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)

Generative Models

Hung Chau
Deep Generative Models
Generative Models

Given training data, generate new samples from same distribution

CIFAR-10 dataset (Krizhevsky and Hinton, 2009)

Training data $\sim p_{\text{data}}(x)$
Generated sampled $\sim p_{\text{model}}(x)$

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

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

Generative Models

Explicit density
- Tractable density
- Approximate density

Implicit density
- Markov Chain
- Direct

Variational
- Variational Autoencoder

Markov Chain
- Boltzmann Machine

Fully visible belief nets
- NADE
- MADE
- PixelRNN/CNN

Change of variables models (nonlinear ICA)

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

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

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

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

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

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

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

Energy function: $E(x, h) = -x^T Rx - x^T Wh - h^T Sh - c^T x - b^T h$
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.

\[
p(x,h) = \frac{\exp(-E(x,h))}{Z} = \frac{\exp(h^T W x + c^T x + b^T h)}{Z} = \frac{\exp(h^T W x) \cdot \exp(c^T x) \cdot \exp(b^T h)}{Z}
\]

Factors

\[Z = \sum_{x,h} \exp(-E(x,h))\] 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

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

\[
p(x, h) = \frac{1}{Z} \prod_j \prod_k \exp(W_{jk}h_jx_k)
\]

\[
\prod_k \exp(c_kx_k)
\]

\[
\prod_j \exp(b_jh_j)
\]

RBM: Inference

**Restricted**: No interaction between hidden variables

Inferring the distribution over the hidden variables is easy

\[
p(h|x) = \prod_j p(h_j|x)
\]

Similarly:

\[
p(x|h) = \prod_k p(x_k|h)
\]

Markov random fields, Boltzmann machines, log-linear models
RBM: Inference

Conditional Distributions

\[
p(h|x) = \prod_j p(h_j|x) \\
p(h_j = 1|x) = \frac{1}{1 + \exp\left(-\left(b_j + W_j.x\right)\right)} \\
= \text{sigm}(b_j + W_j.x)
\]

\(j^{th}\) row if \(W\)

\[
p(x|h) = \prod_k p(x_k|h) \\
p(x_k = 1|h) = \frac{1}{1 + \exp\left(-\left(c_k + h^T W_k\right)\right)} \\
= \text{sigm}(c_k + h^T W_k)
\]

\(k^{th}\) column if \(W\)

RBM: Free Energy

What about computing marginal \(p(x)\)?

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

\[
= \exp\left(c^T x + \sum_{j=1}^{H} \log\left(1 + \exp\left(b_j + W_j.x\right)\right)\right)/Z
\]

\[
= \exp(-F(x))/Z
\]
RBM: Free Energy

What about computing marginal $p(x)$?

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

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

$$= \exp(c^T x) \left( \sum_{h_1 \in \{0,1\}} \exp(h_1 W_1 . x + b_1 h_1) \right) \cdots \left( \sum_{h_H \in \{0,1\}} \exp(h_H W_H . x + b_H h_H) \right) / Z$$

$$= \exp(c^T x) \left( 1 + \exp(b_1 + W_1 . x) \right) \cdots \left( 1 + \exp(b_H + W_H . x) \right) / Z$$

$$= \exp(c^T x) \exp(\log(1 + \exp(b_1 + W_1 . x))) \cdots \exp(\log(1 + \exp(b_H + W_H . x))) / Z$$

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

Also known as **Product of Experts** model

---

**Hung Chau**

Deep Generative Models
RBM: Model Learning

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

\[
\frac{1}{T} \sum_{t} l(f(x^{(t)}) = \frac{1}{T} \sum_{t} -\log p(x^{(t)})
\]

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

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

Key idea behind Contrastive Divergence:

\begin{itemize}
    \item Replace the expectation by a \textbf{point estimate} at $\tilde{x}$
    \item Obtain the point $\tilde{x}$ by Gibbs sampling
    \item Start sampling chain at $x^{(t)}$
\end{itemize}
RBM: Contrastive Divergence

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

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

Hung Chau
Deep Generative Models

RBM: Contrastive Divergence

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

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

Hung Chau
Deep Generative Models
RBM: Deriving Learning Rule

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

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

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

$$= -h_j x_k$$

Hence:

$$\nabla_w E(x, h) = -hx^T$$

Remember:

$$E(x, h) = -h^T W x - c^T x - b^T h$$

RBM: Deriving Learning Rule

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

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

$$= -x_k p(h_j = 1 \mid x)$$

Hence:

$$\mathbb{E}_h [\nabla_w E(x, h) \mid x] = -h(x)x^T$$

Remember:

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

$$= \text{sigm}(b + Wx)$$
RBM: Deriving Learning Rule

\[ x^{(t)} \quad \bar{x} \quad \theta = W \]

\[ W \leftarrow W - \alpha (\nabla W - \log p(x^{(t)})) \]
\[ \leftarrow W - \alpha (E_h[\nabla W E(x^{(t)}, h)|x^{(t)}] - E_{x,h}[\nabla W E(x, h)]) \]
\[ \leftarrow W - \alpha (E_h[\nabla W E(x^{(t)}, h)|x^{(t)}] - E_h[\nabla W E(\bar{x}, h)|\bar{x}]) \]
\[ \leftarrow W + \alpha (h(x^{(t)})x^{(t)T} - h(\bar{x})\bar{x}^T) \]

Learning rate

RBM: CD-k Algorithm

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

- Generate a negative sample \( \bar{x} \) using k steps of Gibbs sampling, starting at the data point \( x^{(t)} \)
- Update model parameters:
  \[ W \leftarrow W + \alpha (h(x^{(t)})x^{(t)T} - h(\bar{x})\bar{x}^T) \]
  \[ b \leftarrow b + \alpha (h(x^{(t)}) - h(\bar{x})) \]
  \[ c \leftarrow c + \alpha (x^{(t)} - \bar{x}) \]
- 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

\[ x^{(t)} \sim p(h|x), \quad x^1 \sim p(x|h) \]

\[ x^k = \hat{x} \]

Negative sample comes from the previous iteration
Variational Autoencoders (VAE)

Hung Chau

Deep Generative Models

Autoencoders (Recap)

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

Reconstructed input data

Features

Input data

Decoder

Encoder

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN (upconv)

Hung Chau

Deep Generative Models
Autoencoders (Recap)

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

- **Input data** $x$
- **Features** $z$
- **Reconstructed input data** $\hat{x}$

**L2 Loss function**

$||x - \hat{x}||^2$

Doesn’t use labels!

**Encoder - Decoder**

**Input data**

---

Autoencoders (Recap)

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

- **Input data** $x$
- **Features** $z$
- **Predicted label** $\hat{y}$
- **Loss function** (Softmax, etc)

Fine-tune encoder jointly with classifier.

**Predicted label** $\hat{y}$

**Loss function**

Bird, plane, panther, truck, dog
**Autoencoders (Recap)**

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

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

- **Input data** \( x \)
- **Encoder** \( z \)
- **Features**
- **Decoder** \( \hat{x} \)
- **Reconstructed input data**

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 \( Z \).

- **Sample from true conditional**
  \( p_{\theta^*}(x|z^{(0)}) \)
  \( \hat{x} \)
- **Sample from true prior**
  \( p_{\theta^*}(z) \)
  \( z \)

**Intuition:** \( x \) is an image, \( z \) is latent factors used to generate \( x \): attributes, orientation, etc.
We want to estimate the true parameters $\theta^*$ of this generative model.

**Variational Autoencoders**

**Sample from true conditional** $p_{\theta^*}(x|z^{(0)})$

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

$\hat{x}$

Decoder network

$z$

**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.

**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 \( \phi \))

Decoder network \( p_{\theta}(x|z) \) (parameter \( \theta \))
Variational Autoencoders

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

Sample $z$ from $z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$

Encoder network
$q_{\phi}(z|x)$
(parameter $\phi$)

Decoder network
$p_{\theta}(x|z)$
(parameter $\theta$)

$x$

$z$

$p_{\theta}(z|x)$ intractable (saw earlier), can’t compute this KL term. But we know KL divergence always $\geq 0$. 

$p_{\phi}(z|x)$ intracatable (saw earlier), can’t compute this KL term. But we know KL divergence always $\geq 0$. 

Variational Autoencoders

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

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

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!
Variational Autoencoders

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

\[
\log p_\theta(x^{(i)}) = E_{z \sim q_\phi(z|x^{(i)})} \left[ \log p_\theta(x^{(i)}) \right]
\]  
\[
= E_x \left[ \log \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})} \right] \quad \text{ (Bayes’ Rule)}
\]

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

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

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

\[
\geq 0 \quad \mathcal{L}(x^{(i)}, \theta, \phi)
\]

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

Variational lower bound (“ELBO”)

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

Training: Maximize lower bound
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

\[ \mathcal{L}(x^{(i)}, \theta, \phi) \]
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

Make approximate posterior distribution close to prior

Hung Chau
Deep Generative Models
Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$E_x[\log p_\theta(x^{(i)}|x)] - D_{KL}(q_\phi(z|x^{(i)}) || p_\theta(z))$$

Make approximate posterior distribution close to prior

Decoder network

$p_\theta(x|z)$

Sample $z$ from $z|x \sim \mathcal{N}(\mu_x|z, \Sigma_x|z)$

Encoder network

$q_\phi(z|x)$

Input data $x$

Hung Chau

Deep Generative Models

Maximize likelihood of original input being reconstructed

Hung Chau

Deep Generative Models
Variational Autoencoders

Use decoder network. Now sample $z$ from prior!

$\hat{x}$

Sample $x|z$ from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$

Decoder network
$p_\theta(x|z)$

Sample $z \sim \mathcal{N}(0, I)$

Variational Autoencoders

Diagonal prior on $z$ => independent latent variables

Different dimensions of $z$ encode interpretable factors of variation

Degree of smile

Vary $z_1$

Head pose

Vary $z_2$
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 $z_1$

Vary $z_2$

Head pose

Hung Chau

Deep Generative Models

Variational Autoencoders: Generating Data

32x32 CIFAR-10

Labeled Faces in the Wild

Hung Chau

Deep Generative Models
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

- Diederik P Kingma, Max Welling: Auto-Encoding Variational Bayes. ICLR 2014
- Restricted Boltzmann Machines (CMU): https://www.youtube.com/watch?v=Ifp9CY1EFO
- 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=G06EcZ-Q7g

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