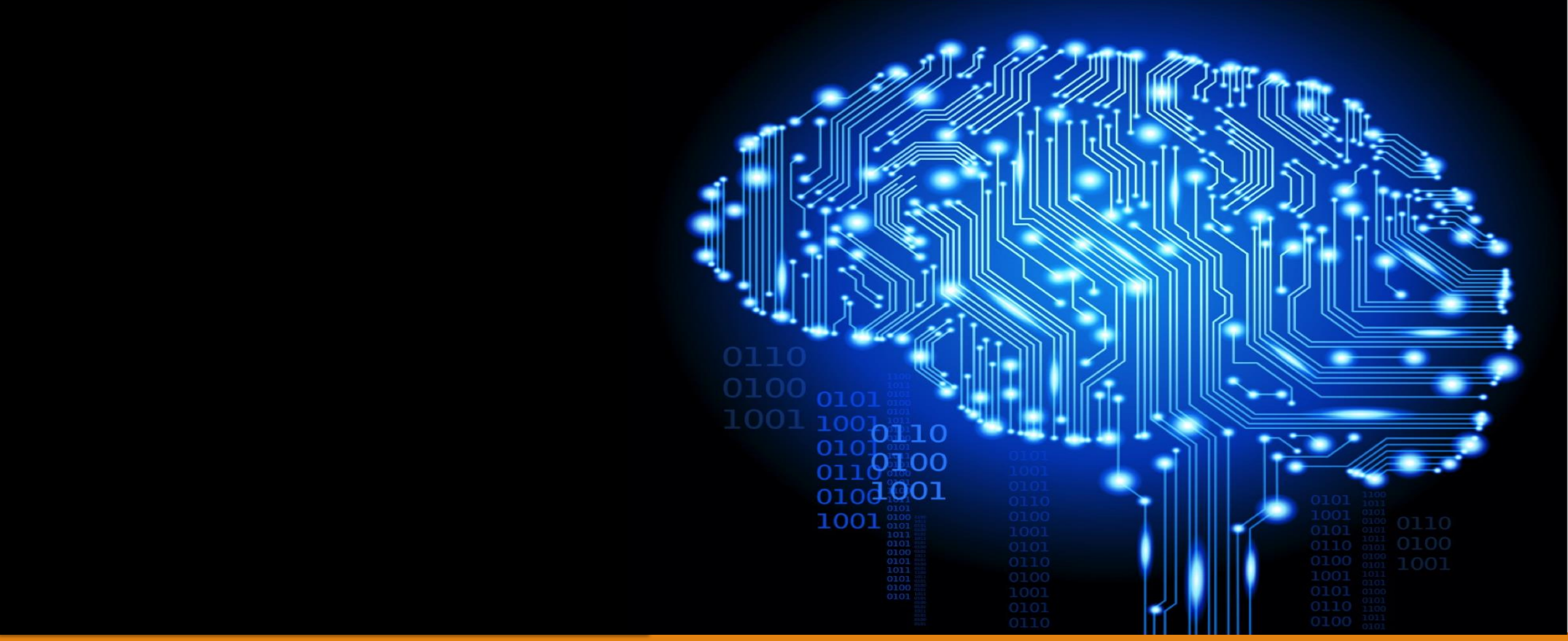


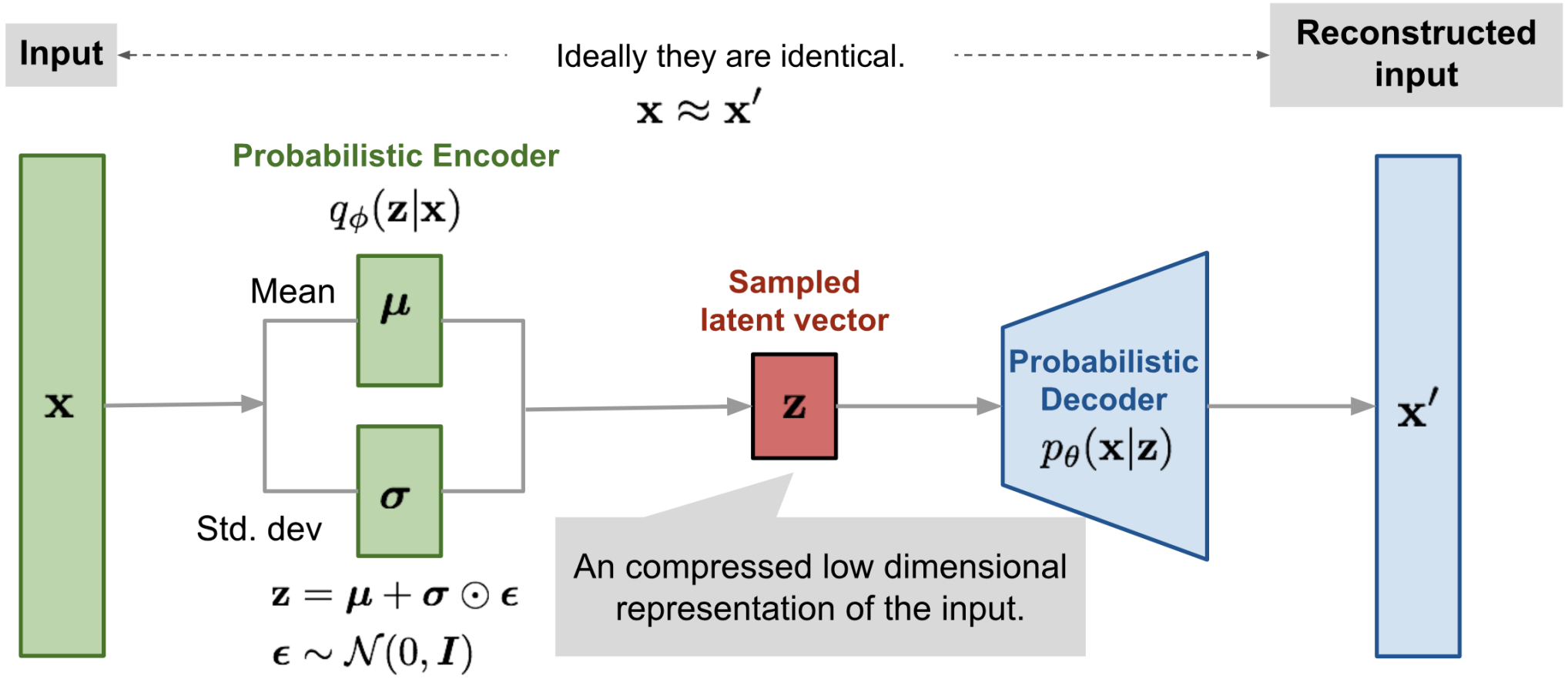
Lecture 9: Deep Sampling & Stochastic Gradients

Efstratios Gavves

Lecture overview

- How to train a VAE
- A bit of Monte Carlo Simulation
- How to sample from a stochastic computation graph
- How to estimate gradients when analytic computations are not possible

Variational Autoencoders



https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

VAE Training Pseudocode

Data:

$\mathcal{D}$: Dataset

$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$: Inference model

$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$: Generative model

Result:

$\boldsymbol{\theta}, \boldsymbol{\phi}$: Learned parameters

$(\boldsymbol{\theta}, \boldsymbol{\phi}) \leftarrow$ Initialize parameters

while *SGD not converged* **do**

  $\mathcal{M} \sim \mathcal{D}$ (Random minibatch of data)

  $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ (Random noise for every datapoint in $\mathcal{M}$)

  Compute $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M}, \boldsymbol{\epsilon})$ and its gradients $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M}, \boldsymbol{\epsilon})$

  Update $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ using SGD optimizer

end

The ELBO's gradients

Variational Autoencoders

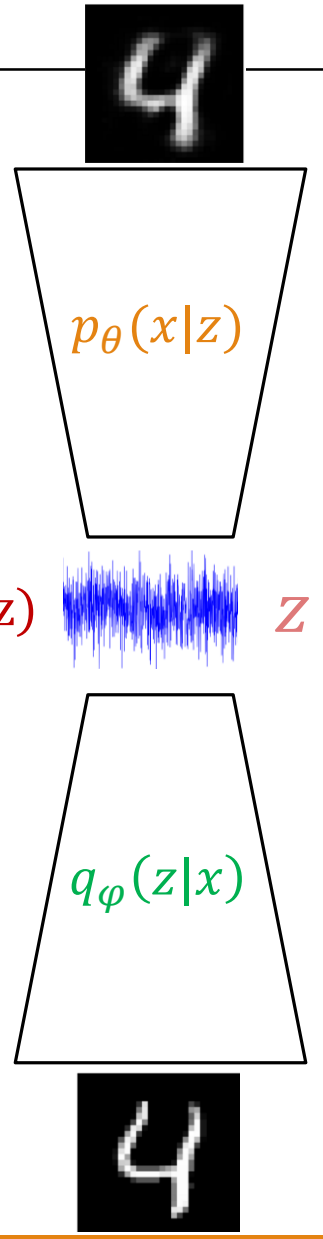
- $\text{ELBO}_{\theta,\varphi}(x) = \mathbb{E}_{q_\varphi(z|x)}[\log p_\theta(x|z)] - \text{KL}(q_\varphi(z|x)||\text{p}_\lambda(\text{z}))$
- How to model $p_\theta(x|z)$ and $q_\varphi(z|x)$?

Variational Autoencoders

- $\text{ELBO}_{\theta,\varphi}(x) = \mathbb{E}_{q_\varphi(z|x)}[\log p_\theta(x|z)] - \text{KL}(q_\varphi(z|x)||\text{p}_\lambda(\text{z}))$
- How to model $p_\theta(x|z)$ and $q_\varphi(z|x)$?
- What about modelling them as neural networks

Variational Autoencoders

- The approximate posterior $q_\varphi(z|x)$ is a CovnNet (or MLP)
  - Input x is an image
  - Given input the output is a feature map from a latent variable z
  - Also known as encoder or inference or recognition network, because it infers/recognizes the latent codes
- The likelihood density $p_\theta(x|z)$ is an inverted ConvNet (or MLP)
  - Given the latent z as input, it reconstructs the input $\tilde{x}$
  - Also known as decoder or generator network
- If we ignore the distribution of the latents z, $\mathrm{p}_\lambda(\mathrm{z})$, then we get the Vanilla Autoencoder



Training Variational Autoencoders

- Maximize the Evidence Lower Bound (ELBO)
  - Or minimize the negative ELBO

$$\mathcal{L}(\theta, \varphi) = \mathbb{E}_{q_\varphi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\varphi(z|x)||\mathrm{p}_\lambda(\mathrm{z}))$$

- How to we optimize the ELBO?

Training Variational Autoencoders

- Maximize the Evidence Lower Bound (ELBO)
  - Or minimize the negative ELBO

$$\mathcal{L}(\theta, \varphi) = \mathbb{E}_{q_\varphi(Z|x)}[\log p_\theta(x|Z)] - \mathrm{KL}(q_\varphi(Z|x)||\mathrm{p}_\lambda(\mathrm{Z}))$$

$$= \int_z q_\varphi(z|x) \log p_\theta(x|z)\, dz - \int_z q_\varphi(z|x) \log \frac{q_\varphi(z|x)}{p_\lambda(z)} dz$$

- Forward propagation → compute the two terms
- The first term is an integral (expectation) that we cannot solve analytically. So, we need to sample from the pdf instead
  - When $p_\theta(x|z)$ contains even a few nonlinearities, like in a neural network, the integral is hard to compute analytically

Training Variational Autoencoders

- Maximize the Evidence Lower Bound (ELBO)
  - Or minimize the negative ELBO

$$\mathcal{L}(\theta, \varphi) = \mathbb{E}_{q_\varphi(Z|x)}[\log p_\theta(x|Z)] - \mathrm{KL}(q_\varphi(Z|x)||\mathrm{p}_\lambda(\mathrm{Z}))$$

$$= \int_z q_\varphi(z|x) \log p_\theta(x|z)\, dz - \int_z q_\varphi(z|x) \log \frac{q_\varphi(z|x)}{p_\lambda(z)} dz$$

- Forward propagation → compute the two terms
- The first term is an integral (expectation) that we cannot solve analytically.
  - When $p_\theta(x|z)$ contains even a few nonlinearities, like in a neural network, the integral is hard to compute analytically
- So, we need to sample from the pdf instead
- VAE is a stochastic model
- The second term is the KL divergence between two distributions that we know

Training Variational Autoencoders

- $\int_z q_\varphi(z|x) \log p_\theta(x|z)\, dz$
- The first term is an integral (expectation) that we cannot solve analytically.
  - When $p_\theta(x|z)$ contains even a few nonlinearities, like in a neural network, the integral is hard to compute analytically
- As we cannot compute analytically, we sample from the pdf instead
  - Using the density $q_\varphi(z|x)$ to draw samples
  - Usually one sample is enough → stochasticity reduces overfitting
- VAE is a stochastic model
- The second term is the KL divergence between two distributions that we know

Training Variational Autoencoders

- $\int_z q_\varphi(z|x) \log \frac{q_\varphi(z|x)}{p_\lambda(z)} dz$
- The second term is the KL divergence between two distributions that we know
- E.g., compute the KL divergence between a centered $N(0,1)$ and a non-centered $N(\mu, \sigma)$ gaussian

Typical VAE

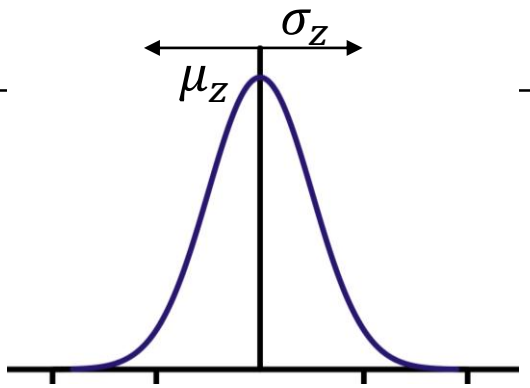
- We set the prior $\mathrm{p}_\lambda(\mathrm{z})$ to be the unit Gaussian

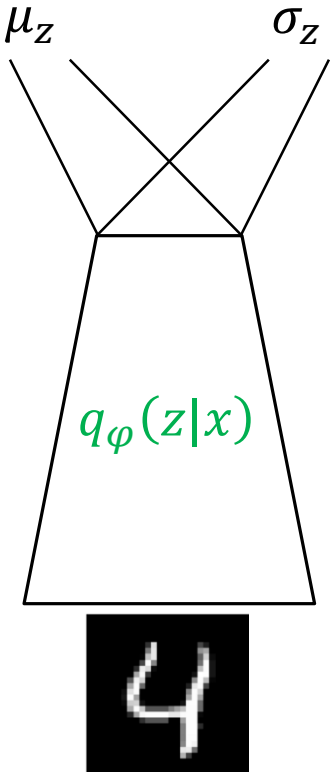
$$\mathrm{p}(z) \sim N(0,1)$$

- We set the likelihood to be a Bernoulli for binary data

$$p(x|z) \sim Bernoulli(\pi)$$

- We set $q_\varphi(\mathrm{z}|\mathrm{x})$ to be a neural network (MLP, ConvNet), which maps an input $\mathbf{x}$ to the Gaussian distribution, specifically it's mean and variance
  - $\mu_z, \sigma_z \sim q_\varphi(\mathrm{z}|\mathrm{x})$
  - The neural network has two outputs, one is the mean μ_x and the other the σ_x, which corresponds to the covariance of the Gaussian

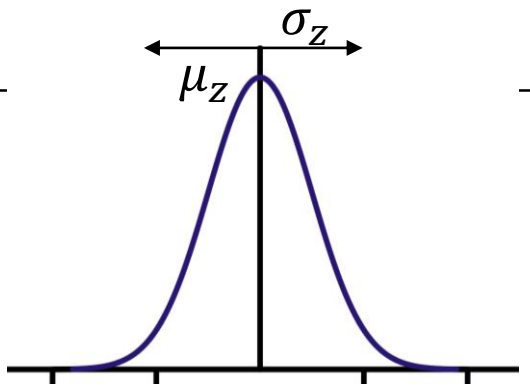


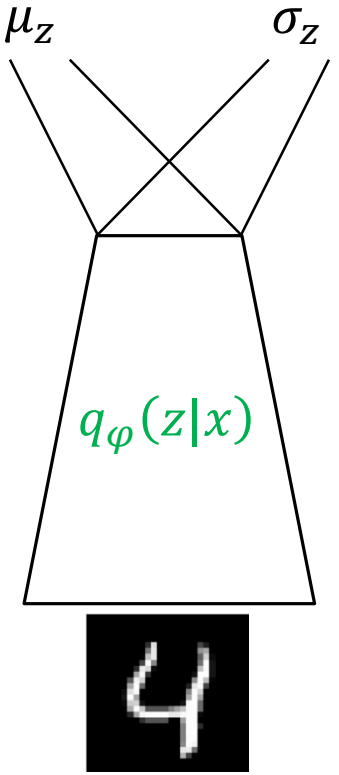


Typical VAE

- We set $p_\theta(\mathrm{x}|\mathrm{z})$ to be an inverse neural network, which maps Z to the Bernoulli distribution if our outputs binary (e.g. Binary MNIST)

- Good exercise: Derive the ELBO for the standard VAE
  - What does the reconstruction term look like with a Gaussian latent space and Bernoulli output?





Forward propagation in VAE

- Sample z from the approximate posterior density $z \sim q_\varphi(Z|x)$
  - As q_φ is a neural network that outputs values from a specific and known parametric pdf, e.g. a Gaussian, sampling from it is rather easy
  - Often even a single draw is enough
- Second, compute the $\log p_\theta(x|Z)$
  - As p_θ is a a neural network that outputs values from a specific and known parametric pdf, e.g. a Bernoulli for white/black pixels, computing the log-prob is easy
- Computing the ELBO is rather straightforward in the standard case
- How should we optimize the ELBO?

Forward propagation in VAE

- Sample z from the approximate posterior density $z \sim q_\varphi(Z|x)$
  - As q_φ is a neural network that outputs values from a specific and known parametric pdf, e.g. a Gaussian, sampling from it is rather easy
  - Often even a single draw is enough
- Second, compute the $\log p_\theta(x|Z)$
  - As p_θ is a a neural network that outputs values from a specific and known parametric pdf, e.g. a Bernoulli for white/black pixels, computing the log-prob is easy
- Computing the ELBO is rather straightforward in the standard case
- How should we optimize the ELBO? Backpropagation?

Backward propagation in VAE

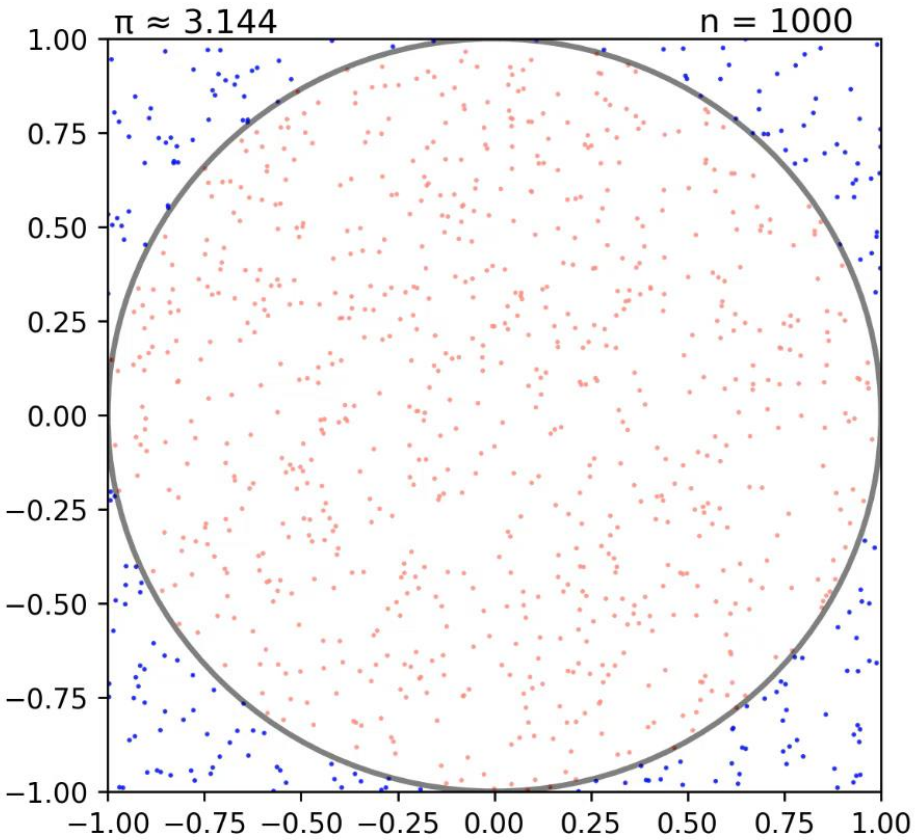
- Backpropagation → compute the gradients of

$$\mathcal{L}(\theta, \varphi) = \mathbb{E}_{z \sim q_\varphi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\varphi(z|x)||\mathrm{p}_\lambda(\mathrm{z}))$$

- We must take the gradients with respect to the trainable parameters
- The generator network parameters θ
- The inference network/approximate posterior parameters φ

- But how? Both the $\mathbb{E}(\cdot)$ and the $\mathrm{KL}(\cdot)$ are integrals
  - How can we backprop through them?
- Monte Carlo to the rescue

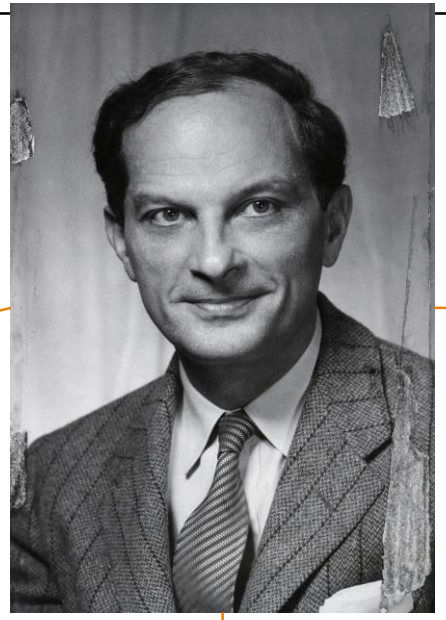
Monte Carlo Estimation
Monte Carlo Simulation

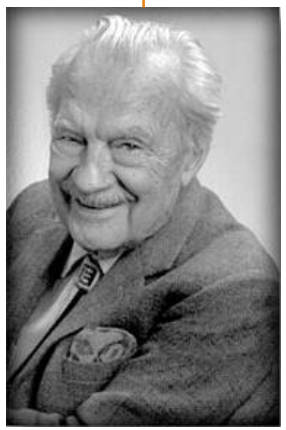


How it started

John von Neumann

Stanislav Ulam

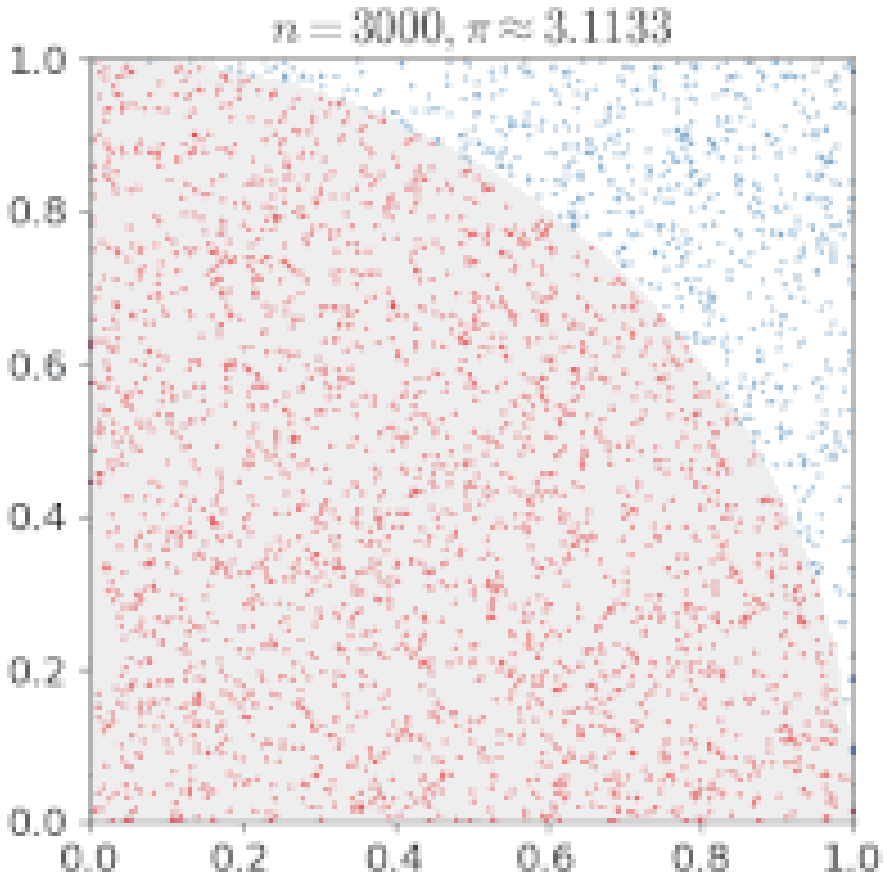




Manhattan project

Estimating π with Monte Carlo: A toy example

- One can estimate the value of π numerically
- In this visualization only the upper right quadrant of the circle
- Basically, we count
  how many points are in the circle vs
  how many are in the square but not the circle



Monte Carlo Integration

- Let's try to compute the following integral

$$\mathbb{E}(f) = \int_x p(x) f(x) dx$$

where $p(x)$ is a probability density function for x

- Often complex if $p(x)$ and $f(x)$ is slightly complicated
- As a consequence, often intractable or too expensive to compute

Monte Carlo Integration

- Instead, we can approximate the integral as a summation

$$\mathbb{E}(f) = \int_x p(x) f(x) dx \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i) = \hat{f},$$

where x_i is sampled from $p(x)$

- $\hat{f}$ is an estimator because it approximately estimates the value of f
- This means that $\hat{f}$ itself is an RV (random variable) with a mean and variance

Monte Carlo Integration

- The estimator is unbiased: $\mathbb{E}(f) = \mathbb{E}(\hat{f})$
- Check the Law of Large Numbers
  - "As the number of identically distributed, randomly generated variables increases, their sample mean (average) approaches their theoretical mean."
- The estimator variance is

$$Var(\hat{f}_n) = \frac{1}{n}\mathbb{E}[(f - \mathbb{E}(\hat{f}_n))]$$

Monte Carlo Integration: The gist

- Again, let's say we have the estimator

$$\mathbb{E}(f) = \int_x p(x) f(x) \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i) = \hat{f}$$

- If $p(x)$ is a probability density function easy to sample, then instead of computing the integral, we can randomly sample x_i and approximate the integral instead
- The error reduces at a rate of $O(\sqrt{N})$
- This is possible only if the integral has the form of a pdf
  - That is there is a pdf in it

More generally

- Let's assume you want to compute the value of an arbitrary integral

$$F = \int_a^b f(x)dx$$

- With a suitable pdf $p(x)$ to sample from then we can use the estimator

$$\hat{F}_n = \frac{1}{n}\sum_n \frac{f(x_i)}{p(x_i)}, x_i \sim p(x)$$

- In fact, it can be shown the expectation of $\hat{F}_n$ is exactly equal to the integral

$$\mathbb{E}[\hat{F}_n] = \int_a^b f(x)dx$$

(task for the interested reader)

Why sampling?

- Estimate density of a function

$$Pr(a \leq X \leq b) = \int_a^b p_X(x)\, dx$$

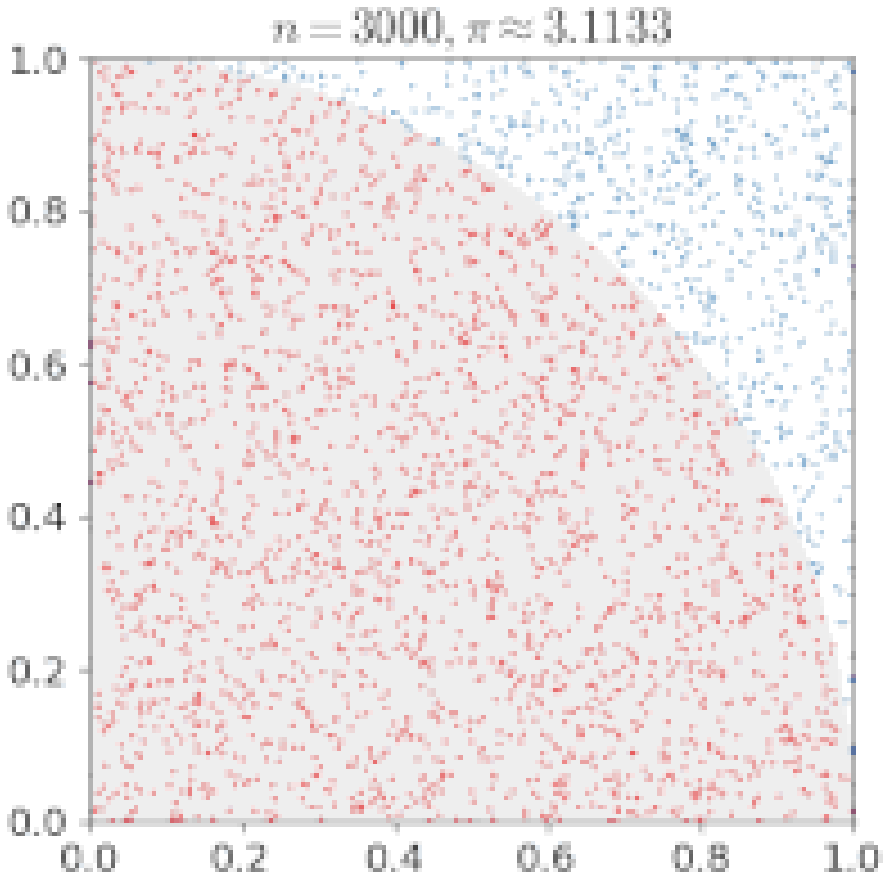
- Estimate some quantities of interests, e.g., means or variances

$$\mu(x) = \int_a^b x\, p(x)\, dx$$

- Optimize a function, e.g., locate that sample that minimizes or maximizes our objective, e.g., ELBO

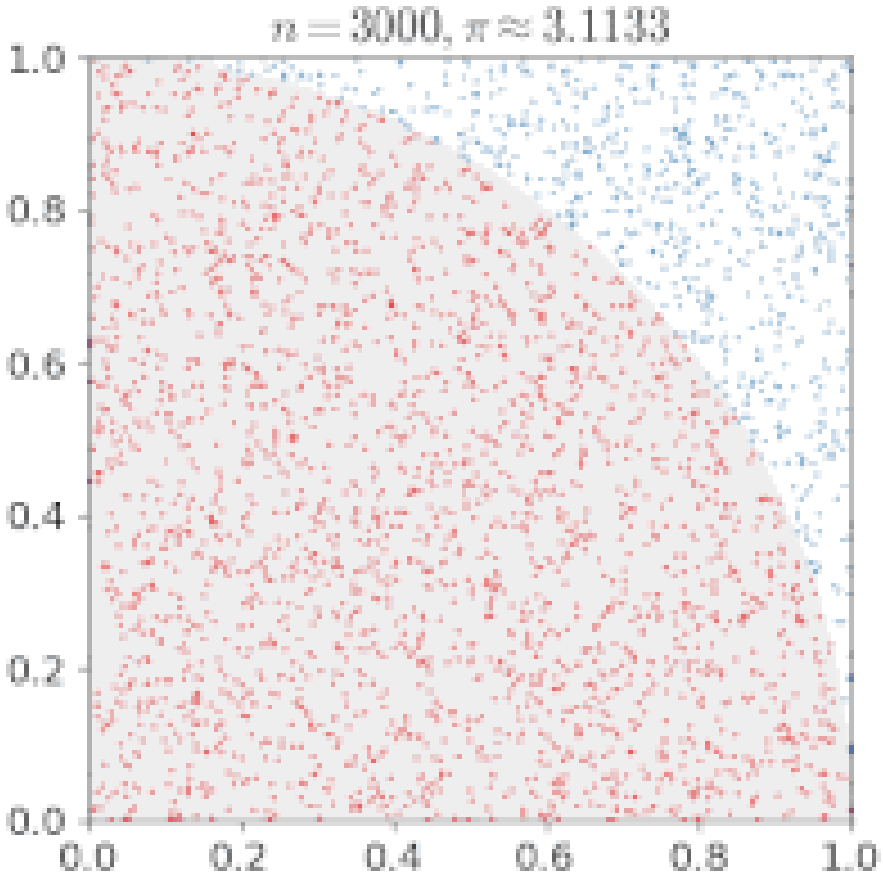
Estimating π with Monte Carlo: Revisiting

- How is it done algorithmically?
- What is our $\mathbf{F}(x)$, $f(x)$ and $p(x)$?



Estimating π with Monte Carlo: Revisiting

- How is it done algorithmically?
- What is our F, $f(x)$ and $p(x)$?
- We know that the area of the circle is $\mathrm{E_c} = \pi\rho^2$
- We define an inscribing unit square with an area $\mathrm{E_s} = (2\rho)^2 = 4\rho^2$
- Their ratio is therefore $\frac{\mathrm{E_c}}{\mathrm{E_s}} = \frac{\pi}{4}$
- We set our pdf to be the uniform distribution in the unit square and sample points from it
- By theory we know that the ideal value of the integral is $F = \frac{\pi}{4}$
- We set $f(x) = 1$ if the point is in the circle (distance from center smaller than radius), otherwise 0, and then count $\frac{\pi}{4} = \frac{n_{in}}{n}$



Sampling == Simulation

- In the “simulation view” everything is a sample. Also your training data are samples. Not deterministic data points, random samples
- So, no point in *optimizing* for these samples. Only makes sense to optimize for the process the generates these samples.
- But how can we optimize something that we don’t know yet, something that we in fact try to learn in the first place?

Sampling == Simulation

- By simulation: we simulate the function we attempt to learn using the existing samples, and then generate new samples to see if they make sense
- In typical optimization we make a call to (deterministic) function value. In stochastic optimization we call random variable and call for an estimate of function value
- Estimate of loss, estimate of gradient, estimate of Hessian
- Doubly stochastic optimization, if combined with mini-batching

Types of Monte Carlo Sampling

- Direct sampling
  - Sample directly from the pdf
- Importance sampling
  - Instead of sampling directly from the target pdf, use another simpler pdf to sample from.
  - Then, reweight the results according to the ratio between the simpler sampling pdf and the target pdf
- Rejection sampling
  - Sample from a broader distribution
  - Reject samples that are outside a predefined region

Applications

- High-energy Physics
- Finance
- Machine Learning
- All sort of simulations

Monte Carlo in Learning (aka stochastic gradient estimation)

- We have a general probabilistic objective we want to compute

$$\mathcal{F}(\theta) = \int p(x;\theta) f(x;\varphi) dx$$

  - The pdf $p(x;\theta)$ is also known as the measure with distributional parameters θ
  - The structured function $f(x;\varphi)$ is called the *cost* with structural parameters φ
  - We assume a pdf that is continuous in its domain and differentiable wrt θ

- Then we want to learn

$$\eta = \nabla_\theta \mathcal{F}(\theta) = \nabla_\theta \mathbb{E}[f(x;\varphi)]$$

- This gradient of expectation is called sensitivity analysis

Challenges with sensitivity analysis

$$\eta = \nabla_\theta \mathcal{F}(\theta) = \nabla_\theta \mathbb{E}[f(x;\varphi)]$$

Challenges with sensitivity analysis

$$\eta = \nabla_\theta \mathcal{F}(\theta) = \nabla_\theta \mathbb{E}[f(x;\varphi)]$$

- x typically high dimensional
- In high dimensions quadrature is not reliable and numerical integration is hard
- Often the parameters θ are too many in the order of thousands
- Often the cost function is not differentiable or even not known (black box)

- In short the integral (expectation) is often intractable

Desired properties of Monte Carlo estimators

- Consistency
  - With larger samples should the estimate should converge to the true value of the integral
- Unbiasedness $\mathbb{E}_{p(x;\theta)}[\hat{f}_n] = \mathbb{E}_{p(x;\theta)}[f]$
  - Important for gradient estimation → guarantees convergence of stochastic optimization
- Minimum variance
  - Learning more efficient (updates in more consistent direcitons)
  - More accurate gradient estimates
- Computational efficiency
  - The fewer samples the better, even 1 sample if possible

Stochastic gradients: A pipeline

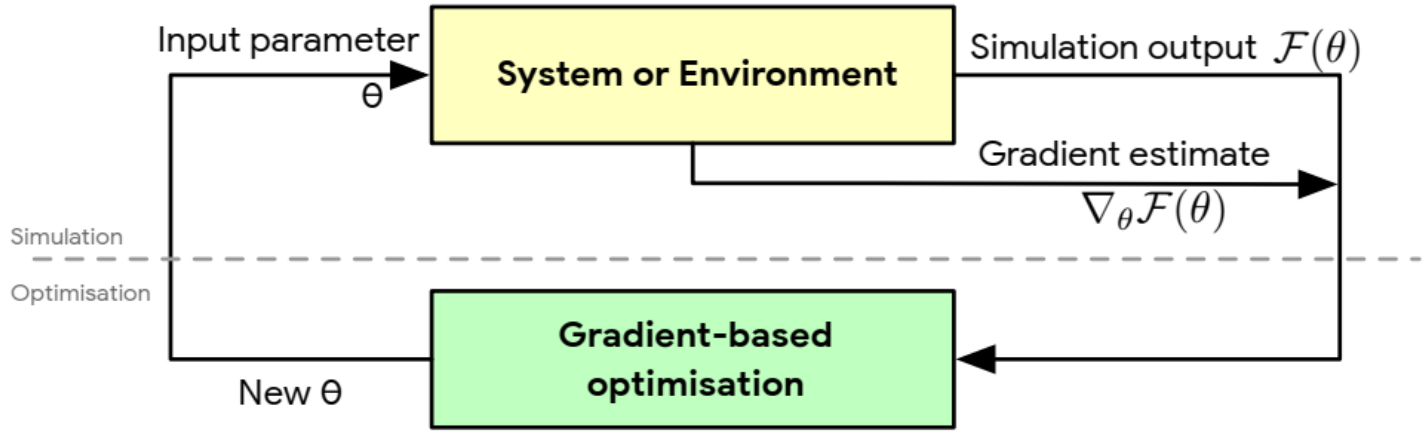


Figure 1: Stochastic optimisation loop comprising a simulation phase and an optimisation phase. The simulation phase produces a simulation of the stochastic system or interaction with the environment, as well as unbiased estimators of the gradient.

Credit: Monte Carlo Gradient Estimation in Machine Learning

Applications of stochastic gradients

- Variational inference: $\eta = \nabla_\theta \mathbb{E}_{q_\theta(z|x)}[\log p(x|z) - \log \frac{q_\theta(z|x)}{p(z)}]$
- Reinforcement learning: $\eta = \nabla_\theta \mathbb{E}_{p_\theta(\tau)}[\sum_t \gamma_t \, r(s_t, a_t)]$
- Sensitivity analysis
- Discrete event systems and queuing theory
- Experimental design

Families of MC gradient estimators

- Score-function estimator
- Pathwise gradient estimators
- Measure-valued gradient estimators

Score-function estimator (aka REINFORCE)

- This estimator revolves around the following derivative

$$\frac{d \log f(x)}{dx} = \frac{1}{f(x)} \cdot \frac{df(x)}{dx}$$

- Or with pdfs

$$\nabla_\theta \log p(x;\theta) = \frac{1}{p(x;\theta)} \nabla_\theta p(x;\theta)$$

- $\nabla_\theta \log p(x;\theta)$ is more generally called the score-function

Deriving the score-function estimator

$$\boldsymbol{\eta} = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{p(\mathbf{x};\boldsymbol{\theta})}\left[f(\mathbf{x})\right] = \nabla_{\boldsymbol{\theta}} \int p(\mathbf{x};\boldsymbol{\theta}) f(\mathbf{x}) d\mathbf{x} = \int f(\mathbf{x}) \nabla_{\boldsymbol{\theta}} p(\mathbf{x};\boldsymbol{\theta}) d\mathbf{x}$$

$$= \int p(\mathbf{x};\boldsymbol{\theta}) f(\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x};\boldsymbol{\theta}) d\mathbf{x}$$

$$= \mathbb{E}_{p(\mathbf{x};\boldsymbol{\theta})}\left[f(\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x};\boldsymbol{\theta})\right]$$

$$\bar{\boldsymbol{\eta}}_N = \frac{1}{N} \sum_{n=1}^{N} f(\hat{\mathbf{x}}^{(n)}) \nabla_{\boldsymbol{\theta}} \log p(\hat{\mathbf{x}}^{(n)};\boldsymbol{\theta}); \quad \hat{\mathbf{x}}^{(n)} \sim p(\mathbf{x};\boldsymbol{\theta}).$$

Score-function estimator properties

- Any type of function $f(x)$ can be used, so simulators or black box functions (Reinforcement Learning) go well with it
- The measure (pdf) must be differentiable wrt to its parameters
- It must be easy to sample from the measure
- Even with 1 sample you get something
- But, very high variance in general → not always working as desired → variance reduction methods are usually needed

Pathwise gradient estimator (aka reparameterization trick)

- Instead of sampling directly from a complex pdf
  - sometimes it is possible to rewrite as a simpler pdf
  - Then deterministically (backprop possible ;)) transform the sample

$$\hat{\mathbf{x}} \sim p(\mathbf{x};\boldsymbol{\theta}) \quad \equiv \quad \hat{\mathbf{x}} = g(\hat{\boldsymbol{\epsilon}},\boldsymbol{\theta}), \quad \hat{\boldsymbol{\epsilon}} \sim p(\boldsymbol{\epsilon}),$$

- At the heart of this method is the change of variables formula

$$p(\mathbf{x};\boldsymbol{\theta}) = p(\boldsymbol{\epsilon}) \left|\nabla_{\boldsymbol{\epsilon}} g(\boldsymbol{\epsilon};\boldsymbol{\theta})\right|^{-1}.$$

Deriving pathwise gradients

$$\boldsymbol{\eta} = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{p(\mathbf{x};\boldsymbol{\theta})} \left[f(\mathbf{x}) \right] = \nabla_{\boldsymbol{\theta}} \int p(\mathbf{x};\boldsymbol{\theta}) f(\mathbf{x}) d\mathbf{x}$$

$$= \nabla_{\boldsymbol{\theta}} \int p(\boldsymbol{\epsilon}) f(g(\boldsymbol{\epsilon};\boldsymbol{\theta})) d\boldsymbol{\epsilon}$$

$$= \mathbb{E}_{p(\boldsymbol{\epsilon})} \left[\nabla_{\boldsymbol{\theta}} f(g(\boldsymbol{\epsilon};\boldsymbol{\theta})) \right]$$

$$\bar{\boldsymbol{\eta}}_N = \frac{1}{N} \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} f(g(\hat{\boldsymbol{\epsilon}}^{(n)};\boldsymbol{\theta})); \quad \hat{\boldsymbol{\epsilon}}^{(n)} \sim p(\boldsymbol{\epsilon}).$$

An example: The Gaussian distribution

- Let's say we want to take samples from

$$z \sim N(\mu, \sigma^2)$$

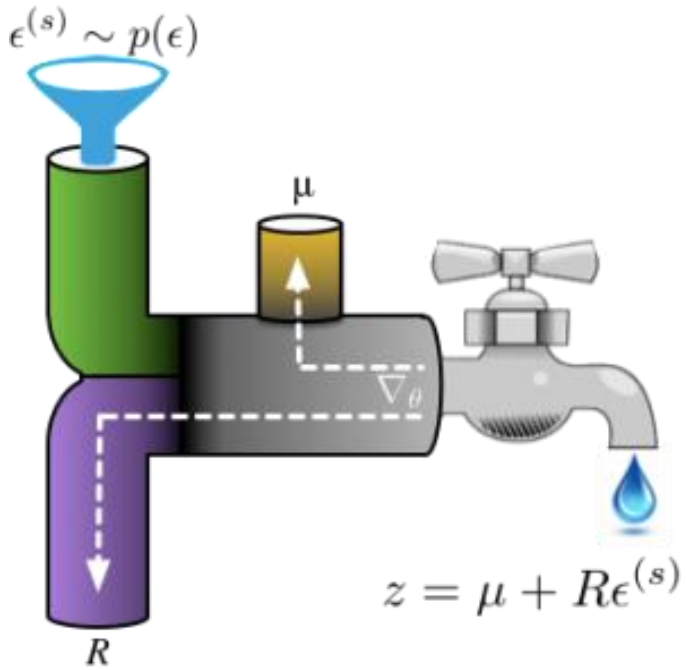
- We can instead first sample from

$$\varepsilon \sim N(0, 1)$$

and then transform our random ε sample like

$$z = \mu + \varepsilon \cdot \sigma$$

- Can be seen also as standardization



http://blog.shakirm.com/2015/10/machine-learning-trick-of-the-day-4-reparameterisation-tricks/

Pathwise gradient estimator properties

- Only differentiable cost functions
  - Unlike score-function estimators that work with any cost function
- We do not need to know the measure explicitly. Only the deterministic transformation and the base sampling distribution
- Very efficient. Even a single sample suffices no matter dimensionality
- Low variance in general, lower than the score-function estimator

Qualitative comparison between estimators (1)

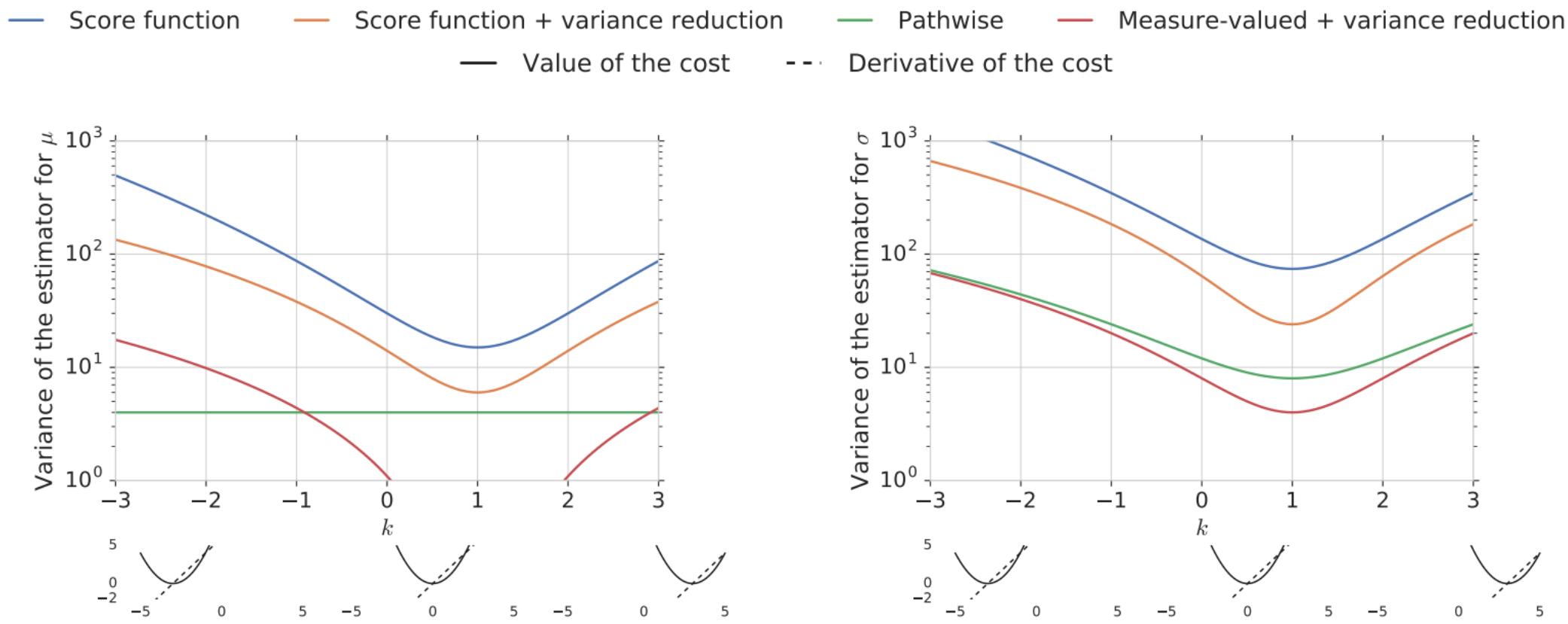


Figure 2: Variance of the stochastic estimates of $\nabla_\theta \mathbb{E}_{\mathcal{N}(x|\mu,\sigma^2)}\left[(x-k)^2\right]$ for $\mu = \sigma = 1$ as a function of k for three different classes of gradient estimators. Left: $\theta = \mu$; right: $\theta = \sigma$. The graphs in the bottom row show the function (solid) and its gradient (dashed) for $k \in \{-3, 0, 3\}$.

Qualitative comparison between estimators (1)

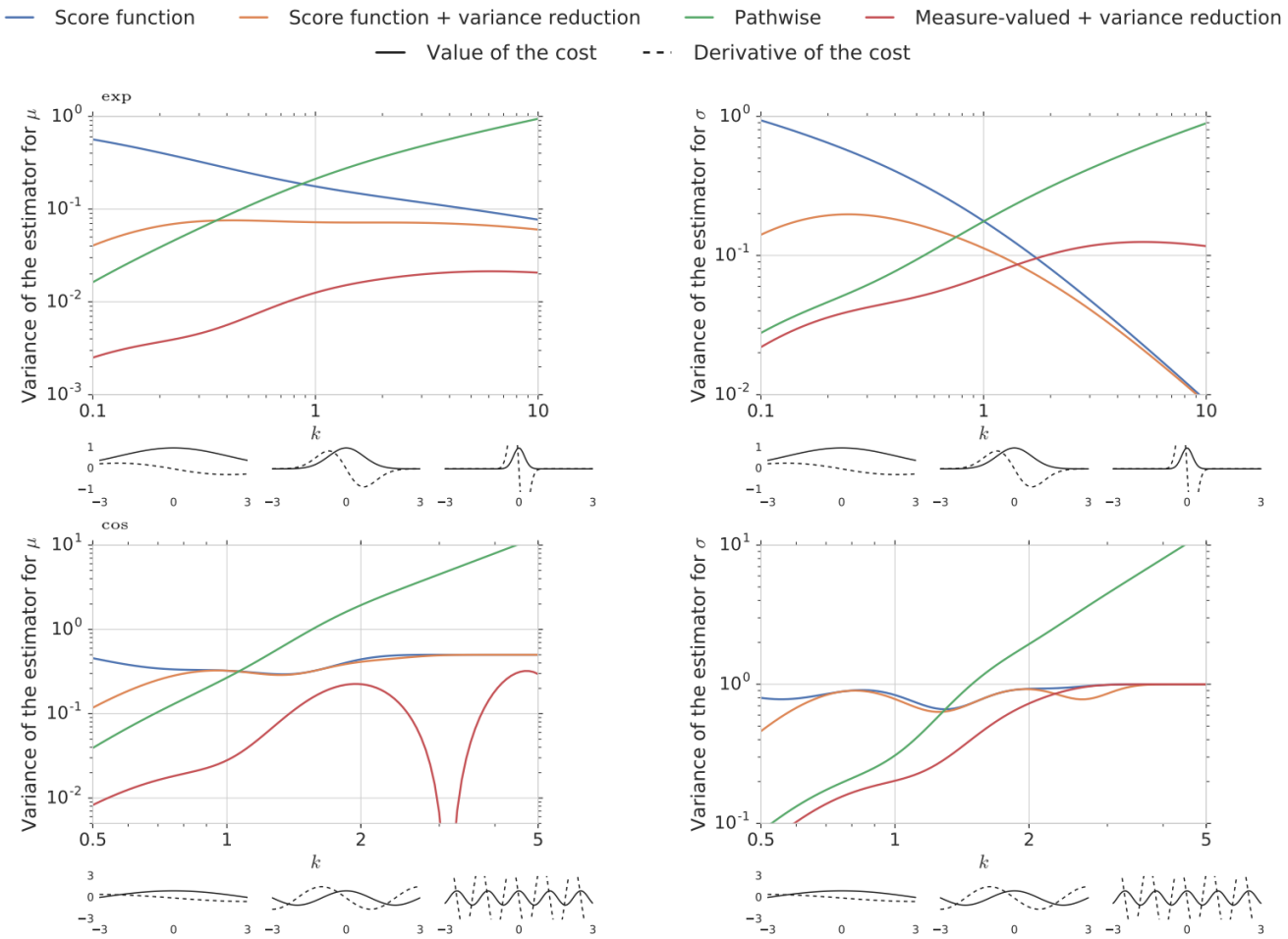


Figure 3: Variance of the stochastic estimates of $\nabla_\theta \mathbb{E}_{\mathcal{N}(x|\mu,\sigma^2)}\left[f(x;k)\right]$ for $\mu = \sigma = 1$ as a function of k. Top: $f(x;k) = \exp(-kx^2)$, bottom: $f(x;k) = \cos kx$. Left: $\theta = \mu$; right: $\theta = \sigma$. The graphs in the bottom row show the function (solid) and its gradient (dashed): for $k \in \{0.1, 1, 10\}$ for the exponential function, and $k \in \{0.5, 1.58, 5.\}$ for the cosine function.

Variance reduction

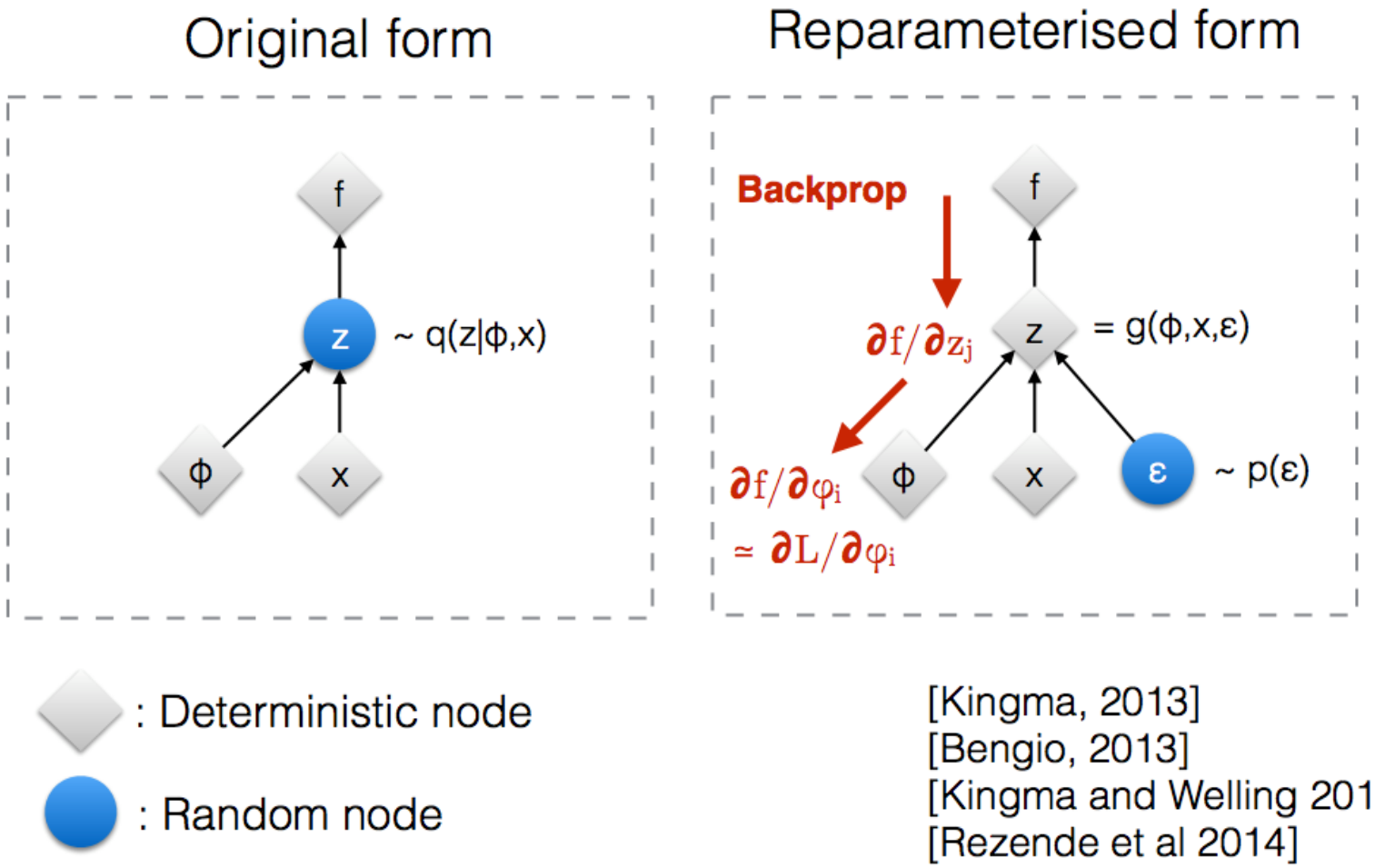
- To make estimated gradients more usable, it is often important to reduce their variance → variance reduction
- A popular method is by control variates
- If we have a function $h(x)$ for which we know the expectation
  - Control variate
- Then we can estimate the gradient like

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}) - \beta(h(\mathbf{x}) - \mathbb{E}_{p(\mathbf{x};\boldsymbol{\theta})}[h(\mathbf{x})])$$

$$\bar{\eta} = \frac{1}{N}\sum_{n=1}^{N}\tilde{f}(\hat{\mathbf{x}}^{(n)}) = \bar{f} - \beta(\bar{h} - \mathbb{E}_{p(\mathbf{x};\boldsymbol{\theta})}[h(\mathbf{x})]),$$

- In expectation the effect of $h(x)$ vanishes, but during sampling variance is reduced

Revisiting VAE learning



[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

Gradients w.r.t. the generator parameters θ

- Backpropagation → compute the gradients of

$$\mathcal{L}(\theta, \varphi) = \mathbb{E}_{z \sim q_\varphi(Z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\varphi(Z|x)||\mathrm{p}_\lambda(\mathrm{Z}))$$

Gradients w.r.t. the generator parameters θ

- Backpropagation → compute the gradients of

$$\mathcal{L}(\theta, \varphi) = \mathbb{E}_{z \sim q_\varphi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\varphi(z|x)||\mathrm{p}_\lambda(\mathrm{z}))$$

with respect to θ and φ

- $\nabla_\theta \mathcal{L} = \mathbb{E}_{z \sim q_\varphi(z|x)}[\nabla_\theta \log p_\theta(x|z)]$
- The expectation and sampling in $\mathbb{E}_{z \sim q_\varphi(z|x)}$ do not depend on θ
- Also, the KL does not depend on θ, so no gradient from over there!
- So, no problem → Just Monte-Carlo integration using samples z drawn from $q_\varphi(z|x)$

Gradients w.r.t. the recognition parameters φ

- Backpropagation → compute the gradients of

$$\mathcal{L}(\theta, \varphi) = \mathbb{E}_{z \sim q_\varphi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\varphi(z|x)||\mathrm{p}_\lambda(\mathrm{z}))$$

- Our latent variable z is a Gaussian (in standard VAE) represented by μ_z, σ_z
- So, we can train by sampling randomly from that Gaussian $z \sim N(\mu_Z, \sigma_Z)$
- Problem?
- Sampling $z \sim q_\varphi(z|x)$ is not differentiable
  - And after sampling z, it's a fixed value (not a function), so we cannot backprop
- Not differentiable → no gradients
- No gradients → No backprop → No training!

Solution: Monte Carlo Differentiation?

- $\nabla_\varphi \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] = \nabla_\varphi \int_z q_\varphi(z|x) \log p_\theta(x|z) dz$

$$= \int_z \nabla_\varphi [q_\varphi(z|x)] \log p_\theta(x|z) dz$$

- Problem: Monte Carlo integration not possible anymore
  - No density function inside the integral
  - Only the gradient of a density function
- Similar to Monte Carlo integration, we want to have an expression where there is a density function inside the summation
- That way we can express it again as Monte Carlo integration

Solution: Monte Carlo Differentiation?

- $\nabla_\varphi \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] = \nabla_\varphi \int_z q_\varphi(z|x) \log p_\theta(x|z) dz$

$$= \int_z \nabla_\varphi [q_\varphi(z|x)] \log p_\theta(x|z) dz$$

- $\int_z \nabla_\varphi [q_\varphi(z|x)] \log p_\theta(x|z) dz =$

$$= \int_z \frac{q_\varphi(z|x)}{q_\varphi(z|x)} \nabla_\varphi [q_\varphi(z|x)] \log p_\theta(x|z) dz$$

$$\text{NOTE: } \nabla_x \log f(x) = \frac{1}{f(x)} \nabla_x f(x)$$

$$= \int_z q_\varphi(z|x) \nabla_\varphi [\log q_\varphi(z|x)] \log p_\theta(x|z) dz$$

$$= \mathbb{E}_{z\sim q_\varphi(z|x)} [\nabla_\varphi [\log q_\varphi(z|x)] \log p_\theta(x|z)]$$

Solution: Monte Carlo Differentiation == REINFORCE

- $\nabla_\varphi \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] = \mathbb{E}_{z\sim q_\varphi(z|x)}[\nabla_\varphi[\log q_\varphi(z|x)]\log p_\theta(x|z)]$

$$= \sum_i \nabla_\varphi[\log q_\varphi(z_i|x)]\log p_\theta(x|z_i), z_i \sim q_\varphi(z|x)$$

- Also known as REINFORCE or score-function estimator
  - $\log p_\theta(x|z)$ is called score function
  - Used to approximate gradients of non-differentiable function
  - Highly popular in Reinforcement Learning, where we also sample from policies
- Problem: Typically high-variance gradients →
- → Slow and not very effective learning

Solution: Reparameterization trick

- Remember, we have a Gaussian output $z \sim N(\mu_Z, \sigma_Z)$
- For certain pdfs, including the Gaussian, we can rewrite their random variable z as deterministic transformations of an auxiliary and simpler random variable ε

$$z \sim N(\mu, \sigma) \Leftrightarrow z = \mu + \varepsilon \cdot \sigma, \qquad \varepsilon \sim N(0, 1)$$

- μ, σ are deterministic (not random) values
- Long story short:
- We can model μ, σ by our NN encoder/recognition
- And ε comes externally

What do we gain?

- Change of variables

$$z = g(\varepsilon)$$
$$\mathrm{p}(z)dz = p(\varepsilon)d\varepsilon$$

  - Intuitively, think that the probability mass must be invariant after the transformation

- In our case

$$\varepsilon \sim q(\varepsilon) = \mathrm{N}(0,1), z = g_\varphi(\varepsilon) = \mu_\varphi + \varepsilon \cdot \sigma_\varphi$$

- $\nabla_\varphi \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] = \nabla_\varphi \int_z q_\varphi(z|x) \log p_\theta(x|z)dz$

$$= \nabla_\varphi \int_\varepsilon q(\varepsilon) \log p_\theta\left(x|\mu_\varphi, \sigma_\varphi, \varepsilon\right)d\varepsilon$$

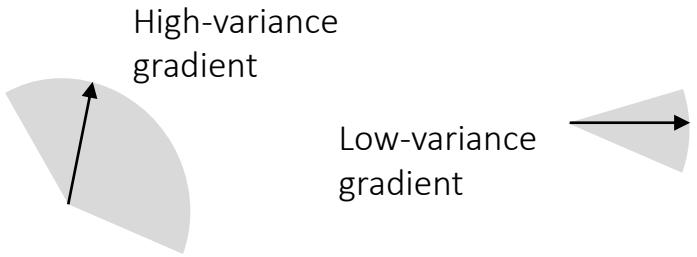
$$= \int_\varepsilon q(\varepsilon)\nabla_\varphi \log p_\theta\left(x|\mu_\varphi, \sigma_\varphi, \varepsilon\right)d\varepsilon$$

- $\nabla_\varphi \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] \approx \sum_i \nabla_\varphi \log p_\theta\left(x|\mu_\varphi, \sigma_\varphi, \varepsilon_i\right), \varepsilon_i \sim N(0,1)$

  - The Monte Carlo integration does not depend on the parameter of interest φ anymore

Solution: Reparameterization trick

- Sampling directly from $\varepsilon \sim N(0,1)$ leads to low-variance estimates compared to sampling directly from $z \sim N(\mu_z, \sigma_z)$
  - Why low variance? Exercise for the interested reader
- Remember: since we are sampling for z, we are also sampling gradients
  - Stochastic gradient estimator
- More distributions beyond Gaussian possible: Laplace, Student-t, Logistic, Cauchy, Rayleight, Pareto



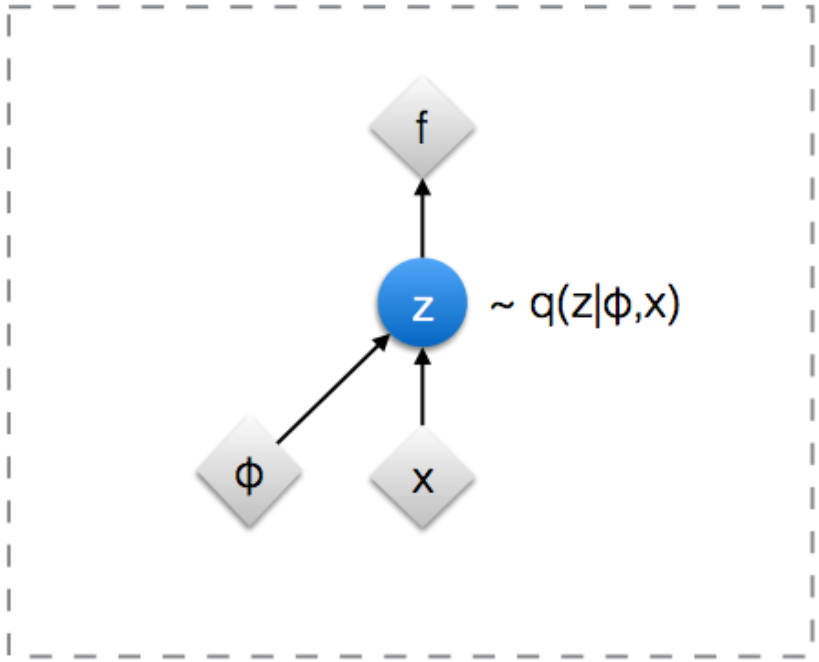
http://blog.shakirm.com/2015/10/machine-learning-trick-of-the-day-4-reparameterisation-tricks/

Once more: what is random in the reparameterization trick?

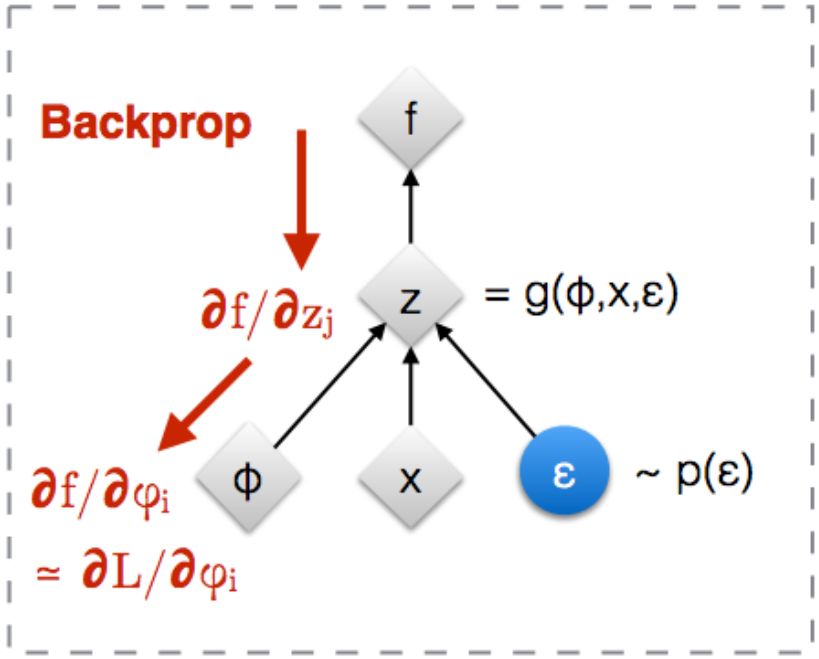
- Again, the latent variable is $z = \mu_\varphi + \varepsilon \cdot \sigma_\varphi$
- μ_φ and σ_φ are deterministic functions (via the neural network encoder)
- ε is a random variable, which comes **externally**
- The z as a result is itself a random variable, because of ε
- However, now the randomness is not associated with the neural network and its parameters that we have to learn
  - The randomness instead comes from the external ε
  - The gradients flow through μ_φ and σ_φ

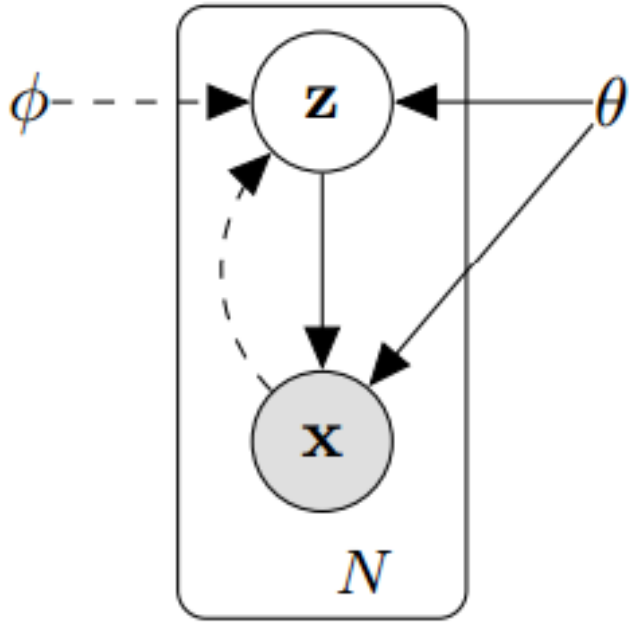
Reparameterization Trick (graphically)

Original form



Reparameterised form





: Deterministic node

: Random node

[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

To sum up on VAEs

- So, our latent variable z is a Gaussian (in the standard VAE) represented by the mean and variance μ_z, σ_z, which are the output of a neural net
- So, we can train by sampling randomly from that Gaussian

$$z \sim N(\mu_z, \sigma_z)$$

- Once we have that z, however, it's a fixed value (not a function), so we cannot backprop
- Better not use REINFORCE algorithm to approximate gradients
  - High-variance gradients → slow and not very effective learning
- Instead the reparameterization trick (pathwise gradients) assuming a differentiable cost function

Summary

- How to train a VAE
- A bit of Monte Carlo Simulation
- How to sample from a stochastic computation graph
- How to estimate gradients when analytic computations are not possible
- Variance reduction