CS 3750 Advanced Machine Learning

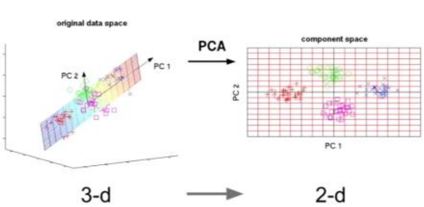
Deep Generative Models

Hung Chau
hkc6@pitt.edu

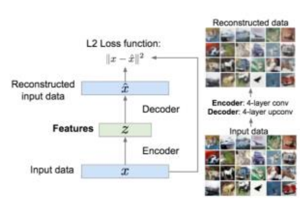
Unsupervised Learning

Data: x
Just data, no labels

Goal: Learn some underlying hidden structure of the data



Principle Component Analysis
(Dimensionality reduction)



Autoencoders
(Feature learning)

Ø

P(x, z)

P(x|z)

P(z|x)

Generative Models

Generative Models

Given training data, generate new samples from same distribution

CIFAR-10 dataset (*Krizhevsky and Hinton, 2009*)

Training data ~ $p_{data}(x)$

Generated sampled ~ $p_{model}(x)$

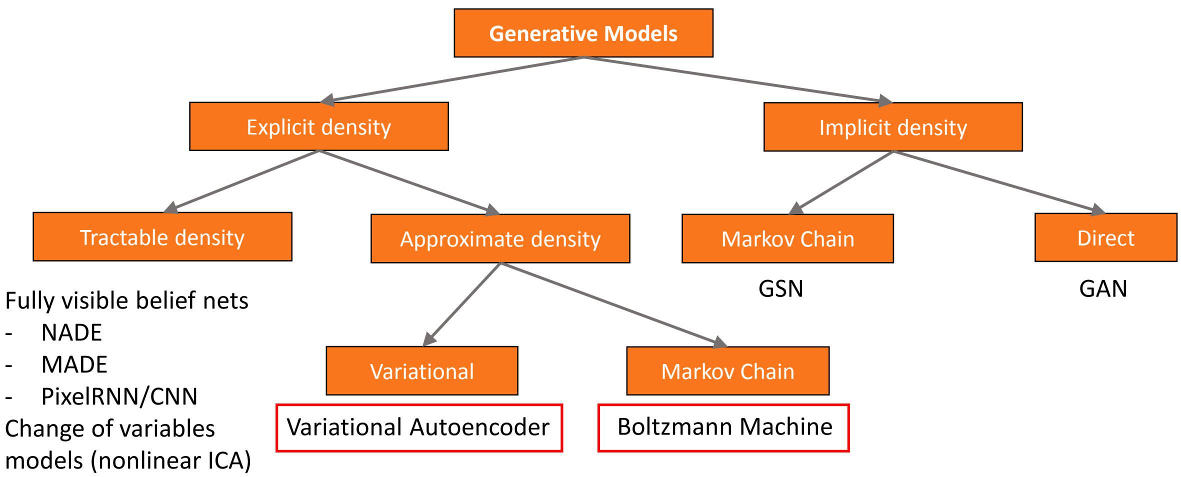
Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

Why Generative Models?

- Realistic samples for artwork, super-resolution, colorization, etc.

- Generative models of time-series data can be used for simulation and planning (reinforcement learning applications!)
- Training generative models can also enable inference of latent representation that can be useful as general features

Taxonomy of Generative Models



Restricted Boltzmann Machines (RBM)

Restricted Boltzmann Machines

Many interesting theoretical results about undirected models depends on the assumption that $\forall x, \tilde{p}(x) > 0$. A convenient way to enforce this condition is to use an *energy-based model* where

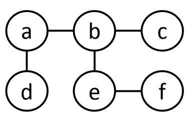
$$\tilde{p}(\boldsymbol{x}) = \exp(-E(\boldsymbol{x}))$$

- E(x) is known as the *energy function*

Remember: normalized probability distribution

$$p(\boldsymbol{x}) = \frac{1}{Z}\tilde{p}(\boldsymbol{x})$$

Any distribution of this form is an example of a *Boltzmann distribution*. For this reason, many energy-based models are called *Boltzmann machines*.



E(a, b, c, d, e, f) can be written as

$E_{a,b}$ (a,b) + $E_{b,c}$ (b,c) + $E_{a,d}$ (a,d) + $E_{b,e}$ (b,e) + $E_{e,f}$ (e,f)

$$\phi_{a,b}(a, b) = \exp(-E(a, b))$$

Restricted Boltzmann Machines

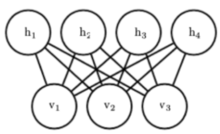
- *Boltzmann machines* were originally introduced as a general “connectionist” approach to learning arbitrary probability distributions over binary vectors
- While Boltzmann machines were defined to encompass both models with and without latent variables, the term Boltzmann machine is today most often used to designate models with latent variables

Joint probability distribution: $p(\boldsymbol{x}, \boldsymbol{h}) = \frac{1}{Z}\exp(-E(\boldsymbol{x}, \boldsymbol{h}))$

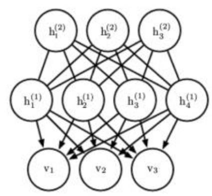
Energy function: $E(\boldsymbol{x}, \boldsymbol{h}) = -\boldsymbol{x}^T\boldsymbol{R}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{W}\boldsymbol{h} - \boldsymbol{h}^T\boldsymbol{S}\boldsymbol{h} - \boldsymbol{c}^T\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{h}$

Restricted Boltzmann Machines

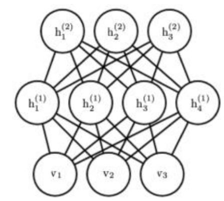
- *Restricted Boltzmann machines (RBMs)* are undirected probabilistic graphical models containing a layer of observable variables and a single layer of latent variables
- RBM is a bipartite graph, with no connections permitted between any variables in the observed layer or between any units in the latent layer



Restricted Boltzmann machine

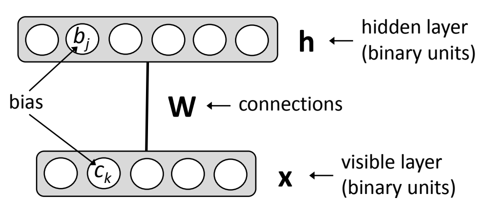


Deep belief network



Deep Boltzmann machine

Restricted Boltzmann Machines



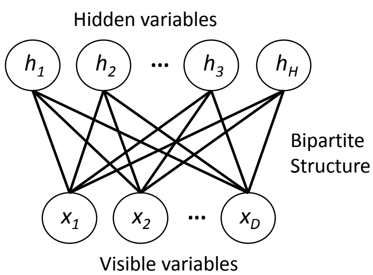
$$p(\boldsymbol{x},\boldsymbol{h}) = \exp\left(-E(\boldsymbol{x},\boldsymbol{h})\right)/Z$$
$$= \exp(\boldsymbol{h}^T\boldsymbol{W}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{h})/Z$$
$$= \underbrace{\exp(\boldsymbol{h}^T\boldsymbol{W}\boldsymbol{x})\exp(\boldsymbol{c}^T\boldsymbol{x})\exp(\boldsymbol{b}^T\boldsymbol{h})/Z}_{\text{Factors}}$$

$$Z = \sum_{\boldsymbol{x},\boldsymbol{h}} \exp(-E(\boldsymbol{x},\boldsymbol{h}))$$

partition function (intractable)

The notation based on an energy function is simply an alternative to the representation as the product of factors

Restricted Boltzmann Machines



Pair-wise Factors

$$p(\boldsymbol{x}, \boldsymbol{h}) = \frac{1}{Z} \prod_j \prod_k exp(W_{j,k} h_j x_k)$$

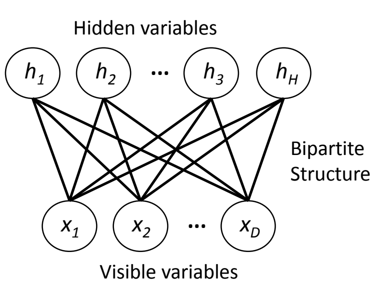
$$\prod_k exp(c_k x_k)$$

$$\prod_j exp(b_j h_j)$$

Unary Factors

The scalar visualization is more informative of the structure within the vectors

RBM: Inference



Restricted: No interaction between hidden variables

Inferring the distribution over the hidden variables is easy

$$p(\boldsymbol{h}|\boldsymbol{x}) = \prod_j p(h_j|\boldsymbol{x})$$

Factorizes: easy to compute

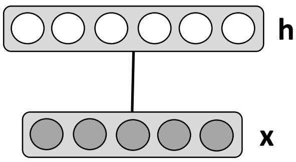
Similarly:

$$p(\boldsymbol{x}|\boldsymbol{h}) = \prod_k p(x_k|\boldsymbol{h})$$

Markov random fields, Boltzmann machines, log-linear models

RBM: Inference

Conditional Distributions

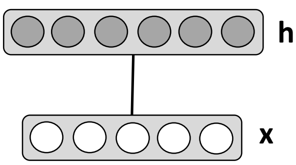


$$p(\boldsymbol{h}|\boldsymbol{x}) = \prod_j p(h_j|\boldsymbol{x})$$

$$p(h_j = 1|\boldsymbol{x}) = \frac{1}{1 + exp\left(-(b_j + \boldsymbol{W}_{j\cdot}\boldsymbol{x})\right)}$$

$$= sigm(b_j + \boldsymbol{W}_{j\cdot}\boldsymbol{x})$$

j^{th} row if W



$$p(\boldsymbol{x}|\boldsymbol{h}) = \prod_k p(x_k|\boldsymbol{h})$$

$$p(x_k = 1|\boldsymbol{h}) = \frac{1}{1 + exp(-(c_k + \boldsymbol{h}^T\boldsymbol{W}_{\cdot k}))}$$

$$= sigm(c_k + \boldsymbol{h}^T\boldsymbol{W}_{\cdot k})$$

k^{th} column if W

RBM: Free Energy

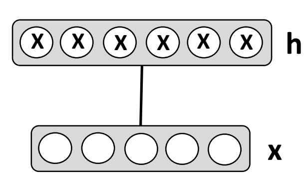
What about computing marginal *p*(x)?

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{h}\in\{0,1\}^H} p(\boldsymbol{x},\boldsymbol{h}) = \sum_{\boldsymbol{h}\in\{0,1\}^H} exp(-E(\boldsymbol{x},\boldsymbol{h}))/Z$$

$$= exp\left(\boldsymbol{c}^T\boldsymbol{x} + \sum_{j=1}^{H}\log\left(1 + exp\left(b_j + \boldsymbol{W}_{j\cdot}\boldsymbol{x}\right)\right)\right)/Z$$

$$= exp(-F(\boldsymbol{x}))/Z$$

Free energy



RBM: Free Energy

What about computing marginal $p(\mathrm{x})$?

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{h}\in\{0,1\}^H} exp(\boldsymbol{h}^T\boldsymbol{W}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{h})/Z$$

$$= exp(\boldsymbol{c}^T\boldsymbol{x}) \sum_{h_1\in\{0,1\}} \dots \sum_{h_H\in\{0,1\}} \exp\left(\sum_j h_j \boldsymbol{W}_j.\boldsymbol{x} + b_j h_j\right)/Z$$

$$= exp(\boldsymbol{c}^T\boldsymbol{x}) \left(\sum_{h_1\in\{0,1\}} \exp(h_1\boldsymbol{W}_1.\boldsymbol{x} + b_1 h_1)\right) \dots \left(\sum_{h_H\in\{0,1\}} \exp(h_H\boldsymbol{W}_H.\boldsymbol{x} + b_H h_H)\right)/Z$$

$$= exp(\boldsymbol{c}^T\boldsymbol{x})(1 + exp(b_1 + \boldsymbol{W}_1.\boldsymbol{x})) \dots (1 + exp(b_H + \boldsymbol{W}_H.\boldsymbol{x}))/Z$$

$$= exp(\boldsymbol{c}^T\boldsymbol{x})\exp(log(1 + exp(b_1 + \boldsymbol{W}_1.\boldsymbol{x}))) \dots \exp(\log((1 + exp(b_H + \boldsymbol{W}_H.\boldsymbol{x}))/Z$$

$$= exp\left(\boldsymbol{c}^T\boldsymbol{x} + \sum_{j=1}^{H} \log\left(1 + exp(b_j + \boldsymbol{W}_j.\boldsymbol{x})\right)\right)/Z$$

Also known as *Product of Experts* model

RBM: Free Energy

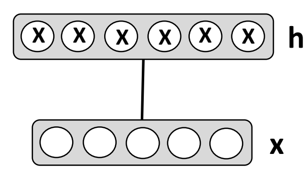
$$p(\boldsymbol{x}) = exp\left(\boldsymbol{c}^T\boldsymbol{x} + \sum_{j=1}^{H} \log\left(1 + exp(b_j + \boldsymbol{W}_j.\boldsymbol{x})\right)\right)/Z$$

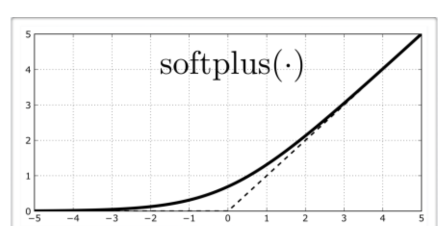
$$= exp\left(\boldsymbol{c}^T\boldsymbol{x} + \sum_{j=1}^{H} \mathrm{softplus}(b_j + \boldsymbol{W}_j.\boldsymbol{x})\right)/Z$$

bias of the probability of each x_i

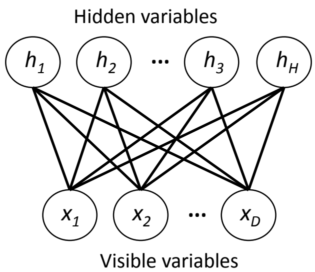
bias of each feature

feature expected in x





RBM: Model Learning



Given a set of *i.i.d.* training examples we want to minimize the average negative log-likelihood:

$$\frac{1}{T}\sum_t l(f(\boldsymbol{x}^{(t)}) = \frac{1}{T}\sum_t -\log p(\boldsymbol{x}^{(t)})$$

Remember:

$$\log p(x^{(t)}) = \log\left(\sum_h p(x^{(t)}, h)\right)$$
$$= \log\left(\sum_h \frac{\exp(-E(x^{(t)}, h))}{Z}\right)$$
$$= \log\left(\sum_h \exp(-E(x^{(t)}, h))\right) - \log Z$$

Derivative of the negative log-likelihood objective (stochastic gradient descent):

$$\frac{\partial -\log p(\boldsymbol{x}^{(t)})}{\partial\theta} = \underbrace{E_h\left[\frac{\partial E(\boldsymbol{x}^{(t)}, \boldsymbol{h})}{\partial\theta} | \boldsymbol{x}^{(t)}\right]}_{\text{Positive phase}} - \underbrace{E_{x,h}\left[\frac{\partial E(\boldsymbol{x}, \boldsymbol{h})}{\partial\theta}\right]}_{\text{Negative phase}}$$

Positive phase

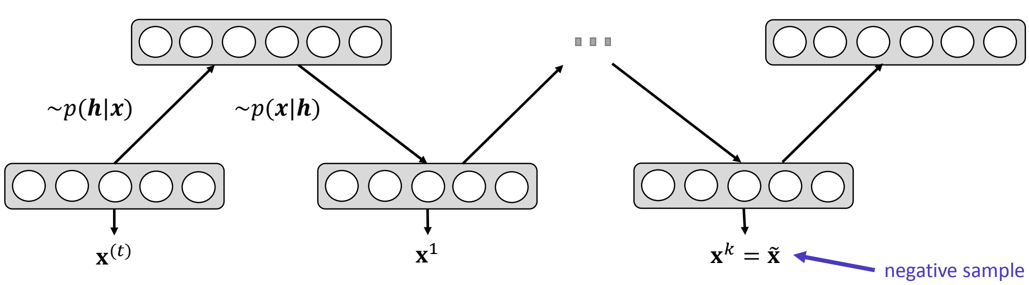
Negative phase
Hard to compute

Restricted Boltzmann machines Tutorial - Chris Maddison

RBM: Contrastive Divergence

Key idea behind Contrastive Divergence:

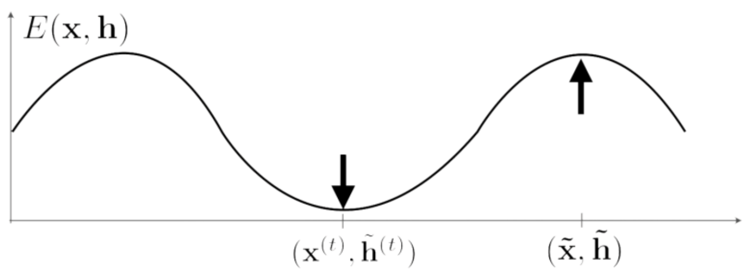
- Replace the expectation by a point estimate at $\tilde{\mathbf{x}}$
- Obtain the point $\tilde{\mathbf{x}}$ by Gibbs sampling
- Start sampling chain at $\mathbf{x}^{(t)}$



RBM: Contrastive Divergence

Intuition: $\frac{\partial -\log p(\boldsymbol{x}^{(t)})}{\partial \theta} = E_h\left[\frac{\partial E(\boldsymbol{x}^{(t)},\boldsymbol{h})}{\partial \theta} | \boldsymbol{x}^{(t)}\right] - E_{x,h}\left[\frac{\partial E(\boldsymbol{x},\boldsymbol{h})}{\partial \theta}\right]$

$E_h\left[\frac{\partial E(\boldsymbol{x}^{(t)},\boldsymbol{h})}{\partial \theta} | \boldsymbol{x}^{(t)}\right] \approx \frac{\partial E(\boldsymbol{x}^{(t)},\tilde{\boldsymbol{h}}^{(t)})}{\partial \theta}$ $E_{x,h}\left[\frac{\partial E(\boldsymbol{x},\boldsymbol{h})}{\partial \theta}\right] \approx \frac{\partial E(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{h}})}{\partial \theta}$

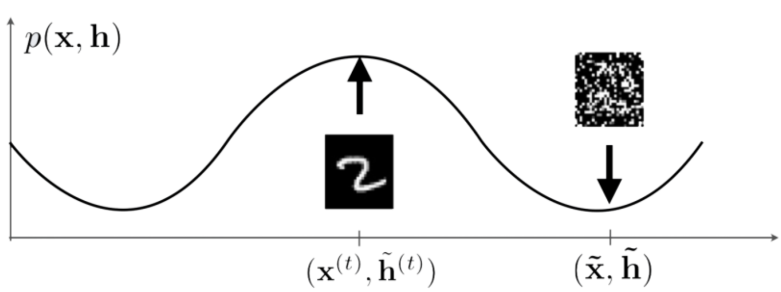


Hung Chau Deep Generative Models

RBM: Contrastive Divergence

Intuition: $\frac{\partial -\log p(\boldsymbol{x}^{(t)})}{\partial \theta} = E_h\left[\frac{\partial E(\boldsymbol{x}^{(t)},\boldsymbol{h})}{\partial \theta} | \boldsymbol{x}^{(t)}\right] - E_{x,h}\left[\frac{\partial E(\boldsymbol{x},\boldsymbol{h})}{\partial \theta}\right]$

$E_h\left[\frac{\partial E(\boldsymbol{x}^{(t)},\boldsymbol{h})}{\partial \theta} | \boldsymbol{x}^{(t)}\right] \approx \frac{\partial E(\boldsymbol{x}^{(t)},\tilde{\boldsymbol{h}}^{(t)})}{\partial \theta}$ $E_{x,h}\left[\frac{\partial E(\boldsymbol{x},\boldsymbol{h})}{\partial \theta}\right] \approx \frac{\partial E(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{h}})}{\partial \theta}$



Hung Chau Deep Generative Models

RBM: Deriving Learning Rule

Let us look at derivative of $\frac{\partial E(\boldsymbol{x},\boldsymbol{h})}{\partial \theta}$ for $\theta = W_{jk}$

$$\frac{\partial E(\boldsymbol{x},\boldsymbol{h})}{\partial \theta} = \frac{\partial}{\partial W_{jk}}\left(-\sum_{jk} W_{j,k}h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j\right)$$

$$= -\frac{\partial}{\partial W_{jk}}\sum_{jk} W_{j,k}h_j x_k$$

$$= -h_j x_k$$

Remember:
$E(\boldsymbol{x},\boldsymbol{h}) = -\boldsymbol{h}^T \boldsymbol{W}\boldsymbol{x} - \boldsymbol{c}^T\boldsymbol{x} - \boldsymbol{b}^T\mathbf{h}$

Hence:

$$\nabla_W E(\boldsymbol{x},\boldsymbol{h}) = -\boldsymbol{h}\boldsymbol{x}^T$$

RBM: Deriving Learning Rule

Let us now derive $\mathbb{E}_h\left[\frac{\partial E(\boldsymbol{x},\boldsymbol{h})}{\partial \theta} | x\right]$

$$\mathbb{E}_h\left[\frac{\partial E(\boldsymbol{x},\boldsymbol{h})}{\partial W_{j,k}} | \boldsymbol{x}\right] = \mathbb{E}_h\left[-h_j x_k | \boldsymbol{x}\right] = \sum_{h_j \in \{0,1\}} -h_j x_k p(h_j | \boldsymbol{x})$$

$$= -x_k p(h_j = 1 | x)$$

$$h(x) = \begin{pmatrix} p(h_1 = 1|x) \\ p(h_H = 1|x) \end{pmatrix}$$

$$= sigm(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{x})$$

Hence:

$$E_h[\nabla_W E(\boldsymbol{x},\boldsymbol{h}) | \boldsymbol{x}] = -\boldsymbol{h}(\boldsymbol{x})\boldsymbol{x}^T$$

RBM: Deriving Learning Rule

$\boldsymbol{x}^{(t)}$ $\qquad \tilde{\boldsymbol{x}}$ $\qquad\qquad \theta = \boldsymbol{W}$

$$\boldsymbol{W} \Leftarrow \boldsymbol{W} - \alpha\left(\nabla_{\boldsymbol{W}} - \log p(x^{(t)})\right)$$

$$\Leftarrow \boldsymbol{W} - \alpha\left(E_h\left[\nabla_{\boldsymbol{W}} E\left(x^{(t)}, h\right) | x^{(t)}\right] - E_{x,h}\left[\nabla_{\boldsymbol{W}} E(x, h)\right]\right)$$

$$\Leftarrow \boldsymbol{W} - \alpha\left(E_h\left[\nabla_{\boldsymbol{W}} E\left(x^{(t)}, h\right) | x^{(t)}\right] - E_h\left[\nabla_{\boldsymbol{W}} E(\tilde{x}, h) | \tilde{x}\right]\right)$$

$$\Leftarrow \boldsymbol{W} + \alpha\left(h(x^{(t)}) x^{(t)^T} - h(\tilde{x}) \tilde{x}^T\right)$$

Learning rate

RBM: CD-k Algorithm

For each training example $x^{(t)}$

- Generate a negative sample $\tilde{x}$ using k steps of Gibbs sampling, starting at the data point $x^{(t)}$
- Update model parameters:

$$\boldsymbol{W} \Leftarrow \boldsymbol{W} + \alpha\left(h(x^{(t)}) x^{(t)^T} - h(\tilde{x}) \tilde{x}^T\right)$$

$$\boldsymbol{b} \Leftarrow \boldsymbol{b} + \alpha\left(h(x^{(t)}) - h(\tilde{x})\right)$$

$$\boldsymbol{c} \Leftarrow \boldsymbol{c} + \alpha\left(x^{(t)} - \tilde{x}\right)$$

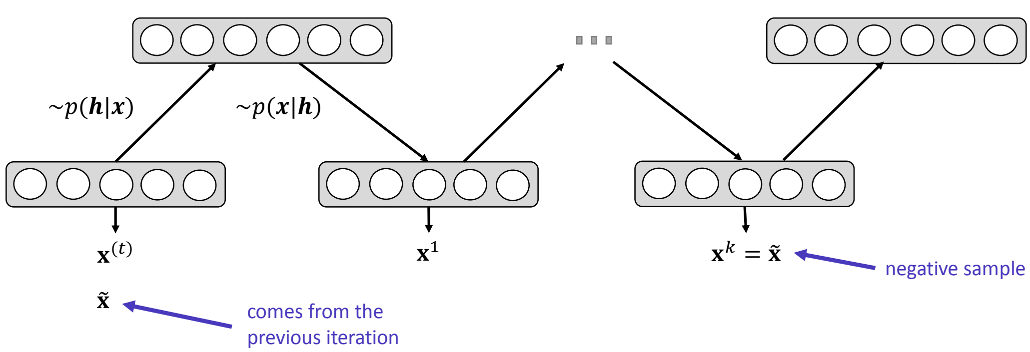
- Go back to the first step until stopping criteria

RBM: CD-k Algorithm

- CD-k: contrastive divergence with k iterations of Gibbs sampling
- In general, the bigger k is, the less biased the estimate of the gradient will be
- In practice, k = 1 works pretty well for learning good features and for pre-training

RBM: Persistent CD: Stochastic ML Estimator

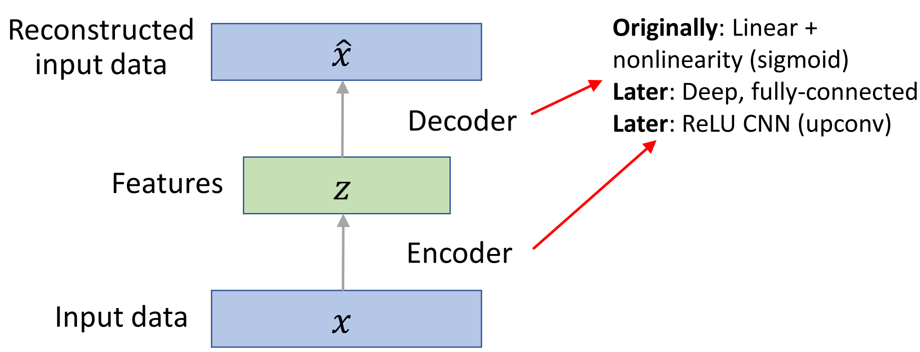
- Idea: instead of initializing the chain of $x^{(t)}$, initialize the chain to the negative sample of the last iteration

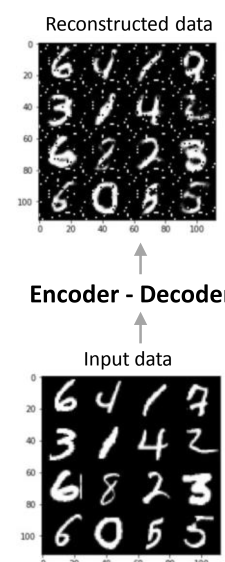


Variational Autoencoders (VAE)

Autoencoders (Recap)

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

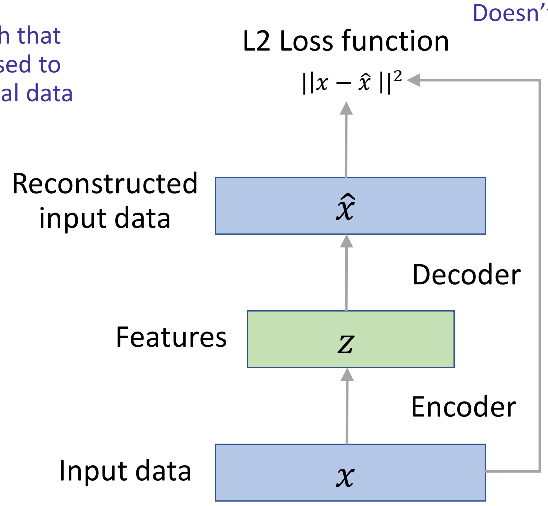


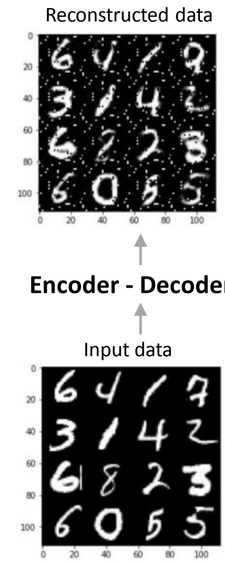


Autoencoders (Recap)

Train a model such that features can be used to reconstruct original data

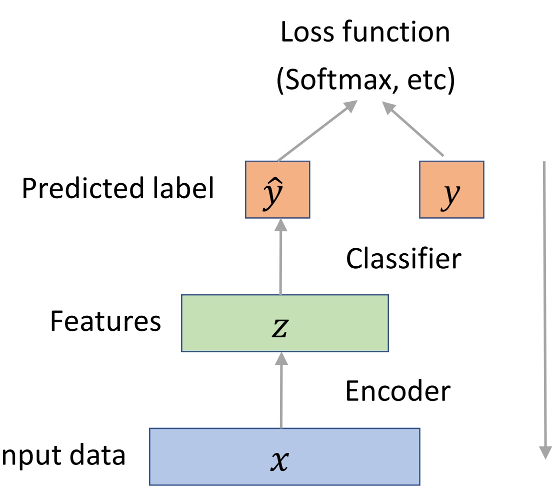
Doesn't use labels!





Autoencoders (Recap)

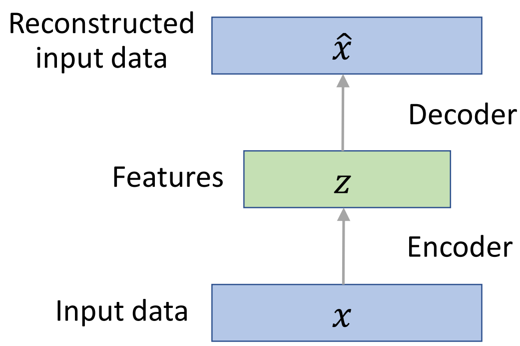
Encoder can be used to initialize a **supervised** model



Fine-tune encoder jointly with classifier

bird plane
panther truck dog

Autoencoders (Recap)



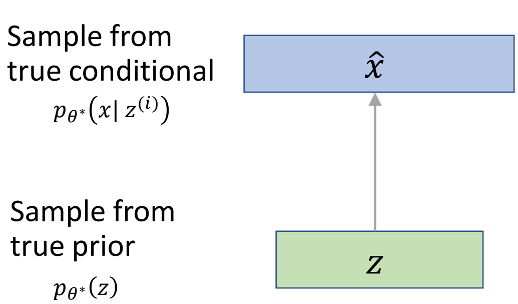
Autoencoders can reconstruct data, and can learn features to initialize a supervised model

Features capture factors of variation in training data. Can we generate new data from an autoencoder?

Variational Autoencoders

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

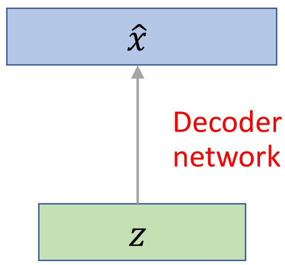
Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation **Z**



Intuition: x is an image, z is latent factors used to generate x: attributes, orientation, etc.

Variational Autoencoders

Sample from
true conditional
$p_{\theta^*}(x|\, z^{(i)})$



Sample from
true prior
$p_{\theta^*}(z)$

We want to estimate the true parameters θ^* of this generative model

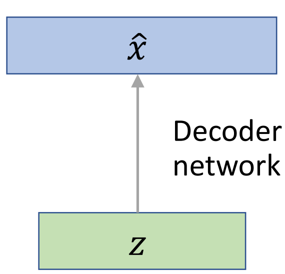
How should we represent this model?

Choose prior p(z) to be simple, e.g. Gaussian

Conditional p(x|z) is complex (generates image) => represent with neural network

Variational Autoencoders

Sample from
true conditional
$p_{\theta^*}(x|\, z^{(i)})$



Sample from
true prior
$p_{\theta^*}(z)$

We want to estimate the true parameters θ^* of this generative model

How train this model?

Learn model parameters to maximize likelihood of training data

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

Variational Autoencoders

Data likelihood: $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$

Intractable to compute p(x|z) for every z!

Posterior density also intractable: $p_\theta(z|x) = \frac{p_\theta(x|z) p_\theta(z)}{p_\theta(x)}$

Solution: in addition to decoder network modeling $p_\theta(x|z)$, define additional encoder network $q_\phi(z|x)$ that approximates $p_\theta(z|x)$

Will see that this allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize

Variational Autoencoders

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic

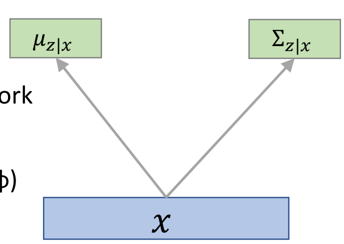
Mean and (diagonal) covariance of **z**|**x**

Mean and (diagonal) covariance of **x**|**z**

Encoder network

$q_\phi(z|x)$

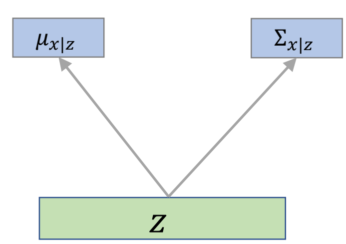
(parameter ϕ)



Decoder network

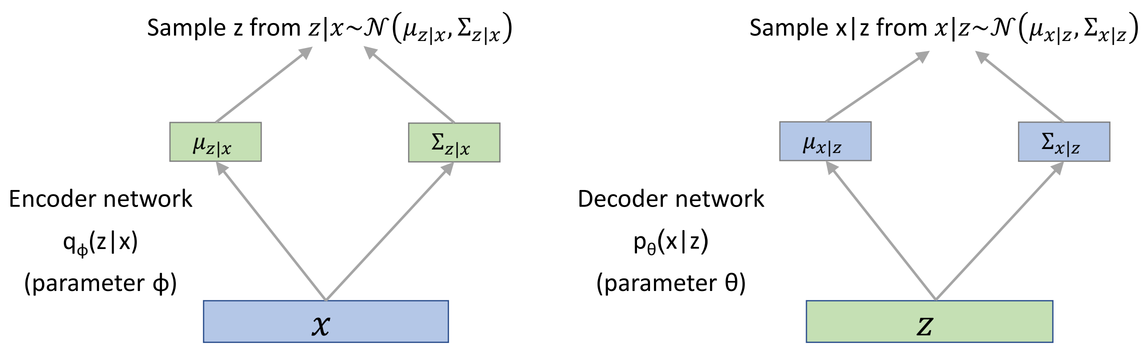
$p_\theta(x|z)$

(parameter θ)



Variational Autoencoders

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic



Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathrm{E}_{z \sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ does not depend on } z)$$

$$= \mathrm{E}_z\left[\log \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathrm{E}_z\left[\log \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})} \frac{q_\phi(z|x^{(i)})}{q_\phi(z|x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathrm{E}_z\left[\log p_\theta(x^{(i)}|z)\right] - \mathrm{E}_z\left[\log \frac{q_\phi(z|x^{(i)})}{p_\theta(z)}\right] + \mathrm{E}_z\left[\log \frac{q_\phi(z|x^{(i)})}{p_\theta(z|x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \mathrm{E}_z\left[\log p_\theta(x^{(i)}|z)\right] - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z)) + D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)}))$$

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling

This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!

$p_\theta(z|x)$ intractable (saw earlier), can't compute this KL term. But we know KL divergence always >= 0.

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathrm{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ does not depend on } z)$$

$$= \mathrm{E}_z\left[\log \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathrm{E}_z\left[\log \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})}\frac{q_\phi(z|x^{(i)})}{q_\phi(z|x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathrm{E}_z\left[\log p_\theta(x^{(i)}|z)\right] - \mathrm{E}_z\left[\log \frac{q_\phi(z|x^{(i)})}{p_\theta(z)}\right] + \mathrm{E}_z\left[\log \frac{q_\phi(z|x^{(i)})}{p_\theta(z|x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \underbrace{\mathrm{E}_z\left[\log p_\theta(x^{(i)}|z)\right] - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)} + \underbrace{D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)}))}_{\geq 0}$$

Tractable lower bound which we can take gradient of and optimize! ($p_\theta(x|z)$ differentiable, KL term differentiable)

Hung Chau   Deep Generative Models

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathrm{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ does not depend on } z)$$

$$= \mathrm{E}_z\left[\log \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathrm{E}_z\left[\log \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})}\frac{q_\phi(z|x^{(i)})}{q_\phi(z|x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathrm{E}_z\left[\log p_\theta(x^{(i)}|z)\right] - \mathrm{E}_z\left[\log \frac{q_\phi(z|x^{(i)})}{p_\theta(z)}\right] + \mathrm{E}_z\left[\log \frac{q_\phi(z|x^{(i)})}{p_\theta(z|x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \underbrace{\mathrm{E}_z\left[\log p_\theta(x^{(i)}|z)\right] - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)} + \underbrace{D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)}))}_{\geq 0}$$

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)},\theta,\phi)$$

Variational lower bound ("ELBO")

$$\theta^*,\phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)},\theta,\phi)$$

Training: Maximize lower bound

Hung Chau   Deep Generative Models

Variational Autoencoders

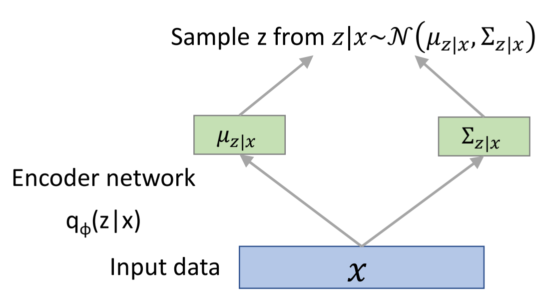
Putting it all together: maximizing the likelihood lower bound

$$\underbrace{E_z\left[\log p_\theta(x^{(i)}|z)\right] - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

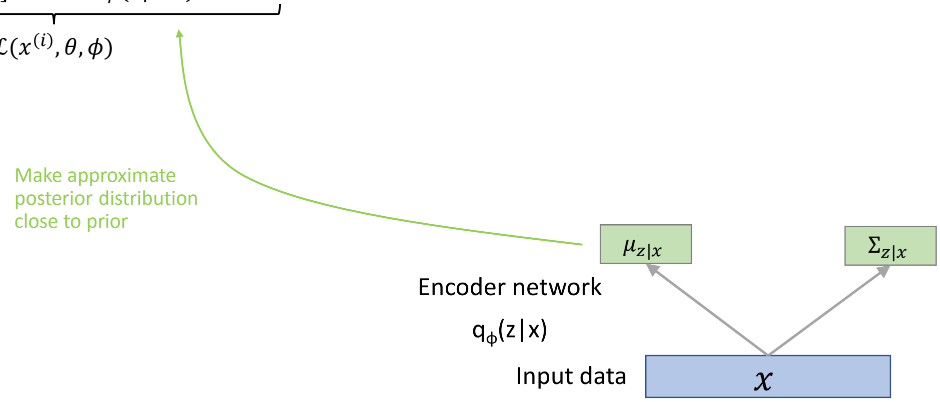
$$\underbrace{E_z\left[\log p_\theta(x^{(i)}|z)\right] - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

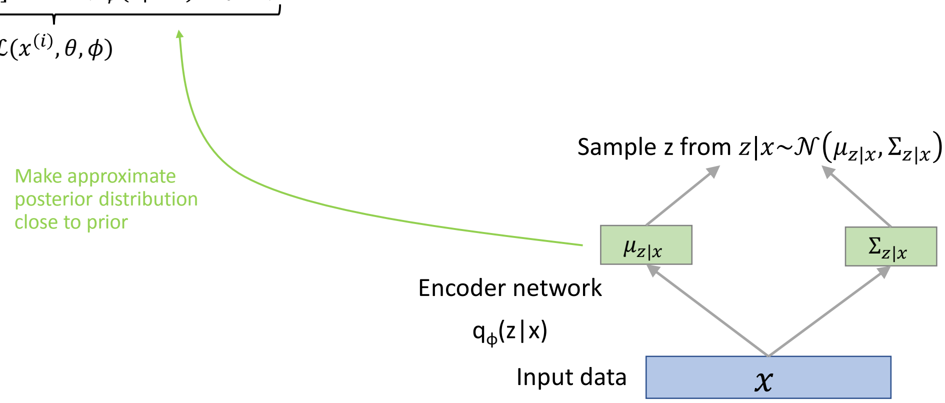
$$\underbrace{E_z\left[\log p_\theta(x^{(i)}|z)\right] - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

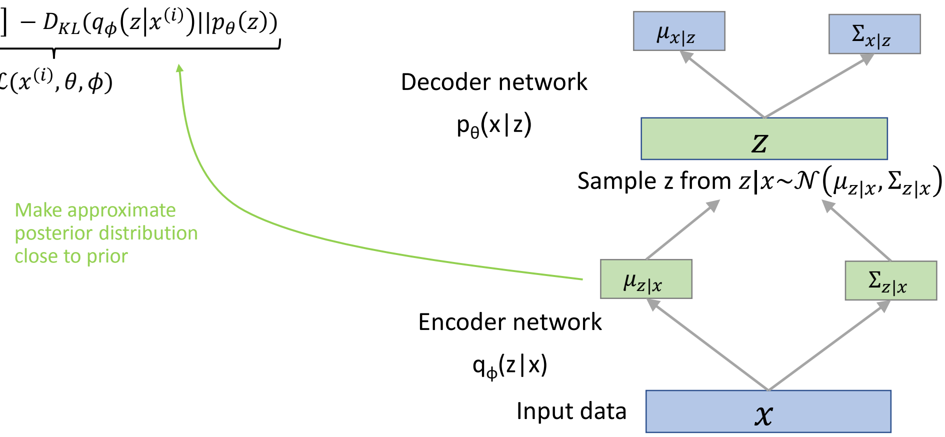
$$\underbrace{E_z\left[\log p_\theta(x^{(i)}|z)\right] - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

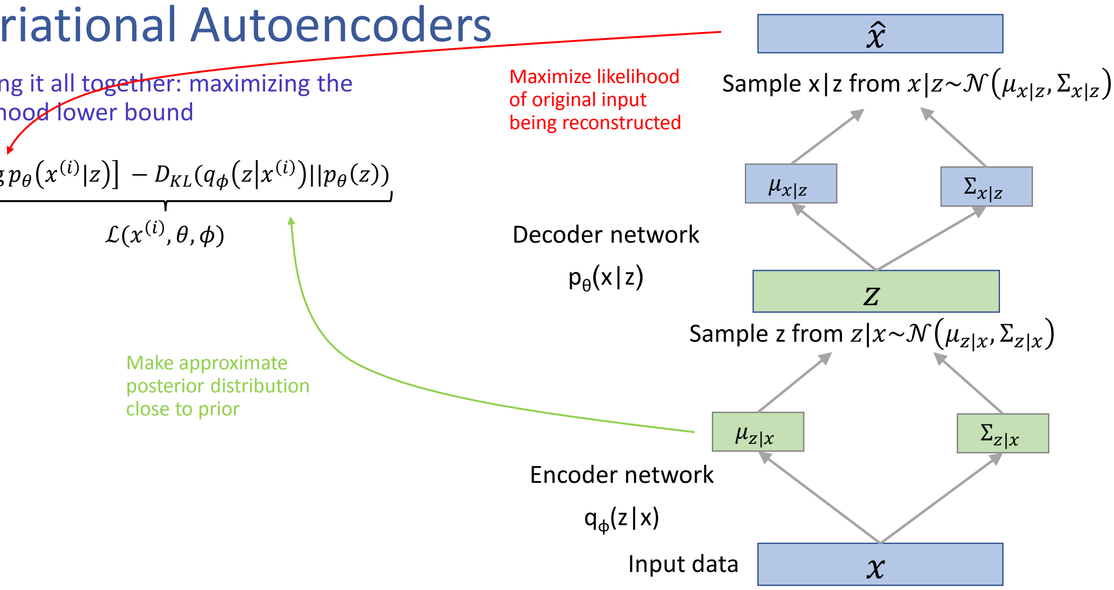
$$\underbrace{E_z\left[\log p_\theta(x^{(i)}|z)\right] - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$



Variational Autoencoders

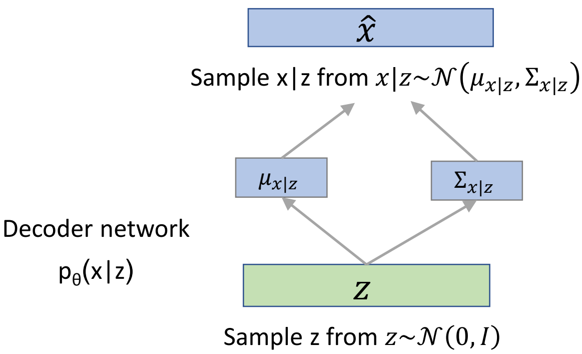
Putting it all together: maximizing the likelihood lower bound

$$\underbrace{E_z\left[\log p_\theta(x^{(i)}|z)\right] - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$



Variational Autoencoders

Use decoder network. Now sample z from prior!

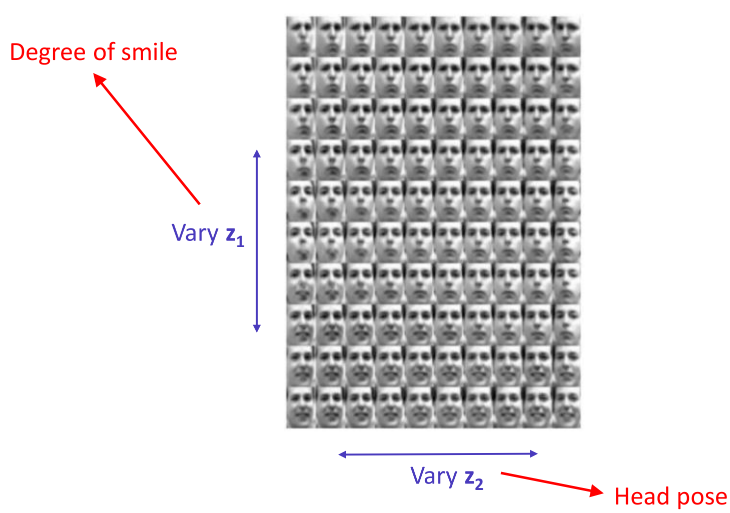


Hung Chau | Deep Generative Models

Variational Autoencoders

Diagonal prior on **z** => independent latent variables

Different dimensions of **z** encode interpretable factors of variation



Hung Chau | Deep Generative Models

Variational Autoencoders

Diagonal prior on **z**
=> independent
latent variables

Different dimensions
of **z** encode
interpretable factors
of variation

Also good feature representation that
can be computed using $q_\phi(z|x)$!

Degree of smile

Vary $\mathbf{z_1}$

Vary $\mathbf{z_2}$

Head pose

Variational Autoencoders: Generating Data

32x32 CIFAR-10

Labeled Faces in the Wild

Figures (L) from Dirk Kingma et al. 2016; (R) from Anders Larsen et al. 2017

Variational Autoencoders

Probabilistic spin to traditional autoencoders => allows generating data

Pros:
- Principled approach to generative models
- Allows inference of q(z|x), can be useful feature representation for other tasks

Cons:
- Maximizes lower bound of likelihood: okay
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

Active areas of research:
- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian
- Incorporating structure in latent variables

Tools

- Python library for Bernoulli Restricted Boltzmann Machines: *sklearn.neural_network.BernoulliRBM*
- Python Keras for Variational auto-encoder
- Generative models (including RBM and VAE): https://github.com/wiseodd/generative-models
- Variational Auto-encoder:
  - Tutorials: http://pyro.ai/examples/vae.html
  - Codes: https://github.com/uber/pyro/tree/dev/examples/vae

References and Resources

- Goodfellow, Ian, et al. Deep learning. Vol. 1. Cambridge: MIT press, 2016.
- Diederik P Kingma, Max Welling: *Auto-Encoding Variational Bayes. ICLR 2014*
- *Restricted Boltzmann machines Tutorial - Chris Maddison:* *https://www.cs.toronto.edu/~tijmen/csc321/documents/maddison_rbmtutorial.pdf*
- Geoffrey E. Hinton: *Training products of experts by minimizing contrastive divergence.* Neural Computation (2002)
- Generative Models Lecture (Stanford University): http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf
- Restricted Boltzmann Machines (CMU): https://www.youtube.com/watch?v=JfpP9CY1EFo
- Hugo Larochelle's class on Neural Networks: http://info.usherbrooke.ca/hlarochelle/neural_networks/content.html
- Image Generation (ICLR 2018): https://www.youtube.com/watch?v=G06dEcZ-QTg

THANK YOU!!!

Hung Kim Chau

University of Pittsburgh
School of Computing and Information
Department of Informatics and Networked Systems

135 N Bellefield Ave, Pittsburgh, PA, 15260
Phone: +1 (412) 552-3055