Denoising Algorithms: Path to Neural Networks I

Image courtesy Bako et al. [2017]
Recap:
Reconstruction using Spatio-temporal Sampling
Image-space Adaptive Sampling

Hachisuka et al. [2008]
Image-space Adaptive Sampling

Multidimensional Adaptive Sampling

Hachisuka et al. [2008]
Depth of field

- Show a few viewpoints
- Show x-u slice
- Integrate over aperture

Slide from Jakko Lehtinen
Depth of field

- Show a few viewpoints
- Show x-u slice
- Integrate over aperture

Slide from Jakko Lehtinen
Depth of field

- Show a few viewpoints
- Show x-u slice
- Integrate over aperture

Slide from Jakko Lehtinen
Visibility: SameSurface

Slide from Jakko Lehtinen
Visibility: SameSurface

The trajectories of samples originating from a single apparent surface never intersect.
The trajectories of samples originating from a single apparent surface never intersect.

Visibility: SameSurface
The trajectories of samples originating from a single apparent surface never intersect.
Visibility: SameSurface

The trajectories of samples originating from a single apparent surface never intersect.
Introduction
Denoising using Data
Path to Machine Learning
Introduction
Denoising using Data
Path to Machine Learning

MLP based Denoising
Filtering Monte Carlo Noise From Random Parameters

Sen and Darabi [2012]
input Monte Carlo (8 samples/pixel)

after RPF (8 samples/pixel)
High-dimensional Monte Carlo Integration

\[
I(i, j) = \int_{i-\frac{1}{2}}^{i+\frac{1}{2}} \int_{j-\frac{1}{2}}^{j+\frac{1}{2}} \cdots \int_{-1}^{1} \int_{-1}^{1} \int_{t_0}^{t_1} f(x, y, \cdots, u, v, t) \, dt \, dv \, du \, \cdots \, dy \, dx
\]
(a) Input MC (8 spp)  
(b) Dependency on \((u, v)\)  
(c) Our approach (RPF)
Parameters in Monte Carlo estimator

Random parameters: \( \mathbf{r} = \{r_1, r_2, \ldots, r_n\} \)

Color: \( c_i \leftarrow f(p_{i,1}, p_{i,2}; r_{i,1}, r_{i,2}, \ldots, r_{i,n}) \)

- screen position
- random parameters
Random Parameters Classification

Random parameter for each pixel:

\[ x_i \leftarrow f(p_{i,1}, p_{i,2}; r_{i,1}, r_{i,2}, \ldots, r_{i,n}) \]

\[ x_i = \{ p_{i,1}, p_{i,2}; r_{i,1}, \ldots, r_{i,n}; f_{i,1}, \ldots, f_{i,m}; c_{i,1}, c_{i,2}, c_{i,3} \} \]

- **screen position**
- **random parameters**
- **scene features**
- **sample color**
Random Parameters Classification

Random parameter for each pixel:

$$x_i \leftarrow f(p_{i,1}, p_{i,2}; r_{i,1}, r_{i,2}, \ldots, r_{i,n})$$

$$x_i = \{p_{i,1}, p_{i,2}, r_{i,1}, \ldots, r_{i,n}, f_{i,1}, \ldots, f_{i,m}, c_{i,1}, c_{i,2}, c_{i,3}\}$$

- screen position
- random parameters
- scene features
- sample color
Random Parameters Classification

Random parameter for each pixel:

\[ x_i \leftarrow f(p_{i,1}, p_{i,2}; r_{i,1}, r_{i,2}, \ldots, r_{i,n}) \]

\[ x_i = \{ p_{i,1}, p_{i,2}, r_{i,1}, \ldots, r_{i,n}, f_{i,1}, \ldots, f_{i,m}, c_{i,1}, c_{i,2}, c_{i,3} \} \]

- screen position
- random parameters
- scene features
- sample color
Random Parameters Classification

Random parameter for each pixel:

\[ x_i \leftarrow f(p_{i,1}, p_{i,2}; r_{i,1}, r_{i,2}, \ldots, r_{i,n}) \]

\[ x_i = \{p_{i,1}, p_{i,2}, r_{i,1}, \ldots, r_{i,n}, f_{i,1}, \ldots, f_{i,m}, c_{i,1}, c_{i,2}, c_{i,3}\} \]

- screen position
- random parameters
- scene features
- sample color
Random Parameters Classification

Random parameter for each pixel:

\[ \mathbf{x}_i \leftarrow f(p_{i,1}, p_{i,2}; r_{i,1}, r_{i,2}, \ldots, r_{i,n}) \]

\[ \mathbf{x}_i = \{p_{i,1}, p_{i,2}, r_{i,1}, \ldots, r_{i,n}, f_{i,1}, \ldots, f_{i,m}, c_{i,1}, c_{i,2}, c_{i,3}\} \]

- screen position
- random parameters
- scene features
- sample color
Gaussian Filtering

\[ GC[I]_p = \sum_{q \in S} G_\sigma(\|p - q\|) I_q, \quad G_\sigma(x) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right) \]
Bilateral Filtering

\[ BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q \]

\[ W_p = \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) \]
Bilateral Filtering

\[ BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q \]

\[ W_p = \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) \]
Bilateral Filtering

\[ BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q \]

\[ W_p = \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) \]

Bilateral filter weights at the central pixel

Spatial weight

Range weight

Input

Multiplication of range and spatial weights

Result
Bilateral Filtering

\[
BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) \cdot G_{\sigma_r}(\|I_p - I_q\|) \cdot I_q
\]

\[
W_p = \sum_{q \in S} G_{\sigma_s}(\|p - q\|) \cdot G_{\sigma_r}(\|I_p - I_q\|)
\]
Bilateral Filtering

\[
BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(||p - q||) G_{\sigma_r}(|I_p - I_q|) I_q
\]

\[
W_p = \sum_{q \in S} G_{\sigma_s}(||p - q||) G_{\sigma_r}(|I_p - I_q|)
\]
Bilateral Filtering

\[ BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q \]

\[ W_p = \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) \]

Bilateral filter weights at the central pixel

Spatial weight

Range weight

Multiplication of range and spatial weights
Bilateral Filtering

\[ BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q \]

\[ W_p = \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) \]

Bilateral filter weights at the central pixel

Spatial weight
Range weight

Multiplication of range and spatial weights

Result
Bilateral vs Gaussian Filtering

\[ \sigma_s / \sigma_r \]

<table>
<thead>
<tr>
<th></th>
<th>0.05</th>
<th>0.2</th>
<th>0.8</th>
<th>GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
<tr>
<td>8</td>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
<td><img src="image7.png" alt="Image" /></td>
<td><img src="image8.png" alt="Image" /></td>
</tr>
<tr>
<td>16</td>
<td><img src="image9.png" alt="Image" /></td>
<td><img src="image10.png" alt="Image" /></td>
<td><img src="image11.png" alt="Image" /></td>
<td><img src="image12.png" alt="Image" /></td>
</tr>
</tbody>
</table>
Bilateral Filtering of Features

\[ w_{ij} = \exp\left[ -\frac{1}{2\sigma_p^2} \sum_{1 \leq k \leq 2} (\bar{p}_{i,k} - \bar{p}_{j,k})^2 \right] \times \]

\[ \exp\left[ -\frac{1}{2\sigma_c^2} \sum_{1 \leq k \leq 3} \alpha_k (\bar{c}_{i,k} - \bar{c}_{j,k})^2 \right] \times \]

\[ \exp\left[ -\frac{1}{2\sigma_f^2} \sum_{1 \leq k \leq m} \beta_k (\bar{f}_{i,k} - \bar{f}_{j,k})^2 \right], \]
Dependency on Random Parameters

Input Monte Carlo (8 spp)  Dependency of color on random parameters \( (D^c) \)  Dependency of color on screen position \( (D^p) \)  Fractional dependency on random parameters \( (W^c) \)  Our approach (RPF)  Reference MC (512 spp)
Bilateral Weights

\[ W_{f,k}^r = \frac{D_{f,k}^r}{D_{f,k}^r + D_{f,k}^p} \]

\[ \beta_k = 1 - W_{f,k}^r \]

\[ W_{c,k}^r = \frac{D_{c,k}^r}{D_{c,k}^r + D_{c,k}^p} \]

\[ \alpha_k = 1 - W_{c,k}^r \]
Pixels, Random Params, Features

(a) Screen position
(b) Random parameters
(c) World space coords.
Pixels, Random Params, Features

(d) Surface normals  (e) Texture value  (f) Sample color
The algorithm computes the statistical dependency of (c-f) on the random parameters in (b)
Random Parameter Filtering

(a) Reference  (b) MC Input  (c) RPF  (d) no clustering  (e) no DoF params
Random Parameter Filtering

(a) $W_{c,k}^{r,1}$ and $W_{c,k}^{r,2}$  

(b) $W_{c,k}^{r}$  

(c) Our output (RPF)
Statistical Dependency

Mutual information between two random variables:

\[ \mu(X; Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \]

where, these probabilities are computed over the neighborhood of samples around a given pixel.
Statistical Dependency

Functional dependency of the k-th scene parameter:

\[ D_{f,k}^{r} = \sum_{1 \leq l \leq n} D_{f,k}^{r,l} = \sum_{1 \leq l \leq n} \mu(\bar{f}_{N,k}; \bar{r}_{N,l}) \]

\[ D_{f,k}^{p} = \sum_{1 \leq l \leq 2} D_{f,k}^{p,l} = \sum_{1 \leq l \leq 2} \mu(\bar{f}_{N,k}; \bar{p}_{N,l}) \]

\[ D_{c,k}^{r} = \sum_{1 \leq l \leq n} D_{c,k}^{r,l} = \sum_{1 \leq l \leq n} \mu(\bar{c}_{N,k}; \bar{r}_{N,l}) \]

\[ D_{c,k}^{p} = \sum_{1 \leq l \leq 2} D_{c,k}^{p,l} = \sum_{1 \leq l \leq 2} \mu(\bar{c}_{N,k}; \bar{p}_{N,l}) \]
Statistical Dependency

\[ D_{f,k}^r = \sum_{1 \leq l \leq n} D_{f,k}^{r,l} = \sum_{1 \leq l \leq n} \mu(\mathbf{f}_N,k; \mathbf{r}_N,l) \]

\[ W_{c,k}^f = \frac{D_{c,k}^f}{D_c^r + D_c^p + D_c^f} \]

\[ D_c^r = \sum_{1 \leq k \leq 3} D_{c,k}^r, \quad D_c^p = \sum_{1 \leq k \leq 3} D_{c,k}^p, \quad D_c^f = \sum_{1 \leq k \leq 3} D_{c,k}^f. \]
Weighted Average Bilateral Filtering

\[ c'_{i,k} = \frac{\sum_{j \in N} w_{ij} c_{j,k}}{\sum_{j \in N} w_{ij}} \]
Results

(a) MC Input (8 spp)  
(b) Our approach (RPF)  
(c) $\alpha_k = 0$, $\beta_k = 0$
(c) $\alpha_k = 0, \beta_k = 0$
(d) $\alpha_k = 1, \beta_k = 0$
(e) $\alpha_k = 0, \beta_k = 1$
(f) $\alpha_k = 1, \beta_k = 1$
Results

Reference (8,192 spp)  Input Monte Carlo (8 spp)  MDAS  AWR  À-Trous  Our approach (RPF)
Multi-Layer Perceptrons
History of Neural Networks

• In 1943, McCulloch and Pitts created a computational model for neural networks

• In 1975, Werbos's back propagation algorithm generally accelerated the training of multi-layer networks.

• In 1980s, Recurrent Neural Networks were developed
Classifiers

\[ y_j = f(w_j x_j + b_j) \]
Classifiers

\[ y_j = f(w_j x_j + b_j) \]
Complex Classifiers

\[ y_j = f(w_j x_j + b_j) \]

Complex classifier
Complex Classifiers

What features can produce this decision rule?
Perceptron Classifier

\[ x_1 \quad . \quad . \quad . \quad \cdot \quad . \quad . \quad 1 \]
Perceptron Classifier

\[ x_1 \]  
\[ x_2 \]  
\[ x_3 \]  
\[ x_4 \]  
\[ x_5 \]  
\[ \ldots \]  
\[ 1 \]
Perceptron Classifier
Perceptron Classifier

\[ y = f(w_1 x_1 + w_2 x_2 + ... + w_0) \]
Multi-layer Perceptron

\[ x_1 \]

1
Multi-layer Perceptron

\[ x_1 \]

\[ \sum \rightarrow f \]

\[ 1 \]

\[ \sum \rightarrow f \]

\[ \sum \rightarrow f \]
Multi-layer Perceptron

\[ x_1 \]

\[ \sum \rightarrow f \]

\[ 1 \]

\[ \sum \rightarrow f \]

\[ \sum \rightarrow f \]
Multi-layer Perceptron

\[ x_1 \rightarrow f \]

\[ 1 \rightarrow f \]

\[ w_{11} \]

\[ w_{21} \]

\[ w_{31} \]
Multi-layer Perceptron

\[ x_1 \]

\[ 1 \]

\[ w_{11} \]

\[ w_{10} \]

\[ w_{21} \]

\[ w_{20} \]

\[ w_{31} \]

\[ w_{30} \]

\[ \sum \rightarrow f \]

\[ \sum \rightarrow f \]

\[ \sum \rightarrow f \]
Multi-layer Perceptron

\[ x_1 \]

\[ w_{11} \]
\[ w_{10} \]
\[ w_{21} \]
\[ w_{20} \]
\[ w_{31} \]
\[ w_{30} \]

\[ \sum \rightarrow f \]

\[ \sum \rightarrow f \]

\[ \sum \rightarrow f \]

\[ x_1 w_{11} + w_{10} \]
\[ x_1 w_{21} + w_{20} \]
\[ x_1 w_{31} + w_{30} \]
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

\[ y_1 = f(x_1 w_{11} + w_{10}) \]
\[ y_2 = f(x_1 w_{21} + w_{20}) \]
\[ y_3 = f(x_1 w_{31} + w_{30}) \]
Multi-layer Perceptron

\[ y_1 = f(x_1 (w_{11} + w_{10})) \]
\[ y_2 = f(x_1 (w_{21} + w_{20})) \]
\[ y_3 = f(x_1 (w_{31} + w_{30})) \]
Multi-layer Perceptron

\[
y_1 = f(x_1 w_{11} + w_{10}) \\
y_2 = f(x_1 w_{21} + w_{20}) \\
y_3 = f(x_1 w_{31} + w_{30})
\]
Multi-layer Perceptron

Input features

\[ x_1 \]

Hidden layers

\[ \sum \rightarrow f \]

Output layers

\[ \sum \rightarrow \text{Output} \]

\[ y_1 = f(x_1 (w_{11} + w_{10})) \]
\[ y_2 = f(x_1 (w_{21} + w_{20})) \]
\[ y_3 = f(x_1 (w_{31} + w_{30})) \]
Multi-layer Perceptron

Input features

Hidden layers

Output layers

\[ y_1 = f(x_1 w_{11} + w_{10}) \]
\[ y_2 = f(x_1 w_{21} + w_{20}) \]
\[ y_3 = f(x_1 w_{31} + w_{30}) \]
Multi-layer Perceptron

"Features" are outputs of perceptrons

Matrix of first layer weights

\[
\begin{pmatrix}
  w_{11} & w_{10} \\
  w_{21} & w_{20} \\
  w_{31} & w_{30}
\end{pmatrix}
\]

Matrix of second layer weights

\[
\begin{pmatrix}
  w_1 \\
  w_2 \\
  w_3
\end{pmatrix}
\]
Multi-layer Perceptron

Input features

Hidden layers

Output layers

Matrix of first layer weights

Matrix of second layer weights

"Features" are outputs of perceptrons
Multi-layer Perceptron

Input features

Hidden layers

Output layers

Perceptrons

Matrix of first layer weights

Matrix of second layer weights

"Features" are outputs of perceptrons

\[
\begin{align*}
\sum & \rightarrow f & \sum & \rightarrow f \\
\sum & \rightarrow f
\end{align*}
\]

\[
\begin{pmatrix}
w_{11} & w_{10} \\
w_{21} & w_{20} \\
w_{31} & w_{30}
\end{pmatrix}
\]

\[
\begin{pmatrix}
w_1 \\
w_2 \\
w_3
\end{pmatrix}
\]
Features of MLPs

Input features

Perceptron: Step function with linear decision boundary
Features of MLPs

Layer 1

2-layer:
These outputs are now input features to the next layer
Features of MLPs

2-layer:
These outputs are now input features to the next layer
"Features" are now decision boundaries (partitions)
Features of MLPs

2-layer:
- These outputs are now input features to the next layer
- "Features" are now decision boundaries (partitions)
- All linear combination of those partitions give complex partitions
Features of MLPs

Layer 1

Layer 2

These complex outputs become the features for the new layer.
Features of MLPs

Deep Neural Networks

Layer 1

Layer 2
Computational Graph representation of Neural Networks
Neural Networks

Fully connected layers

\[ W_1 \]

\[ x_1 \]

\[ N \times N \quad N \times 1 \]
Neural Networks

Fully connected layers

\[ W_1 \]

\[ x_1 \]

\[ x_2 \]

\[ N \times N \]

\[ N \times 1 \]

\[ N \times 1 \]
Neural Networks

Fully connected layers

$W_1$

$x_1$

$N \times N$

$N \times 1$

ReLU

0

$W_2$

$x_2$

$N \times N$

$N \times 1$

ReLU

...
Neural Networks

$W_1 \xrightarrow{\text{ReLU}} W_2 \xrightarrow{\text{ReLU}} \ldots$

Data

$\mathbf{x}_1 \rightarrow W_1 \rightarrow \mathbf{x}_2 \rightarrow W_2 \rightarrow \mathbf{x}_3 \rightarrow \ldots$

$N \times N \quad N \times 1 \quad N \times N \quad N \times 1$

$\mathbf{x}_1$ represents data with $N \times N$ pixels.

$\mathbf{x}_2$ represents the output of the hidden layer with $N \times 1$ dimensions.

$\mathbf{x}_3$ represents the output of the last layer with $N \times 1$ dimensions.

$N$ represents the number of pixels in an image.
Neural Networks

Unstructured data → Fully connected layers

Computational Graph

ReLU

$W_1 \rightarrow x_1 \rightarrow W_2 \rightarrow x_2 \rightarrow \ldots$

ReLU

$W_1$ $W_2$

ReLU

ReLU

ReLU
Neural Networks

Unstructured data

Fully connected layers

Computational Graph

Realistic Image Synthesis SS2021
Two-layer model

Fully connected layers

What can be a loss function?
Two-layer model

Fully connected layers

What can be a loss function?
Two-layer model

Fully connected layers

What can be a loss function?
What can be a loss function?
What can be a loss function?
Two-layer model

What can be a loss function?
Two-layer model: Back propagation
Two-layer model: Back propagation

Gradient Descent Algorithm for back propagation
Two-layer model: Back propagation

Gradient Descent Algorithm for back propagation

Random initialization
Two-layer model: Back propagation

Gradient Descent Algorithm for back propagation

Random initialization
Two-layer model: Back propagation

Gradient Descent Algorithm for back propagation

Random initialization

Global cost minimum
Back Propagation

\[ f(x, y, z) \]
Back Propagation

\[ f \]

\[ \frac{\partial z}{\partial x}, \frac{\partial z}{\partial y} \]

"local gradients"
Back Propagation

\[ f(x, y, z) \]

\[ \frac{\partial z}{\partial x}, \frac{\partial z}{\partial y} \]

"local gradients"

\[ \frac{\partial L}{\partial z} \]

gradients
Back Propagation

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}
\]

"local gradients"

\[
\frac{\partial z}{\partial x} \quad \frac{\partial z}{\partial y}
\]

\[
\frac{\partial L}{\partial z}
\]

gradients
Back Propagation

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} \\
\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y}
\]

"local gradients"
Back Propagation

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}
\]

\[
\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y}
\]

"local gradients"
Back Propagation

\[ \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} \]

\[ \frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y} \]

“local gradients”

\[ f \]

\[ \frac{\partial L}{\partial z} \]

gradients
Back Propagation

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

\[ \text{e.g. } x = -2, y = 5, z = -4 \]

Slides courtesy: Stanford Online Course
Back Propagation

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz, \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} \]

Slides courtesy: Stanford Online Course
Machine Learning for Filtering Monte Carlo Noise

Kalantari et al. [SIGGRAPH 2015]
Reconstruction / Denoising

\[ \hat{c}_i = \frac{\sum_{j \in N(i)} d_{i,j} \bar{c}_j}{\sum_{j \in N(i)} d_{i,j}} \quad , \quad \hat{c} = \{\hat{c}_r, \hat{c}_g, \hat{c}_b\} \]
Reconstruction / Denoising

\[ \hat{c}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{c}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}} \quad , \quad \hat{c} = \{\hat{c}_r, \hat{c}_g, \hat{c}_b\} \]
Reconstruction / Denoising

\[ \hat{c}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{c}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}} \]

\[ \hat{c} = \{ \hat{c}_r, \hat{c}_g, \hat{c}_b \} \]

Pixel neighborhood
Reconstruction / Denoising

\[ \hat{c}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{c}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}} \], \quad \hat{c} = \{ \hat{c}_r, \hat{c}_g, \hat{c}_b \} 

Pixel neighborhood
Reconstruction / Denoising

\[ \hat{c}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{c}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}} , \quad \hat{\mathbf{c}} = \{ \hat{c}_r, \hat{c}_g, \hat{c}_b \} \]

Pixel neighborhood

Filter weights
Filter weights

\[ \hat{c}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{c}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}} \]

Pixel neighborhood
Filter weights

\[
\hat{c}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{c}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}}
\]

For cross Bilateral filters:

Pixel neighborhood
Filter weights

\[
\hat{c}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{c}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}}
\]

Pixel neighborhood

For cross Bilateral filters:

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

\[ \hat{c}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{c}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}} \]

For cross Bilateral filters:

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

Filter weights

For cross Bilateral filters:

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

Sen and Darabi [2012]
Filter weights

For cross Bilateral filters:

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

Sen and Darabi [2012]
Filter weights

For cross Bilateral filters:

\[ d_{i,j} = \exp\left[-\frac{||\bar{p}_i - \bar{p}_j||^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{c}_i, \bar{c}_j)}{2\beta_i^2}\right] \]

\[ \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{f}_i,k, \bar{f}_j,k)}{2\gamma_{k,i}^2}\right], \]
Filter weights

For cross Bilateral filters:

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

Pixel screen coordinates
Filter weights

For cross Bilateral filters:

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

Pixel screen coordinates

Mean sample color value

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

83 (f) Sample color
Filter weights

For cross Bilateral filters:

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

- 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
Filter weights

For cross Bilateral filters:

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

For cross Bilateral filters:

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

What are the optimal parameters?
Neural Network Approach

- Feed-forward Neural network
- Best part: We can learn weights in a training phase
- Back propagation: Important for training weights
- For Back propagation, the Loss function should be differentiable and all the intermediate functionals should be differentiable.
One Hidden-layer model

Relative Mean Square Error:

$$E_i = \frac{n}{2} \sum_{q \in \{r, g, b\}} \frac{(\hat{c}_{i,q} - c_{i,q})^2}{c_{i,q}^2 + \varepsilon}$$
One Hidden-layer model

Relative Mean Square Error:

\[ E_i = \frac{n}{2} \sum_{q \in \{r, g, b\}} \frac{(\hat{c}_{i,q} - c_{i,q})^2}{c_{i,q}^2 + \varepsilon} \]

\[ \frac{\partial E_i}{\partial w_{t,s}^l} = \sum_{m=1}^{M} \left[ \sum_{q \in \{r, g, b\}} \left[ \frac{\partial E_{i,q}}{\partial \hat{c}_{i,q}} \frac{\partial \hat{c}_{i,q}}{\partial \theta_{m,i}} \right] \frac{\partial \theta_{m,i}}{\partial w_{t,s}^l} \right] \]

\[ \frac{\partial E_i}{\partial \hat{c}_{i,q}} = ??? \]
One Hidden-layer model

Relative Mean Square Error:

\[
E_i = \frac{n}{2} \sum_{q \in \{r, g, b\}} \frac{(\hat{c}_{i,q} - c_{i,q})^2}{c_{i,q}^2 + \varepsilon}
\]

\[
\frac{\partial E_i}{\partial w_{t,s}^l} = \sum_{m=1}^{M} \left[ \sum_{q \in \{r, g, b\}} \left( \frac{\partial E_{i,q}}{\partial \hat{c}_{i,q}} \frac{\partial \hat{c}_{i,q}}{\partial \theta_{m,i}} \right) \frac{\partial \theta_{m,i}}{\partial w_{t,s}^l} \right]
\]

\[
\frac{\partial E_i}{\partial \hat{c}_{i,q}} = n \frac{\hat{c}_{i,q} - c_{i,q}}{c_{i,q}^2 + \varepsilon}
\]
Results
Introduction to CNNs
Introduction to CNNs

Kernel Predicting Denoising
Introduction to CNNs

Kernel Predicting Denoising

Sample-based MC Denoising
Convolution

No zero padding
Convolution

No zero padding
Stride-1 Convolution

No zero padding
Stride-1 Convolution

No zero padding
Stride-1 Convolution

No zero padding
Stride-1 Convolution

No zero padding
Stride-1 Convolution

No zero padding
Stride-1 Convolution

No zero padding
Stride-2 Convolution

Stride-2 Convolution
Stride-2 Convolution

The figure shows a 3D representation of a stride-2 convolution operation. The input and output feature maps are visualized with 0s and 1s, illustrating the kernel movement and resulting activations.

Stride-2 convolution is a technique used in convolutional neural networks to reduce the spatial size of the input while increasing the computational efficiency.
Zero Padding and Strides

1D image to illustrate the strides and zero padding

Stride 1

zero padding
Zero Padding and Strides

1D image to illustrate the strides and zero padding

Stride 1

zero padding
Strides

1D image to illustrate the strides and zero padding
Max Pooling / Down Sampling
Overview on Convolutional Neural Networks

Image Courtesy: Mathworks (online tutorial)
Multi-layer Perceptron vs. CNNs
Multi-layer Perceptron vs. CNNs

Multi-layer perceptron

- All nodes are fully connected in all layers
- In theory, should be able to achieve good quality results in small number of layers.
- Number of weights to be learnt are very high

CNNs

- Weights are shared across layers
- Requires significant number of layers to capture all the features (e.g. Deep CNNs)
- Relatively small number of weights required
Introduction to CNNs

Kernel-Predicting Denoising
Kernel-Predicting Networks for Denoising Monte-Carlo Renderings

Bako et al. [2017]
Limitations of MLP based Denoiser

Kernel was pre-selected to be joint bilateral filter
Limitations of MLP based Denoiser

Kernel was pre-selected to be joint bilateral filter

- Unable to explicitly capture all details
Limitations of MLP based Denoiser

Kernel was pre-selected to be joint bilateral filter

- Unable to explicitly capture all details
- Lacked flexibility to handle wide range of MC noise in production scenes
Limitations of MLP based Denoiser

Kernel was pre-selected to be joint bilateral filter

- Unable to explicitly capture all details
- Lacked flexibility to handle wide range of MC noise in production scenes

Fixed
Limitations of MLP based Denoiser

Kernel was pre-selected to be joint bilateral filter

- Unable to explicitly capture all details
- lacked flexibility to handle wide range of MC noise in production scenes

Fixed
- can cause unstable weights causing bright ringing and color artifacts
Limitations of MLP based Denoiser

Kernel was pre-selected to be joint bilateral filter

- Unable to explicitly capture all details
- lacked flexibility to handle wide range of MC noise in production scenes

Fixed

- can cause unstable weights causing bright ringing and color artifacts

Too many parameters to optimize
Requirements

The function must be flexible to capture complex relationship between input data and reference colors over wide range of scenarios.

Choice of loss function is crucial. Should capture perceptual aspects of the scene.

To avoid overfitting, large dataset required
Using a Vanilla CNN

Denoising a raw, noisy color buffer causes overblurring
Using a Vanilla CNN

Denoising a raw, noisy color buffer causes overblurring

- difficulty in distinguishing scene details and MC noise
Using a Vanilla CNN

Denoising a raw, noisy color buffer causes overblurring

- difficulty in distinguishing scene details and MC noise

High dynamic range
Using a Vanilla CNN

Denoising a raw, noisy color buffer causes overblurring

- difficulty in distinguishing scene details and MC noise

High dynamic range

- can cause unstable weights causing bright ringing and color artifacts
Vanilla CNN
Denoising Model

\[ \hat{\theta}_p = \underset{\theta}{\text{argmin}} \ell(\tilde{c}_p, g(X_p; \theta)) \]

Reference image

\[ \hat{c}_p = g(X_p; \hat{\theta}_p) \]

Denoised value

\[ \ell(\bar{c}, \hat{c}) \]

Loss function

Denoised function with parameters
Computational Model

\[ \hat{\theta}_p = \arg \min_\theta \sum_{q \in \mathcal{N}(p)} \left( c_q - \theta^T \phi(x_q) \right)^2 \omega(x_p, x_q) \]

Neighborhood

\[ \hat{c}_p = g(X_p; \hat{\theta}_p) \]

Denoised value

\[ \hat{c}_p = \hat{\theta}_p^T \phi(x_p) \]

Final denoised value

\[ \phi : \mathbb{R}^{3+D} \rightarrow \mathbb{R}^M \]

Kernel weights

\[ \omega(x_p, x_q) \]
Direct Prediction Network

Direct prediction convolution network: outputs denoised image

\[ \hat{c}_p = g_{\text{direct}}(X_p; \theta) = z_p^L \]
Direct Prediction Network

Direct prediction convolution network: outputs denoised image

$$\hat{c}_p = g_{\text{direct}}(X_p; \theta) = z_p^L$$

Issues:

The constrained nature and complexity of the problem makes optimization difficult.

The magnitude and variance of stochastic gradients computed during training can be large, which slows convergence of training loss.
Kernel Prediction Network

Kernel prediction convolution network: outputs learned kernel weights

\[ w_{pq} = \frac{\exp([z_p^L]_q)}{\sum_{q' \in N(p)} \exp([z_p^L]_{q'})} \]
\[ 0 \leq w_{pq} \leq 1 \]

Softmax activation to enforce weights within range

Denoised color values:

\[ \hat{c}_p = g_{\text{weighted}}(X_p; \theta) = \sum_{q \in N(p)} c_q w_{pq} \]
Kernel Prediction Network

\[ w_{pq} = \frac{\exp([z^L_p]_q)}{\sum_{q' \in N(p)} \exp([z^L_p]_{q'})} \]

\[ 0 \leq w_{pq} \leq 1 \]

\[ \hat{c}_p = g_{\text{weighted}}(X_p; \theta) = \sum_{q \in N(p)} c_q w_{pq} \]

Final color estimate always lies within the convex hull of the respective neighborhood (avoid color shifts).

Ensures well-behaved gradients of the error w.r.t the kernel weights.
Proposed Architecture
Diffuse/Specular components

Each component is denoised separately

Diffuse components are well-behaved and typically has small ranges

- albedo is factored out to allow large range kernels

\[ \tilde{c}_{\text{diffuse}} = c_{\text{diffuse}} \odot (f_{\text{albedo}} + \epsilon) \]

Specular components are challenging due to high dynamic ranges: uses logarithmic transform

\[ \tilde{c}_{\text{specular}} = \log(1 + c_{\text{specular}}) \]
Training Dataset: 600 frames
Training

8-hidden layers used with 100 kernels of 5x5 in each layer for each network

For KPCN (kernel-predicting network), output kernel size used = 21

Weights for 128 app and 32 spp networks were initialized using Xavier method

Diffuse and specular components were independently trained with L1 loss metric
Learning rate of DPCN vs. KPCN

On Cars 3 dataset, KPCN converges 5-6x faster
Results

relative $\ell_2$
1 – SSIM

Input (128 spp)

NFOR (log)

Ours

Ref. (32K spp)

29.15e-3
0.562
0.90e-3
0.019
0.69e-3
0.017

38.57e-3
0.552
1.12e-3
0.025
0.92e-3
0.024

121
Ablation study

Input (32 spp)
Ablation study

Input (32 spp)
Ablation study

Input (32 spp)

w/o Decomposition, w/o Albedo divide
Ablation study

w/o Decomposition, w/o Albedo divide
w/ Decomposition, w/o Albedo divide
w/o Decomposition, w/ Albedo divide
w/ Decomposition, w/ Albedo divide
Ref. (2K spp)
Ablation study

Also works on Piper short movie frames
Interactive Reconstruction of Monte Carlo Sequences

Chaitanya et al. [2017]
Motivation: Interactive Reconstruction

Limited to a few rays per pixel @ 1080p @ 30Hz

Never enough to reconstruct an image

Deep learning approach for interactive graphics
Motivation: Interactive Reconstruction

Limited to a few rays per pixel @ 1080p @ 30Hz

Never enough to reconstruct an image

Deep learning approach for interactive graphics
Problem Statement

Handle generic effects:

- Soft shadows
- Diffuse and specular reflections
- Global illumination (one-bounce)
- No Motion blur or depth of field
System setup: Path tracing
System setup: Path tracing
System setup: Path tracing
System setup: Path tracing

- Rasterize primary hits in G-buffers
- Path-tracing from the primary paths
  - 1 ray for direct shadows
  - 2 rays for indirect (sample + connect)

1 direct + 1 indirect path (spp)
Denoising Autoencoder (DAE)

Train auto encoders to reconstruct image from 1spp
Recurrent Autoencoder

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 $2 \times 2$ max pooling. A decoder stage applies a $2 \times 2$ 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 $3 \times 3$-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.

[Chaitanya et al. 2017]
Recurrent Neural Networks

Encoder and decoder stages for dimensionality reduction
Recurrent Neural Networks

Encoder and decoder stages for dimensionality reduction
Recurrent Neural Networks

Encoder and decoder stages for dimensionality reduction

Encoder

Decoder
Recurrent Neural Networks

Encoder and decoder stages for dimensionality reduction

Skip connections to reintroduce lost information
Auxillary Features

Untextured color

View space normals

Linearize depth
Training sequences

SponzaDiffuse  SponzaGlossy  Classroom
Training sequences

SponzaDiffuse  SponzaGlossy  Classroom
1spp approx. 70 ms
DAE 1spp
approx. 70 ms + approx. 60 ms
Recurrent Denoising Autoencoder

Feedback loops to retain important information after every encoding stage
Recurrent Denoising Autoencoder

Feedback loops to retain important information after every encoding stage
Recurrent Neural Networks vs. Simple Feed-Forward NN

Recurrent Neural Network

Feed-Forward Neural Network

Source link
Recurrent Neural Networks

An unrolled recurrent neural network.
Recurrent Neural Networks

Fully convolutional blocks to support arbitrary image resolution

6 RNN blocks, one per pool layer in the encoder

Design:
- 1 conv layer (3x3) for current features
- 2 conv layers (3x3) for previous features
Recurrent Neural Networks
Recurrent Neural Networks
Recurrent Neural Networks
Recurrent Neural Networks

CNNs, fixed input, fixed output

one to one
Recurrent Neural Networks

CNNs, fixed input, fixed output

one to one
Recurrent Neural Networks

CNNs, fixed input, fixed output

one to one | one to many
Recurrent Neural Networks

CNNs, fixed input, fixed output

<table>
<thead>
<tr>
<th>one to one</th>
<th>one to many</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="Diagram" /></td>
<td><img src="image.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

e.g., image captioning takes an image as input and outputs a sentence of words
Recurrent Neural Networks

CNNs, fixed input, fixed output

Sequence input

one to one  

one to many

many to one

Sequence output

e.g., to know the sentiments of a sentence
Recurrent Neural Networks

CNNs, fixed input, fixed output

Sequence input

one to one

one to many

many to one

many to many

Sequence output

Sequence input, Sequence output.

E.g. Machine translation
Recurrent Neural Networks

- CNNs, fixed input, fixed output
- Sequence input
- Synced sequence input & output
- Sequence output

-one to one
-one to many
-many to one
-many to many

Sequence input, Sequence output.

- e.g. Machine translation
- e.g., video classification where we want to label each frame
Training

Input is a sequence of 7 frames

128x128 random image crop per sequence

Play the sequence forward/backward

Each frame advance the camera or random seed
Loss Functions

Spatial Loss to emphasize more the dark regions

\[ L_s = \frac{1}{N} \sum_{i}^{N} |P_i - T_i| \]
Loss Functions

Spatial Loss to emphasize more the dark regions

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

Temporal loss

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

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

\[ L = w_s L_s + w_g L_g + w_t L_t \]
Training Loss depends on Auxiliary Features

Auxiliary Features

Epochs

Training loss

- Color only
- Untextured color
- Untextured + depth
- Untextured + normal
- Untextured + normal + depth
- Untextured + normal + depth + roughness
Temporal Stability
Recurrent autoencoder with temporal AA

Recurrent autoencoder

Autoencoder with skips
Recurrent autoencoder

1 sample/pixel input
Recurrent autoencoder

1 sample/pixel input
Introduction to CNNs

Kernel Predicting Denoising

Sample-based MC Denoising (next lecture)
Acknowledgments

Thanks to Chaitanya et al. for making their slides publicly available.