Scalable Bayesian methods

Dmitry Vetrov

Research professor at HSE

Senior researcher at Yandex

Head of Bayesian methods research group

http://bayesgroup.ru

Deep|Bayes

Outline

- Probabilistic PCA
- Variational auto-encoder
- REINFORCE and Reparemeterization trick
- Stochastic computation graphs

Continuous latent variables

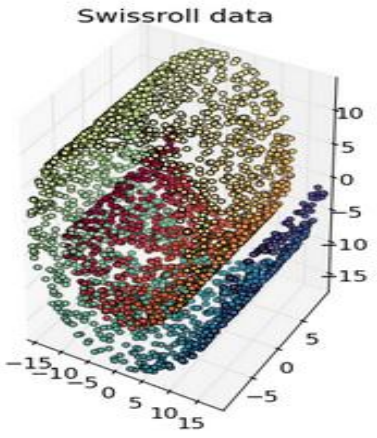
- Continuous varuables can be regarded as a mixture of a continuum of distributions

$$p(x_i|\theta) = \int p(x_i, z_i|\theta)dz_i = \int p(x_i|z_i, \theta)p(z_i|\theta)dz_i$$

- E-step can be done in closed form only in case of **conjugate distributions**, otherwise the true posterior is intractable

$$q(z_i) = p(z_i|x_i, \theta) = \frac{p(x_i|z_i, \theta)p(z_i|\theta)}{\int p(x_i|z_i, \theta)p(z_i|\theta)dz_i}$$

- Typically continuous latent variables are used for dimension reduction also known as **representation learning**

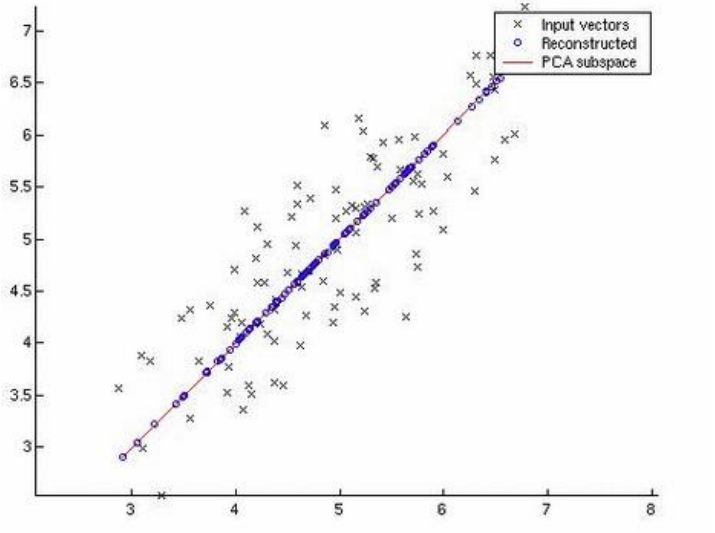


Example: PCA model

- Consider $x \in \mathbb{R}^D$, $z \in \mathbb{R}^d$, such that $D \gg d$
- Joint distribution

$$p(X, Z|\theta) = \prod_{i=1}^{n} p(x_i|z_i, \theta)p(z_i|\theta) = \prod_{i=1}^{n} \mathcal{N}(x_i|Vz_i, \sigma^2 I)\mathcal{N}(z_i|0, I)$$

- θ consists of $D \times d$ matrix V and scalar σ
- Can use EM-algorithm to find $\arg\max_\theta p(X_{tr}|\theta)$



Advantages of EM PCA

In PCA the explicit equation for θ can be obtained analytically. Then why use EM?..

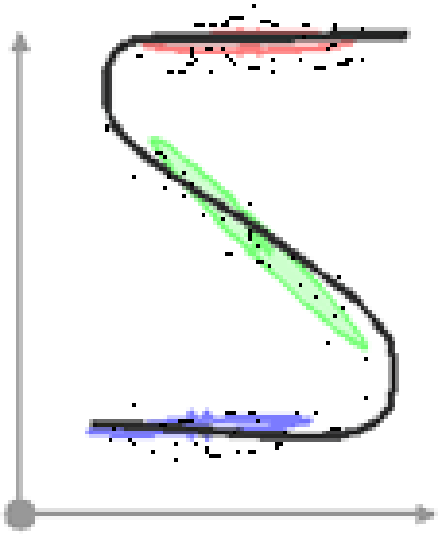
- EM updates have complexity $O(nDd)$ instead of $O(nD^2)$ in analytic solution
- Can process missing parts in X and present parts in Z
- Can determine d if $p(\theta)$ is established
- Can be extended to more general models such as mixture of PCA

Mixture of PCA

- Two types of latent variables: discrete $t \in \{1, \ldots, K\}$ and continuous $z \in \mathbb{R}^d$
- Joint distribution

$$p(X, Z, T|\theta) = \prod_{i=1}^{n} p(x_i|t_i, z_i, \theta)p(z_i|\theta)p(t_i|\theta) = \prod_{i=1}^{n} \mathcal{N}(x_i|V_{t_i} z_i, \sigma_{t_i}^2 I)\mathcal{N}(z_i|0, I)\pi_{t_i}$$

- θ consists of matrices $\{V_k\}$, scalars $\{\sigma_k\}$, and vector of probabilities θ such that $p(t_i = k) = \pi_k$
- Can be used for non-linear dimension reduction



EM-algorithm for mixture of PCA

- Two types of latent variables: discrete $t \in \{1, \ldots, K\}$ and continuous $z \in \mathbb{R}^d$
- Joint distribution

$$p(X, Z, T|\theta) = \prod_{i=1}^{n} p(x_i|t_i, z_i, \theta)p(t_i|\theta)p(z_i|\theta) = \prod_{i=1}^{n} \mathcal{N}(x_i|V_{t_i} z_i, \sigma_{t_i}^2 I)\mathcal{N}(z_i|0, I)\pi_{t_i}$$

- E-step: Estimate

$$q(Z, T) = \prod_{i=1}^{n} q(z_i, t_i) = \prod_{i=1}^{n} p(z_i, t_i|x_i, \theta)$$

EM-algorithm for mixture of PCA

- Two types of latent variables: discrete $t \in \{1, \ldots, K\}$ and continuous $z \in \mathbb{R}^d$
- Joint distribution

$$p(X, Z, T|\theta) = \prod_{i=1}^{n} p(x_i|t_i, z_i, \theta)p(t_i|\theta)p(z_i|\theta) = \prod_{i=1}^{n} \mathcal{N}(x_i|V_{t_i} z_i, \sigma_{t_i}^2 I)\mathcal{N}(z_i|0, I)\pi_{t_i}$$

- E-step: Estimate

$$q(Z, T) = \prod_{i=1}^{n} q(z_i, t_i) = \prod_{i=1}^{n} p(z_i, t_i|x_i, \theta) = \prod_{i=1}^{n} \frac{\mathcal{N}(x_i|V_{t_i} z_i, \sigma_{t_i}^2 I)\mathcal{N}(z_i|0, I)\pi_{t_i}}{\sum_{k=1}^{K} \int \mathcal{N}(x_i|V_{t_i} z_i, \sigma_{t_i}^2 I)\mathcal{N}(z_i|0, I)\pi_k dz_i}$$

EM-algorithm for mixture of PCA

- Two types of latent variables: discrete $t \in \{1, \ldots, K\}$ and continuous $z \in \mathbb{R}^d$
- Joint distribution

$$p(X, Z, T|\theta) = \prod_{i=1}^{n} p(x_i|t_i, z_i, \theta)p(t_i|\theta)p(z_i|\theta) = \prod_{i=1}^{n} \mathcal{N}(x_i|V_{t_i}z_i, \sigma_{t_i}^2 I)\mathcal{N}(z_i|0, I)\pi_{t_i}$$

- E-step: Estimate

$$q(Z, T) = \prod_{i=1}^{n} q(z_i, t_i) = \prod_{i=1}^{n} p(z_i, t_i|x_i, \theta) = \prod_{i=1}^{n} \frac{\mathcal{N}(x_i|V_{t_i}z_i, \sigma_{t_i}^2 I)\mathcal{N}(z_i|0, I)\pi_{t_i}}{\sum_{k=1}^{K} \int \mathcal{N}(x_i|V_{t_i}z_i, \sigma_{t_i}^2 I)\mathcal{N}(z_i|0, I)\pi_k dz_i}$$

- M-step: Optimize

$$\mathbb{E}_{Z,T} \log p(X, Z, T|\theta) = \sum_{i=1}^{n} \mathbb{E}_{z_i,t_i} \left(\log p(x_i|t_i, z_i, \theta) + \log p(z_i|\theta) + \log p(t_i|\theta)\right) \to \max_{\theta}$$

Non-linear model

- PCA contructs only linear subspaces while data often lives in non-linear manifolds
- When there is sufficient training data one may try more complicated probabilistic models for dimension reduction
- We may establish either simple prior on latent representations or simple generator

$$p(x, z|\theta) = p(x|z, \theta)p(z|\theta)$$

Generator Prior on latents

- Why not to use deep neural networks that are ideal for working with large datasets?

Variational auto-encoder

- Deep generalization of probabilistic PCA

$$p(X, Z|\theta) = \prod_{i=1}^{n} p(x_i|z_i, \theta)p(z_i) = \prod_{i=1}^{n} \left(\prod_{j=1}^{D} \mathcal{N}(x_{ij}|\mu_j(z_i), \sigma_j^2(z_i)) \right) \mathcal{N}(z_i|0, I)$$

- Generator returns factorized gaussian whose mean and variance are non-linear functions of latent variable modelled by deep neural network parameterized by θ
- Simple prior over latent representations and complicated generator
- Allows to convert complicated large-dimensional distributions into simple low-dimensional ones

Intractability problem

- Since the latent vaiables Z are unknown we cannot train the model directly
- If we could integrate out Z this would be possible but we cannot

$$\int p(X, Z|\theta)dZ = \prod_{i=1}^{n} \int p(x_i|z_i, \theta)p(z_i)dz_i$$

INTRACTABLE

- What about using EM-alrorithm?..

EM for VAE

- At E-step we need to compute a distribution over latent variables given data

$$q(Z) = \prod_{i=1}^{n} q(z_i) = \prod_{i=1}^{n} p(z_i|x_i, \theta) = \prod_{i=1}^{n} \frac{p(x_i|z_i, \theta)p(z_i)}{\int p(x_i|z_i, \theta)p(z_i)dz_i}$$

EM for VAE

- At E-step we need to compute a distribution over latent variables given data

$$q(Z) = \prod_{i=1}^{n} q(z_i) = \prod_{i=1}^{n} p(z_i|x_i, \theta) = \prod_{i=1}^{n} \frac{p(x_i|z_i, \theta)p(z_i)}{\int p(x_i|z_i, \theta)p(z_i)dz_i}$$

INTRACTABLE

- Denominator is still intractable
- But now we may use ideology of variational Bayes and convert inference problem to optimization!

Variational inference

- Instead of direct infering of $p(z_i|x_i, \theta)$ let us define flexible variational approximation

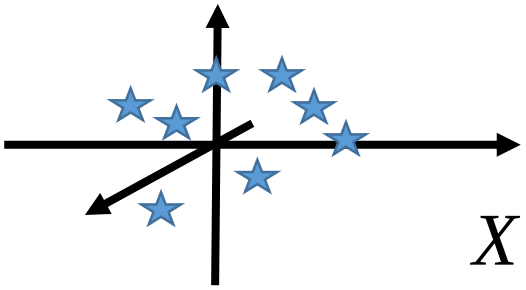
$$q(z_i|x_i, \phi) \approx p(z_i|x_i, \theta)$$

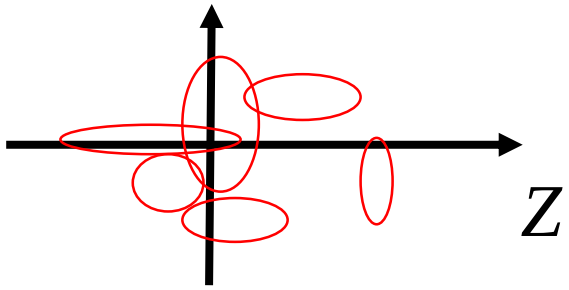
- The variational approximation has a form of factorized gaussian

$$q(z_i|x_i, \phi) = \prod_{j=1}^{d} \mathcal{N}(z_{ij}|\mu_j(x_i), \sigma_j^2(x_i)),$$

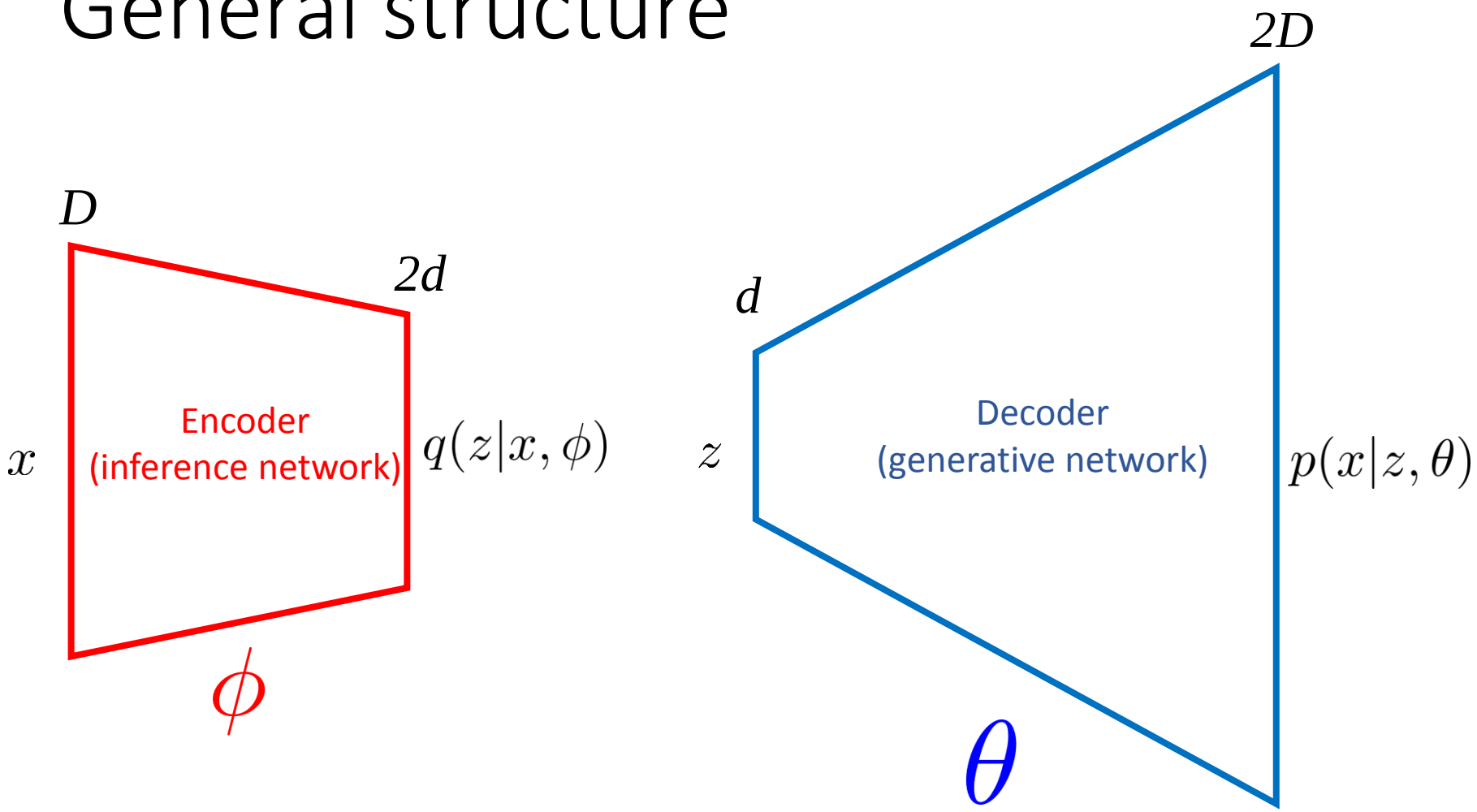
whose the mean and variance are given by **another neural network** parameterized by weights ϕ

- This additional neural network supplies tractability of the distribution while being very flexible





General structure



ELBO for VAE

- By performing variational approximation we minimize the corresponding KL-divergence

$$q(Z|X,\phi) = \arg\min_{\phi} KL(q(Z|X,\phi)||p(Z|X,\theta))$$

- This is equivalent to optimizing ELBO

$$\mathcal{L}(\phi,\theta) = \int q(Z|X,\phi)\log\frac{p(X|Z,\theta)p(Z)}{q(Z|X,\phi)}dZ \to \max_{\phi,\theta}$$

- Block-coordinate optimization w.r.t. encoder parameters ϕ corresponds to E-step
- Block-coordinate optimization w.r.t. decoder parameters θ corresponds to M-step

Stochastic optimization

$$\mathcal{L}(\phi, \theta) = \int q(Z|X, \phi) \log \frac{p(X|Z, \theta)p(Z)}{q(Z|X, \phi)} dZ$$

- Problem 1. The training data is assumed to be large $\Rightarrow$ iterations might be expensive
- Problem 2. The integral in ELBO is still intractable
- Solution: Compute stochastic gradients by using **mini-batching** and **Monte-Carlo** estimation

Optimization w.r.t. θ

$$\mathcal{L}(\phi, \theta) = \int q(Z|X, \phi) \log \frac{p(X|Z, \theta)p(Z)}{q(Z|X, \phi)} dZ$$

- Stochastic gradients w.r.t. θ can be obtained quite straightforwardly
- Mini-batching:

$$\nabla_\theta \mathcal{L}(\phi, \theta) = \nabla_\theta \sum_{i=1}^{n} \int q(z_i|x_i, \phi) \log \frac{p(x_i|z_i, \theta)p(z_i)}{q(z_i|x_i, \phi)} dz_i =$$

$$\sum_{i=1}^{n} \int q(z_i|x_i, \phi) \nabla_\theta \log p(x_i|z_i, \theta) dz_i \approx$$

$$n \int q(z_i|x_i, \phi) \nabla_\theta \log p(x_i|z_i, \theta) dz_i, \quad i \sim \mathcal{U}\{1, \ldots, n\}$$

Optimization w.r.t. θ

$$\mathcal{L}(\phi, \theta) = \int q(Z|X, \phi) \log \frac{p(X|Z, \theta)p(Z)}{q(Z|X, \phi)} dZ$$

- Stochastic gradients w.r.t. θ can be obtained quite straightforwardly
- Mini-batching:

$$\nabla_\theta \mathcal{L}(\phi, \theta) = \nabla_\theta \sum_{i=1}^{n} \int q(z_i|x_i, \phi) \log \frac{p(x_i|z_i, \theta)p(z_i)}{q(z_i|x_i, \phi)} dz_i =$$

$$\sum_{i=1}^{n} \int q(z_i|x_i, \phi) \nabla_\theta \log p(x_i|z_i, \theta) dz_i \approx$$

$$n \int q(z_i|x_i, \phi) \nabla_\theta \log p(x_i|z_i, \theta) dz_i, \quad i \sim \mathcal{U}\{1, \ldots, n\}$$

- Monte-Carlo estimation

$$n \int q(z_i|x_i, \phi) \nabla_\theta \log p(x_i|z_i, \theta) dz_i \approx n \nabla_\theta \log p(x_i|z_i^*, \theta), \quad z_i^* \sim q(z_i|x_i, \phi)$$

Optimization w.r.t. ϕ

- Stochastic gradient w.r.t. θ

  $$\text{stoch. grad}_\theta \mathcal{L}(\phi, \theta) = n \nabla_\theta \log p(x_i | z_i^*, \theta), \quad i \sim \mathcal{U}\{1, \ldots, n\}, \quad z_i^* \sim q(z_i | x_i, \phi)$$

- For differentiation w.r.t. ϕ the situation is different
- The density function itself depends on the parameters w.r.t. which we want to differentiate the expression
- Can no longer move gradient inside integral

$$\nabla_x \int p(y|x) h(x, y) dy \neq \int p(y|x) \nabla_x h(x, y) dy$$

Optimization w.r.t. ϕ

- Stochastic gradient w.r.t. θ

  $$\text{stoch. grad}_\theta \mathcal{L}(\phi, \theta) = n \nabla_\theta \log p(x_i | z_i^*, \theta), \quad i \sim \mathcal{U}\{1, \ldots, n\}, \quad z_i^* \sim q(z_i | x_i, \phi)$$

- For differentiation w.r.t. ϕ the situation is different
- The density function itself depends on the parameters w.r.t. which we want to differentiate the expression
- Can no longer move gradient inside integral

$$\nabla_x \int p(y|x) h(x, y) dy \neq \int p(y|x) \nabla_x h(x, y) dy$$

- Question: Why can't we simply do

$$\nabla_x \int p(y|x) h(x, y) dy = \int \nabla_x (p(y|x) h(x, y)) dy?$$

Stochastic gradients

Function	Stochastic gradient
$f(x) = \sum_{i=1}^{N} f_i(x)$	$N \nabla f_i(x), \quad i \sim \mathcal{U}\{1, \dots, n\}$
$f(x) = \mathbb{E}_y h(x, y) = \int p(y) h(x, y) dy$	$\frac{\partial}{\partial x} h(x, y_0), \quad y_0 \sim p(y)$
$f(x) = \sum_{i=1}^{N} \mathbb{E}_y h(x_i, y) = \sum_{i=1}^{N} \int p(y) h_i(x, y) dy$	$N \frac{\partial}{\partial x} h_i(x, y_0), \quad y_0 \sim p(y)$
$f(x) = \mathbb{E}_{y\|x} h(x, y) = \int p(y\|x) h(x, y) dy$	$\frac{\partial}{\partial x} h(x, y_0) + h(x, y_0) \frac{\partial}{\partial x} \log p(y_0\|x)$

Last example has extremely large variance!
Variance reduction is needed

Log-derivative trick

Consider differentiation of $\mathbb{E}_{y|x} h(x, y)$ in details

$$\frac{\partial}{\partial x} \int p(y|x) h(x, y) dy = \int \frac{\partial}{\partial x} \left(p(y|x) h(x, y) \right) dy =$$

$$\int \left(h(x, y) \frac{\partial}{\partial x} p(y|x) + p(y|x) \frac{\partial}{\partial x} h(x, y) \right) dy =$$

$$\int p(y|x) \frac{\partial}{\partial x} h(x, y) dy + \int h(x, y) \frac{\partial}{\partial x} p(y|x) dy$$

Log-derivative trick

Consider differentiation of $\mathbb{E}_{y|x} h(x, y)$ in details

$$\frac{\partial}{\partial x} \int p(y|x) h(x, y) dy = \int \frac{\partial}{\partial x} \left(p(y|x) h(x, y) \right) dy =$$

$$\int \left(h(x, y) \frac{\partial}{\partial x} p(y|x) + p(y|x) \frac{\partial}{\partial x} h(x, y) \right) dy =$$

$$\int p(y|x) \frac{\partial}{\partial x} h(x, y) dy + \int h(x, y) \frac{\partial}{\partial x} p(y|x) dy$$

The first term is ok since it can be replaced with a Monte Carlo estimate of expectation. To deal with second term we need to use **log-derivative trick**

$$\frac{\partial}{\partial x} p(y|x) = p(y|x) \frac{\partial}{\partial x} \log p(y|x)$$

Log-derivative trick

Consider differentiation of $\mathbb{E}_{y|x} h(x, y)$ in details

$$\frac{\partial}{\partial x} \int p(y|x) h(x, y) dy = \int \frac{\partial}{\partial x} \left(p(y|x) h(x, y) \right) dy =$$

$$\int \left(h(x, y) \frac{\partial}{\partial x} p(y|x) + p(y|x) \frac{\partial}{\partial x} h(x, y) \right) dy =$$

$$\int p(y|x) \frac{\partial}{\partial x} h(x, y) dy + \int h(x, y) \frac{\partial}{\partial x} p(y|x) dy$$

The first term is ok since it can be replaced with a Monte Carlo estimate of expectation. To deal with second term we need to use **log-derivative trick**

$$\frac{\partial}{\partial x} p(y|x) = p(y|x) \frac{\partial}{\partial x} \log p(y|x)$$

Appying the trick yields to

$$\frac{\partial}{\partial x} \int p(y|x) h(x, y) dy = \int p(y|x) \frac{\partial}{\partial x} h(x, y) dy + \int p(y|x) h(x, y) \frac{\partial}{\partial x} \log p(y|x) dy \approx$$

$$\frac{\partial}{\partial x} h(x, y_0) + h(x, y_0) \frac{\partial}{\partial x} \log p(y_0|x), \quad y_0 \sim p(y|x)$$

Log-derivative trick for ELBO

$$\frac{\partial}{\partial x}\int p(y|x)h(x,y)dy \approx \frac{\partial}{\partial x}h(x,y_0) + h(x,y_0)\frac{\partial}{\partial x}\log p(y_0|x)$$

- Now remember our ELBO

$$\mathcal{L}(\phi,\theta) = \int q(Z|X,\phi)\log\frac{p(X|Z,\theta)p(Z)}{q(Z|X,\phi)}dZ = \int q(Z|X,\phi)\log p(X|Z,\phi)dZ - KL(q(Z|X,\phi)||p(Z))$$

- Note that second term is simply a sum of KL-divergences between factorized gaussians $q(z_i|x_i,\phi)$ and $p(z_i)$ that can be computed and differentiated in closed form

- Consider the first term and apply mini-batching and log-derivative trick

$$\frac{\partial}{\partial\phi}\int q(Z|X,\phi)\log p(X|Z,\theta)dZ \approx \log p(x_i|z_i^*,\theta)\frac{\partial}{\partial\phi}\log q(z_i^*|x_i,\phi),$$

where $i \sim \mathcal{U}\{1,\ldots,n\}$ and $z_i^* \sim q(z_i|\phi)$

Log-derivative trick for ELBO

$$\frac{\partial}{\partial x}\int p(y|x)h(x,y)dy \approx \frac{\partial}{\partial x}h(x,y_0) + h(x,y_0)\frac{\partial}{\partial x}\log p(y_0|x)$$

- Now remember our ELBO

$$\mathcal{L}(\phi,\theta) = \int q(Z|X,\phi)\log\frac{p(X|Z,\theta)p(Z)}{q(Z|X,\phi)}dZ = \int q(Z|X,\phi)\log p(X|Z,\phi)dZ - KL(q(Z|X,\phi)||p(Z))$$

- Note that second term is simply a sum of KL-divergences between factorized gaussians $q(z_i|x_i,\phi)$ and $p(z_i)$ that can be computed and differentiated in closed form

- Consider the first term and apply mini-batching and log-derivative trick

$$\frac{\partial}{\partial\phi}\int q(Z|X,\phi)\log p(X|Z,\theta)dZ \approx \log p(x_i|z_i^*,\theta)\frac{\partial}{\partial\phi}\log q(z_i^*|x_i,\phi),$$

where $i \sim \mathcal{U}\{1,\ldots,n\}$ and $z_i^* \sim q(z_i|\phi)$

ACHTUNG!

REINFORCE

ACHTUNG!

$$\frac{\partial}{\partial \phi} \int q(Z|X,\phi) \log p(X|Z,\theta) dZ \approx \log p(x_i|z_i^*,\theta) \frac{\partial}{\partial \phi} \log q(z_i^*|x_i,\phi),$$

- Term $\log p(z_i^*|x_i,\theta))$ can be arbitrary large negative that leads to very unstable stochastic gradients
- A partial solution is to use **baselines**
- Consider a function $b(z_i,\phi)$ such that

$$\int q(z_i|\phi) b(z_i,\phi) dz_i = 0$$

- A good example is

$$b(z_i,\phi) = B(\phi) \frac{\partial}{\partial \phi} \log q(z_i|\phi)$$

the so-called **score function**

REINFORCE

Then

$$\frac{\partial}{\partial \phi} \int q(z_i|x_i, \phi) \log p(x_i|z_i, \theta) dz_i = \frac{\partial}{\partial \phi} \int q(z_i|x_i, \phi)(\log p(x_i|z_i, \theta) - b(z_i, \phi)) dz_i \approx$$

$$(\log p(x_i|z_i^*, \theta) - B(\phi)) \frac{\partial}{\partial \phi} \log q(z_i^*|x_i, \phi),$$

where $z_i^* \sim q(z_i|x_i, \phi)$

- We can make baseline data-dependent $B(\phi, x_i)$
- If baseline $B(\phi, x_i)$ is close to $\mathbb{E}_{z_i} \log p(x_i|z_i, \theta)$ then it may reduce the variance of stochastic gradient
- Unfortunately still impractical for high-dimensional z

Reparameterization trick

- Consider differentiation of complex expectation

$$\frac{\partial}{\partial x} \int p(y|x)h(x,y)dy$$

- Express y as a deterministic function $g(.)$ of random ϵ and x and perform change-of-variables rule

$$\int p(y|x)h(x,y)dx = \int r(\epsilon)h(x,g(\epsilon,x))d\epsilon$$

- Then stochastic differentiation is simply

$$\frac{\partial}{\partial x} \int p(y|x)h(x,y)dx = \frac{\partial}{\partial y} \int r(\epsilon)h(x,g(\epsilon,x))d\epsilon \approx \frac{d}{dx} h(x,g(x,\hat{\epsilon})),$$

  where $\hat{\epsilon} \sim r(\epsilon)$

- Such **reparameterization trick** allows to reduce variace of stochastic gradient by orders of magnitude!

Examples of repameterization

$p(x\|y)$	$r(\epsilon)$	$g(\epsilon, y)$
$\mathcal{N}(x\|\mu, \sigma^2)$	$\mathcal{N}(\epsilon\|0, 1)$	$x = \sigma\epsilon + \mu$
$\mathcal{G}(x\|1, \beta)$	$\mathcal{G}(\epsilon\|1, 1)$	$x = \beta\epsilon$
$\mathcal{E}(x\|\lambda)$	$\mathcal{U}(\epsilon\|0, 1)$	$x = -\frac{\log \epsilon}{\lambda}$
$\mathcal{N}(x\|\mu, \Sigma)$	$\mathcal{N}(\epsilon\|0, I)$	$x = A\epsilon + \mu$, where $AA^T = \Sigma$

- Not all continuous distributions can be effectively reparameterized
- Discrete distiributions **cannot** be reparameterized

Reparameterization trick for ELBO

- Return to the first term in our ELBO

$$\int q(Z|X,\phi)\log p(X|Z,\theta)dZ$$

- To get stochastic gradient w.r.t. ϕ first apply mini-batching

$$\frac{\partial}{\partial\phi}\int q(Z|X,\phi)\log p(X|Z,\theta)dZ \approx n\frac{\partial}{\partial\phi}\int q(z_i|x_i,\phi)\log p(x_i|z_i,\theta),$$

where $i \sim \mathcal{U}\{1,\ldots,N\}$

Reparameterization trick for ELBO

- Return to the first term in our ELBO

$$\int q(Z|X,\phi)\log p(X|Z,\theta)dZ$$

- To get stochastic gradient w.r.t. ϕ first apply mini-batching

$$\frac{\partial}{\partial\phi}\int q(Z|X,\phi)\log p(X|Z,\theta)dZ \approx n\frac{\partial}{\partial\phi}\int q(z_i|x_i,\phi)\log p(x_i|z_i,\theta),$$

  where $i \sim \mathcal{U}\{1,\ldots,N\}$

- Now perform reparameterization trick $z_i = g(\epsilon, x_i, \phi)$
- Then

$$n\frac{\partial}{\partial\phi}\int q(z_i|x_i,\phi)\log p(x_i|z_i,\theta) = n\frac{\partial}{\partial\phi}\int r(\epsilon)\log p(x_i|g(\epsilon,x_i,\phi)z_i,\theta)d\epsilon \approx$$

$$n\frac{\partial}{\partial\phi}\log p(x_i|g(\hat{\epsilon},x_i,\phi)z_i,\theta), \quad \hat{\epsilon}\sim r(\epsilon)$$

VAE: final algorithm

- Input: Training data X, dimension of latent space d
- Pick random $i \sim \mathcal{U}\{1, \ldots, n\}$ and compute stochastic gradients of ELBO w.r.t. θ and ϕ
  - Differentiate w.r.t. θ

$$\text{stoch.grad}_\theta \mathcal{L}(\phi, \theta) = n \frac{\partial}{\partial \theta} \log p(x_i | z_i^*, \theta),$$

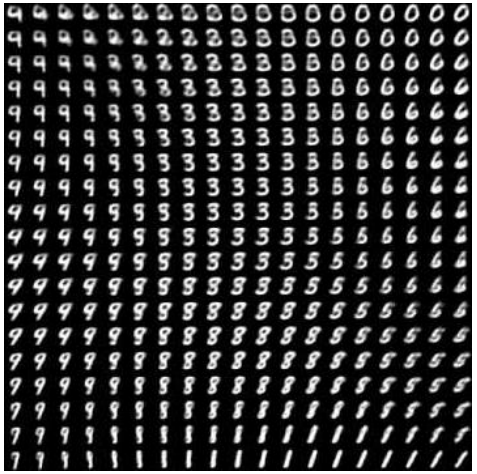
    where $z_i^* \sim q(z_i | x_i, \phi)$
  - Differentiate w.r.t. ϕ

$$\text{stoch.grad}_\phi \mathcal{L}(\phi, \theta) = n \frac{\partial}{\partial \phi} \log p(x_i | g(\hat{\epsilon}, x_i, \phi), \theta) - \frac{\partial}{\partial \phi} KL(q(z_i | x_i, \phi) || p(z_i)),$$

    where $\hat{\epsilon} \sim r(\epsilon)$
- Update θ and ϕ according to selected stochastic optimization method

Advantages of VAE

- Better representation learning in terms of marginal likelihood $p(X|\theta)$ and visual feeling
- New basic model for multiple extensions
  - importance reweighted auto-encoders (Burda15)
  - adversarial auto-encoders (Makhzani15)
  - auto-encoders with normalizing flows (Rezende15)
  - Structured VAE (Johnson16)
  - Grammar VAE(Kusner17)
- Opens way for more accurate variational lower bounds



VAE vs. GAN

Pros:

- No mode-collapsing
- More robust training
- May restore latent representation of real object
- Better interpolation between instances

Cons:

- GANs generate more realistic objects
- The divergence measure is trainable in GANs

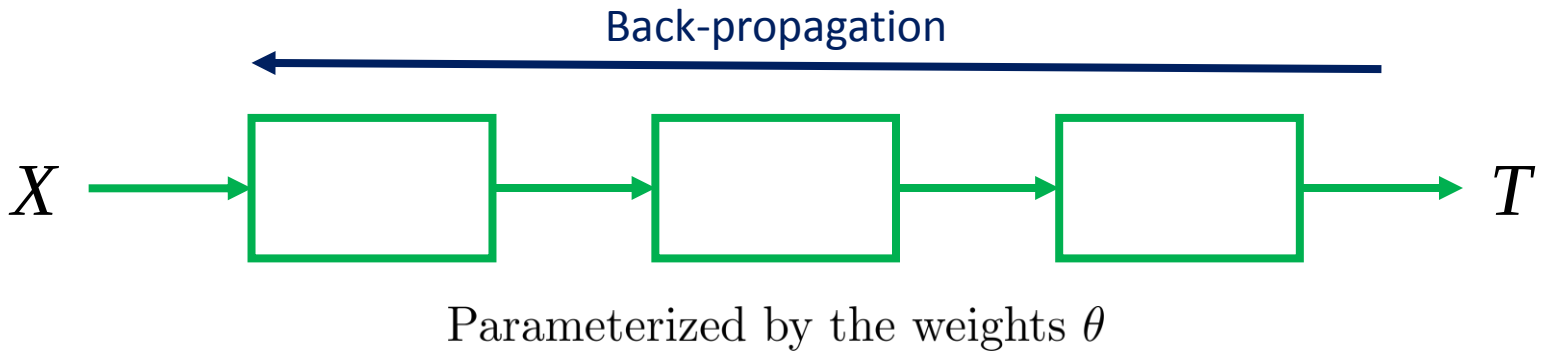
Both models seem to be particular cases of a more general (hopefully Bayesian) model. Work in progress...

Automatic differentiation

- When given a **training sample** (X_{tr}, T_{tr}) one may try to minimize the error on training sample
- For example in case of **probabilistic model** $p(T|X,\theta)$ we compute **maximum likelihood** estimate

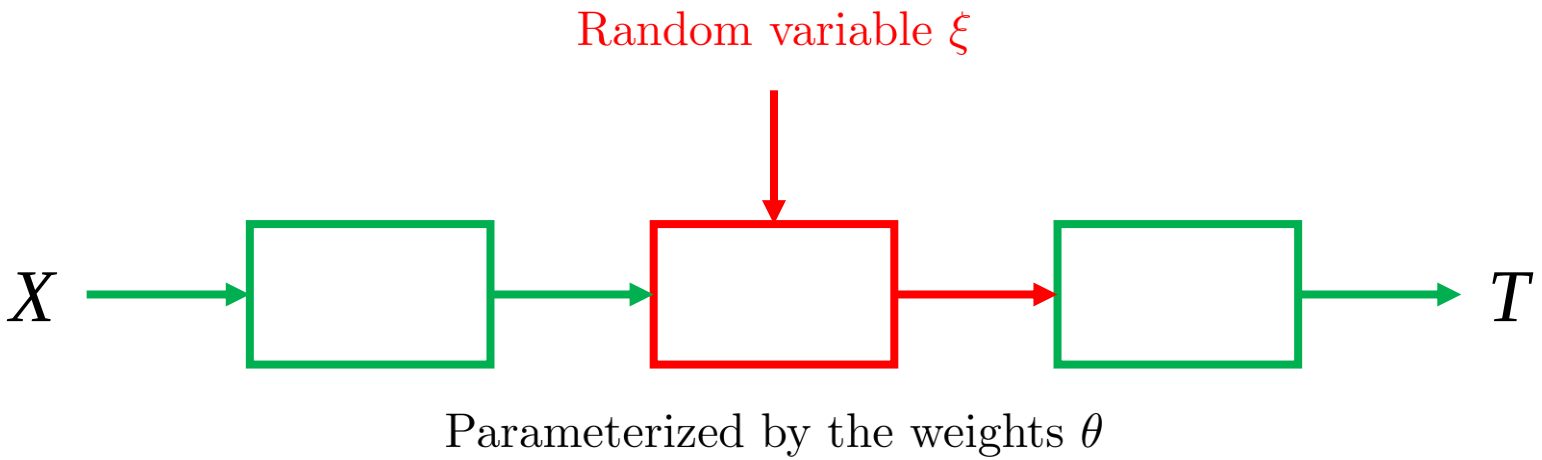
$$\theta_{ML} = \arg\max_{\theta} p(T_{tr}|X_{tr},\theta)$$

- Automatic differentiation allows us to propagate gradients and compute all derivatives w.r.t. θ efficiently via chain rule



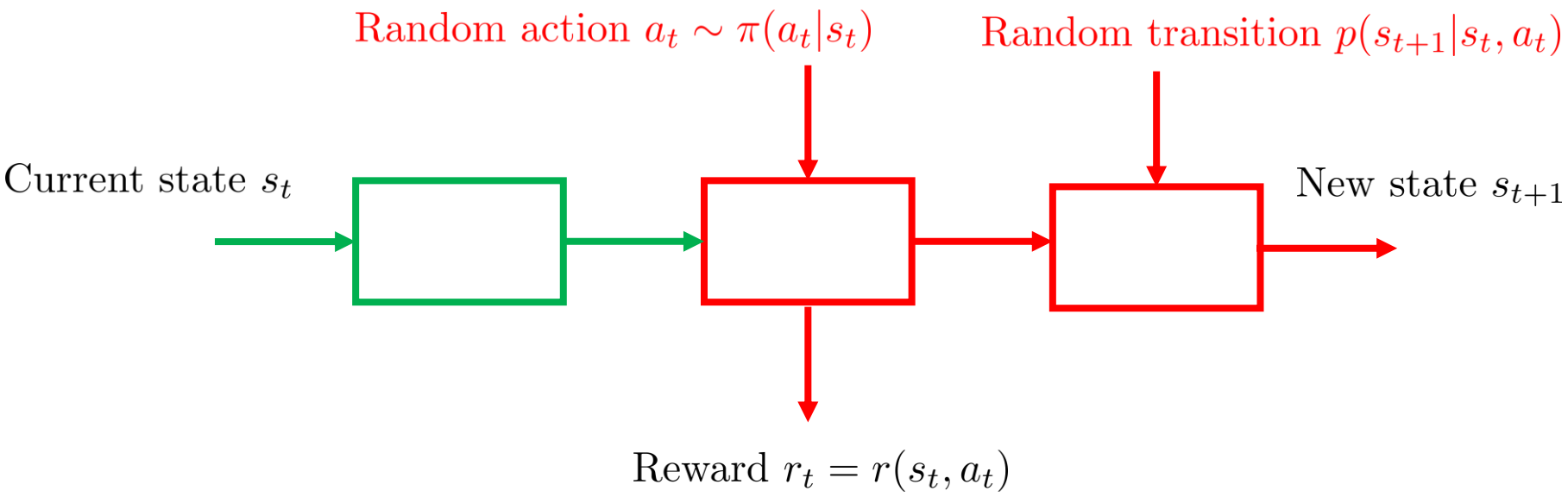
Stochastic computational graphs

- In many popular models we inject random elements in our computational graph
- The ML model becomes more complex but obtains many useful properties
- Switching to stochastic computational graphs is current trend in modern ML



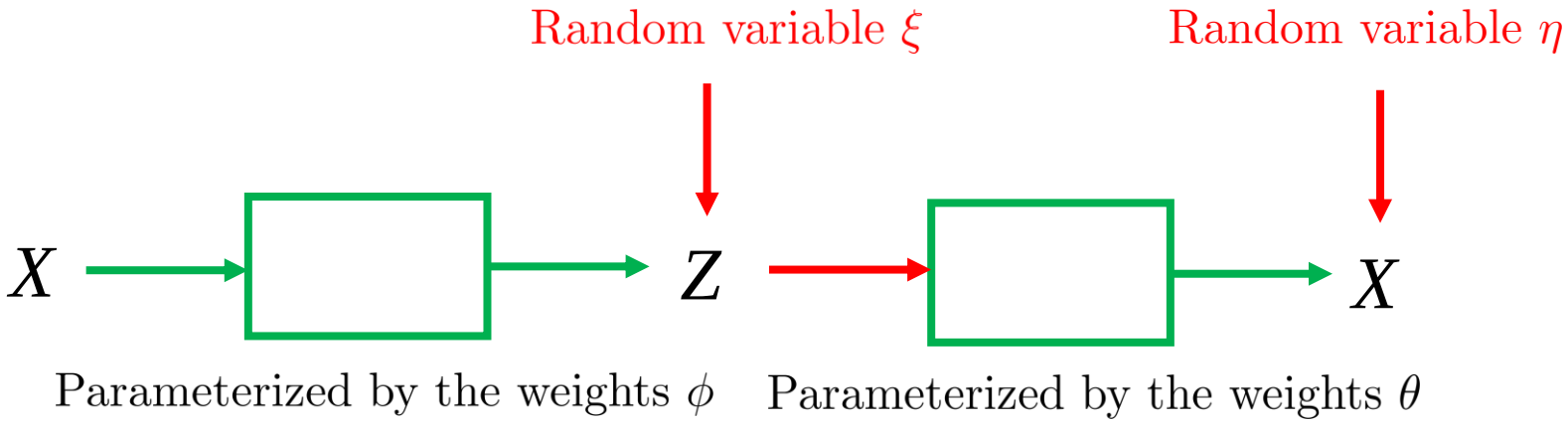
Reinforcement learning

- States s_t are defined as Markov decision process: $s_t \sim p(s_t|s_{t-1}, a_{t-1})$
- Actions are defined by current policy: $a_t \sim \pi(a_t|s_t)$
- Rewards are defined as a function of state and action: $r_t = r(s_t, a_t)$



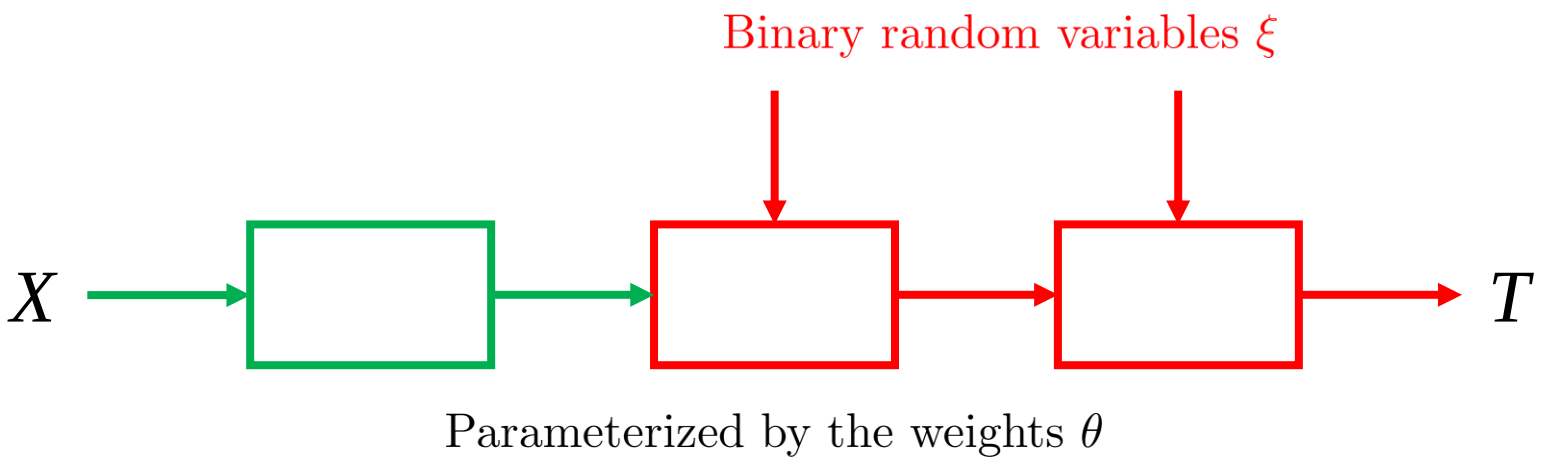
Variational auto-encoders

- New way for dimension reduction (learning representations)
- High-dimensional data X are generated from low-dimensional latent representations Z via deep neural network (decoder) that models $p(X|Z,\theta)$
- Inference is made by another deep network (encoder) that returns a distribution $q(Z|X,\phi)$ in the space of latent variables rather than a point estimate



Dropout

- Dropout is an heuristic regularization mechanism for deep neural networks
- Some neurons are randomly nullified during training
- It corresponds to training an ensemble of networks



Artistic style

- Artistic style allows to transfer style from one image and apply it to another image
- The content is kept fixed
- Modern methods synthesize the image from scratch

Random variables ξ

Content image X_2

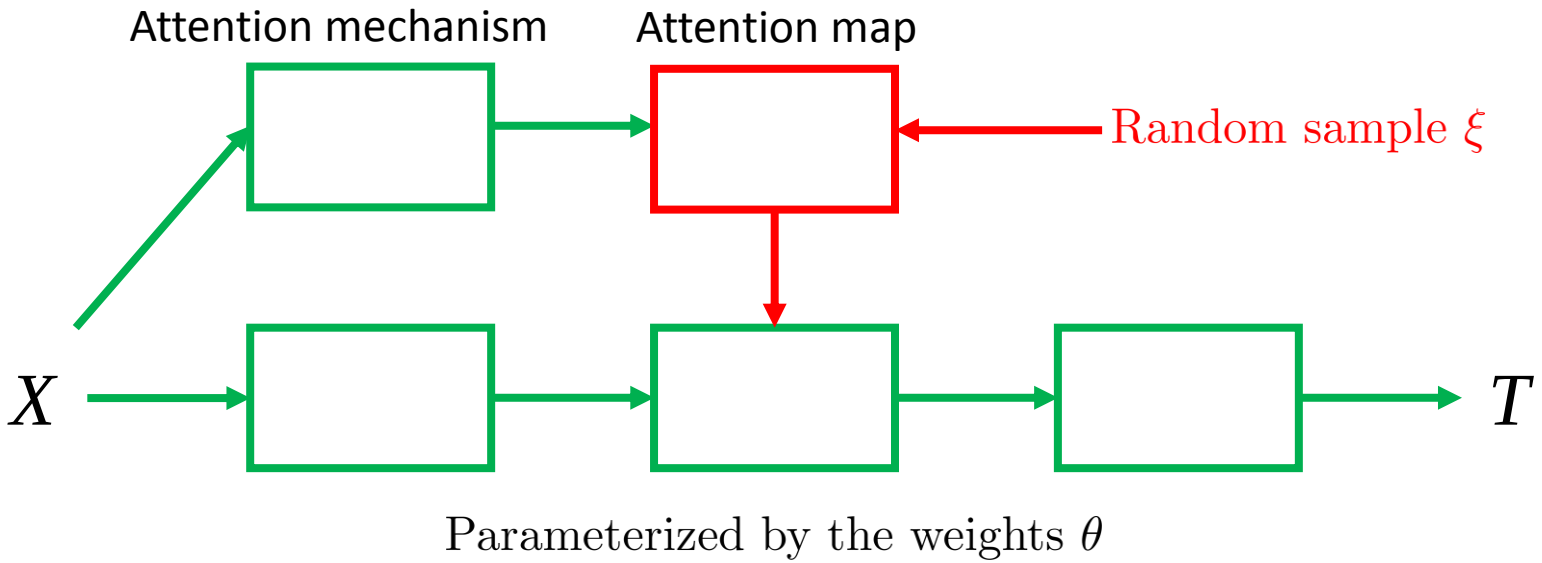
Style image X_1

Synthesized image T

Parameterized by the weights θ

Attention mechanism

- New framework for detecting what computations should be made for a particular data
- Attention mechanism is modeled by an additional neural network that returns a distribution on the space of computations
- Surprisingly trainable scheme



Applications of attention

- Image2caption
- Machine translation
- Adaptive computation time
- Q&A systems

A large white bird standing in a forest.

A woman holding a clock in her hand.

A man wearing a hat and a hat on a skateboard.

A person is standing on a beach with a surfboard.

A woman is sitting at a table with a large pizza.

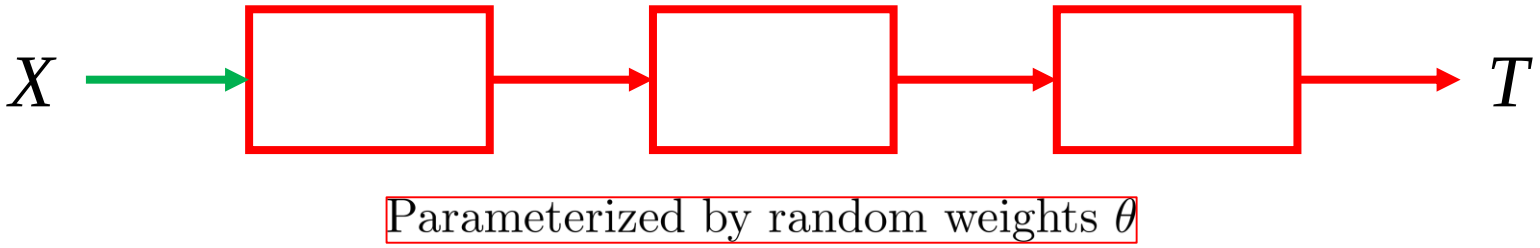
A man is talking on his cell phone while another man watches.

Bayesian regularization

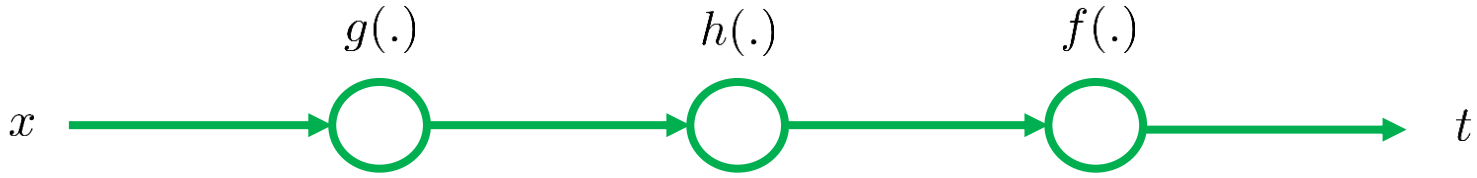
- In Bayesian framework the weights θ are treated as random variables that have a distribution

$$p(T, \theta|X) = p(T|X, \theta)p(\theta)$$

- Prior $p(\theta)$ allows to **regularize** the model thus preventing overfitting on training set
- We need to take into account the randomness of θ during training



Gradient propagation in CG

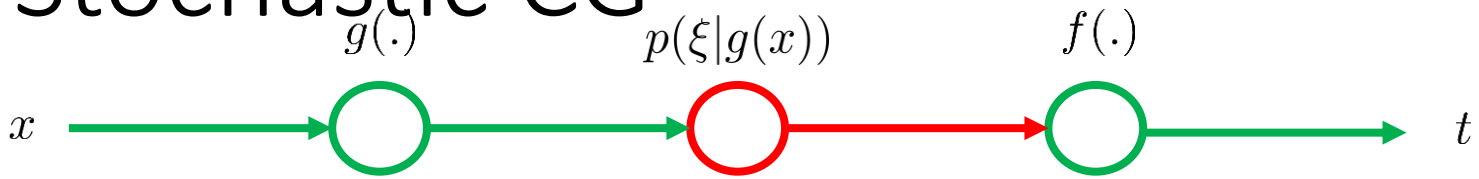


- Any computational graph defines **superposition** of functions to be computed
- Remember how we propagate the gradients in computational graphs $t = f(h(g(x)))$
- Using chain rule we have

$$\frac{\partial t}{\partial x} = \frac{\partial f(h)}{\partial h} \frac{\partial h(g)}{\partial g} \frac{\partial g(x)}{\partial x}$$

- Also known as **back-propagation**

Gradient propagation in Stochastic CG

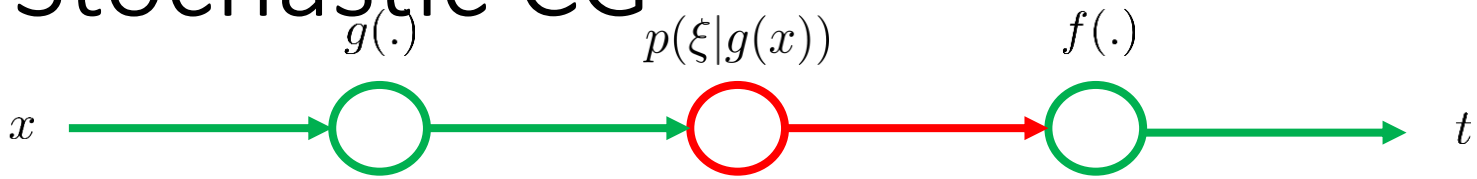


- Now the situation has changed

$$t = \mathbb{E}f(\xi) = \int p(\xi|g(x))f(\xi)d\xi$$

- Using either log-derivative or reparameterization tricks we are still able to propagate the (stochastic) gradients

Gradient propagation in Stochastic CG



- Now the situation has changed

$$t = \mathbb{E}f(\xi) = \int p(\xi|g(x))f(\xi)d\xi$$

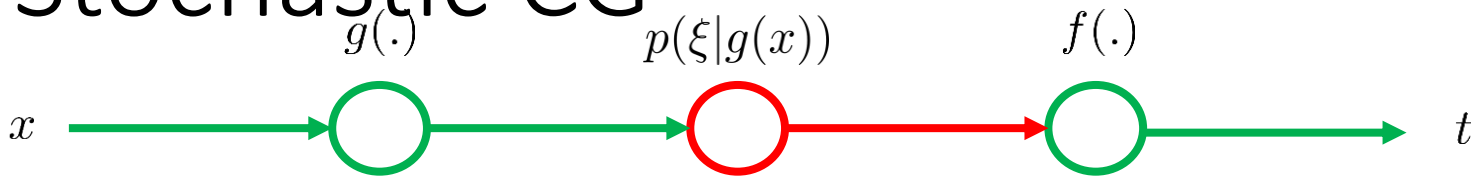
- **Log-derivative trick**

$$\frac{\partial t}{\partial x} = \frac{\partial}{\partial x}\int p(\xi|g(x))f(\xi)d\xi = \int \frac{\partial p(\xi|g(x))}{\partial x} f(\xi)d\xi =$$

$$\int p(\xi|g(x))\frac{\partial \log p(\xi|g(x))}{\partial x} f(\xi)d\xi \approx \frac{\partial \log p(\hat{\xi}|g(x))}{\partial x} f(\hat{\xi}) = \frac{\partial \log p(\hat{\xi}|g(x))}{\partial g}\frac{\partial g(x)}{\partial x} f(\hat{\xi}),$$

where $\hat{\xi} \sim p(\xi|g(x))$

Gradient propagation in Stochastic CG



- Now the situation has changed

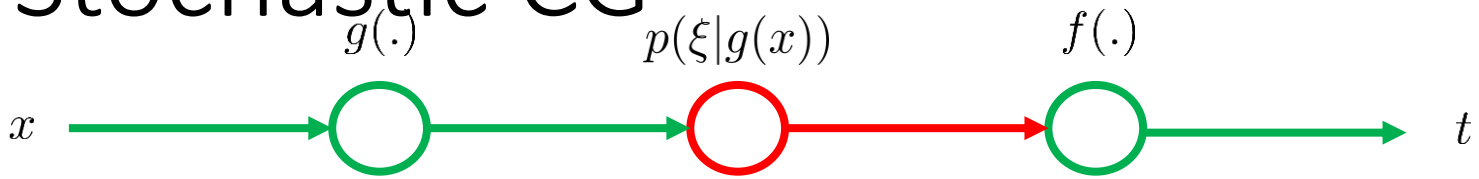
$$t = \mathbb{E}f(\xi) = \int p(\xi|g(x))f(\xi)d\xi$$

- **Reparameterization trick.** Let $\xi = h(\epsilon, g(x))$. Then

$$\frac{\partial t}{\partial x} = \frac{\partial}{\partial x}\int p(\xi|g(x))f(\xi)d\xi = \frac{\partial}{\partial x}\int p(\epsilon)f(h(\epsilon, g(x)))d\epsilon =$$
$$\int p(\epsilon)\frac{\partial f(h(\epsilon, g(x)))}{\partial x}d\epsilon \approx \frac{\partial f(h(\hat{\epsilon}, g(x)))}{\partial x} = \frac{\partial f(h)}{\partial h}\frac{\partial h(g)}{\partial g}\frac{\partial g(x)}{\partial x},$$

where $\hat{\epsilon} \sim p(\epsilon)$

Gradient propagation in Stochastic CG



- Now the situation has changed

$$t = \mathbb{E}f(\xi) = \int p(\xi|g(x))f(\xi)d\xi$$

- **Reparameterization trick.** Let $\xi = h(\epsilon, g(x))$. Then

$$\frac{\partial t}{\partial x} = \frac{\partial}{\partial x}\int p(\xi|g(x))f(\xi)d\xi = \frac{\partial}{\partial x}\int p(\epsilon)f(h(\epsilon, g(x)))d\epsilon =$$
$$\int p(\epsilon)\frac{\partial f(h(\epsilon, g(x)))}{\partial x}d\epsilon \approx \frac{\partial f(h(\hat{\epsilon}, g(x)))}{\partial x} = \boxed{\frac{\partial f(h)}{\partial h}}\frac{\partial h(g)}{\partial g}\frac{\partial g(x)}{\partial x},$$

where $\hat{\epsilon} \sim p(\epsilon)$

- Note that unlike previous slide here we use the information about gradient of $f(.)$

Conclusion

- Variational Bayes is highly scalable inference technique
- Deep neural networks may be used for approximating intractable distributions via stochastic variational inference
- Stochastic computational graphs allow much more flexibility when modeling dependencies within the data
- May transform complex high-dimensional distributions into simple lower-dimensional ones and vice versa
- Most promising mean for generative models
- May process and generate “raw data”(texts, images, videos, graphs, etc.)
- OPEN PROBLEM: How to deal with discrete latent random variables?