Neural Network Part 5: Unsupervised Models

CS 760@UW-Madison
Goals for the lecture

you should understand the following concepts

• autoencoder
• restricted Boltzmann machine (RBM)
• Nash equilibrium
• minimax game
• generative adversarial network (GAN)
Autoencoder
Autoencoder

• Neural networks trained to attempt to copy its input to its output

• Contain two parts:
  • Encoder: map the input to a hidden representation
  • Decoder: map the hidden representation to the output
Autoencoder

- Input: \( x \)
- Hidden representation (the code): \( h \)
- Reconstruction: \( r \)
Autoencoder

Encoder $f(\cdot)$

Decoder $g(\cdot)$

$h = f(x), \ r = g(h) = g(f(x))$
Why want to copy input to output

• Not really care about copying

• Interesting case: NOT able to copy exactly but strive to do so

• Autoencoder forced to select which aspects to preserve and thus hopefully can learn useful properties of the data

• Historical note: goes back to (LeCun, 1987; Bourlard and Kamp, 1988; Hinton and Zemel, 1994).
Undercomplete autoencoder

- Constrain the code to have smaller dimension than the input
- Training: minimize a loss function

\[ L(x, r) = L(x, g(f(x))) \]
Undercomplete autoencoder

• Constrain the code to have smaller dimension than the input
• Training: minimize a loss function

\[ L(x, r) = L(x, g(f(x))) \]

• Special case: \( f, g \) linear, \( L \) mean square error
• Reduces to Principal Component Analysis
Undercomplete autoencoder

- What about nonlinear encoder and decoder?

- Capacity should not be too large

- Suppose given data \( x_1, x_2, \ldots, x_n \)
  - Encoder maps \( x_i \) to \( i \)
  - Decoder maps \( i \) to \( x_i \)

- One dim \( h \) suffices for perfect reconstruction
Regularization

- Typically NOT
  - Keeping the encoder/decoder shallow or
  - Using small code size

- Regularized autoencoders: add regularization term that encourages the model to have other properties
  - Sparsity of the representation (sparse autoencoder)
  - Robustness to noise or to missing inputs (denoising autoencoder)
Sparse autoencoder

- Constrain the code to have sparsity
- Training: minimize a loss function

\[ L_R = L(x, g(f(x))) + R(h) \]
Probabilistic view of regularizing $h$

- Suppose we have a probabilistic model $p(h, x)$
- MLE on $x$
  \[
  \log p(x) = \log \sum_{h'} p(h', x)
  \]
- 😞 Hard to sum over $h'$
• Suppose we have a probabilistic model $p(h, x)$
• MLE on $x$

$$\max \log p(x) = \max \log \sum_{h'} p(h', x)$$

• Approximation: suppose $h = f(x)$ gives the most likely hidden representation, and $\sum_{h'} p(h', x)$ can be approximated by $p(h, x)$
Probabilistic view of regularizing $h$

- Suppose we have a probabilistic model $p(h, x)$
- Approximate MLE on $x, h = f(x)$

$$\max \log p(h, x) = \max \log p(x|h) + \log p(h)$$
Sparse autoencoder

- Constrain the code to have sparsity
- Laplacian prior: \( p(h) = \frac{\lambda}{2} \exp(-\frac{\lambda}{2} |h|_1) \)

- Training: minimize a loss function
  \[
  L_R = L(x, g(f(x))) + \lambda |h|_1
  \]
Denoising autoencoder

- Traditional autoencoder: encourage to learn $g(f(\cdot))$ to be identity

- Denoising: minimize a loss function

$$L(x, r) = L(x, g(f(\tilde{x})))$$

where $\tilde{x}$ is $x + noise$
Boltzmann Machine
Boltzmann machine

• Introduced by Ackley et al. (1985)

• General “connectionist” approach to learning arbitrary probability distributions over binary vectors

• Special case of energy model: \( p(x) = \frac{\exp(-E(x))}{Z} \)
Boltzmann machine

- Energy model:
  \[ p(x) = \frac{\exp(-E(x))}{Z} \]

- Boltzmann machine: special case of energy model with
  \[ E(x) = -x^T U x - b^T x \]

where \( U \) is the weight matrix and \( b \) is the bias parameter
Boltzmann machine with latent variables

• Some variables are not observed

\[ x = (x_v, x_h), \quad x_v \text{ visible, } x_h \text{ hidden} \]

\[ E(x) = -x_v^T R x_v - x_v^T W x_h - x_h^T S x_h - b^T x_v - c^T x_h \]

• Universal approximator of probability mass functions
Maximum likelihood

• Suppose we are given data $X = (x_v^1, x_v^2, \ldots, x_v^n)$

• Maximum likelihood is to maximize

$$\log p(X) = \sum_i \log p(x_v^i)$$

where

$$p(x_v) = \sum_{x_h} p(x_v, x_h) = \sum_{x_h} \frac{1}{Z} \exp(-E(x_v, x_h))$$

• $Z = \sum \exp(-E(x_v, x_h))$: partition function, difficult to compute
Restricted Boltzmann machine

• Invented under the name *harmonium* (Smolensky, 1986)
• Popularized by Hinton and collaborators to *Restricted Boltzmann machine*
Restricted Boltzmann machine

- Special case of Boltzmann machine with latent variables:

\[
p(v, h) = \frac{\exp(-E(v, h))}{Z}
\]

where the energy function is

\[
E(v, h) = -v^TWh - b^Tv - c^T h
\]

with the weight matrix \( W \) and the bias \( b, c \)

- Partition function

\[
Z = \sum_v \sum_h \exp(-E(v, h))
\]
Restricted Boltzmann machine

Figure from *Deep Learning*, Goodfellow, Bengio and Courville
Restricted Boltzmann machine

• Conditional distribution is factorial

\[
p(h|v) = \frac{p(v, h)}{p(v)} = \prod_j p(h_j|v)
\]

and

\[
p(h_j = 1|v) = \sigma(c_j + v^T W_{.:j})
\]

is logistic function
Restricted Boltzmann machine

• Similarly,

\[ p(v|h) = \frac{p(v, h)}{p(h)} = \prod_i p(v_i|h) \]

and

\[ p(v_i = 1|h) = \sigma(b_i + W_{i,:}h) \]

is logistic function
Generative Adversarial Networks (GAN)

See Ian Goodfellow’s tutorial slides:
Generative Adversarial Networks

- **Approach:** Set up zero-sum game between deep nets to
  - **Generator:** Generate data that looks like training set
  - **Discriminator:** Distinguish between real and synthetic data

- **Motivation:**
  - Building accurate generative models is hard (e.g., *learning and sampling* from Markov net or Bayes net)
  - Want to use all our great progress on *supervised learners* to do this *unsupervised* learning task better
  - Deep nets may be our favorite supervised learner, especially for image data, if nets are convolutional (use tricks of sliding windows with parameter tying, cross-entropy transfer function, batch normalization)
Does It Work?

Thanks, Ian Goodfellow, NIPS 2016 Tutorial on GANS, for this and most of what follows…
A Bit More on GAN Algorithm

$D(x)$ tries to be near 1

Differentiable function $D$

$x$ sampled from data

$D$ tries to make $D(G(z))$ near 0, $G$ tries to make $D(G(z))$ near 1

$x$ sampled from model

Differentiable function $G$

Input noise $z$
The Rest of the Details

• Use deep convolutional neural networks for Discriminator D and Generator G

• Let $x$ denote trainset and $z$ denote random, uniform input

• Set up zero-sum game by giving D the following objective, and G the negation of it:

$$-\frac{1}{2}\mathbb{E}_{x \sim p_{data}} \log D(x) - \frac{1}{2}\mathbb{E}_{z} \log (1 - D(G(z)))$$

• Let D and G compute their gradients simultaneously, each make one step in direction of the gradient, and repeat until neither can make progress… Minimax
More Math on GAN

• Real data distribution $q(x)$
• Fake data distribution $p_\theta(x) := [x = G_\theta(z), z \sim p(z)]$
• Discriminator $D(x) := \sigma(f_\beta(x))$

• GAN: $\min_\theta \max_\beta E_{q(x)} \log \sigma(f_\beta(x)) + E_{p_\theta(x)} \log(1 - \sigma(f_\beta(x)))$

• Given $\theta$, the optimal $f_\beta^*(x) = \log \frac{q(x)}{p_\theta(x)}$ (not necessarily realizable by the discriminator neural network)
• Plug $f_\beta^*(x)$ back in GAN: $\min_\theta JS(p_\theta, q)$

• Recall
  • Jensen-Shannon divergence $JS(p, q) = KL\left(p \parallel \frac{p+q}{2}\right) + KL\left(q \parallel \frac{p+q}{2}\right)$
  • Kullback-Leibler divergence $KL(p \parallel q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$
  • Maximum Likelihood Estimate $\min_\theta KL(q \parallel p_\theta)$
THANK YOU

Some of the slides in these lectures have been adapted/borrowed from materials developed by Yingyu Liang, Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Matt Gormley, Elad Hazan, Tom Dietterich, Pedro Domingos, Geoffrey Hinton, and Ian Goodfellow.