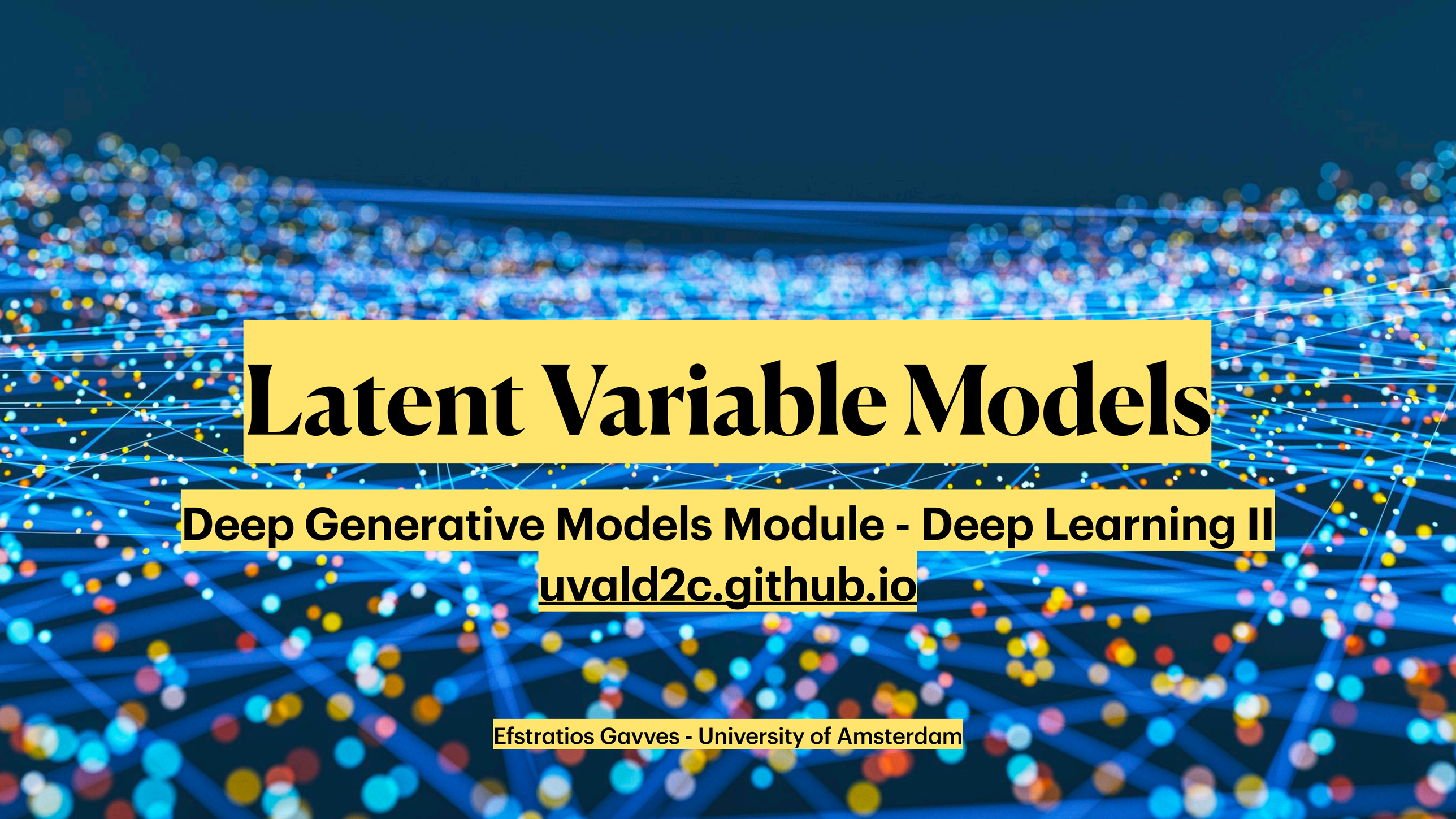


Latent Variable Models

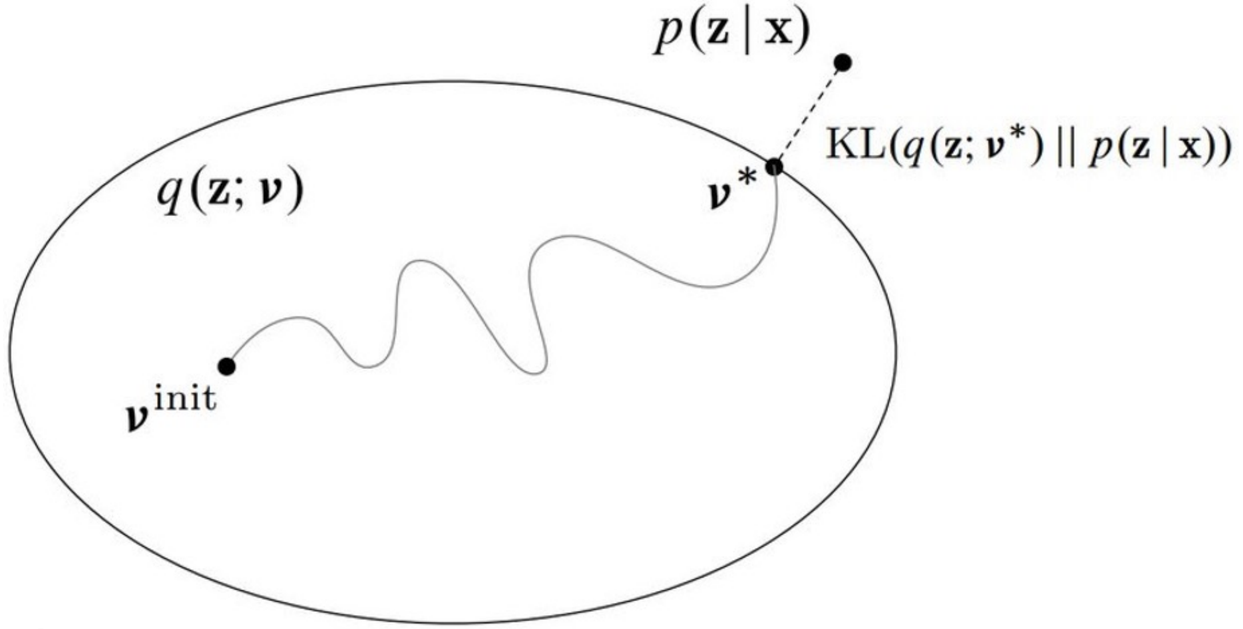
Deep Generative Models Module - Deep Learning II

uvald2c.github.io

Efstratios Gavves - University of Amsterdam

Lecture overview

- What are latent variable models
- Variational Inference
- Variational Autoencoders
- Reparameterization trick and variance reduction
- Insights



Generative models: Recap

- A probabilistic generative model will usually look like some form of

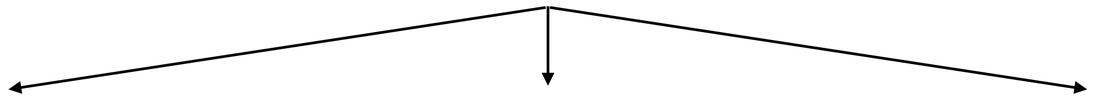
$$p(\mathbf{x}) = \frac{\exp\left(f(\mathbf{x})\right)}{Z}, \text{ where } Z = \int \exp\left(f(\mathbf{x})\right) d\mathbf{x}$$

- The exponential ensures positivity, behaves well with integrals, and interacts well with the logarithms of the log-likelihoods
- However, the normalisation constant is an issue because of computational complexity and analytical intractability

Avoiding normalisation with latent variables

- Avoid normalisation by modelling tractable quantities instead of $p(\mathbf{x})$
- Say we use an auxiliary variable $\mathbf{z}$ that although we do not really know (latent)
- Although counter-intuitive, latent $\mathbf{z}$ could help computations by decomposition ...

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})$$



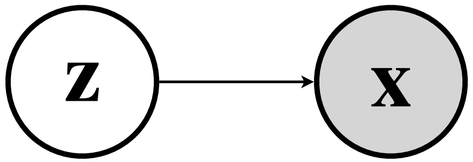
... if we could make sure that $p(\mathbf{x} \mid \mathbf{z})$ is tractable

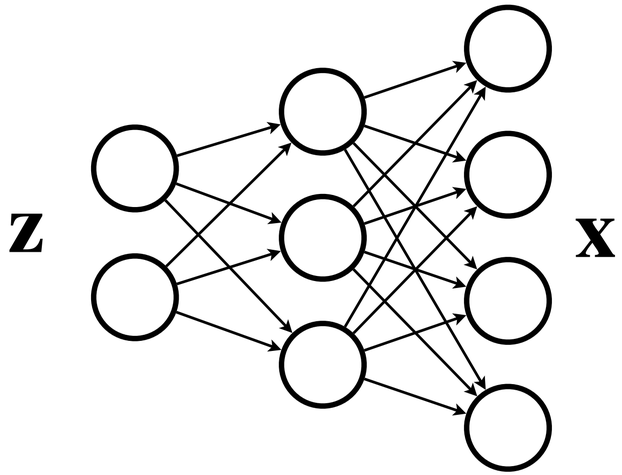
... if we make a convenient choice for $p(\mathbf{z})$

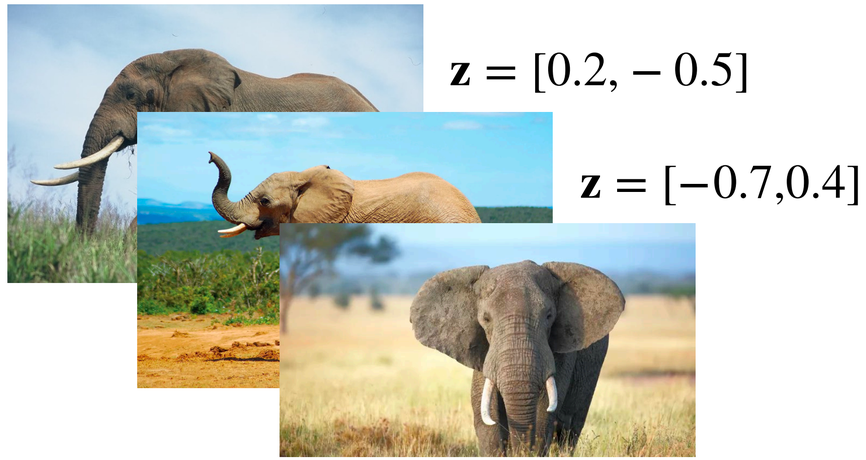
... and if we have a good way to infer the latent $\mathbf{z}$

Latent variables: An oracle intuition

- Latent variables $\mathbf{z}$ are high-level, compressed representations of the input: $p(\mathbf{z} \mid \mathbf{x})$
- Given this latent code, we can recover the input accurately: $p(\mathbf{x} \mid \mathbf{z})$

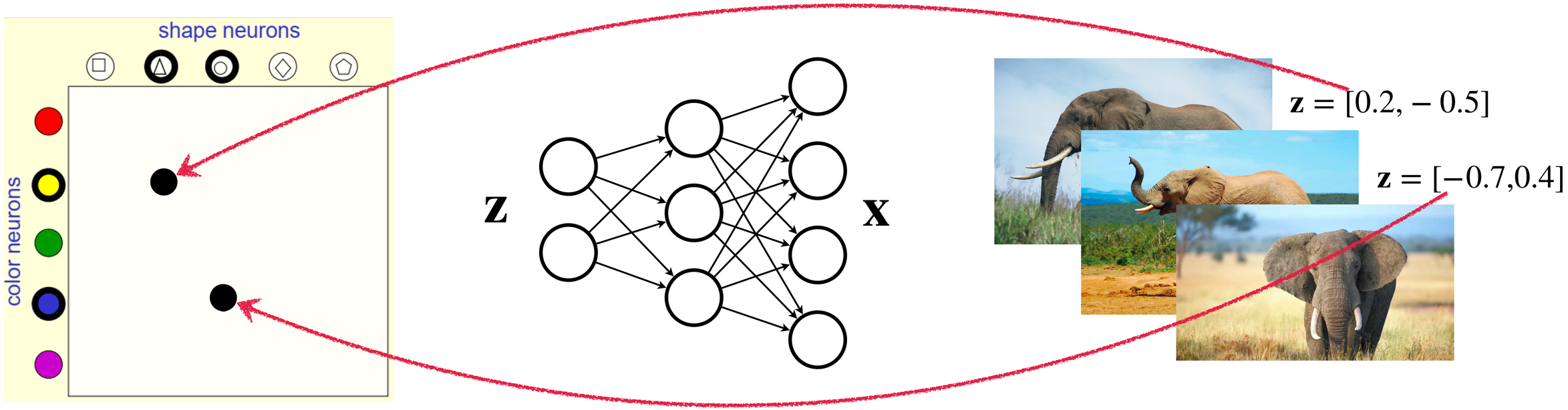






Distributed representations

- Our neural network learns to distribute salient features to multiple neurons



Variational Inference

From latent to joint likelihood to marginals

- Since we have latent variables, we have the joint likelihood: $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})$
- However, we are not really interested in $\mathbf{z}$, so what we really care is the

$$p(\mathbf{x}) = \int_z p(\mathbf{x}, \mathbf{z}) d\mathbf{z}$$

- The $p(\mathbf{x})$ is called the **marginal** because we integrate out/marginalise all possible influences of $\mathbf{z}$ on the joint likelihood

An example of marginal likelihood

- Let's assume x is the eye colour and z is the nationality and our joint likelihoods are
- If we wanted to find out the likelihood for each eye colour, we need to marginalise the nationality:

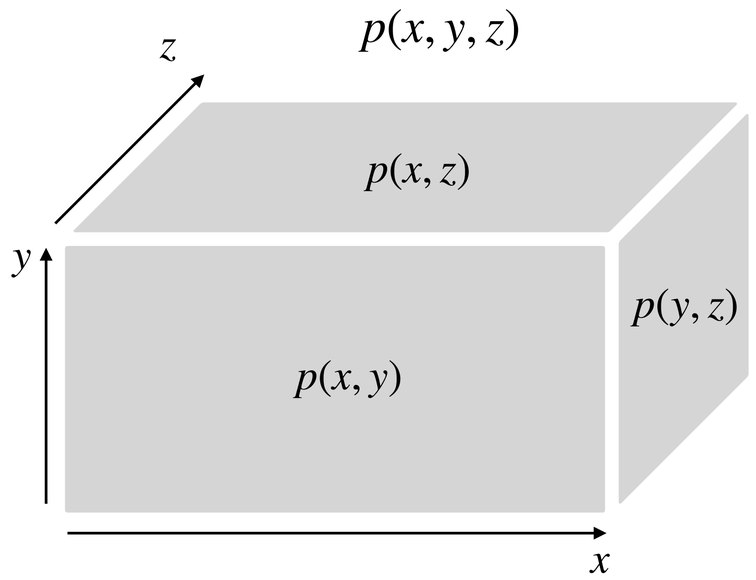
$$p(x) = \sum_{z} p(x, z)$$

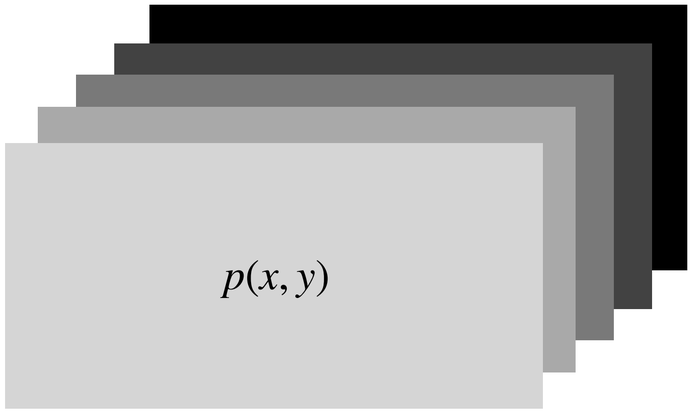
$p(x, z)$	**Dutch**	**Greek**	**Chinese**	**Indian**	**Italian**	**German**	**US**	**Spanish**
Brown	0.02	0.03	0.02	0.01	0.07	0.03	0.01	0.00
Blue	0.09	0.01	0.03	0.03	0.08	0.04	0.03	0.06
Green	0.01	0.01	0.08	0.06	0.07	0.08	0.06	0.06

Color	$p(x)$
Brown	0.19
Blue	0.37
Green	0.44
Total	1.00

Integrals and marginals geometrically

- Integrals are volumes
- Marginals are ‘collapsed’ volumes where the mass along a dimension is pushed on all others





From the definition of the mean

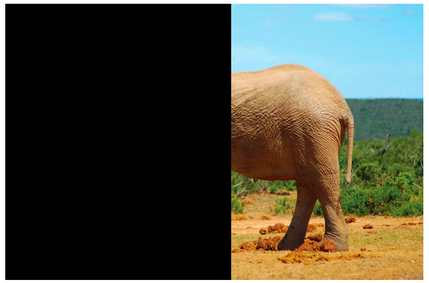
$$\mu_z = \mathbb{E}[z] = \frac{\sum z}{N_z}$$

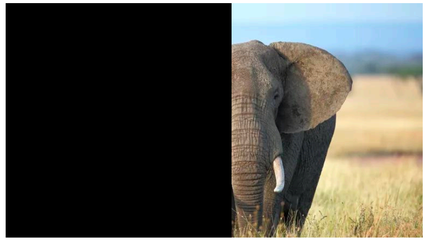
We can replace the sum by multiplying the average subvolume

$$p(x, y) = \sum p(x, y, z)$$
$$= |Z| \cdot \mathbb{E}_z[p(x, y, z)]$$

Another example

Find how likely these images are →
Marginalise out all possible latent pixels (all possible completions)





Computing the marginal is … intractable

- Let's say we want to maximise log-likelihood

$$\log p_\theta(\mathbf{x}) = \log \prod_j p_\theta(\mathbf{x}^{(j)}) = \sum_j \log \int_{\mathbf{z}} p_\theta(\mathbf{x}^{(j)}, \mathbf{z}) d\mathbf{z} \quad \text{or} \quad \sum_j \log \sum_i p_\theta(\mathbf{x}^{(j)}, \mathbf{z}^{(i)})$$

- The number of summands for discrete $\mathbf{z}$ soon becomes unfathomable
  - e.g., 2^{20} possible combinations for 20 dimensional $\mathbf{z} \in \{0,1\}^{20}$
- And if/when we must compute the integral, most likely it cannot be done tractably

We can always approximate: Naive Monte Carlo

- (Focusing on a single $\mathbf{x}^{(j)}$) we can rewrite the log-sum

$$\log \sum_{\mathbf{z}} p_\theta(\mathbf{x}^{(j)}, \mathbf{z}) = \log \underbrace{|Z|}_{\text{volume in z}} \cdot \underbrace{\mathbb{E}_z\left[p_\theta(\mathbf{x}, \mathbf{z})\right]}_{\text{mean over z}}$$

- Computationally, still the same: we need to still compute the exact mean over z
- We can save computations, however, by replacing the exact mean with sample mean
- This is called Monte Carlo estimation of the integral

Intermezzo:
Monte Carlo Estimation

Monte Carlo Estimation

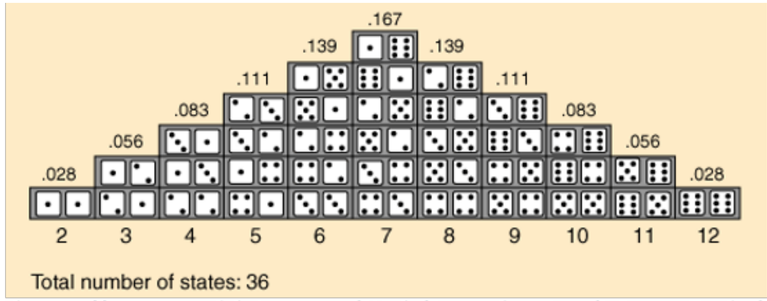
J. Von Neumann *S. Ulam* *N. Metropolis*

Manhattan Project

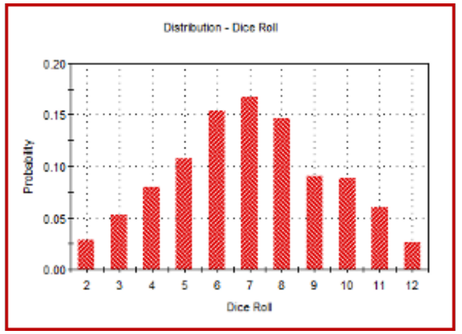
- Monte Carlo estimation was designed for problems with vast number of outcomes, whose distribution can be controlled statistically
  - ❖ Cannot derive a perfect drug response model (too complex)
  - ❖ Cannot enumerate all possible dice combinations (too lazy)
  - ❖ Computationally infeasible (intractable integrals)
- Examples: high-energy physics, finance, medicine, ML/DL

Key idea

- Instead of repeating computations exhaustively for all possible combinations, do random sampling of variables of interest and aggregate
  - One random sample will give a poor estimate
  - With more samples we increase computations but get a decent idea
- We want our estimator to have zero bias (the result to match the true expectation) and low variance (the result to not vary too much if repeat the experiment)



https://www.goldsim.com/Web/Introduction/MonteCarlo/



Key idea: more formally

- We want to compute a quantity like $y = \int_x f(x)p(x)dx$
- Variable x is random variable distributed according to the distribution $p(x)$
- This integral is basically also the definition of statistical expected value

$$y = \int_x f(x)p(x)dx = \mathbb{E}_{x\sim p(x)}\Big[f(x)\Big]$$

- Instead of computing the precise integral, we approximate it with a few samples

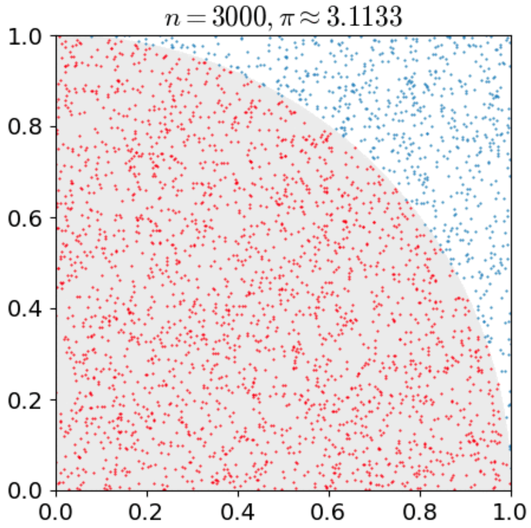
$$y = \int_x f(x)p(x)dx = \mathbb{E}_{x\sim p(x)}\Big[f(x)\Big] \approx \sum_i f(x^{(i)}) = \hat{y} \text{ for } x^{(i)} \text{ sampled from } p(x)$$

- The $\hat{y}$ is called an estimator because it estimates the true quantity y

Example: Monte Carlo estimation of π

- Estimate the value of π numerically by counting how many red dots over red and blue dots
  - Red dots are those with distance <1 from (0,0)
- Our estimator $\hat{y} = \frac{x_{\text{red}}}{x_{\text{red}} + x_{\text{blue}}}$ *estimates* the area of the circle quadrant compared to the whole square, for which we know the relation

$$y = \frac{\frac{1}{4}\pi r^2}{r^2} = \frac{\pi}{4} \Rightarrow \hat{\pi} = 4\hat{y}$$



Estimators are RVs themselves

- Since every time we repeat the whole Monte Carlo estimation we get a different result, the quantity is itself a random variable
- Thus, it also has its own mean and variance

$$\mu_{\hat{y}} = \mathbb{E}[\hat{y}] \text{ and } \text{Var}[\hat{y}] = \mathbb{E}\left[\left(\hat{y} - \mu_{\hat{y}}\right)^2\right]$$

- The estimator is unbiased if $\mu_{\hat{y}} = y$, otherwise it has bias bias $= \mu_{\hat{y}} - y$
-

Error and bias in MC estimators

- Unbiased is better, although sometimes quite hard to have (*e.g.* stochastic gradients)
- Then some bias is ok, especially if our neural network can work around this
- The MC estimators are unbiased due to law of large numbers
  - "As the number of identically distributed, randomly generated variables increases, their sample mean (average) approaches their theoretical mean."
- The standard error of the sample mean $\sigma_{\hat{y}} = \frac{\sigma}{\sqrt{n}}$ for n samples
- More samples better but with diminishing returns: 4x more samples $\Rightarrow$ 2x lower error

Monte Carlo in Deep Learning

- If we want to compute a quantity y
  - that we can express it as an integral of a function f over a probability space$x$
  - that has a known and easy to sample pdf $p(x)$
  - we can replace the exact intractable computation with a tractable MC estimator
- If we can't translate the quantity as such an integral over distributions, we can't estimate it

$$\int_x f(x) \nabla p(x) dx$$

- In Deep Learning many quantities can be casted as integrals over a distribution

Continuing with Variational Inference

We can always approximate: Naive Monte Carlo

- (Focusing on a single $\mathbf{x}^{(j)}$) we can rewrite the log-sum

$$\log \sum_{\mathbf{z}} p_\theta(\mathbf{x}^{(j)}, \mathbf{z}) = \log \underbrace{|Z|}_{\text{volume in z}} \cdot \underbrace{\mathbb{E}_z\left[p_\theta(\mathbf{x}, \mathbf{z})\right]}_{\text{mean over z}}$$

- Computationally, still the same: we need to still compute the exact mean over z
- We can save computations, however, by replacing the exact mean with sample mean
- This is called Monte Carlo estimation of the integral

We can always approximate: Naive Monte Carlo

- Instead of **computing** the exact mean, we **estimate** it with a few samples

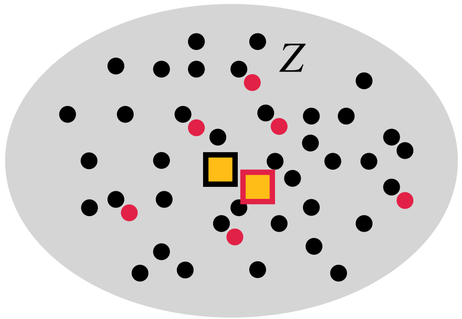
$$\log \sum_{\mathbf{z}} p_\theta(\mathbf{x}^{(j)}, \mathbf{z}) = \log \underbrace{|Z|}_{\text{volume in z}} \cdot \underbrace{\mathbb{E}_z\left[p_\theta(\mathbf{x}, \mathbf{z})\right]}_{\text{mean over z}}$$

$$\approx \frac{1}{K}|Z| \cdot \tilde{\mathbb{E}}^{(K)}_{z\sim\text{uniform}}\left[p_\theta(\mathbf{x}, \mathbf{z})\right]$$

$$= \frac{1}{K}|Z| \cdot \sum_{k=1}^{K} p_\theta(\mathbf{x}, \mathbf{z}^{(k)})$$

for K random samples $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(K)}$ from a uniform distribution

Divide by K to rescale because now our sample mean is not on all possible points in Z but only K samples

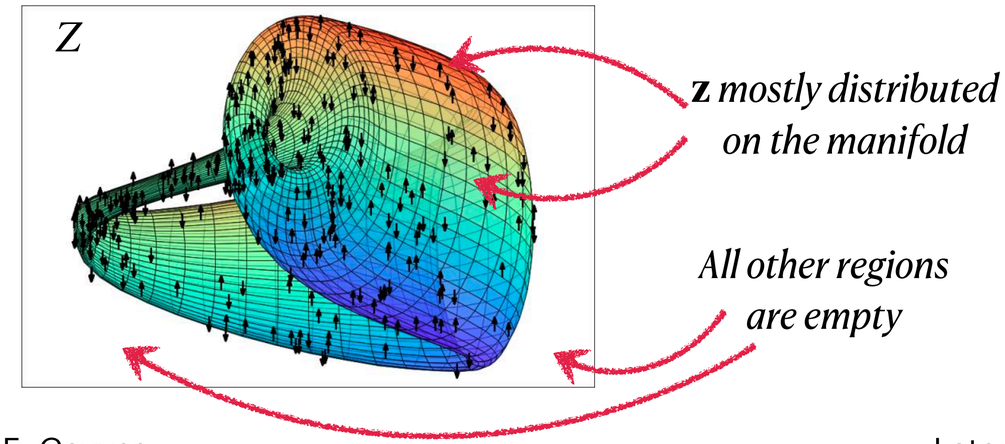


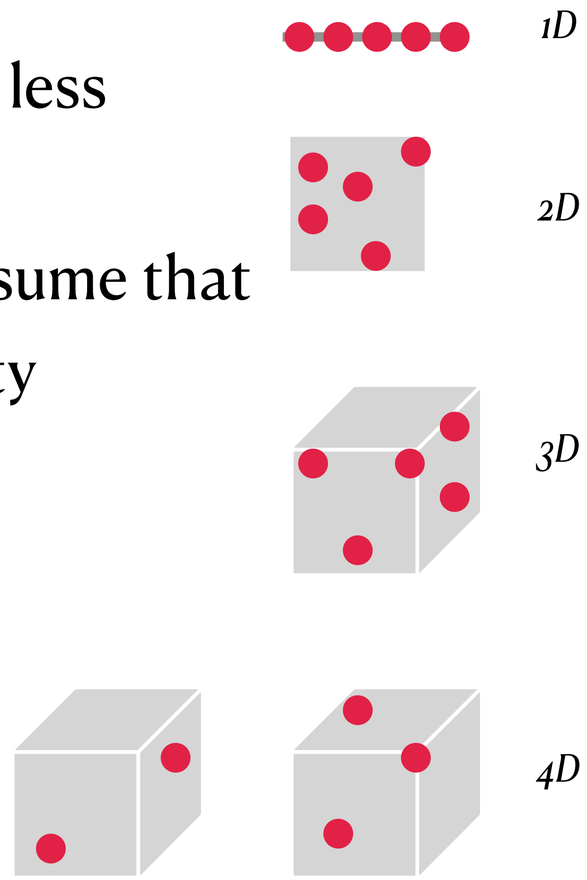
$$\mathbb{E}_z = \frac{\sum \bullet}{\#\,\bullet} \qquad \tilde{\mathbb{E}}_z^{(K=6)} = \frac{\sum \bullet}{\#\,\bullet}\ \}K$$

- We save a lot of compute since we only need K terms

Naive uniform sampling is still sample inefficient

- With increasing dimensions the same number is less and less representative of the whole space
- Thus, the sample mean is not reliable, especially if we assume that $\mathbf{z}$ is on a manifold, and most other regions are low density
- Our estimator has low (or zero bias) but high variance





Importance sampling vs uniform sampling

- We can instead sample from regions likely to contains lots of $\mathbf{z}$ by adding an importance sampling distribution $q_\phi(\mathbf{z})$

$$\log \sum_{\mathbf{z}} p_\theta(\mathbf{x}^{(j)}, \mathbf{z}) = \log \sum_{\mathbf{z}} \frac{q_\phi(\mathbf{z})}{q_\phi(\mathbf{z})} p_\theta(\mathbf{x}^{(j)}, \mathbf{z}) = \log \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z})} \left[\frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z})} \right] \approx \log \frac{1}{K} \sum_{k} \frac{p_\theta(\mathbf{x}, \mathbf{z}^{(k)})}{q_\phi(\mathbf{z}^{(k)})}$$

- We then use $q_\phi(\mathbf{z})$ as a distribution (instead of uniform) to sample from, e.g. a Gaussian, that hopefully returns more samples on the manifold
- And we use $q_\phi(\mathbf{z})$ as a rescaling factor (i.e., we use it as a function, not a sampler)
- Scales much better with dimensions and has much lower sample variance

Learning importance sampling distribution

- Still, we do not know the manifold ← What is our ϕ?
- Solution: we learn the distribution from the data as we go
- However, the quantity we estimate is not very convenient

$$\log \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right] = \log \int_{\mathbf{z}} \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})} q_\phi(\mathbf{z}) d\mathbf{z}$$

(1) draw samples (2) compute joint likelihood (3) rescale (4) sum

- It would be so much nicer if we can move the log in the integral like $\int \log p_\theta \ldots$
- Especially if our type of generative model is in the exponential family like the Gaussian

Learning importance sampling distribution

$$\log \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right] = \log \int_{\mathbf{z}} \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})} q_\phi(\mathbf{z}) d\mathbf{z}$$

(1) draw samples (2) compute joint likelihood (3) rescale (4) sum (5) log

- So much nicer if we could move the log in the integral
  - ❖ Especially if we choose $p_\theta(\mathbf{x},\mathbf{z})$ in the exponential family like the Gaussian
- Then, we could recast the computation as expectation of the logarithm of function and avoid costly computing expensive and unstable exponentials and logarithms

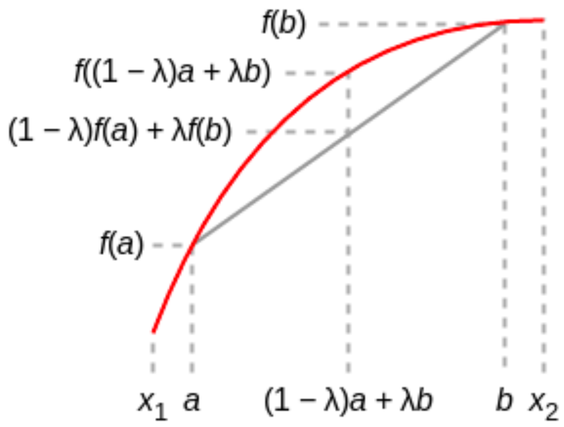
$$\int_{\mathbf{z}} \log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right) q_\phi(\mathbf{z}) d\mathbf{z} = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right]$$

Jensen's equality

- The line connecting two points of a concave function is always below the function

$$h(\lambda a + (1-\lambda)b) \geq \lambda h(a) + (1-\lambda)h(b)$$

- For convex functions the reverse is true



Lower-bounding the marginal

- A logarithm is a concave function and an expectation is a sum (or integral)

$$\log \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right] \geq \underbrace{\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right]}_{\text{Lower bound of marginal (or evidence)}}$$

- By maximising the lower bound, we are guaranteed to maximise the marginal
- Also, the lower bound is an expectation, so we can do Monte Carlo easily
- This *Evidence Lower BOund* (ELBO) is tractable, easy to optimise especially with a good choice for $p_\theta(\mathbf{x},\mathbf{z})$, and although approximate, the gap can be arbitrarily small

ELBO

- We replaced the uncomfortable integral from the joint likelihood with a comfortable one

$$\text{ELBO} = \mathscr{L}(\theta) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right]$$

$$= \mathbb{E}_{\mathbf{z}\sim q_\phi}\Big[\log p_\theta(\mathbf{x},\mathbf{z})\Big] + H(q_\phi(\mathbf{z}))$$

$$= \mathbb{E}_{\mathbf{z}\sim q_\phi}\Big[\log p_\theta(\mathbf{x}\,|\,\mathbf{z})\Big] - \mathrm{KL}(q_\phi(\mathbf{z})\|p(\mathbf{z}))$$

- We can write ELBO as function of the joint likelihood and entropy of approximate posterior
- We can write ELBO as function of the generative model and KL divergence of approximate posterior from the true posterior

Variational Autoencoders

ELBO: recap

- We replaced the uncomfortable integral from the joint likelihood with a comfortable one

$$\text{ELBO} = \mathscr{L}(\theta) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right]$$

$$= \mathbb{E}_{\mathbf{z}\sim q_\phi}\Big[\log p_\theta(\mathbf{x},\mathbf{z})\Big] + H(q_\phi(\mathbf{z}))$$

$$= \mathbb{E}_{\mathbf{z}\sim q_\phi}\Big[\log p_\theta(\mathbf{x}\,|\,\mathbf{z})\Big] - \mathrm{KL}(q_\phi(\mathbf{z})\|p(\mathbf{z}))$$

- We can write ELBO as function of the joint likelihood and entropy of approximate posterior
- We can write ELBO as function of the generative model and KL divergence of approximate posterior from the true posterior

ELBO as neural networks

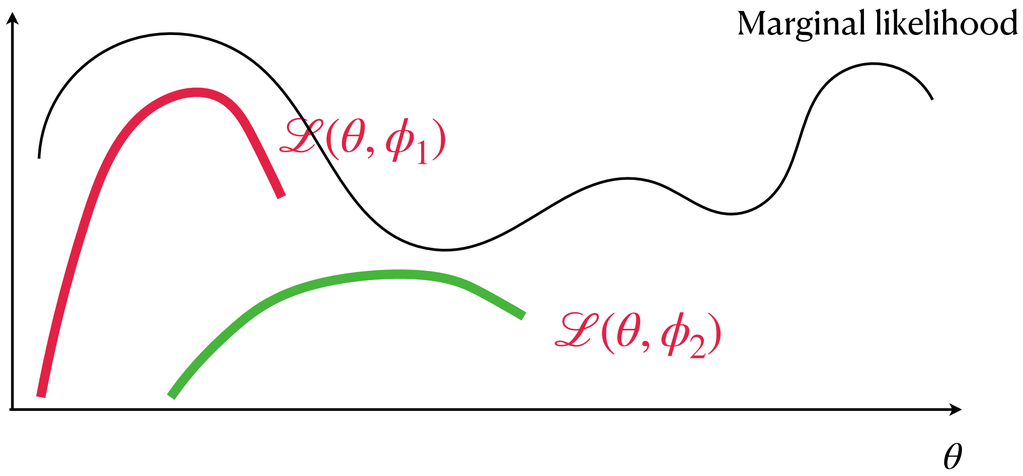
$$\text{ELBO} = \mathscr{L}(\theta) = \mathbb{E}_{\mathbf{z}\sim q_\phi}\Big[\log p_\theta(\mathbf{x}\,|\,\mathbf{z})\Big] - \text{KL}(q_\phi(\mathbf{z})\|p_\theta(\mathbf{z}\,|\,\mathbf{x}))$$

- Where can we place a neural network, that is a function approximation?
- We can use a neural network to implement our generative model $p_\theta(\mathbf{x}\,|\,\mathbf{z})$
- The generative model neural network receives as input "random noise", and should (after training) give a nice-looking image
- (After training, similar random codes for the same model should return similar images. So, it is random in that how the "randomness" is assigned per experiment is arbitrary.)

Many lower bounds satisfy the ELBO

$$\text{ELBO} = \mathscr{L}(\theta) = \mathbb{E}_{\mathbf{z}\sim q_\phi}\Big[\log p_\theta(\mathbf{x}\,|\,\mathbf{z})\Big] - \text{KL}(q_\phi(\mathbf{z})\|p_\theta(\mathbf{z}\,|\,\mathbf{x}))$$

- How we pick the importance sampling distribution $q_\phi(\mathbf{z})$ does not change the ELBO
- Maximise ELBO with respect to both ϕ and $\theta \Rightarrow \mathscr{L}(\theta, \phi)$



ELBO: from a single data point to a dataset

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{\mathbf{z}\sim q_\phi}\Big[\log p_\theta(\mathbf{x}\,|\,\mathbf{z})\Big] - \mathrm{KL}(q_\phi(\mathbf{z})\|p_\theta(\mathbf{z})) = \mathscr{L}(\mathbf{x};\theta,\phi)$$

- Since the ELBO is **per data point** $\mathbf{x}$, for maximising the likelihood we have

$$\sum_i \log p_\theta(\mathbf{x}^{(i)}) \geq \sum_i \mathscr{L}(\mathbf{x}^{(i)};\theta,\phi^{(i)})$$

- Note that we are maximising a different variational parameter $\phi^{(i)}$ per data point $\mathbf{x}^{(i)}$
- The reason is that for each different data point we have a different best possible approximation to the true posterior $p(\mathbf{z}\,|\,\mathbf{x})$

Learning with Stochastic Variational Inference

- We can optimise our variational inference model with SGD

$$\mathscr{L}(\mathbf{x}^{(i)};\theta,\phi^{(i)}) = \mathbb{E}_{\mathbf{z}\sim q_{\phi^{(i)}}}\Big[\log p_\theta(\mathbf{x}^{(i)},\mathbf{z})\Big] + H(q_{\phi^{(i)}}(\mathbf{z}))$$

using gradients $\nabla_\theta \mathscr{L}(\mathbf{x}^{(i)};\theta,\phi^{(i)})$ and $\nabla_{\phi^{(i)}} \mathscr{L}(\mathbf{x}^{(i)};\theta,\phi^{(i)})$

- Randomly select data point $\mathbf{x}^{(i)}$
- Optimise ELBO wrt $\phi^{(i)}$ until convergence: $\phi^{(i)} = \phi^{(i)} + \eta \nabla_{\phi^{(i)}} \mathscr{L}(\mathbf{x}^{(i)};\theta,\phi^{(i)})$
- Then compute $\nabla_\theta \mathscr{L}(\mathbf{x}^{(i)};\theta,\phi^{(i)})$
  update: $\theta = \theta + \eta \nabla_\theta \mathscr{L}(\mathbf{x}^{(i)};\theta,\phi^{(i)*})$
  and re-update $\phi^{(i)}$

Estimating the lower bound

$$\mathscr{L}(\mathbf{x}^{(i)};\theta,\phi^{(i)}) = \mathbb{E}_{\mathbf{z}\sim q_{\phi^{(i)}}}\Big[\log p_\theta(\mathbf{x}^{(i)},\mathbf{z})\Big] + H(q_{\phi^{(i)}}(\mathbf{z}))$$

- We can evaluate the lower bound by sampling a few $\mathbf{z}^{(1)},\mathbf{z}^{(2)},\ldots,\mathbf{z}^{(K)}$ and estimate

$$\mathscr{L}(\mathbf{x}^{(i)};\theta,\phi^{(i)}) \approx \frac{1}{K}\sum_{\mathbf{z}^{(k)}}\Big[\log p_\theta(\mathbf{x}^{(i)},\mathbf{z}^{(k)}) - \log q_{\phi^{(i)}}(\mathbf{z}^{(k)})\Big]$$

Assuming we can easily sample and evaluate $q_\phi(\mathbf{z})$

Computing gradients

$$\mathscr{L}(\mathbf{x}^{(i)};\theta,\phi^{(i)}) = \mathbb{E}_{\mathbf{z}\sim q_{\phi^{(i)}}}\Big[\log p_\theta(\mathbf{x}^{(i)},\mathbf{z})\Big] + H(q_{\phi^{(i)}}(\mathbf{z}))$$

- Computing $\nabla_\theta \mathscr{L}$ is not too hard

$$\nabla_\theta \mathscr{L}(\mathbf{x}^{(i)};\theta,\phi^{(i)}) = \nabla_\theta \mathbb{E}_{\mathbf{z}\sim q_{\phi^{(i)}}}\Big[\log p_\theta(\mathbf{x}^{(i)},\mathbf{z})\Big] + \nabla_\theta H(q_{\phi^{(i)}}(\mathbf{z}))$$

$$\approx \frac{1}{K}\sum_k \Big[\nabla_\theta \log p_\theta(\mathbf{x}^{(i)},\mathbf{z}^{(k)})\Big]$$

$$\approx \frac{1}{K}\sum_k \Big[\nabla_\theta \log p_\theta(\mathbf{x}^{(i)}\,|\,\mathbf{z}^{(k)})\Big]$$

- Again, we used Monte Carlo sampling to estimate the gradient

Computing gradients (2)

- Computing $\nabla_\phi \mathcal{L}$ is not straightforward: parameters ϕ are part of sampling

$$\nabla_\phi \mathcal{L}(\mathbf{x}^{(i)}; \theta, \phi^{(i)}) = \nabla_\phi \mathbb{E}_{\mathbf{z} \sim q_{\phi^{(i)}}} \Big[\log p_\theta(\mathbf{x}^{(i)}, \mathbf{z}) \Big] + \nabla_\phi H(q_{\phi^{(i)}}(\mathbf{z}))$$

- This means, we need to backpropagate via our sampling, as in the gradients must go through the sampling procedure
- However, there is no gradient for sampling
- Besides, we cannot even do Monte Carlo (*i.e.*, no density functions in the integrals), and we really like our Monte Carlo's

$$\nabla_\phi \mathbb{E}_{\mathbf{z} \sim q_{\phi^{(i)}}} \Big[\log p_\theta(\mathbf{x}^{(i)}, \mathbf{z}) \Big] = \int_{\mathbf{z}} \log p_\theta(\mathbf{x}^{(i)}, \mathbf{z}) \nabla_\phi q_{\phi^{(i)}}(\mathbf{z}) d\mathbf{z}$$

Solution: Reparameterization trick

- We want to compute the gradient of an integral containing a sampling operation

$$\nabla_\phi \mathbb{E}_{\mathbf{z}\sim q_{\phi^{(i)}}}\Big[\log p_\theta(\mathbf{x}^{(i)}, \mathbf{z})\Big] = \nabla_\phi \int_{\mathbf{z}} \log p_\theta(\mathbf{x}^{(i)}, \mathbf{z}) q_{\phi^{(i)}}(\mathbf{z}) d\mathbf{z}$$

- For some probability distributions we can rewrite them such that their randomness is generated from a "canonical" parameter-free random generator
- For Gaussian random variables

$$z \sim \mathcal{N}(\mu, \sigma) \Rightarrow z = \mu + \epsilon\sigma \quad \text{where} \quad \epsilon \sim \mathcal{N}(0,1)$$

- Or more generally, we write $z = g(\epsilon; \phi)$

Solution: Reparameterization trick

- Effectively, change the RV in the integral so that it does not depend on parameters

$$\nabla_\phi \mathbb{E}_{\mathbf{z}\sim q_{\phi^{(i)}}}\Big[\log p_\theta(\mathbf{x}^{(i)},\mathbf{z})\Big] = \nabla_\phi \int_{\mathbf{z}} q_{\phi^{(i)}}(\mathbf{z}) \log p_\theta(\mathbf{x}^{(i)},\mathbf{z}) d\mathbf{z}$$

$$= \nabla_\phi \int_{\epsilon} p(\epsilon) \log p_\theta\Big(\mathbf{x}^{(i)}, g(\epsilon;\phi)\Big) d\epsilon$$

$$= \mathbb{E}_{\epsilon\sim p(\epsilon)}\Big[\nabla_\phi \log p_\theta\Big(\mathbf{x}^{(i)}, g(\epsilon;\phi)\Big)\Big]$$

Solution: Reparameterization trick

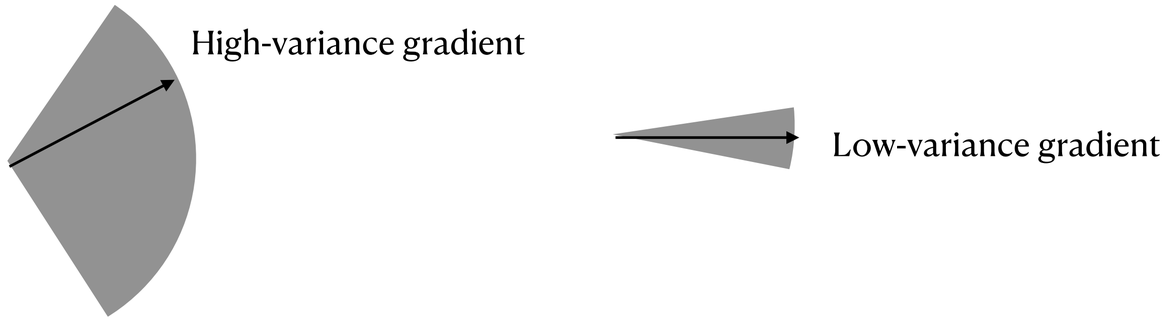
- We can now use a nice Monte Carlo estimation for our gradients

$$\nabla_\phi \mathbb{E}_{\mathbf{z}\sim q_{\phi^{(i)}}}\Big[\log p_\theta(\mathbf{x}^{(i)}, \mathbf{z})\Big] = \mathbb{E}_{\epsilon\sim p(\epsilon)}\Bigg[\nabla_\phi \log p_\theta\Big(\mathbf{x}^{(i)}, g(\epsilon;\phi)\Big)\Bigg]$$

$$= \frac{1}{K}\sum_k \Bigg[\nabla_\phi \log p_\theta\Big(\mathbf{x}^{(i)}, g(\epsilon^{(k)};\phi)\Big)\Bigg]$$

Low variance estimator

- Sampling from $\epsilon \ \mathcal{N}(0,1)$ leads to lower-variance estimates compared to $z \sim \mathcal{N}(\mu, \sigma)$
- Particularly important because in reality we also sample for gradients and we want our gradients to not vary too much



Amortised inference

- So far variational inference is directly an optimisation and optimise a different variational parameter $\phi^{(i)}$ per data point $\mathbf{x}^{(i)}$
- A more efficient way is to have a neural network $\phi_f^{(i)} = f(\mathbf{x}^{(i)})$ that learns to return "near-optimal" variational parameters
- Instead of having the optimal parameters $q(z)$ for our (approximate) variational distribution, we have near-optimal parameters for $q(\mathbf{z} \mid \mathbf{x})$ conditioned on input $\mathbf{x}$
- Not quite as the optimal ones, but we skip the optimisation loop over ϕ

Variational Inference vs Amortised Variational Inference

- In variational inference we want to find the optimal distribution parameters for our approximate posterior distribution

$$q_{\phi^{(i)}}(\mathbf{z}) \approx p_\theta(\mathbf{z} \mid \mathbf{x}^{(i)})$$

by directly optimising one variational parameter $\phi^{(i)}$ per data point $\mathbf{x}^{(i)}$ (but not really conditioning on the data point)

- In amortised variational inference we want to find near-optimal distribution parameters for our approximate posterior distribution

$$q_\phi(\mathbf{z} \mid \mathbf{x}^{(i)}) \approx p_\theta(\mathbf{z} \mid \mathbf{x}^{(i)})$$

By directing optimising a neural network with shared weights that condition on input $\mathbf{x}^{(i)}$

Learning with amortised variational inference

- We modify our ELBO for amortised variational inference

$$\text{ELBO} = \mathscr{L}(\theta, \phi) = \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}\Big[\log p_\theta(\mathbf{x} \,|\, \mathbf{z})\Big] - \text{KL}(q_\phi(\mathbf{z} \,|\, \mathbf{x}) \| p_\theta(\mathbf{z}))$$

and optimise using gradient descend (technically ascend)

- Initialise our generative NN parameters $\theta^{(0)}$ and variational NN parameters $\phi^{(0)}$
- Randomly sample a data point $\mathbf{x}^{(i)}$
- Compute gradients $\nabla_\theta \mathscr{L}(\theta, \phi)$ and $\nabla_\phi \mathscr{L}(\theta, \phi)$
- Update on gradient direction and repeat

VAE 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 $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M}, \boldsymbol{\epsilon})$

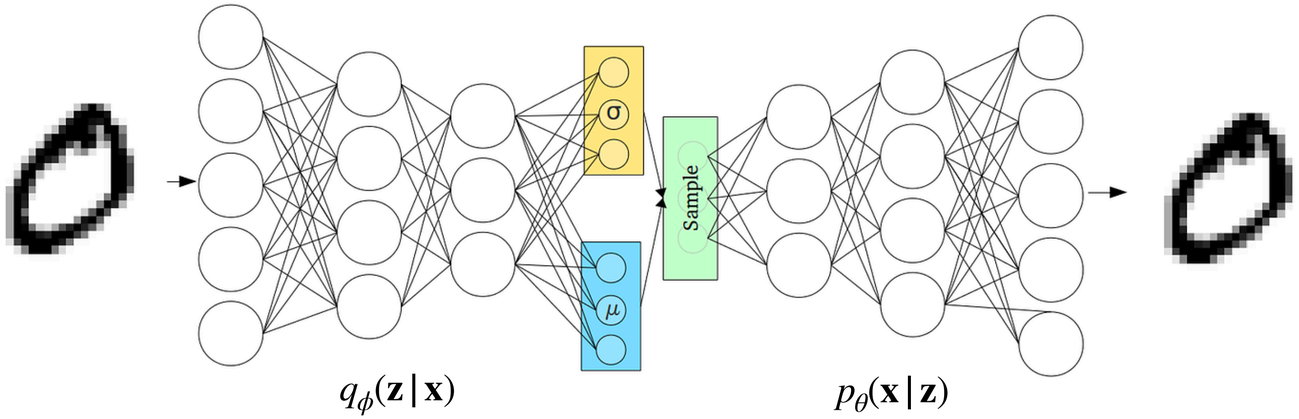
  Update $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ using SGD optimizer

end

The ELBO's gradients

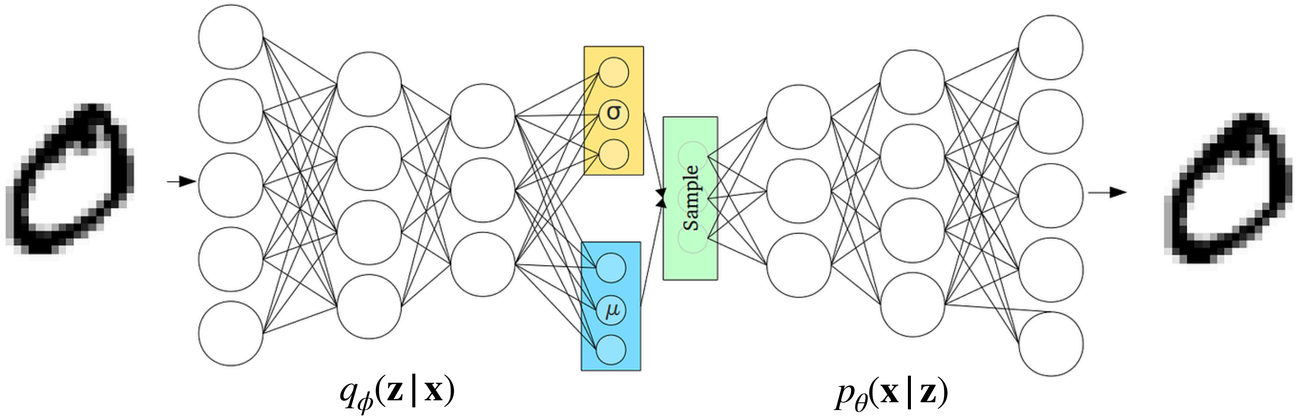
VAE Architecture

- Our approximate posterior (or inference) is a function (neural net) that receives as input an image and returns as output (encodes) a code …
- Which in aggregation with all other codes in the dataset resembles a distribution
- Our generative model (or decodes) is a function (neural net) that receives as input random noise and returns as output (or generates or decodes) an image



VAE architecture

- For Gaussian latent codes we have 1 NN for the μ parameter and 1 NN for σ
- The output is modelled according to the data type: *e.g.* continuous values often modelled as Gaussian variables, binary values as Bernoulli variables
- For generating novel images, simply sample new random samples $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$ (not use the approximate posterior) and decode

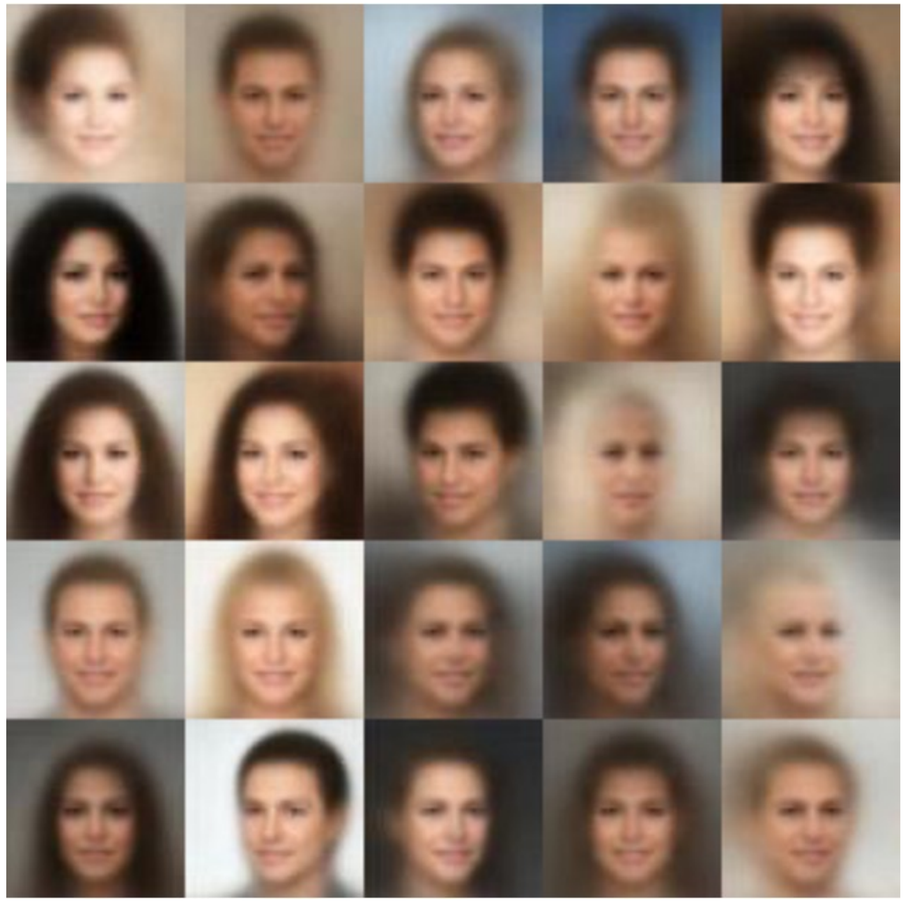


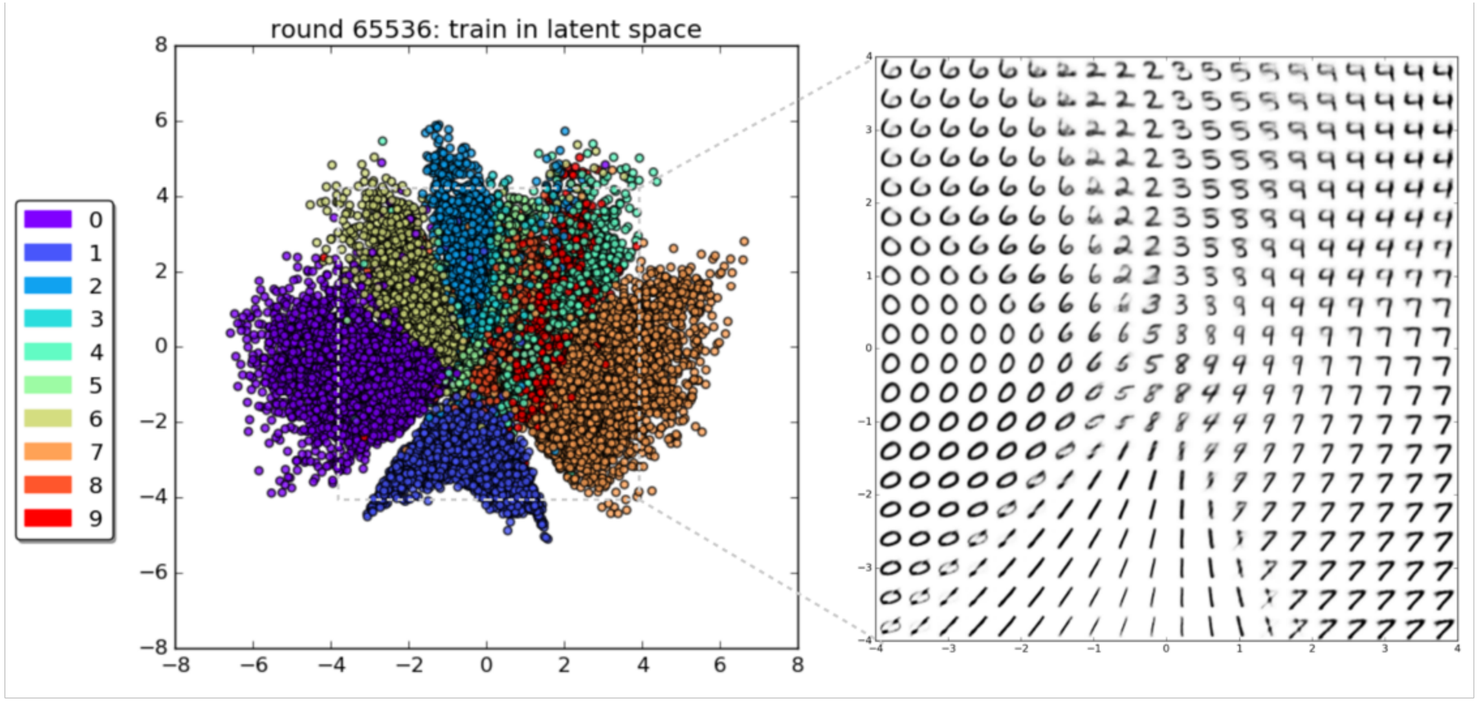
Prior distribution

- Pick a prior distribution that resembles the manifold we expect our latents to lie on
- For sparse latent for instance, we should not use a Gaussian prior
- The prior distribution can also be seen as a regulariser that makes sure latents are reasonably distributed in the space and do not overfit to specific values
- Often the KL divergence term $\mathrm{KL}(q_\phi(\mathbf{z}\,|\,\mathbf{x})\|p_\theta(\mathbf{z}))$ can be analytically derived, *e.g.* for Gaussian prior distribution
- When not, Monte Carlo: $\mathrm{KL}(q_\phi(\mathbf{z}\,|\,\mathbf{x})\|p_\theta(\mathbf{z})) = \int_{\mathbf{z}} q_\phi(\mathbf{z} \log \frac{q_\phi(\mathbf{z}\,|\,\mathbf{x})}{p_\theta(\mathbf{z})} d\mathbf{z}$

  at the cost of increased variance

Interpolation in latent space





A bit more on ELBO

Another way for intractability, another way for ELBO

- We derived ELBO so that to address the intractability in computing

$$\log \sum_{\mathbf{z}} p_\theta(\mathbf{x}^{(j)}, \mathbf{z}) \text{ or } \log \int_{\mathbf{z}} p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z}$$

- There exist other ways to describe our intractability (there is always an integral)
- For instance, let's say we are mainly interested in how to compute latent representations from the input

$$p(\mathbf{z} \mid \mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{x})}{p(\mathbf{x})} = \frac{p(\mathbf{x}, \mathbf{z})}{\int_{\mathbf{z}} p(\mathbf{x}, \mathbf{z})}$$

ELBO against intractability

- We can derive the ELBO by minimising the difference of the approximate from the intractable true posterior

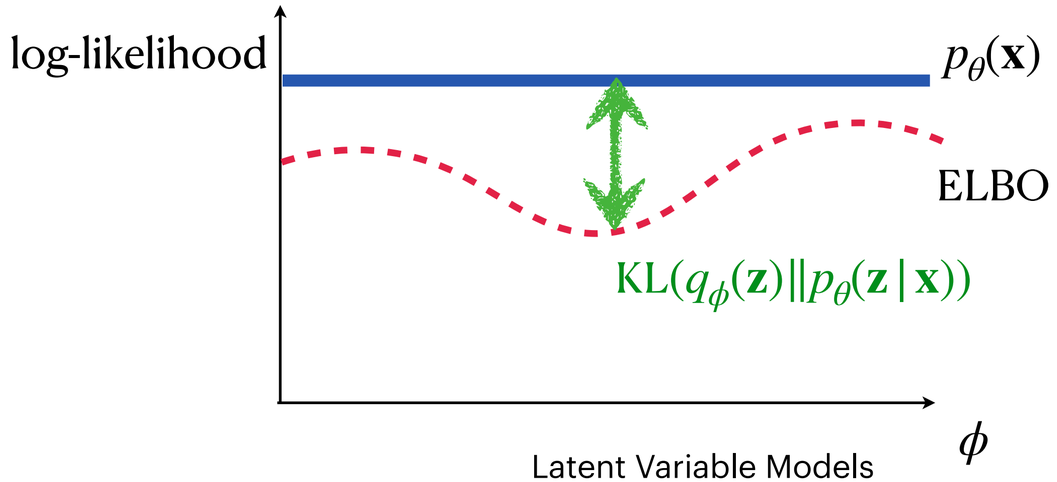
$$\begin{aligned}
\mathrm{KL}(q_\phi(\mathbf{z})\|p(\mathbf{z})) &= \int_z q_\phi(\mathbf{z})\log\frac{q_\phi(\mathbf{z})}{p(\mathbf{z}\,|\,\mathbf{x})}d\mathbf{z} = \int_z q_\phi(\mathbf{z})\log\frac{q_\phi(\mathbf{z})p(\mathbf{x})}{p(\mathbf{x},\mathbf{z})}d\mathbf{z} \\
&= -\int_z q_\phi(\mathbf{z})\log\frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})p(\mathbf{x})} \\
&= -\int_z q_\phi(\mathbf{z})\log\frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})} + \int_z q_\phi(\mathbf{z})\log p(\mathbf{x}) \\
&= -\mathbb{E}_{q_\phi(\mathbf{z})}\left[\log\frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right] + \log p(\mathbf{x}) \Rightarrow
\end{aligned}$$

$$\log \underbrace{p(\mathbf{x})}_{\text{Evidence}} \geq \underbrace{\mathbb{E}_{q_\phi(\mathbf{z})}\left[\log\frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right]}_{\text{ELBO}} \quad \text{since the KL term is always positive}$$

High ELBO means what?

- (By optimising a) Higher ELBO means that our approximate posterior gets closer to the true and intractable posterior

$$\log \underbrace{p_\theta(\mathbf{x})}_{\text{Evidence}} = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right]}_{\text{ELBO}} + \underbrace{\mathrm{KL}(q_\phi(\mathbf{z})\|p_\theta(\mathbf{z}\,|\,\mathbf{x}))}_{\text{Gap with true posterior}}$$



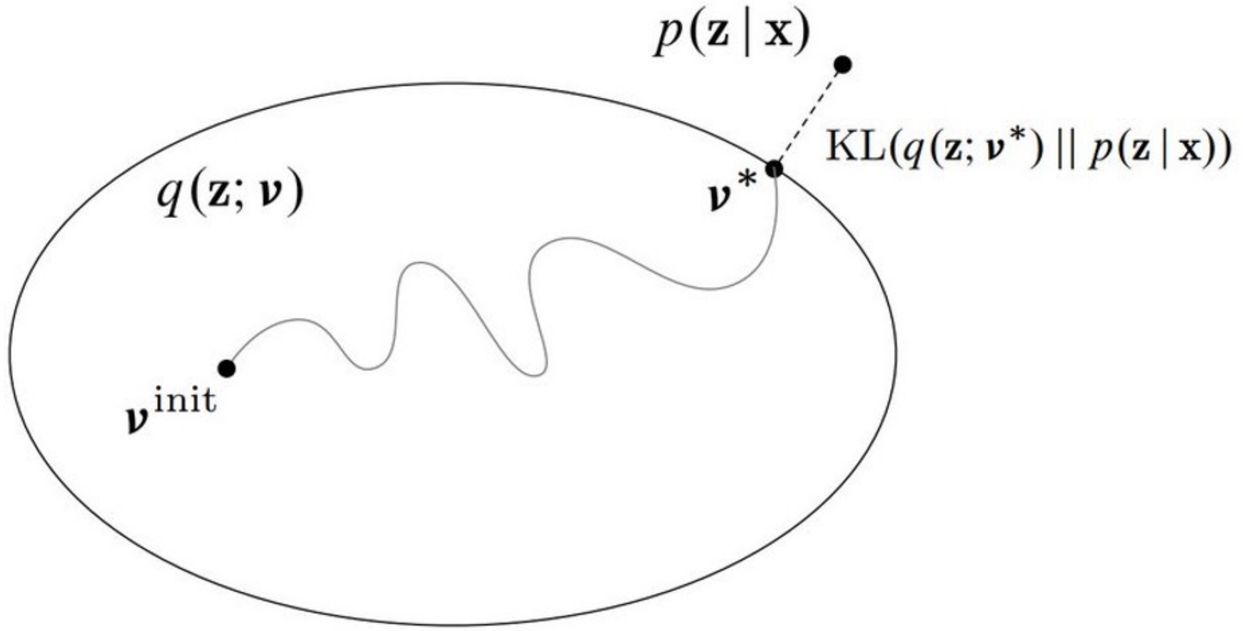
High ELBO means what?

$$\log \underbrace{p(\mathbf{x})}_{\text{Evidence}} = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right]}_{\text{ELBO}} + \underbrace{\mathrm{KL}(q_\phi(\mathbf{z})\|p(\mathbf{z}\,|\,\mathbf{x}))}_{\text{Gap with true posterior}} \geq \underbrace{\mathbb{E}_{q_\phi(\mathbf{z})}\left[\log \frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right]}_{\text{ELBO}}$$

- Higher ELBO means the gap becomes smaller → closer to true marginal likelihood
- Higher ELBO means that our latent vectors become better since they are closer to the true posterior

Variational Inference

Graphically



Interpreting ELBO: reconstructions and priors

$$\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right] = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\log \frac{p_\theta(\mathbf{x}\,|\,\mathbf{z})p(\mathbf{z})}{q_\phi(\mathbf{z})}\right]$$

$$= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\log p_\theta(\mathbf{x}\,|\,\mathbf{z})\right] - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\log \frac{q_\phi(\mathbf{z})}{p(\mathbf{z})}\right]$$

$$= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\log p_\theta(\mathbf{x}\,|\,\mathbf{z})\right] - \mathrm{KL}(q_\phi(\mathbf{z})\|p(\mathbf{z}))$$

- The ELBO tries to reconstruct as best as possible in the first term
- And the KL terms ensures that does not overfit but the approximate posterior is reasonable overall by being close to the prior

Interpreting ELBO: entropy regularisation

$$\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right] = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\log p_\theta(\mathbf{x},\mathbf{z})\right] - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\log q_\phi(\mathbf{z})\right]$$

$$= \underbrace{\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z})}\left[\log p_\theta(\mathbf{x},\mathbf{z})\right]}_{\text{Joint likelihood}} + \underbrace{H\left[q_\phi(\mathbf{z})\right]}_{\text{Entropy}}$$

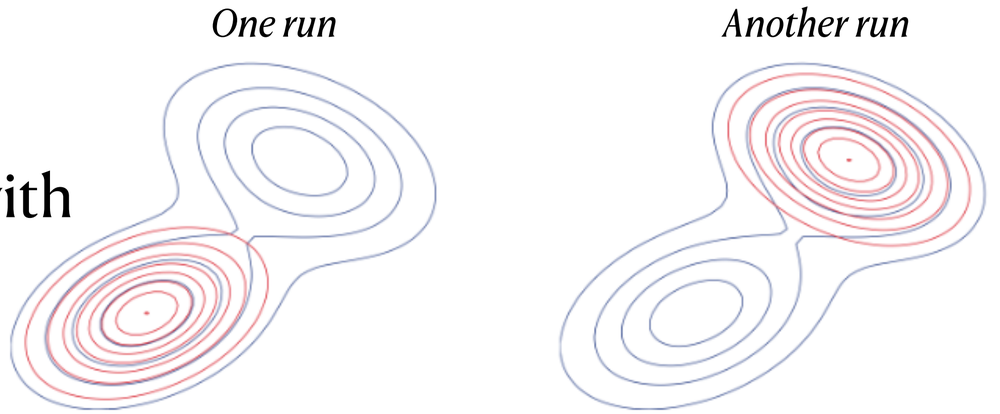
- The ELBO tries to maximise the joint likelihood on the observed points
- And via the entropy ensures the model spreads out and does not overfit to the training set

Variational Inference underestimates variance

- ELBO upper bounds the divergence of approximate and true posteriors

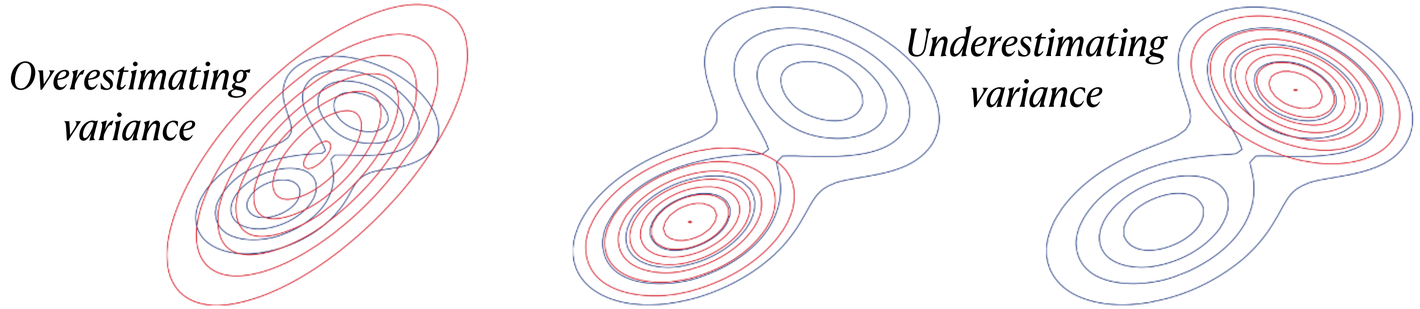
$$\mathrm{KL}(q_\phi(\mathbf{z})\|p(\mathbf{z}\,|\,\mathbf{x})) = \int_z q_\phi(\mathbf{z})\log\frac{q_\phi(\mathbf{z})}{p(\mathbf{z}\,|\,\mathbf{x})}d\mathbf{z}$$

- The model wants to approximate $p(\mathbf{z}\,|\,\mathbf{x})$ with $q_\phi(\mathbf{z})$ but it cannot know where $p(\mathbf{z}\,|\,\mathbf{x})$ is high and where low
- The model prefers to 'bias' $q_\phi(\mathbf{z})$ towards a single random mode rather than spread to regions with very low $p(\mathbf{z}\,|\,\mathbf{x})$ as $\frac{q_\phi(\mathbf{z})}{p(\mathbf{z}\,|\,\mathbf{x})}$ would go to infinity



Forward KL to overestimate variance

- Forward KL divergence causes the opposite $\mathrm{KL}(p\|q_\phi) = \int_z p(\mathbf{z}\,|\,\mathbf{x}) \log \frac{p(\mathbf{z}\,|\,\mathbf{x})}{q_\phi(\mathbf{z})} d\mathbf{z}$
- To prevent $\log \frac{p(\mathbf{z}\,|\,\mathbf{x})}{q_\phi(\mathbf{z})}$ from infinity by $q_\phi(\mathbf{z}) \to 0$ where $p(\mathbf{z}\,|\,\mathbf{x})$ is large, the model overestimate variance and cover all true modes with a single approximate one
- Not easy to model forward KL since we cannot use $p(\mathbf{z}\,|\,\mathbf{x})$ as a sampler



Tractability vs Flexibility

Wrapping up

- What are latent variable models
- Variational Inference
- Variational Autoencoders
- Reparameterization trick and variance reduction
- Insights