Rise of the Machines in MC Integration for Rendering

Toshiya Hachisuka

The University of Tokyo
This talk is NOT about
but about this

https://www.tensorflow.org/versions/r0.10/tutorials/image_recognition/index.html
Deep (learning) impact

- Significant step on classification tasks in 2012
- 10% improvement (where 1% was typical)
- Better than a “man-made” classifier

ImageNet [Deng et al.09]
Before deep learning

Feature Detector
Before deep learning
Before deep learning

Feature Detector → Classifier → dog
Before deep learning

Manually designed

Feature Detector

Classifier

e.g., “Gradients should be useful to classify images.”
After deep learning
After deep learning
After deep learning
After deep learning

Learned from examples

Machine Learning

dog

dog
Successes of deep learning

• Achieving significant results in many applications
  • Image segmentation [Long et al.14]
  • Image captioning (many groups in 2014)
  • Playing video games [Mnih et al.13]
  • Lip reading [Ngiam et al.11]
• and more…
How can we utilize deep learning in MC?
Deep learning in MC
Deep learning in MC
Deep learning in MC

course samples
Deep learning in MC

coarse samples

functions

Machine Learning
Deep learning in MC

Machine Learning

“optimal” estimator

coarse samples

functions
Rise of the Machines

• Learn the “optimal” estimator for a given function
  • Uses coarse samples as “images”
  • Assume a set of specific (non-arbitrary) functions
  • Automatically exploit “hidden” structures
  • Similar to adaptive Monte Carlo, but we use machine learning to decide how to adapt
Automatic Mixing of MC Integrators

joint work with S. Kinuwaki and H. Otsu (in submission)
Light transport simulation

• Solution of the following governing equation

\[ L(x, \vec{\omega}) = L_e(x, \vec{\omega}) + \int_\Omega f_r(x, \vec{\omega}, \vec{\sigma})L(x, \vec{\omega})(\vec{\omega} \cdot \vec{n})d\vec{\omega} \]

• Can be formulated as a simple MC integration

• Different algorithms coverage to the same results
Varying efficiency

- Efficiency of each algorithm varies for different inputs

Algorithm 1

Algorithm 2
Varying efficiency

- Efficiency of each algorithm varies for different inputs

Algorithm 2 is more efficient
Varying efficiency

- Efficiency of each algorithm varies for different inputs
Varying efficiency

- Efficiency of each algorithm varies for different inputs

Algorithm 1 is more efficient
Mixing different algorithms

- Combine the results by a weighted sum
  - More efficient algorithm should have larger weight
  - Often called “blending” in graphics

\[ w \cdot \text{Algorithm 1} + (1 - w) \cdot \text{Algorithm 2} \]
Pixel-wise blending

- Each pixel has its own blending weight
Key idea

Machine Learning

"optimal" estimator

coarse samples

functions
Key idea

Machine Learning

"optimal" weight

alg.1

alg.2

rendered images

examples
Method
Two algorithms

- Stochastic progressive photon mapping (SPPM) [Hachisuka & Jensen 2009]
- Manifold exploration (MLT) [Jacob & Marschner 2012]
Learning

References

SPPM

MLT
Learning

References

SPPM

MLT

Relative Contrib.

Optimal weight

...
Learning

References

SPPM

MLT

Relative Contrib.

Optimal weight

...
Learning

References

SPPM

MLT

Relative Contrib.

Optimal weight

Regression forest
Regression forests

- Machine learning technique which uses an ensemble of randomized regression trees

Body segmentation
[Shotton et al. 2011]

Fluid simulation
[Ladický et al. 2015]
Runtime

Regression forest
Runtime

SPPM

MLT

Regression forest
Runtime

SPPM

MLT

Relative Contrib.

Regression forest
Runtime

SPPM

MLT

Relative Contrib.

Weights

Regression forest
Runtime

SPPM

MLT

Relative Contrib.

Weights

Regression forest

Result
Relative contributions
Relative contributions

\[ I = I_1 + I_2 + I_3 + I_4 + \cdots \]
Relative contributions

• Vector of relative intensities of each integral

$$\vec{\phi}(I) = \left( \frac{I_1, I_2, I_3, I_4, \ldots}{I} \right)$$

• Similar to relative magnitudes of subsets of samples

• Samples are clustered by some fixed rules
Optimal weights

- Given pixel intensities from two algorithm ($\alpha$ and $\beta$), the optimal weight (minimizes error) is defined as

$$w_{opt} = \arg \min_w \left| \left( w\hat{I}_\alpha + (1 - w)\hat{I}_\beta \right) - I \right|$$
Optimal weights

• Given pixel intensities from two algorithm ($\alpha$ and $\beta$), the optimal weight (minimizes error) is defined as

$$w_{opt} = \arg\min_w \left| \left( w\hat{I}_\alpha + (1 - w)\hat{I}_\beta \right) - I \right|$$

blended result  reference
Optimal weights

- Given pixel intensities from two algorithm (α and β), the optimal weight (minimizes error) is defined as

\[
    w_{\text{opt}} = \arg \min_{w} \left| \left( w\hat{I}_\alpha + (1 - w)\hat{I}_\beta \right) - I \right|
\]

\[
    w_{\text{opt}} \approx w_{\text{reg}}(\vec{\phi}(\hat{I}_\alpha), \vec{\phi}(\hat{I}_\alpha))
\]

what we learn
Optimal weights

- Given pixel intensities from two algorithm (\(\alpha\) and \(\beta\)), the optimal weight (minimizes error) is defined as

\[
\omega_{\text{opt}} = \arg \min_w \left| \left( w \hat{I}_\alpha + (1 - w) \hat{I}_\beta \right) - I \right|
\]

\[
\omega_{\text{opt}} \approx \omega_{\text{reg}}(\vec{\phi}(\hat{I}_\alpha), \vec{\phi}(\hat{I}_\alpha))
\]

what we learn

note: \(\omega_{\text{opt}}(\hat{I}_\alpha, \hat{I}_\beta, I) \approx \omega_{\text{reg}}(\hat{I}_\alpha, \hat{I}_\beta) \quad \lim \hat{I}_\alpha = \lim \hat{I}_\beta = I\)
Results
Training scenes
Evaluation

One scene for testing

Other scenes for training
Evaluation

One scene for testing

Other scenes for training
Scene: babylonian-city
Weights (Optimal)
Weights (Ours)

<table>
<thead>
<tr>
<th>SPPM</th>
<th>MLT</th>
</tr>
</thead>
</table>

Weights (Optimal)
Weights (Ours)
Automatic mixing of MC

- Tries to optimally mix two Monte Carlo integrators
  - Learn optimal weights by examples
  - Better results than the baseline

- The same approach can be used for any other problems where we have multiple MC integrators
  - No modification to MC integrators themselves
Auto-adaptive MCMC

joint work with H. Otsu, M. Šik, and J. Křivánek (in progress)
Idea

- Adapt proposal distributions according to pre-trained data for various functions

Machine Learning

“optimal” estimator

coarse samples
Idea

- Adapt proposal distributions according to pre-trained data for various functions
Auto-adaptive MCMC

• Learning
  • Given coarse samples + current state, maximize the expected jumping distance
  • Learn the relationship with the proposal distribution

• Runtime
  • Given coarse samples + current state, output the parameters of the proposal distribution
Auto-adaptive MCMC

• **Learning**
  • Given coarse samples + current state, maximize the expected jumping distance
  • Learn the relationship with the proposal distribution

• **Runtime**
  • Given coarse samples + current state, output the parameters of the proposal distribution

Adaptively maximize the expected jumping distance
Early experiments

- Samples from a mixture of two non-gaussians
- Radial basis functions for learning
- Instead of coarse samples, used parameters of a mixture distribution (still contains complex relations)
Histogram

- Baseline
Histogram

- Our machine-learned MCMC
Trace plot

- Baseline
Trace plot

- Our machine-learned MCMC
Expected jumping distance

- Numerically maximized at each point
Expected jumping distance

- Our machine-learned MCMC
Conclusions
Conclusions

• Two general frameworks on how to utilize machine learning to improve MC/MCMC estimators
  
  • Auto-mixing MC
  
  • Auto-adaptive MCMC
  
• Same key idea
  
  • Consider samples as “images”, then learns the relationship between them and optimal estimators