What we learned last time

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

  \[ g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a)-1}{\exp(2a)+1} \]

**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: \( a^{(l)} = W^{(l)}x + b^{(l)} \)
  - Hidden layer activation: \( h^{(l)}(x) = g(a^{(l)}) \)
  - Output layer activation: \( f(x) = a^{(2)} = \text{softmax}(a^{(2)}) = \frac{\exp(a^{(2)}_1)}{\sum_{k=1}^{K} \exp(a^{(2)}_k)} \) (strictly positive, sums to one, predicted class is the one with highest estimated probability)

**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:

\[ a^{(2)} = \text{softmax}(a^{(2)}) = \frac{\exp(a^{(2)}_1)}{\sum_{k=1}^{K} \exp(a^{(2)}_k)} \]

**CAPACITY OF NEURAL NETWORK**

- **Topics**: single hidden layer neural network

**NEURAL NETWORK**

- **Topics**: multilayer neural network
  - Could have \( L \) hidden layers:
    - layer pre-activation for \( l = 1 \) to \( L \):
      \[ a^{(l+1)}(x) = W^{(l+1)}h^{(l)}(x) + b^{(l+1)} \]
    - hidden layer activation:
      \[ h^{(l)}(x) = g(a^{(l)}(x)) \]
    - output layer activation:
      \[ f(x) = a^{(L+1)}(x) = o(a^{(L+1)}(x)) = f(x) \]

(from Pascal Vincent’s slides)
How to train a neural network?

Empirical Risk Minimization

The Learning Algorithm

Unsupervised Learning with Neural Networks

Restrained Boltzmann Machine
Inference

- Conditional distributions: \( P(h|x), P(x|h) \)
- Sample distribution: \( P(x) \)

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

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

\[
p(h|x) = \frac{p(x, h)}{\sum_{h'} p(x, h')}
= \frac{\exp(h^T W x + e^T x + b^T h) / Z}{\sum_{h' \in \{0,1\}^n} \exp(h'^T W x + e^T x + b^T h') / Z}
- \exp(\sum_{j \in \{0,1\}} h_j W_j x + b_j h_j) / Z
\]
Inference

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

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

\[ p(x) = \exp(c^T x) \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_v \in \{0,1\}} \exp \left( \sum_j h_j W_j x + b_j h_j \right)/Z \]

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

\[ p(x) = \exp(c^T x) \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_v \in \{0,1\}} \exp \left( \sum_j h_j W_j x + b_j h_j \right)/Z \]

\[ p(x) = \exp(c^T x) \left(1 + \exp(h + W x)\right)^d \cdots \left(1 + \exp(h + W x)\right)/Z \]

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

\[ p(x) = \exp(c^T x) \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_v \in \{0,1\}} \exp \left( \sum_j h_j W_j x + b_j h_j \right)/Z \]

\[ p(x) = \exp(c^T x) \left(1 + \exp(h + W x)\right)^d \cdots \left(1 + \exp(h + W x)\right)/Z \]

\[ p(x) = \exp \left( c^T x + \sum_{j=1}^d \log \left(1 + \exp(h_j + W_j x)\right) \right)/Z \]
Today’s Outline

- Unsupervised learning for neural networks
  - Restricted Boltzmann machine (continue)
  - Autoencoders
  - Sparse coding

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

Training RBM

Topics: training objective

- To train an RBM, we’d like to minimize the average negative log-likelihood (NLL)
  \[
  \frac{1}{T} \sum_{t} l(f(x^{(t)})) = \frac{1}{T} \sum_{t} \log p(x^{(t)})
  \]

- We’d like to proceed by stochastic gradient descent
  \[
  \frac{\partial - \log p(x^{(0)})}{\partial \theta} = \text{Ex}[\frac{\partial E(x^{(0)}, h)}{\partial \theta}] x^{(0)} - \text{Ex}[\frac{\partial E(x, h)}{\partial \theta}]
  \]

CONTRASTIVE DIVERGENCE (CD)

(HINTON, NEURAL COMPUTATION, 2002)

Topics: contrastive divergence, negative sample

- Idea:
  1. replace the expectation by a point estimate at \( \hat{x} \)
  2. obtain the point \( \hat{x} \) by Gibbs sampling
  3. start sampling chain at \( \hat{x}^{(t)} \)

\[
\text{~}\begin{array}{c}
\hat{x}^{(0)} \\
\hat{x}^{(1)} \\
\hat{x}^{(T)} = \hat{x}
\end{array}
\]

- if a "feature" (h) is observed in the sample x, then p(x) should be high.
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/istm.html]
DERIVATION OF THE LEARNING RULE

**Topics:** contrastive divergence

- Derivation of $\frac{\partial E(x, h)}{\partial W_{jk}}$ for $\theta = W_{jk}$

\[
\frac{\partial E(x, h)}{\partial W_{jk}} = \frac{\partial}{\partial W_{jk}} \left( \sum_{t} W_{jk} h_{j} r_{k} - \sum_{k} z_{k} r_{k} - \sum_{j} b_{j} h_{j} \right) = \frac{\partial}{\partial W_{jk}} \left( W_{jk} h_{j} r_{k} \right) = h_{j} r_{k}
\]

\[h_{j} r_{k} = -h \cdot x^T \]

---

**Topics:** contrastive divergence

- Derivation of $E_{h} \left[ \frac{\partial E(x, h)}{\partial W_{jk}} \right]$ for $\theta = W_{jk}$

\[E_{h} \left[ \frac{\partial E(x, h)}{\partial W_{jk}} \right] = E_{h} [-h_{j} r_{k}] = -h_{j} r_{k} \]

\[h(x) \text{ def } \frac{e^{\langle x, h \rangle}}{1 + e^{\langle x, h \rangle}} \]

\[E_{h} [h(x)] = \text{ sign}(h + W x) \]

---

**Topics:** contrastive divergence

- Given $x^{(t)}$ and $\bar{x}$, the learning rule for $\theta = W$ becomes

\[W \leftarrow W - \alpha \left( W_{W} - \log P(\bar{x}^{(t)}) \right) \]

\[W \leftarrow W - \alpha \left( E_{h} \left[ \frac{\partial E(x, h)}{\partial W_{jk}} \right] \right) \]

\[W \leftarrow W - \alpha \left( E_{h} \left[ \frac{\partial E(x, h)}{\partial W_{jk}} \right] \right) \]

\[W \leftarrow W + \alpha \left( h(x^{(t)}) x^{(t)} - h(\bar{x}) \bar{x} \right) \]

---

CD-K: PSEUDOCODE

**Topics:** contrastive divergence

1. For each training example $x^{(t)}$
   - generate a negative sample $\bar{x}$ using $k$ steps of Gibbs sampling starting at $x^{(t)}$
   - update parameters
     - $W \leftarrow W + \alpha \left( h(x^{(t)}) x^{(t)} - h(\bar{x}) \bar{x} \right)$
     - $b \leftarrow b + \alpha \left( h(x^{(t)}) - h(\bar{x}) \right)$
     - $e \leftarrow e + \alpha \left( x^{(t)} - \bar{x} \right)$
   - Go back to 1 until stopping criteria

---

**Topics:** persistent contrastive divergence

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

---

PERSISTENT CD (PCD)

**Topics:** persistent contrastive divergence

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

---

10
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/)
Today’s Outline

- Unsupervised learning for neural networks
  - Restricted Boltzmann machine (continue)
  - Autoencoders
  - Sparse coding

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

**EXAMPLE OF DATA SET: MNIST**

By Restricted Boltzmann Machine

**FILTERS**

By Restricted Boltzmann Machine

**FILTERS (AUTOENCODER)**
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

\[ \hat{x} = o(h(x)) \]
\[ = \text{sigmoid}(c + W^T h(x)) \]
\[ \text{for binary inputs} \]

\[ h(x) = g(o(x)) \]
\[ = \text{sigmoid}(b + Wx) \]

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 \( \tilde{x} \) computed from the corrupted input \( \hat{x} \)
- Loss function compares \( \tilde{x} \) 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 \( x \)
- Noisy function computing \( \hat{x} \) reconstruction with the noiseless input \( x \)

**Formulas:**

\[
x = \text{sign}(e + W^T h(x)) \quad \text{and} \quad p(\tilde{x}|x)
\]

**Filters (Denoising Autoencoder)**

(Vincent, Larochelle, Bengio and Manzagol, ICML 2008)

- No corrupted inputs (cross-entropy loss)
- 25% corrupted inputs
Today’s Outline

- Unsupervised learning for neural networks
  - Restricted Boltzmann machine (continue)
  - Autoencoders
  - Sparse coding

Some slides are borrowed from Hugo Larochelle.
http://info.usherbrooke.ca/larochelle/neural_networks/content.html
Why sparse coding?

- Can also write $\tilde{x}^{(t)} = D \mathbf{h}(x^{(t)}) = \sum_{k \in \mathbf{D} \setminus \mathbf{h}(x^{(t)})} D_k h(x^{(t)})$.

$$z = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

- We also refer to $\mathbf{D}$ as the dictionary.
  - In certain applications, we know what dictionary matrix to use.
  - Often however we have to learn it.
**SPARSE CODING**

**Topics:** dictionary
- Can also write \( \hat{x}^{(0)} = D h^{(0)} = \sum_{k} h_k^{(0)} D_k \)

\[ \hat{x}^{(0)} = \sum_{k} h_k^{(0)} D_k \]

- 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:** sparse coding
- For each \( x^{(0)} \) find a latent representation \( h^{(0)} \) such that:
  - \( D \) is sparse; the vector \( h^{(0)} \) has many zeros.
  - We can reconstruct the original input \( x^{(0)} \) as much as possible.
- More formally:
  \[ \min_{h^{(0)}} \frac{1}{2} || x^{(0)} - D h^{(0)} ||_2^2 + \lambda \sum_{k} h_k^{(0)} \]

\[ \text{reconstruction error} \quad \text{sparsity penalty} \]

\( D \) is equivalent to the autoencoder output weight matrix.
- However, \( h^{(0)} \) is now a complicated function of \( x^{(0)} \).
- Encoder is the minimization \( h^{(0)} = \arg \min_{h^{(0)}} \frac{1}{2} || x^{(0)} - D h^{(0)} ||_2^2 + \lambda \sum_{k} h_k^{(0)} \).

**Fix \( D \), compute \( h \)**

**Topics:** inference of sparse codes
- Given \( D \), how do we compute \( h^{(0)} \)?
- We want to optimize \( f(x^{(0)}) = \frac{1}{2} || x^{(0)} - D h^{(0)} ||_2^2 + \lambda || h^{(0)} ||_1 \) wrt. \( h^{(0)} \)

\[ h^{(0)} = \text{arg min}_{h^{(0)}} f(x^{(0)}) \]

- We could use a gradient descent method:
  \[ \nabla_h f(x^{(0)}) = D^T (D h^{(0)} - x^{(0)}) + \lambda \text{sign}(h^{(0)}) \]

**Topics:** inference of sparse codes
- For a single hidden unit:
  \[ \frac{\partial}{\partial h_k^{(0)}} f(x^{(0)}) = (D^T A)_{k} (D h^{(0)} - x^{(0)}) + \lambda \text{sign}(h_k^{(0)}) \]

- Issues: L1 norm not differentiable at 0.
  - Very unlikely for gradient descent to “land” on 0 (even if it’s the solution).
  - Solution: if \( h_k^{(0)} \) changes sign because of L1 norm gradient, clamp to 0.
- Each hidden unit update would be performed as follows:
  - \( h_k^{(0)} \leftarrow h_k^{(0)} - \alpha \text{sign}(h_k^{(0)}) (D h^{(0)} - x^{(0)}) \)
  - If \( \text{sign}(h_k^{(0)}) \neq \text{sign}(h_k^{(0)} - \alpha \text{sign}(h_k^{(0)})) \) then \( h_k^{(0)} \leftarrow 0 \)
  - Else: \( h_k^{(0)} \leftarrow h_k^{(0)} - \alpha \lambda \text{sign}(h_k^{(0)}) \)

**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) = \left[ \ldots, \text{sign}(a_i), \max\{ |a_i| - b, 0 | \} \ldots \right] \)**
- Will converge if \( \frac{1}{\alpha \lambda} \) is bigger than the largest eigenvalue of \( D^T D \)
How to get $D$ and $h$

**Topics:** sparse coding

- For each $x^{(t)}$, find a latent representation $h^{(t)}$ such that:
  - it is sparse: the vector $h^{(t)}$ has many zeros
  - we can reconstruct the original input $X^{(t)}$ as much as possible
- More formally:
  \[
  \min_{h} \frac{1}{2} \| \sum_{t} h^{(t)} - D h^{(t)} \|_2^2 + \lambda \| h^{(t)} \|_1
  \]
- $D$ is equivalent to the autoencoder output weight matrix
- however, $h^{(t)}$ is now a complicated function of $X^{(0)}$
- encoder is the minimization $h^{(t)} = \arg\min_{h} \frac{1}{2} \| \sum_{t} h^{(t)} - D h^{(t)} \|_2^2 + \lambda \| h^{(t)} \|_1$

**Fix $h$, compute $D$**

**Topics:** dictionary $h$, compute $D$

- Going back to our original problem
  \[
  \min_{h} \frac{1}{2} \| X - D h^{(t)} \|_2^2 + \lambda \| h^{(t)} \|_1
  \]
- Let's assume $h^{(t)}$ doesn't depend on $D$ (which is fine)
  - we must minimize
    \[
    \min_{D} \frac{1}{2} \| \sum_{t} h^{(t)} - D h^{(t)} \|_2^2
    \]
  - we must also constrain the columns of $D$ to be of unit norm

**An alternative is to solve for each column $D_{ij}$ in cycle:**

- setting the gradient for $D_{ij}$ to zero, we have
  \[
  0 = \frac{1}{T} \sum_{t} (x^{(t)} - \sum_{j} D_{ij} h^{(t)}_{ij}) \cdot h^{(t)}_{ij}
  \]
  \[
  0 = \frac{1}{T} \sum_{t} \sum_{j} (x^{(t)} - \sum_{j} D_{ij} h^{(t)}_{ij}) \cdot (x^{(t)} - \sum_{j} D_{ij} h^{(t)}_{ij})
  \]
  \[
  D_{ij} = \frac{1}{\sum_{t} h^{(t)}_{ij}} \sum_{t} (x^{(t)} - \sum_{j} D_{ij} h^{(t)}_{ij}) \cdot h^{(t)}_{ij}
  \]
- we don't need to specify a learning rate to update $D_{ij}$

**While $D$ hasn't converged**

- for each column $D_{ij}$ perform updates
  \[
  D_{ij} \leftarrow \frac{1}{A_{ij}} (B_{ij} - D A_{ij} + D_{ij} A_{ij})
  \]
  \[
  D_{ij} \leftarrow \frac{D_{ij}}{||D_{ij}||_2}
  \]
- 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 $D_{ij}$
The full learning algorithm

Topics: learning algorithm (putting it all together)
• Learning alternates between inference and dictionary learning

• While D has not converged
  • Find the sparse codes $\mathbf{h}(\mathbf{x}^{(t)})$ for all $\mathbf{x}^{(t)}$ in my training set with ISTA
  • Update the dictionary:
    - $\mathbf{A} = \sum_{t=1}^{T} \mathbf{x}^{(t)} \mathbf{h}(\mathbf{x}^{(t)})^T$
    - $\mathbf{D} = \sum_{t=1}^{T} \mathbf{h}(\mathbf{x}^{(t)}) \mathbf{h}(\mathbf{x}^{(t)})^T$
    - run block-coordinate descent algorithm to update $\mathbf{D}$

Use sparse coding to extract features

Topics: feature learning
• A sparse coding model can be used to extract features
  • Given a labeled training set $\{(\mathbf{x}^{(t)}, y^{(t)})\}$
  • Train sparse coding dictionary only on training inputs $\{\mathbf{x}^{(t)}\}$
    - This yields a dictionary $\mathbf{D}$ from which to infer sparse codes $\mathbf{h}(\mathbf{x}^{(t)})$
  • Train favorite classifier on transformed training set $\{\mathbf{h}(\mathbf{x}^{(t)}), y^{(t)}\}$

• When classifying test input $\mathbf{x}$, must infer its sparse representation $\mathbf{h}(\mathbf{x})$ first, then feed it to the classifier

What we learned today

• Unsupervised learning for neural networks
  – Restricted Boltzmann machine (continue)
  – Autoencoders
  – Sparse coding
Homework

• Read slides from Hugo Larochelle.
  – http://info.usherbrooke.ca/hlarochelle/neural_networks/content.html