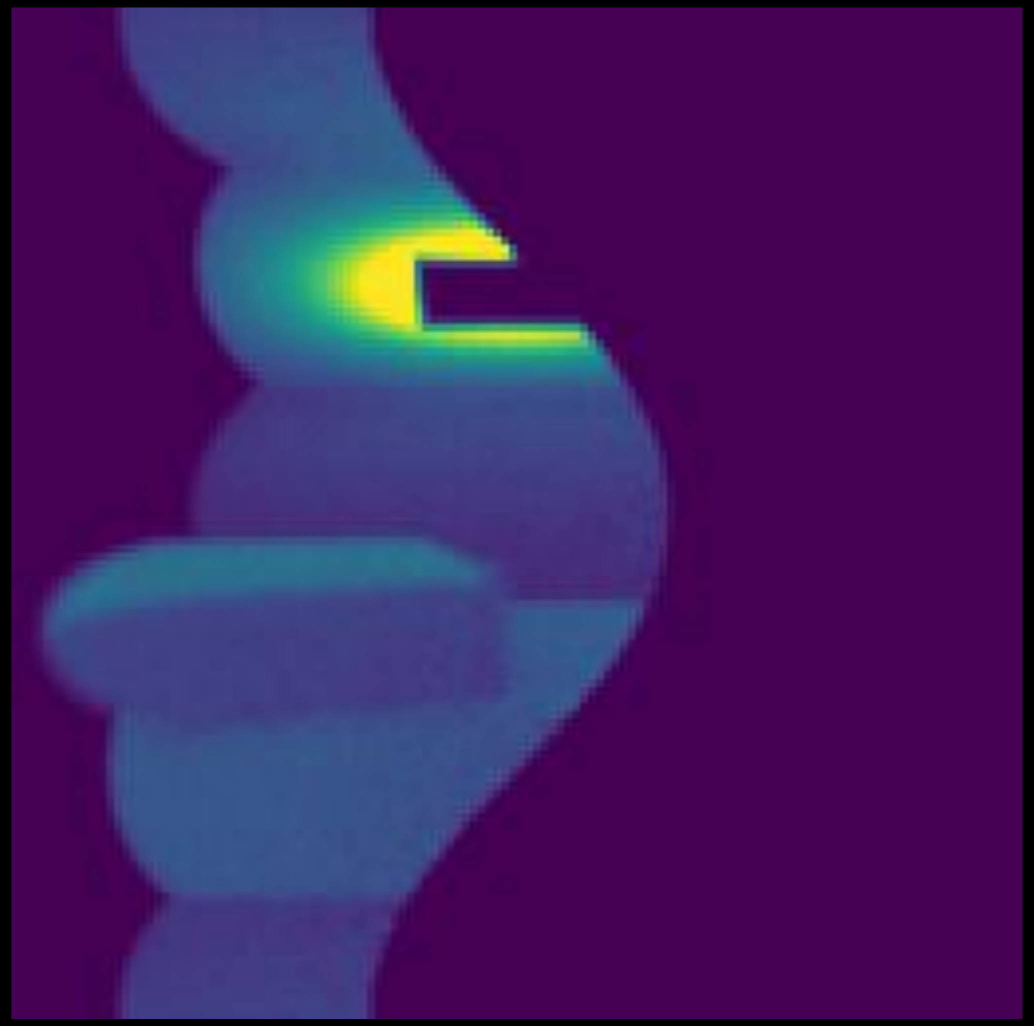


SD-tree [Müller et al. 2017]

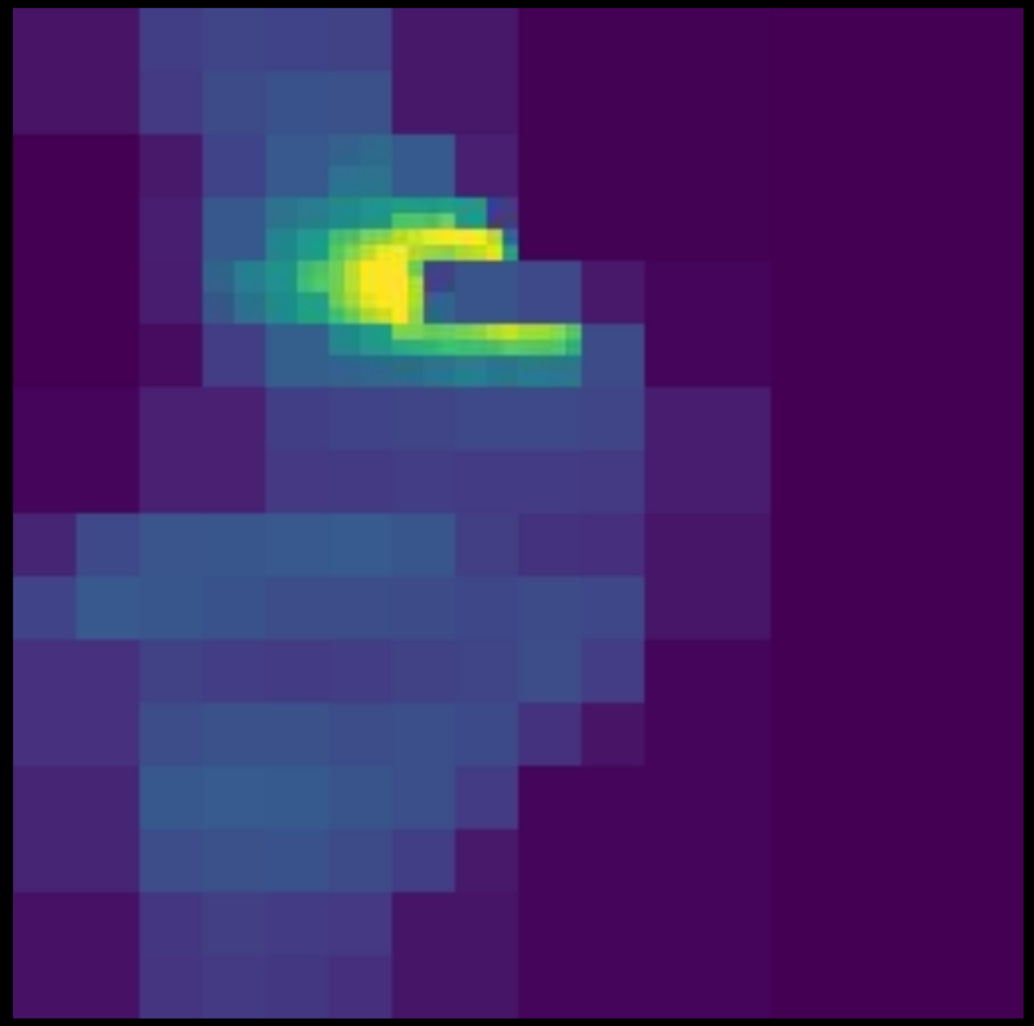


Neural networks as function approximators

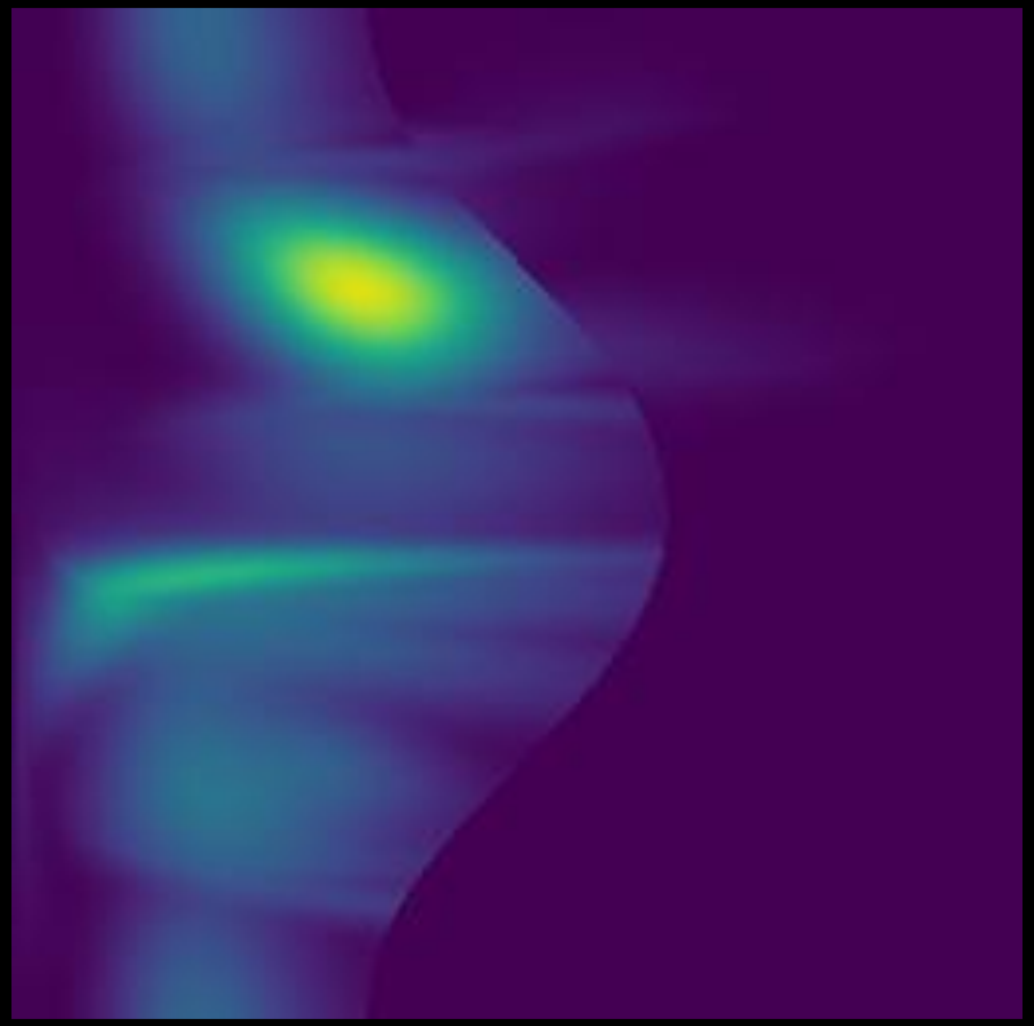
Reference



SD-tree [Müller et al. 2017]

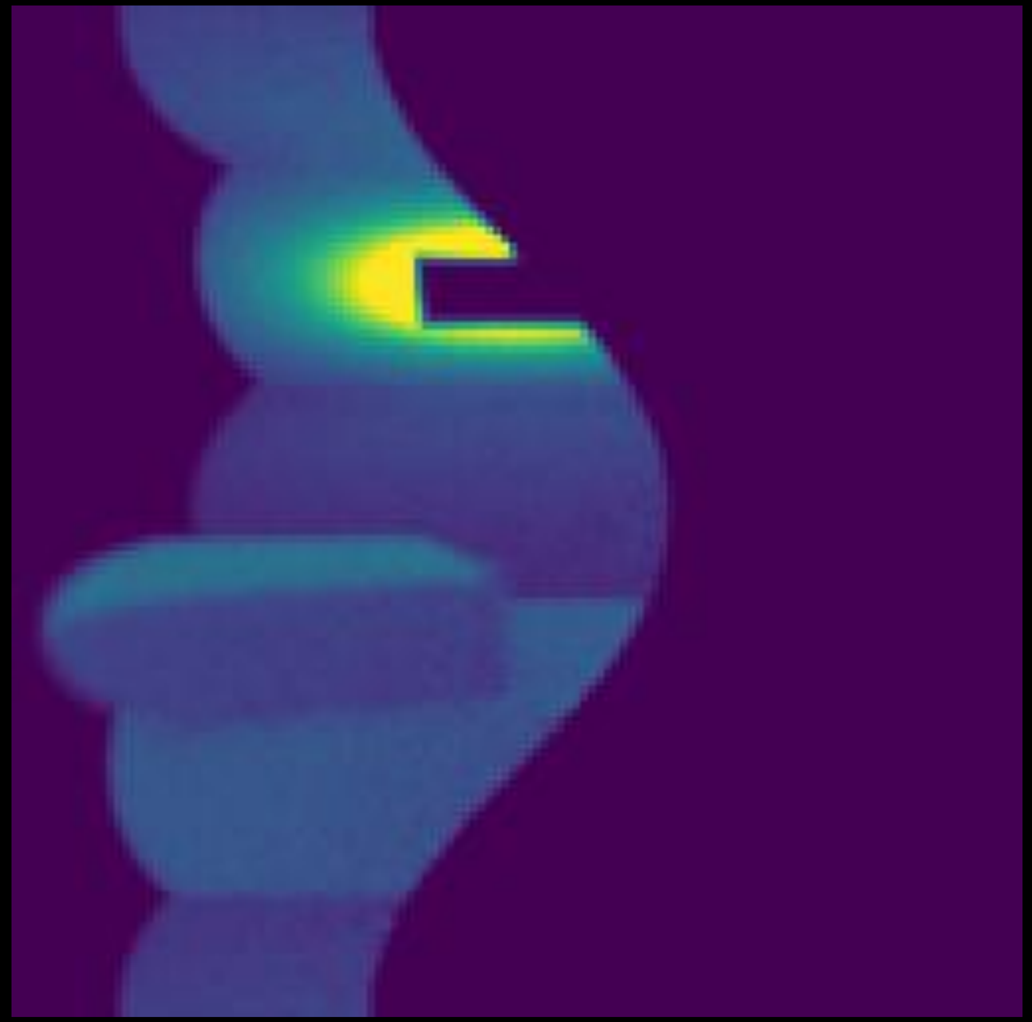


GMM [Vorba et al. 2014]

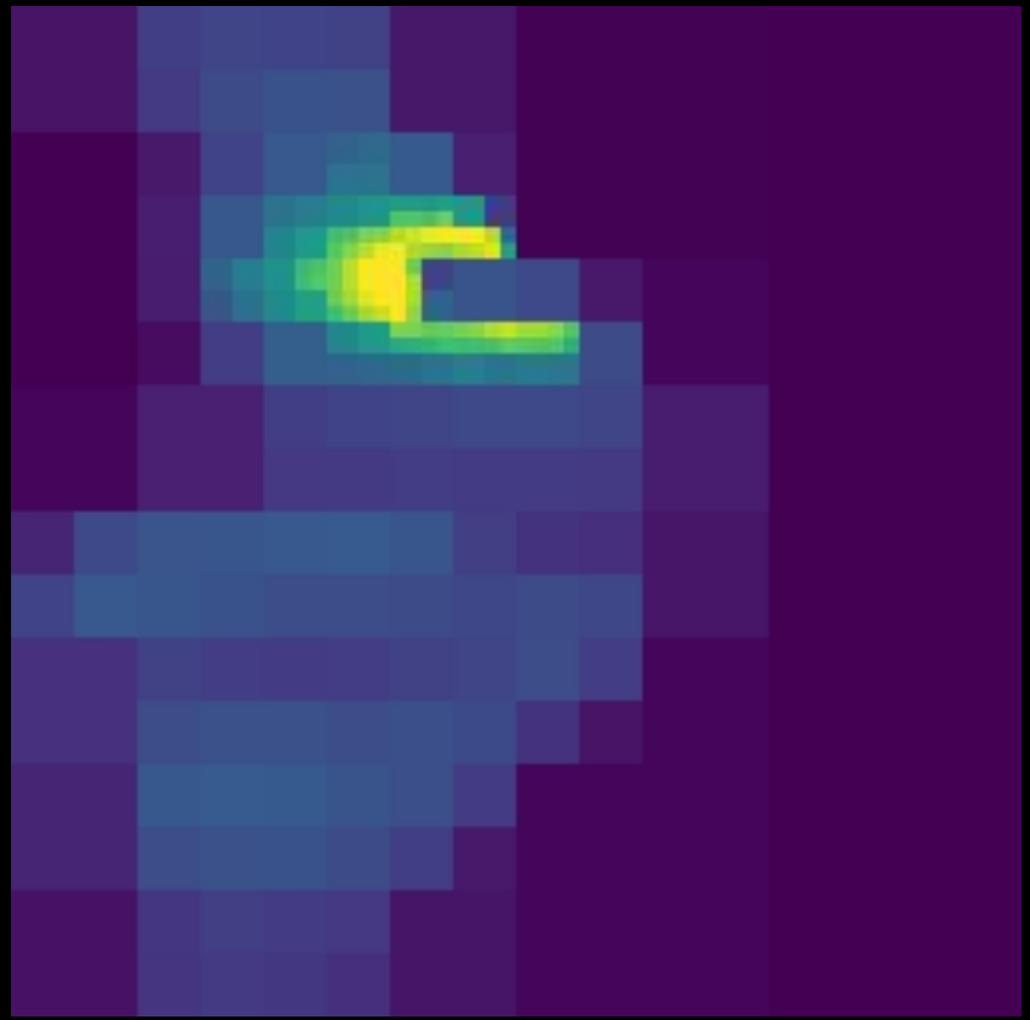


Neural networks as function approximators

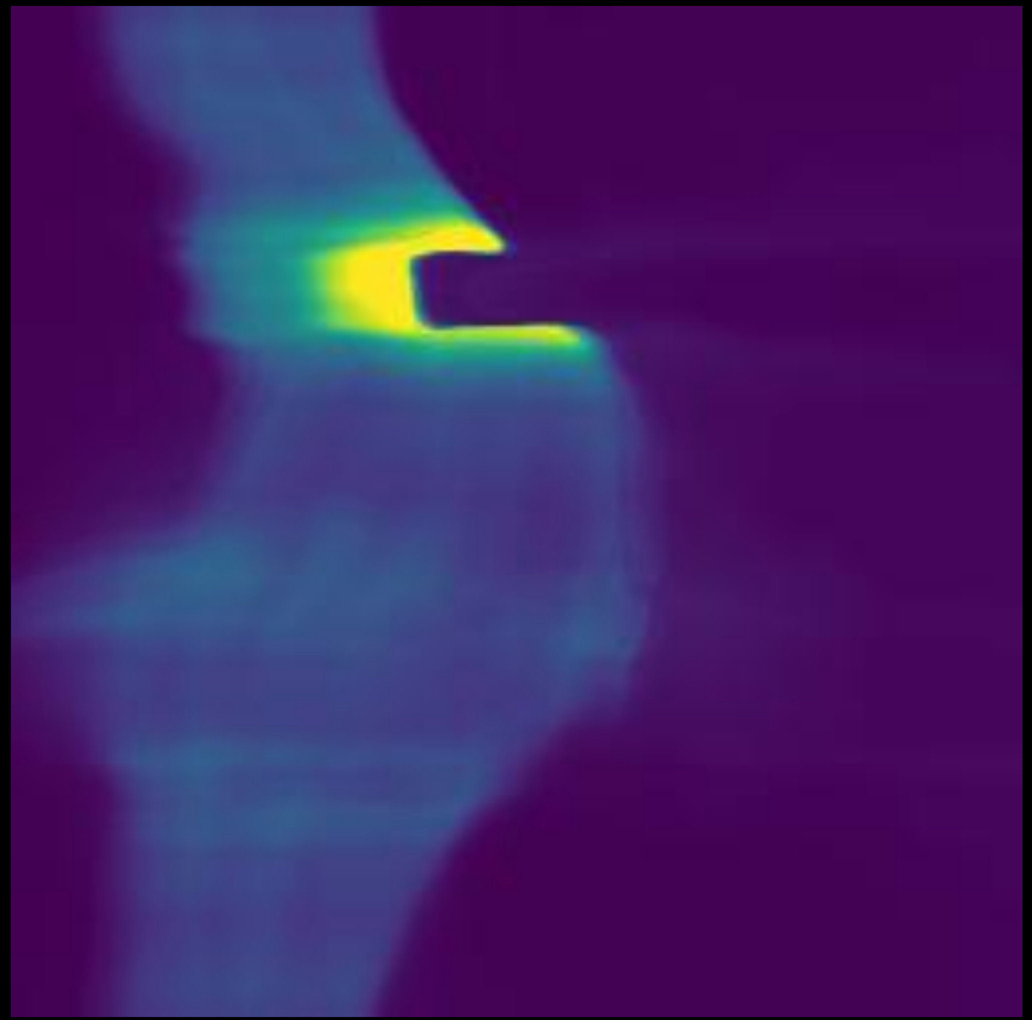
Reference



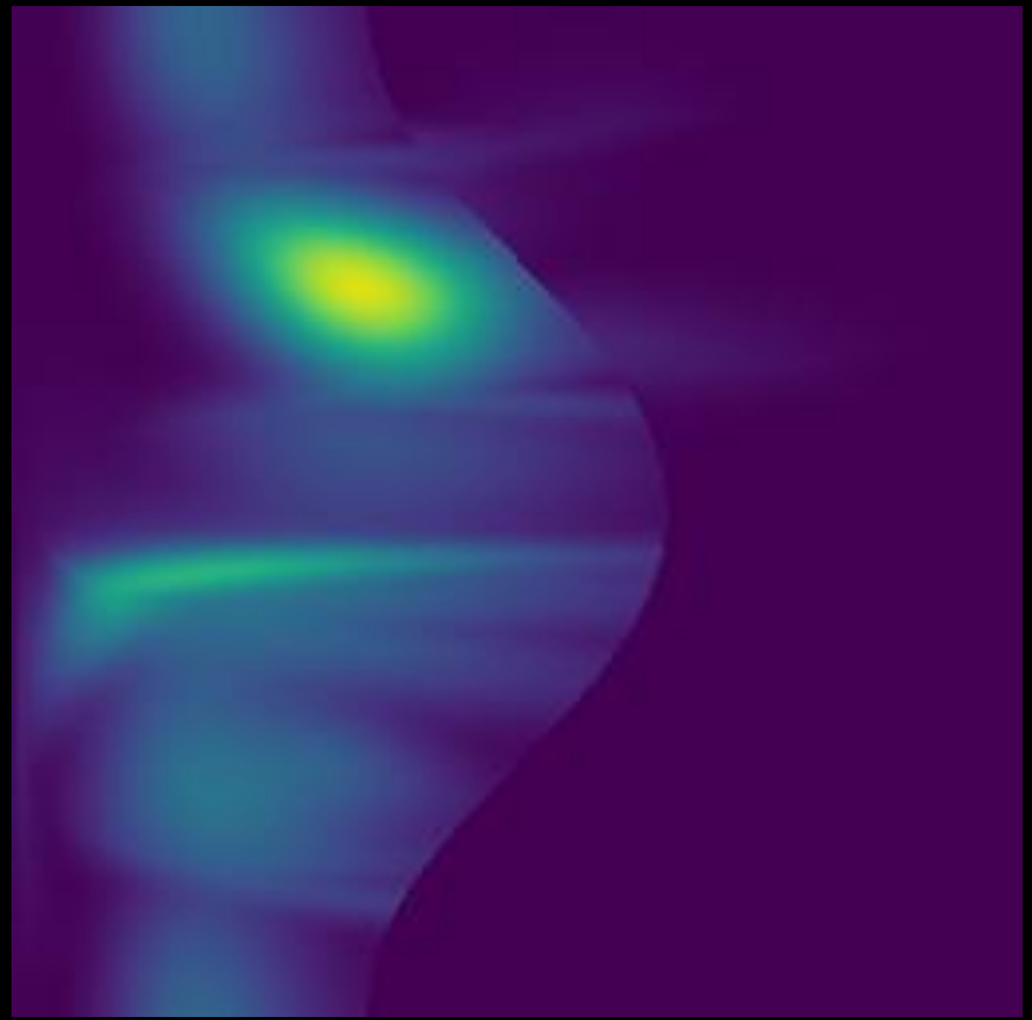
SD-tree [Müller et al. 2017]



Neural Network



GMM [Vorba et al. 2014]

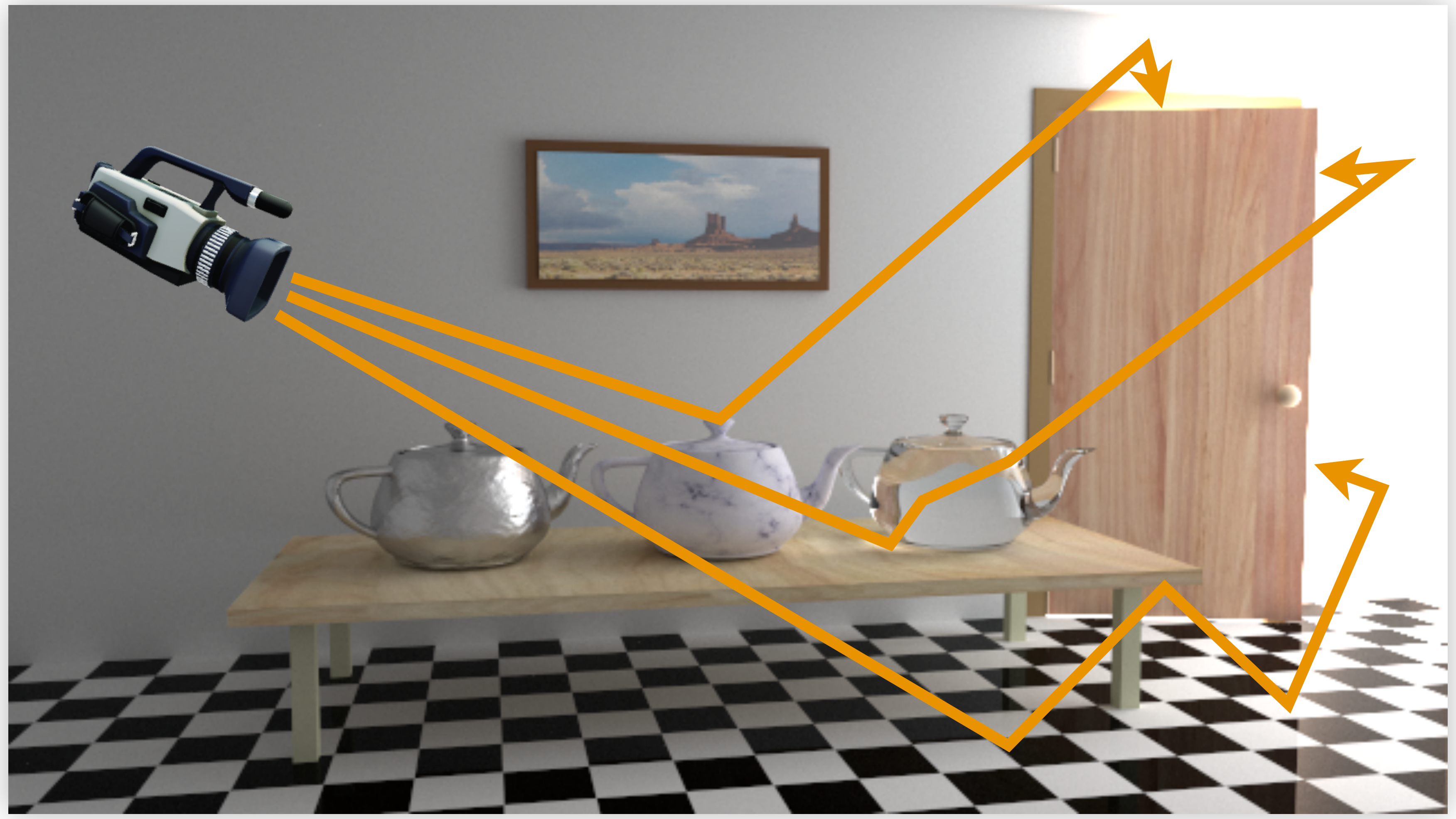



Directional distribution
Neural network
Reference
SD-tree
Gaussian mixture

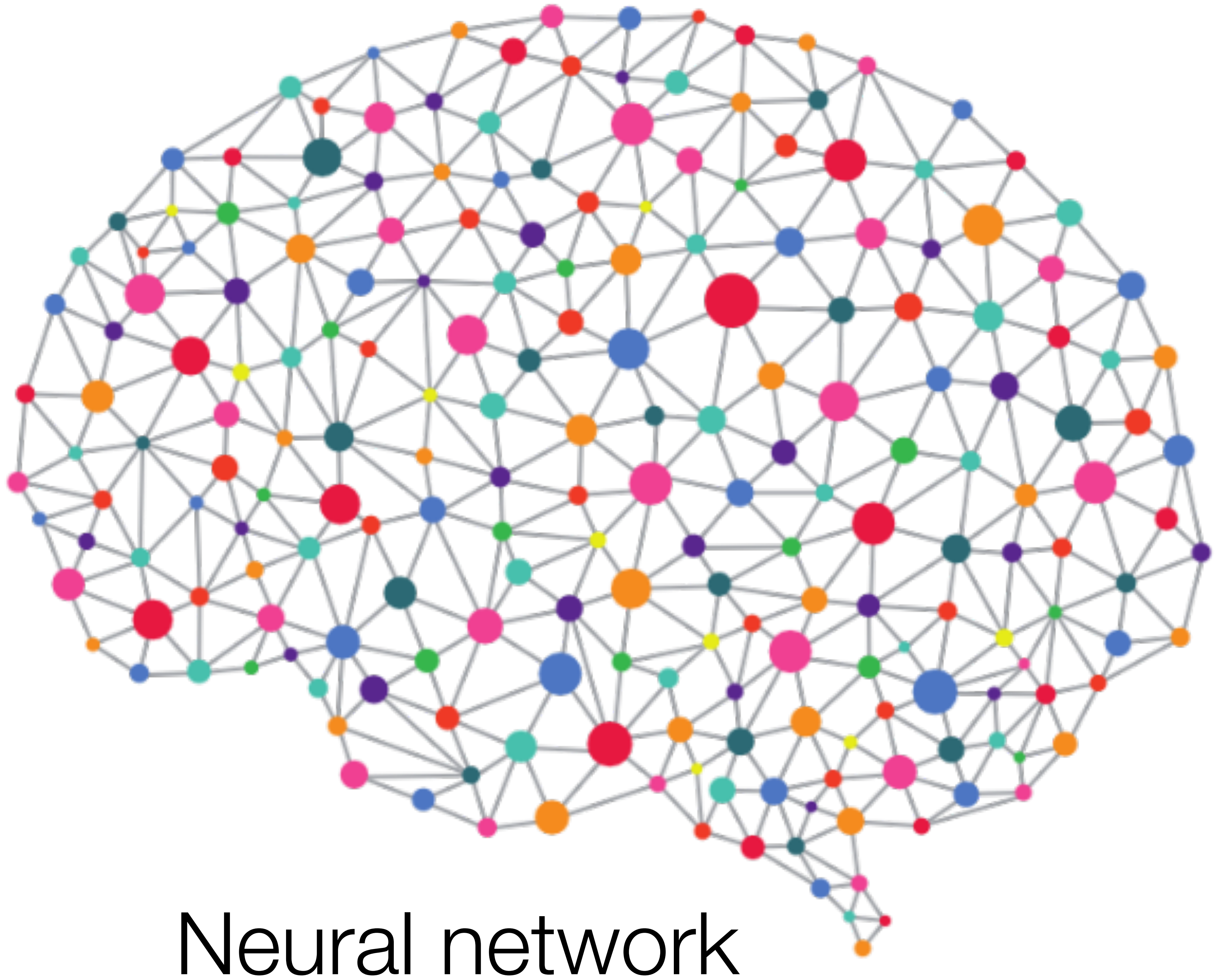

Directional distribution
Neural network
Reference
SD-tree
Gaussian mixture


Directional distribution
Neural network
Reference
SD-tree
Gaussian mixture

Neural path guiding overview



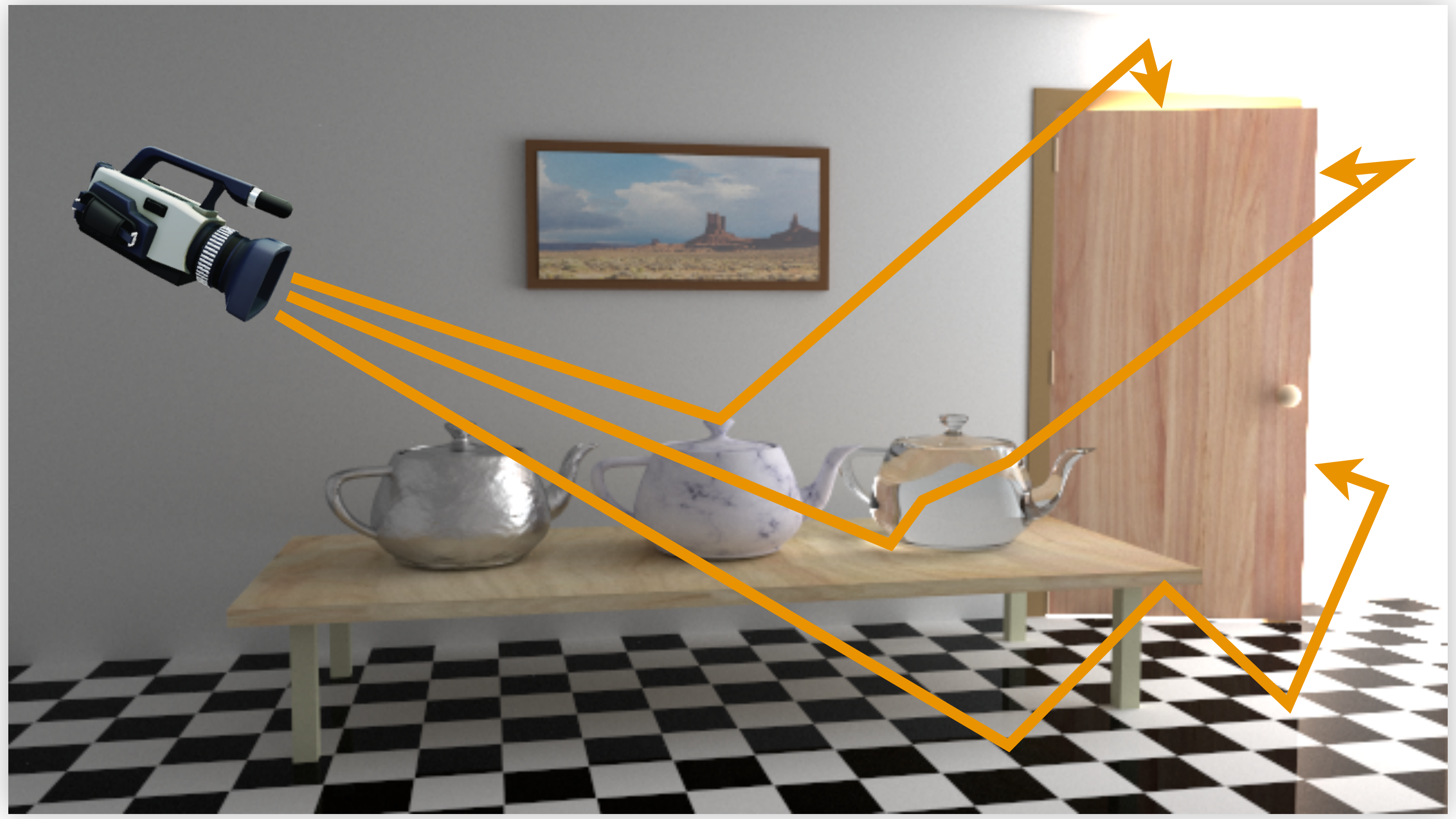
Path tracer



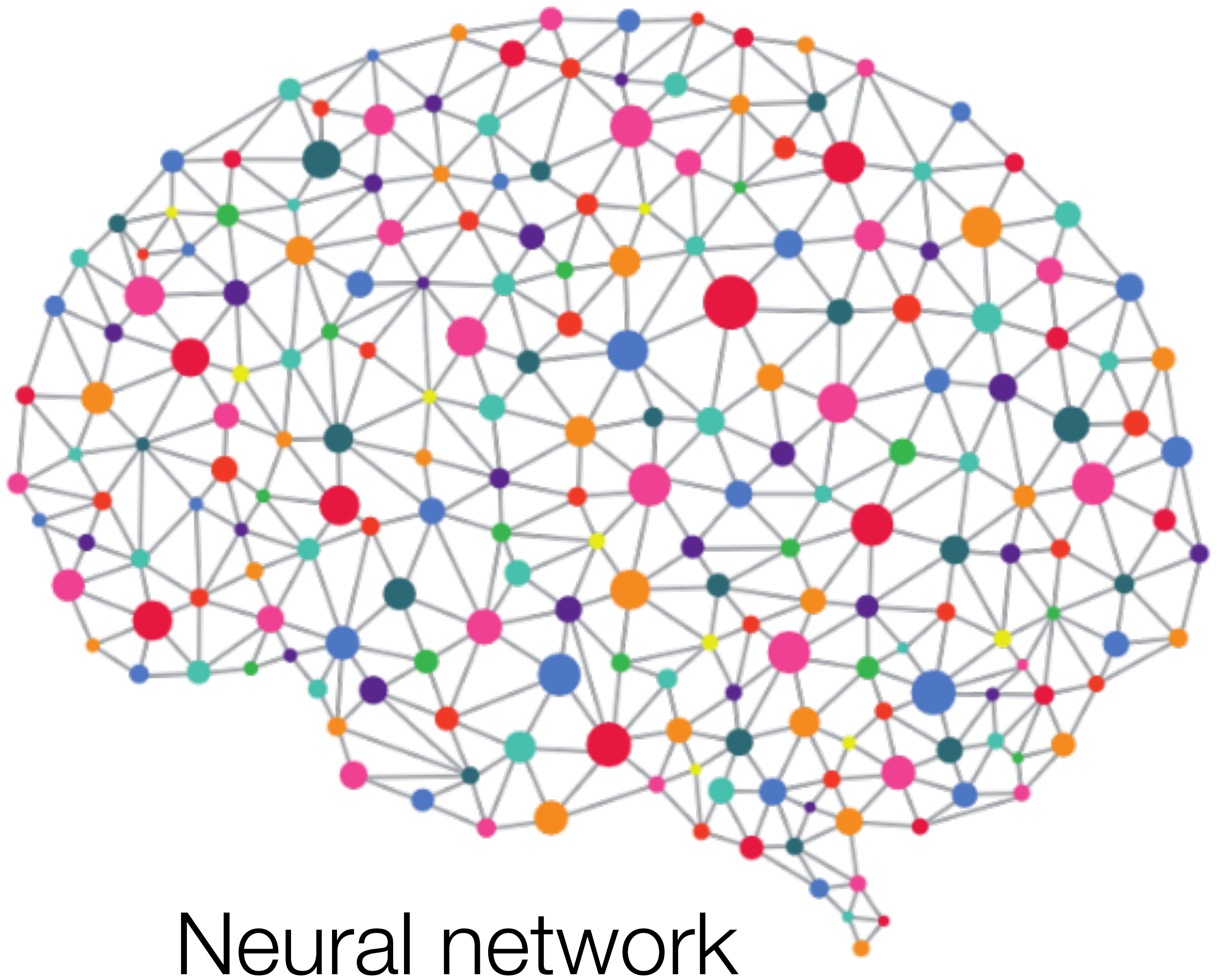
Neural network

Neural path guiding overview

Sample

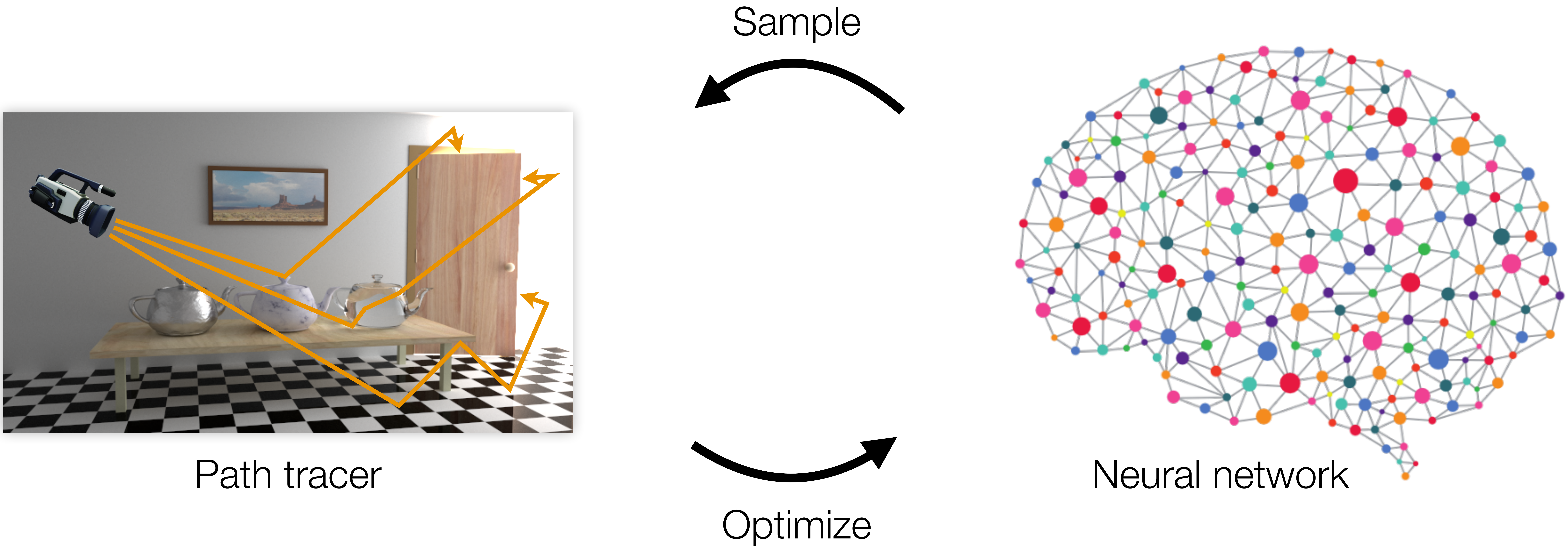


Path tracer



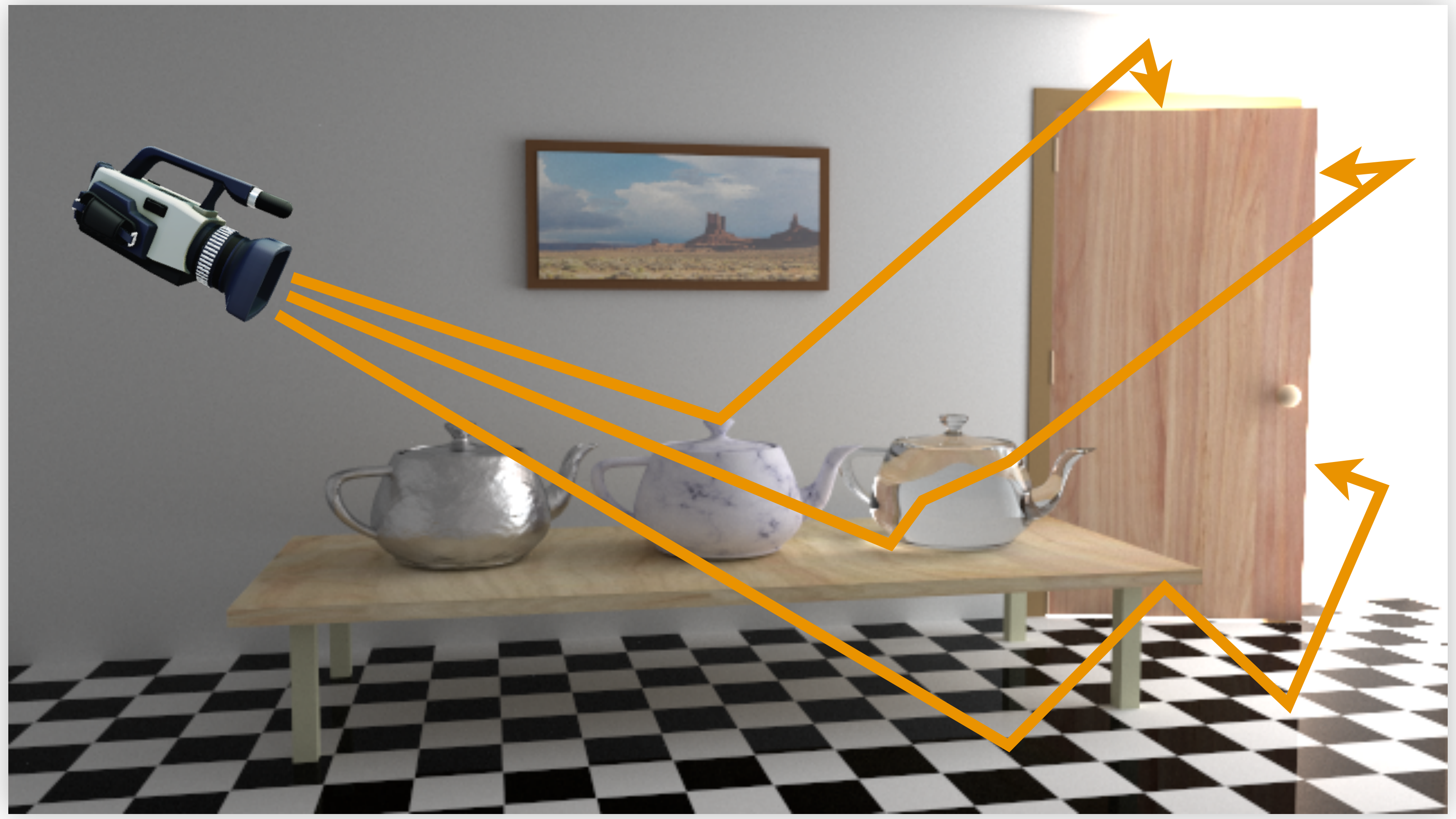
Neural network

Neural path guiding overview

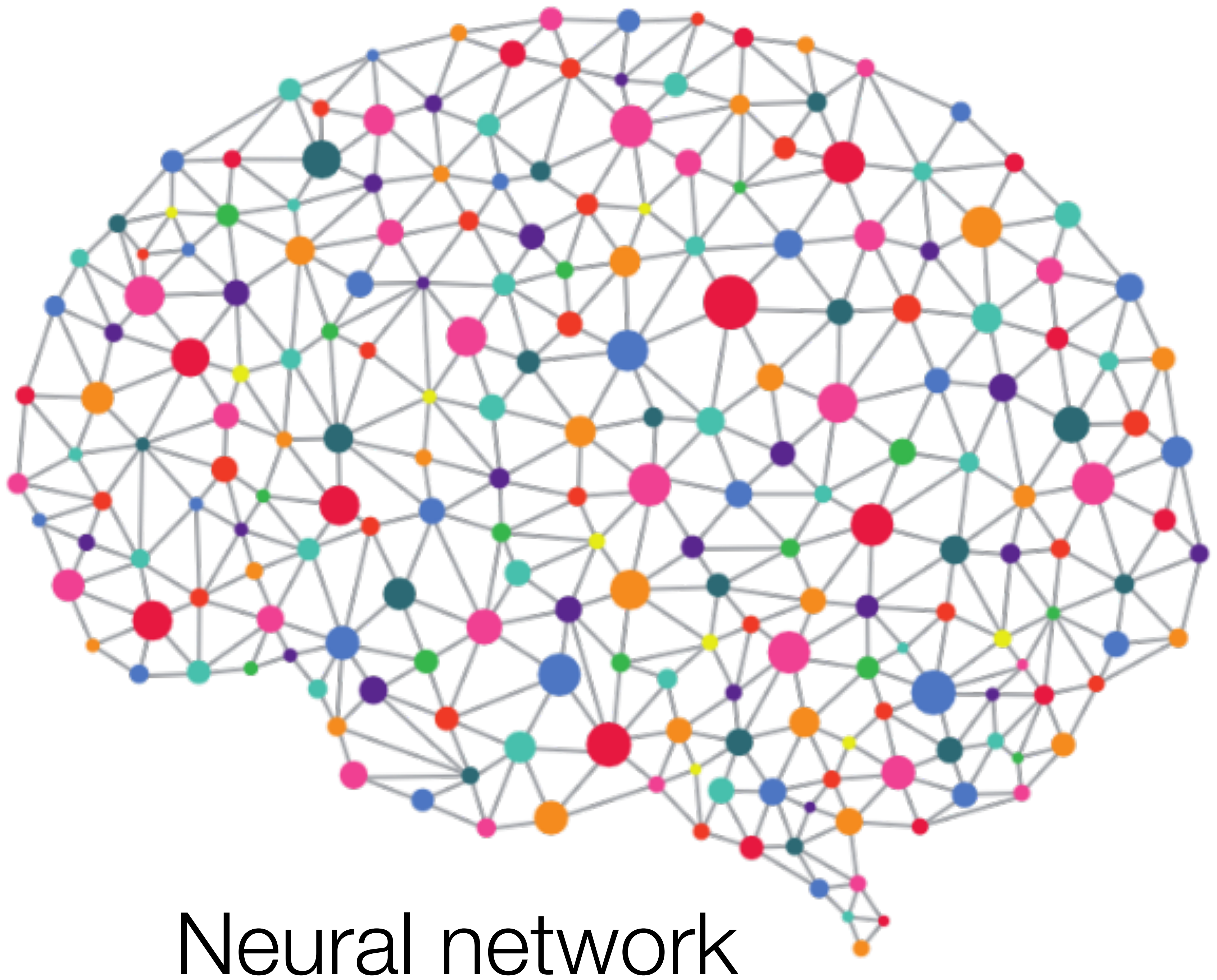


Neural path guiding overview

Sample



Path tracer

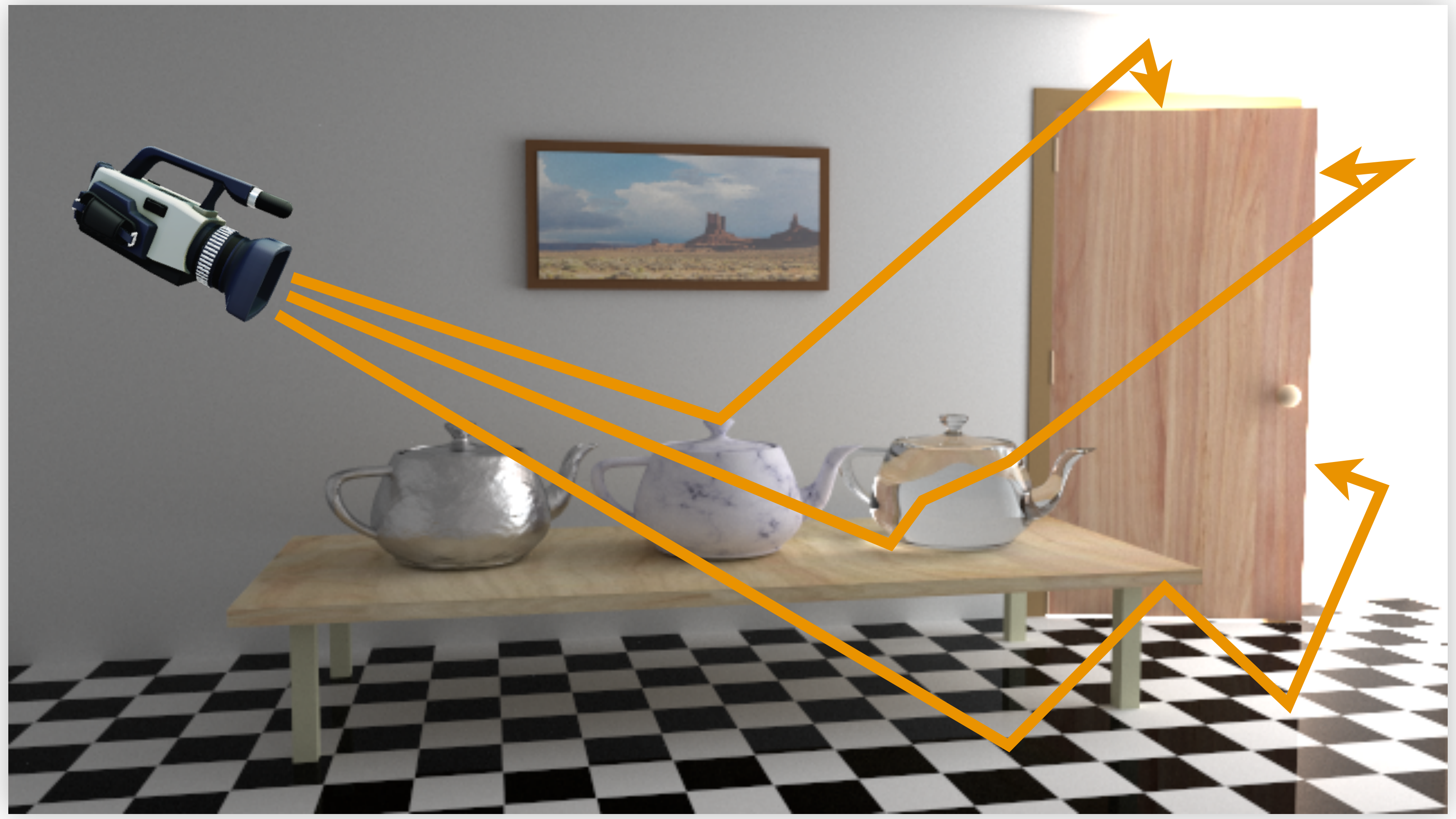


Neural network

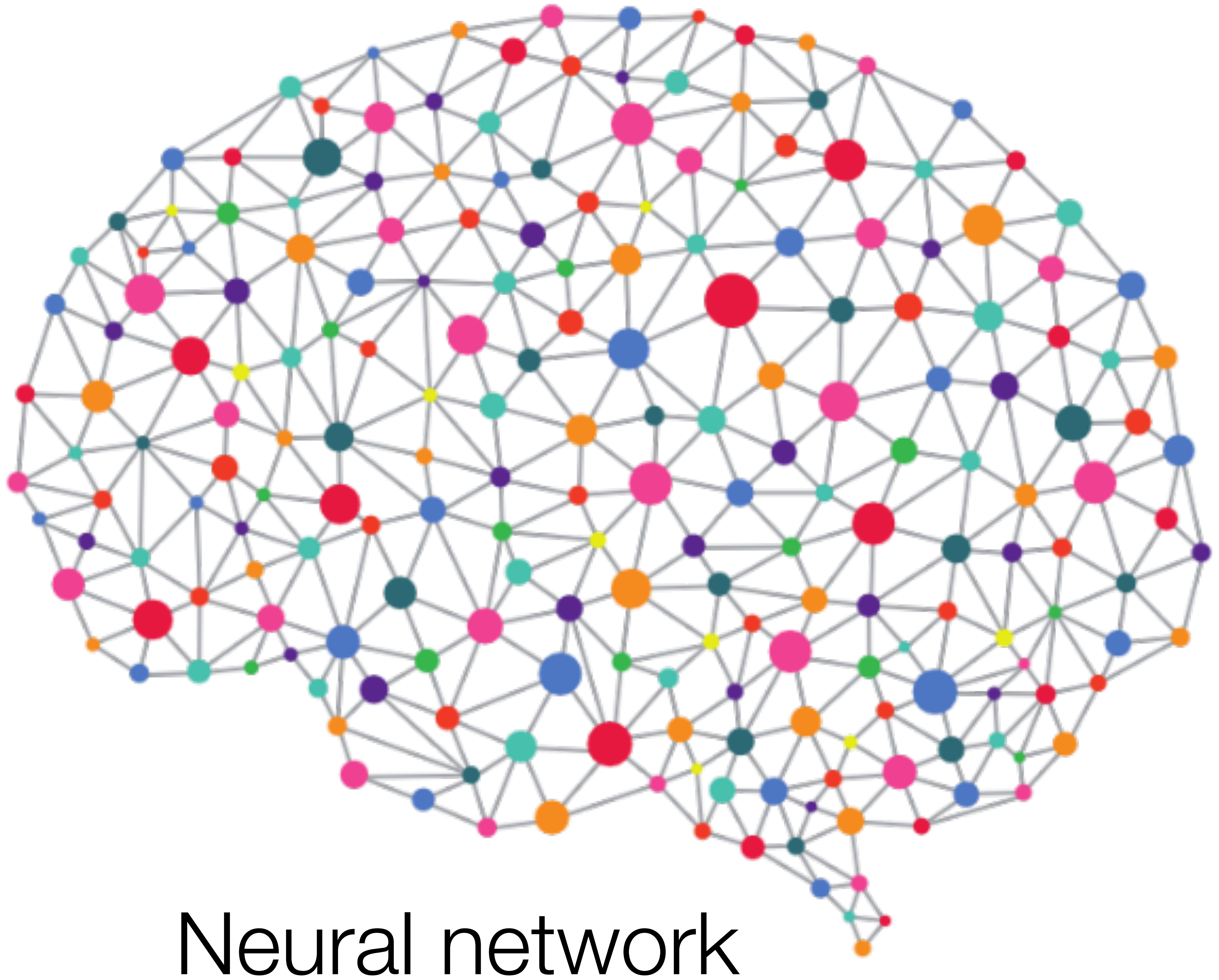
Optimize

Neural path guiding overview

Sample



Path tracer

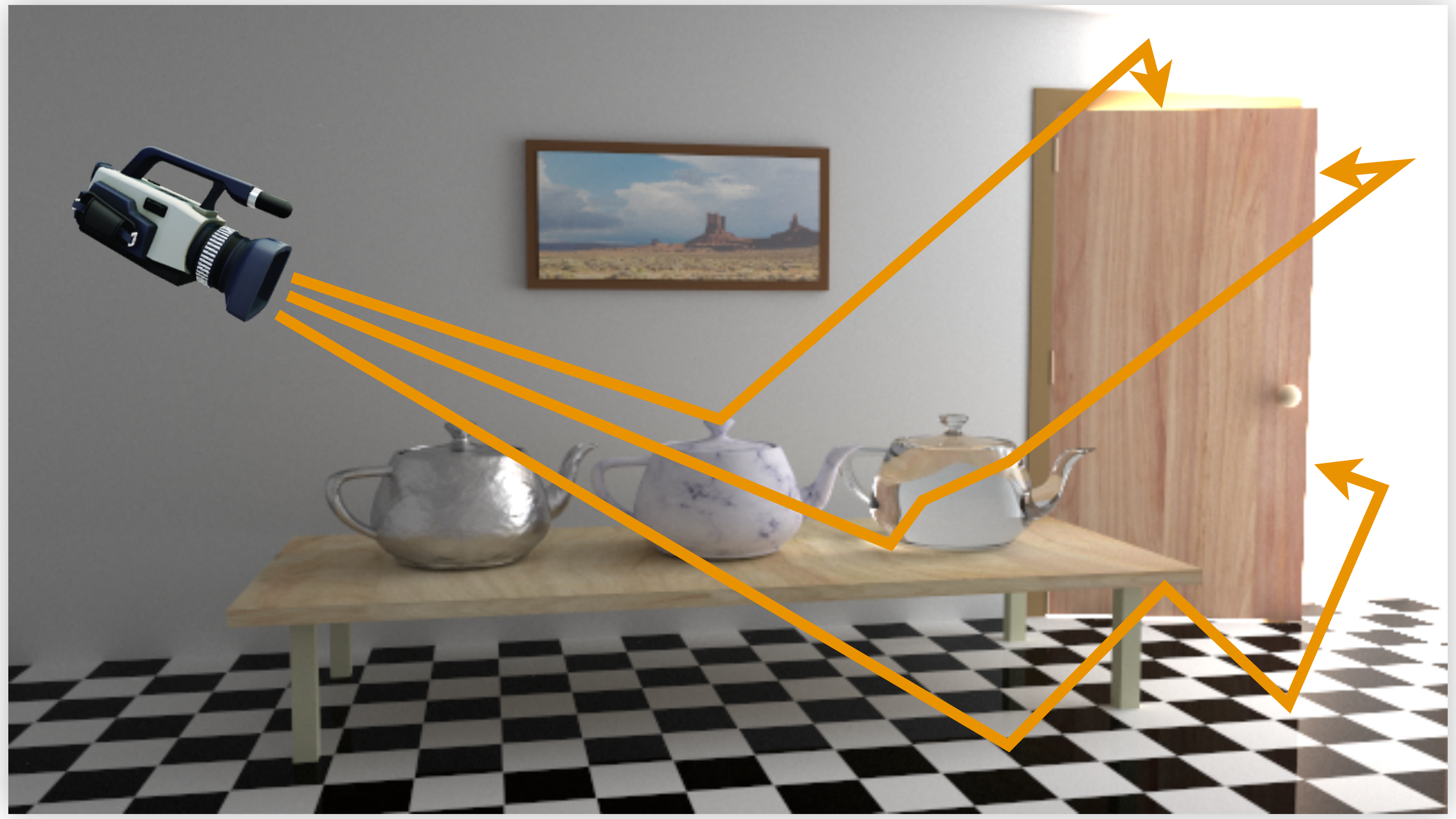


Neural network

Optimize

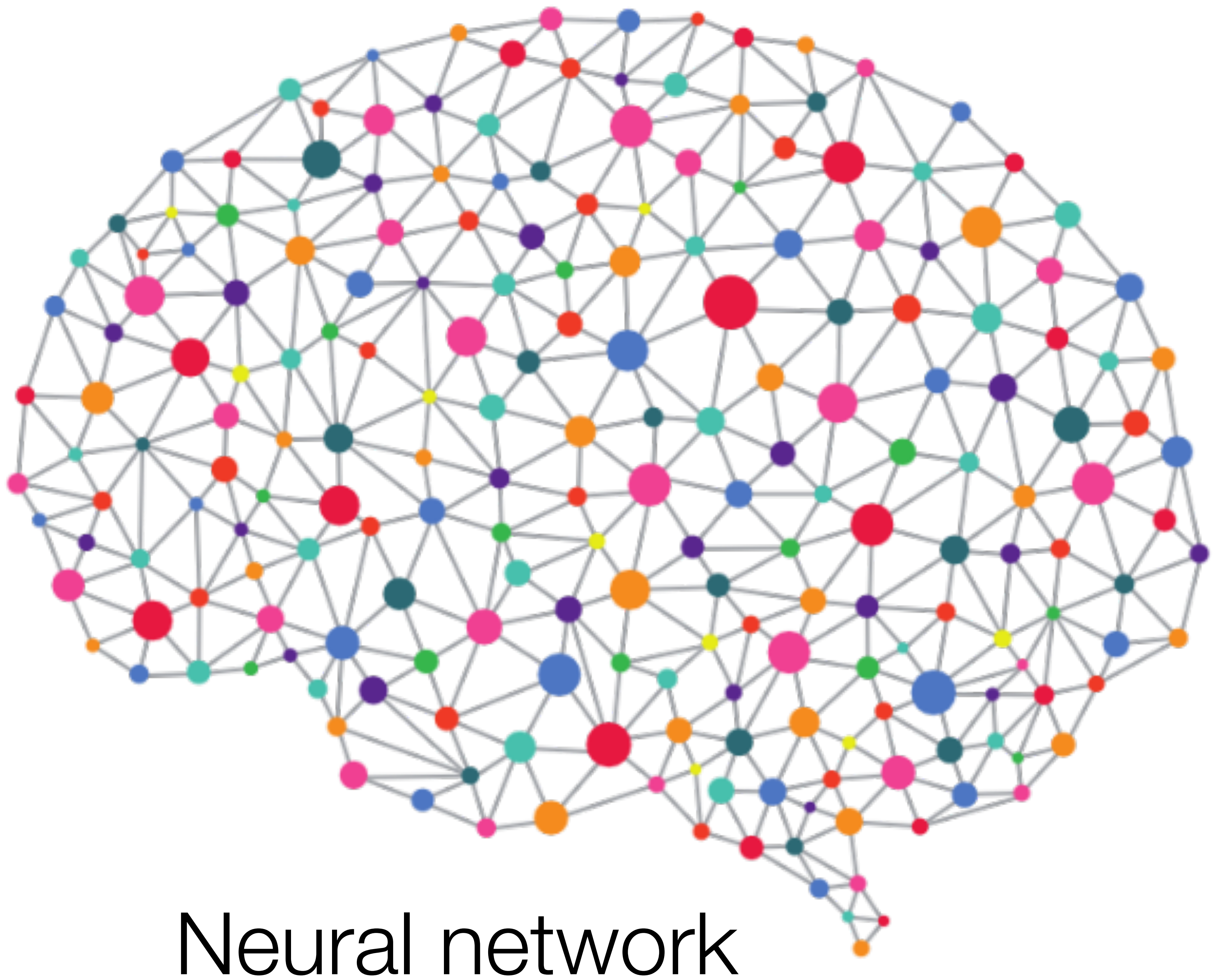
Neural path guiding overview

Sample



Path tracer

Feedback loop



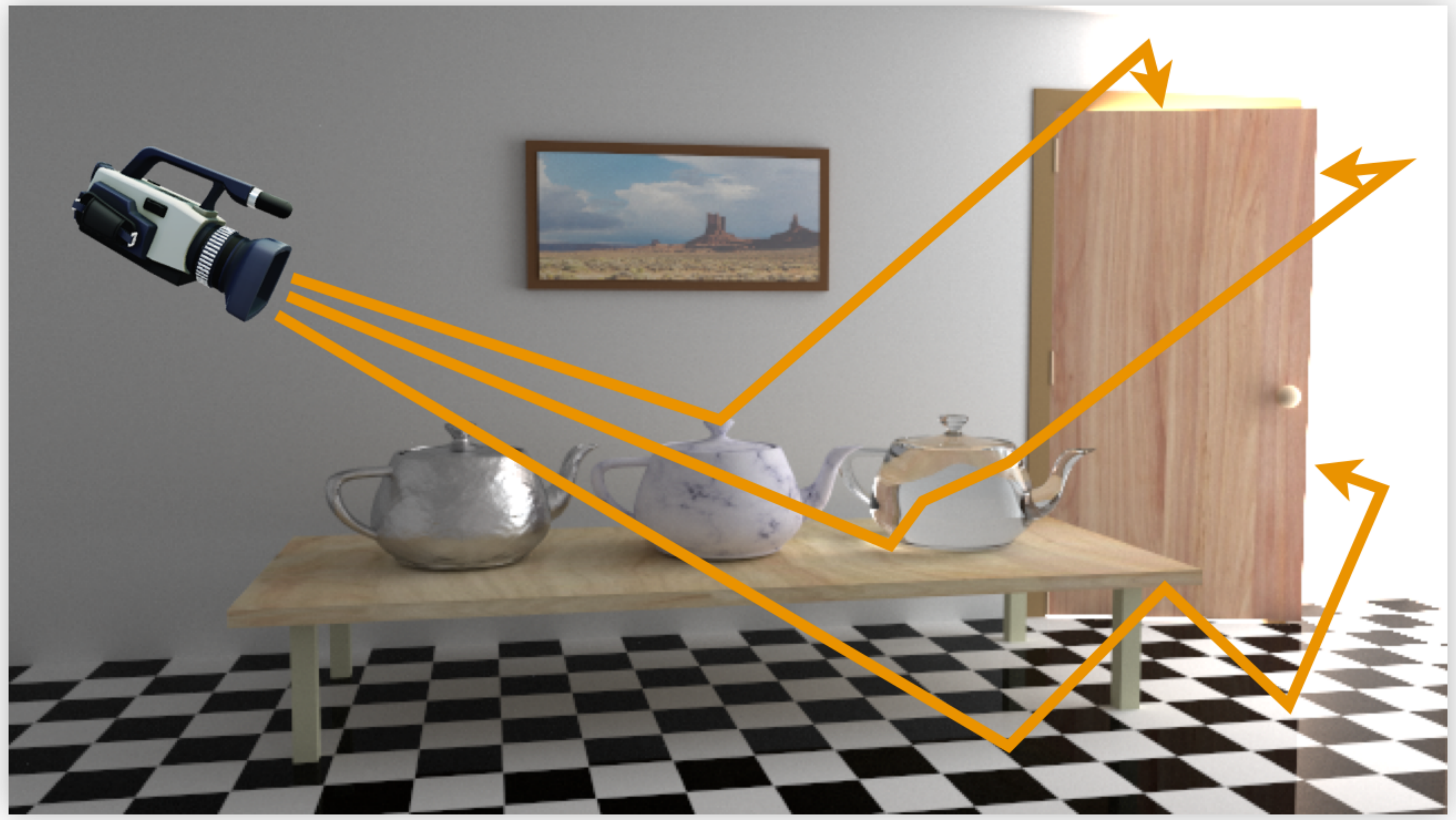
Neural network

Optimize

Neural path guiding overview

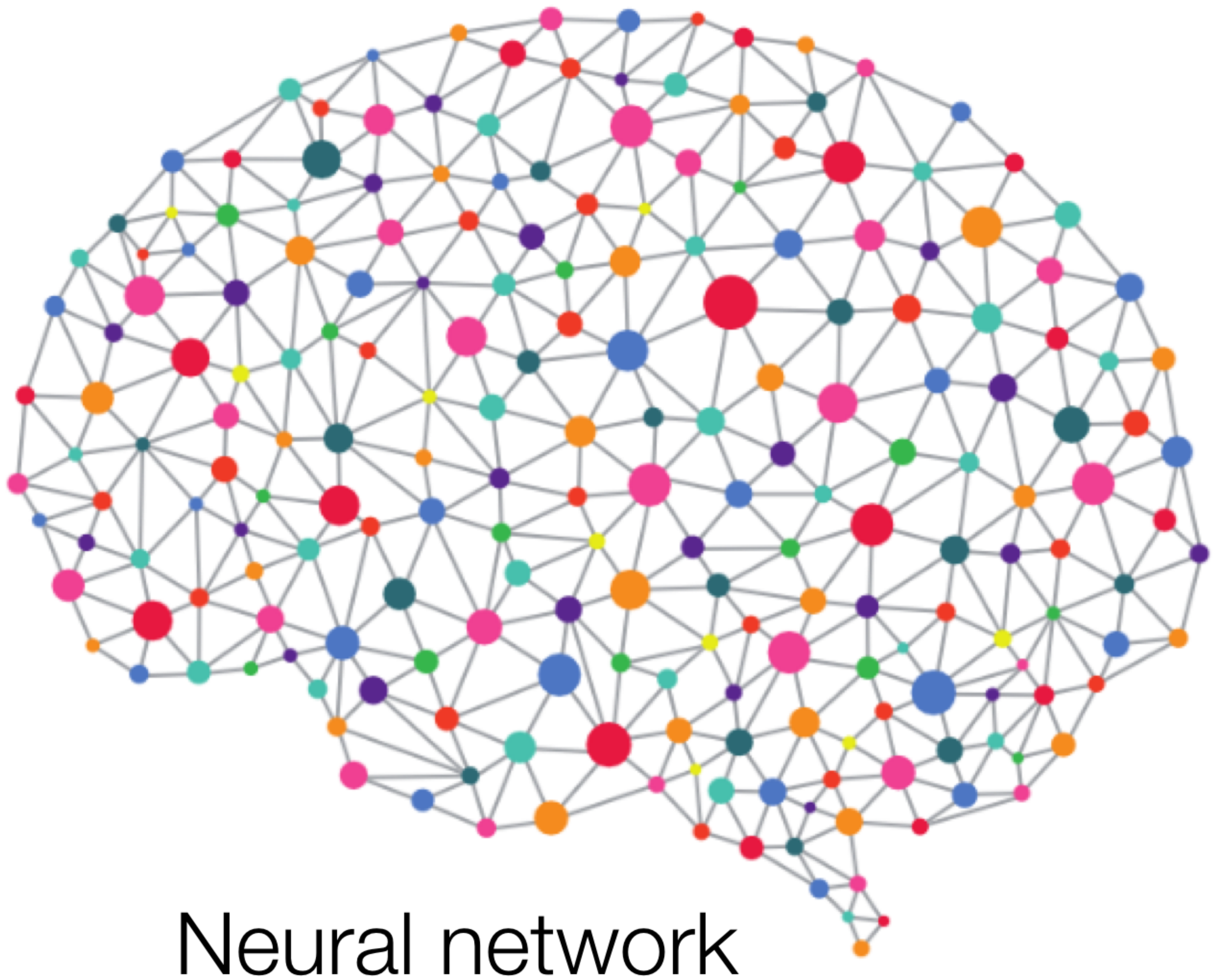
How?

Sample



Path tracer

Feedback loop



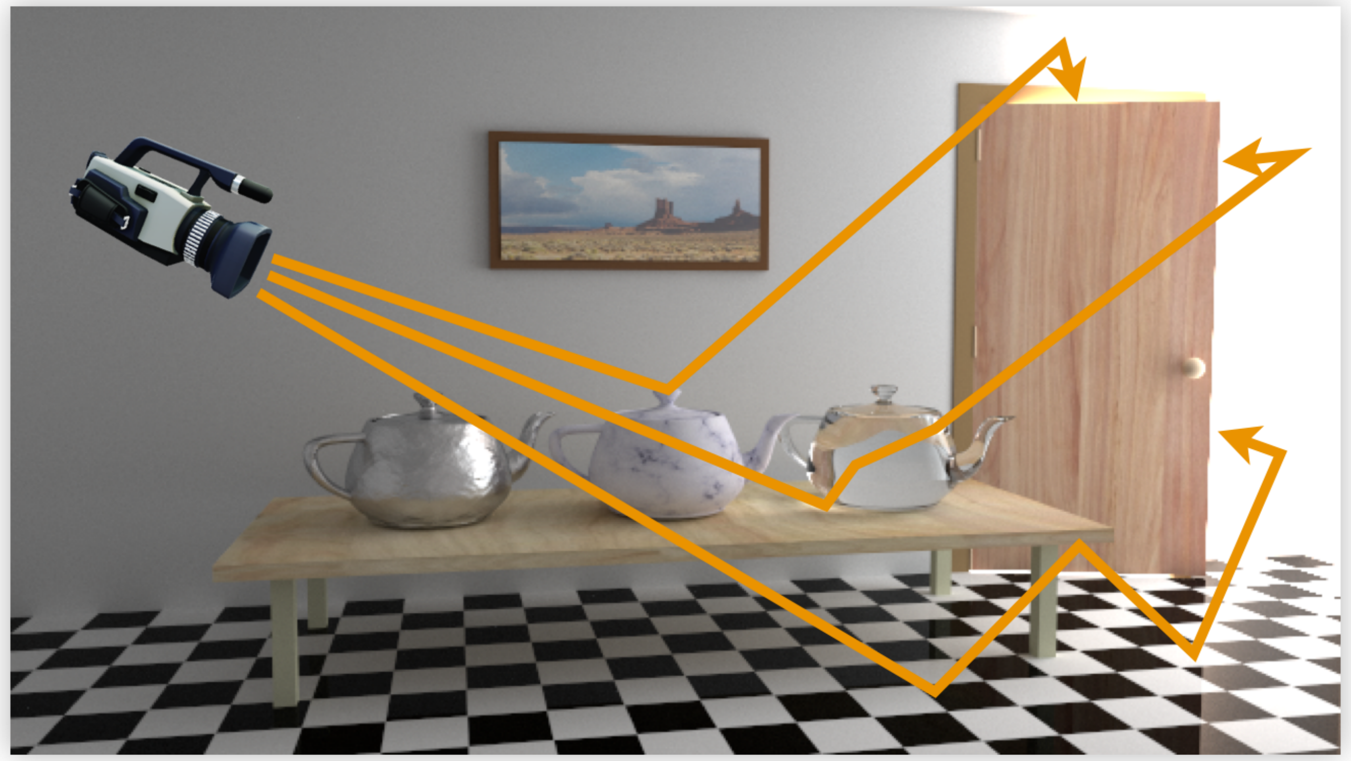
Neural network

Optimize

How?

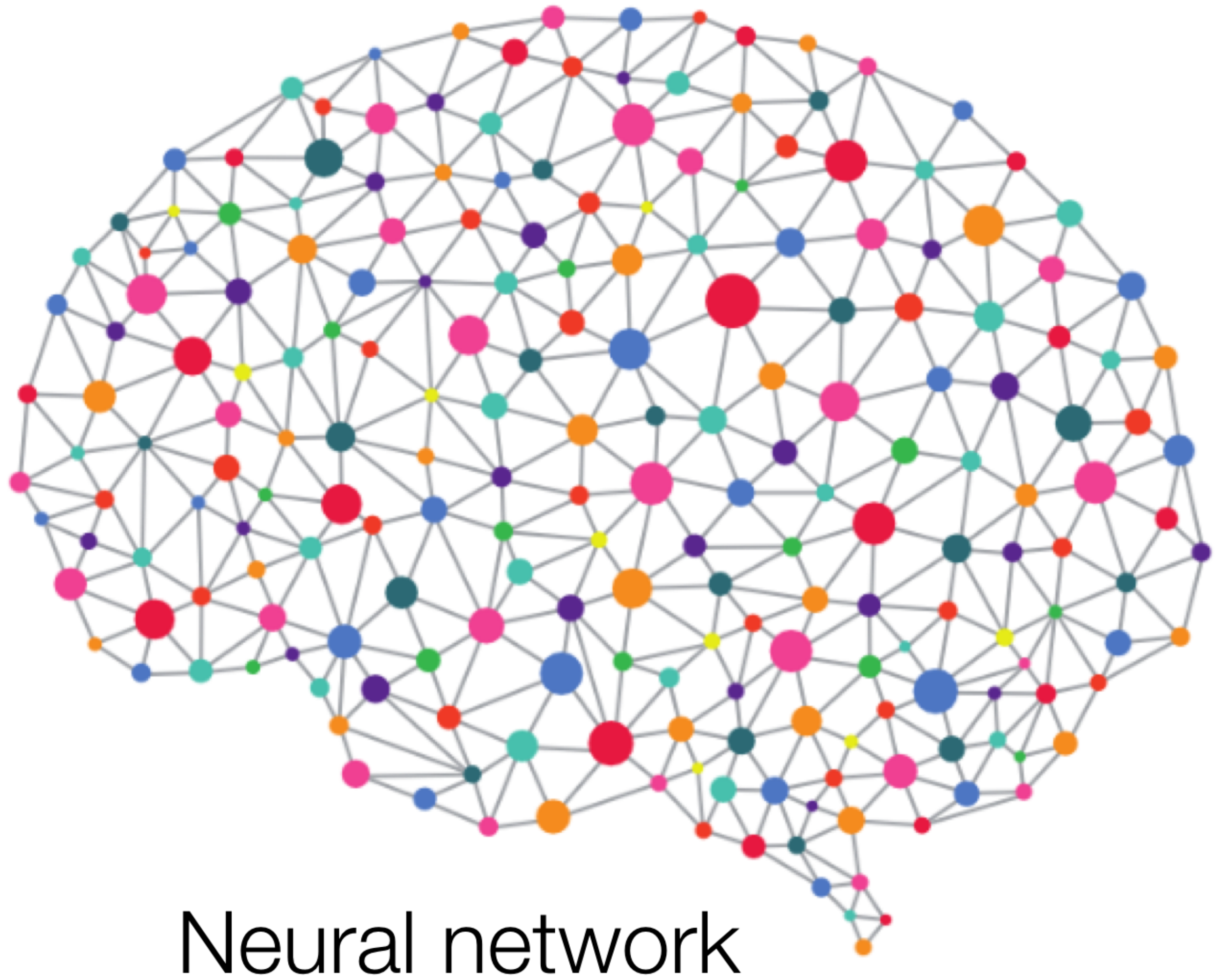
How to draw samples?

Sample



Path tracer

Feedback loop



Neural network

Optimize

Goal: warp random numbers to good distribution with NN

Goal: warp random numbers to good distribution with NN

Random number

$z \rightarrow$

$\rightarrow x$

Goal: warp random numbers to good distribution with NN

Random number

$z \rightarrow$

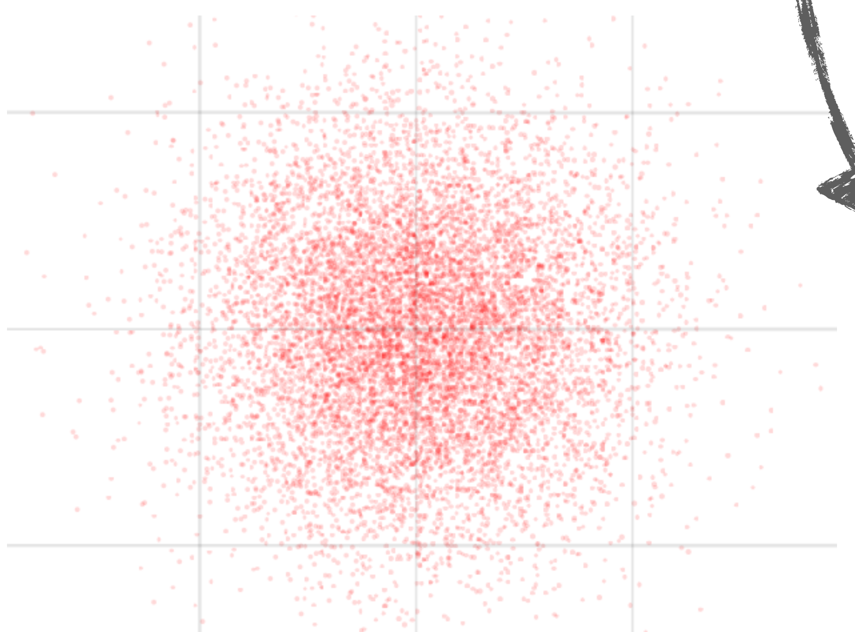
$\rightarrow x$

Sample

Goal: warp random numbers to good distribution with NN

Goal: warp random numbers to good distribution with NN

Random number

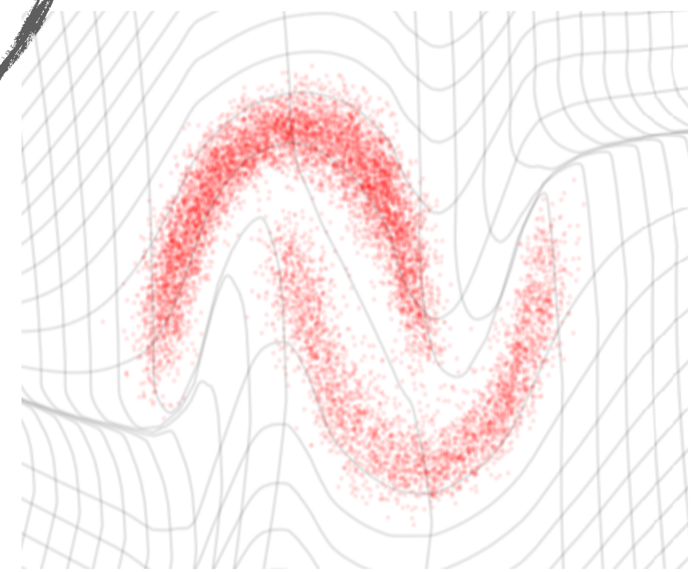


[Dinh et al. 2016]

$z \longrightarrow$

$\longrightarrow x$

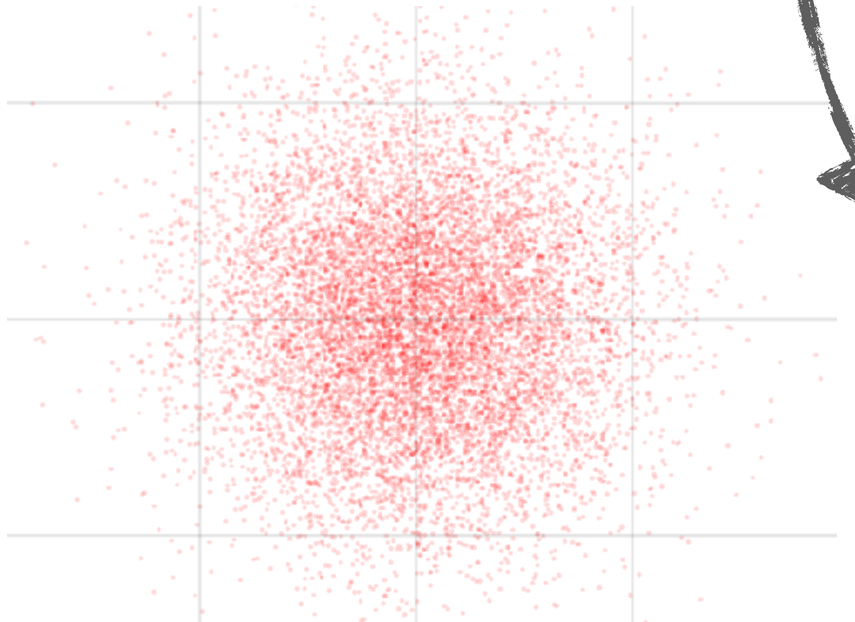
Sample



[Dinh et al. 2016]

Goal: warp random numbers to good distribution with NN

Random number

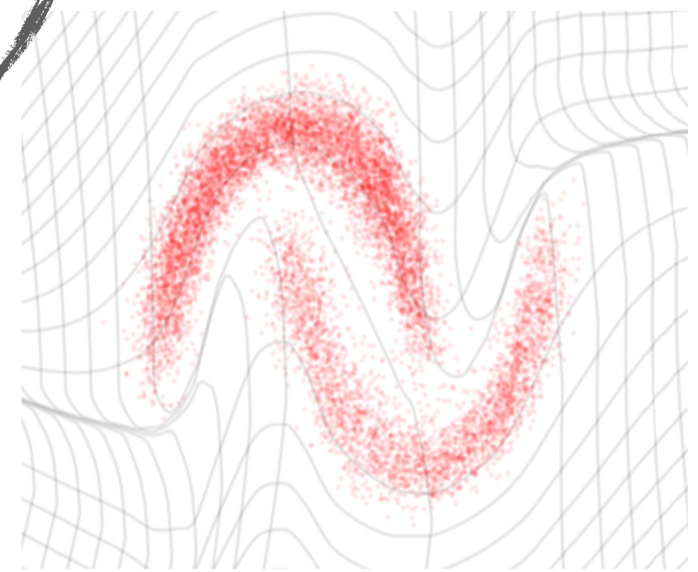


[Dinh et al. 2016]

$z \rightarrow$

$\rightarrow x$

Sample



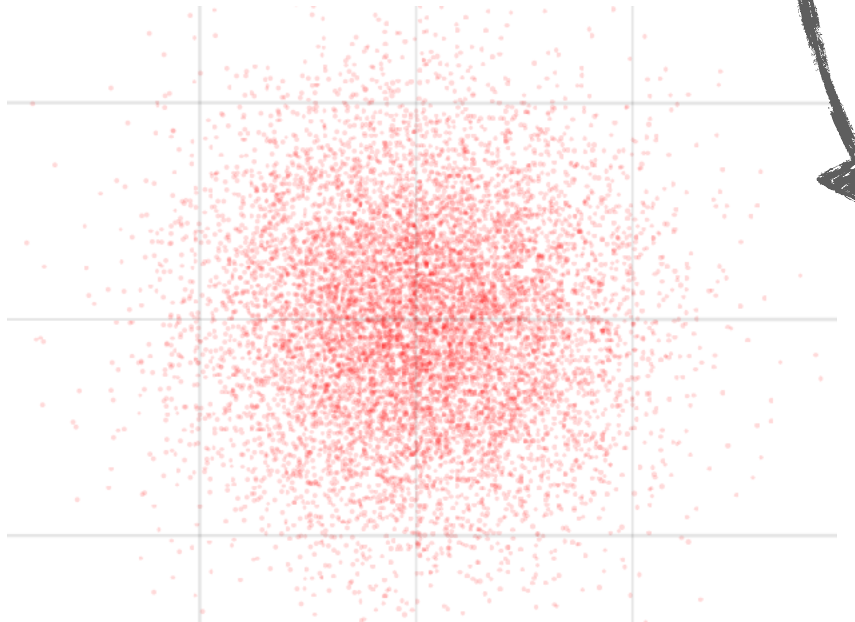
[Dinh et al. 2016]

Monte Carlo estimator

$$F \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}$$

Goal: warp random numbers to good distribution with NN

Random number

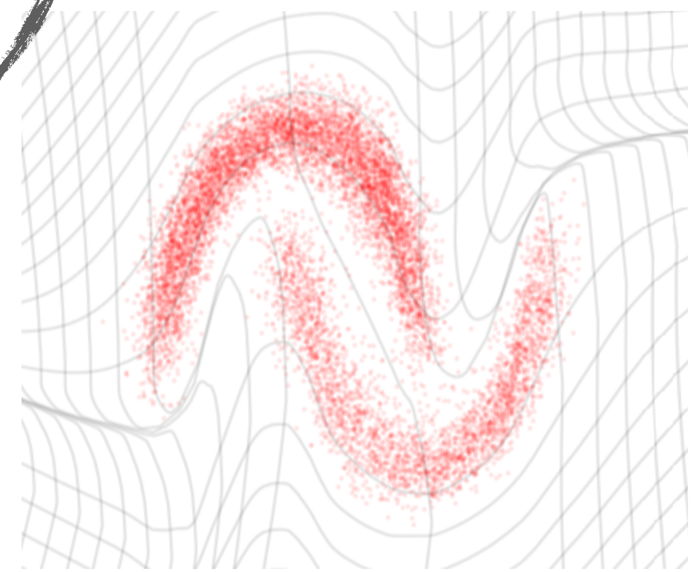


[Dinh et al. 2016]

$z \rightarrow$

$\rightarrow x$

Sample



[Dinh et al. 2016]

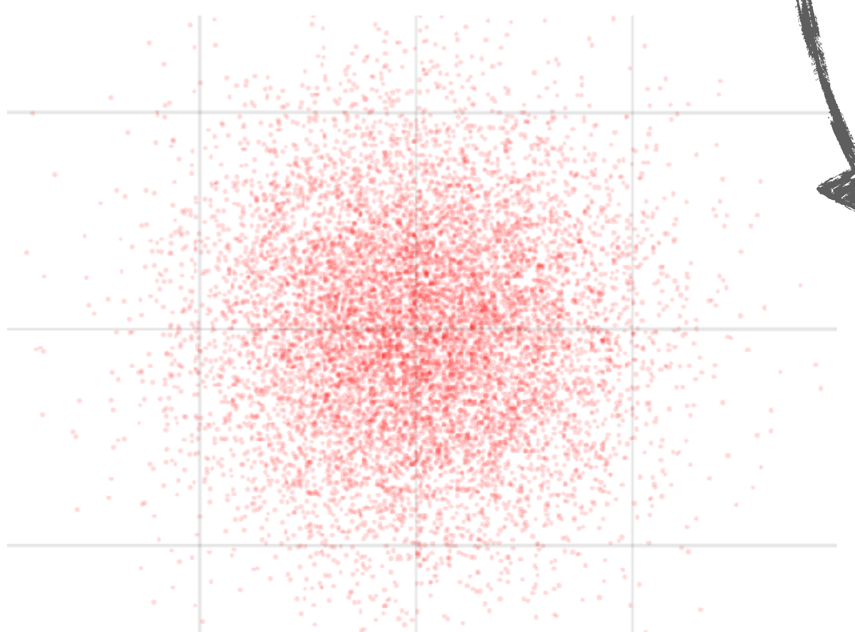
Monte Carlo estimator

$$F \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}$$

Need p in closed form!

Goal: warp random numbers to good distribution with NN

Random number

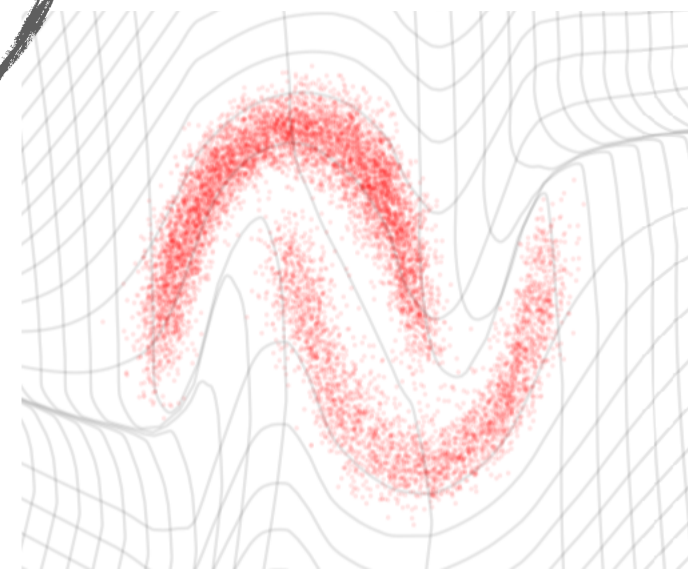


[Dinh et al. 2016]

$z \longrightarrow$

$\longrightarrow x$

Sample



[Dinh et al. 2016]

Monte Carlo estimator

$$F \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}$$

Need p in closed form!

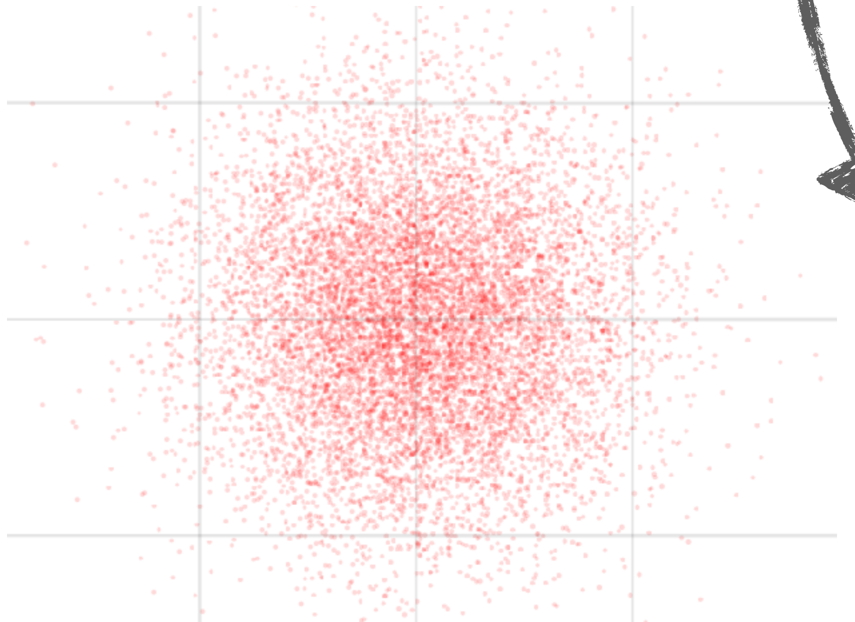
Addressed by "normalizing flows"

Parameterizing a bijection allows using the change-of-variable formula

Random number

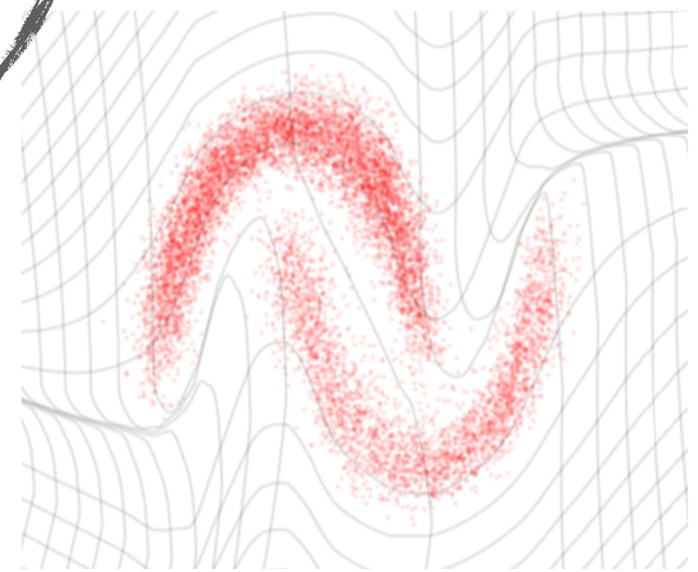
Parametric bijective function

Sample



[Dinh et al. 2016]

$$z \longrightarrow m(z;\phi) \longrightarrow x$$



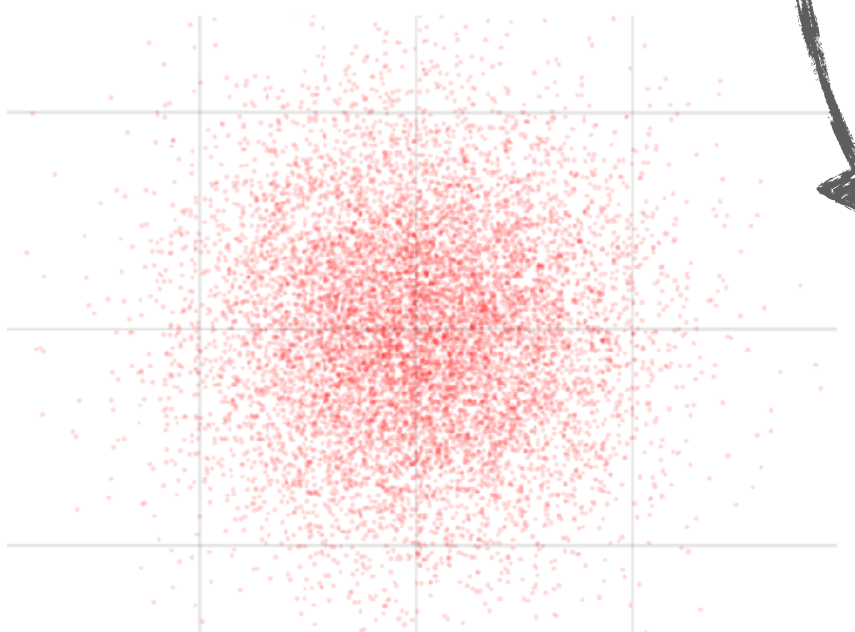
[Dinh et al. 2016]

Parameterizing a bijection allows using the change-of-variable formula

Random number

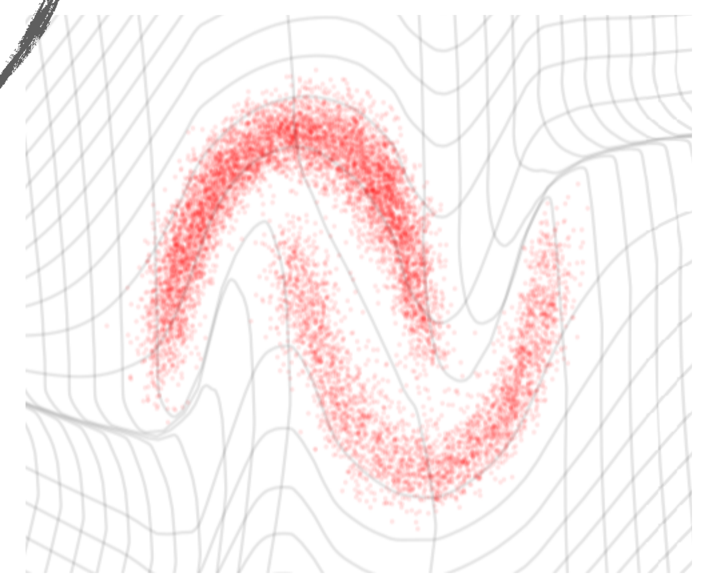
Parametric bijective function

Sample



[Dinh et al. 2016]

$$z \longrightarrow m(z;\phi) \longrightarrow x$$



[Dinh et al. 2016]

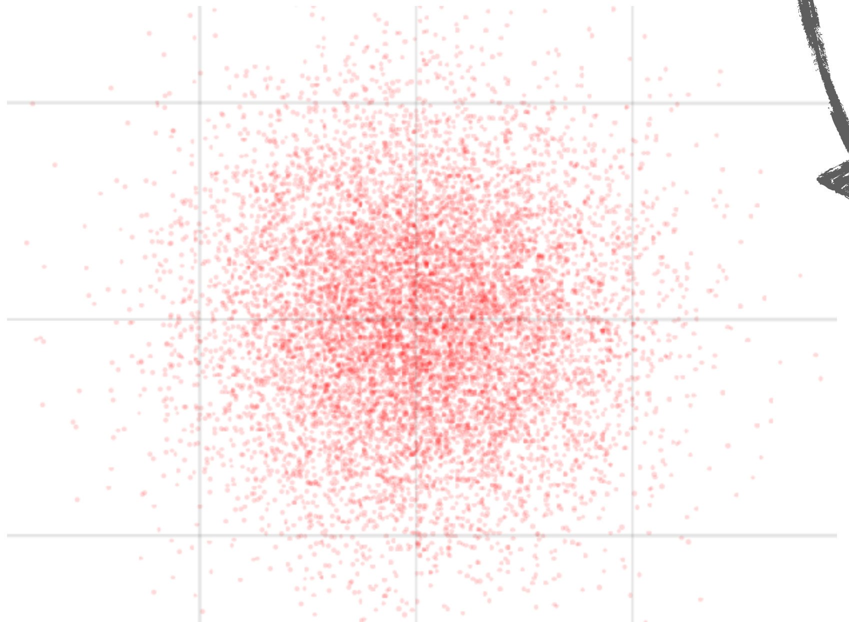
$$p(x) = p(z) \cdot \left| det \left(\frac{\partial m(z)}{\partial z^T} \right) \right|^{-1}$$

Parameterizing a bijection allows using the change-of-variable formula

Random number

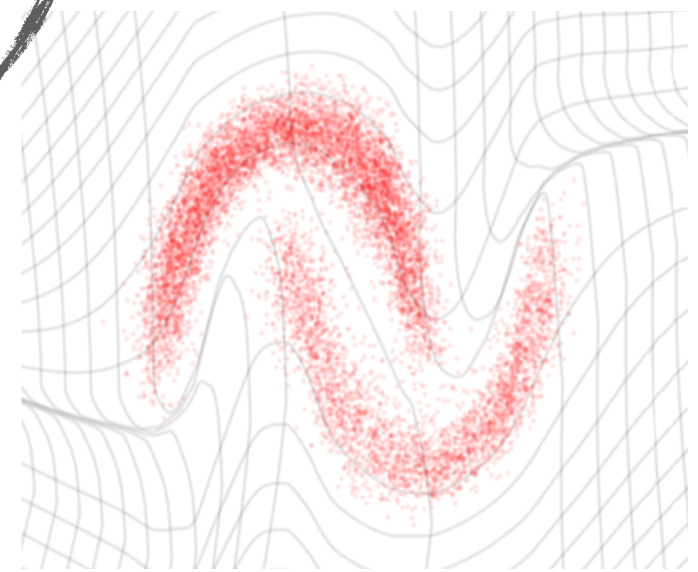
Parametric bijective function

Sample



[Dinh et al. 2016]

$$z \longrightarrow m(z;\phi) \longrightarrow x$$



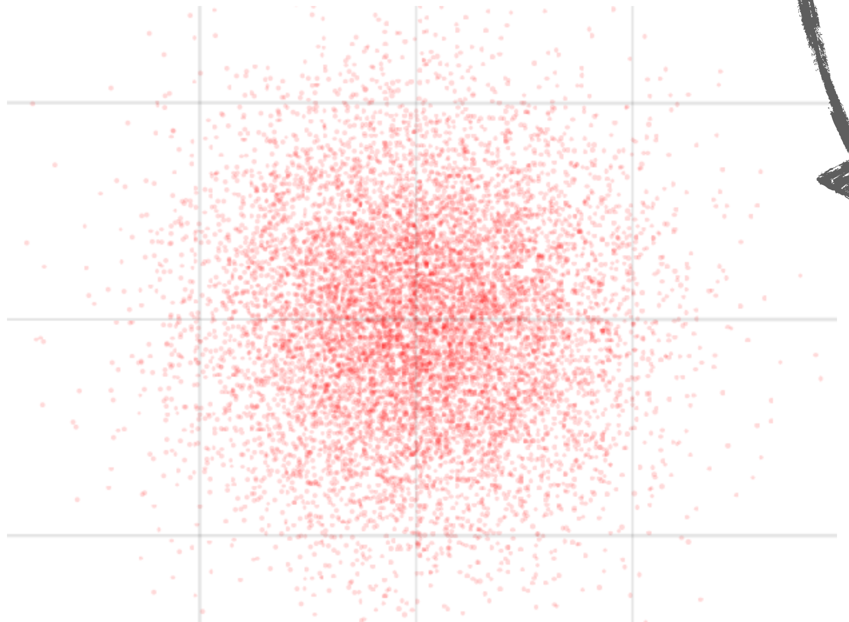
[Dinh et al. 2016]

$$p(x) = p(z) \cdot \left| det\left(\frac{\partial m(z)}{\partial z^T}\right) \right|^{-1}$$

Our choice, e.g. Gaussian

Parameterizing a bijection allows using the change-of-variable formula

Random number



[Dinh et al. 2016]

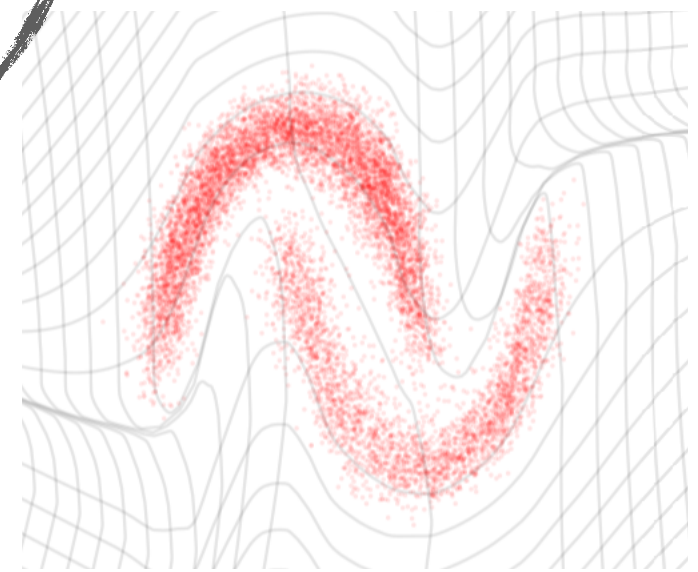
$z \longrightarrow$

Parametric bijective function

$$m(z;\phi)$$

$\longrightarrow x$

Sample



[Dinh et al. 2016]

$$p(x) = p(z) \cdot \left| det\left(\frac{\partial m(z)}{\partial z^T}\right)\right|^{-1}$$

Our choice, e.g. Gaussian

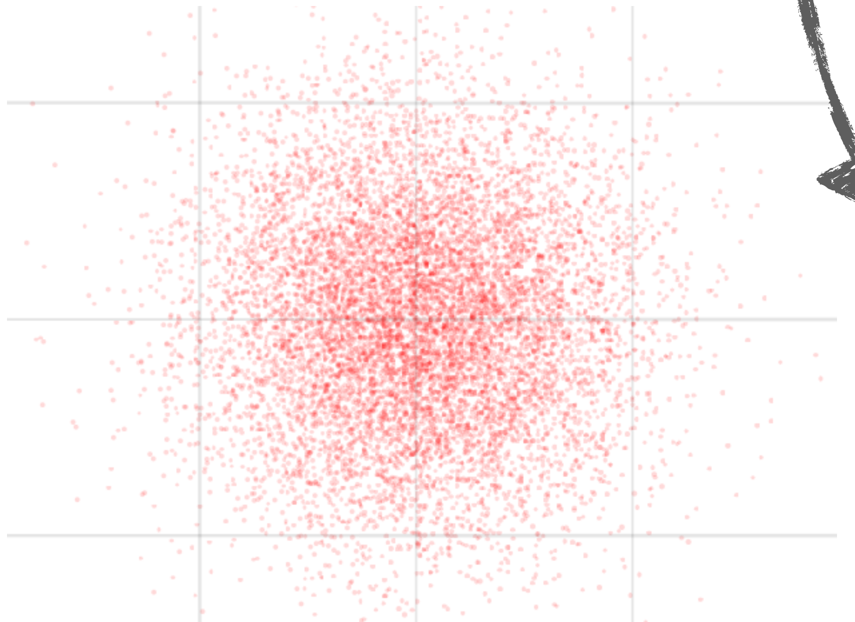
Squishing/stretching by m

Parameterizing a bijection allows using the change-of-variable formula

Random number

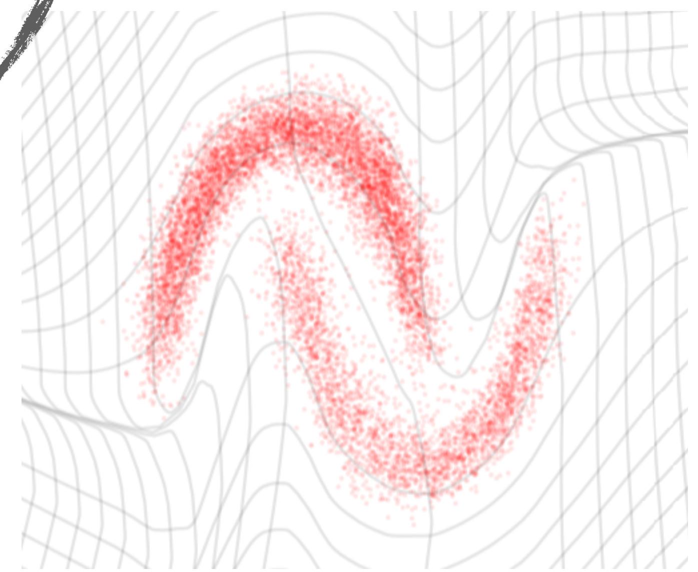
Parametric bijective function

Sample



[Dinh et al. 2016]

$$z \longrightarrow m(z;\phi) \longrightarrow x$$



[Dinh et al. 2016]

$$p(x) = p(z) \cdot \left| det\left(\frac{\partial m(z)}{\partial z^T}\right)\right|^{-1}$$

Our choice, e.g. Gaussian

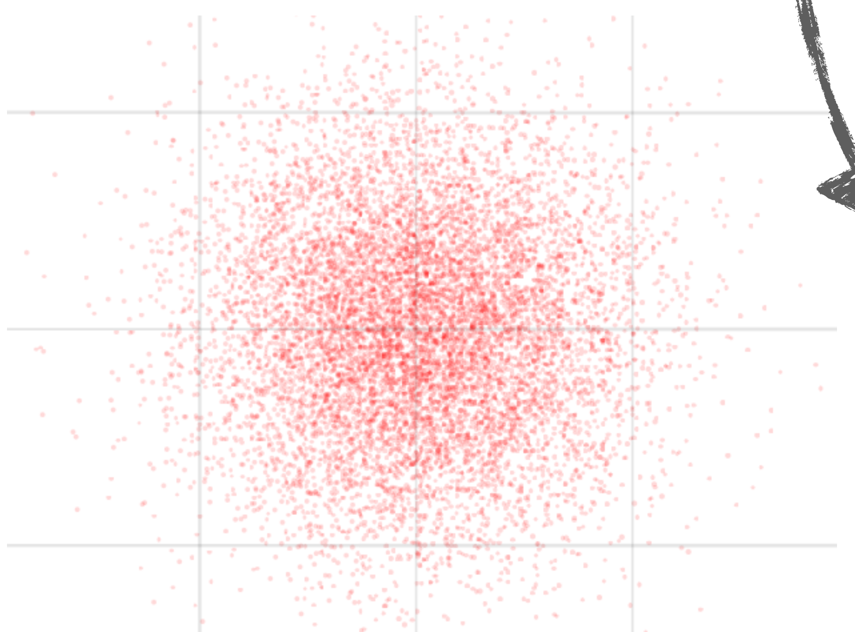
Squishing/stretching by m

A chain of simple bijections can model complicated functions

Parametric bijective functions
(normalizing flow)

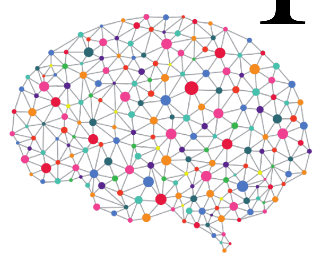
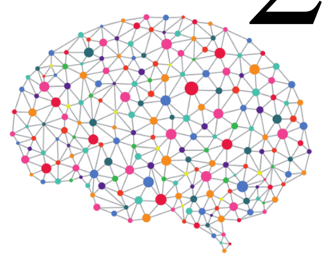
Random number

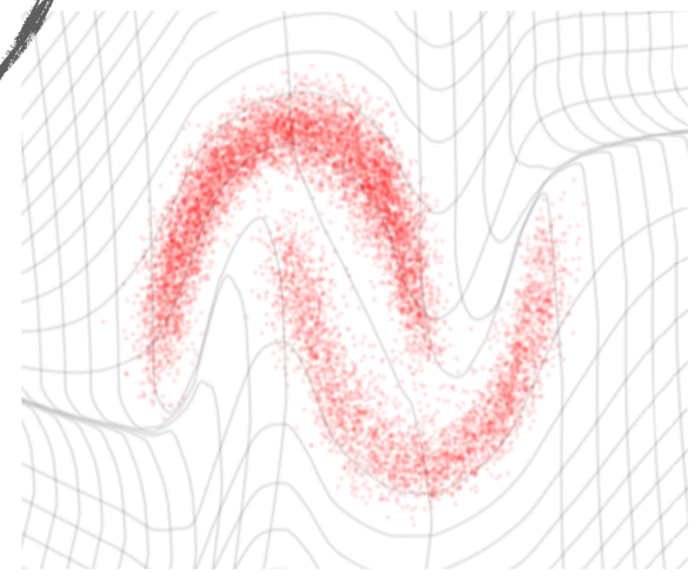
Sample



[Dinh et al. 2016]

$$z \longrightarrow m_1 \circ m_2 \circ \ldots \circ m_L \longrightarrow x$$

[Dinh et al. 2016]

$$p(x) = p(z) \cdot \prod_{i=1}^{L} \left| det\left(\frac{\partial m_i(z)}{\partial z^T}\right)\right|^{-1}$$

Our choice, e.g. Gaussian

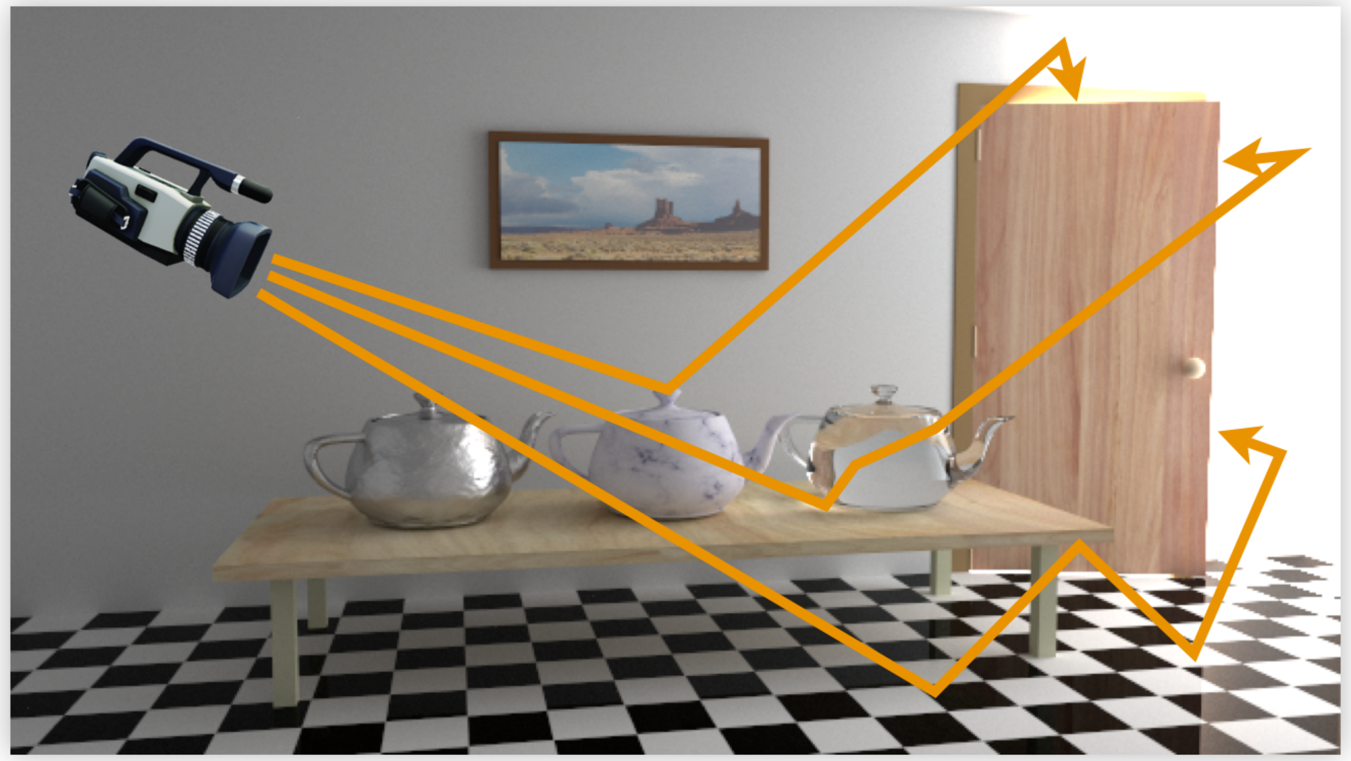
Squishing/stretching by m

How to optimize?

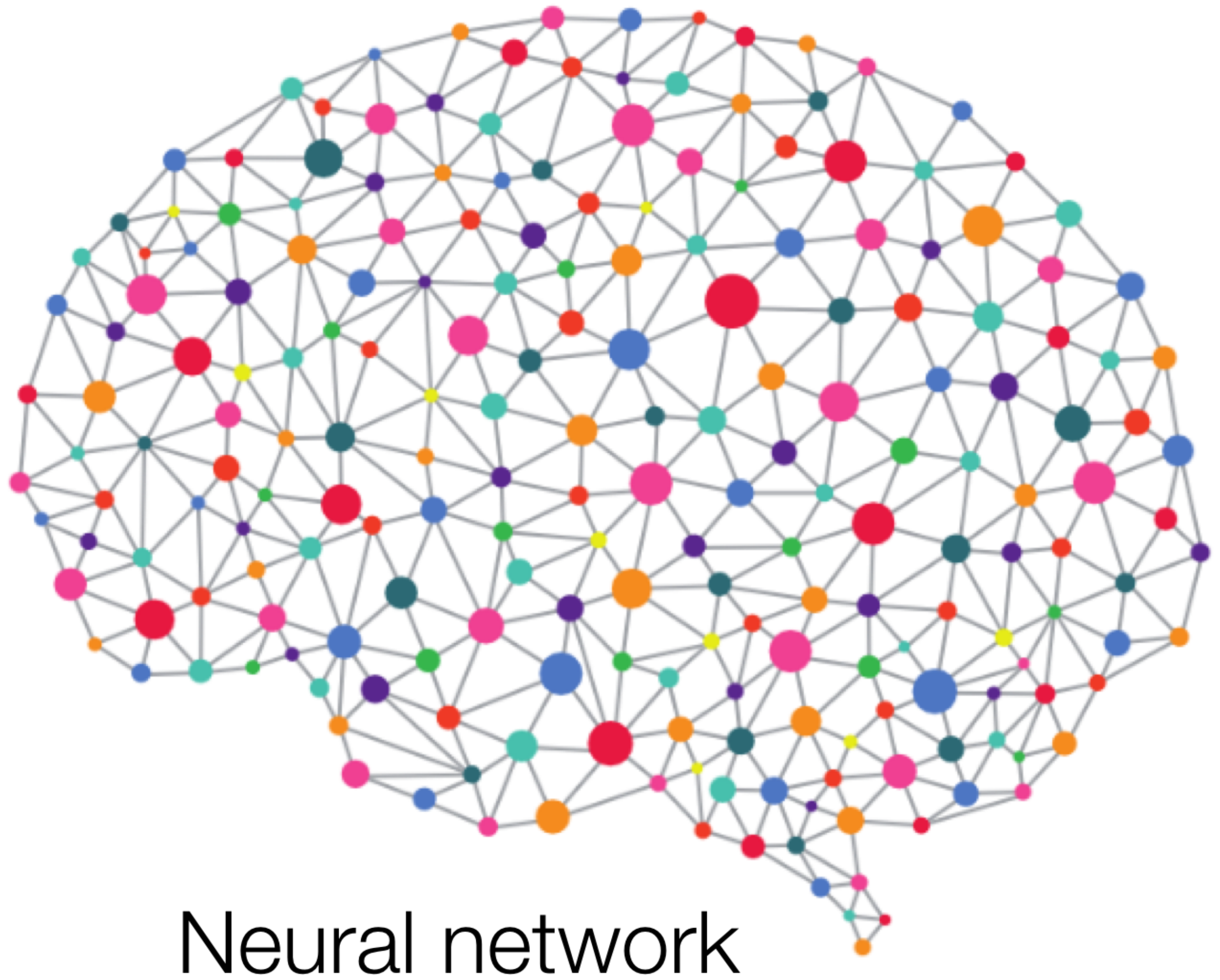
Sample

Feedback loop

Optimize



Path tracer

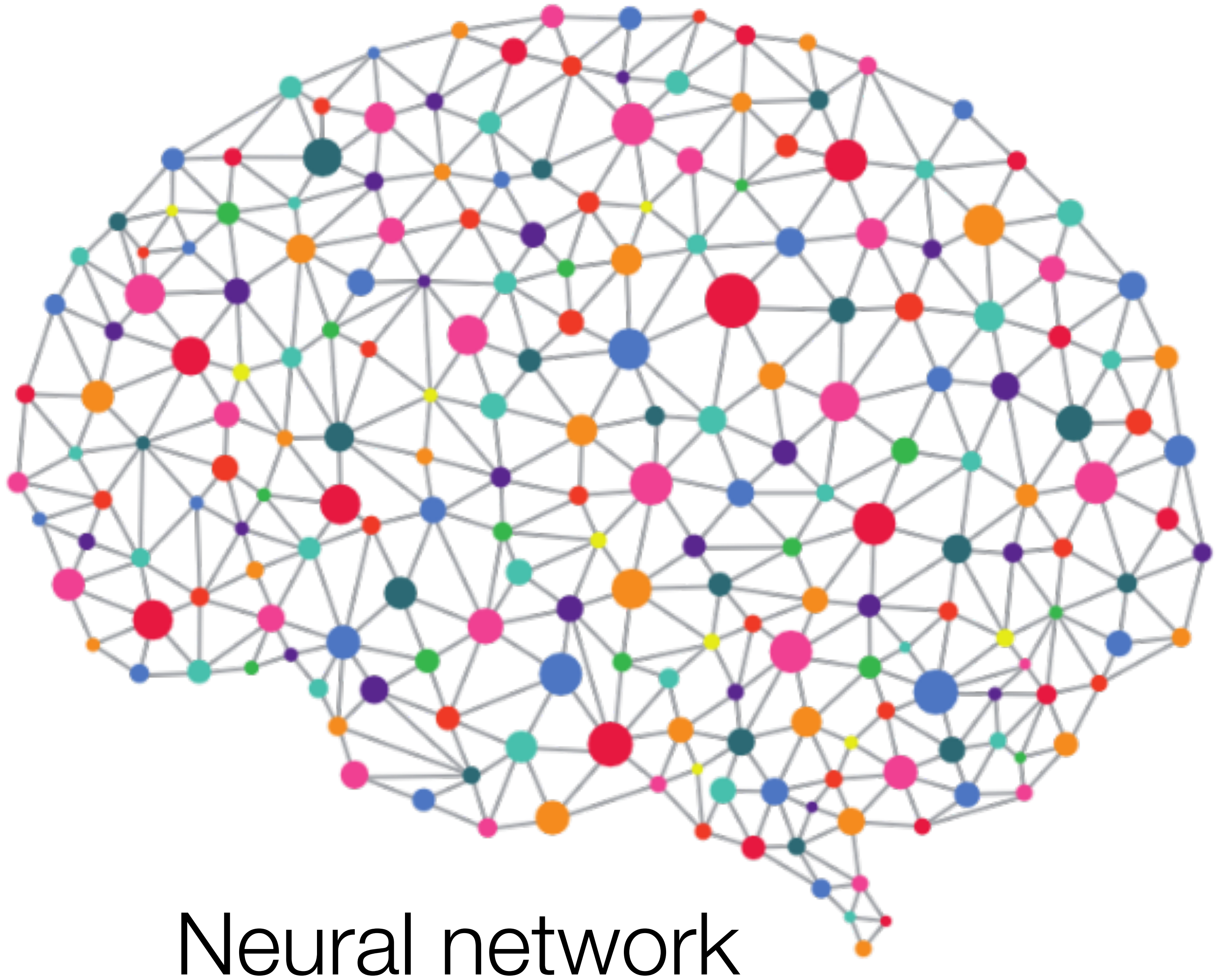


Neural network

Training with data from the correct distribution is simple

Training data

Optimize



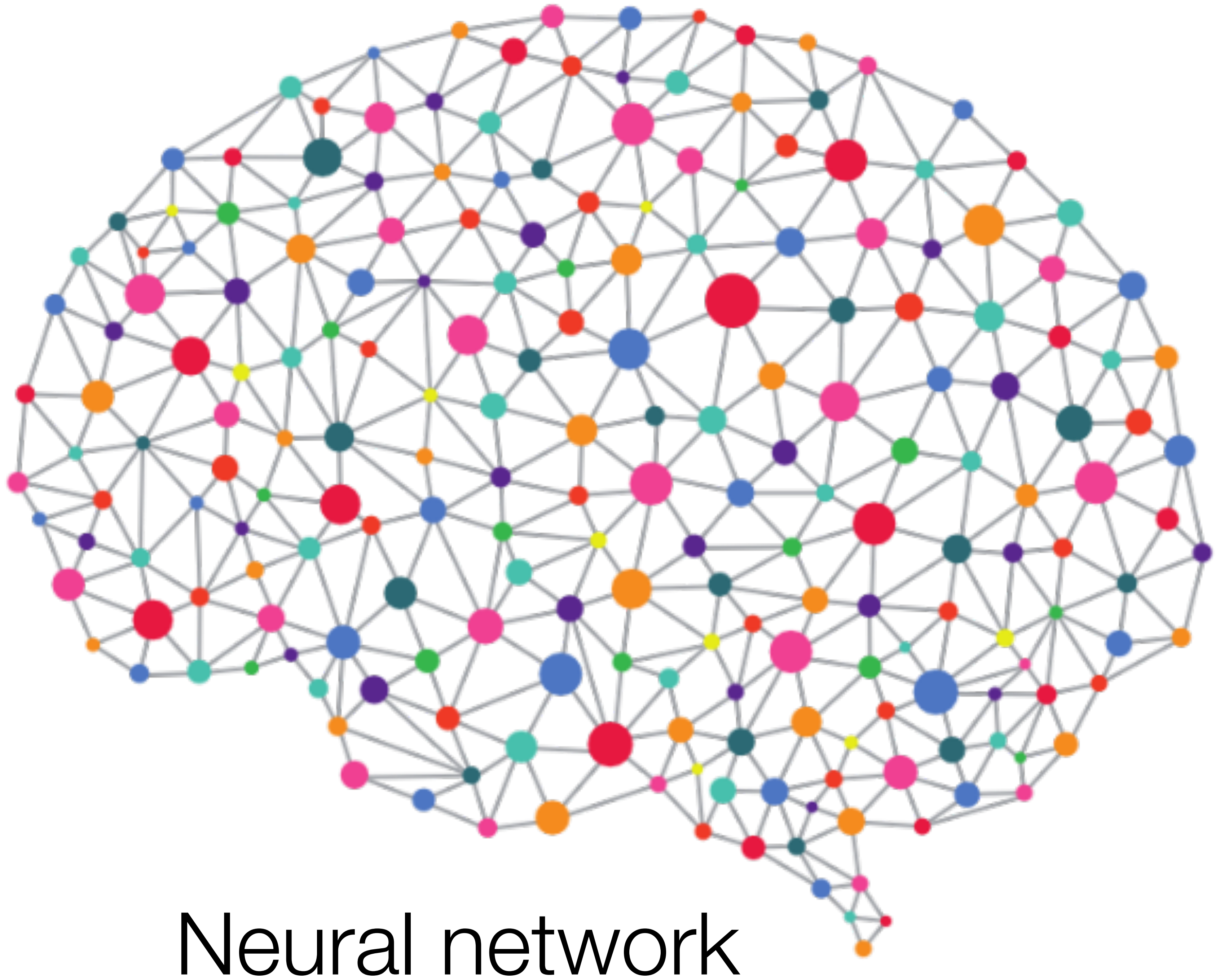
Neural network

Training with data from the correct distribution is simple

Desired distribution

Training data

Optimize



Neural network

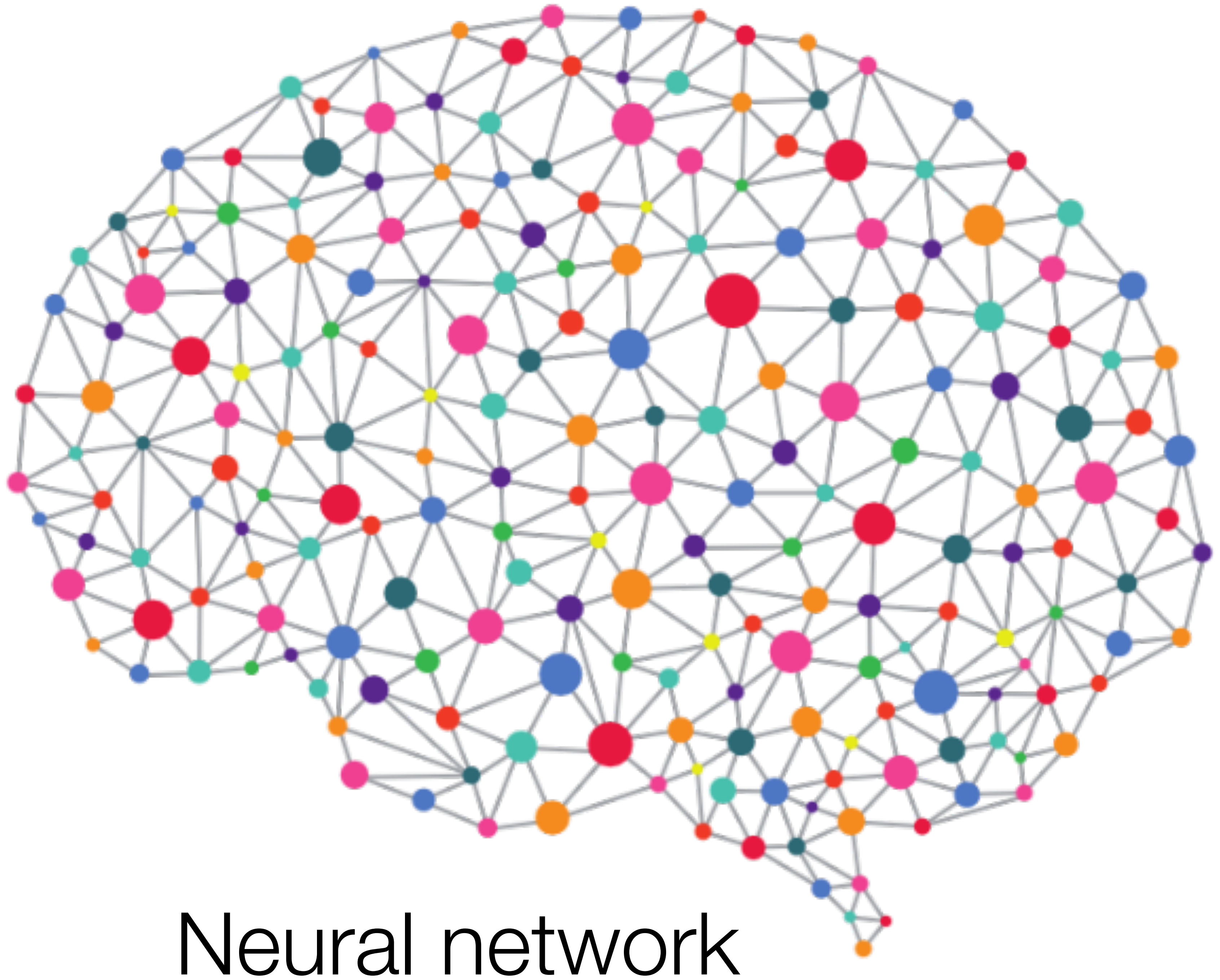
Training with data from the correct distribution is simple

Training data

Desired distribution

Optimize

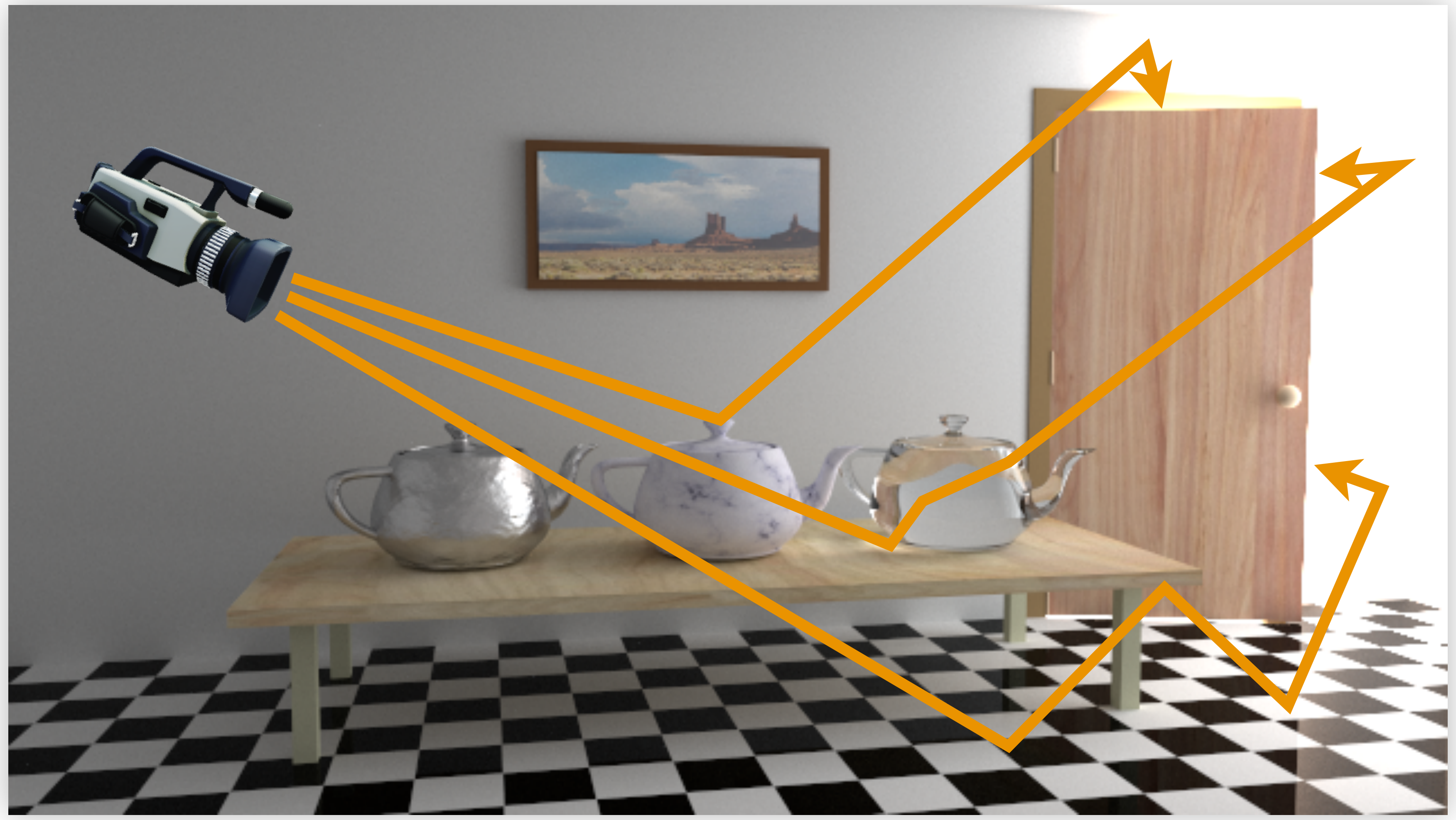
$$\nabla_\theta \log p(x;\theta)$$



Neural network

Training from Monte Carlo samples requires careful weighting

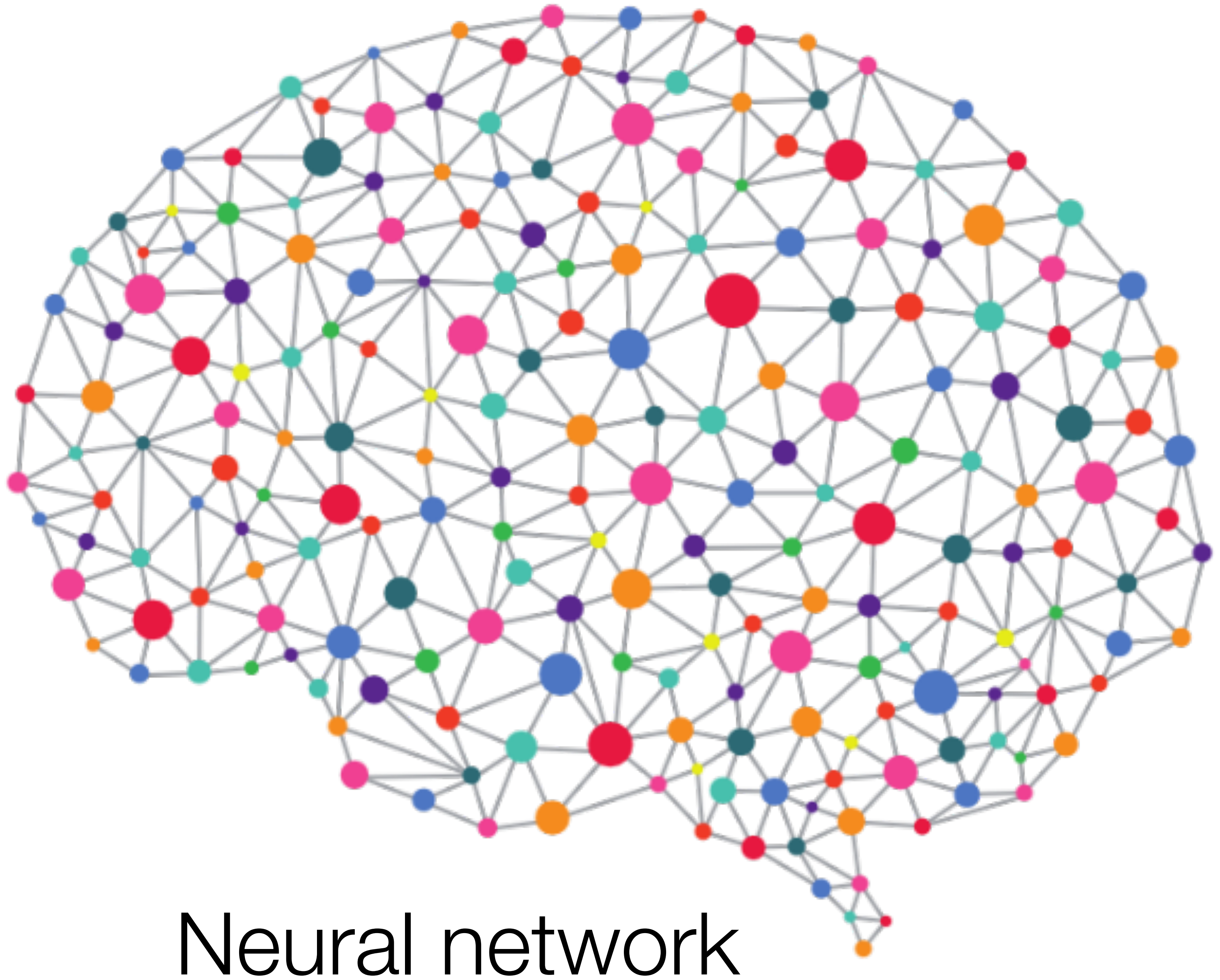
Arbitrarily distributed



Training data

Optimize

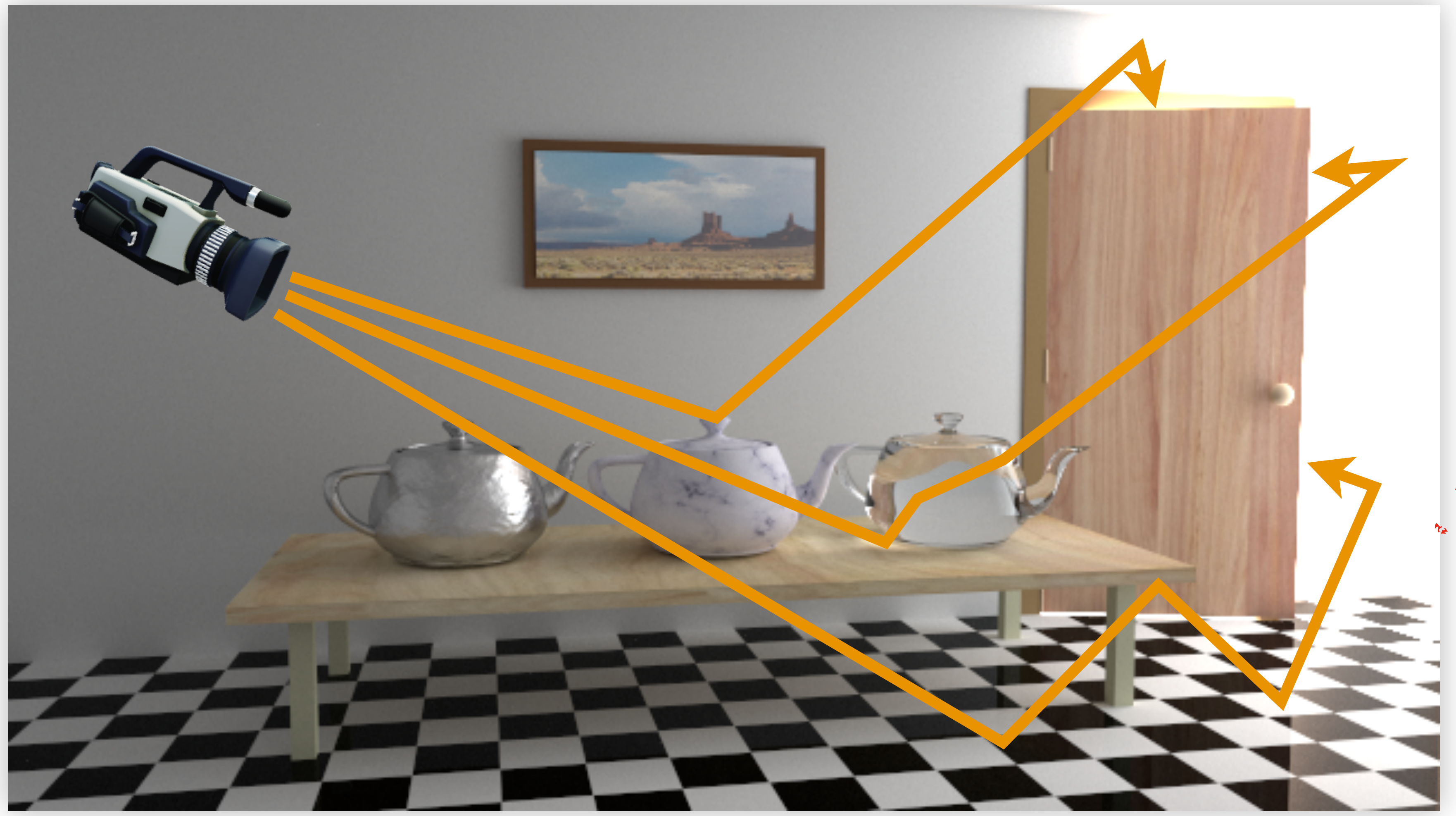
$$\nabla_\theta \log p(x;\theta)$$



Neural network

Training from Monte Carlo samples requires careful weighting

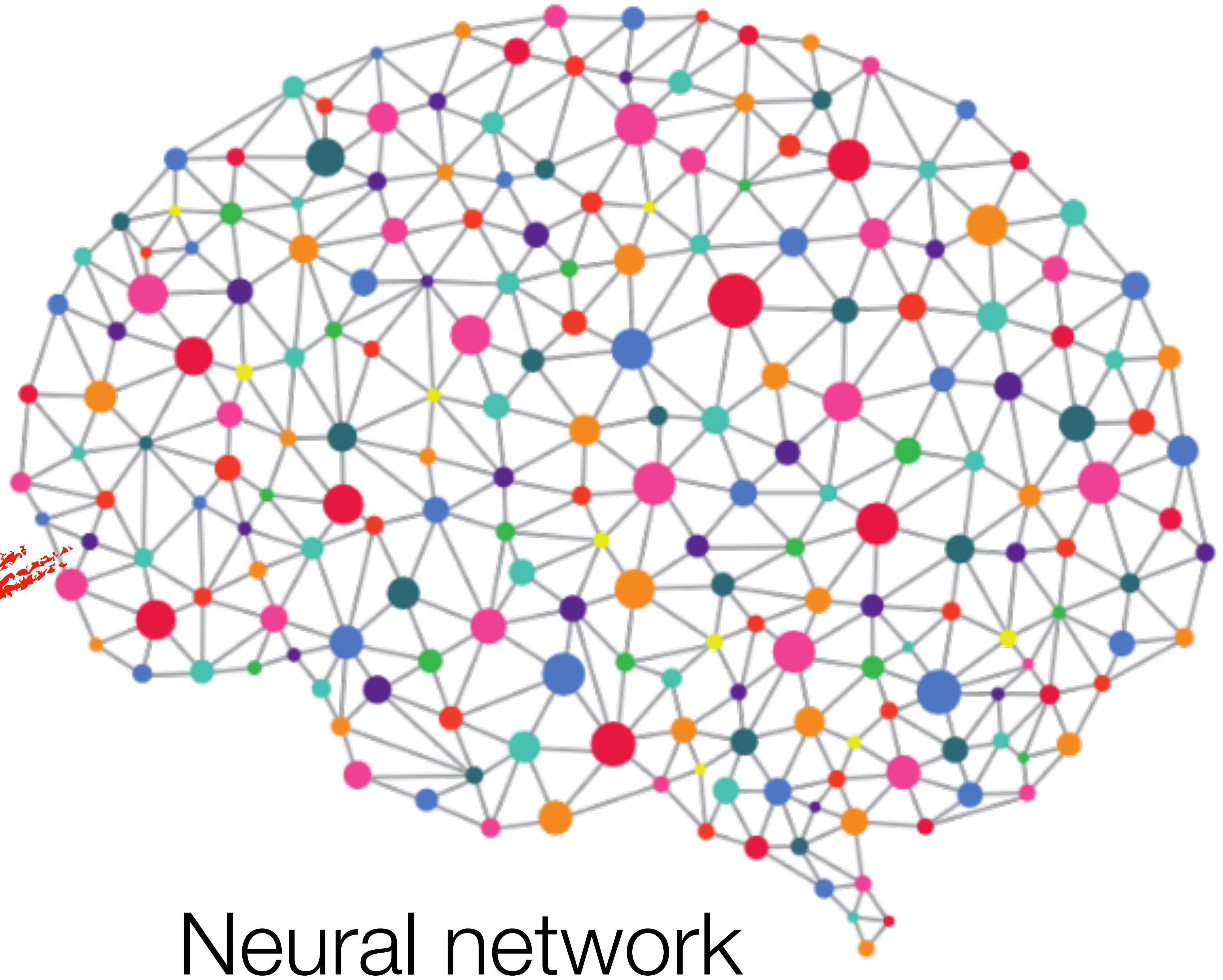
Arbitrarily distributed



Training data

Optimize

$\cancel{\nabla_\theta \log p(x;\theta)}$



Neural network

Training with data from the correct distribution is simple

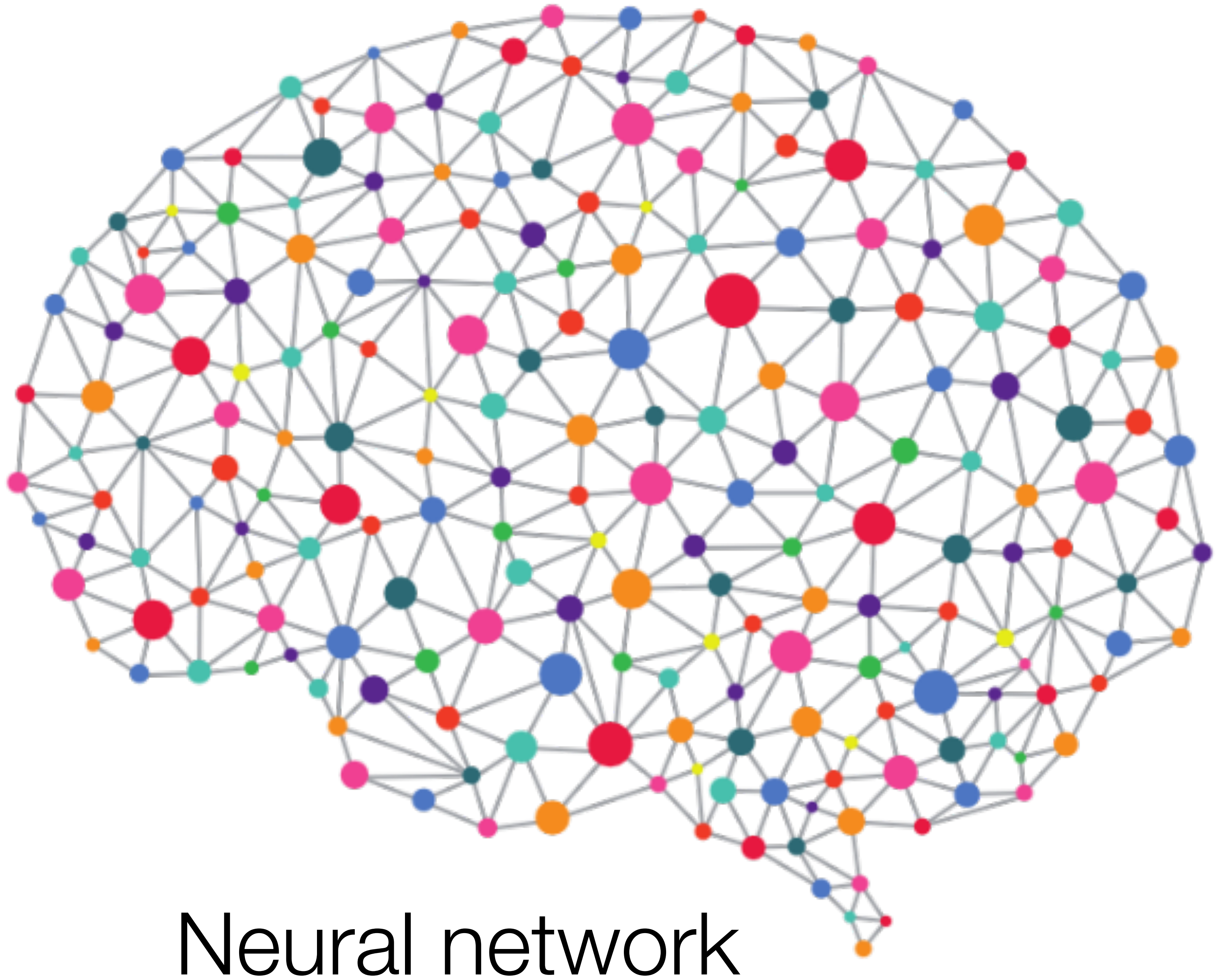
Desired distribution

Training data

Optimize

$$\nabla_\theta \log p(x; \theta)$$

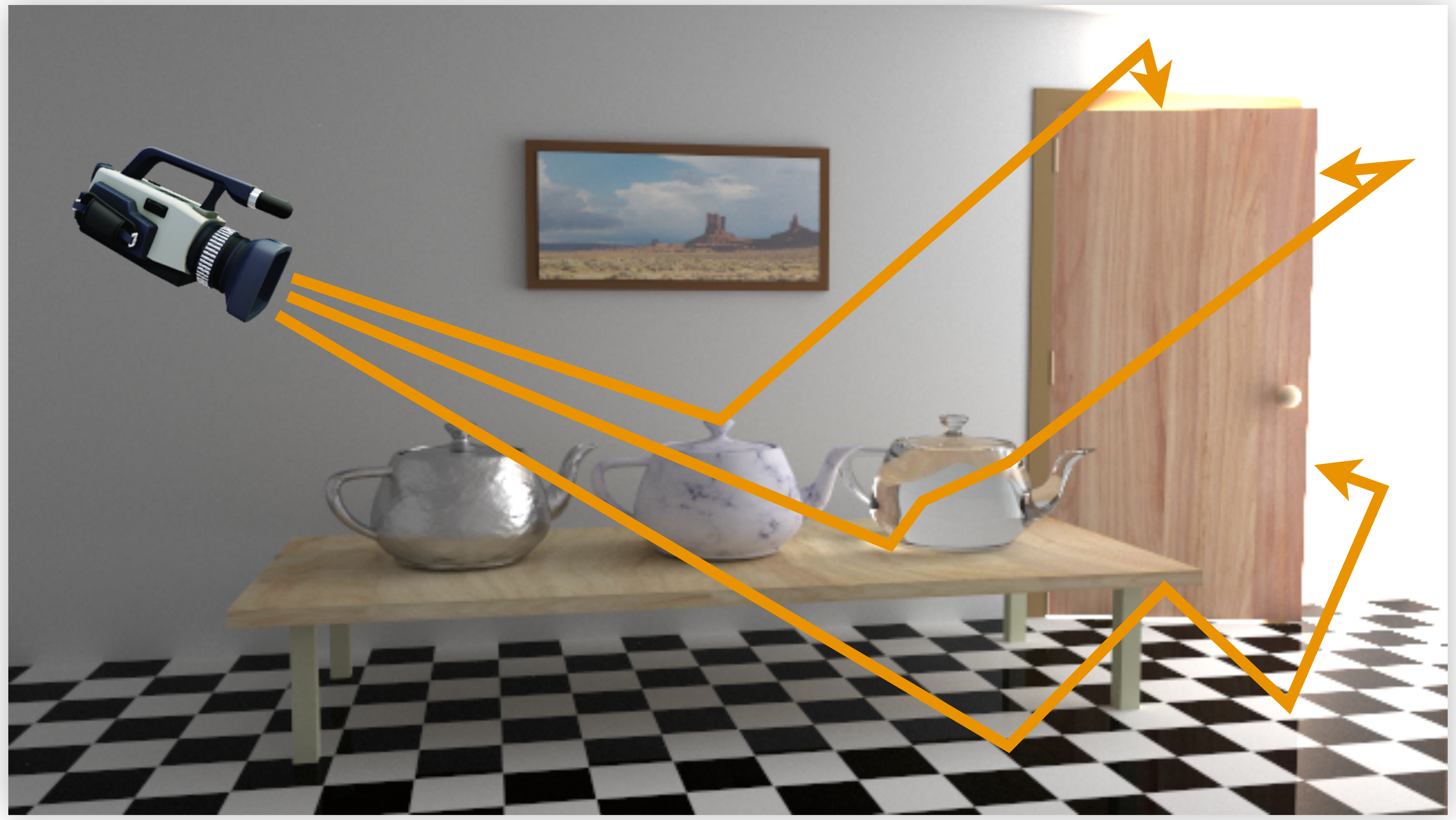
min KL-divergence



Neural network

Training from Monte Carlo samples requires careful weighting

$p(x;\theta)$ distributed

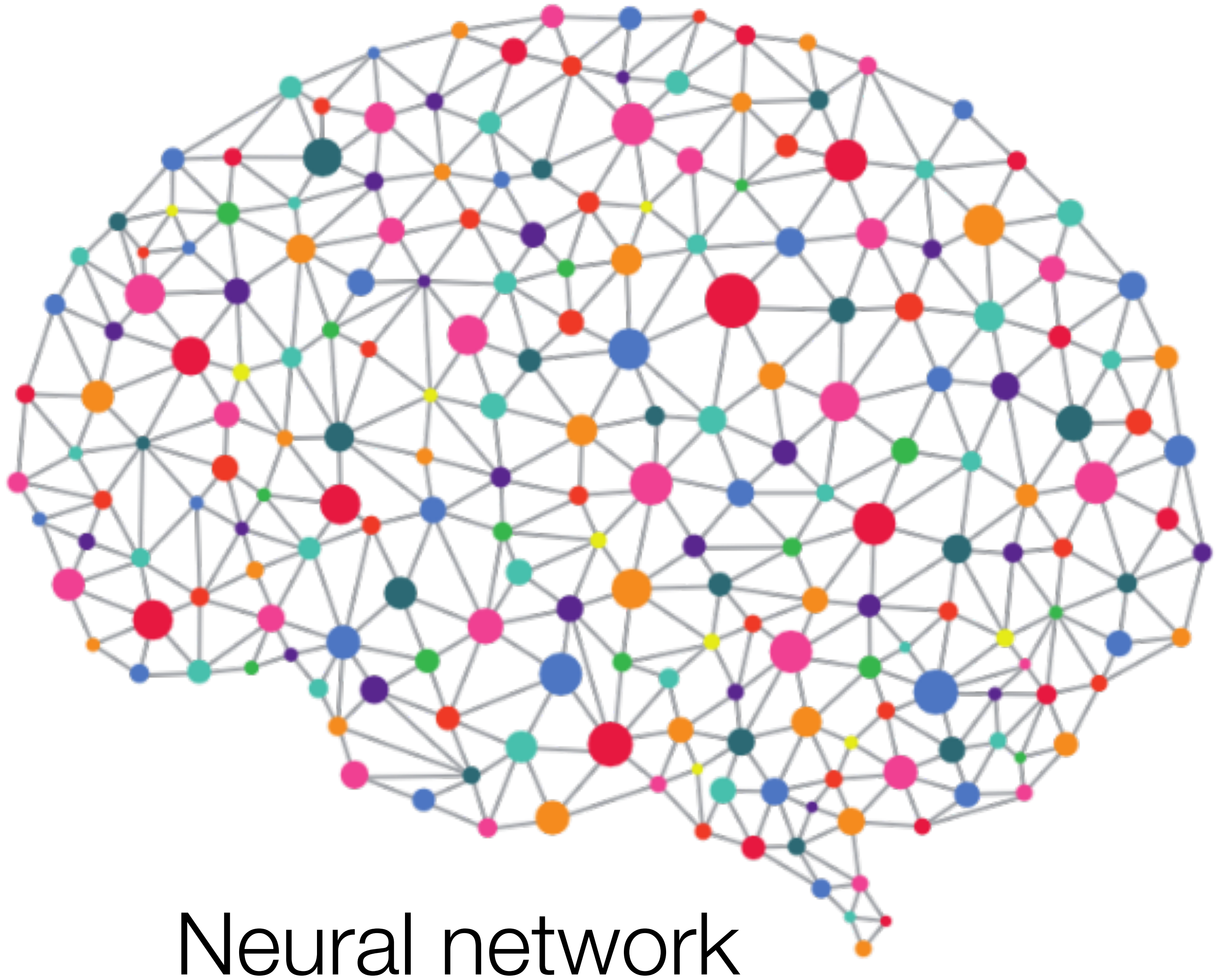


Training data

Optimize

$$\frac{f(x)}{p(x;\theta)} \nabla_\theta \log p(x;\theta)$$

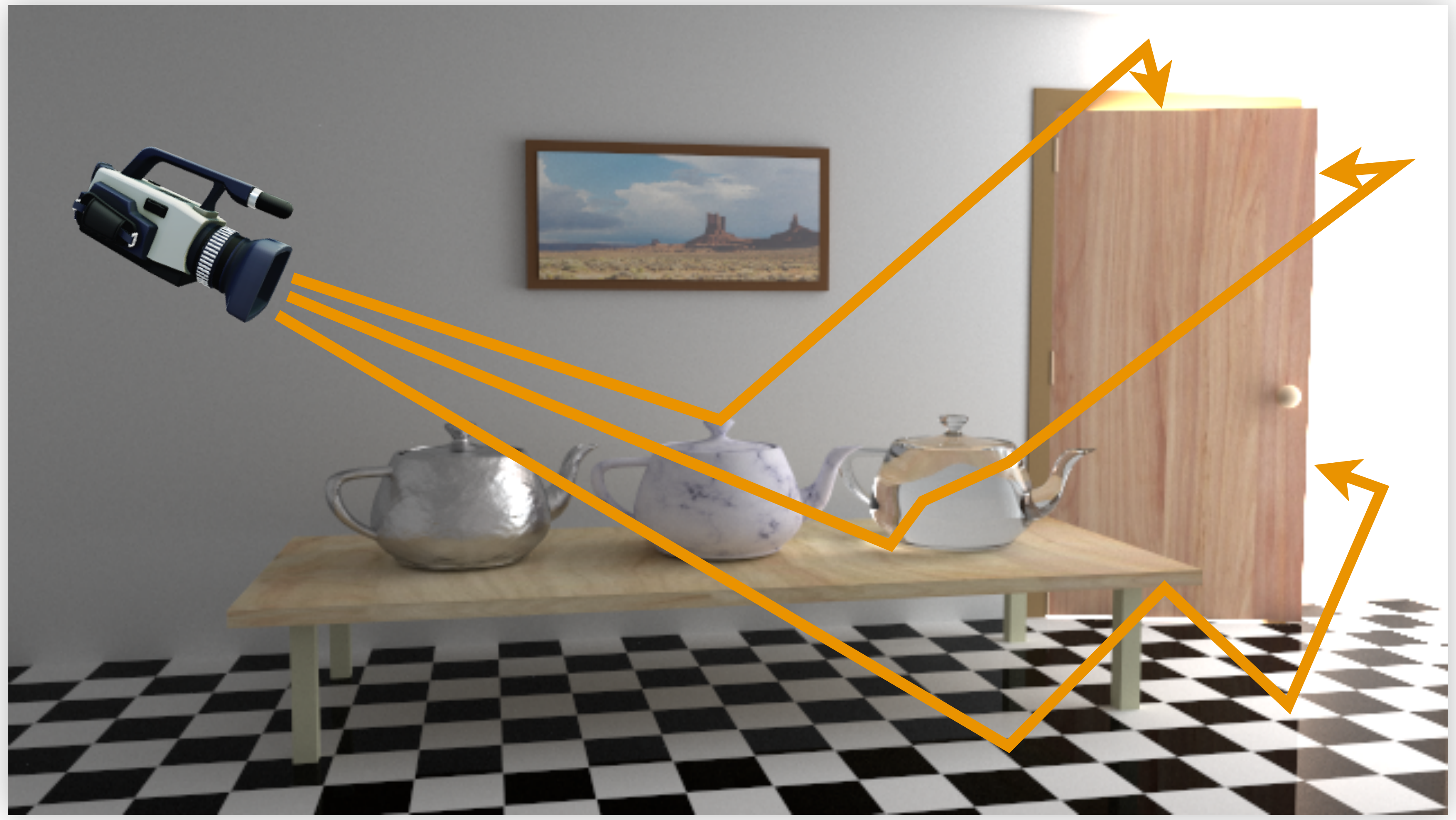
min KL-divergence



Neural network

Training from Monte Carlo samples requires careful weighting

$p(x;\theta)$ distributed

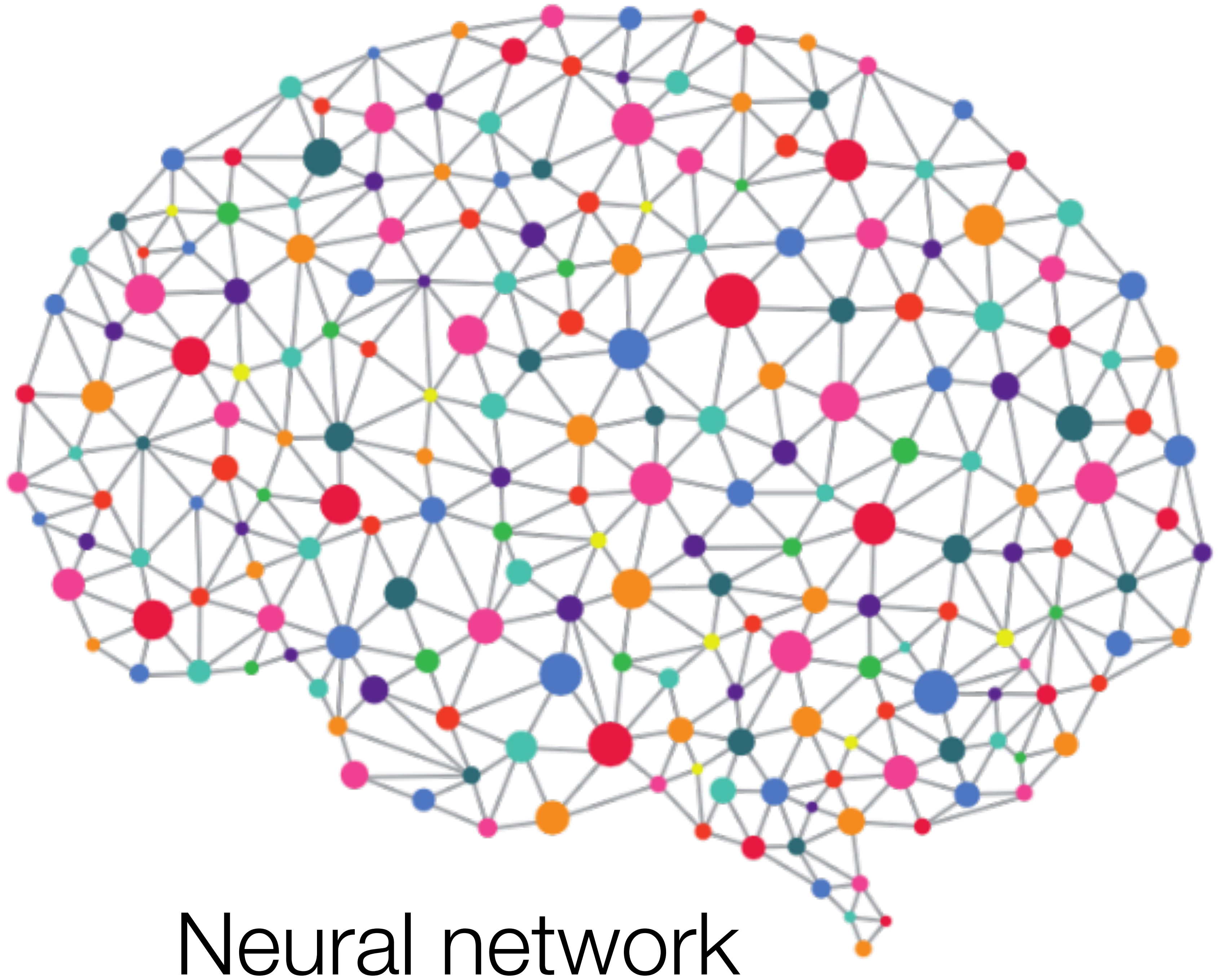


Training data

Optimize

$$\frac{f(x)^2}{p(x;\theta)^2}\nabla_\theta \log p(x;\theta)$$

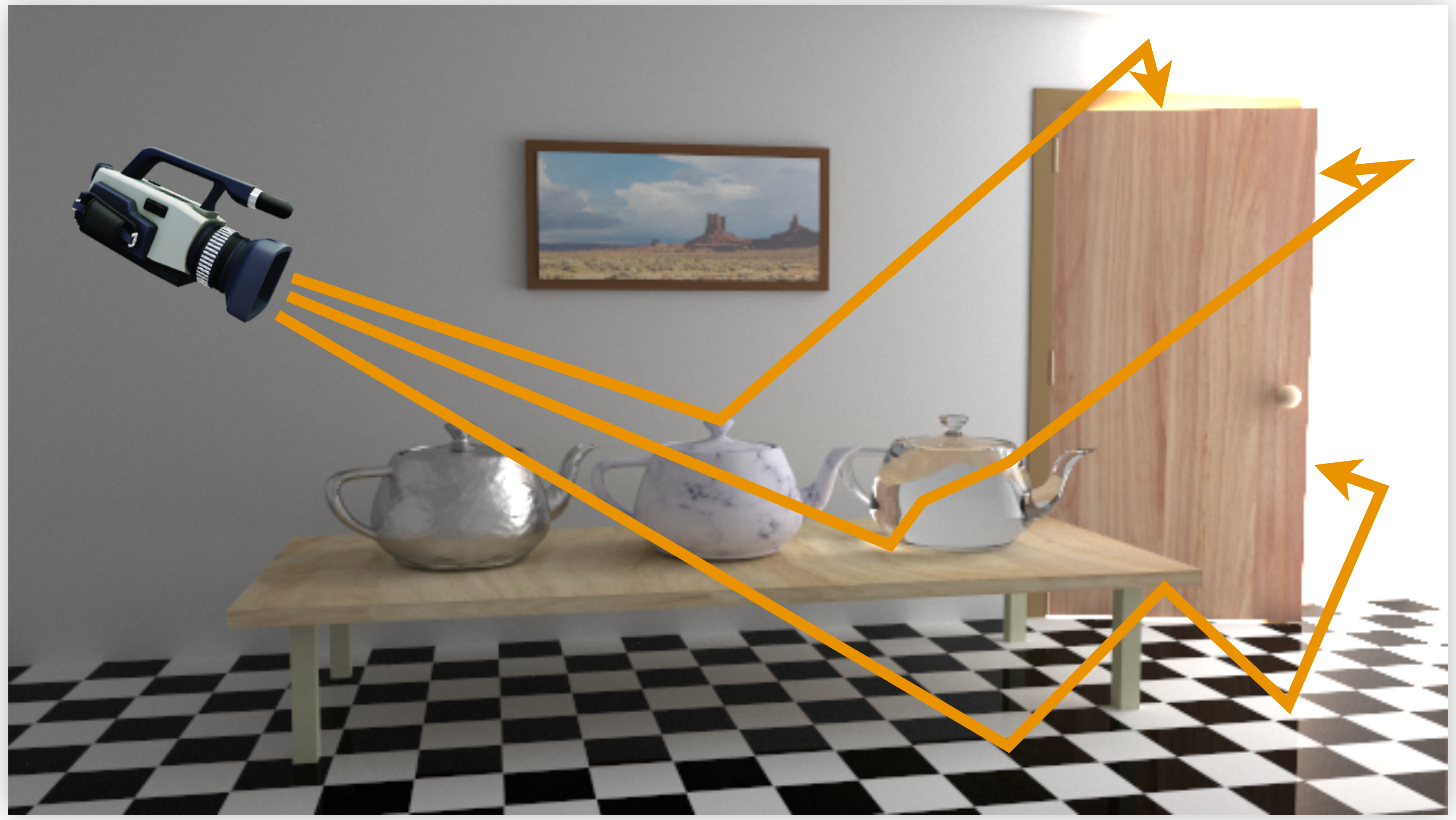
min variance



Neural network

Training from Monte Carlo samples requires careful weighting

$p(x;\theta)$ distributed

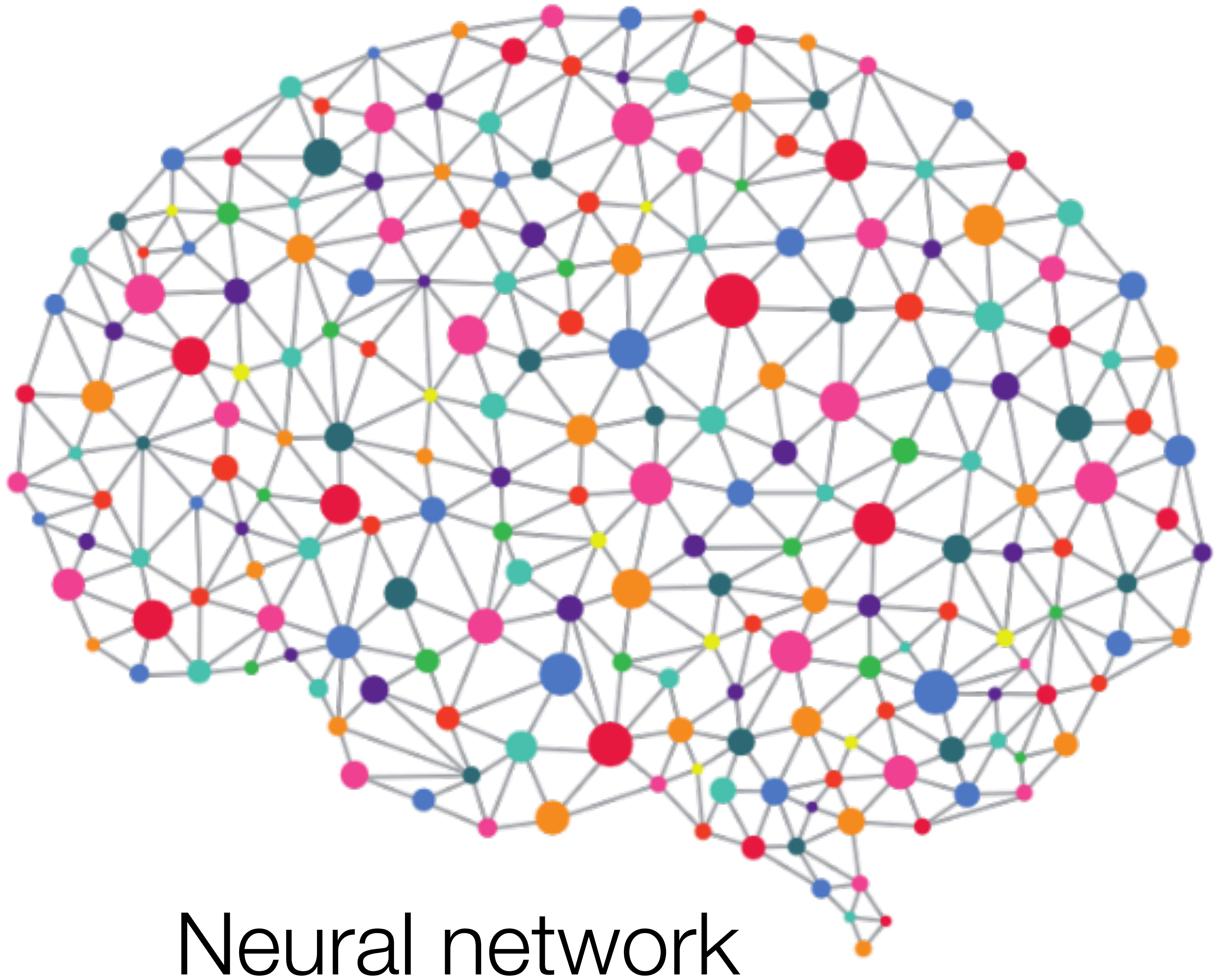


Training data

Optimize

$$\frac{f(x)^2}{p(x;\theta)^2}\nabla_\theta \log p(x;\theta)$$

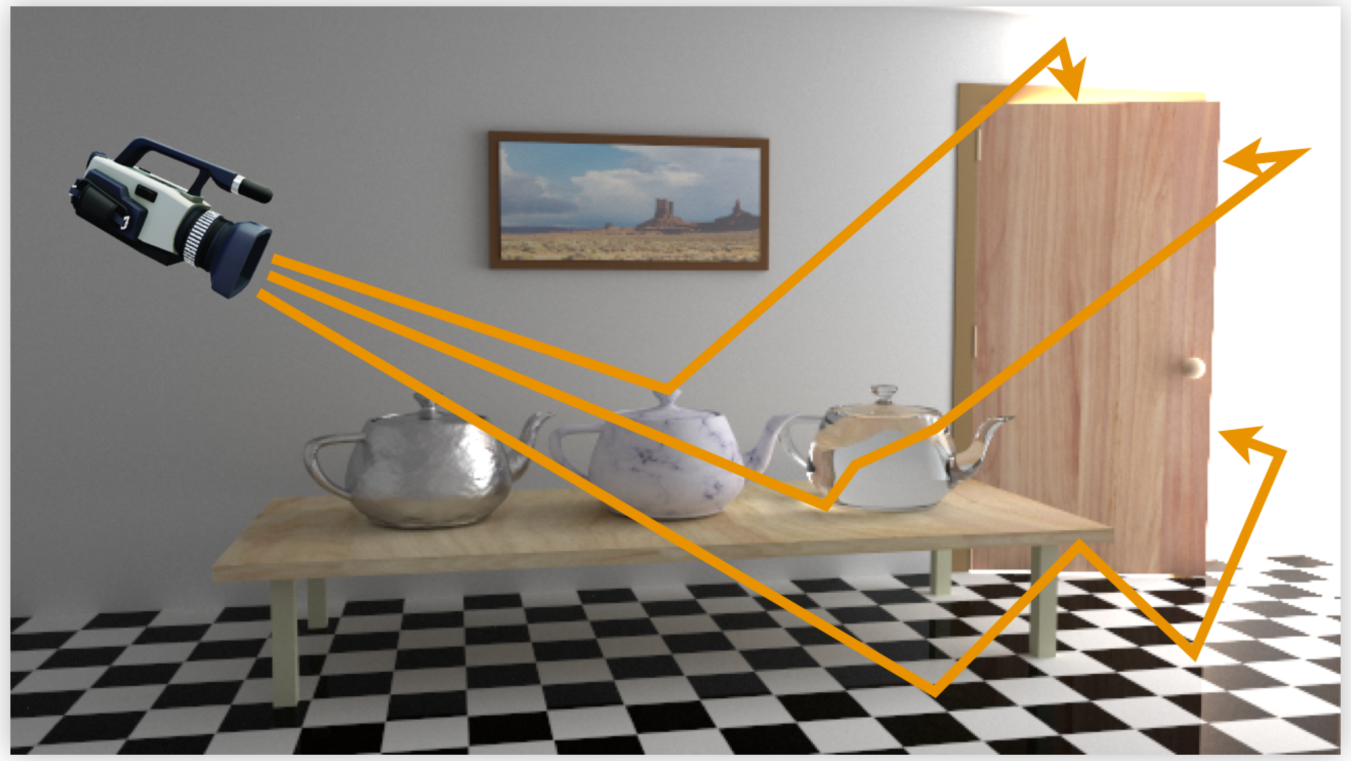
min χ^2-divergence



Neural network

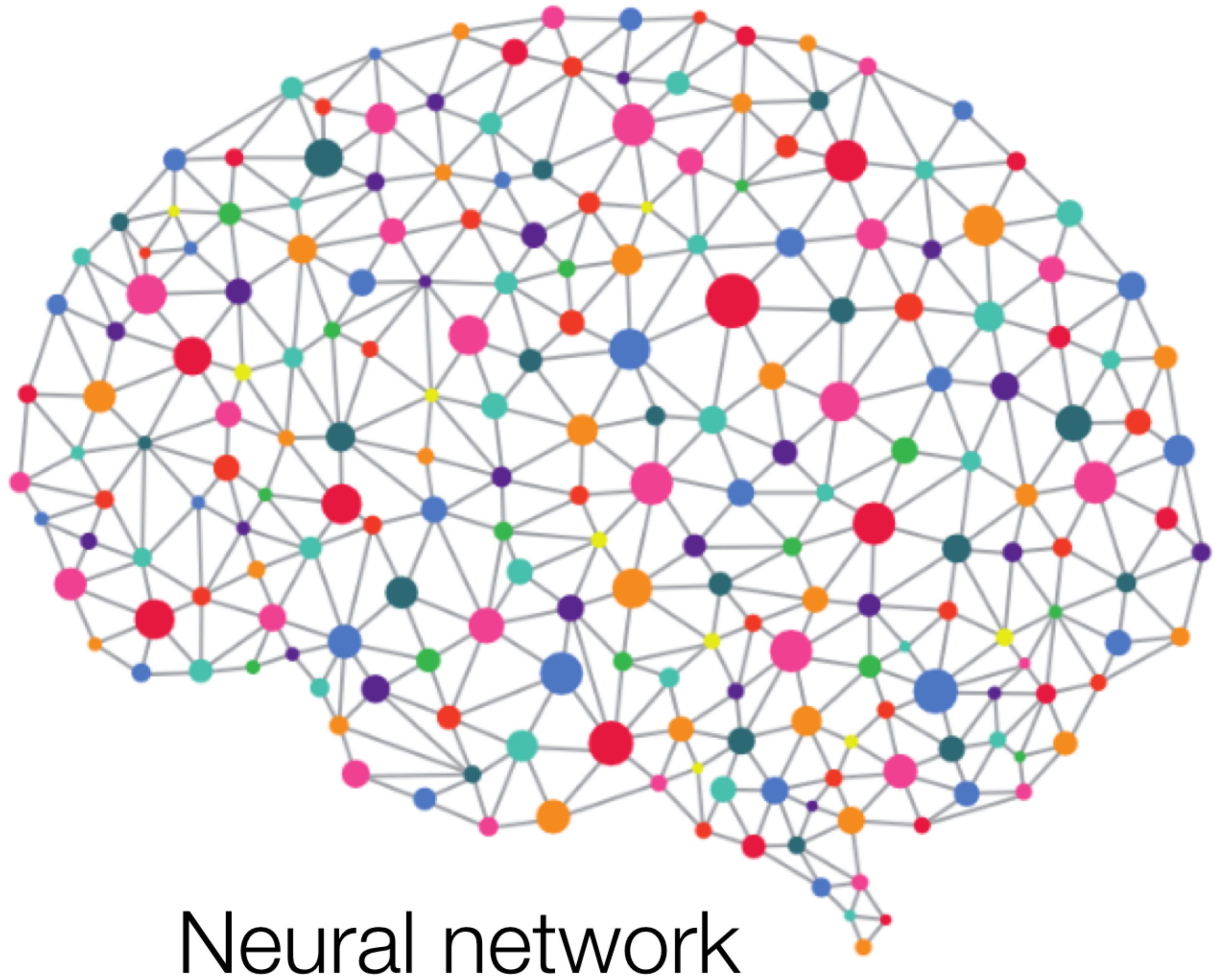
Putting it together...

Sample



Path tracer

Feedback loop



Neural network

Optimize

1 path per pixel

Path tracing **Neural path guiding**

2 paths per pixel

Path tracing | **Neural path guiding**

4 paths per pixel

Path tracing **Neural path guiding**

8 paths per pixel

16 paths per pixel


32 paths per pixel
Path tracing
Neural path guiding

64 paths per pixel

128 paths per pixel
Path tracing
Neural path guiding

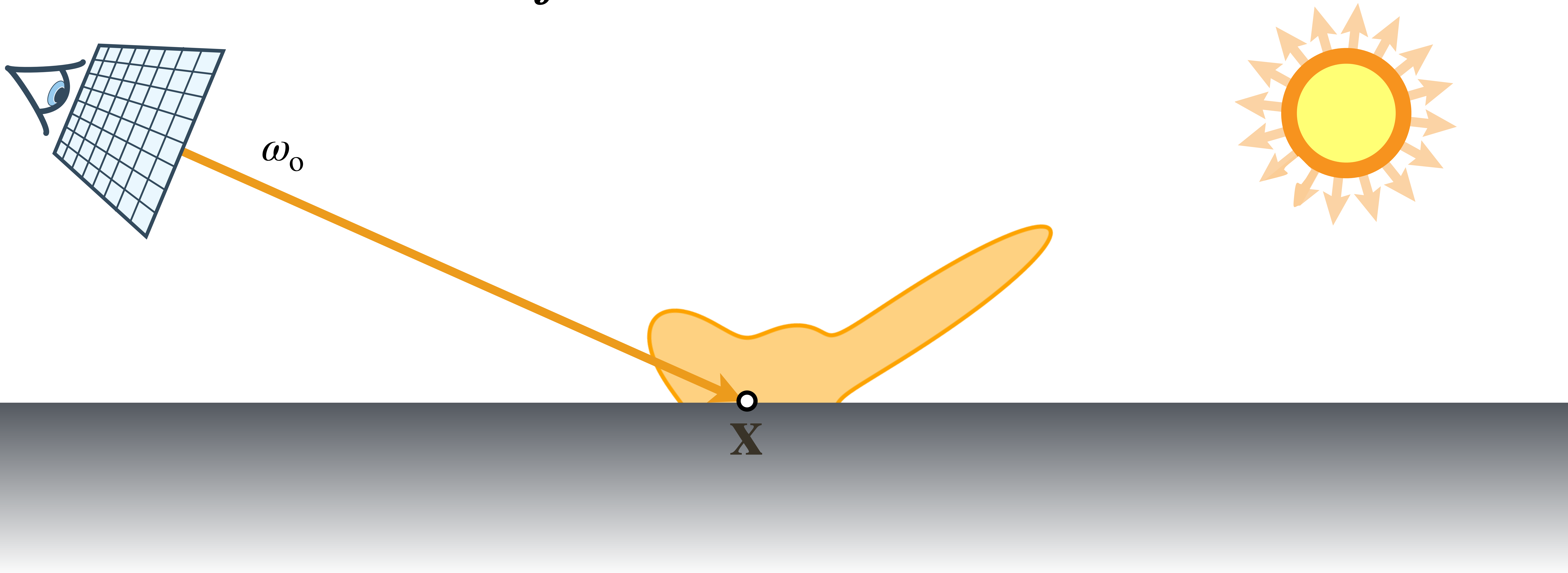
256 paths per pixel


512 paths per pixel
Path tracing
Neural path guiding

Product guiding

Product path guiding

$$L_r(\mathbf{x}, \omega_o) = \int L_i(\mathbf{x}, \omega_i) f(\mathbf{x}, \omega_i, \omega_o) \cos\theta \, d\omega_i$$

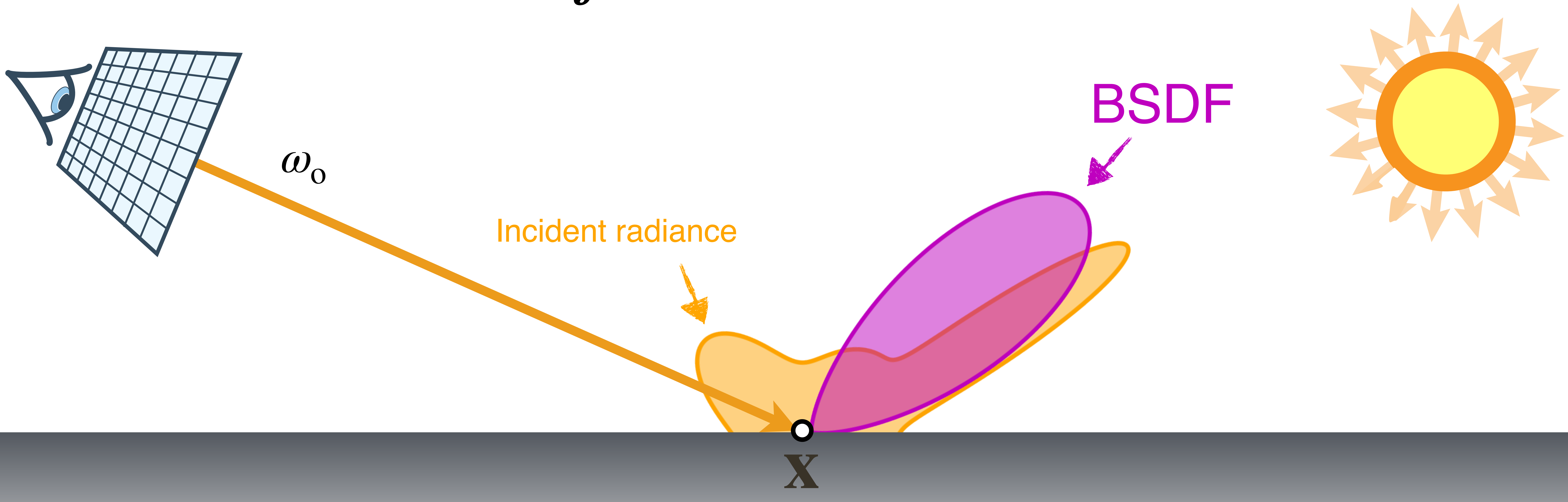


Product path guiding

$$L_r(\mathbf{x}, \omega_o) = \int L_i(\mathbf{x}, \omega_i) f(\mathbf{x}, \omega_i, \omega_o) \cos\theta \, d\omega_i$$

Product path guiding

$$L_r(\mathbf{x}, \omega_o) = \int L_i(\mathbf{x}, \omega_i) f(\mathbf{x}, \omega_i, \omega_o) \cos\theta \, d\omega_i$$

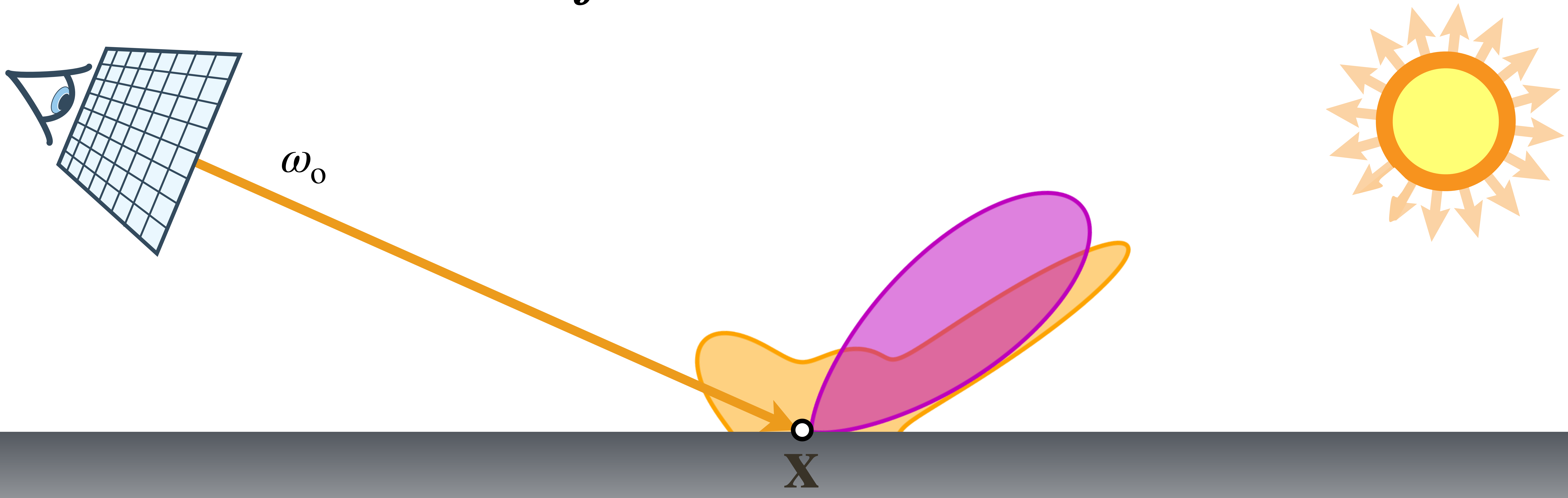


Product path guiding

$$L_r(\mathbf{x}, \omega_o) = \int L_i(\mathbf{x}, \omega_i) f(\mathbf{x}, \omega_i, \omega_o) \cos\theta \; d\omega_i$$

Product path guiding

$$L_r(\mathbf{x}, \omega_o) = \int L_i(\mathbf{x}, \omega_i)\, f(\mathbf{x}, \omega_i, \omega_o) \cos\theta \; d\omega_i$$

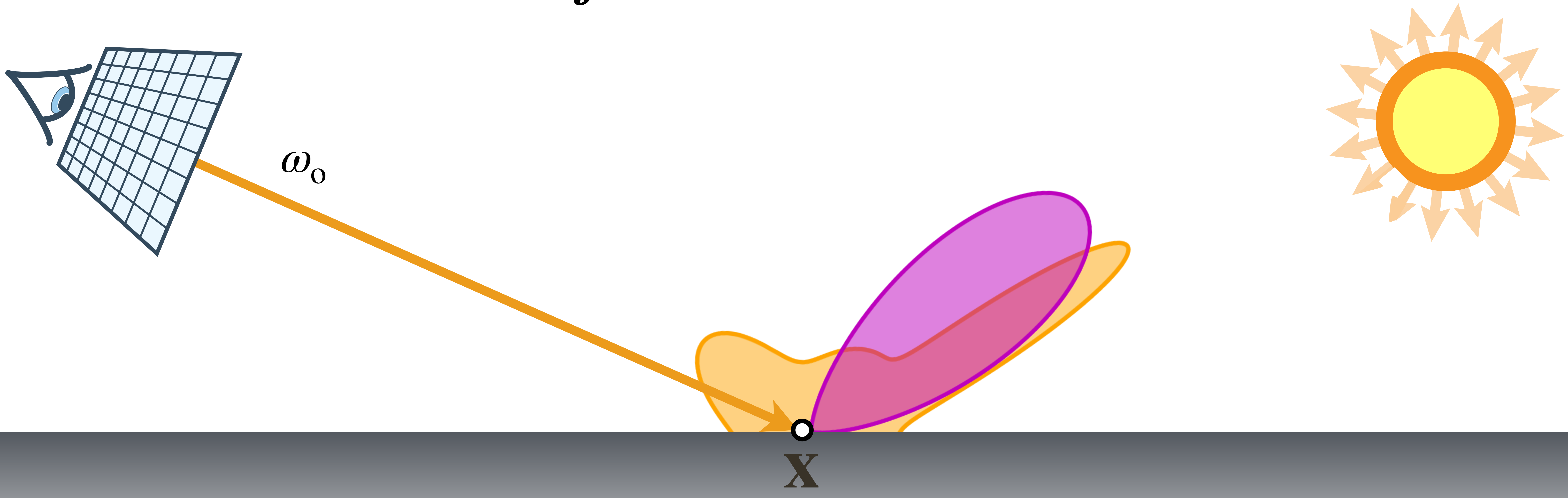


Product path guiding

$$L_r(\mathbf{x}, \omega_o) = \int L_i(\mathbf{x}, \omega_i) f(\mathbf{x}, \omega_i, \omega_o) \cos\theta \, d\omega_i$$

Product path guiding

$$L_\mathrm{r}(\mathbf{x}, \omega_\mathrm{o}) = \int L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i})\, f(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o}) \cos\theta \; d\omega_\mathrm{i}$$

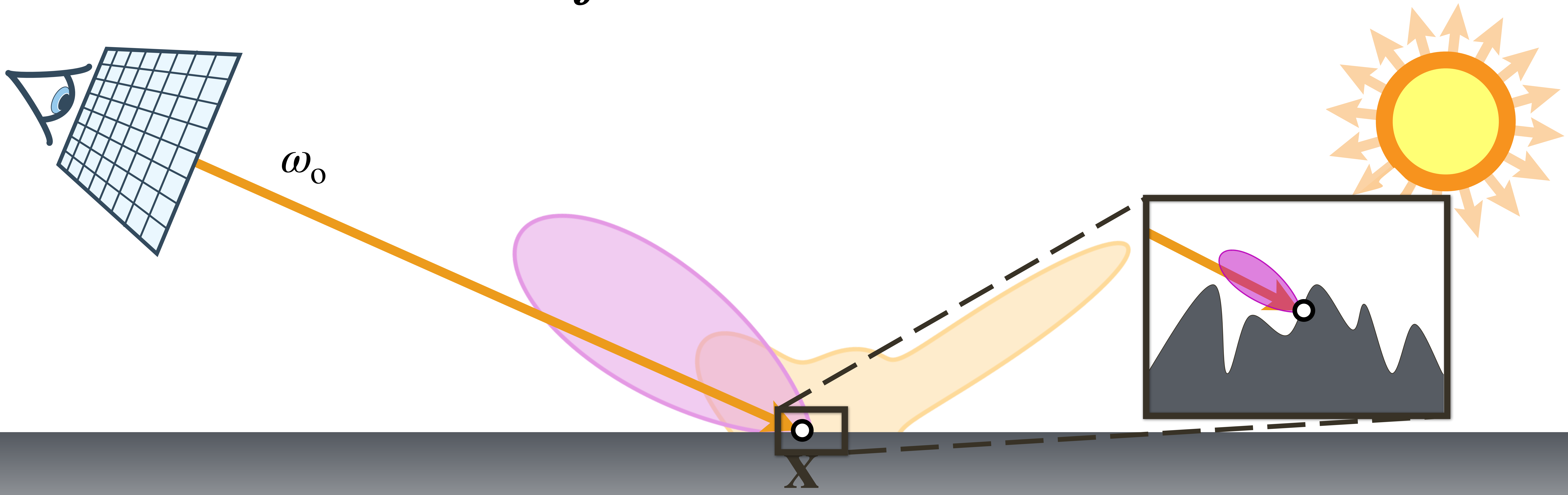


Product path guiding

$$L_r(\mathbf{x}, \omega_o) = \int L_i(\mathbf{x}, \omega_i) f(\mathbf{x}, \omega_i, \omega_o) \cos\theta \, d\omega_i$$

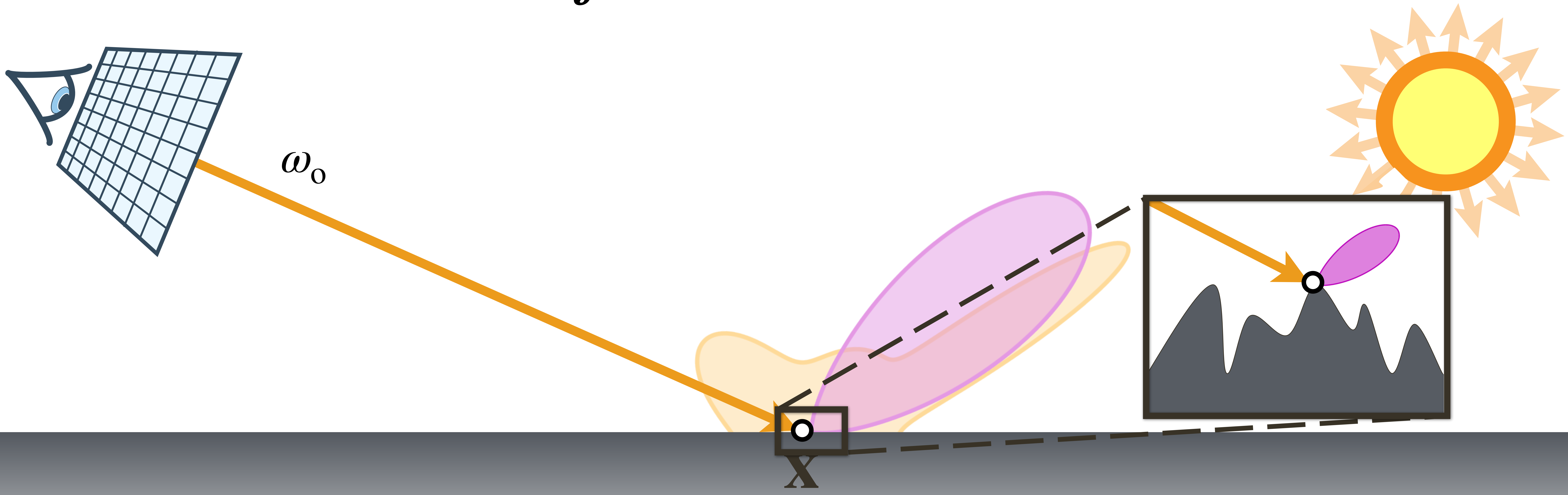
Product path guiding

$$L_r(\mathbf{x}, \omega_o) = \int L_i(\mathbf{x}, \omega_i) f(\mathbf{x}, \omega_i, \omega_o) \cos\theta \; d\omega_i$$



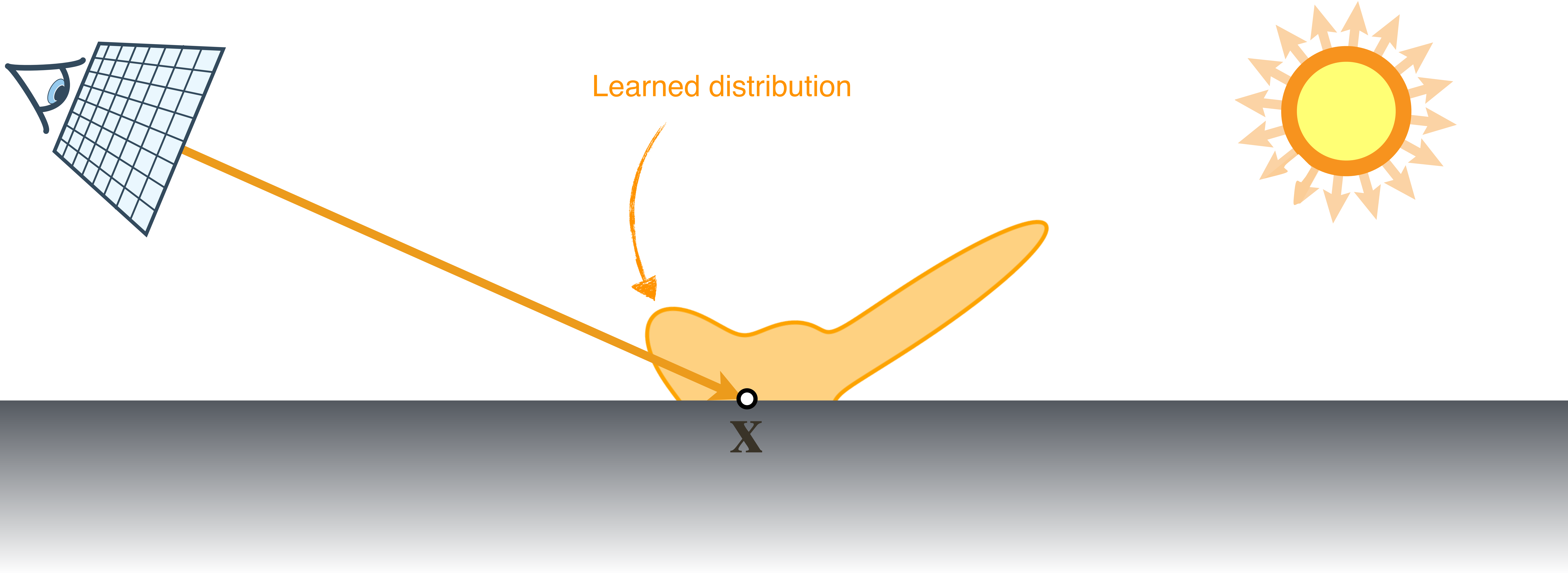
Product path guiding

$$L_r(\mathbf{x}, \omega_o) = \int L_i(\mathbf{x}, \omega_i) f(\mathbf{x}, \omega_i, \omega_o) \cos\theta \, d\omega_i$$

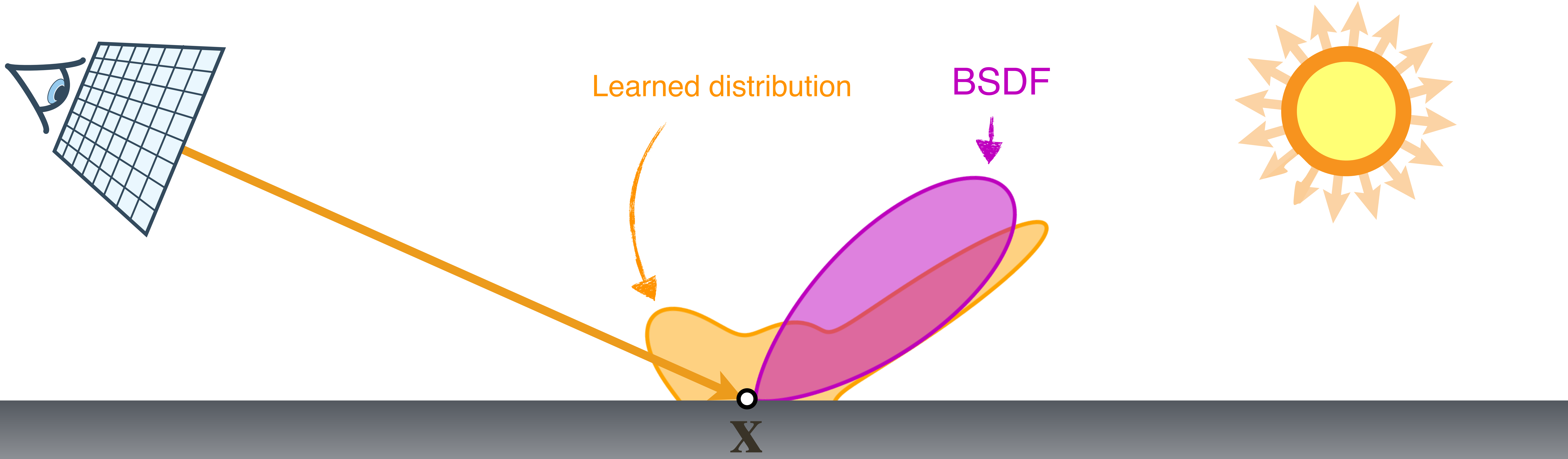


MIS optimization

MIS-aware optimization

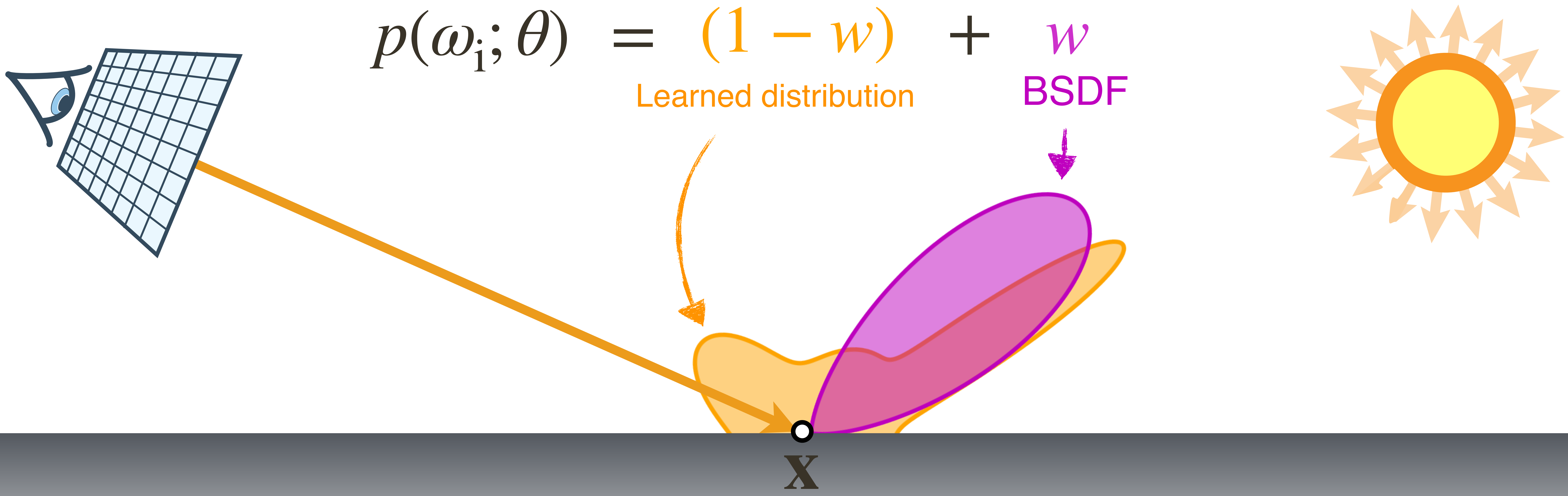


MIS-aware optimization



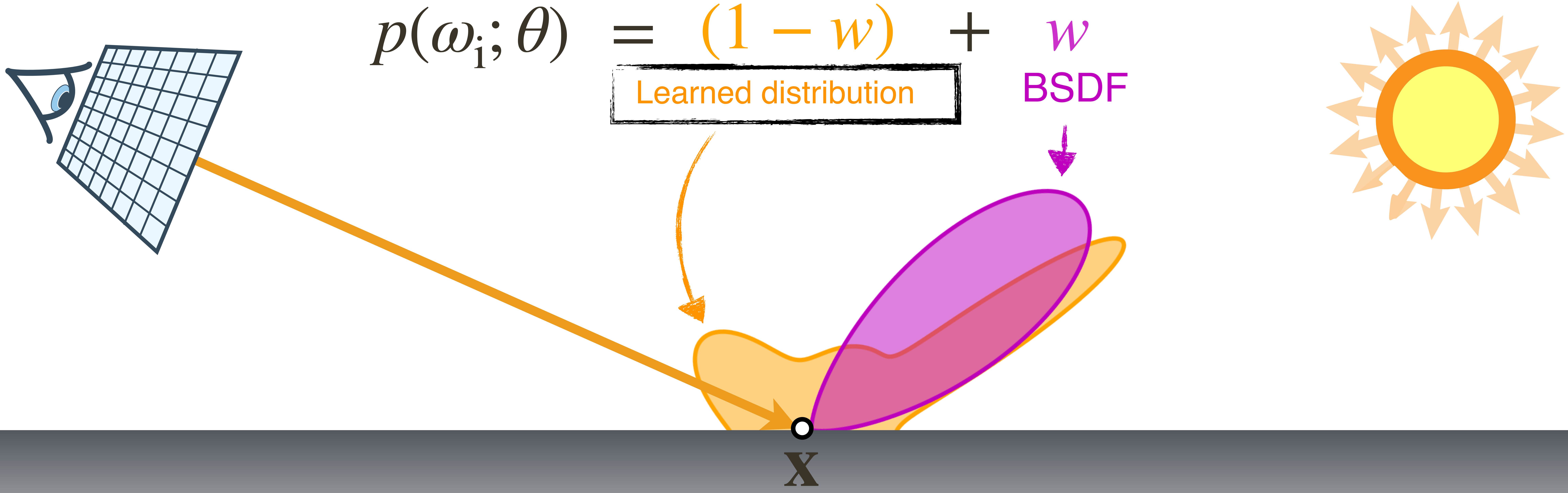
MIS-aware optimization

$$p(\omega_\mathrm{i};\theta) \; = \; (1-w) \; + \; w$$



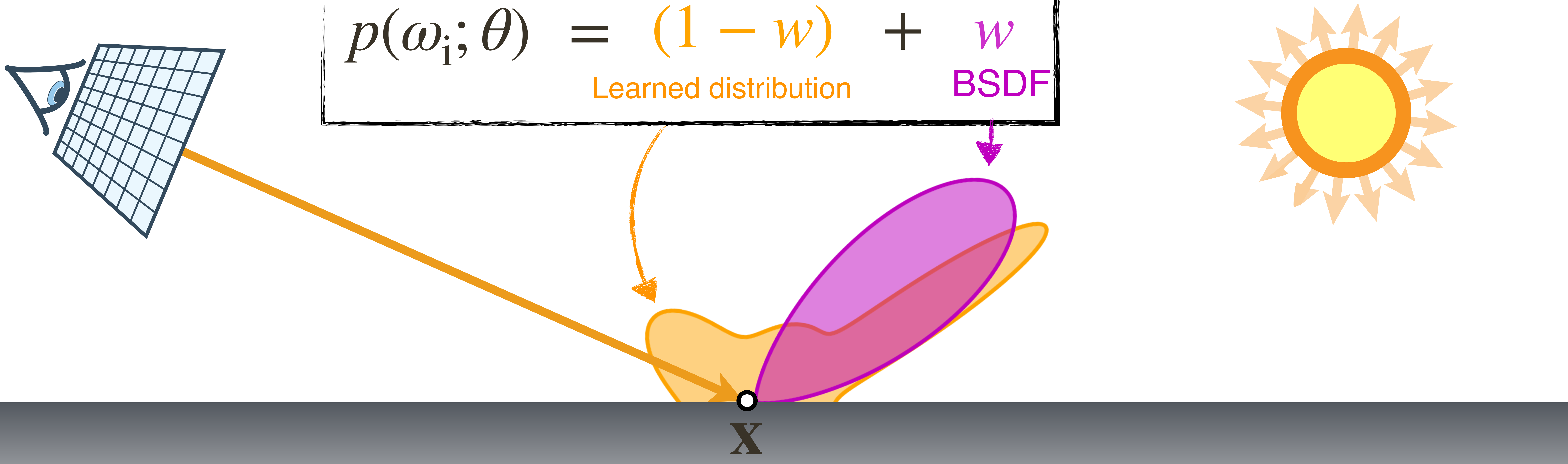
MIS-aware optimization

$$p(\omega_{\mathrm{i}};\theta) = \underbrace{(1-w)}_{\text{Learned distribution}} + \underbrace{w}_{\text{BSDF}}$$



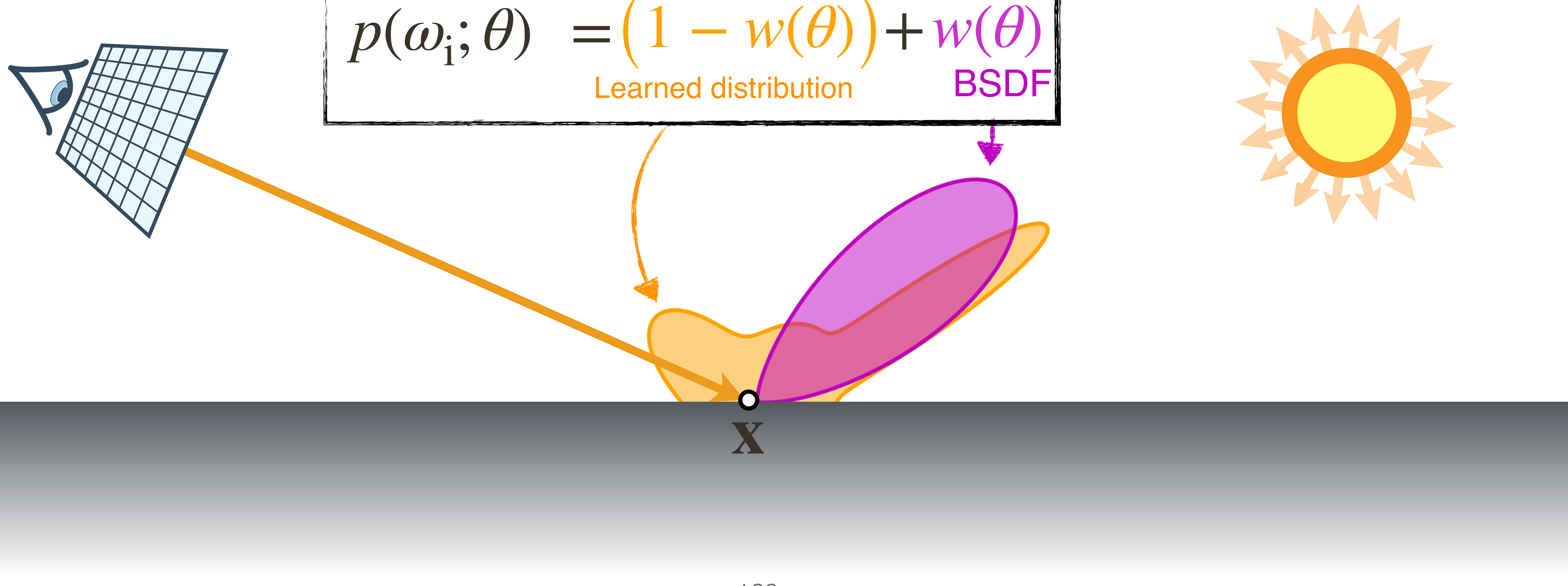
MIS-aware optimization

$$p(\omega_{\mathrm{i}};\theta) \;=\; \underbrace{(1-w)}_{\text{Learned distribution}} \;+\; \underbrace{w}_{\text{BSDF}}$$



MIS-aware optimization

$$p(\omega_{\mathrm{i}};\theta) = \underbrace{\big(1 - w(\theta)\big)}_{\text{Learned distribution}} + \underbrace{w(\theta)}_{\text{BSDF}}$$



Results

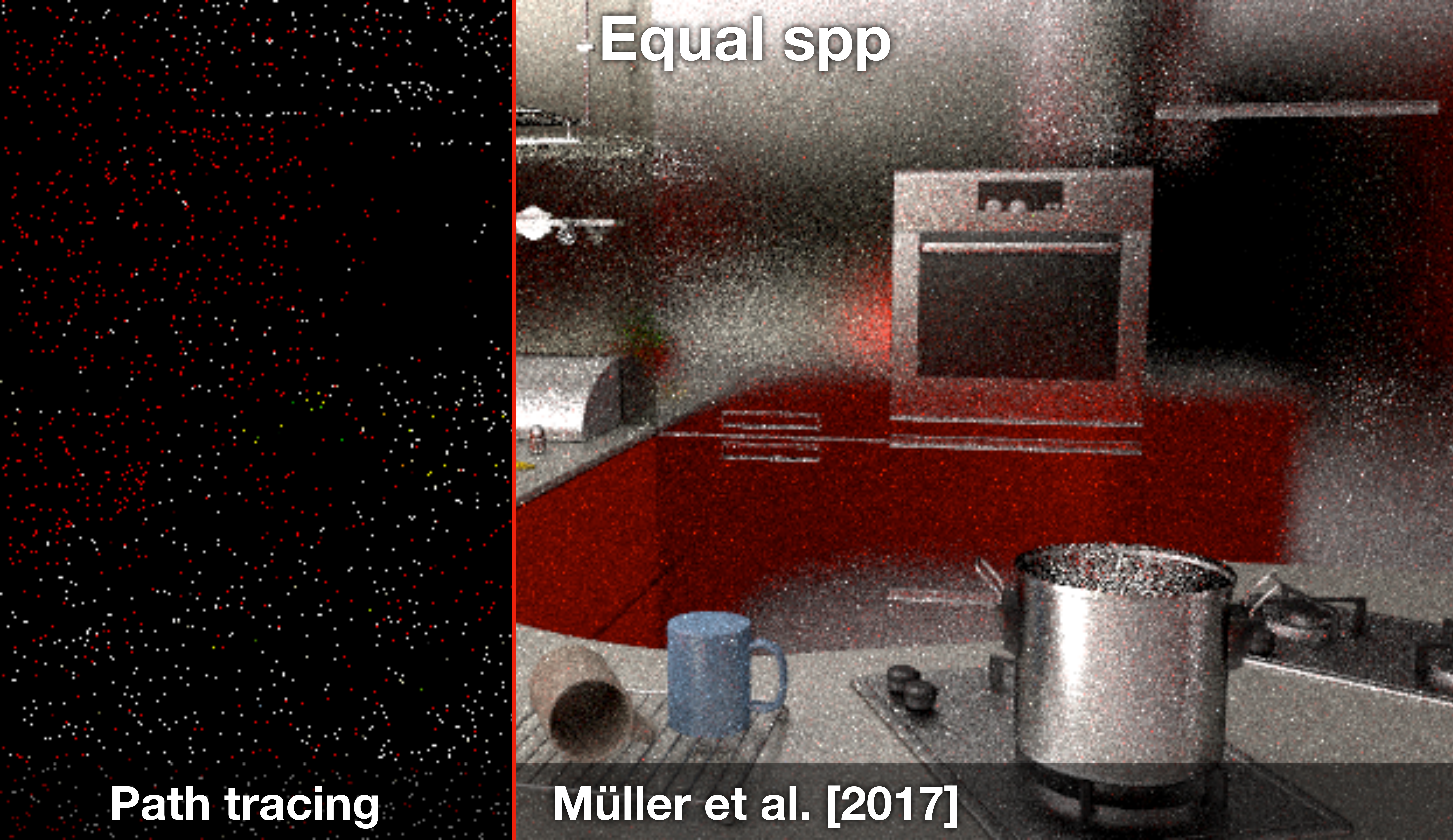

Equal time
Path tracing


Equal time
Path tracing
Müller et al. [2017]


Equal time
Path tracing
Müller et al. [2017]
Neural path guiding

Equal spp
Path tracing
Müller et al. [2017]
Neural path guiding

Equal spp


Equal spp
Path tracing
Müller et al. [2017]


Equal spp
Path tracing
Müller et al. [2017]
Neural path guiding

Conclusion

Conclusion

- Neural networks can drive unbiased MC integration
- Complicated integrands (e.g. product path guiding)
- Computational cost of neural path guiding is high, but quality is state of the art

References

A frequency analysis of light transport, Durand et al. SIGGRAPH 2005

Frequency Analysis and Sheared Reconstruction for Rendering Motion Blur, Egan et al. SIGGRAPH 2009

Temporal Light Field Reconstruction for Rendering Distribution Effects, Lehtinen et al. SIGGRAPH 2011

On Filtering the Noise from the Random Parameters in Monte Carlo Rendering, Sen and Darabi 2012

A Machine Learning Approach for Filtering Monte Carlo Noise, Kalantari et al. SIGGRAPH 2015

Interactive Reconstruction of Monte Carlo Image Sequences using a Recurrent Denoising Autoencoder, Chaitanya et al. SIGGRAPH 2017

Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings, Bako et al. SIGGRAPH 2017

Sample-based Monte Carlo Denoising using a Kernel-Splatting Network, Gharbi et al. SIGGRAPH 2019

NICE: Non-linear Independent Components Estimation

Normalizing Flows: An Introduction and Review of Current Methods

Neural Importance Sampling SIGGRAPH 2019