**CS 6140: Machine Learning**  
Spring 2016

Instructor: Lu Wang  
College of Computer and Information Science  
Northeastern University  
Webpage: [www.ccs.neu.edu/home/luwang](http://www.ccs.neu.edu/home/luwang)  
Email: luwang@ccs.neu.edu

---

**Logistics**

- Assignment 1 solution is expected to be out by the end of next week.
- Assignment 2 is due on 03/02.

---

**What we learned last time**

- Feedforward neural network
- Training neural networks
- Restricted Boltzmann machine

---

**NEURAL NETWORK**

- Single hidden layer neural network
- Hidden layer pre-activation:  \( a(x) = b(1) + W(1)x \)
- Hidden layer activation:  \( h(x) = g(a(x)) \)
- Output layer activation:  \( f(x) = a(2) + w(2)^T h(1)x \)

**ACTIVATION FUNCTION**

- Linear activation function:  \( g(a) = \alpha \)
  - Performs no input squashing
  - Not very interesting...

- Sigmoid activation function:  \( g(a) = \frac{1}{1 + \exp(-a)} \)
  - Squashes the neuron's pre-activation between 0 and 1
  - Always positive
  - Bounded
  - Strictly increasing
**ACTIVATION FUNCTION**

**Topics**
- Hyperbolic tangent ("tanh") activation function
  - Squashes the neuron's pre-activation between -1 and 1
  - Can be positive or negative
  - Bounded
  - Strictly increasing

\[ g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(3a) - 1}{\exp(3a) + 1} \]

**ACTIVATION FUNCTION**

**Topics**
- Rectified linear activation function
  - Bounded below by 0 (always non-negative)
  - Not upper bounded
  - Strictly increasing
  - Tends to give neurons with sparse activities

\[ g(a) = \text{relu}(a) = \max(0, a) \]

**NEURAL NETWORK**

**Topics**
- Single hidden layer neural network
  - Hidden layer pre-activation:
    \[ a(x) = b^{(1)} + W^{(1)}x \]
  - Hidden layer activation:
    \[ h(x) = g(h(x)) \]
  - Output layer activation:
    \[ f(x) = \sigma(b^{(2)} + W^{(2)T}h(x)) \]

**NEURAL NETWORK**

**Topics**
- Softmax activation function
  - For multi-class classification:
    - We need multiple outputs (1 output per class)
    - We would like to estimate the conditional probability \( p(y = c|x) \)

We use the softmax activation function at the output:

\[ o(a) = \text{softmax}(a) = \left( \frac{\exp(a_1)}{\sum \exp(a_i)}, \ldots, \frac{\exp(a_n)}{\sum \exp(a_i)} \right) \]

- Strictly positive
- Sums to one
- Predicted class is the one with highest estimated probability

**CAPACITY OF NEURAL NETWORK**

**Topics**
- Single hidden layer neural network

(from Pascal Vincent's slides)

**NEURAL NETWORK**

**Topics**
- Multilayer neural network
  - Could have \( L \) hidden layers:
    - Layer pre-activation for \( k \), \( b^{(k)}(a) = a \)
      \[ a^{(k)}(x) = b^{(k)} + W^{(k)}h^{(k-1)}(x) \]
    - Hidden layer activation (\( k \) from 1 to \( L - 1 \))
      \[ h^{(k)}(x) = g(a^{(k)}(x)) \]
    - Output layer activation (\( k = L - 1 \))
      \[ h^{(L-1)}(x) = o(a^{(L-1)}(x)) = f(x) \]
How to train a neural network?

Empirical Risk Minimization

The Learning Algorithm

Unsupervised Learning with Neural Networks

Today's Outline

Some slides are borrowed from Hugo Larochelle.
http://info.usherbrooke.ca/hlarochelle/neural_networks/content.html
Inference

- Conditional distributions: $P(h | x), P(x | h)$

- Sample distribution: $P(x)$

\[
p(h | x) = \frac{p(x, h)}{\sum_{h'} p(x, h')}
\]

\[
p(h | x) = \frac{\exp(h^T W x + c^T x + b^T h) / Z}{\sum_{h' \in \{0,1\}^p} \exp(h'^T W x + c^T x + b^T h') / Z}
\]
Inference

- Conditional distributions: $P(h|x), P(x|h)$
- Sample distribution: $P(x)$
\[ p(x) = \sum_{h \in \{0,1\}^n} \exp(h^\top W x + c^\top x + b^\top h) / Z \]

\[ p(x) = \sum_{h \in \{0,1\}^n} \exp(h^\top W x + c^\top x + b^\top h) / Z \]

\[ p(x) = \sum_{h \in \{0,1\}^n} \exp(h^\top W x + c^\top x + b^\top h) / Z \]

\[ p(x) = \sum_{h \in \{0,1\}^n} \exp(h^\top W x + c^\top x + b^\top h) / Z \]
If a "feature" $h_j$ is observed in the sample $x$, then $p(x)$ should be high.

**Training RBM**

- To train an RBM, we'd like to minimize the average negative log-likelihood (NLL)
  \[
  \frac{1}{T} \sum_{t=1}^{T} l(f(x^{(t)})) = \frac{1}{T} \sum_{t=1}^{T} -\log p(x^{(t)})
  \]
- We'd like to proceed by stochastic gradient descent
  \[
  \partial -\log p(x^{(t)}) / \partial \theta
  \]

**Contrastive Divergence (CD)**

- Idea:
  1. replace the expectation by a point estimate at $\hat{x}$
  2. obtain the point $\hat{x}$ by Gibbs sampling
  3. start sampling train at $x^{(t)}$
Each row represents a mini-batch of negative particles (samples from independent Gibbs chains). 1000 steps of Gibbs sampling were taken between each of those rows.

[http://deeplearning.net/tutorial/rbm.html]
Movie Example

- Star Wars and Lord of the Rings might have strong associations with science fiction and fantasy.

- Users who like Wall-E and Toy Story might have strong associations with a latent Pixar factor.

[example from http://blog.echen.me/2011/07/18/introduction-to-restricted-boltzmann-machines/]

Movie Example

- Alice: (Harry Potter = 1, Avatar = 1, LOTR 3 = 1, Gladiator = 0, Titanic = 0, Glitter = 0). Big SF/fantasy fan.

- Bob: (Harry Potter = 1, Avatar = 0, LOTR 3 = 1, Gladiator = 0, Titanic = 0, Glitter = 0). SF/fantasy fan, but doesn’t like Avatar.

- Carol: (Harry Potter = 1, Avatar = 1, LOTR 3 = 1, Gladiator = 0, Titanic = 0, Glitter = 0). Big SF/fantasy fan.

- David: (Harry Potter = 0, Avatar = 0, LOTR 3 = 1, Gladiator = 1, Titanic = 1, Glitter = 0). Big Oscar winners fan.

- Eric: (Harry Potter = 0, Avatar = 0, LOTR 3 = 1, Gladiator = 1, Titanic = 1, Glitter = 0). Oscar winners fan, except for Titanic.

- Fred: (Harry Potter = 0, Avatar = 0, LOTR 3 = 1, Gladiator = 1, Titanic = 1, Glitter = 0). Big Oscar winners fan.
Today's Outline

- Unsupervised learning for neural networks
  - Restricted Boltzmann machine
  - Autoencoders
  - Sparse coding
**AUTOENCODER**

**Topics:** Loss function
- For binary inputs:
  \[ l(f(x)) = -\sum_k (x_k \log(\hat{x}_k) + (1 - x_k) \log(1 - \hat{x}_k)) \]
- Cross entropy (more precisely sum of Bernoulli cross-entropies)
- For real-valued inputs:
  \[ l(f(x)) = \frac{1}{2} \sum_k (\hat{x}_k - x_k)^2 \]
- Sum of squared differences (squared Euclidean distance)
- We use a linear activation function at the output.

**Reconstruction loss**

---

**AUTOENCODER**

**Topics:** Loss function gradient
- For both cases, the gradient \( \nabla_{\theta}(l(f(x))) \) has a very simple form:
  \[ \nabla_{\theta}(l(f(x))) = \hat{x}^{(i)} - x^{(i)} \]
- Parameter gradients are obtained by backpropagating the gradient \( \nabla_{\theta}(l(f(x))) \) like in a regular network.
- **Important:** When using tied weights (\( W = W^t \)), \( \nabla_{\theta}(l(f(x))) \) is the sum of two gradients!
- This is because \( W \) is present in the encoder and in the decoder.

---

**EXAMPLE OF DATA SET: MNIST**

---

**FILTERS**

By Restricted Boltzmann Machine

---

**FILTERS (AUTOENCODER)**

(Larochelle et al., 2009)
Undercomplete v.s. Overcomplete hidden layer

**Autoencoder**

**Topics**: autoencoder, encoder, decoder, tied weights
- Feed-forward neural network trained to reproduce its input at the output layer

**Undercomplete hidden layer**

**Topics**: undercomplete representation
- Hidden layer is undercomplete if smaller than the input layer
  - Hidden layer “compresses” the input
  - will compress well only for the training distribution
- Hidden units will be good features for the training distribution
  - but bad for other types of input

**Overcomplete hidden layer**

**Topics**: overcomplete representation
- Hidden layer is overcomplete if greater than the input layer
  - no compression in hidden layer
  - Each hidden unit could copy a different input component
  - No guarantee that the hidden units will extract meaningful structure

**Denoising autoencoder**

**Topics**: denoising autoencoder
- Idea: representation should be robust to introduction of noise
  - Random assignment of subset of inputs to 0, with probability p
  - Gaussian additive noise
- Reconstruction \( \hat{x} \) computed from the corrupted input \( \tilde{x} \)
  - Loss function compares \( \hat{x} \) reconstruction with the noiseless input \( x \)
DENOISING AUTOENCODER

**Topics:**
- Denoising autoencoder
- Idea: representation should be robust to introduction of noise:
  - Random assignment of subset of inputs to 0, with probability \( p \)
  - Gaussian additive noise
- Reconstruction \( \hat{x} \) computed from the corrupted input \( x \)
- Loss function compares \( \hat{x} \) reconstruction with the noiseless input \( x \)

\[
\hat{x} = \text{sign}(e + W^T h(\tilde{x}))
\]
Today’s Outline

• Unsupervised learning for neural networks
  – Restricted Boltzmann machine
  – Autoencoders
  – Sparse coding

Some slides are borrowed from Hugo Larochelle.
http://info.usherbrooke.ca/larochelle/neural_networks/content.html
SPARSE CODING

Topics: sparse coding
- For each $x^{(i)}$ find a latent representation $h^{(i)}$ such that:
  - it is sparse: the vector $h^{(i)}$ has many zeros
  - we can reconstruct the original input $x^{(i)}$ as well as possible
- More formally:
  $$\min_{w} \frac{1}{2} \sum_{x \in X} \|x(Dw) - x\|^2 + \lambda \|h\|^2$$

- we also constrain the columns of $D$ to be of norm 1
- otherwise $D$ could grow big while $h^{(i)}$ becomes small to satisfy the prior
- sometimes the columns are constrained to be no greater than 1

Why sparse coding?

Topics: dictionary
- Can also write $\tilde{x}^{(i)} = D h^{(i)} = \sum_{k \in T} D_{ik} h^{(i)k}$

$$\begin{align*}
z &= 1 \quad + \quad 1 \quad - \quad 1 \quad - \quad 1 \\
&= 1 \quad + \quad 1 \quad + \quad 0.8 \quad + \quad 0.8
\end{align*}$$

- we also refer to $D$ as the dictionary
- in certain applications, we know what dictionary matrix to use
- often however, we have to learn it

How to get $D$ and $h$

Topics: dictionary
- Can also write $\tilde{x}^{(i)} = D h^{(i)} = \sum_{k \in T} D_{ik} h^{(i)k}$

$$\begin{align*}
z &= 1 \quad + \quad 1 \quad - \quad 1 \quad - \quad 1 \\
&= 1 \quad + \quad 1 \quad + \quad 0.8 \quad + \quad 0.8
\end{align*}$$

- we also refer to $D$ as the dictionary
- in certain applications, we know what dictionary matrix to use
- often however, we have to learn it

- $D$ is equivalent to the autoencoder output weight matrix
- however, $h^{(i)}$ is now a complicated function of $x^{(i)}$
Fix D, compute h

Topics: inference of sparse codes
- Given D, how do we compute h(x(0))
  - we want to optimize f(x(0)) = \frac{1}{2} ||x(0) - D h(0)||_2^2 + \lambda ||h(0)||_1
  - we could use a gradient descent method:
    \nabla_{h(0)} f(x(0)) = D^T (D h(0) - x(0)) + \lambda \text{sign}(h(0))

Topics: inference of sparse codes
- For a single hidden unit:
  \frac{\partial}{\partial h_i^{(0)}} f(x(0)) = (D_{i, :})^T (D h(0) - x(0)) + \lambda \text{sign}(h_i^{(0)})
  - issue L1 norm not differentiable at 0
  - very unlikely for gradient descent to "land" on h_i^{(0)} = 0 (even if it's the solution)
  - solution if h_i^{(0)} changes sign because of L1 norm gradient, clamp to 0
  - each hidden unit update would be performed as follows:
    - h_i^{(0)} \leftarrow h_i^{(0)} - \alpha (D_{i, :})^T (D h(0) - x(0))
    - if sign(h_i^{(0)}) ≠ sign(h_i^{(0)} - \alpha \lambda \text{sign}(h_i^{(0)})) then h_i^{(0)} \leftarrow 0
      update from reconstruction
    - else: h_i^{(0)} \leftarrow h_i^{(0)} - \alpha \lambda \text{sign}(h_i^{(0)})
      update from sparsity

Topics: ISTA (Iterative Shrinkage and Thresholding Algorithm)
- This process corresponds to the ISTA algorithm:
  - initialize h(0) (for instance to 0)
  - while h(0) has not converged
    - h(0) \leftarrow h(0) - \alpha D^T (D h(0) - x(0))
    - h(0) \leftarrow \text{shrink}(h(0), \alpha \lambda)
  - return h(0)

  Here shrink(a, b) = [\ldots, \text{sign}(a_i) \max(|a_i| - b_i, 0), \ldots]

  Will converge if \frac{\lambda}{\alpha} is bigger than the largest eigenvalue of D^T D

How to get D and h

Topics: sparse coding
- For each x(0) find a latent representation h(0) such that:
  - it is sparse; the vector h(0) has many zeros
  - we can reconstruct the original input x(0) as much as possible

  More formally:

  reconstruction error
  \sum_i \frac{1}{2} ||x(i) - D h(i)||_2^2 + \lambda ||h(i)||_1

  D is equivalent to the autoencoder output weight matrix
  - however, h(x(0)) is now a complicated function of x(0)
    - encoder is the minimization h(x(0)) = \text{argmin} ||x(0) - D h(i)||_2^2 + \lambda ||h(i)||_1
Fix $h$, compute $D$

**Topics:** dictionary update (algorithm 2)
- Going back to our original problem
- Let’s assume $h(x^{(l)})$ doesn’t depend on $D$ (which is fine)
  - we must minimize
    \[
    \min_{D} \frac{1}{2} \sum_{l=1}^{L} \|x^{(l)} - D h(x^{(l)})\|_2^2
    \]
  - we must also constrain the columns of $D$ to be of unit norm

An alternative is to solve for each column $D_{j}$ in cycle:
- we can rewrite
  \[
  D_{j} = \frac{1}{\sum_{l=1}^{L} |h(x^{(l)})|^2} \left( \sum_{l=1}^{L} h(x^{(l)}) x^{(l)T} \right) h(x^{(l)})
  \]
- this way, we only need to store:
  - $A \leftarrow \sum_{l=1}^{L} h(x^{(l)}) h(x^{(l)})^T$
  - $B \leftarrow \sum_{l=1}^{L} x^{(l)} h(x^{(l)})^T$

**The full learning algorithm**

**Topics:** learning algorithm (putting it all together)
- While $D$ has not converged
  - Find the sparse codes $h(x^{(l)})$ for all $x^{(l)}$ in my training set with ISTA
  - update the dictionary:
    - $A \leftarrow \sum_{l=1}^{L} h(x^{(l)}) h(x^{(l)})^T$
    - $B \leftarrow \sum_{l=1}^{L} x^{(l)} h(x^{(l)})^T$
    - run block-coordinate descent algorithm to update $D$

An alternative is to solve for each column $D_{j}$ in cycle:
- setting the gradient for $D_{j}$ to zero, we have
  \[
  0 = \frac{1}{2} \sum_{l=1}^{L} (x^{(l)} - D h(x^{(l)}) h(x^{(l)})^T)
  \]
- $D_{j} = \frac{1}{\sum_{l=1}^{L} |h(x^{(l)})|^2} \left( \sum_{l=1}^{L} h(x^{(l)}) x^{(l)T} \right) h(x^{(l)})$

While $D$ hasn’t converged
- for each column $D_{j}$ perform updates
  - $D_{j} \leftarrow \frac{1}{A_{j}} (B_{j} - D A_{j} + D_{j} A_{j})$
  - $D_{j} \leftarrow \frac{D_{j}}{\|D_{j}\|_2}$

This is referred to as a block-coordinate descent algorithm
- a different block of variables are updated at each step
- the “blocks” are the columns $D_{j}$

Use sparse coding to extract features
Use sparse coding to extract features

Topics: feature learning
- A sparse coding model can be used to extract features
  - given a labeled training set \( \{(x^{(i)}, y^{(i)})\} \)
  - train sparse coding dictionary only on training inputs \( \{x^{(i)}\} \)
    - this yields a dictionary \( D \) from which to infer sparse codes \( h(x^{(i)}) \)
  - train a classifier on transformed training set \( \{(h(x^{(i)}), y^{(i)})\} \)
- When classifying test input \( x \), must infer its sparse representation \( h(x) \) first, then feed it to the classifier.

What we learned today
- Unsupervised learning for neural networks
  - Restricted Boltzmann machine
  - Autoencoders
  - Sparse coding

Homework
- Slides from Hugo Larochelle.
  - [http://info.usherbrooke.ca/hlarochelle/neural_networks/content.html](http://info.usherbrooke.ca/hlarochelle/neural_networks/content.html)
- Deep Learning, by Ian Goodfellow, Yoshua Bengio, and Aaron Courville
  - [http://www.deeplearningbook.org/](http://www.deeplearningbook.org/)
  - CH 20.2, CH 14, CH 13.4