M2177.003100
Deep Learning

[14: Variational Autoencoder (VAE)]

Electrical and Computer Engineering
Seoul National University

(last compiled at 09:23:00 on 2020/11/21)

Outline

References

- *Deep Learning* by Goodfellow, Bengio and Courville ▸ Link
  - Chapter 14: Autoencoders
  - Chapter 20: Deep Generative Models
- *Pattern Recognition and Machine Learning* by Bishop
  - Chapter 10: Approximate Inference
- online resources:
  - *Stanford CS231n: CNN for Visual Recognition* ▸ Link
  - *CVPR 2018 GAN Tutorial* ▸ Link
  - *NIPS 2016 Variational Inference Tutorial* ▸ Link

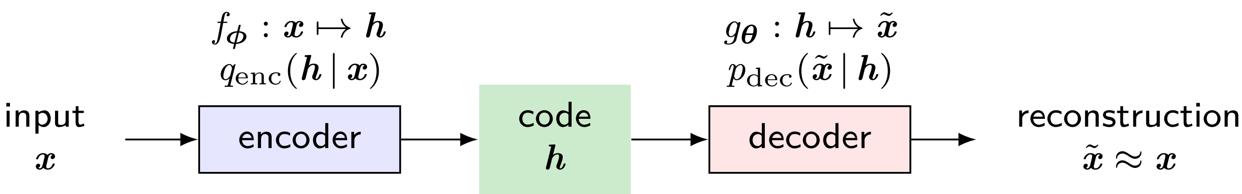
Outline

Autoencoder (AE)

- neural net trained to reconstruct input $\boldsymbol{x}$ as output $\tilde{\boldsymbol{x}}$ (*i.e.* $\boldsymbol{x} \approx \tilde{\boldsymbol{x}}$)
  - learns code (= efficient representation of input data)
  - unsupervised (no label)

- architecture

	deterministic	stochastic	parameterized by
encoder	$f_{\boldsymbol{\phi}}(\boldsymbol{x}) = \boldsymbol{h}$	$q_{\text{enc}}(\boldsymbol{h} \mid \boldsymbol{x})$	$\boldsymbol{\phi}$
decoder	$g_{\boldsymbol{\theta}}(\boldsymbol{h}) = \tilde{\boldsymbol{x}}$	$p_{\text{dec}}(\tilde{\boldsymbol{x}} \mid \boldsymbol{h})$	$\boldsymbol{\theta}$



  - $\dim(\boldsymbol{x}) > \dim(\boldsymbol{h})$: undercomplete $\Rightarrow$ capture important features of input
  - $\dim(\boldsymbol{x}) < \dim(\boldsymbol{h})$: overcomplete

- training

$$\boldsymbol{\phi}^*, \boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\phi}, \boldsymbol{\theta}} L(\boldsymbol{x}, \tilde{\boldsymbol{x}})$$

  - L: loss (*e.g.* mean squared error)

- applications
  - dimensionality reduction
  - feature learning
  - forefront of generative modeling

- linear f, g and MSE loss
  - $\boldsymbol{h}$ spans the same subspace as principal component analysis (PCA)

- stacked AE (= deep AE)
  - AE with multiple hidden layers

Need for regularization

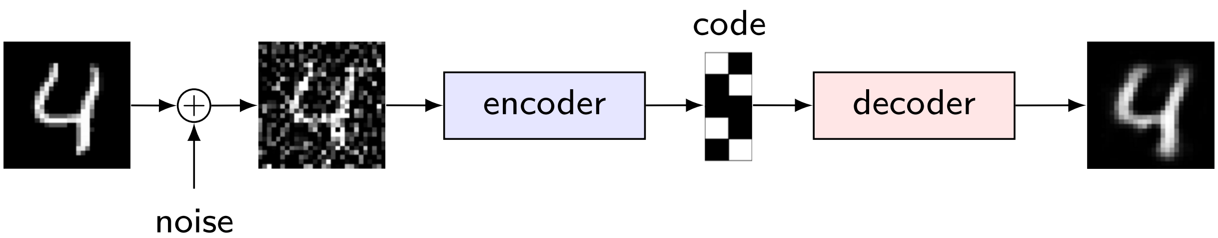
- encoder/decoder with too much capacity
  - simply learn to copy input to output
  - learn nothing useful about data distribution

- regularizing AE
  - limit model capacity
  - use a loss encouraging desirable properties

- these properties
  - robustness to noise/missing inputs
  - sparse representation
  - small derivative of representation

Denoising autoencoder (DAE)

- one way to force AE to learn useful features
  - add noise to inputs
  - then train AE to recover original (noise-free) input



- types of noise
  - Gaussian noise (shown above)
  - dropout: randomly switched off inputs

image source: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Sparse autoencoder

- training criterion involves
  1. reconstruction error
  2. sparsity penalty $\Omega(\boldsymbol{h})$ on code $\boldsymbol{h}$

$$J(\boldsymbol{\phi}, \boldsymbol{\theta}) = \underbrace{L(\boldsymbol{x}, \tilde{\boldsymbol{x}})}_{\text{reconstruction loss}} + \underbrace{\Omega(\boldsymbol{h})}_{\text{sparsity loss}}$$

  - sparsity loss example ▸ Link
    - $\Omega(\boldsymbol{h}) = \lambda D_{\mathrm{KL}}[p(\boldsymbol{h}; \boldsymbol{\rho}) \,||\, p(\boldsymbol{h}; \hat{\boldsymbol{\rho}})]$ ← λ: regularization coefficient
    - $p(\boldsymbol{h}; \boldsymbol{\rho})$: desired (sparse) distribution of $\boldsymbol{h}$ ← parameterized by $\boldsymbol{\rho}$
    - $p(\boldsymbol{h}; \hat{\boldsymbol{\rho}})$: observed distribution of $\boldsymbol{h}$ ← parameterized by $\hat{\boldsymbol{\rho}}$

- typically used to learn features for another task

  e.g. classification

Contractive autoencoder (CAE)

- introduces explicit regularizer on code $\boldsymbol{h} = f(\boldsymbol{x})$
  - encourage derivative of f to be as small as possible

$$\Omega(\boldsymbol{h}) = \lambda \left|\left| \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right|\right|_F^2$$

  - $||\cdot||_F^2$: squared Frobenius norm ($=$ sum of squared elements)

- effect[1]
  - better captures $\underbrace{\text{local directions of variation}}_{\uparrow}$ dictated by data

$= $ a lower-dimensional non-linear manifold

  - f "contracts" input neighborhood to a smaller output neighborhood

i.e. similar inputs $\mapsto$ similar codings

[1] Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 833–840. Omnipress

More autoencoder examples

- stacked convolutional autoencoders[2]
  - learn to extract visual features through conv layers
- generative stochastic network (GSN)[3]
  - generalization of DAE (added capability to generate data)
- winner-take-all (WTA) autoencoder[4]
  - only top $k\%$ activation for each neuron are preserved (the rest: set to 0)

  $\Rightarrow$ leads to sparse coding
- adversarial autoencoders[5]
  - AE1: trained to reproduce its input
  - AE2: trained to find inputs AE1 cannot properly reconstruct

$\Rightarrow$ AE2 pushes AE1 to learn robust code

---

[2] Masci, J., Meier, U., Cireșan, D., and Schmidhuber, J. (2011). Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pages 52–59. Springer

[3] Bengio, Y., Laufer, E., Alain, G., and Yosinski, J. (2014). Deep generative stochastic networks trainable by backprop. In *International Conference on Machine Learning*, pages 226–234

[4] Makhzani, A. and Frey, B. J. (2015). Winner-take-all autoencoders. In *Advances in Neural Information Processing Systems*, pages 2791–2799

[5] Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., and Frey, B. (2015). Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*

Outline

Variational autoencoder (VAE)

- a popular approach to unsupervised learning of complex distributions
  - uses learned approximate inference
  - can be trained purely with gradient-based methods[6,7]
- appealing because
  - built on top of standard function approximators (neural nets)
  - can be trained with SGD

[6] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*

[7] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*

Comparison

- augoregressive models
  - define a tractable density function

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(x_i \mid x_1, \ldots, x_{i-1})$$

- VAE
  - define an intractable density function with latent variable $\mathbf{z}$:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) d\boldsymbol{z}$$

- GAN
  - give up explicit $p(\boldsymbol{x})$ modeling
  - just want to ability to sample from it

Related models

- autoencoders: reconstruct given input $\boldsymbol{x}$

$$\boldsymbol{x} \xrightarrow[\text{encoder}]{p(\boldsymbol{z}\,|\,\boldsymbol{x})} \boldsymbol{z} \xrightarrow[\text{decoder}]{p(\boldsymbol{x}\,|\,\boldsymbol{z})} \boldsymbol{x} \qquad \text{(training)}$$

$$\boldsymbol{x} \xrightarrow[\text{encoder}]{p(\boldsymbol{z}\,|\,\boldsymbol{x})} \boldsymbol{z} \qquad \text{(main use: feature learner for downstream learning)}$$

- generative models: generate new sample $\boldsymbol{x}$

$$\xrightarrow{\qquad\quad p(\boldsymbol{x}) \qquad\quad} \boldsymbol{x}$$

  - one challenge: modeling dependencies between dimensions

- latent generative models: generate new sample $\boldsymbol{x}$ from latent $\boldsymbol{z}$

$$\xrightarrow[\text{prior}]{p(\boldsymbol{z})} \boldsymbol{z} \xrightarrow{p(\boldsymbol{x}\,|\,\boldsymbol{z})} \boldsymbol{x}$$

  - introducing $\boldsymbol{z}$ improves dependency modeling
  - how to train? maximize data likelihood $p(\boldsymbol{x}) = \int p(\boldsymbol{z})p(\boldsymbol{x}\,|\,\boldsymbol{z})d\boldsymbol{z}$
  - challenge: integration $\int d\boldsymbol{z}$ is often intractable

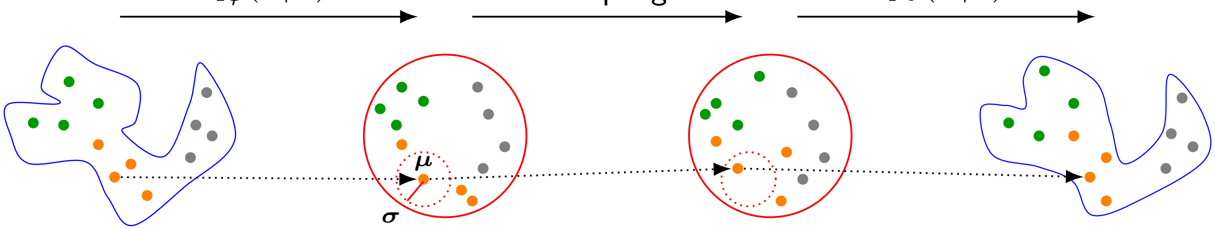
Idea behind VAE

- training

encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$

sampling

decoder $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$



$\mathbf{x}$ (input space) $\mathbf{z}$ (code space) $\tilde{\mathbf{x}}$ (input space)

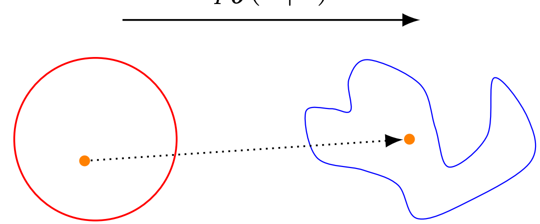
minimize
$D_{\mathrm{KL}}[q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \,||\, \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \mathbf{I})]$

encoder prior

maximize
$\mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\phi}}(z|\boldsymbol{x})} [\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})]$

- sample generation
  - $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$: easy to sample from

decoder
$p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$



$\mathbf{z}$ (code space) $\tilde{\mathbf{x}}$ (input space)

Quick look

- goal: find $\boldsymbol{\theta}$ that maximizes likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ for training data $\boldsymbol{x}$

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int \underbrace{p_{\boldsymbol{\theta}}(\boldsymbol{z})}_{\substack{\text{simple} \\ \text{Gaussian} \\ \text{prior}}} \underbrace{p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})}_{\substack{\text{learn with} \\ \text{neural net} \\ \text{“}\textbf{decoder}\text{”}}} d\boldsymbol{z} \tag{1}$$

- challenge: cannot optimize (1) directly
  - $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$ are okay but the integration is intractable

- solution: derive and optimize a tractable lower bound on the likelihood

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq \underbrace{\mathcal{L}(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{\phi})}_{\text{ELBO}}$$

  - ELBO: depends on $\underbrace{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}_{\substack{\text{learn with} \\ \text{neural net} \\ \text{“}\textbf{encoder}\text{”}}} \quad \underbrace{\approx}_{\substack{\text{approximate} \\ \text{using} \\ \text{variational inference}}} \quad \underbrace{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}_{\substack{\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})} \\ \nearrow \text{intractable}}}$
  - what to learn: **model** parameters $\boldsymbol{\theta}$ and **variational** parameters $\boldsymbol{\phi}$

decoder encoder

GNU and VAE

- GNU
  - GNU's Not Unix! (recursive acronym)
  - an operating system and an extensive collection of computer software
- VAE
  - VAE's not AutoEncoder!
  - a generative model to learn complex distributions

image source: `https://en.wikipedia.org/wiki/GNU#/media/File:Heckert_GNU_white.svg`

Why are VAEs called "autoencoders"?

- mathematical basis of VAE[8]
  - has relatively little to do with classical AEs (*e.g.* sparse AE, denoising AE)
- they are called "AEs" only because
  - training objective has encoder/decoder ($\Rightarrow$ *resembles* traditional AE)
- differences
  - unlike sparse/denoising AEs
    - VAE allows us to sample directly from $p(\boldsymbol{x})$ (without doing MCMC)
  - typical AE: decoders are sometimes removed after training
    - $p(\boldsymbol{z} \mid \boldsymbol{x})$ learned by encoder: used for *e.g.* supervised learning
  - VAE: encoders are removed after training
    - $p(\boldsymbol{x} \mid \boldsymbol{z})$ learned by decoder: used for data generation

[8] Doersch, C. (2016). Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*

Comparison

- autoencoders

$$x \xrightarrow[\text{encoder}]{p(z \mid x)} z \xrightarrow[\text{decoder}]{p(x \mid z)} x \qquad \text{(training)}$$

$$x \xrightarrow[\text{encoder}]{p(z \mid x)} z \qquad \text{(main use: feature learner for downstream learning)}$$

- generative models

$$\xrightarrow{\qquad p(x) \qquad} x$$

- latent generative models

$$\xrightarrow[\text{prior}]{p(z)} z \xrightarrow{p(x \mid z)} x$$

- VAEs

$$x \xrightarrow[\text{encoder}]{q(z \mid x)} z \xrightarrow[\text{decoder}]{p(x \mid z)} x \qquad \text{(trained as an AE)}$$

$$\xrightarrow[\text{prior}]{p(z)} z \xrightarrow[\text{decoder}]{p(x \mid z)} x \qquad \text{(main use: data generation)}$$

Outline

Problem formulation

- objective: for each x in training set
  - maximize the probability of x under the entire generative process:

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x} \mid \boldsymbol{z}) p(\boldsymbol{z}) d\boldsymbol{z} \tag{2}$$

- to solve (2), VAEs must deal with three problems

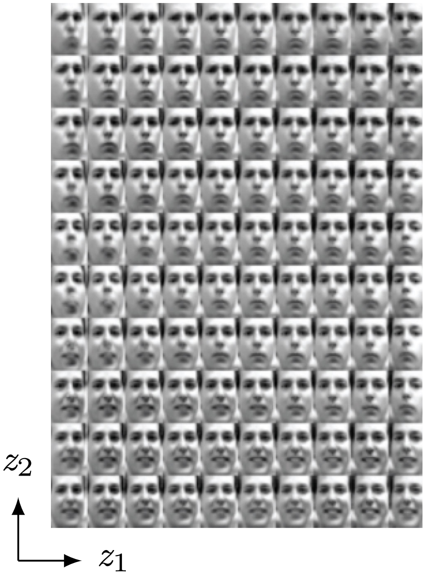
P1 how to define latent variables $\mathbf{z}$
(*i.e.* decide what information they represent)

P2 how to define output distribution $p(\boldsymbol{x} \mid \boldsymbol{z})$

P3 how to deal with integral over $\mathbf{z}$

Solution 1: prior $p(\boldsymbol{z})$

- choose prior $p(\boldsymbol{z})$ to be simple
  - reasonable for latent attributes (*e.g.* pose, how much smile)
- common choice: unit Gaussian $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \mathbf{I})$
  - diagonal prior $\Rightarrow$ independent latent variables
  - different dimensions of $\boldsymbol{z}$ encode interpretable factors of variation



- varying z_1
  - varying *head* pose
- varying z_2
  - varying degree of *smile*

image source: Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*

Solution 2: output distribution $p(\boldsymbol{x} \mid \boldsymbol{z})$

- use a differentiable generator ("decoder") network $g(\boldsymbol{z}; \boldsymbol{\theta})$
  - $\boldsymbol{\theta}$ denotes parameters of decoder neural net
  - $g(\boldsymbol{z}; \boldsymbol{\theta})$ produces parameters of $p(\boldsymbol{x} \mid \boldsymbol{z})$

$$p(\boldsymbol{x} \mid \boldsymbol{z}) = p(\boldsymbol{x}; g(\boldsymbol{z}; \boldsymbol{\theta})) \triangleq p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$$

- common parametric distributions for $p(\boldsymbol{x} \mid \boldsymbol{z})$
  - continuous $\mathbf{x}$: a Gaussian parameterized by $g(\boldsymbol{z}; \boldsymbol{\theta})$
  - binary $\mathbf{x}$: a Bernoulli parameterized by $g(\boldsymbol{z}; \boldsymbol{\theta})$

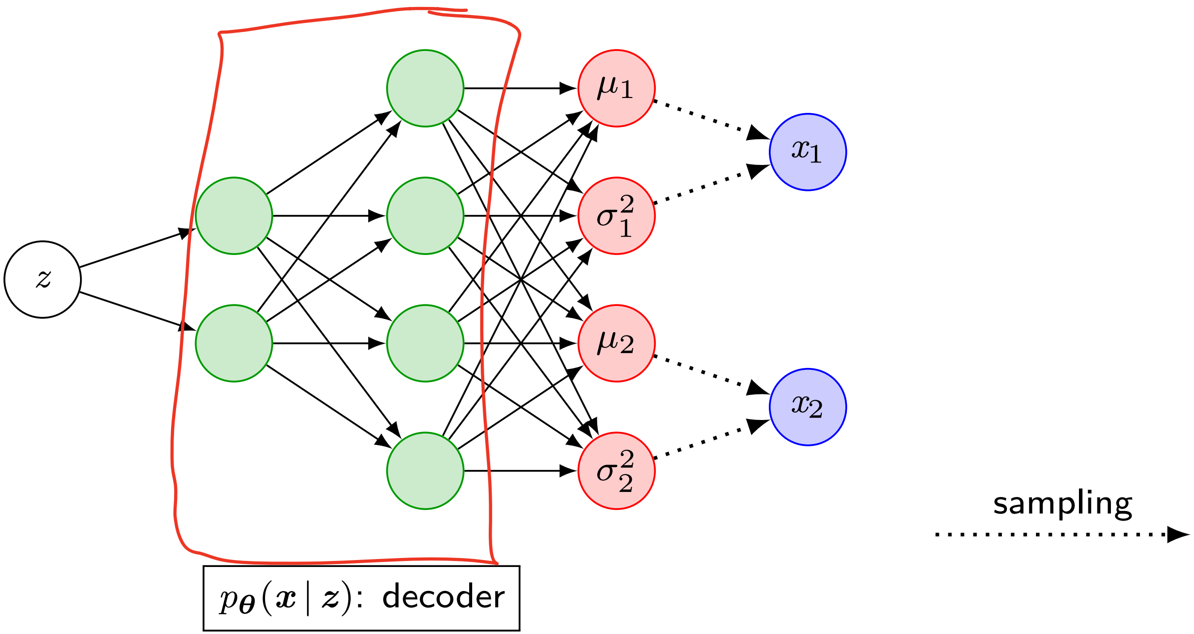
- at training time
  - likelihood of each input $\boldsymbol{x}^{(i)}$ can be computed using $p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)} \mid \boldsymbol{z})$

  $\Rightarrow$ this generates error signal for doing SGD by backprop

- after training
  - a new $\boldsymbol{x}$ is sampled from $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$

- *e.g.* Gaussian with diagonal covariance: $p(\boldsymbol{x} \mid z) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}(z), \boldsymbol{\Sigma}(z))$
  - $\boldsymbol{x} \in \mathbb{R}^2$, $z \in \mathbb{R}$

$$\boldsymbol{\mu}(z) = \begin{bmatrix} \mu_1(z) \\ \mu_2(z) \end{bmatrix}, \quad \boldsymbol{\Sigma}(z) = \begin{bmatrix} \sigma_1^2(z) & 0 \\ 0 & \sigma_2^2(z) \end{bmatrix}$$



Solution 3: integration over $\boldsymbol{z}$

- from Solutions 1 & 2:

$$p(\boldsymbol{x}) = \int \underbrace{p(\boldsymbol{z})}_{\substack{\text{simple} \\ \text{Gaussian} \\ \text{prior}}} \underbrace{p(\boldsymbol{x} \mid \boldsymbol{z})}_{\substack{\text{learn with} \\ \text{neural net} \\ \text{``\textbf{decoder}''}}} \, d\boldsymbol{z}$$

  - $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$ are okay but integration remains intractable

- conceptually: easy to compute $p(\boldsymbol{x})$ approximately
  - sample a large number of $\boldsymbol{z}$ values $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}$ and compute

$$p(\boldsymbol{x}) \approx \frac{1}{n} \sum_{i=1}^{n} p(\boldsymbol{x} \mid \boldsymbol{z}_i)$$

- challenge: in high-dimensional space
  - n : extremely large before having an accurate estimate of $p(\boldsymbol{x})$

- in practice: $p(\boldsymbol{x} \mid \boldsymbol{z}) \approx 0$ for most $\boldsymbol{z} \rightarrow$ contribute nothing to estimate of $p(\boldsymbol{x})$
- key idea behind VAE #1:
  - try to sample values of $\boldsymbol{z}$ that are likely to have produced $\boldsymbol{x}$, and
  - compute $p(\boldsymbol{x})$ just from those
- this means: we need a new function $q(\boldsymbol{z} \mid \boldsymbol{x})$, which can
  - take a value of $\boldsymbol{x}$, and
  - give us $\boxed{\text{a distribution over } \boldsymbol{z} \text{ that are likely to produce } \boldsymbol{x}}$
- why not exact posterior $p(\boldsymbol{z} \mid \boldsymbol{x})$?
  - also intractable: $p(\boldsymbol{z} \mid \boldsymbol{x}) = p(\boldsymbol{x} \mid \boldsymbol{z})p(\boldsymbol{z})/p(\boldsymbol{x})$
- key idea #2: resort to variational inference, which allows us to
  1. find an approximate posterior $q(\boldsymbol{z} \mid \boldsymbol{x}) \approx p(\boldsymbol{z} \mid \boldsymbol{x})$
  2. $\underbrace{\text{maximize } \mathrm{ELBO}(q)}_{\text{tractable}}$ rather than $\underbrace{\text{maximizing } p(\boldsymbol{x})}_{\text{intractable}}$ directly
  - issue: $\boxed{\text{traditional VI using optimization is slow/requires closed form}}$

- key idea #3: use a $\boxed{\text{differentiable inference ("encoder") network } f(\boldsymbol{x};\boldsymbol{\phi})}$
  - $\boldsymbol{\phi}$ denotes parameters of encoder neural net
  - $f(\boldsymbol{x};\boldsymbol{\phi})$ produces parameters of $q(\boldsymbol{z}\,|\,\boldsymbol{x})$

$$q(\boldsymbol{z}\,|\,\boldsymbol{x}) = q(\boldsymbol{z}; f(\boldsymbol{x};\boldsymbol{\phi})) \triangleq q_{\boldsymbol{\phi}}(\boldsymbol{z}\,|\,\boldsymbol{x})$$

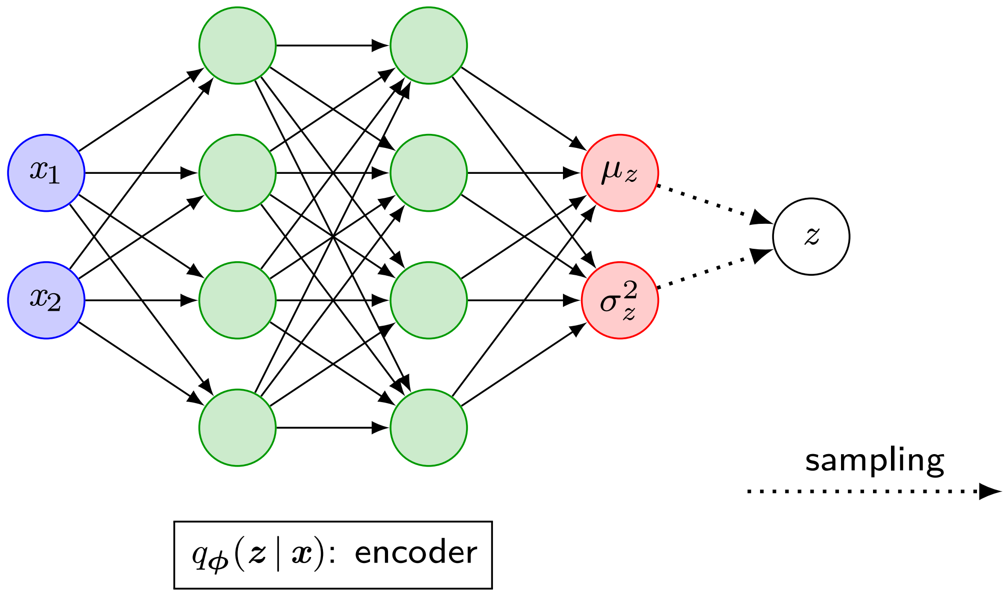
  - training the encoder: often easier than optimization in traditional VI
    - backed by various efficient techniques (*e.g.* SGD, backprop)

- common parametric distributions for $q(\boldsymbol{z}\,|\,\boldsymbol{x})$
  - a Gaussian parameterized by $f(\boldsymbol{x};\boldsymbol{\phi})$
  - its covariance $\boldsymbol{\Sigma}$: constrained to be diagonal (for computational issues)

- $\mathbf{z}$ is sampled from
  - $q_{\boldsymbol{\phi}}(\boldsymbol{z}\,|\,\boldsymbol{x})$ in training
  - $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ after training

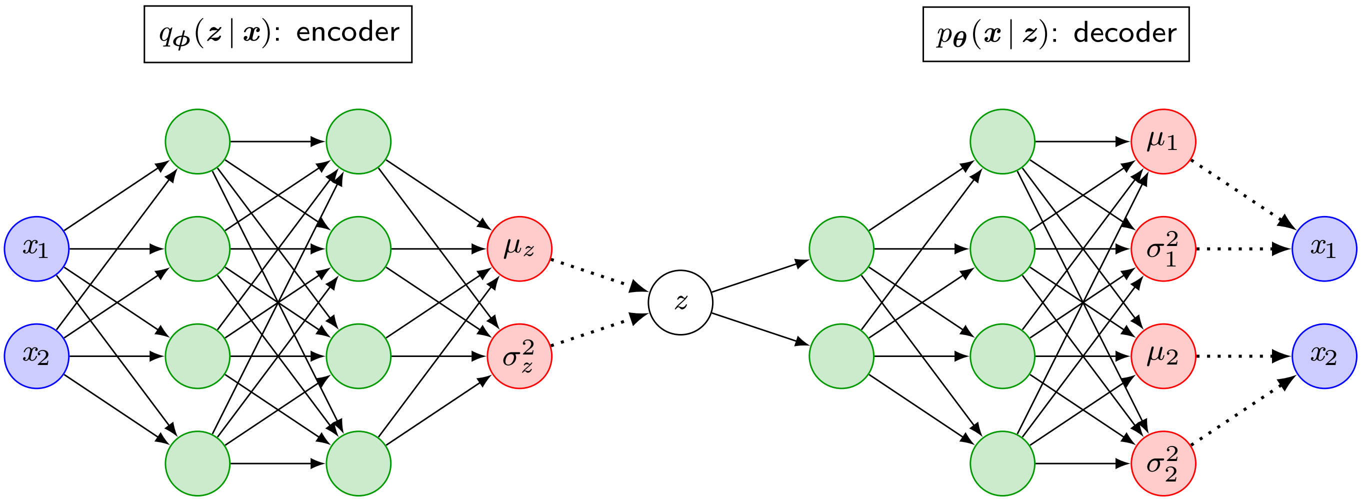
$\Rightarrow$ and then the sampled $\mathbf{z}$ is fed into decoder to generate $\mathbf{x}$

- *e.g.* Gaussian $q(z \mid \boldsymbol{x}) = \mathcal{N}(z; \mu(\boldsymbol{x}), \sigma^2(\boldsymbol{x}))$
  - $\boldsymbol{x} \in \mathbb{R}^2$, $z \in \mathbb{R}$



VAE architecture

- encoder and decoder are stacked



- we can learn parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ via ________

Outline

Training objective

- as in VI, decomposing $\log p(\boldsymbol{x})$ reveals it:

$$
\begin{aligned}
\log p(\boldsymbol{x}) &= \mathbb{E}_{\mathbf{z}\sim q(z|\boldsymbol{x})}[\log p(\boldsymbol{x})] \\
&= \mathbb{E}_{\mathbf{z}}\left[\log \frac{p(\boldsymbol{x}\mid \boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{z}\mid \boldsymbol{x})}\right] \\
&= \mathbb{E}_{\mathbf{z}}\left[\log \frac{p(\boldsymbol{x}\mid \boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{z}\mid \boldsymbol{x})}\frac{q(\boldsymbol{z}\mid \boldsymbol{x})}{q(\boldsymbol{z}\mid \boldsymbol{x})}\right] \\
&= \mathbb{E}_{\mathbf{z}}\left[\log p(\boldsymbol{x}\mid \boldsymbol{z})\right] - \mathbb{E}_{\mathbf{z}}\left[\log \frac{q(\boldsymbol{z}\mid \boldsymbol{x})}{p(\boldsymbol{z})}\right] + \mathbb{E}_{\mathbf{z}}\left[\log \frac{q(\boldsymbol{z}\mid \boldsymbol{x})}{p(\boldsymbol{z}\mid \boldsymbol{x})}\right] \\
&= \underbrace{\mathbb{E}_{\mathbf{z}}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}\mid \boldsymbol{z})\right] - D_{\mathrm{KL}}[q_{\boldsymbol{\phi}}(\boldsymbol{z}\mid \boldsymbol{x}) \,||\, p(\boldsymbol{z})]}_{\text{ELBO: } \mathcal{L}(\boldsymbol{x},\boldsymbol{\theta},\boldsymbol{\phi})} + \underbrace{D_{\mathrm{KL}}[q_{\boldsymbol{\phi}}(\boldsymbol{z}\mid \boldsymbol{x}) \,||\, p(\boldsymbol{z}\mid \boldsymbol{x})]}_{\geq 0 \text{ (intractable)}}
\end{aligned}
$$

- note: *any* distribution over $\mathbf{z}$ can be proxy q for posterior $p(\boldsymbol{z}\mid \boldsymbol{x})$
  - EM/VI: using $q(\boldsymbol{z})$ is more common
  - VAE: using $q(z|x)$ is more common[9]

[9] since we want to infer $p(\boldsymbol{x})$, it makes sense to construct q that does depend on $\boldsymbol{x}$ [see Doersch, C. (2016). Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*]

Interpretation[10]

- VAE: trained by maximizing variational lower bound $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{\phi})$

$$
\begin{aligned}
\log p(\boldsymbol{x}) &\geq \mathcal{L}(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{\phi}) \\
&= \underbrace{\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(z|x)}\left[\log p(\boldsymbol{z}, \boldsymbol{x})\right]}_{①} + \underbrace{H[q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})\;]}_{②\ \text{entropy}} \\
&= \underbrace{\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(z|x)}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})\right]}_{③\ \text{reconstruction}} - \underbrace{D_{\mathrm{KL}}[q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \,||\, p(\boldsymbol{z})\;]}_{④\ \text{regularizer}}
\end{aligned}
$$

① joint log-likelihood of visible/hidden variables

- under approximate posterior over latent variables

② entropy of approximate posterior: maximizing this encourages to

- place high prob mass on many z values that could have generated x
- rather than collapsing to a single point estimate of the most likely value

③ reconstruction log-likelihood found in other autoencoders

④ tries to make approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ and model prior $p(\boldsymbol{z})$ close

- tries to reflect prior knowledge $\Rightarrow$ $\boxed{\text{regularizer}}$

[10] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT Press

Interpretation[11]

- rearranging decomposition of $\log p(\boldsymbol{x})$ gives

$$\underbrace{\log p(\boldsymbol{x})}_{①} - \underbrace{D_{\mathrm{KL}}[q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \,||\, p(\boldsymbol{z} \mid \boldsymbol{x})]}_{②}$$

$$= \underbrace{\mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})]}_{③ \text{ decoding objective}} - \underbrace{D_{\mathrm{KL}}[q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \,||\, p(\boldsymbol{z})]}_{④ \text{ encoding objective}}$$

- **left hand side**
  - ① quantity we want to maximize
  - ② error term (this will become small if q is high-capacity)
    - ▷ makes q produce $\boldsymbol{z}$'s that can reproduce a given $\boldsymbol{x}$

- **right hand side**: something we can optimize via SGD (given right choice of q)
  - suddenly takes a form which looks like an autoencoder
    - ▷ q is "encoding" $\boldsymbol{x}$ into $\boldsymbol{z}$ (quality measured by ④)
    - ▷ p is "decoding" it to reconstruct $\boldsymbol{x}$ (quality measured by ③)

[11] Doersch, C. (2016). Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*

Training

- perform SGD to maximize the objective:

$$\begin{aligned} J &= \mathcal{L}(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{\phi}) \\ &= \underbrace{-D_{\mathrm{KL}}[q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \,||\, p(\boldsymbol{z})]}_{\text{① encoding objective}} + \underbrace{\mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})]}_{\text{② decoding objective}} \end{aligned}$$

- usual choice: $q(\boldsymbol{z} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\Sigma}(\boldsymbol{x}))$
  - $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$: learned/represented via neural nets
  - $\boldsymbol{\Sigma}$: a diagonal matrix

① D_{KL} between two multivariate Gaussians

- can be computed in closed form

② a bit more tricky

- involves random Sampling $\Rightarrow$ **how to backprop** gradient for SGD?

Training: encoding objective

- optimizing ①: encoding objective $-D_{\mathrm{KL}}[q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \,||\, p(\boldsymbol{z})]$
  - in general (K: dimensionality of variable)

$$D_{\mathrm{KL}}[\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \,||\, \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)] = \frac{1}{2}\left(\mathrm{tr}\left(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Sigma}_0\right) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}_1^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) - K + \ln\left(\frac{\det \boldsymbol{\Sigma}_1}{\det \boldsymbol{\Sigma}_0}\right)\right)$$

  - in VAE this simplifies[12] to

$$\begin{aligned} D_{\mathrm{KL}}[\mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\Sigma}(\boldsymbol{x})) \,||\, \mathcal{N}(\mathbf{0}, \mathbf{I})] &= \frac{1}{2}\left(\mathrm{tr}\left(\boldsymbol{\Sigma}(\boldsymbol{x})\right) + \boldsymbol{\mu}(\boldsymbol{x})^\top \boldsymbol{\mu}(\boldsymbol{x}) - K - \ln\det \boldsymbol{\Sigma}(\boldsymbol{x})\right) \\ &= \frac{1}{2}\left(\sum_{k=1}^{K} \sigma_k^2(\boldsymbol{x}) + \sum_{k=1}^{K} \mu_k^2(\boldsymbol{x}) - \sum_{k=1}^{K} 1 - \ln \prod_{k=1}^{K} \sigma_k^2(\boldsymbol{x})\right) \\ &= \boxed{\frac{1}{2}\sum_{k=1}^{K}\left(\sigma_k^2(\boldsymbol{x};\boldsymbol{\phi}) + \mu_k^2(\boldsymbol{x};\boldsymbol{\phi}) - 1 - \ln \sigma_k^2(\boldsymbol{x};\boldsymbol{\phi})\right)} \end{aligned}$$

  - differentiable $\Rightarrow$ can be minimized wrt $\boldsymbol{\phi}$ via SGD[13]

[12] for diagonal $\boldsymbol{\Sigma} = \mathrm{diag}[\sigma_1^2, \sigma_2^2, \ldots]$, $\mathrm{tr}(\boldsymbol{\Sigma}) = \sum_k \sigma_k^2$ and $\det \boldsymbol{\Sigma} = \prod_k \sigma_k^2$

[13] to avoid clutter, revealed dependence on $\boldsymbol{\phi}$ only in the last equation

Training: decoding objective

- forward prop to compute ②: decoding objective $\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})]$
  - use Monte Carlo sampling: for each input $\boldsymbol{x}^{(i)}$
    1. $\boldsymbol{z}^{(i,l)}$ is sampled from encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}^{(i)})$
    2. $\boldsymbol{z}^{(i,l)}$ is fed to decoder $\Rightarrow$ gives $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)} \mid \boldsymbol{z}^{(i,l)})$
    3. repeat above steps L times to approximate

$$\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)} \mid \boldsymbol{z}^{(i,l)})\right] \approx \frac{1}{L}\sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)} \mid \boldsymbol{z}^{(i,l)})$$

  - in practice, often $L = 1$ (as in SGD; the same motivation)

- however, to maximize ② via SGD with backprop[14]
  - we need a trick called reparameterization
  - without it, we cannot backprop gradient through the sampling process
    - sampling: not continuous $\Rightarrow$ not differntiable

[14] SGD via backprop can handle stochastic inputs but not stochastic units within the network

Reparameterization trick

- idea: move sampling to an input layer
- given $\boldsymbol{\mu}(\boldsymbol{x})$ and $\boldsymbol{\Sigma}(\boldsymbol{x})$ (*i.e.* mean and covariance of $q(\boldsymbol{z} \mid \boldsymbol{x})$):
  - instead of sampling $\boldsymbol{z}^{(i,l)} \sim \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}^{(i)}), \boldsymbol{\Sigma}(\boldsymbol{x}^{(i)}))$ directly
  - we can compute $\boldsymbol{z}^{(i,l)}$ by
    1. first sampling $\boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    2. then computing

$$\boldsymbol{z}^{(i,l)} = \boldsymbol{\mu}(\boldsymbol{x}^{(i)}) + \boldsymbol{\sigma}(\boldsymbol{x}^{(i)}) \odot \boldsymbol{\epsilon}^{(l)}$$

      - $\odot$: element-wise multiplication
  - $\boldsymbol{z}^{(i,l)}$ has the same distribution as before, but now we can do backprop

original form | reparameterized form

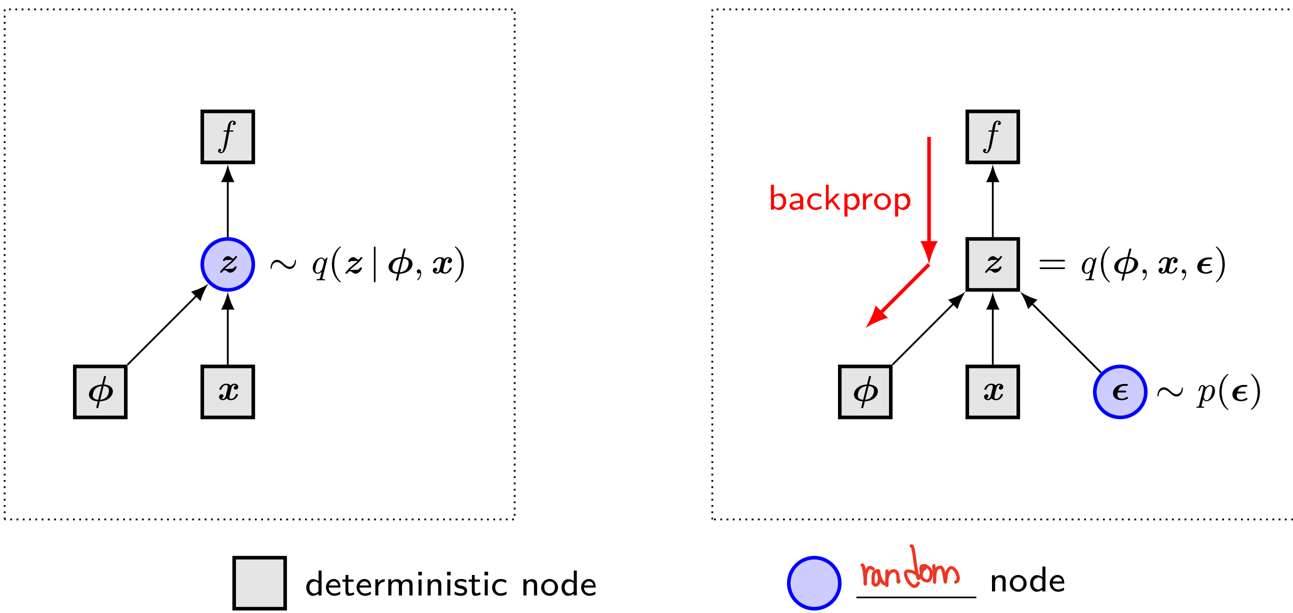


image source: Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*

Training: combined objectives

- finally, training objective for datapoint $\boldsymbol{x}^{(i)}$:

$$\mathcal{L}(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}, \boldsymbol{\phi}) = \underbrace{-D_{\mathrm{KL}}[q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \,||\, p(\boldsymbol{z})]}_{\textbf{encoding} \text{ objective}} + \underbrace{\mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})]}_{\textbf{decoding} \text{ objective}} \tag{3}$$

$$\simeq \frac{1}{2} \sum_{k=1}^{K} \left(1 + \ln \sigma_k^2(\boldsymbol{x}^{(i)}; \boldsymbol{\phi}) - \mu_k^2(\boldsymbol{x}^{(i)}; \boldsymbol{\phi}) - \sigma_k^2(\boldsymbol{x}^{(i)}; \boldsymbol{\phi})\right) + \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} | \boldsymbol{z}^{(i,l)}\right)$$

  - where

$$\boldsymbol{z}^{(i,l)} = \boldsymbol{\mu}(\boldsymbol{x}^{(i)}) + \boldsymbol{\sigma}(\boldsymbol{x}^{(i)}) \odot \boldsymbol{\epsilon}^{(l)} \text{ and } \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

L : Monte Carlo sample size (often $L = 1$)

K : dimensionality of latent variable $\mathbf{z}$

- given a fixed $\boldsymbol{x}$ and $\boldsymbol{\epsilon}$
  - (3) is deterministic and continuous in $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$

$\Rightarrow$ backprop can compute a gradient that will work for SGD

Training summary

training in variational autoencoders (VAEs):

$$\boldsymbol{\theta}^*, \boldsymbol{\phi}^* = \underset{\boldsymbol{\theta}, \boldsymbol{\phi}}{\operatorname{argmax}} \sum_{i=1}^{N} \mathcal{L}(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}, \boldsymbol{\phi})$$

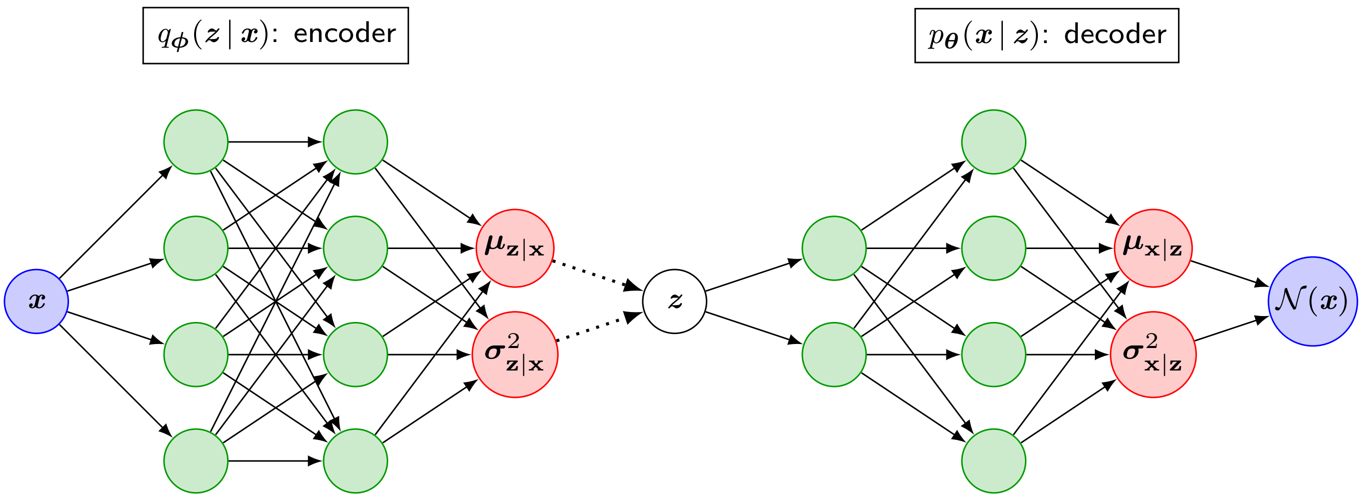
- $\boldsymbol{\theta}$: parameters of generator network ("decoder")
- $\boldsymbol{\phi}$: parameters of inference network ("encoder")
- N : number of training samples ($\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}\}$)
- $\mathcal{L}(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}, \boldsymbol{\phi})$: variational lower bound (see below)

$$\mathcal{L}(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}, \boldsymbol{\phi}) = \underbrace{-D_{\mathrm{KL}}[q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \,||\, p(\boldsymbol{z})]}_{\textbf{encoding}\text{ objective}} + \underbrace{\mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} [\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})]}_{\textbf{decoding}\text{ objective}}$$

$$\simeq \frac{1}{2} \sum_{k=1}^{K} \left(1 + \ln \sigma_k^2(\boldsymbol{x}^{(i)}; \boldsymbol{\phi}) - \mu_k^2(\boldsymbol{x}^{(i)}; \boldsymbol{\phi}) - \sigma_k^2(\boldsymbol{x}^{(i)}; \boldsymbol{\phi})\right) + \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} | \boldsymbol{z}^{(i,l)}\right)$$

$$\boldsymbol{z}^{(i,l)} = \boldsymbol{\mu}(\boldsymbol{x}^{(i)}) + \boldsymbol{\sigma}(\boldsymbol{x}^{(i)}) \odot \boldsymbol{\epsilon}^{(l)} \qquad \text{where} \quad \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

- VAE with Gaussian encoder/decoder:



$$\mathcal{L}(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}, \boldsymbol{\phi}) = \underbrace{-D_{\mathrm{KL}}[q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \,||\, p(\boldsymbol{z})]}_{\text{encoding objective}} + \underbrace{\mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})]}_{\text{decoding objective}}$$

$$\simeq \frac{1}{2} \sum_{k=1}^{K} \left(1 + \ln \sigma^2_{\mathbf{z}|\mathbf{x},k}(\boldsymbol{x}^{(i)}; \boldsymbol{\phi}) - \mu^2_{\mathbf{z}|\mathbf{x},k}(\boldsymbol{x}^{(i)}; \boldsymbol{\phi}) - \sigma^2_{\mathbf{z}|\mathbf{x},k}(\boldsymbol{x}^{(i)}; \boldsymbol{\phi})\right) + \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)} | \boldsymbol{z}^{(i,l)})$$

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)} | \boldsymbol{z}^{(i,l)}) = \log \mathcal{N}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\mu}_{\mathbf{x}|\mathbf{z}}(\boldsymbol{z}^{(i,l)}), \boldsymbol{\sigma}^2_{\mathbf{x}|\mathbf{z}}(\boldsymbol{z}^{(i,l)})\mathbf{I}\right)$$

$$\boldsymbol{z}^{(i,l)} = \boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}}(\boldsymbol{x}^{(i)}) + \boldsymbol{\sigma}_{\mathbf{z}|\mathbf{x}}(\boldsymbol{x}^{(i)}) \odot \boldsymbol{\epsilon}^{(l)} \quad \text{where} \quad \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Sample generation

- when we want to generate a new sample
  - simply input a sample $z \sim \underline{\mathcal{N}(0, I)}$ $= p(z)$: prior into decoder
    - do not use encoder at test time
  - decoder will produce parameters of $p(x \mid z)$
  - we then sample a new x from $p(x \mid z)$

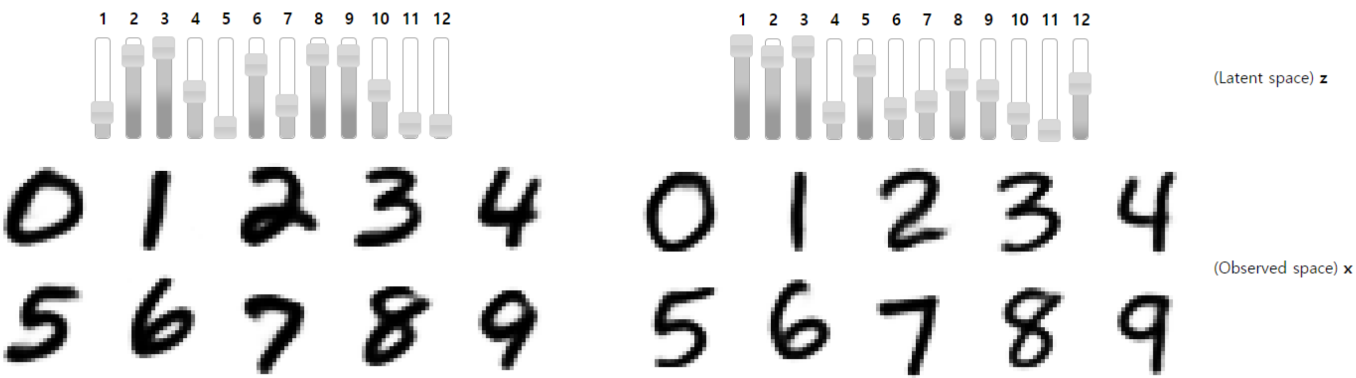


image source: `http://www.dpkingma.com/sgvb_mnist_demo/demo.html`

Outline

VAE in a nutshell

- a latent generative model using **neural nets** and **variational inference**
  - relies on variational inference to resolve the challenge of inference

$$\log p(\boldsymbol{x}) \geq \underbrace{\mathrm{ELBO}(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{\phi})}_{\substack{\boldsymbol{\theta}\text{: parameters for decoder} \\ \boldsymbol{\phi}\text{: parameters for encoder}}} = \underbrace{\mathbb{E}_{\mathbf{z} \sim q(\boldsymbol{z} \mid \boldsymbol{x})}[\log p(\boldsymbol{x} \mid \boldsymbol{z})]}_{\substack{\text{expected log likelihood} \\ \text{of the data}}} - \underbrace{D_{\mathrm{KL}}[q(\boldsymbol{z} \mid \boldsymbol{x}) \,||\, p(\boldsymbol{z})]}_{\substack{D_{\mathrm{KL}}\text{ between} \\ \text{prior } p(\boldsymbol{z}) \text{ and } q(\boldsymbol{z}|\boldsymbol{x})}}$$

  - add "encoder" to learn parametric $q(\boldsymbol{z} \mid \boldsymbol{x})$ that approximates $p(\boldsymbol{z} \mid \boldsymbol{x})$

$$\boldsymbol{x} \xrightarrow[\text{encoder}]{q(\boldsymbol{z} \mid \boldsymbol{x})} \boldsymbol{z} \xrightarrow[\text{decoder}]{p(\boldsymbol{x} \mid \boldsymbol{z})} \boldsymbol{x}$$ (trained as an AE)

    - $p(\boldsymbol{x} \mid \boldsymbol{z})$: now viewed as "decoder" (encoder/decoder: neural nets)
    - encoder learns distribution parameters of $q(\boldsymbol{z} \mid \boldsymbol{x})$ (*e.g.* $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$)
    - $\boldsymbol{z}$ is sampled from $q(\boldsymbol{z} \mid \boldsymbol{x})$ (*e.g.* $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$)
  - train VAE = maximize ELBO (tractable)
    = learn $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ by backprop (+ reparameterization trick)
  - after training: encoder is removed (c.f. decoder is removed in AEs)

$$\xrightarrow[\text{prior}]{p(\boldsymbol{z})} \boldsymbol{z} \xrightarrow[\text{decoder}]{p(\boldsymbol{x} \mid \boldsymbol{z})} \boldsymbol{x}$$

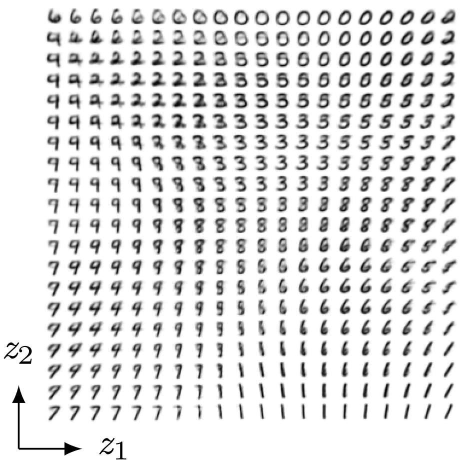
    - use a simple prior $p(\boldsymbol{z}) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ($\Rightarrow D_{\mathrm{KL}}$ above: "regularizer")

Manifold learning by VAE

- nice property of VAE
  - simultaneous training of a parametric encoder with decoder

  $\Rightarrow$ forces to learn a coordinate system the encoder can capture

  $\Rightarrow$ makes VAE an excellent manifold learning algorithm



- example of low-dim manifolds learned by VAE
  - 2D map of MNIST manifold

image source: Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*

Limitations

- main drawback
  - image samples from VAE: blurry
  - cause: some explanations possible but not yet completely known

image source: `https://www.youtube.com/watch?v=XNZIN7Jh3Sg`

Outline

Summary

- autoencoders: stacked/denoising/sparse/contractive and many more
  - copy input to output learning useful representation of input
- variational autoencoder (VAE)
  - probabilistic spin to traditional AEs $\Rightarrow$ allows generating data
  - defines an intractable density $\Rightarrow$ derive and optimize a (variational) lower bound
- VAE advantages: principled approach to generative models
  - allows inference of $q(\boldsymbol{z} \mid \boldsymbol{x})$

  $\Rightarrow$ can be useful feature representation for other tasks
- VAE limitations
  - maximizes lower bound of likelihood (not as good evaluation as PixelCNN)
  - samples blurrier and lower quality compared to state-of-the-art (GANs)
- active areas of research in VAE
  - more flexible approximations (*e.g.* richer approximate posterior than diagonal $\mathcal{N}$)
  - incorporating structure in latent variables