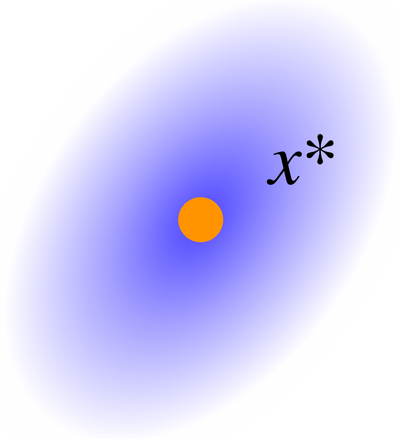
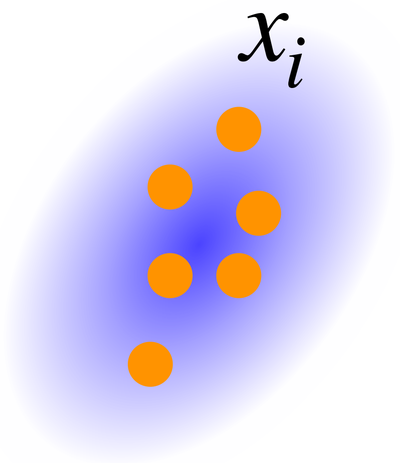
Sampling vs Optimization

UCSD CSE 291 Differentiable Programming

Tzu-Mao Li

Optimization vs sampling





$$x^* = \arg\min_x f(x)$$

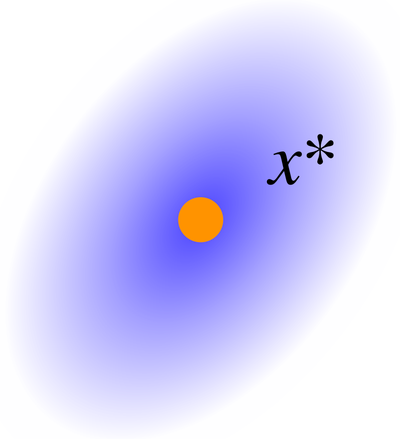
$$x_i \sim \frac{1}{Z} e^{-\frac{f(x_i)}{T}}$$

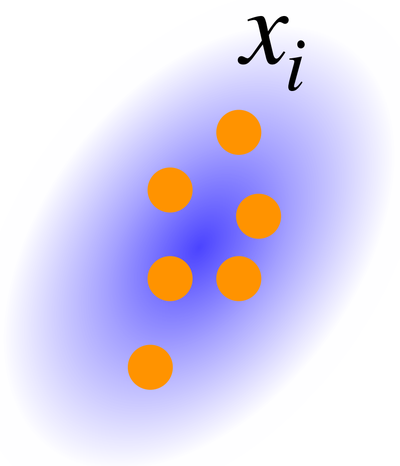
Z: normalization factor

T: “temperature”

Optimization vs sampling

when do we want to optimize, and when do we want to sample?



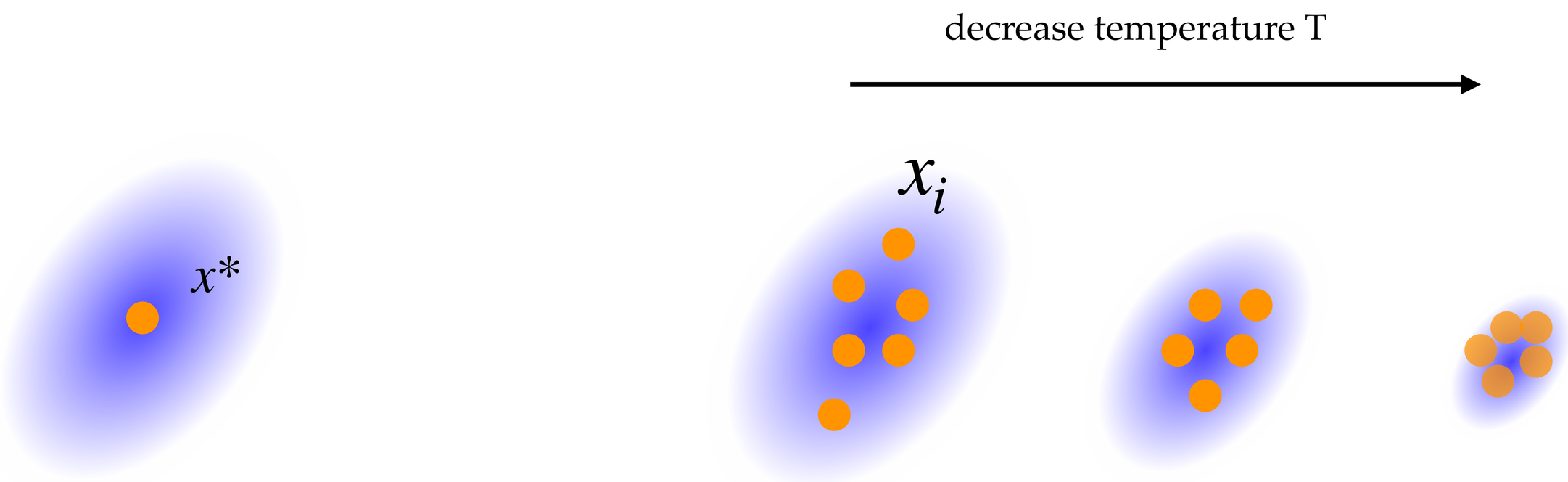


$$x^* = \arg\min_x f(x)$$

$$x_i \sim \frac{1}{Z} e^{-\frac{f(x_i)}{T}}$$

Z: normalization factor
T: “temperature”

Sampling is a generalization of optimization



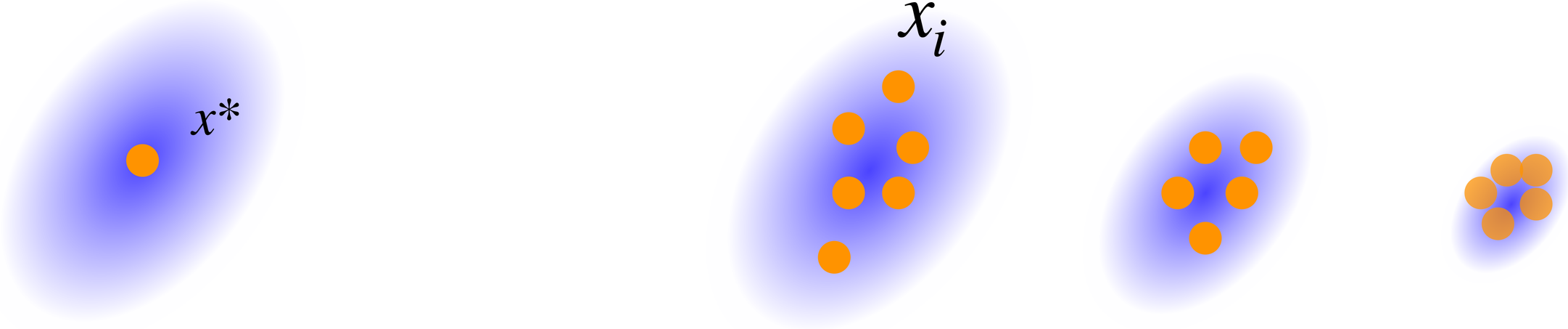
$$x^* = \arg\min_x f(x)$$

$$x_i \sim \frac{1}{Z} e^{-\frac{f(x_i)}{T}}$$

Sampling is a generalization of optimization

does this mean that sampling
is harder than optimization?

decrease temperature T



$$x^* = \arg\min_x f(x)$$

$$x_i \sim \frac{1}{Z} e^{-\frac{f(x_i)}{T}}$$

In higher-dimensional space, sampling behaves differently compared to optimization

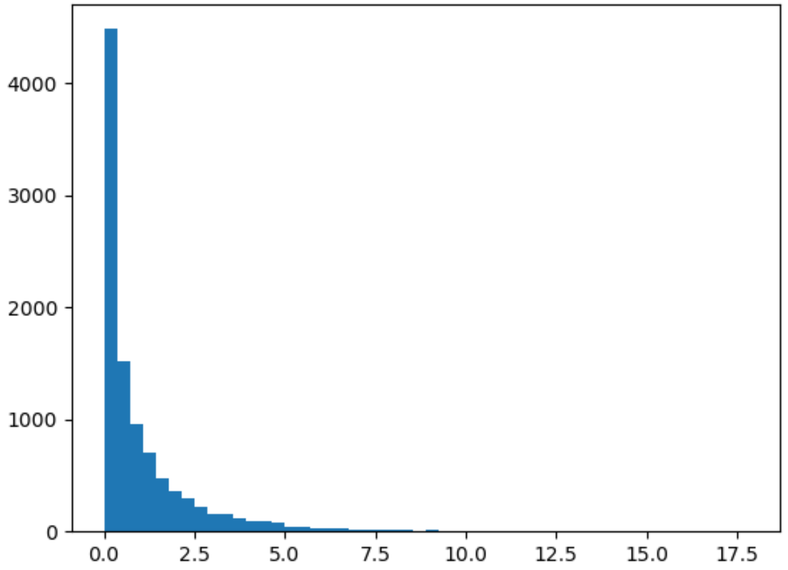
Consider sampling from a zero mean unit variance n-D normal distribution,
the global optimum is at the origin

let's plot the histogram of the squared distance of samples to the origin

In higher-dimensional space, sampling behaves differently compared to optimization

Consider sampling from a zero mean unit variance n-D normal distribution,
the global optimum is at the origin

let's plot the histogram of the squared distance of samples to the origin

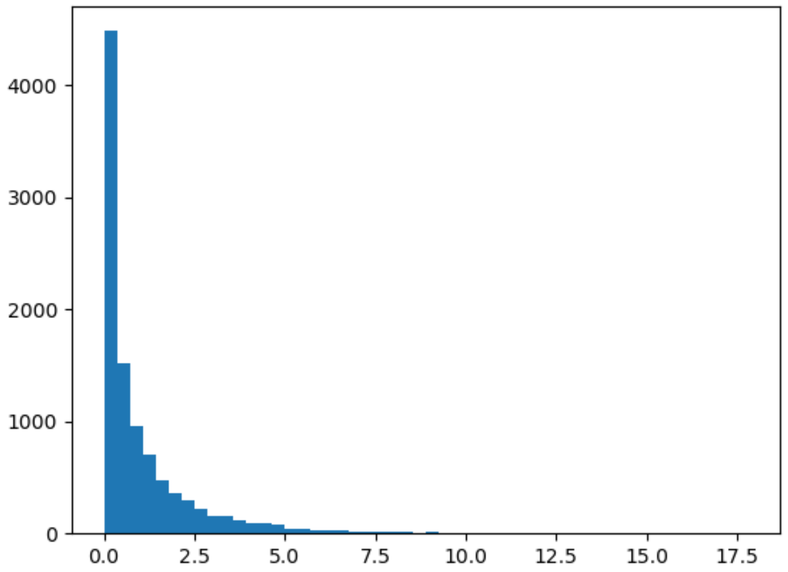


n=1

In higher-dimensional space, sampling behaves differently compared to optimization

Consider sampling from a zero mean unit variance n-D normal distribution,
the global optimum is at the origin

let's plot the histogram of the squared distance of samples to the origin



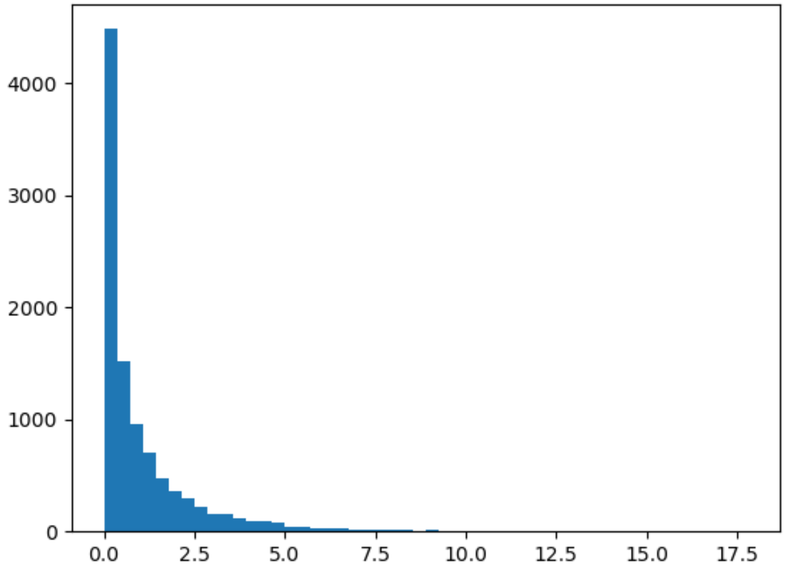
n=1

n=3

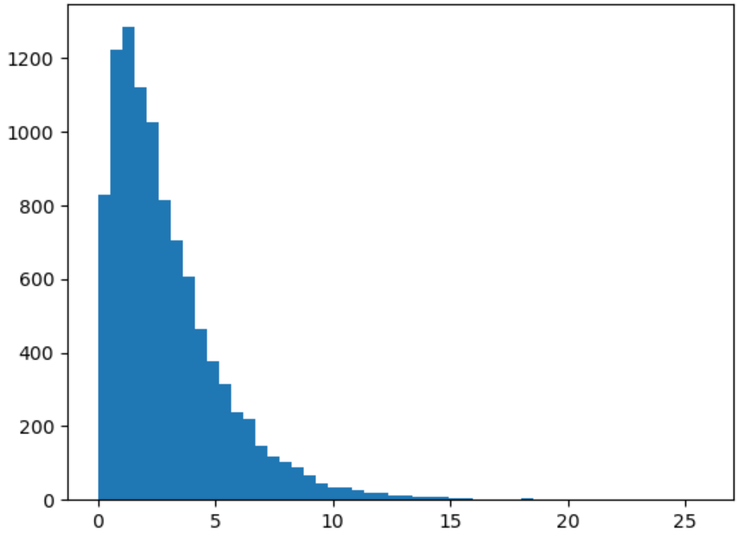
In higher-dimensional space, sampling behaves differently compared to optimization

Consider sampling from a zero mean unit variance n-D normal distribution,
the global optimum is at the origin

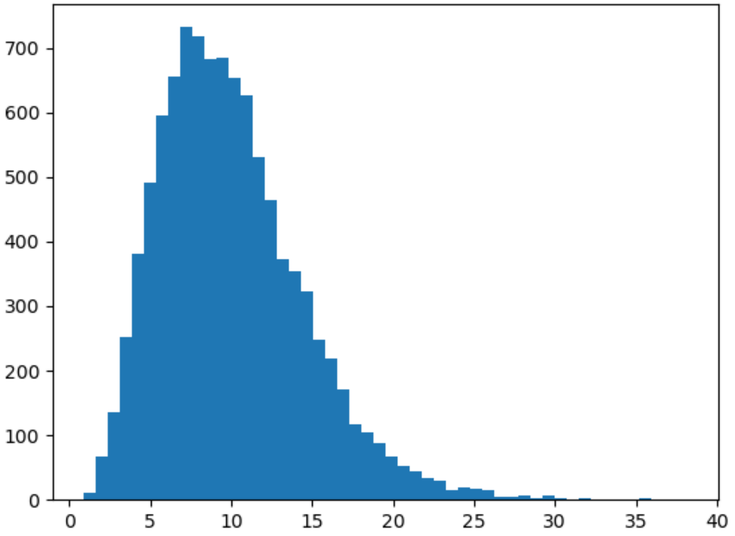
let's plot the histogram of the squared distance of samples to the origin



n=1



n=3



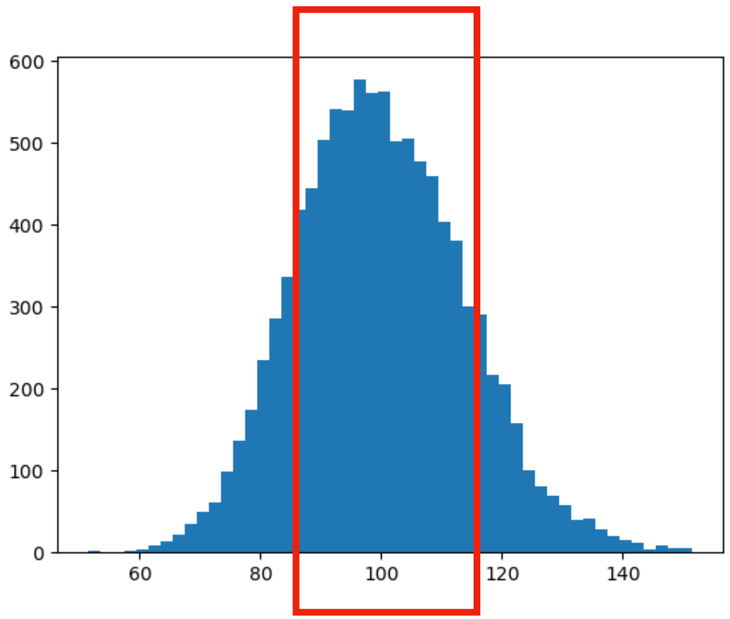
n=10

In higher-dimensional space, sampling behaves differently compared to optimization

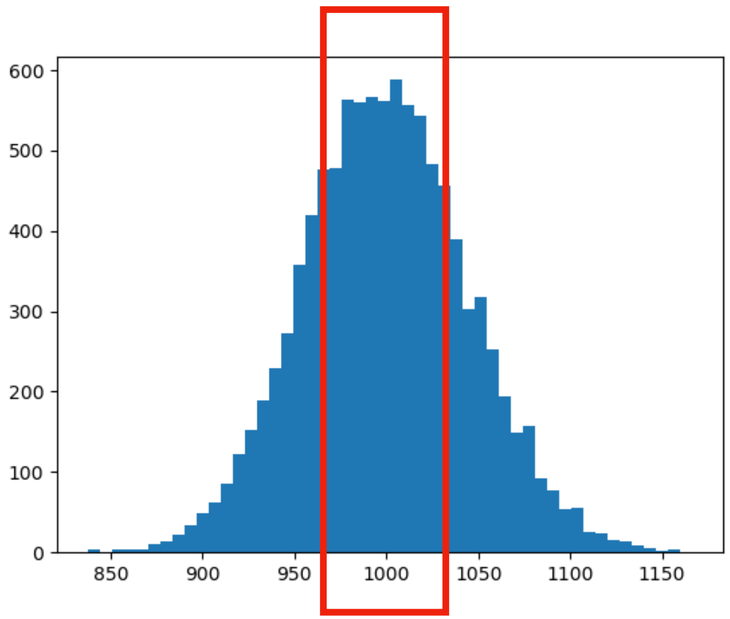
let's plot the histogram of the squared distance of samples to the origin

the squared distance distribution peaks at **n**!!

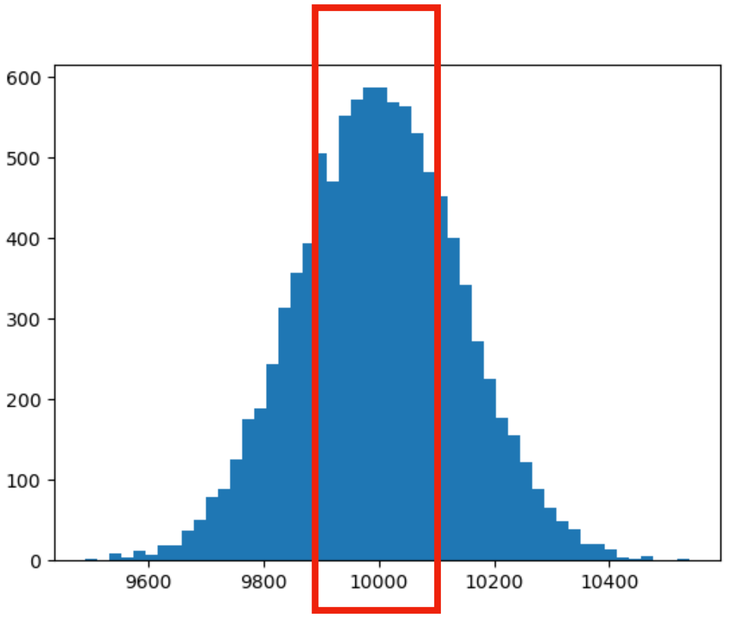
this means that at higher-dimensional space, with fixed temperature sampling will have lower probability to give global minimum!



n=100



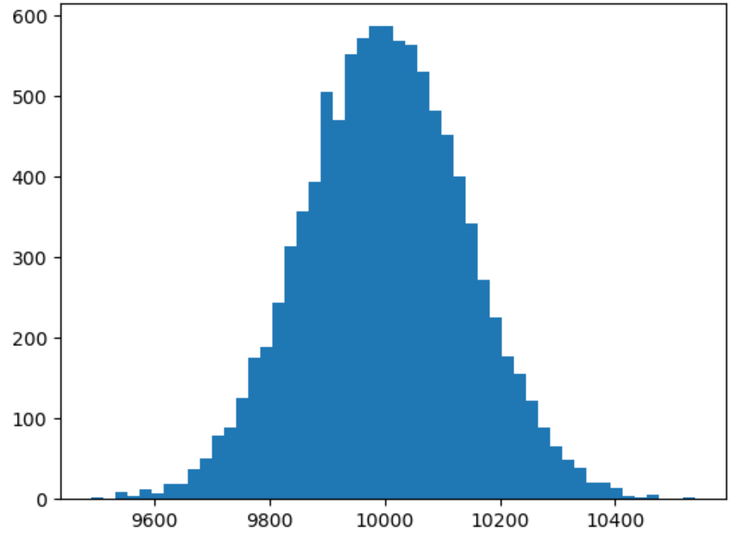
n=1000



n=10000

Intuition: in high-dimensional space, Gaussian distribution looks like a soap bubble

the samples will concentrate at a thin surface
at distance $\sqrt{n}$ to the mode

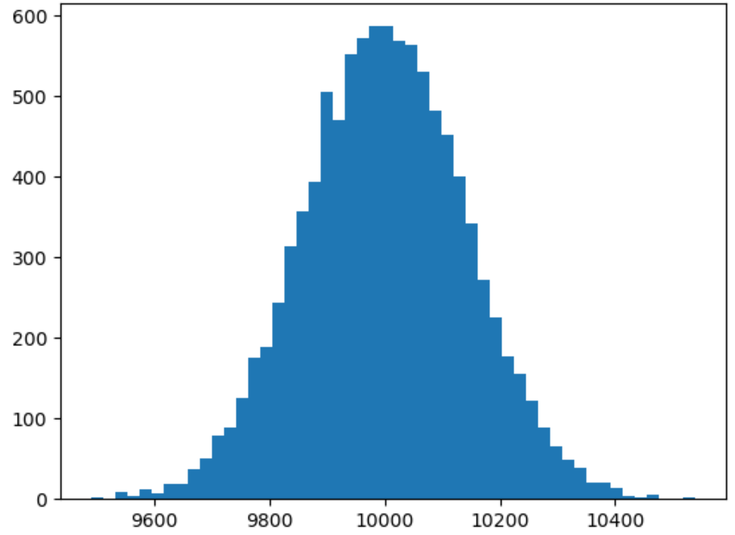


https://www.inference.vc/high-dimensional-gaussian-distributions-are-soap-bubble/

Intuition: in high-dimensional space, Gaussian distribution looks like a soap bubble

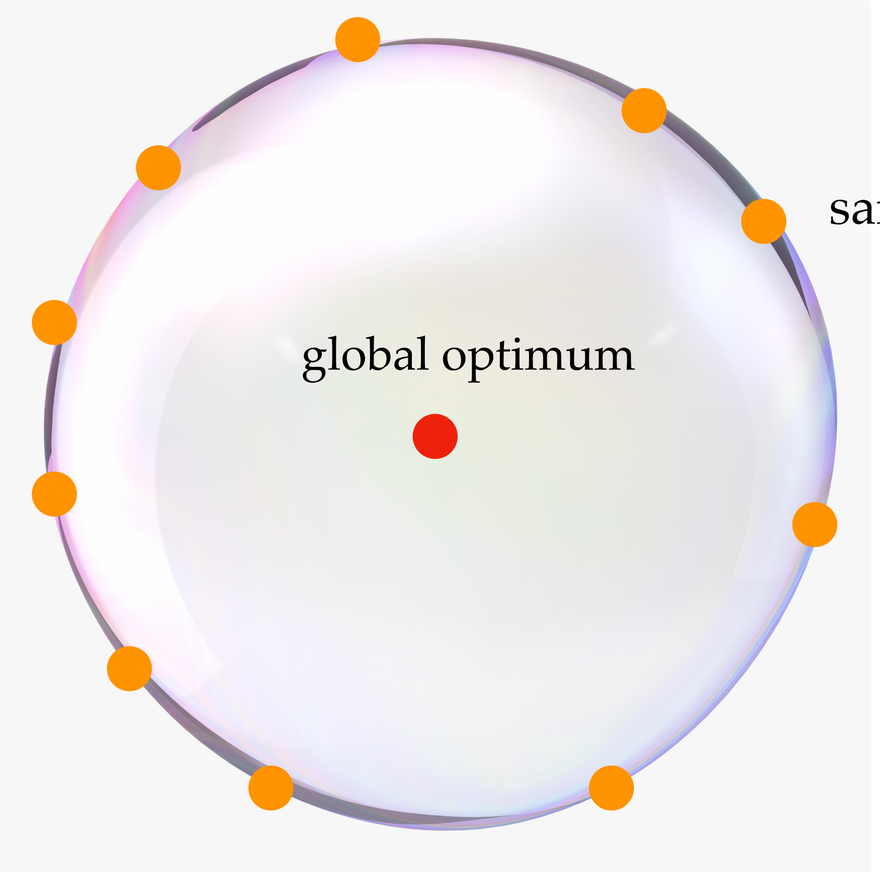
the samples will concentrate at a thin surface
at distance $\sqrt{n}$ to the mode

alternatively, think about this analogy:
a person's face is very unlikely to be an "average face",
they will either have eyes slightly bigger, mouth slightly smaller, etc



https://www.inference.vc/high-dimensional-gaussian-distributions-are-soap-bubble/

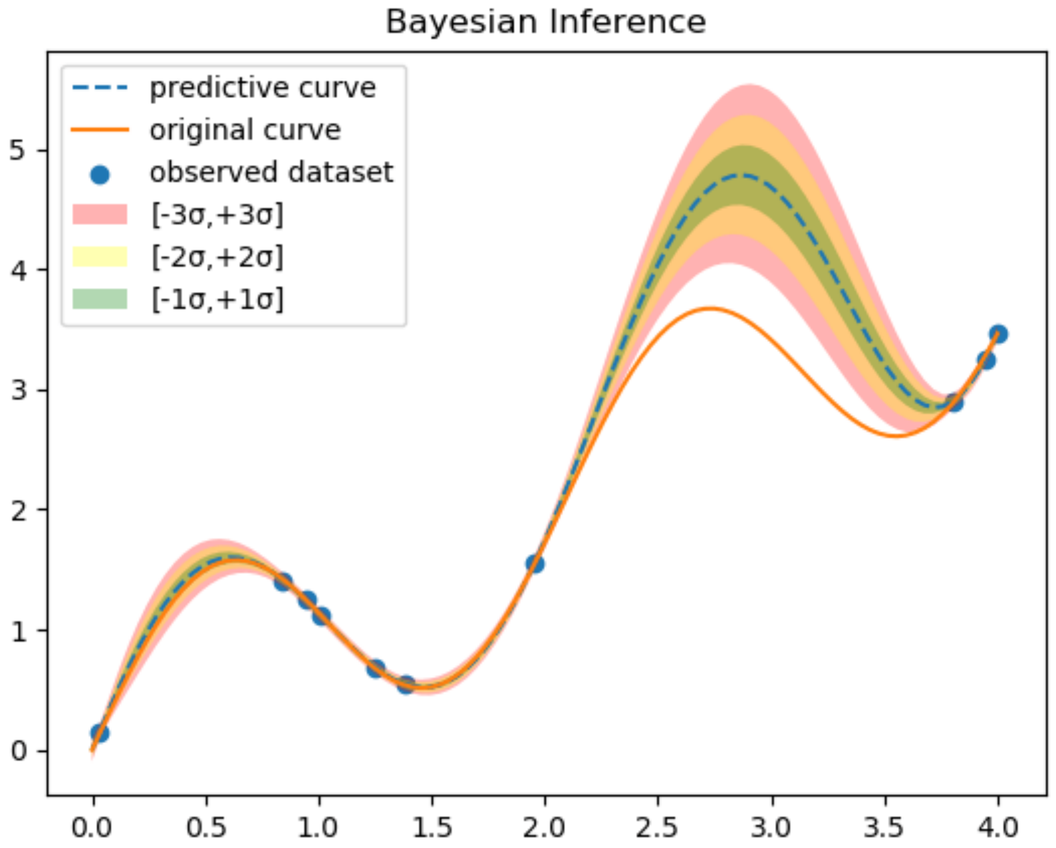
Sampling naturally acts as an regularization in optimization



most of the time, we actually don't want the global optimum, instead we want the "typical set" of our target probability distribution

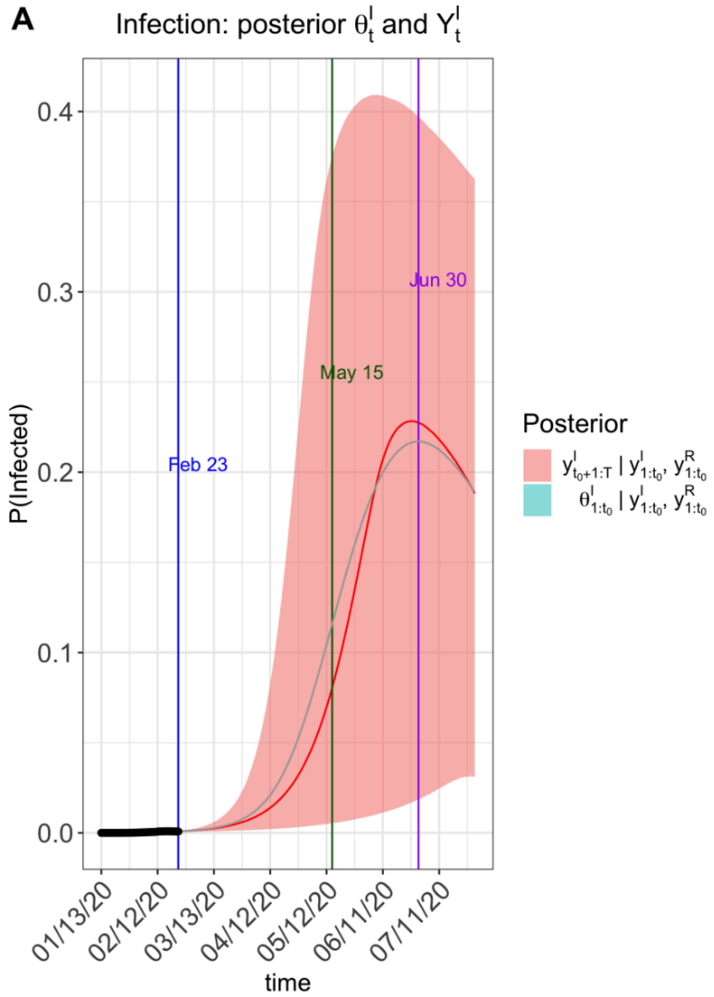
https://www.inference.vc/high-dimensional-gaussian-distributions-are-soap-bubble/

Sampling gives us uncertainty estimates



https://recruit.cct-inc.co.jp/tecblog/machine-learning/markov-chain-monte-carlo/

Sampling gives us uncertainty estimates



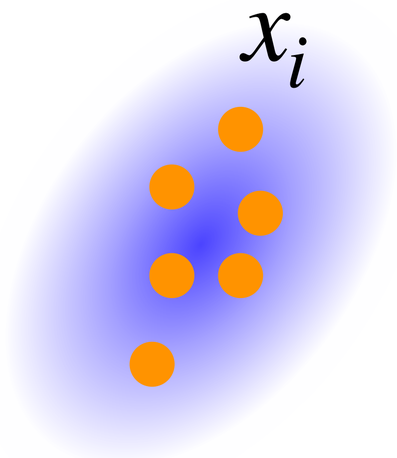
COVID infection prediction

https://www.medrxiv.org/content/10.1101/2020.02.29.20029421v1.full.pdf

Summary: sampling is useful!

- sampling generalizes optimization
- sampling acts as an regularization in high-dimensional space
- sampling estimates uncertainty

How do we sample?



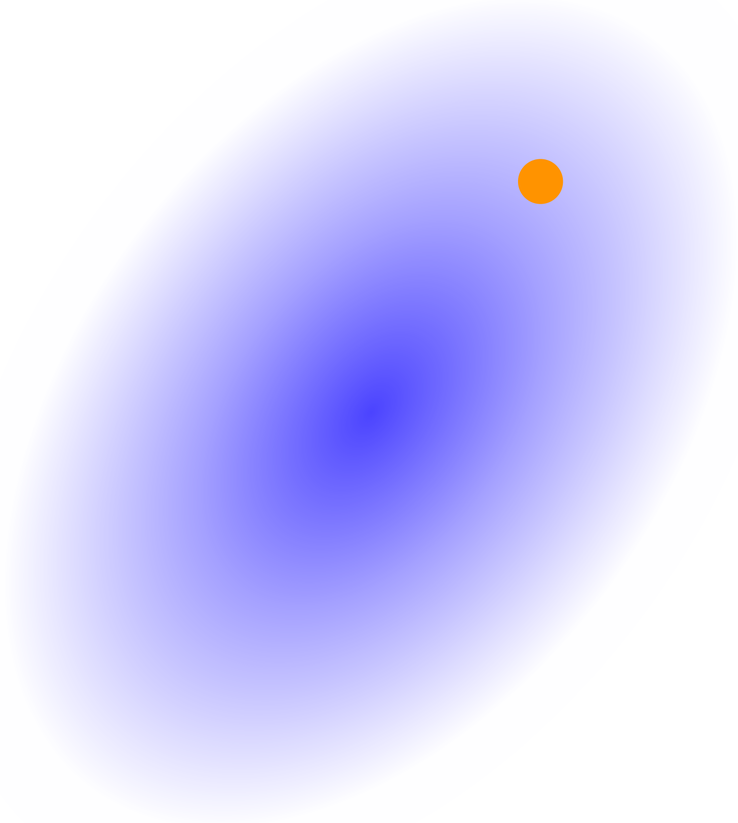
$$x_i \sim \frac{1}{Z} e^{-\frac{f(x_i)}{T}}$$

Markov Chain Monte Carlo (MCMC)

goal: given a target function $g(x) = e^{-\frac{f(x)}{T}}$, generate samples x_i s.t. $x_i \sim \frac{1}{Z} g(x)$

Markov Chain Monte Carlo (MCMC)

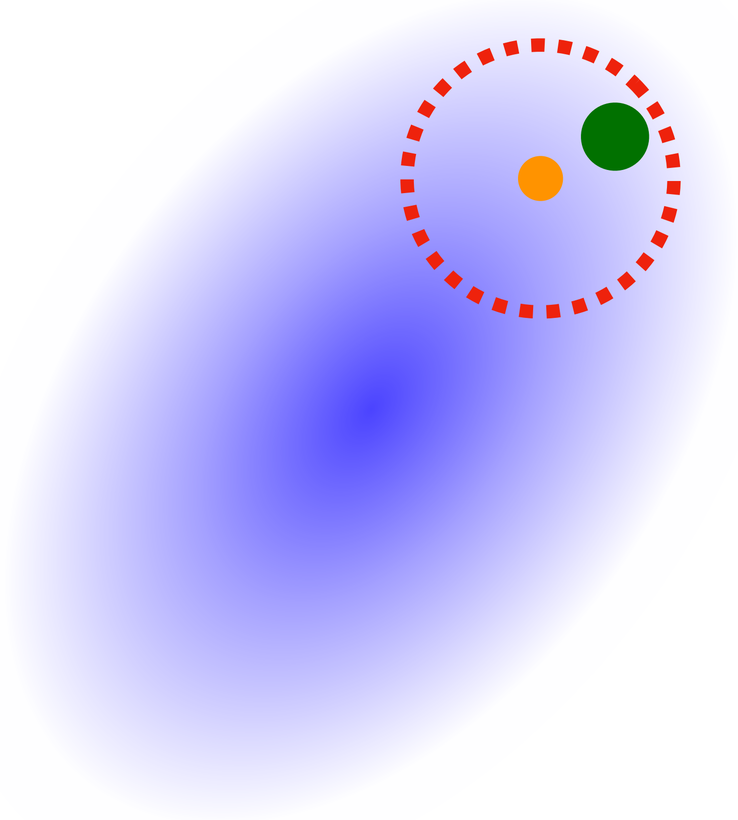
goal: given a target function $g(x) = e^{-\frac{f(x)}{T}}$, generate samples x_i s.t. $x_i \sim \frac{1}{Z} g(x)$



idea: start from a random initial sample x0

Markov Chain Monte Carlo (MCMC)

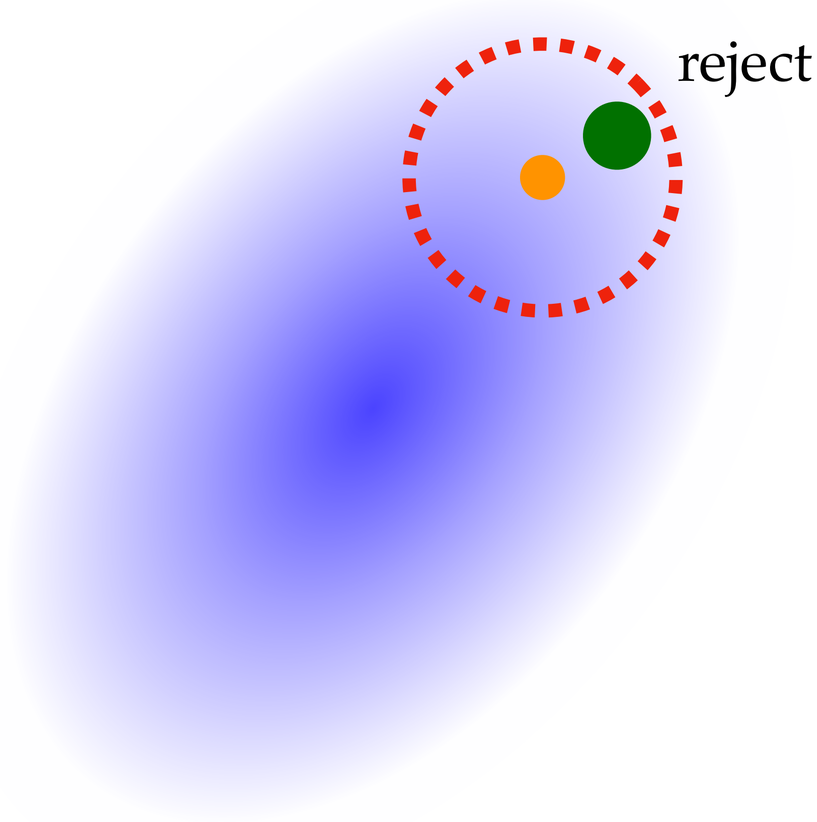
goal: given a target function $g(x) = e^{-\frac{f(x)}{T}}$, generate samples x_i s.t. $x_i \sim \frac{1}{Z} g(x)$



idea: start from a random initial sample x0
mutate it by generating a *candidate* sample x'
(usually using a normal distribution)

Markov Chain Monte Carlo (MCMC)

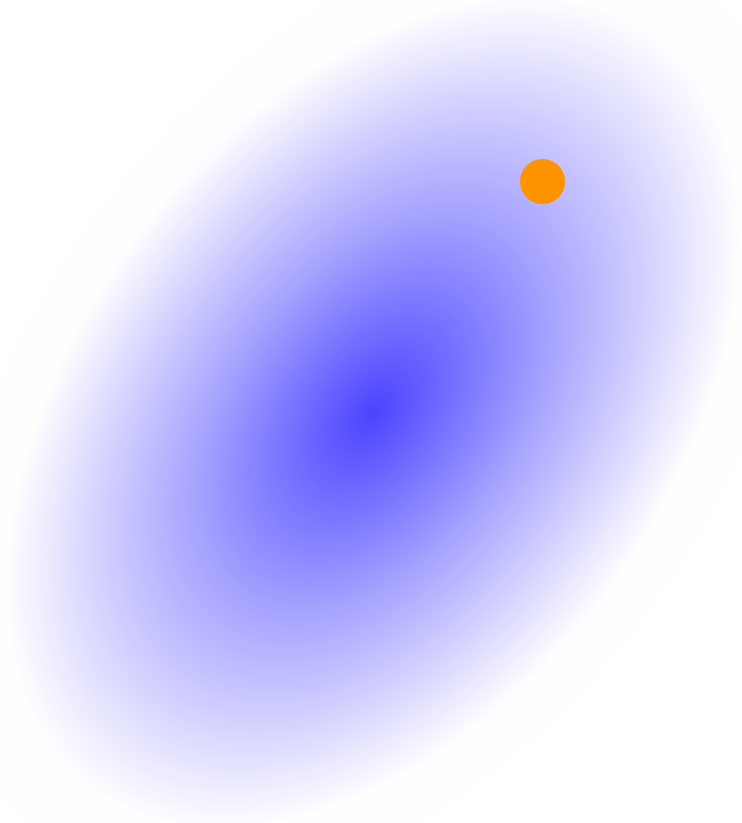
goal: given a target function $g(x) = e^{-\frac{f(x)}{T}}$, generate samples x_i s.t. $x_i \sim \frac{1}{Z} g(x)$



idea: start from a random initial sample x0
mutate it by generating a *candidate* sample x'
(usually using a normal distribution)
probabilistically accept/reject x' based on g(x')

Markov Chain Monte Carlo (MCMC)

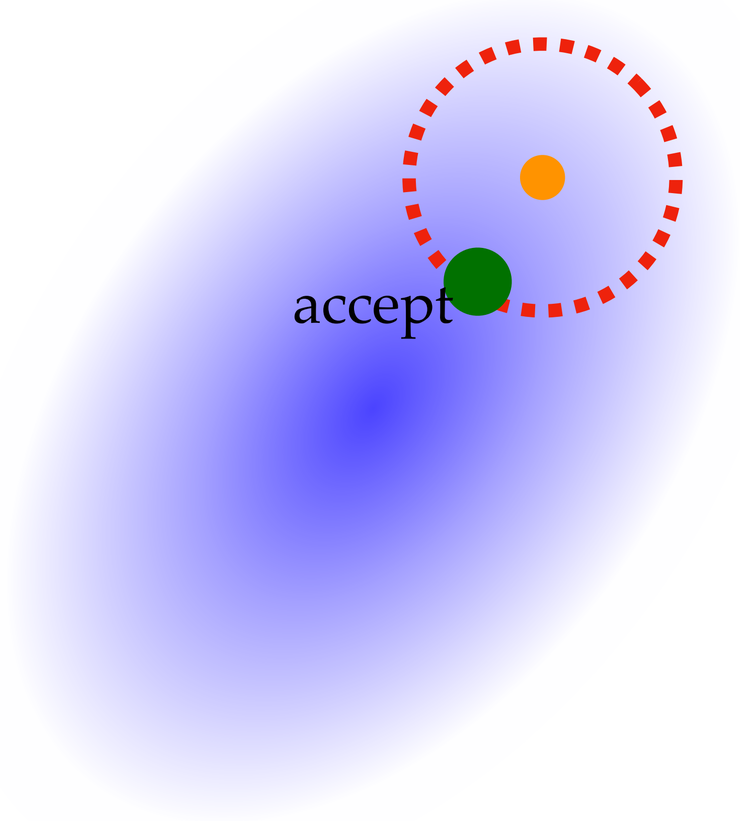
goal: given a target function $g(x) = e^{-\frac{f(x)}{T}}$, generate samples x_i s.t. $x_i \sim \frac{1}{Z} g(x)$



idea: start from a random initial sample x0
mutate it by generating a *candidate* sample x'
(usually using a normal distribution)
probabilistically accept/reject x' based on g(x')
if accept, $x_{i+1} = x'$
otherwise $x_{i+1} = x_i$

Markov Chain Monte Carlo (MCMC)

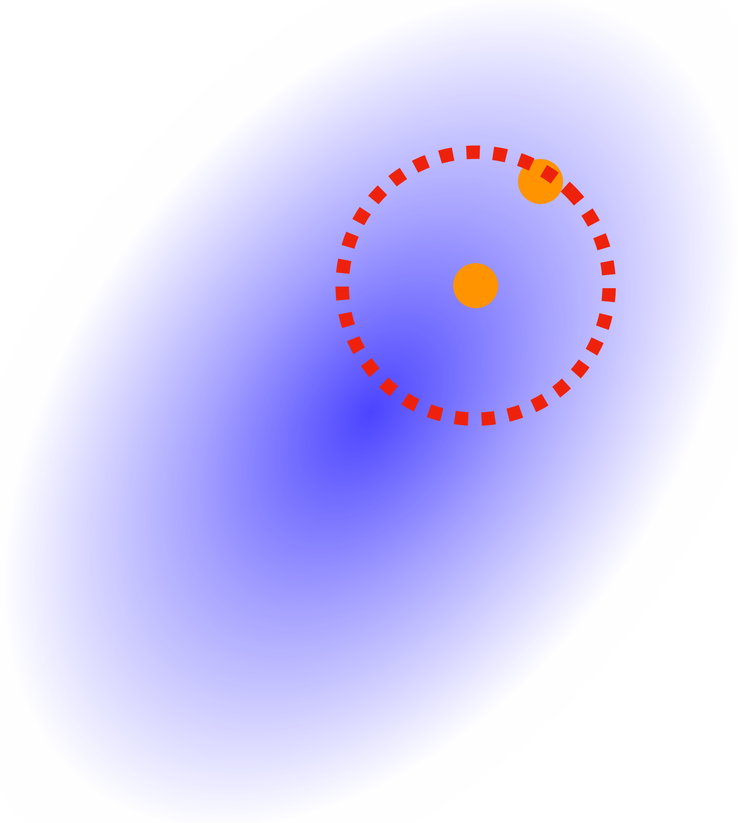
goal: given a target function $g(x) = e^{-\frac{f(x)}{T}}$, generate samples x_i s.t. $x_i \sim \frac{1}{Z} g(x)$



idea: start from a random initial sample x0
mutate it by generating a *candidate* sample x'
(usually using a normal distribution)
probabilistically accept/reject x' based on g(x')
if accept, $x_{i+1} = x'$
otherwise $x_{i+1} = x_i$

Markov Chain Monte Carlo (MCMC)

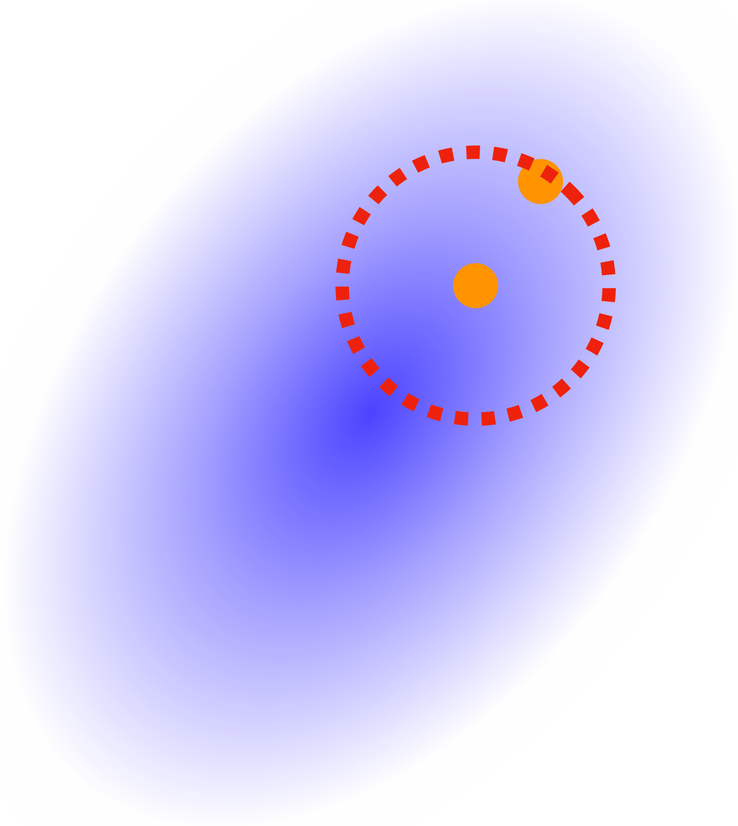
goal: given a target function $g(x) = e^{-\frac{f(x)}{T}}$, generate samples x_i s.t. $x_i \sim \frac{1}{Z} g(x)$



idea: start from a random initial sample x0
mutate it by generating a *candidate* sample x'
(usually using a normal distribution)
probabilistically accept/reject x' based on g(x')
if accept, $x_{i+1} = x'$
otherwise $x_{i+1} = x_i$

Markov Chain Monte Carlo (MCMC)

goal: given a target function $g(x) = e^{-\frac{f(x)}{T}}$, generate samples x_i s.t. $x_i \sim \frac{1}{Z} g(x)$



the Metropolis-Hasting algorithm

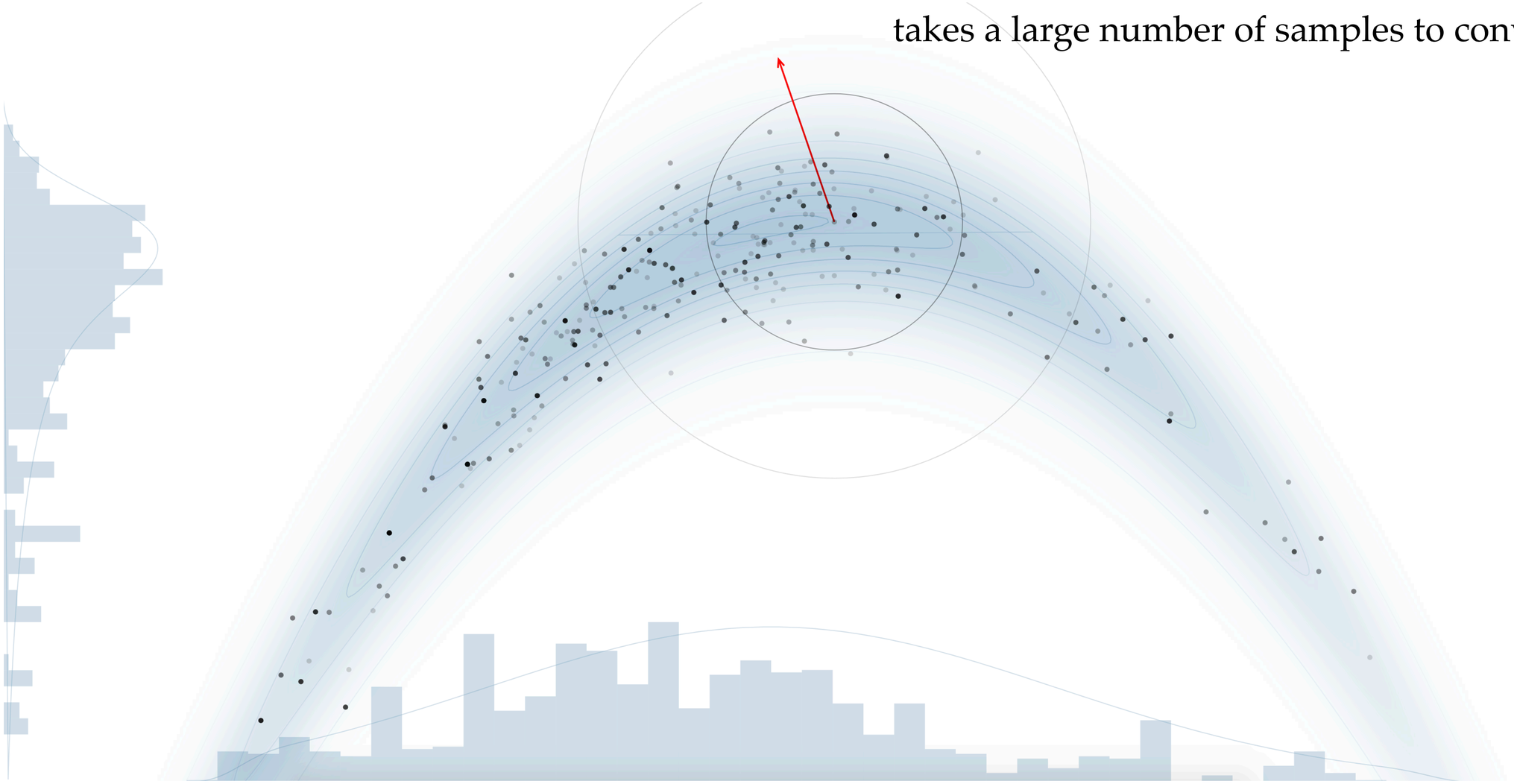
```
x = x0
for i in range(n):
  x' = mutate(x)
  a = min((g(x')/g(x)) *
          (p_m(x'->x)/p_m(x->x')), 1)
  if random() < a:
    x = x'
  samples.append(x)
```

MCMC in action

https://chi-feng.github.io/mcmc-demo/app.html?algorithm=RandomWalkMH&target=standard

Vanilla MCMC is very slow

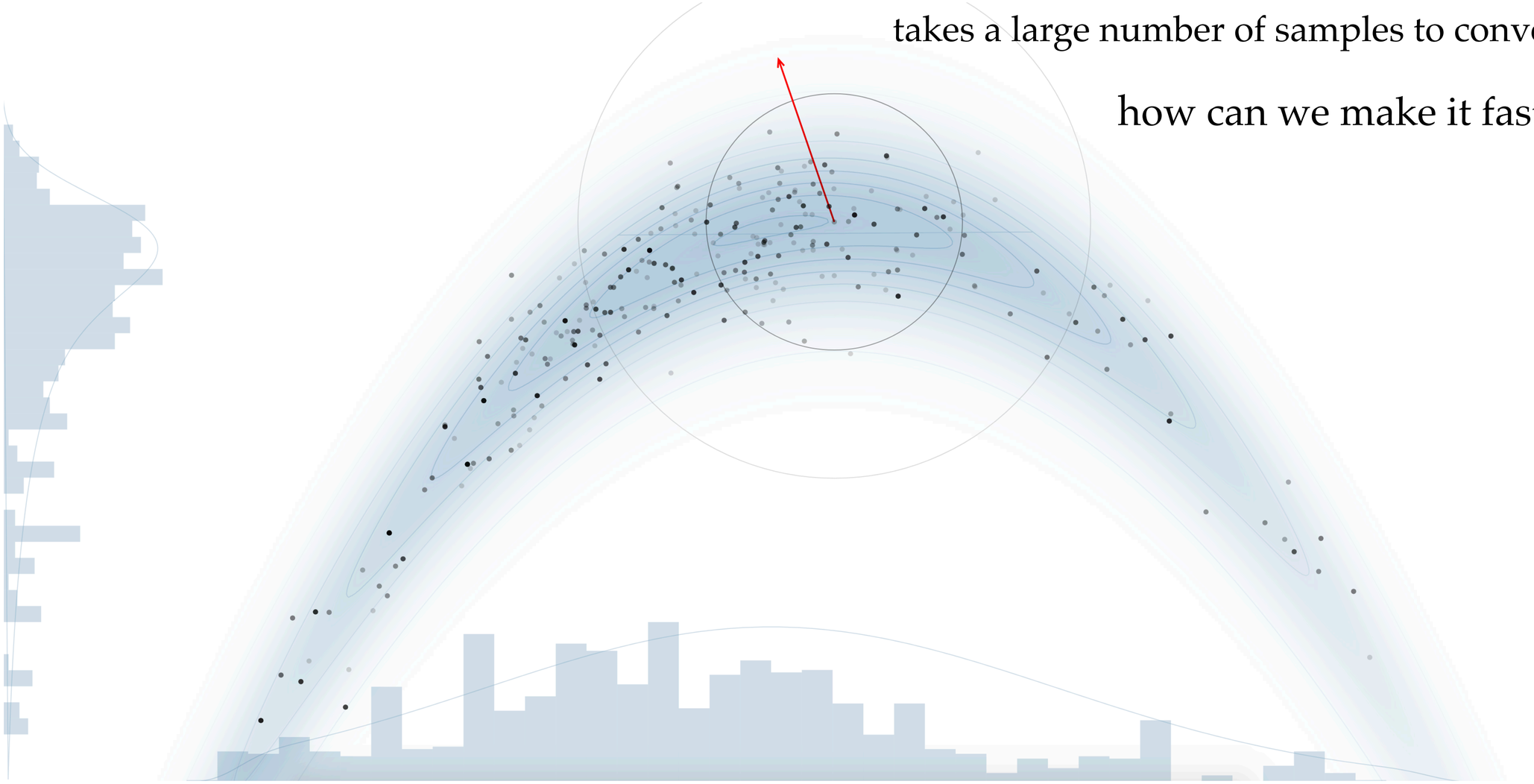
takes a large number of samples to converge to the target



Vanilla MCMC is very slow

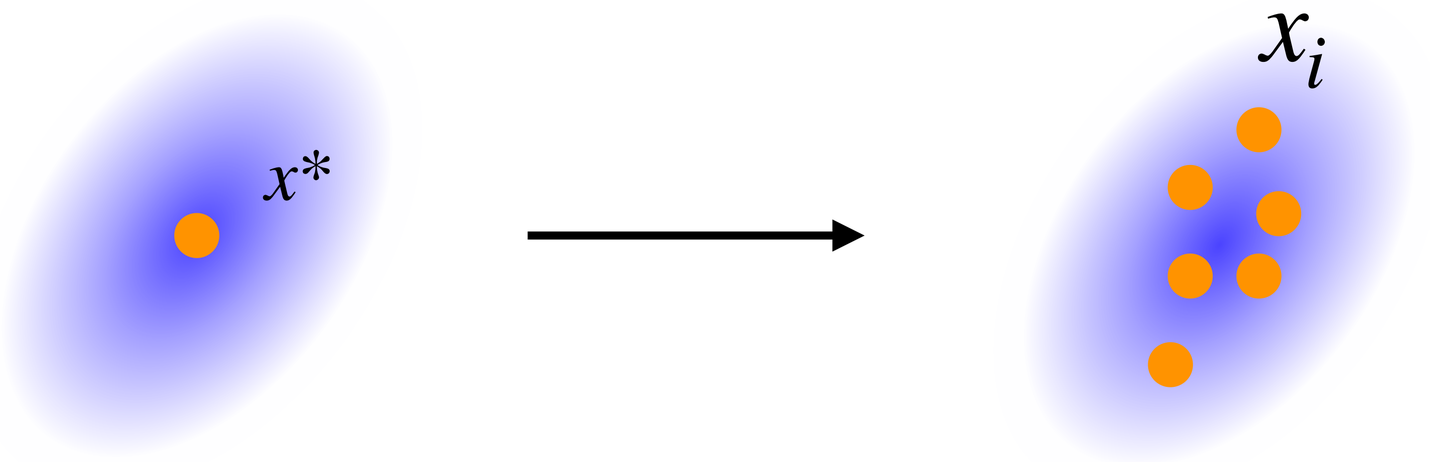
takes a large number of samples to converge to the target

how can we make it faster?



Key insight: we can take inspiration from **optimization** algorithms!

gradient descent
adaptive gradient methods
determinism
line search
Newton's method
…



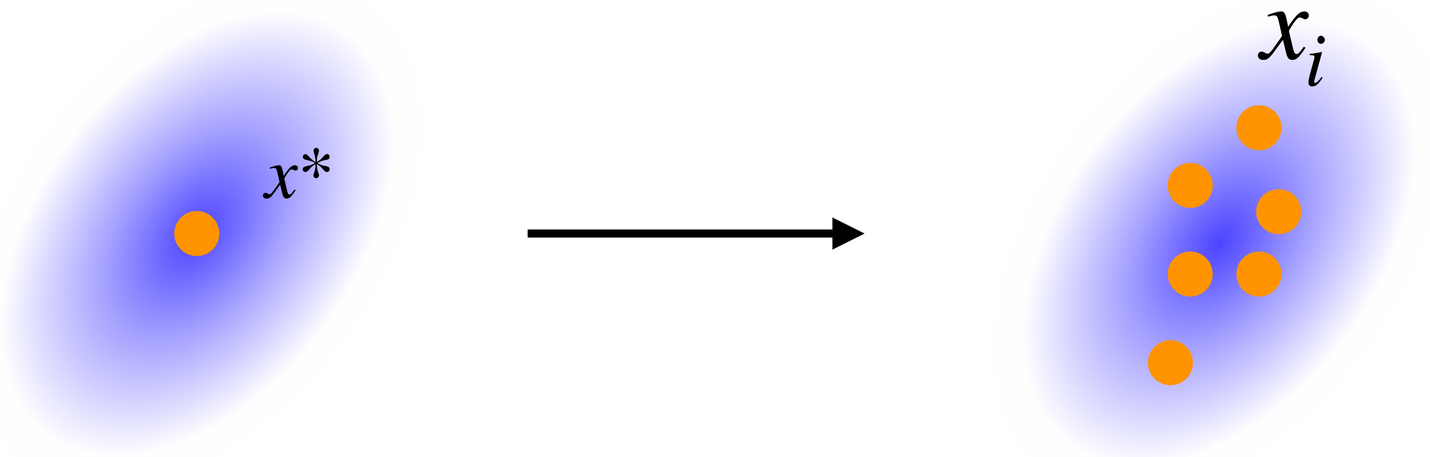
$$x^* = \arg\min_x f(x)$$

$$x_i \sim \frac{1}{Z} e^{-\frac{f(x_i)}{T}}$$

Key insight: we can take inspiration from **optimization** algorithms!

can we port gradient descent to MCMC sampling?

gradient descent
adaptive gradient methods
determinism
line search
Newton's method
…



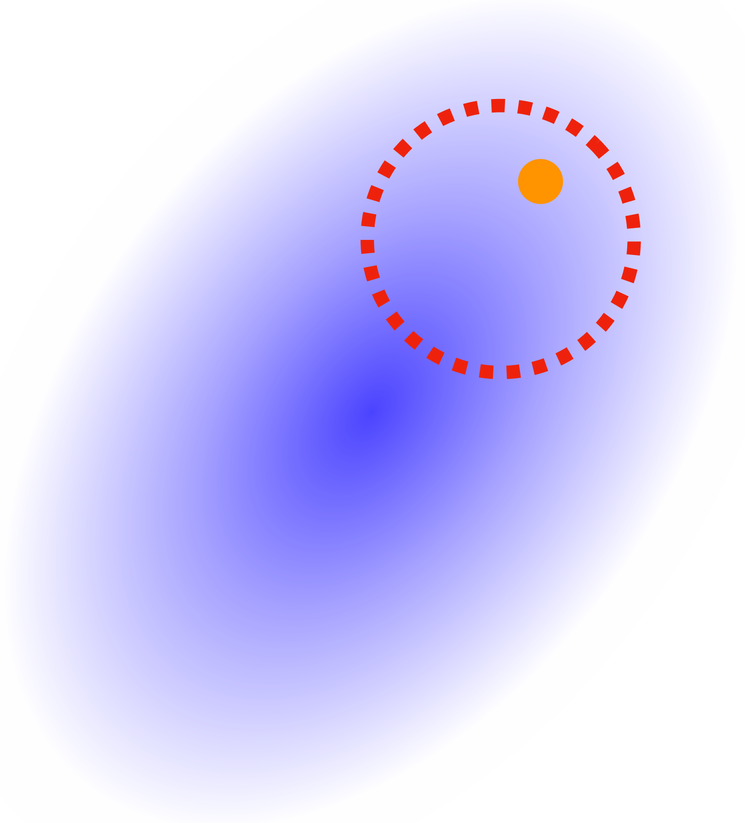
$$x^* = \arg\min_x f(x)$$

$$x_i \sim \frac{1}{Z} e^{-\frac{f(x_i)}{T}}$$

Langevin Monte Carlo
(or “Metropolis-adjusted Langevin algorithm”)

goal: given a target function $g(x) = e^{-\frac{f(x)}{T}}$, generate samples x_i s.t. $x_i \sim \frac{1}{Z} g(x)$

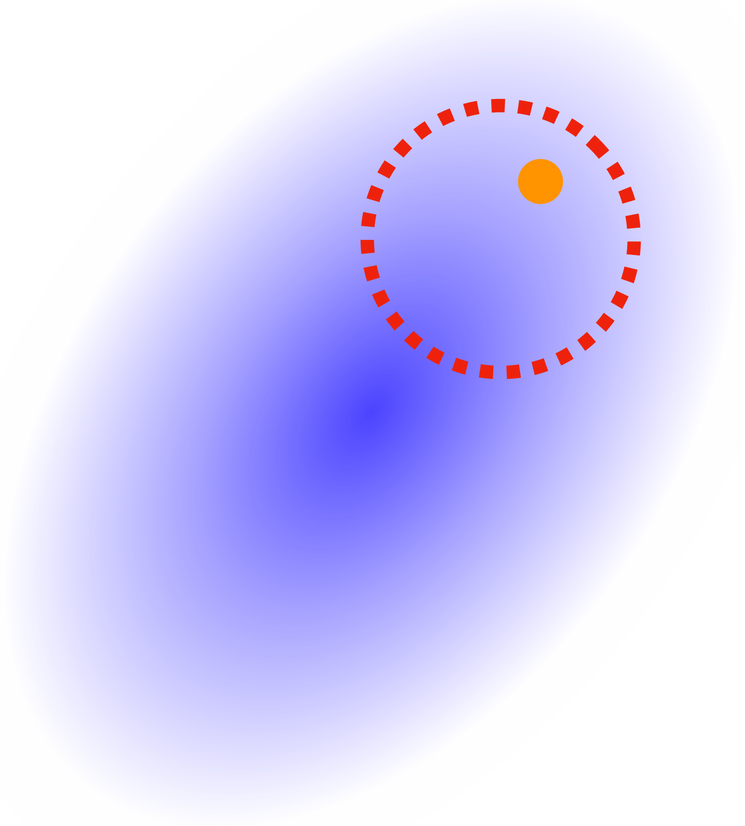
idea: shift the mean of the normal distribution towards high value regions using the gradient



Langevin Monte Carlo (or "Metropolis-adjusted Langevin algorithm")

goal: given a target function $g(x) = e^{-\frac{f(x)}{T}}$, generate samples x_i s.t. $x_i \sim \frac{1}{Z} g(x)$

idea: shift the mean of the normal distribution towards high value regions using the gradient

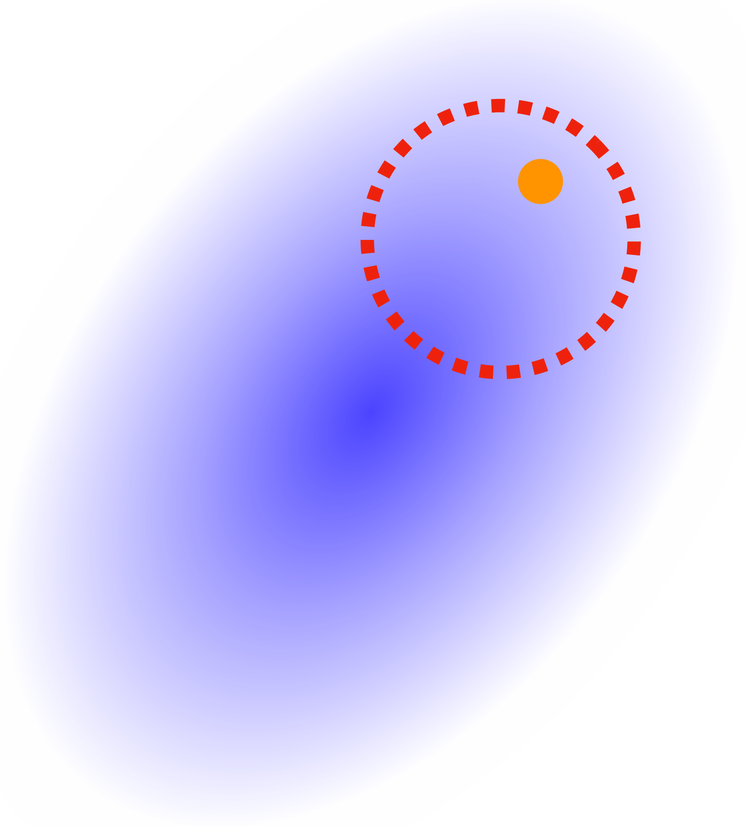


```
x = x0
for i in range(n):
  x' ~ N(x-α∇f/T, σ²I)
  a = min((g(x')/g(x)) *
          (p_m(x'->x)/p_m(x->x')), 1)
  if random() < a:
    x = x'
  samples.append(x)
```

Langevin Monte Carlo (or "Metropolis-adjusted Langevin algorithm")

goal: given a target function $g(x) = e^{-\frac{f(x)}{T}}$, generate samples x_i s.t. $x_i \sim \frac{1}{Z} g(x)$

idea: shift the mean of the normal distribution towards high value regions using the gradient



should we use $-\nabla f$ or ∇g?

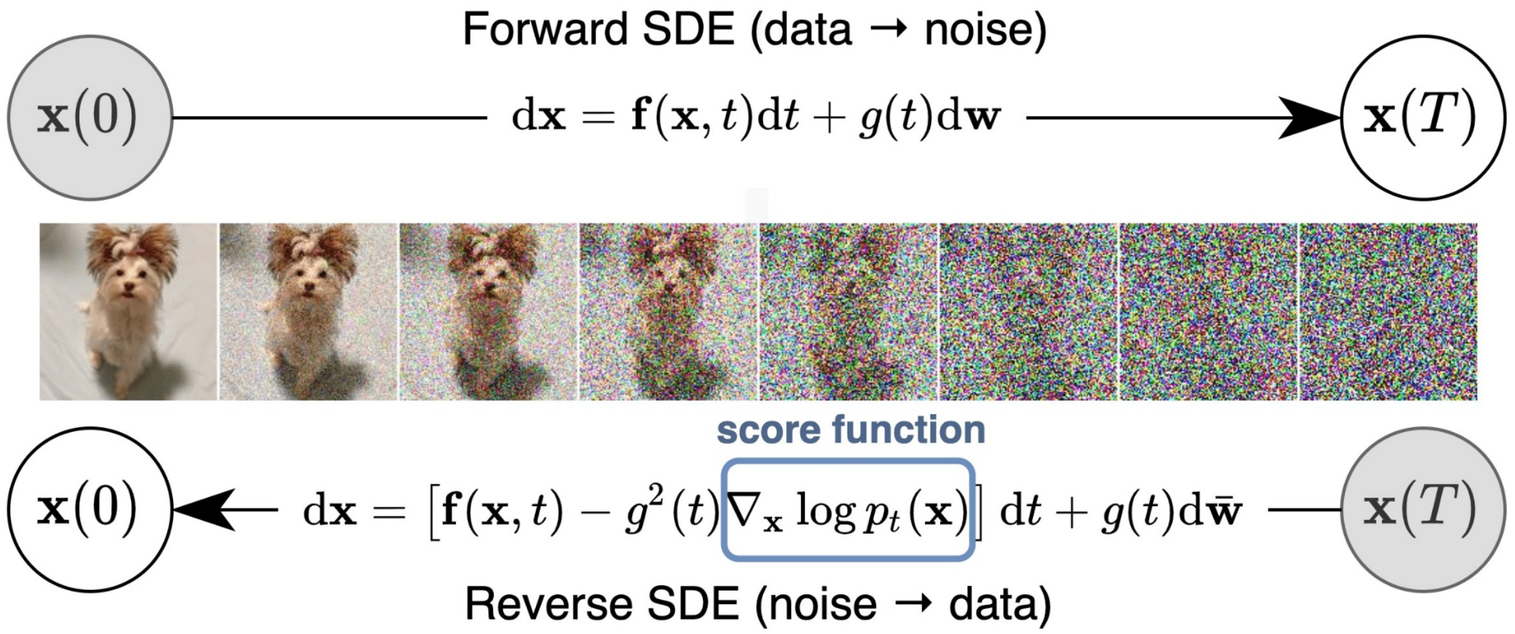
```
x = x0
for i in range(n):
  x' ~ N(x-α∇f/T, σ²I)
  a = min((g(x')/g(x)) *
          (p_m(x'->x)/p_m(x->x')), 1)
  if random() < a:
    x = x'
  samples.append(x)
```

LMC/MALA in action

https://chi-feng.github.io/mcmc-demo/app.html?algorithm=MALA&target=banana

LMC/MALA without accept/reject ~= diffusion models

```
x = x0
for i in range(n):
   x ~ N(x-α∇f, sigma)
return x
```



$$g(x) = e^{-\frac{f(x)}{T}} \longrightarrow \nabla \log g = -\nabla \frac{f}{T}$$

https://theaisummer.com/diffusion-models/

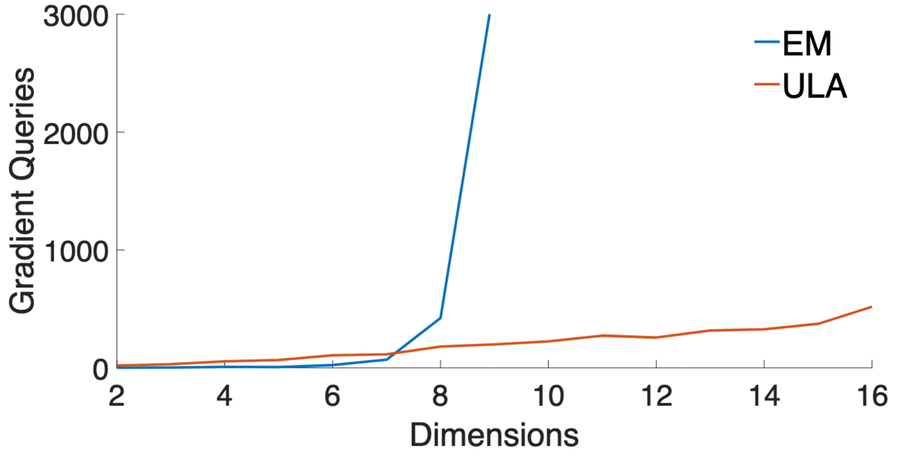
Sampling Can Be Faster Than Optimization

Yi-An Ma[*a], Yuansi Chen[†b], Chi Jin[‡a], Nicolas Flammarion[§a], and Michael I. Jordan[¶a, b]

[a]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720
[b]Department of Statistics, University of California, Berkeley, CA 94720

August 26, 2019

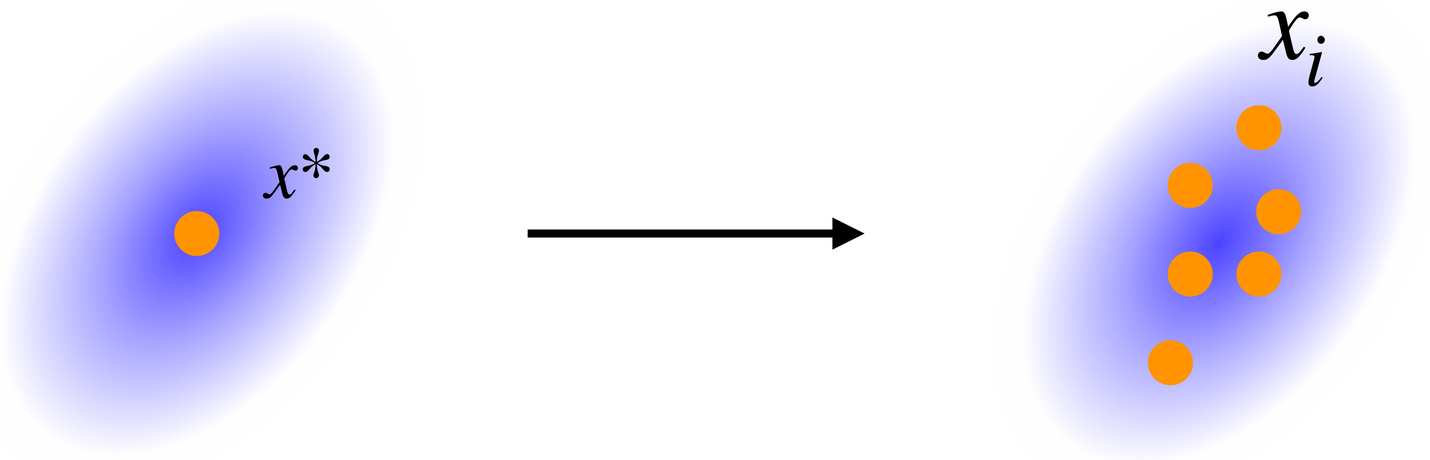
for optimizing certain classes of non-convex functions,
LMC/MALA converges **exponentially** faster in dimension than gradient descent!!



Key insight: we can take inspiration from **optimization** algorithms!

can we port adaptive gradient methods (e.g., Adam) to MCMC sampling?

gradient descent
adaptive gradient methods
determinism
line search
Newton's method
…

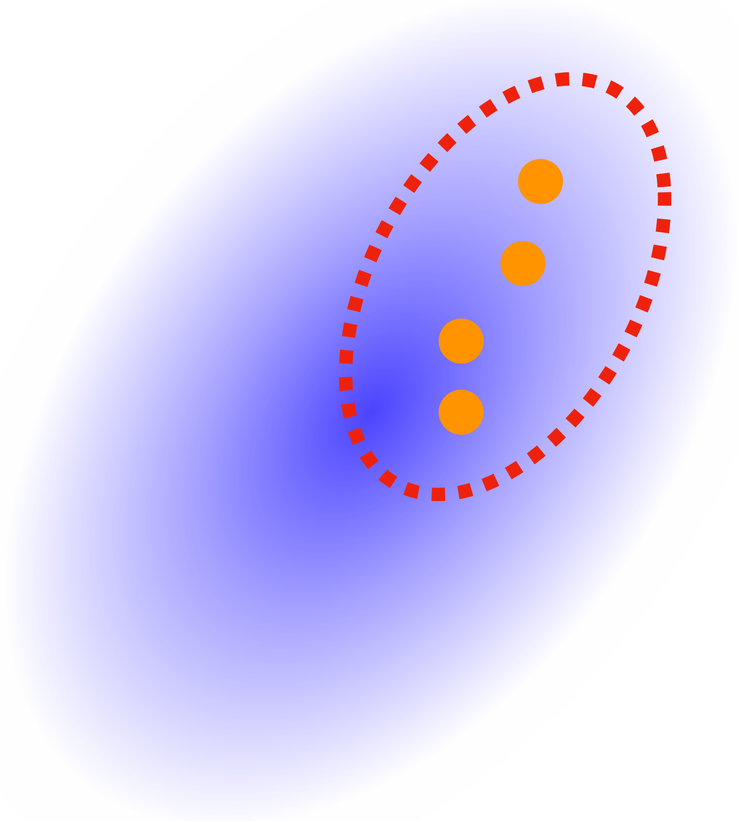


$$x^* = \arg\min_x f(x)$$

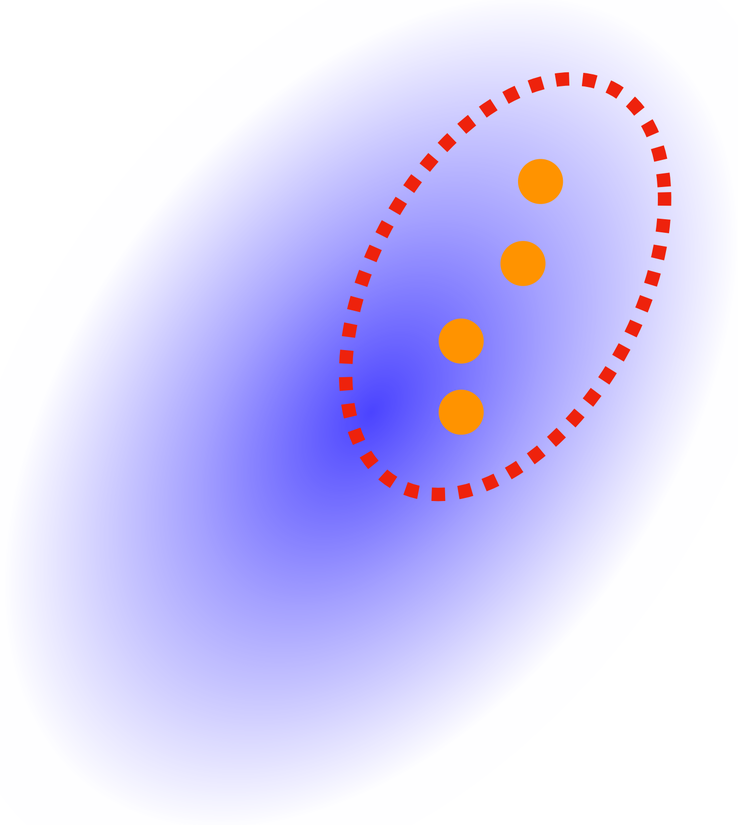
$$x_i \sim \frac{1}{Z} e^{-\frac{f(x_i)}{T}}$$

Adaptive MCMC methods

idea: fit a normal distribution using values from previous samples



Adaptive MCMC methods



idea: fit a normal distribution using values from previous samples

converges if β_i, γ_i goes to 0
(aka diminishing adaptation)

```
x = x0
g = 0, H = 1/σ² I
for i in range(n):
  g = (1-β_i)*g + β_i ∇f/T
  H = (1-γ_i)*H + γ_i (∇f · ∇f^T)/(T*T)
  x' ~ N(x-α*m, H^-1)
  a = min((g(x')/g(x)) *
          (p_m(x'->x)/p_m(x->x')), 1)
  if random() < a:
    x = x'
  samples.append(x)
```

Adaptive MCMC in action

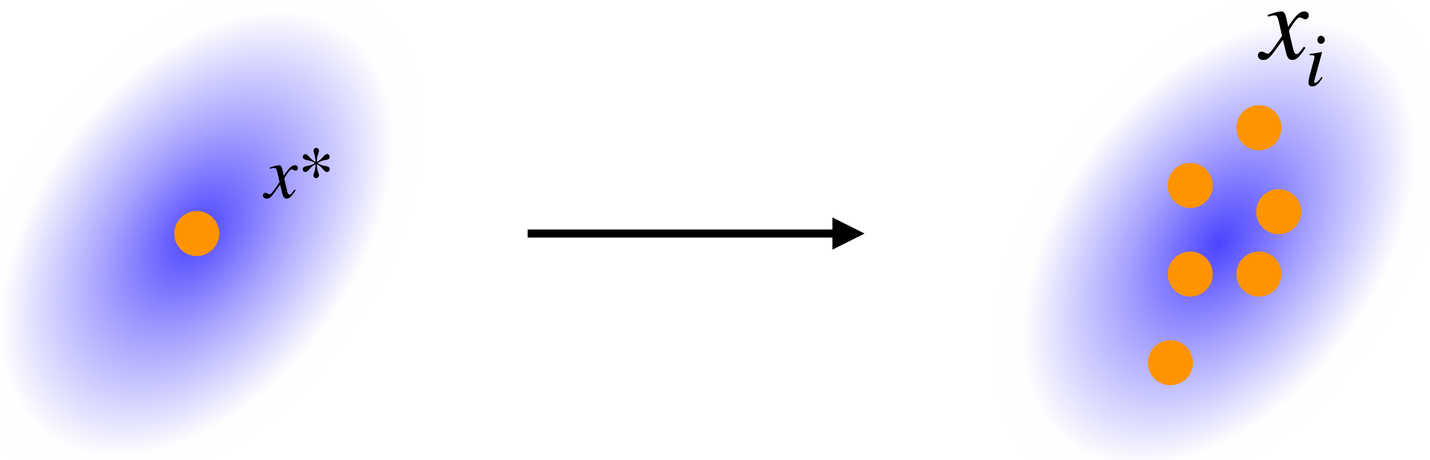
https://chi-feng.github.io/mcmc-demo/app.html?algorithm=AdaptiveMH&target=banana

this shows the algorithm from Haario 2001, which didn't use the gradients, but the idea is the same

Key insight: we can take inspiration from **optimization** algorithms!

can we make MCMC sampling less random?

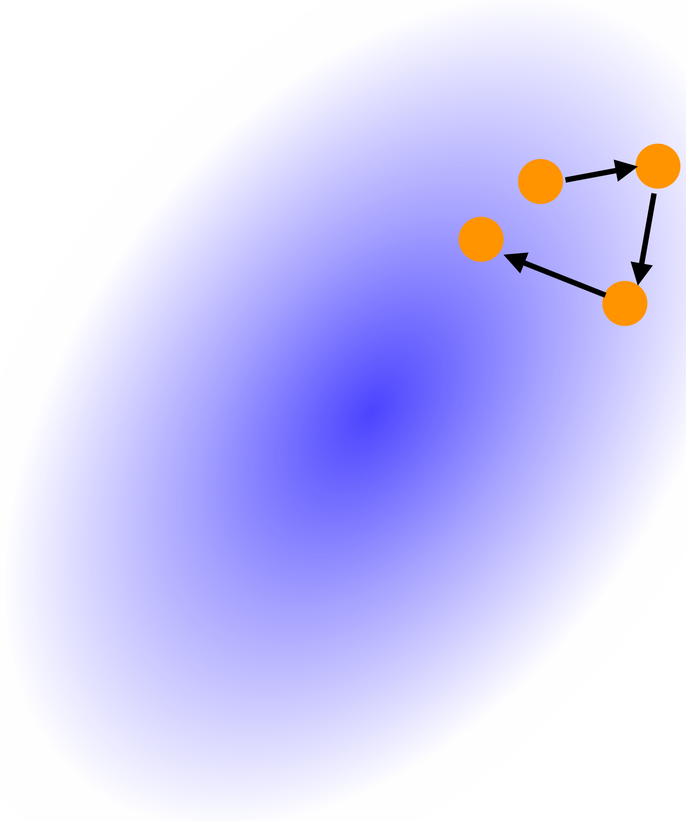
gradient descent
adaptive gradient methods
determinism
line search
Newton's method
…



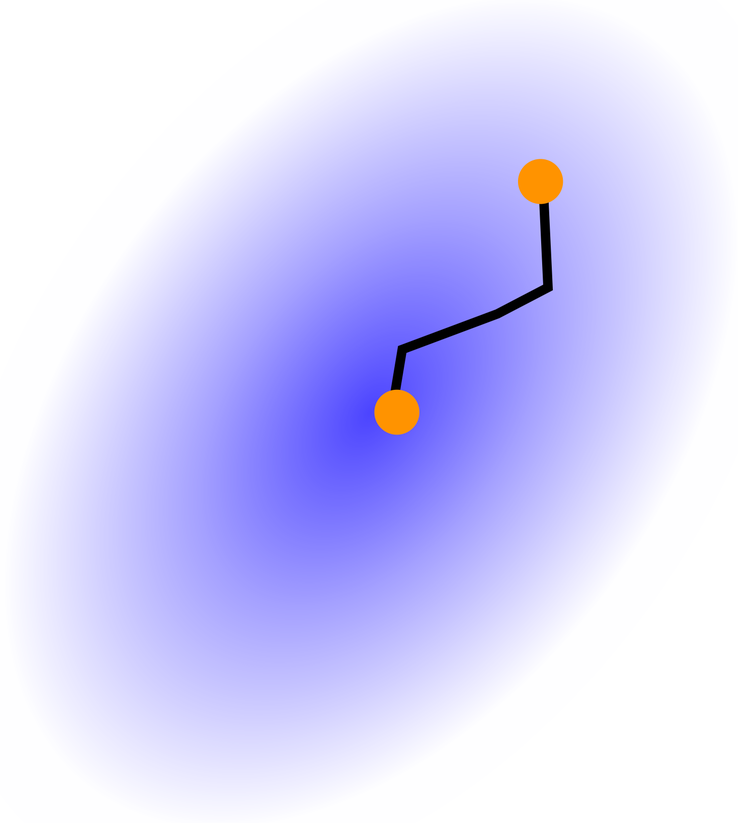
$$x^* = \arg\min_x f(x)$$

$$x_i \sim \frac{1}{Z} e^{-\frac{f(x_i)}{T}}$$

The randomness in MCMC can make convergence slow



Hamiltonian Monte Carlo

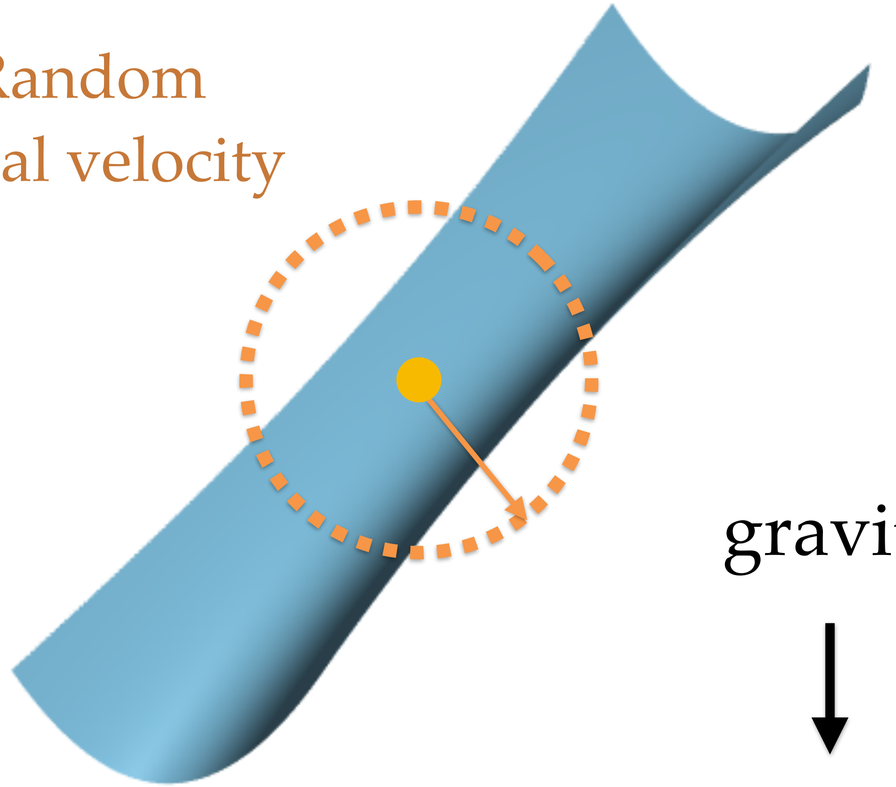


idea: do deterministic optimization for a number of steps
before probabilistically accept/reject

Hamiltonian Monte Carlo

flip the target function g upside down
choose a random initial velocity

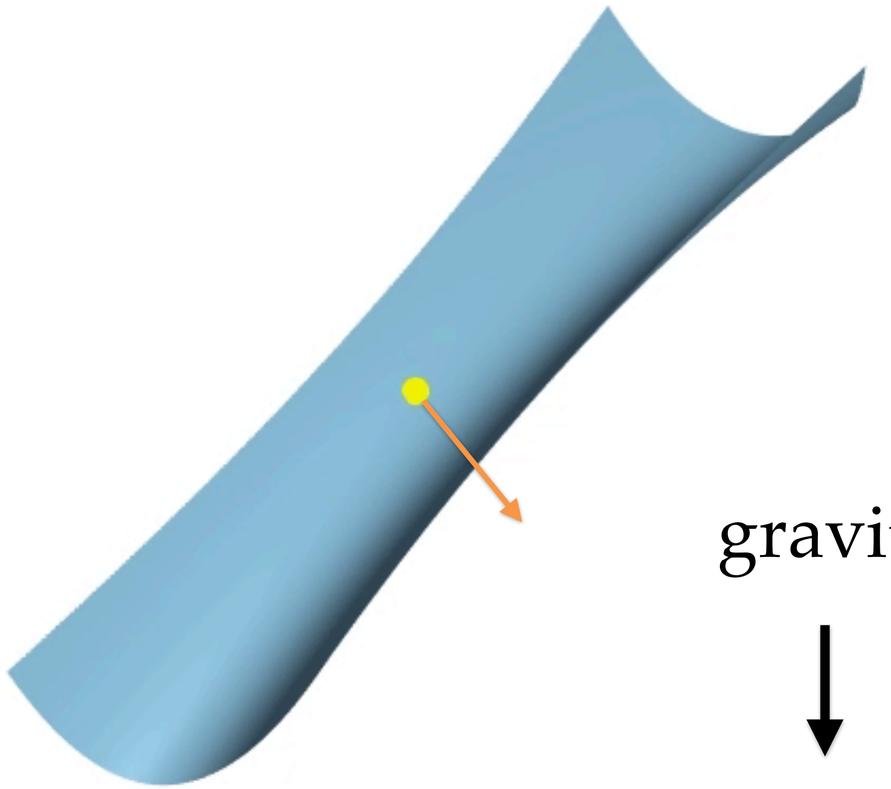
Random
initial velocity



gravity

flipped target function

Hamiltonian Monte Carlo

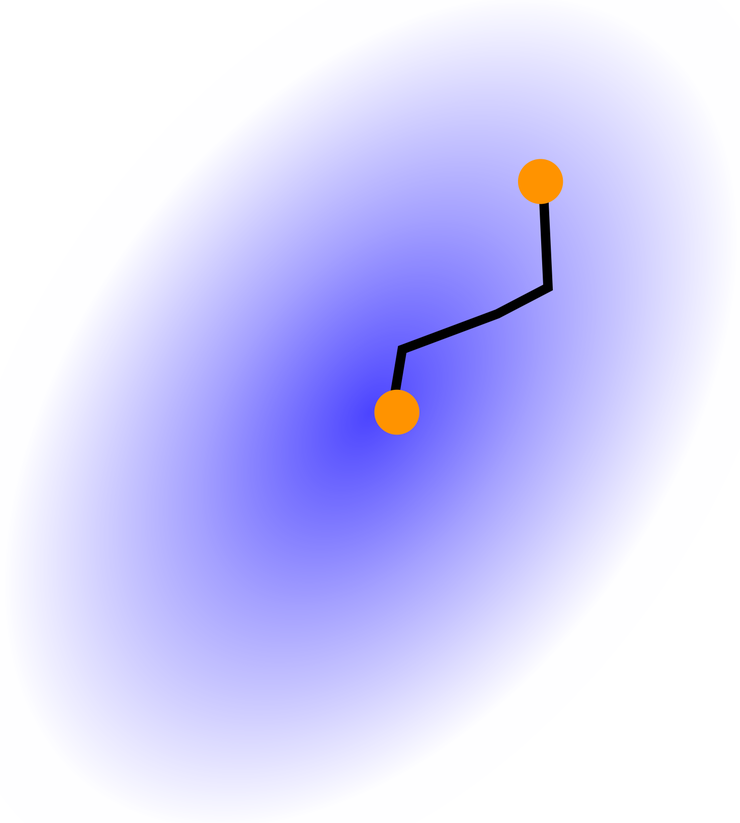


flip the target function g upside down
choose a random initial velocity
simulate physics under gravity
  particle is pulled to low ground (high prob.)
candidate is the final position

gravity

flipped target function

Hamiltonian Monte Carlo



```
x = x0
for i in range(n):
  v ~ N(0, sigma)
  x' = x, v' = v
  for j in range(L):
    v' = v' - α*∇f(x')/T
    x' = x' + α*v'
  a = min((g(x')/g(x)) *
          (p_m(v')/p_m(v)), 1)
  if random() < a:
    x = x'
  samples.append(x)
```

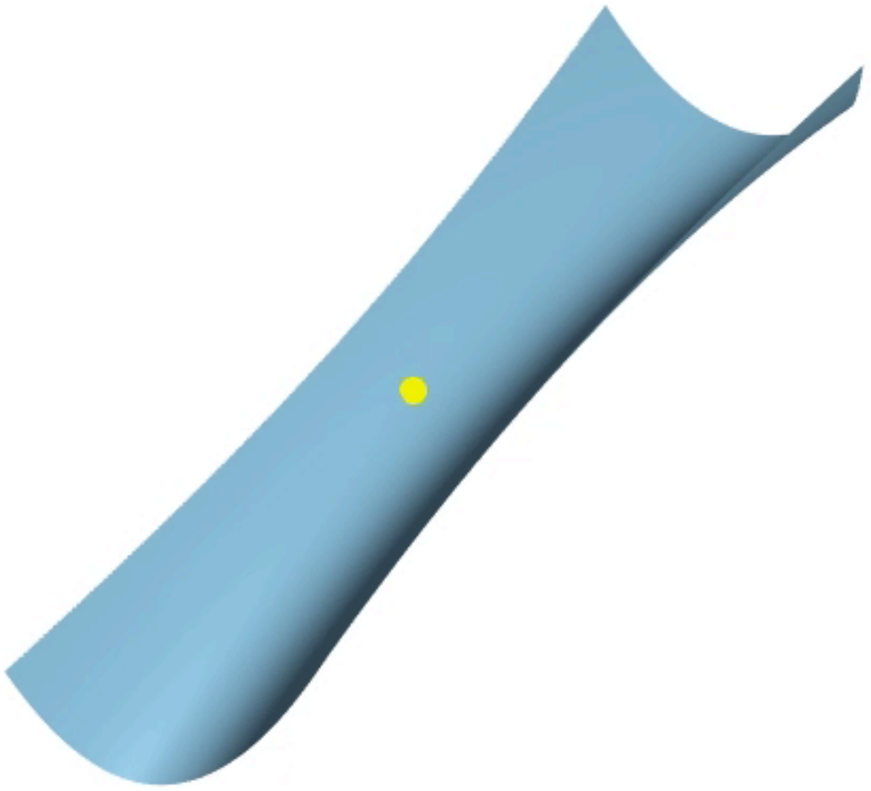
gravity

(in practice, people use Leapfrog integration instead of symplectic Euler)

HMC in action

https://chi-feng.github.io/mcmc-demo/app.html?algorithm=HamiltonianMC&target=banana

A cool theoretical property of HMC



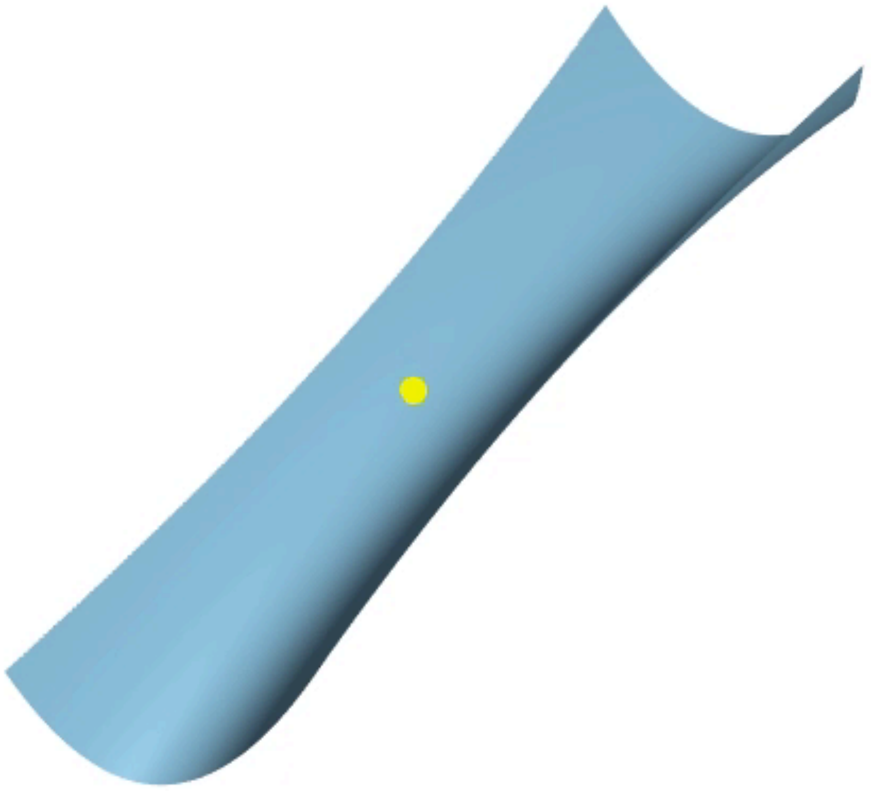
the gravity physical simulation, if simulated perfectly, preserves energy

$$H = \frac{f(x)}{T} + \frac{1}{2}v^T v$$

potential energy

kinetic energy

A cool theoretical property of HMC



the gravity physical simulation, if simulated perfectly, preserves energy

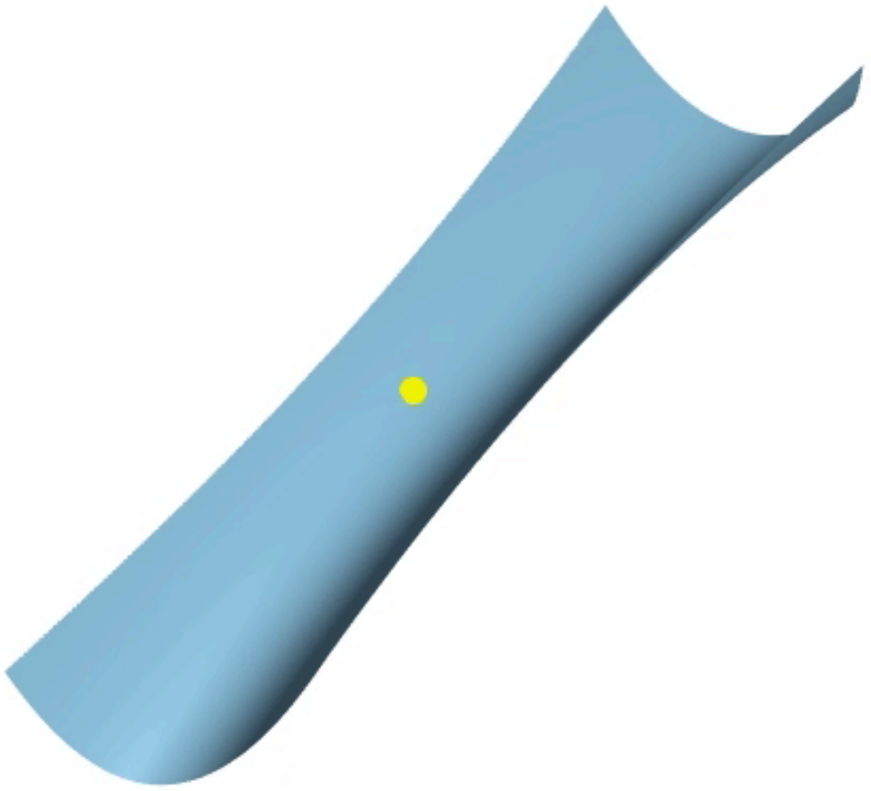
$$H = \frac{f(x)}{T} + \frac{1}{2}v^T v$$

potential energy

kinetic energy

thus, the value of e^{-H} is preserved too

A cool theoretical property of HMC



the gravity physical simulation, if simulated perfectly, preserves energy

$$H = \frac{f(x)}{T} + \frac{1}{2}v^T v$$

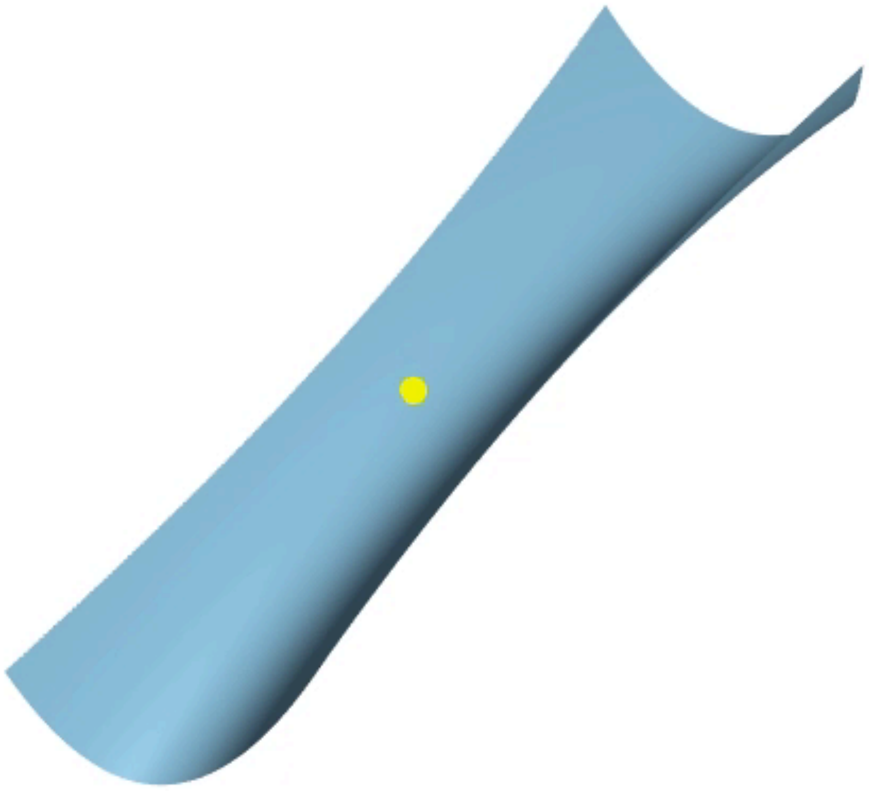
potential energy

kinetic energy

thus, the value of e^{-H} is preserved too

the acceptance rate $a = e^{-H'/-H} = 1$!!!

A cool theoretical property of HMC



in practice, this means infinitely small timesteps (α),
so we never get perfect HMC

the gravity physical simulation, if simulated perfectly, preserves energy

$$H = \frac{f(x)}{T} + \frac{1}{2}v^T v$$

potential energy

kinetic energy

thus, the value of e^{-H} is preserved too

the acceptance rate $a = e^{-H'/-H} = 1$!!!

One step HMC = LMC/MALA

HMC

```
x = x0
for i in range(n):
  v ~ N(0, sigma)
  x' = x, v' = v
  for j in range(L):
    v' = v' - α*∇f(x')/T
    x' = x' + α*v'
  a = min((g(x')/g(x)) *
          (p_m(v')/p_m(v)), 1)
  if random() < a:
    x = x'
  samples.append(x)
```

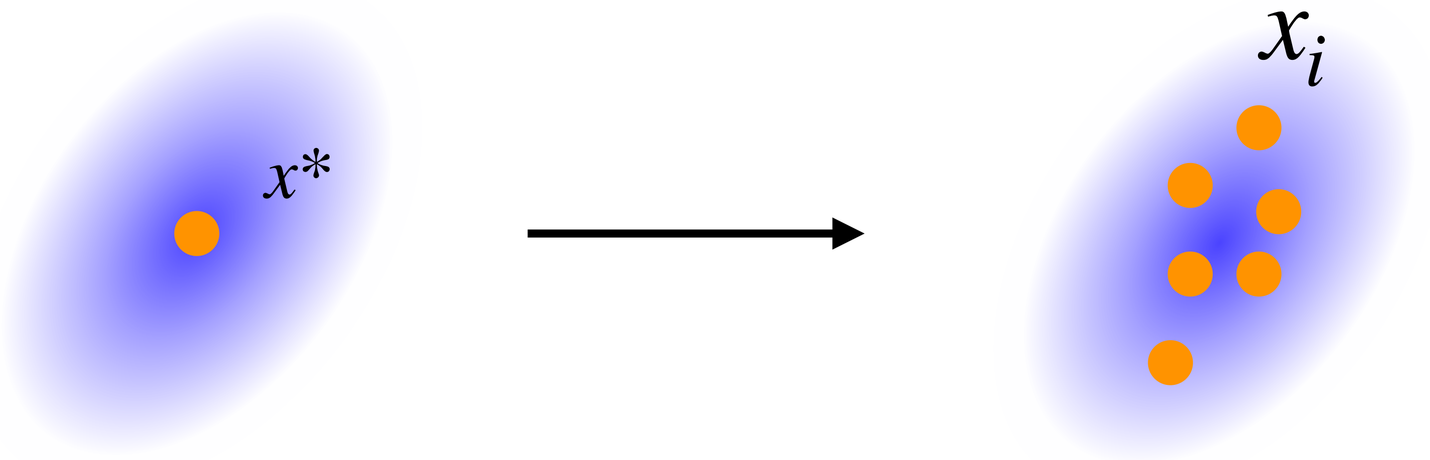
LMC/MALA

```
x = x0
for i in range(n):
  x' ~ N(x-α∇f/T, σ²I)
  a = min((g(x')/g(x)) *
          (p_m(x'->x)/p_m(x->x')), 1)
  if random() < a:
    x = x'
  samples.append(x)
```

Key insight: we can take inspiration from **optimization** algorithms!

how do we determine the integration length of HMC?

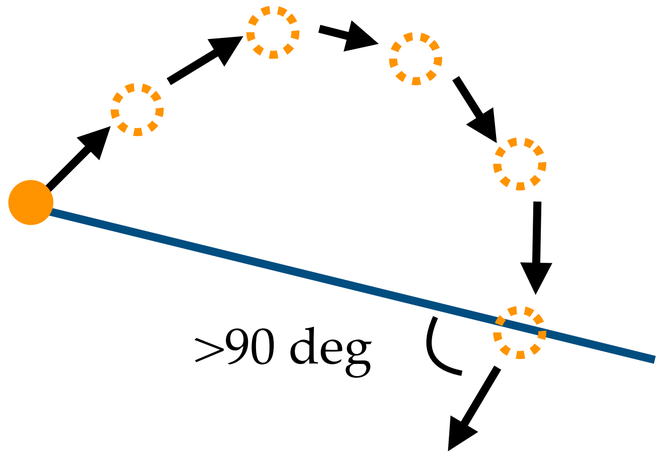
gradient descent
adaptive gradient methods
determinism
line search
Newton's method
…



$$x^* = \arg\min_x f(x)$$

$$x_i \sim \frac{1}{Z} e^{-\frac{f(x_i)}{T}}$$

NUTS (No U-Turn Sampling)



idea: simulate physics in HMC until the candidate is starting to make a “U-turn”

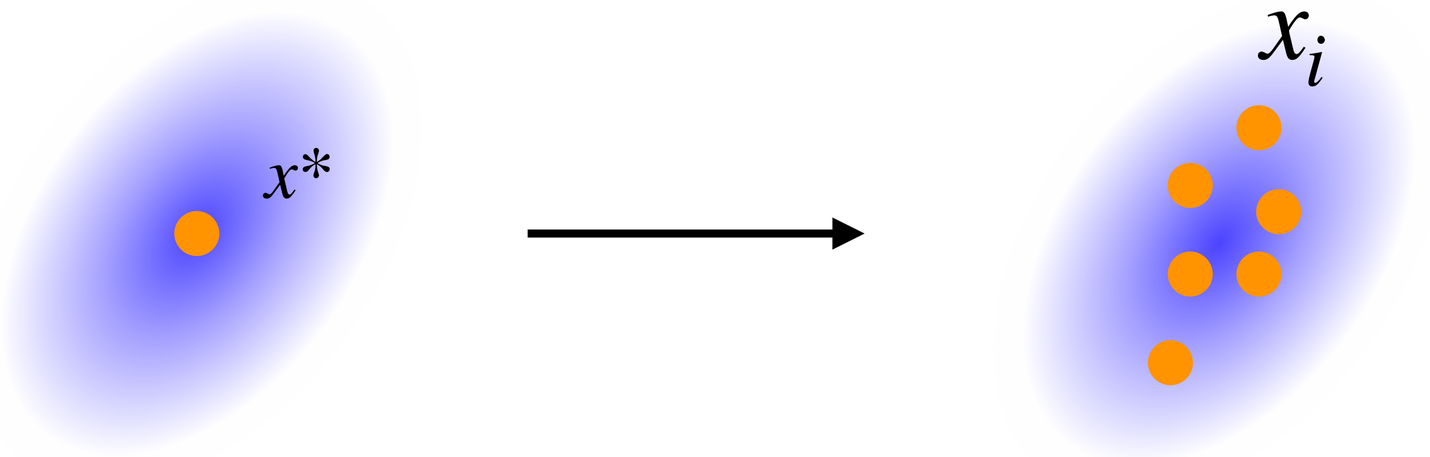
NUTS in action

https://chi-feng.github.io/mcmc-demo/app.html?algorithm=EfficientNUTS&target=banana

Key insight: we can take inspiration from **optimization** algorithms!

can we use 2nd derivatives in MCMC?

gradient descent
adaptive gradient methods
determinism
line search
Newton's method
…



$$x^* = \arg\min_x f(x)$$

$$x_i \sim \frac{1}{Z} e^{-\frac{f(x_i)}{T}}$$

Add Hessian information by making it the mass matrix

extra energy since we change mass over time

$$H = f(x)/T + \frac{1}{2}\log\left((2\pi)^N \left|G(x)\right|\right) + \frac{1}{2}v^T G(x) v$$

G(x) is the Hessian matrix of f at x, projected to positive semi-definite

```
x = x0
for i in range(n):
  v ~ N(0, G(x)^-1)
  x' = x, v' = v
  for j in range(L):
    v' = v' - α*∂H/∂x
    x' = x' + α*∂H/∂v
  a = min((g(x')/g(x)) *
          (p_m(v')/p_m(v)), 1)
  if random() < a:
    x = x'
  samples.append(x)
```

Add Hessian information by making it the mass matrix

extra energy since we change mass over time

$$H = f(x)/T + \frac{1}{2}\log\left((2\pi)^N \left|G(x)\right|\right) + \frac{1}{2}v^T G(x) v$$

G(x) is the Hessian matrix of f at x, projected to positive semi-definite

note that $\frac{\partial H}{\partial x}$ requires us to compute the derivative of Hessian's determinant!

```
x = x0
for i in range(n):
  v ~ N(0, G(x)^-1)
  x' = x, v' = v
  for j in range(L):
    v' = v' - α*∂H/∂x
    x' = x' + α*∂H/∂v
  a = min((g(x')/g(x)) *
          (p_m(v')/p_m(v)), 1)
  if random() < a:
    x = x'
  samples.append(x)
```

See this seminal paper for details

Riemann Manifold Langevin and Hamiltonian Monte Carlo

Mark Girolami
Department of Computing Science, University of Glasgow, UK
Department of Statistics, University of Glasgow, UK
Ben Calderhead
Department of Computing Science, University of Glasgow, UK
Siu A. Chin
Department of Physics, Texas A&M University, Texas, USA

My Hessian-based LMC/MALA in action

doesn't require 3rd order derivatives, but not HMC

https://chi-feng.github.io/mcmc-demo/app.html?algorithm=H2MC&target=banana

Anisotropic Gaussian Mutations for Metropolis Light Transport through Hessian-Hamiltonian Dynamics

Tzu-Mao Li
MIT CSAIL

Jaakko Lehtinen
Aalto University
NVIDIA

Ravi Ramamoorthi
University of California, San Diego

Wenzel Jakob
ETH Zürich

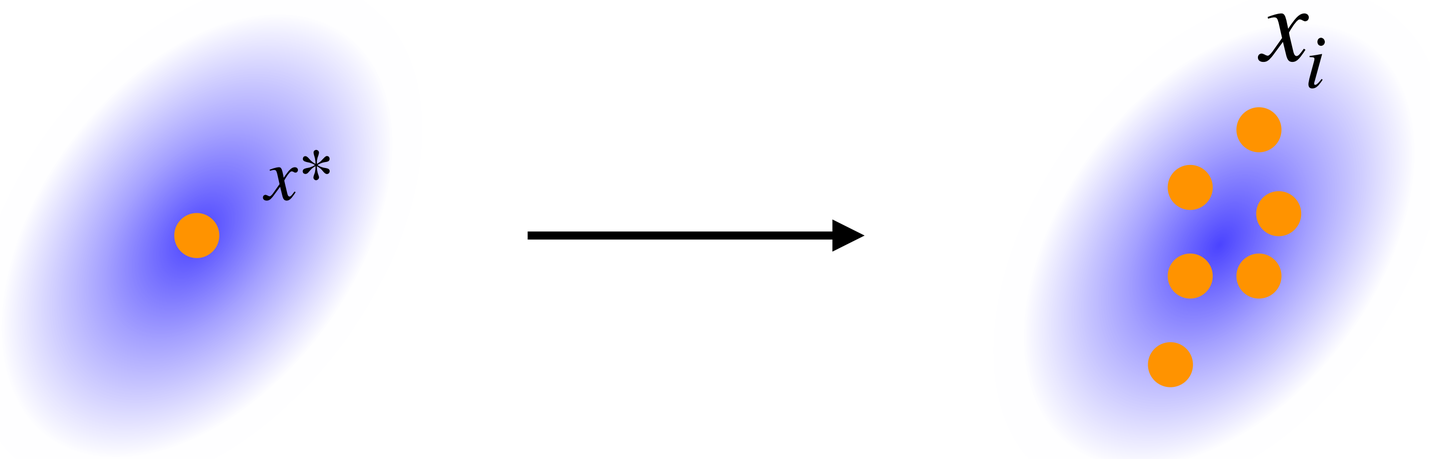
Frédo Durand
MIT CSAIL

Take home msg 1: sampling is useful!

- sampling generalizes optimization
- sampling acts as an regularization in high-dimensional space
- sampling estimates uncertainty

Take home msg 2: sampling can borrow techniques from optimization

gradient descent
adaptive gradient methods
determinism
line search
Newton's method
…



$$x^* = \arg\min_x f(x)$$

$$x_i \sim \frac{1}{Z} e^{-\frac{f(x_i)}{T}}$$

Software

- Stan https://mc-stan.org/
  - check out https://mc-stan.org/users/documentation/case-studies/boarding_school_case_study.html for a case study in epidemiology!
- PyMC https://www.pymc.io/welcome.html
- Pyro https://pyro.ai/