Visual AI

CPSC 532R/533R – 2019/2020 Term 2

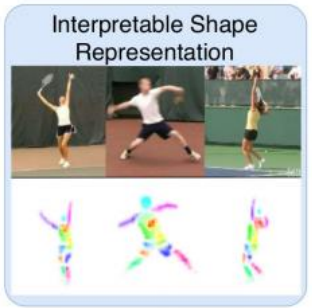
Lecture 9. Sequential decision making

Helge Rhodin

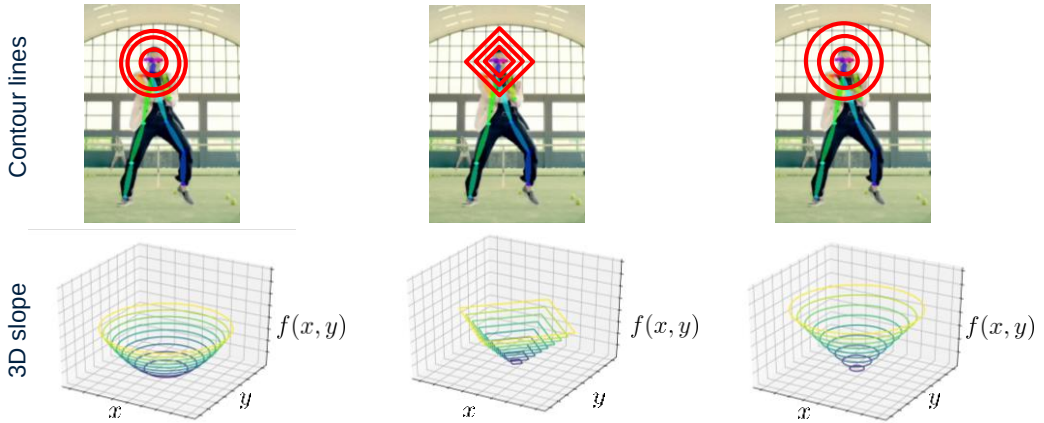
Assignment 2, highlights

Second order moments to restrict size!

```
[41]: def integral_heatmap_layer(dict):
    # compute coordinate matrix
    heatmap = dict['heatmap']

    bias = 1 # 2
    hm_exp = torch.exp(heatmap * bias)
    logsum = torch.sum(torch.sum(hm_exp,
    logsum = logsum.unsqueeze(2).unsquee
    h_norm = hm_exp / (logsum * bias)

    h = heatmap.size()[2]
    w = heatmap.size()[3]

    grid = torch.stack(
        torch.meshgrid(
            torch.linspace(0, 1, h),
            torch.linspace(0, 1, w)
        ),
        dim=2
    )
    grid = torch.unsqueeze(grid, 0)
    grid = torch.unsqueeze(grid, 0)
    grid = grid.to(heatmap.device)

    prods = grid * torch.unsqueeze(h_norm, 4)
    pose = torch.sum(torch.sum(prods, dim=3), dim=2)
    perm = torch.LongTensor([1, 0])
    pose = pose[:,:,perm]

    ##########################################################
    # NOTE: I have experimented with adding a "moment" loss #
    # term which penalizes the network for predicting heat- #
    # maps that are spread-out. Its computation is below    #
    # and is inspired by the moment of inertia of a rigid   #
    # body, which increases as mass is distributed further  #
    # away from the center point.                           #
    ##########################################################
    centers = pose.unsqueeze(2).unsqueeze(2)
    diffs = grid - centers
    sqr_dists = torch.sum(diffs**2, dim=-1)
    sqr_dists_prods = sqr_dists * h_norm
    moments = torch.sum(torch.sum(sqr_dists_prods, dim=-1), dim=-1) / (w * h)

    return DeviceDict({
        'probabilitymap': h_norm,
        'pose_2d': pose,
        'moments': moments
    })
```



L1 loss



Squared error
MSE

Absolute error
MAE

Euclidean distance
MPJPE

Hint on PyTorch data loader

```
torch.utils.data.DataLoader(
  dataset,                # dataset from which to load the data.
  batch_size=1,           # how many samples per batch to load
  shuffle=False,          # set to True to have the data reshuffled at every epoch
  sampler=None,           # defines the strategy to draw samples from the dataset
  batch_sampler=None,     # like sampler, but returns a batch of indices at a time
  num_workers=0,          # how many subprocesses to use for data loading, 0 means using the main process
  collate_fn=None,        # merges a list of samples to form a mini-batch of Tensor(s)
  pin_memory=False,       # if True, the data loader will copy Tensors into CUDA pinned CPU memory
  drop_last=False,        # drop the last incomplete batch, if the size is not divisible by the batch size
  timeout=0,              # if positive, the timeout value for collecting a batch from workers
  worker_init_fn=None, multiprocessing_context=None # threading stuff
)
```

Make sure that you understand all arguments!

Assignment 3

- Rendering
- Learning shape spaces
- Interpolating in shape spaces

- Work independently, don't cheat!
  - disciplinary measures will be reported on your transcripts
    - your future applications may be rejected because of this

Assignment 3: Neural Rendering and Shape Processing

CPSC 532R/533R Visual AI
by Helge Rhodin and Yuchi Zhang

This assignment is on neural rendering and shape processing—computer graphics. We provide you with a dataset of 2D icons and corresponding vector graphics as shown in Figure 1. It stems from a line of work on translating low-resolution icons to visually appealing vector forms and was kindly provided by Sheffer et al. [1] for the purpose of this assignment.

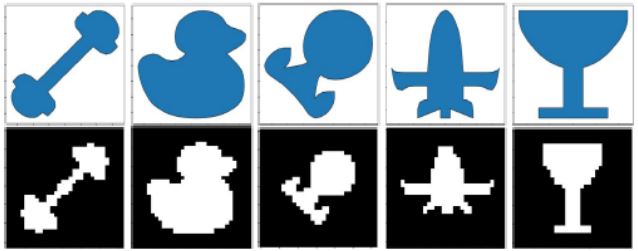
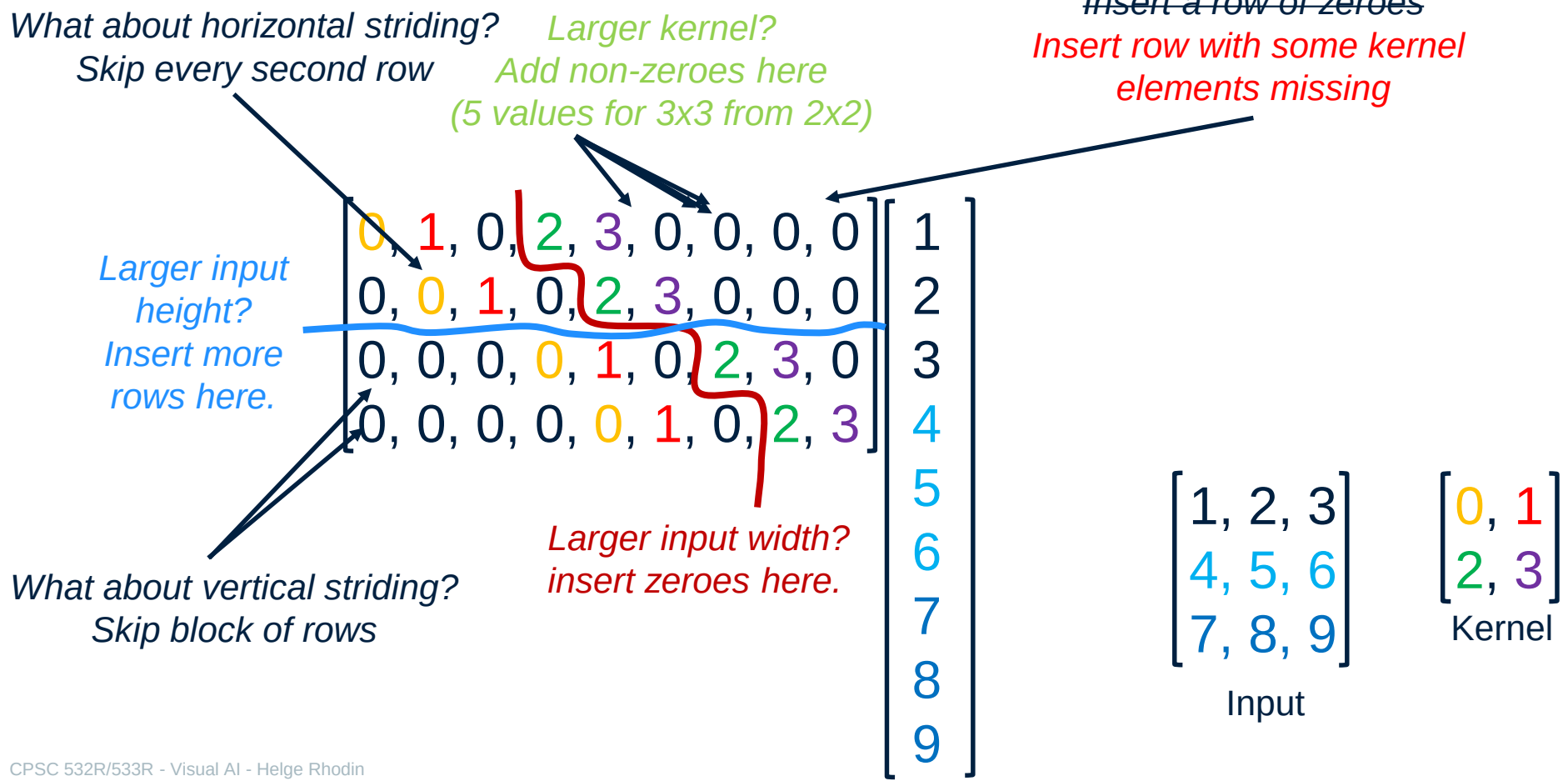


Figure 1: Icon vector graphics and their bitmap representation.

The overall goal of this assignment is to find transformation between icons. We provide the `ImagerIcon` dataset as an HDF5 file. As usual, the `Assignment3_TaskI.ipynb` notebook provides dataloading, training and validation splits, as well as display and training functionality. Compatibility of the developed neural networks with color images is ensured by storing the contained 32 x 32 icon bitmaps as $3 \times W \times H$ tensors. Vector graphics are represented as polygons with $N = 96$ vertices and are stored as $2 \times N$ tensors, with neighboring points stored sequentially. The polygon representation with a fixed number of vertices was attained by subsampling the originally curved vector graphics.

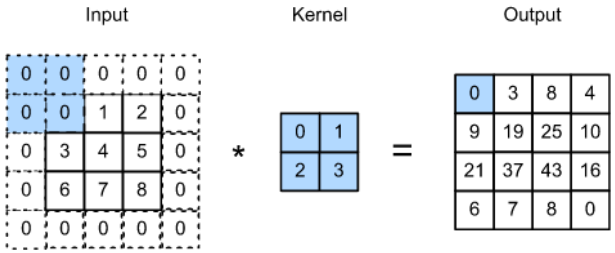
Convolution as matrix multiplication (details)



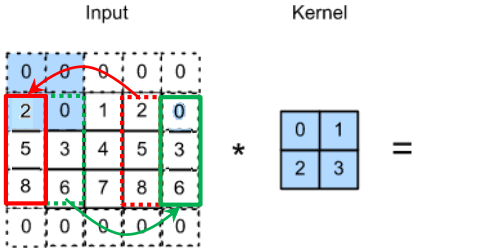
Convolution with cyclic padding, theory

Convolution

- instead of inserting zeroes, copy values from the opposing side of the image



zero padding



cyclic padding

Cyclic padding, broken in PyTorch

- Setting padding equal to zero or one seems to have the same effect
  - in both cases the output resolution is reduced, as without padding
- Experienced instability when using cyclic padding
- Only tested with 1D convolutions

- Don’t use it as of now!

Project ad

Killer whale identification

Andrew W Trites

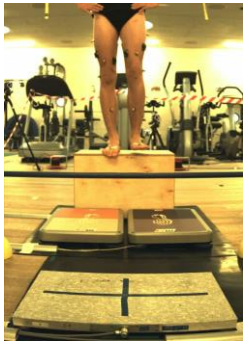
Professor and Director

Institute for the Oceans and Fisheries UBC

Final dataset: 300 images, 40 different whales
Sufficient to distinguish ecotypes: transient and residential orcas

Final data available. Drop me a mail if you would like to inspect it.

mm-accurate 3D pose and force estimation



Dr. Jörg Spörri

Sport medicine head

University Hospital Balgrist

Final dataset: 765 jump videos, 2D and 3D pose, pressure plate measurement, cam. calibration.

Overview

- 11 Lectures (Weeks 1 – 6)
  - Introduction
  - Deep learning basics and best practices
  - Network architectures for image processing
  - Representing images and sparse 2D keypoints
  - Representing dense and 3D keypoints
  - Representing geometry and shape
  - Representation learning I (deterministic)
  - Representation learning II (probabilistic)
  - Sequential decision making
  - Unpaired image translation
  - Attention models

Project pitches next week!

- 3x Assignments
  - Playing with pytorch (5% of points)
  - Pose estimation (10% of points)
  - Shape generation (10% of points)
- 1x Project (40 % of points)
  - Project pitch (3 min, week 6)
  - Project presentation (10 min, week 14)
  - Project report (8 pages, April 14)
- 1x Paper presentation (Weeks 8 – 13)
  - Presentation, once per student (25% of points) (20 min + 15 min discussion, week 8-13)
  - Read and review one out of the two papers presented per session (10% of points)

Course project proposal

Conditions

- groups of up to two students
  - send us your tentative title/topic and team members by Thursday, Feb 6th
- a CV or CG topic of your choice

Project proposal

- 3-minute pitch
- written proposal (one page, 11pt font)
  - research idea
  - possible algorithmic contributions
  - outline of the planned evaluation

W4	Jan 28	Representation learning I (deterministic) lecture slides - principal component analysis (PCA) - auto-encoder (AE) Homework 2 due. ~~Homework 3 release~~	PCA face model Deep Learning Book - Chapter 14
	Jan 30	Representation learning II (probabilistic) lecture slides - variational autoencoder (VAE) - generative adversarial network (GAN) Homework 3 release Assignment3.zip (posted Feb. 1)	Deep Learning Book - Chapter 20
W5	Feb 4	Sequential decision making - Monte Carlo methods - reinforcement learning	Deep Learning Book - Chapter 17
	Feb 6	Unpaired image translation - cycle consistency - style transfer ~~Homework 3 due~~	Cycle Gan Style transfer
W6	Feb 11	Attention models - spatial transformers, RoI pooling, attention maps - camera models and multi-view Homework 3 due (new deadline)	RoI pooling, Spatial Transformer Multi-view Geometry
	Feb 13	Project Pitches (3 min pitch) Project proposal due	
W7		Midterm Break (no class)	-
W8	Feb 25	Conditional content generation Park et al., Semantic Image Synthesis with Spatially-Adaptive Normalization paper Li et al., Putting Humans in a Scene: Learning Affordance in 3D Indoor Environments paper	
	Feb 27	Motion transfer Chan et al, Everybody Dance Now paper Gao et al., Automatic Unpaired Shape Deformation Transfer paper	

Reinforcement learning

Reinforcement learning examples

Can we perform reinforcement learning at ImageNet scales?

28

L., Pastor, Krizhevsky, Quillen '16

[Sergey Levine, UC Berkeley]

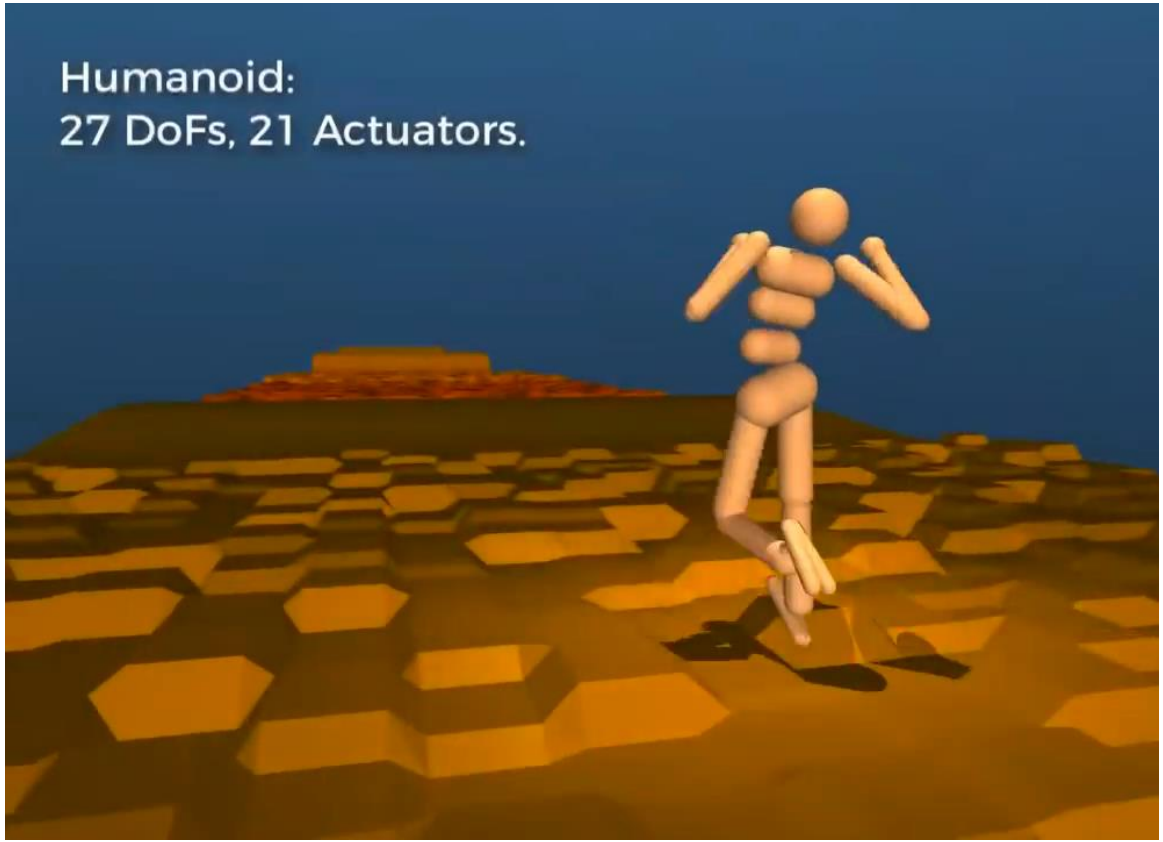
Reinforcement learning examples

Reward:

- get closer to the opposite side
- as quick as possible

Learned policy

- stand upright
- walk
- walk in the right direction
- keep the balance
- run
- jump
- turn



[Heess et al., Emergence of Locomotion Behaviours in Rich Environments]

Reinforcement learning basics

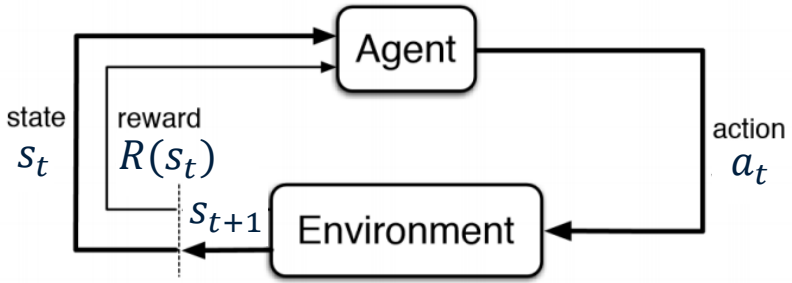
Definitions:

- s_t, the current state of the agent/environment
- $R(s_t)$, the reward/objective at time t
  - might be zero for almost all t
- $R = \sum_{t=0}^{T} R(s_t)$, the return as sum over all rewards
- a, the action, such as moving right or left
- $a_t = \pi(s_t)$, the policy of which action a_t to perform when in state s_t

Goal: finding a good policy π such that R is maximized when executing action $a_t = \pi(s_t)$

Update loop:

- decide on a new action $a_t = \pi(s_t)$
- update the environment state $s_{t+1} = env(s_t, a_t)$
- pay out reward $R(s_t)$



Reinforcement learning unrolled

Unrolling the update loop for N steps

- $R = R(s_0) + R(s_1) + R(s_2) \ldots$

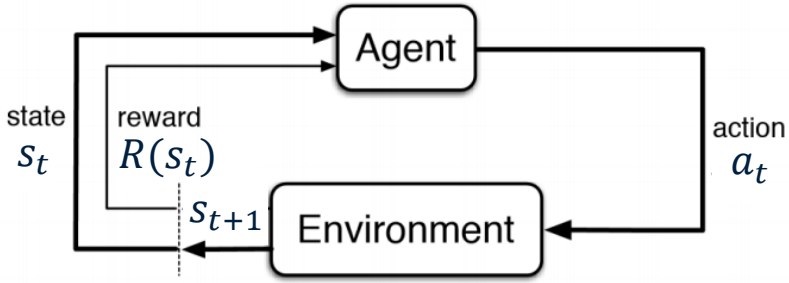
  $= R(s_0) + R(env(s_0, a_0)) + R\big(\text{env}(env(s_0, a_0), a_1)\big) \ldots$

  $= R(s_0) + R(env(s_0, \pi(s_0))) + R\left(\text{env}\big(env(s_0, \pi(s_0)),\ \pi(env(s_0, \pi(s_0)))\big)\right) \ldots$

Goal: finding a good policy π such that R is maximized when executing action $a_t = \pi(s_t)$

Update loop:

- decide on a new action $a_t = \pi(s_t)$
- update the environment state $s_{t+1} = env(s_t, a_t)$
- pay out reward $R(s_t)$



Reinforcement learning: a form of supervised learning

Unrolling the update loop for N steps

- $R = R(s_0) + R(s_1) + R(s_2) \ldots$

  $= R(s_0) + R(env(s_0, a_0)) + R\big(\text{env}(env(s_0, a_0), a_1)\big) \ldots$

  $= R(s_0) + R(env(s_0, \pi(s_0))) + R\left(\text{env}\big(env(s_0, \pi(s_0)),\ \pi(env(s_0, \pi(s_0)))\big)\right) \ldots$

Classical machine learning

- input x, label y, parametric function f

$$O(x, y) = L(f(x), y) \qquad \text{with } f(x) = \text{CNN}_\theta(x, \mathbf{I})$$

Reinforcement learning

- A way of supervised learning
  - $s_0 = x$, R = L(f(x) , y), with f(x) the joint effect of the environment, policy and reward
    - difficult if reward is delayed for many steps

Optimizing the policy

Goal: finding a good policy π such that R is maximized when executing action $a_t = \pi(s_t)$

Difficulty:

- the reward is a complex function of the policy and the environment
  - do we know this function?
    - policy $\pi(s_t)$: yes, we can parametrize it with a neural network $\pi_\theta(s_t)$ with parameters θ
    - environment reaction $env(s_t, a_t)$ to action a_t: often unknown, e.g., a robot in the real world with chaotic behavior and partial observations
      - but we can '*compute*' it once, by performing the action and observing the outcome
  - can we differentiate this function? (needed for gradient decent optimization of π)
    - policy $\pi_\theta(s_t)$:
      - yes? a normal neural network
      - no? binary output not differentiable, e.g. left/right decision when navigating a maze
    - environment: no! we don't even know the function to differentiate…

Binary decisions

Probabilistic interpretation

Changes in the probabilistic model

~~$a_t = \pi_\theta(s_t)$, the policy of which action a_t to perform when in state s_t~~

$[p(a_t^0), p(a_t^1), \ldots p(a_t^N)] = \pi_\theta(s_t)$, the policy that attributes a probability $p(a_t^i)$ to all possible action a_t^i .

- the probability to perform a_t^i when in state s_t
- can also be continuous, a function defined over the domain of states, $\pi_\theta(a_t\ ,\ s_t)$
  - e.g., by how much should the agent turn the steering wheel when driving autonomously?

Probabilistic goal: Maximize the expected return E(R)

- $\arg\max_\theta E(R) = \arg\max_\theta E\ R(s_0) + R(env(s_0, a_0)) + R(\text{env}(env(s_0, a_0), a_1))$, with $a_t \sim \pi_\theta(s_t)$

$$E[R] = \sum_{a_0^i, a_1^j, \ldots \text{ for } i=0\ldots N, j=0..N, \ldots} p(s_0, a_0^i, a_1^i, \ldots) R(s_0, a_0^i, a_1^i, \ldots)$$

Exponentially many combinations, intractable

$$E[f] = \sum_{i=1}^{N} f(x_i)\, p(x_i)$$

Probabilistic interpretation

Simplification for our purposes: only a single decision to be made

- $\arg\max_\theta \mathrm{E}(R) = \arg\max_\theta \mathrm{E}\ R(env(s_0, a_0))$, with $a_0 \sim \pi_\theta(s_0)$

$$\mathrm{E}[R] = \sum_{a_0^i \text{ for } i=0\ldots N} \pi_\theta(s_0) R(\mathrm{env}(s_0, a_0))$$

$$\mathrm{E}[f] = \sum_{i=1}^{N} f(x_i)\, p(x_i)$$

Even a single step is problematic
- large number of possible actions
- sometimes it is not possible to undo an action
  - e.g. if a kitchen robot breaks a glass

Using an estimator

- Monte Carlo approach, can work with a single sample

$$\mathrm{E}[R] \approx N \pi_\theta(s_0) R(\mathrm{env}(s_0, a_0))$$

- simply tries all possible actions at random, quite inefficient

$$\mathrm{E}[f] \approx \frac{N}{K} \sum_{i=1}^{K} f(x_i) p(x_i)$$

with x_i drawn uniformly at random

On-policy reinforcement learning

Probabilistic goal: Maximize the expected return E(R)

- $\arg\max_\theta \mathrm{E}(\mathrm{R}) = \arg\max_\theta \mathrm{E}\ R(env(s_0, a_0))$, with $a_0 \sim \pi_\theta(s_0)$

Using a different estimator (definition of expectation)

- again, with a single decision and single sample

$$\mathrm{E}[R] \approx CR(\mathrm{env}(s_0, a_0)) \text{ with } a_0 \sim \pi_\theta(s_0)$$

- now we try the most likely actions more often
  - a better estimator (lower variance)
  - follows the learned policy, what about exploration?

$$\mathrm{E}[f] = \sum_{i=1}^{C} f(x_i)p(x_i)$$
$$\approx \frac{C}{N}\sum_{i=1}^{N} f(x_i) \text{ with } x_i \sim p$$

- sampling is not differentiable; only for continuous values we could use the reparameterization trick

Recap: Differentiation and sampling

Problem: How to differentiate through the sampling step?

- it's a random process, only statistically dependent on the mean and standard deviation of the sampling distribution

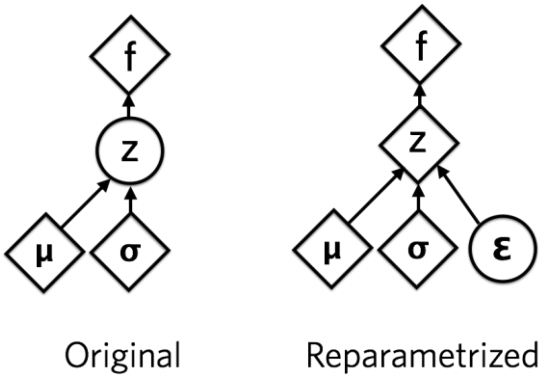
Solutions:

1. The reparameterization trick: Use

$$h = \mu + \sigma \odot \epsilon, \text{ with } \epsilon \sim \mathcal{N}(0, 1)$$

instead of

$$h \sim \mathcal{N}(\mu, \sigma)$$



2. Monte-Carlo solution
   - related to reinforcement learning and importance sampling
   - works for discrete and continuous variables
   - we will cover it next week

Recap: Reparameterization trick, visually and mathematically

Equation: $h = \mu + \sigma \odot \epsilon, \text{ with } \epsilon \sim \mathcal{N}(0, 1)$

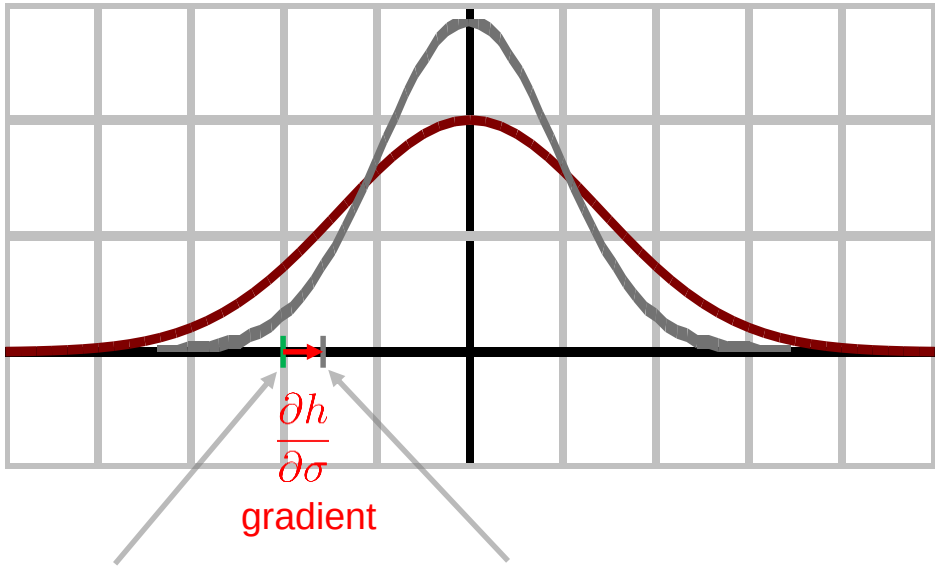
Influence

- changing mu
  - increase -> moves sample right
  - decrease -> moves sample left
- changing sigma
  - increase -> moves away from center
  - decrease -> moves to the center

Gradient

$$\frac{\partial h}{\partial \sigma} = \epsilon, \text{ with } \epsilon \sim \mathcal{N}(0, 1)$$

$$\frac{\partial h}{\partial \mu} = 1$$



Summary

Probabilistic goal: Maximize the expected return E(R)

$$\mathrm{E}[R] \approx CR(\mathrm{env}(s_0, a_0)) \text{ with } a_0 \sim \pi_\theta(s_0)$$

Update loop:

- decide on a new action $a_t \sim \pi_\theta(s_t)$
- update the environment state $s_{t+1} = env(s_t, a_t)$
- pay out reward $R(s_t)$
- repeat N times
- compute gradient of $R(s_t)$ with respect to θ

Problem:

- the reparameterization trick only works for certain kind of continuous distributions (e.g. Gaussian)
- in general, the environment update function, $env(s_0, a_0)$, is unknown/not differentiable
  - it cuts off gradient flow to a_0, backpropagation to θ is not possible

Revisit the initial probabilistic formulation for gradients

Simplification for our purposes: only a single decision to be made

- $\arg\max_\theta \mathrm{E}(R) = \arg\max_\theta \mathrm{E}\ R(env(s_0, a_0))$, with $a_0 \sim \pi_\theta(s_0)$

$$\mathrm{E}[R] = \sum_{a_0^i \text{ for } i=0\ldots N} \pi_\theta(s_0) R(\mathrm{env}(s_0, a_0^i))$$

$$\mathrm{E}[f] = \sum_{i=1}^{N} f(x_i)\, p(x_i)$$

Using an estimator

- Monte Carlo approach, can work with a single sample

$$\mathrm{E}[R] \approx N \pi_\theta(s_0) R(\mathrm{env}(s_0, a_0))$$

- simply tries all possible actions at random, quite inefficient

$$\mathrm{E}[f] \approx \frac{N}{K} \sum_{i=1}^{K} f(x_i) p(x_i)$$

with x_i drawn uniformly at random

Derivative of discrete random variables

1. Start from the sum over all possible classes
2. **Computer gradient** of this equation with respect to NN parameters
3. Multiply by ‘1’

$$\mathrm{E}[f] \approx \sum_{i=1}^{C} f(x_i) p_\theta(x_i)$$

$$\frac{\partial \mathrm{E}[f(X)]}{\partial \theta} \approx \sum_{i=1}^{C} f(x_i) \frac{\partial p_\theta(x_i)}{\partial \theta}$$

$$\frac{\partial \mathrm{E}[f(X)]}{\partial \theta} \approx \sum_{i=1}^{C} \frac{p_\theta(x_i)}{p_\theta(x_i)} f(x_i) \frac{\partial p_\theta(x_i)}{\partial \theta}$$

4. Algebraic transformation

$$\frac{\partial \log f(x)}{\partial x} = \frac{\frac{\partial f(x)}{\partial x}}{f(x)}$$

$$\frac{\partial \mathrm{E}[f(X)]}{\partial \theta} \approx \sum_{i=1}^{C} p_\theta(x_i) f(x_i) \frac{\partial \log\left(p_\theta(x_i)\right)}{\partial \theta}$$

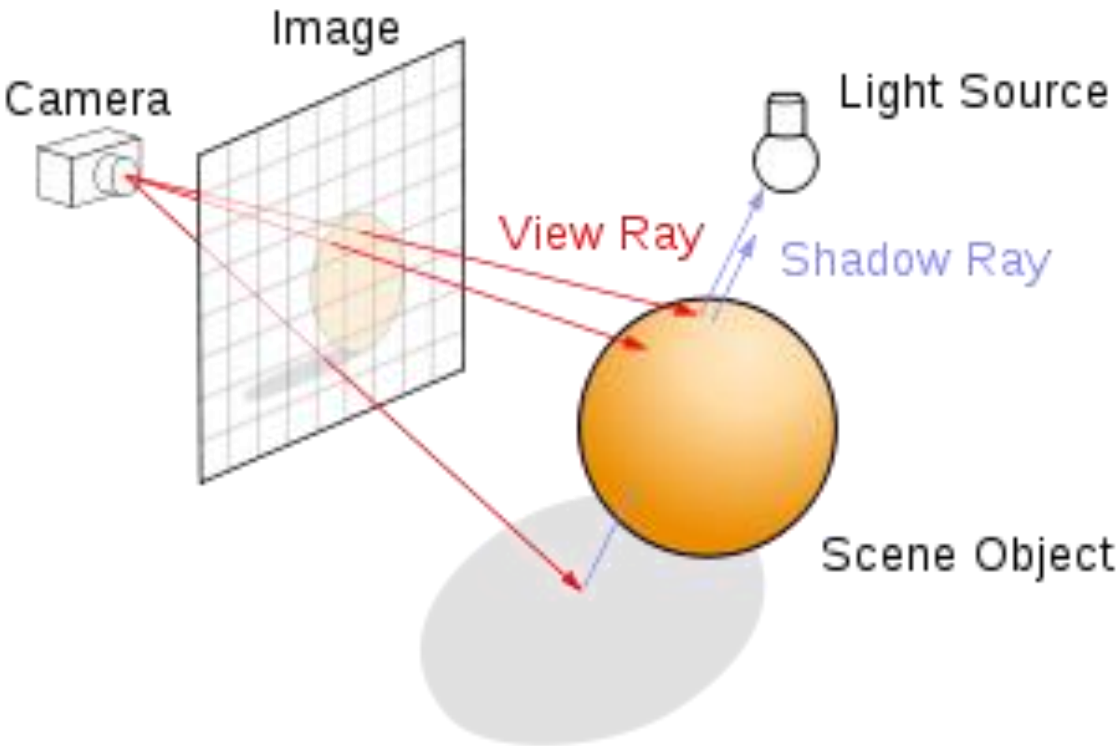
5. Definition of expectation

$$\frac{\partial \mathrm{E}[f(X)]}{\partial \theta} \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i) \frac{\partial \log\left(p_\theta(x_i)\right)}{\partial \theta} \quad \text{with } x_i \sim p_\theta$$

- Now we sample from the policy (efficient) and have a relation on θ for gradient descent
  - How could we do this? Magic? What does this transformation mean?

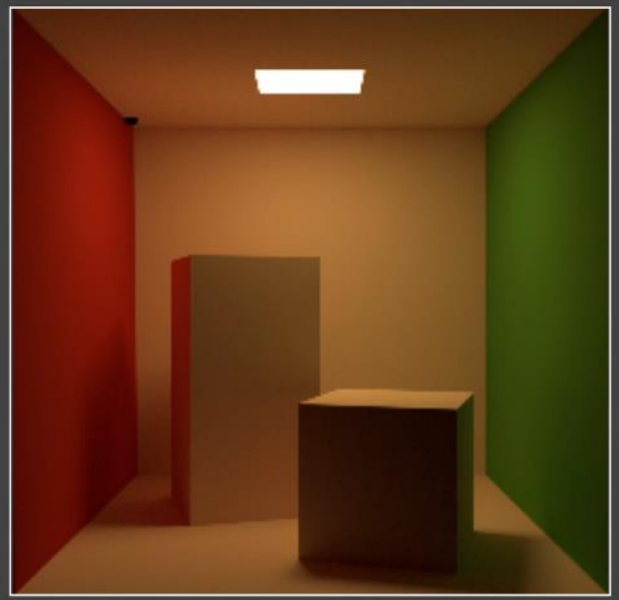
Importance sampling, a computer graphics point of view

Sampling approaches are commonly used in rendering (ray tracing)



[https://en.wikipedia.org/wiki/Ray_tracing_(graphics)]

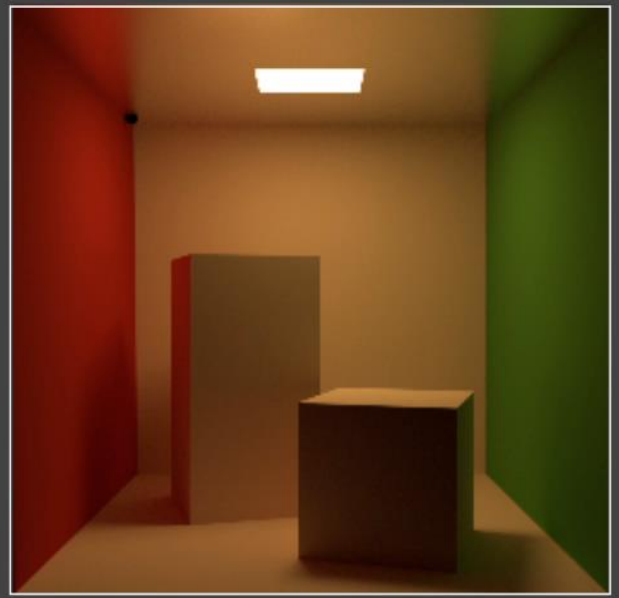
Importance sampling, a computer graphics point of view



$$\int_{S^2} \cdot \, \mathrm{d}\omega$$

$$= \int_{S^2} \mathrm{d}\omega =$$

Importance sampling, a computer graphics point of view



$$\int_{S^2} \cdot \, \mathrm{d}\omega$$

$$= \int_{S^2} \mathrm{d}\omega =$$

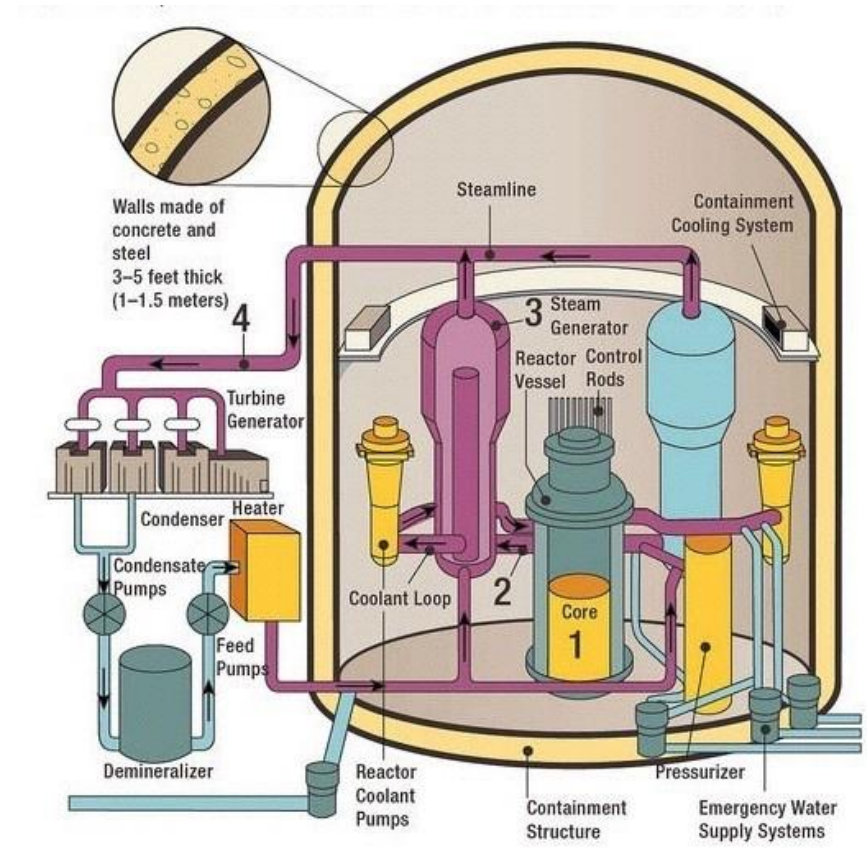
Importance sampling

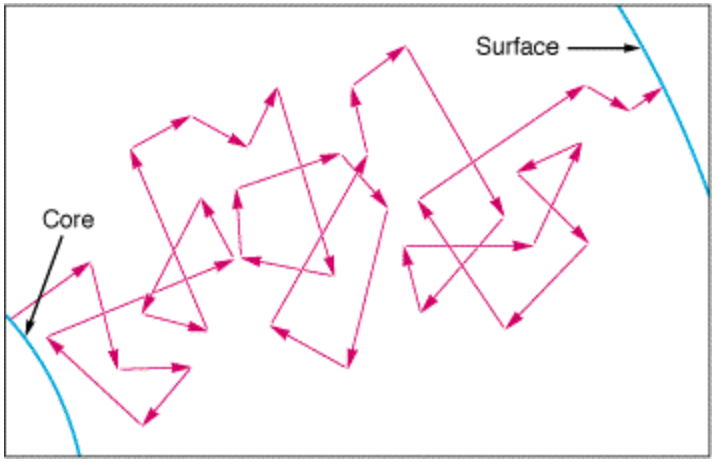
Idea: Sample those important paths more often

Challenges:

- Compute an unbiased estimate
  - if many samples were taken, the estimate should approach the true integral
- Compute a tight approximation
  - low variance over different samples
    - related to how to choose the *important path*

First use of importance sampling, chain reaction etc.

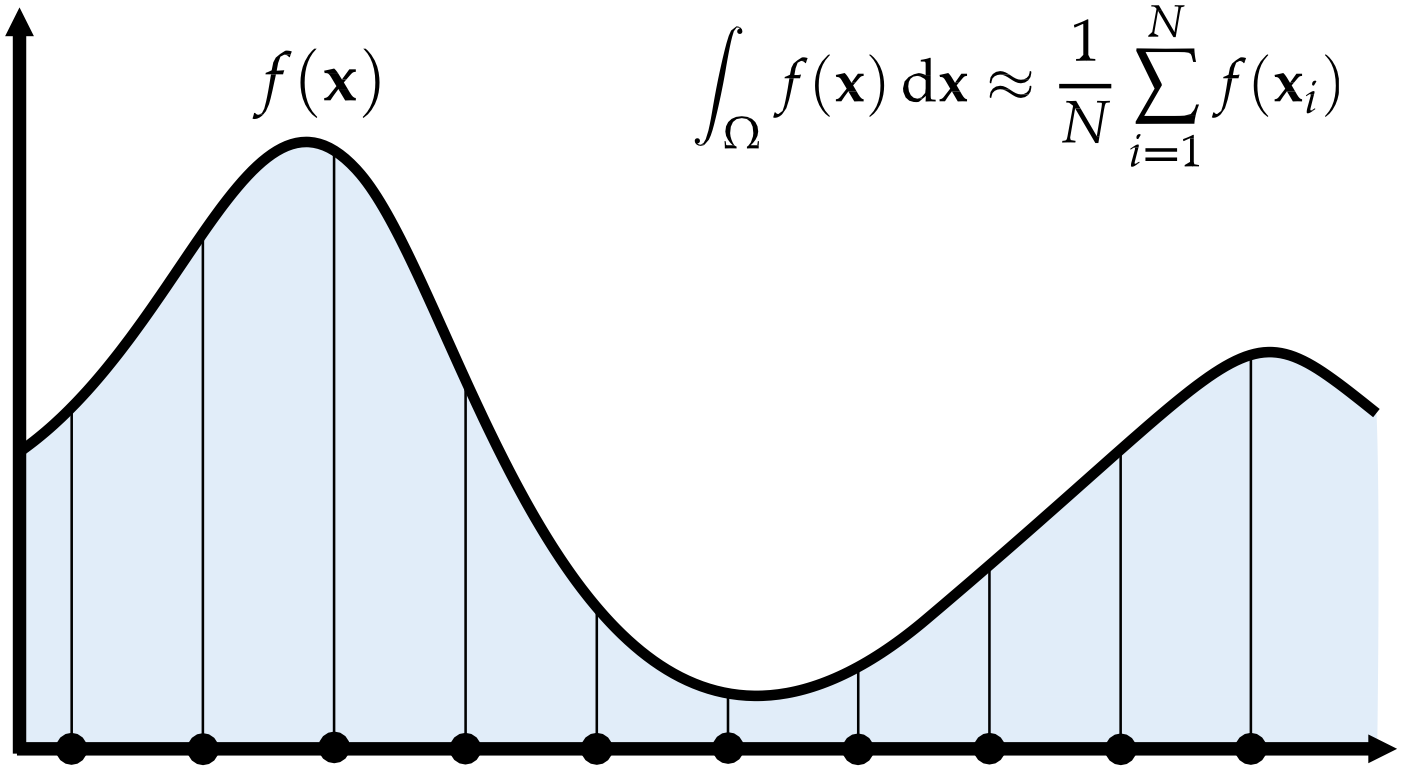




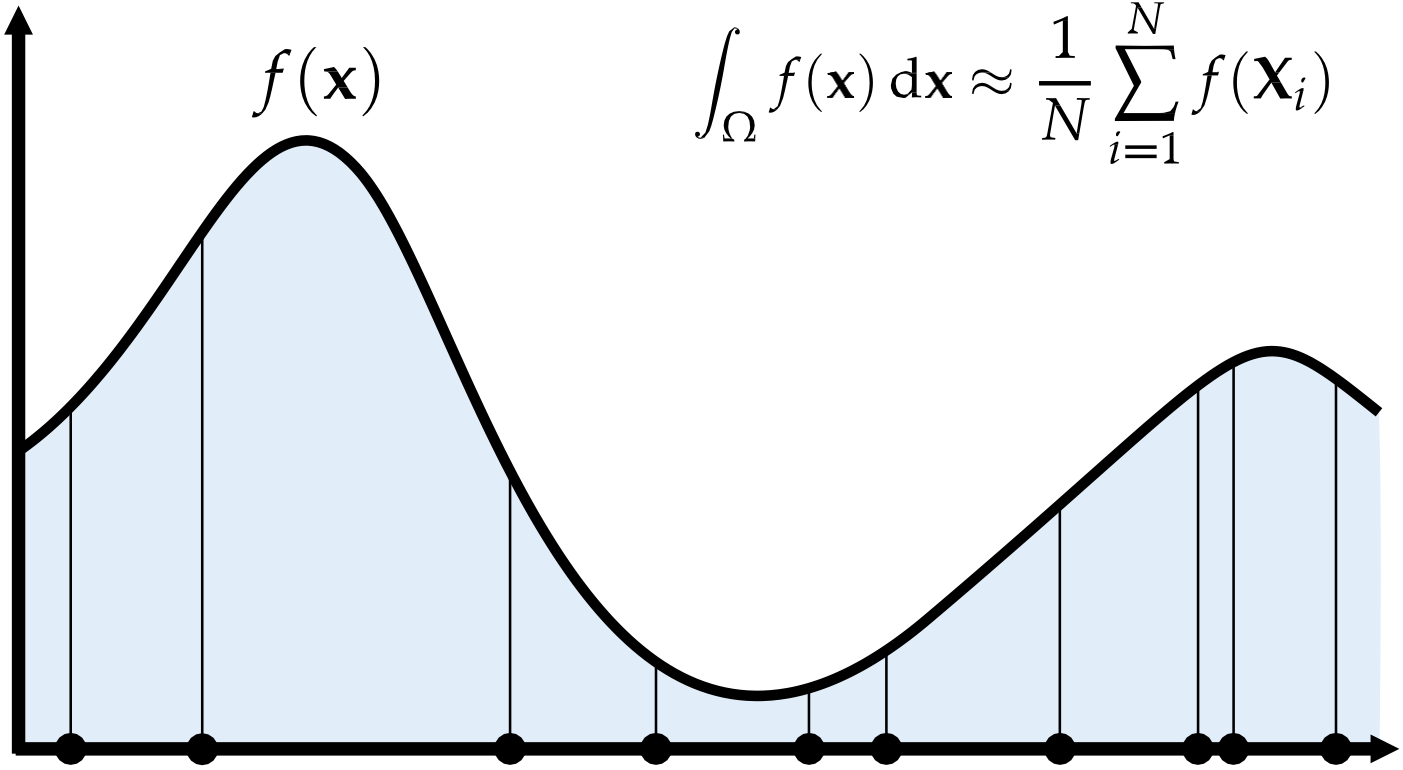
Trajectory of a neutron

Is it safe to stand next to the reactor shielding?

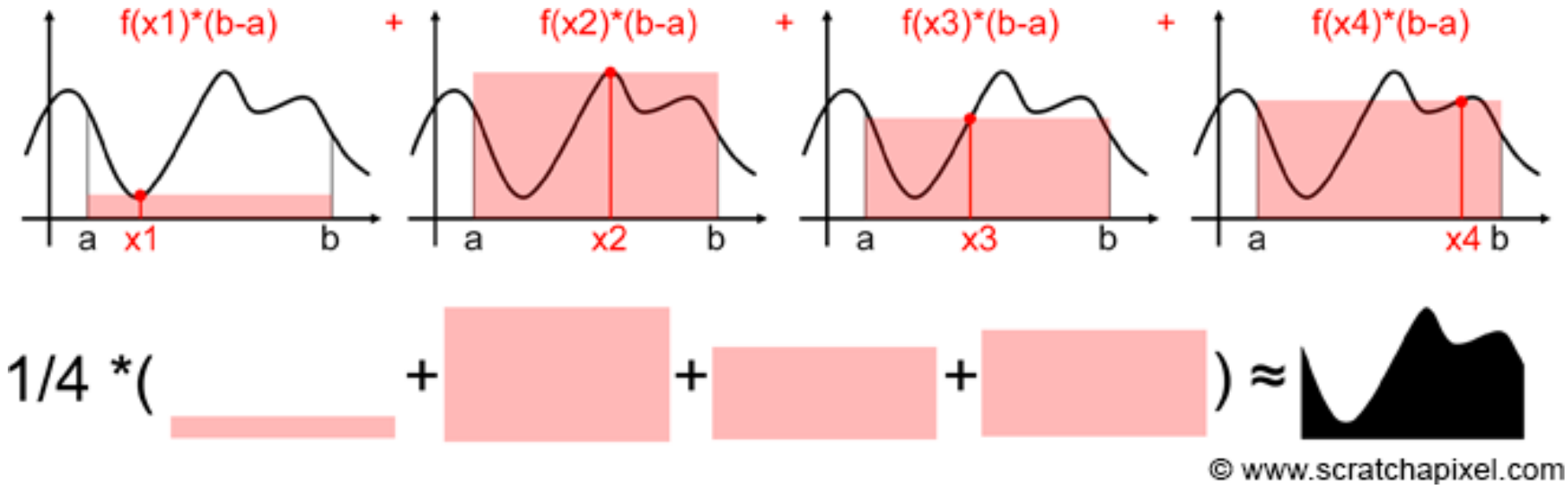
Riemann sum/integral



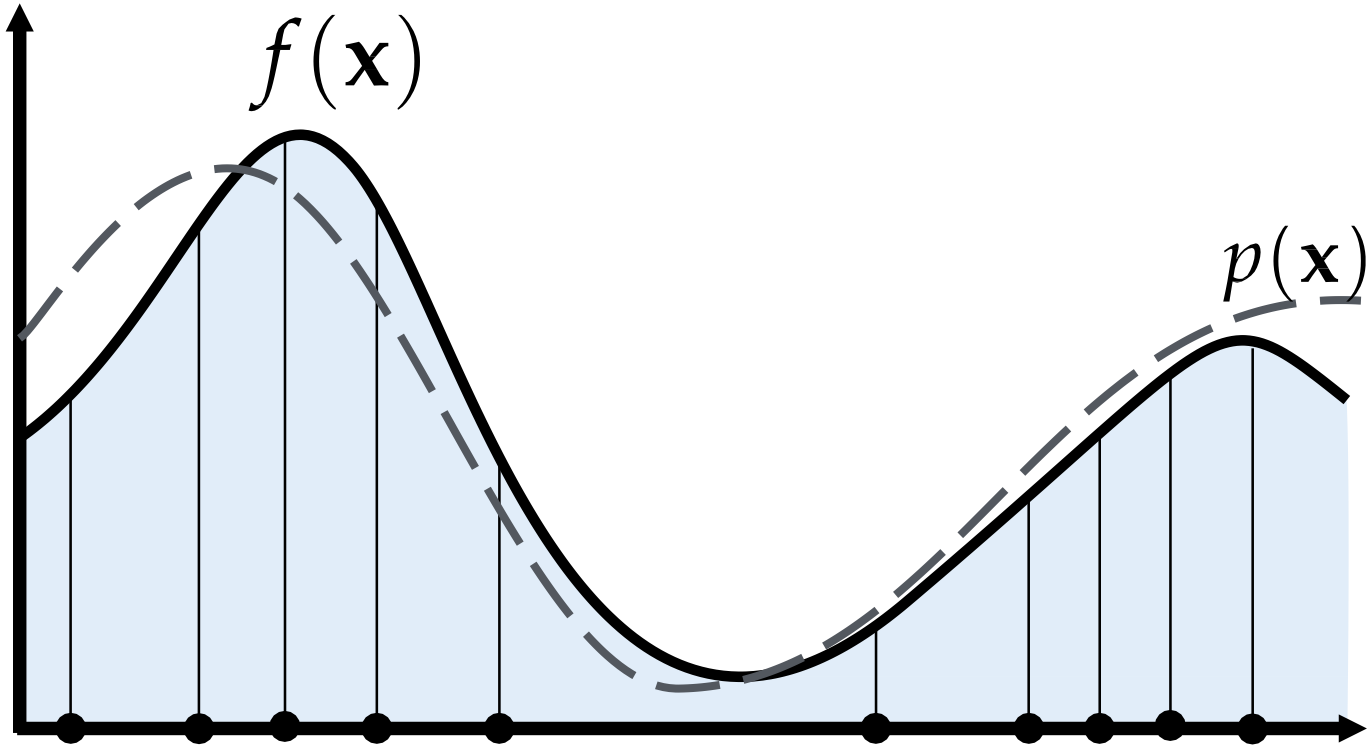
Monte Carlo (MC) integration



Monte Carlo integration: average of boxes



Importance sampling



How can this work?

$$E\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(\mathbf{X}_i)}{p_{\mathbf{X}}(\mathbf{X}_i)}\right] = \frac{1}{N}\sum_{i=1}^{N} E\left[\frac{f(\mathbf{X}_i)}{p_{\mathbf{X}}(\mathbf{X}_i)}\right]$$

Linearity of the expected value

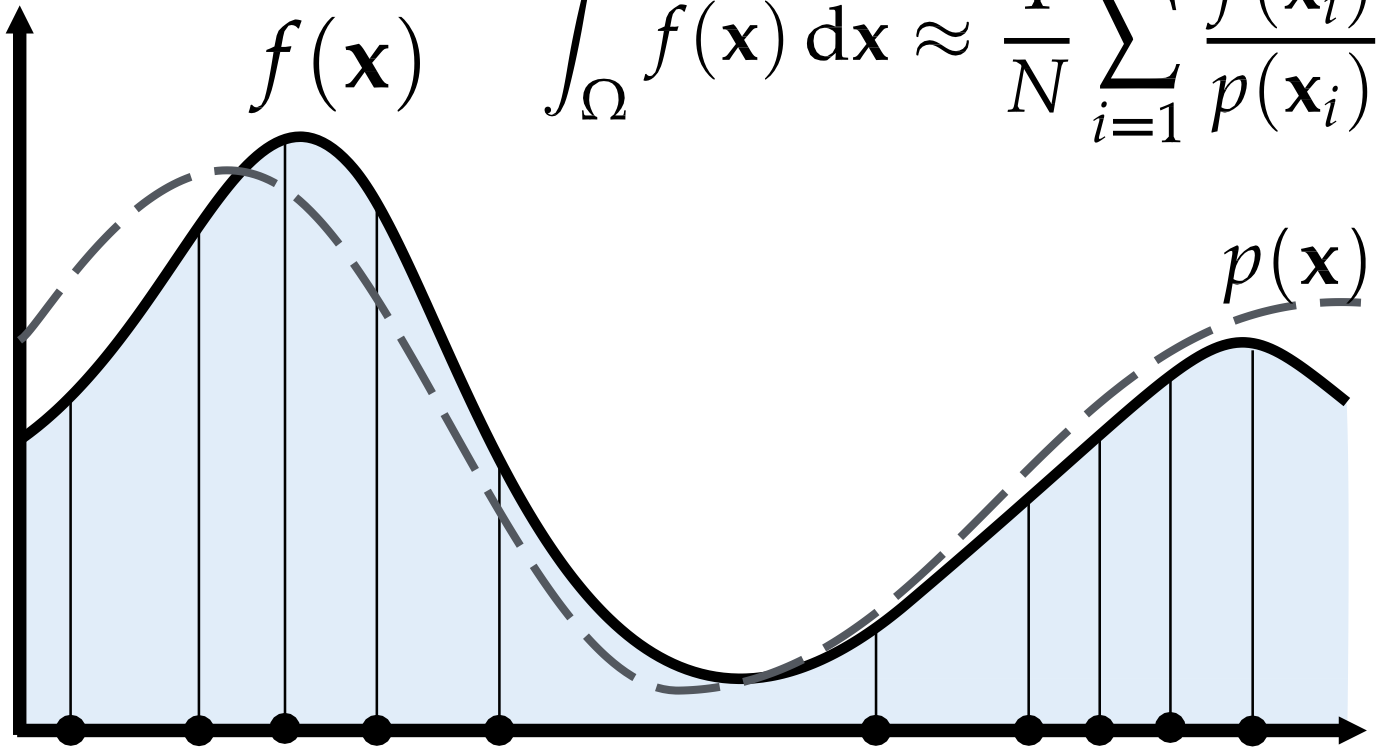
$$= \frac{1}{N}\sum_{i=1}^{N}\int_{\Omega}\frac{f(\mathbf{x})p_{\mathbf{X}}(\mathbf{x})}{p_{\mathbf{X}}(\mathbf{x})}\,\mathrm{d}\mathbf{x}$$

Expectation over the continuous set of possible outcomes

$$= \int_{\Omega} f(\mathbf{x})\frac{p_{\mathbf{X}}(\mathbf{x})}{p_{\mathbf{X}}(\mathbf{x})}\,\mathrm{d}\mathbf{x} \quad \overset{?!}{=} \quad \int_{\Omega} f(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

Importance sampling

$$\int_\Omega f(\mathbf{x})\,\mathrm{d}\mathbf{x} \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}$$



Variance of the estimator

How big is $\text{Var}\left[\frac{f(\mathbf{X}_i)}{p(\mathbf{X}_i)}\right]$?

$$\frac{f(\mathbf{x})}{p(\mathbf{x})} \approx f^2(\mathbf{x}) \qquad \frac{f(\mathbf{x})}{p(\mathbf{x})} \approx f(x) \qquad \frac{f(\mathbf{x})}{p(\mathbf{x})} \approx 1$$

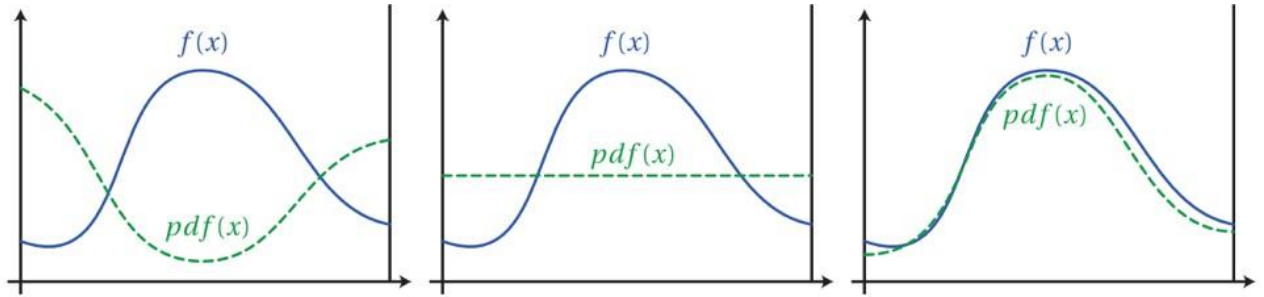


Figure A.2: Comparison of three probability density functions. The PDF on the right provides variance reduction over the uniform PDF in the center. However, using the PDF on the left would significantly increase variance over simple uniform sampling.

[Jarosz etal.]

Derivative of discrete random variables II

1. Start from uniform MC sampling of f
   (note, not yet of the gradient)
2. Importance sample with distribution q

$$\mathrm{E}[f] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)p_\theta(x_i) \quad \text{with } x_i \sim \text{Uniform}$$

$$\mathrm{E}[f] \approx \frac{1}{N}\sum_{i=1}^{N} \frac{p_\theta(x_i)}{q(x_i)} f(x_i) \quad \text{with } x_i \sim q$$

3. Compute gradient
   (before this was the first step)
4. Assume q=p and express as logarithm
   - the same as log trick!
   - but now it makes sense
     - importance sampling with the current policy
     - we don't change the samples, hence, no gradient flow through q

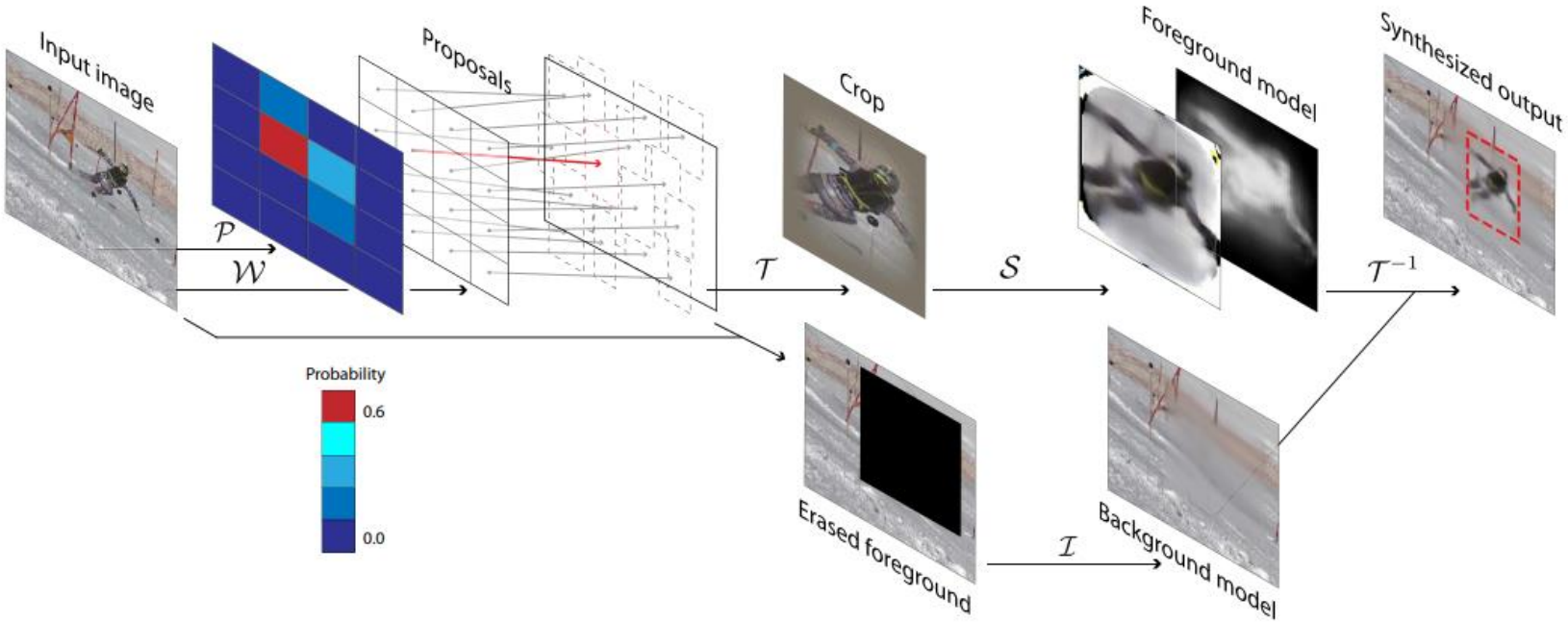
$$\frac{\partial \mathrm{E}[f(X)]}{\partial \theta} \approx \sum_{i=1}^{N} \frac{\frac{\partial p_\theta(x_i)}{\partial \theta}}{q} f(x_i)$$

$$\frac{\partial \mathrm{E}[f(X)]}{\partial \theta} \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \frac{\partial \log\left(p_\theta(x_i)\right)}{\partial \theta}$$

- Advantages: We can sample from q != p, i.e. to encourage exploitation or reduce variance.
  Easier to implement and understand. A large literature on how to improve importance sampling!

Applications, my research I (object detection)

Proposal-based detection



[Katircioglu et al., Self-supervised Training of Proposal-based Segmentation via Background Prediction]

Applications, my research II (learning body models)

Cross-view correspondence finding

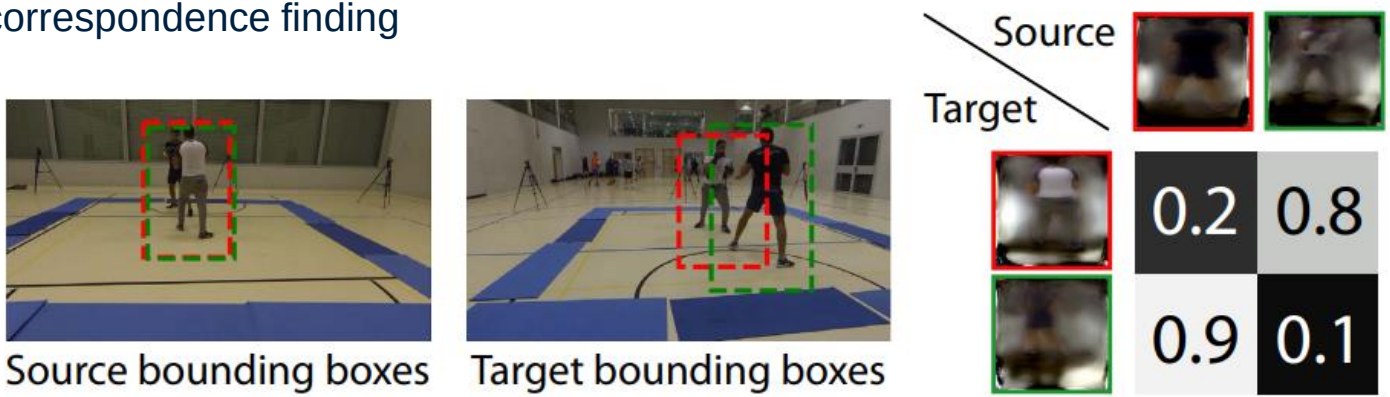


Figure 5. **Identity association.** In this example, the light subject is detected once as the first and once as the second subject, here visualized by red and green boxes. To match subjects across views, we build a similarity matrix from their respective appearance encodings, as shown on the right.

[Rhodin et al., Neural Scene Decomposition for Multi-Person Motion Capture. CVPR'19]

Applications, Michiel van de Panne (character animation & more)