Lecture 21

Associative Memory, Hopfield Networks, and Boltzmann Machines
CS542—Contents 21

- Associative Memory and Recurrent Networks
  - Hopfield Networks
    - Deterministic
    - Stochastic
  - Boltzmann Machines

- Handout:
  - Class Notes

- Reading Assignment for Next Class
  - Class Handouts
Associative Memory

- The Associative Memory Problem
  - Store a set of $k$ patterns $x_k$ in such a way that when presented with a new pattern, the network responds by producing whichever one of the stored patterns resembles the new one the most.

- How to do this?
  - Nearest neighbors
    - But how to deal with hidden values?
    - Insufficient use of resources (low compression)
  - The problem of distance metrics
  - Attractor networks
    - Hope: just give the system all the information about the pattern to be recognized, and it will automatically complete it by means of a dynamic (recurrent) process.
Deterministic Hopfield Networks

- Assumptions:
  - Only binary states (+1/-1)
  - Symmetric weights $w_{ij} = w_{ji}$
  - Each unit is a visible unit
  - Activation rule:

  \[
  x_i = g\left(\sum_{j \in X_i} w_{ij} x_j\right) \quad \text{where} \quad g(a) = \begin{cases} 
  1 & \text{if } x \geq 0 \\
  -1 & \text{if } x < 0 
  \end{cases}
  \]

  (Note: bias term can be included)

- How to update the weights such that the pattern can be stored?
Deterministic Hopfield Nets: Computing the Weights

- One pattern
  - Condition:
  \[ x_i = g\left(\sum_{j \in X_i} w_{ij} x_j\right) = t_i \]
  - Solution
  \[ w_{ij} \propto t_i t_j, \text{ for instance } w_{ij} = \frac{1}{k} t_i t_j \]
  where \( k \) is the number of units in the network

- How to run the network?
  - Initialize the units to pattern to be recognized
  - Synchronously or asynchronously update the units until convergence
Deterministic Hopfield Nets: Computing the Weights

- Multiple patterns
  - Condition:
    \[ x_i = g\left(\sum_{j \in X_i} w_{ij} x_j\right) = t_i \]
  - Solution:
    \[ w_{ij} \propto \sum_{n=1}^{N} t_i^n t_j^n, \text{ for instance } w_{ij} = \frac{1}{k} \sum_{n=1}^{N} t_i^n t_j^n \]
    where \( k \) is the number of units in the network
Deterministic Hopfield Nets: Computing the Weights

- Example
How Many Patterns Can be Stored?

- **Stability of a Pattern:**

  \[ t_i = g\left( \sum_j w_{ij}t_j^m \right) = g\left( \sum_j \left( \frac{1}{N} \sum_{n=1}^{N} t_i^n t_j^n \right) t_j^m \right) \]

  \[ = g\left( \frac{1}{N} \sum_j \sum_{n=1}^{N} t_i^n t_j^n \right) = g\left( t_i^m + \frac{1}{N} \sum_j \sum_{n \neq m} t_i^n t_j^n \right) \]

  Original Term

  One-pattern Term

  Cross-Talk Term

  As long as abs(cross talk) \(< 1\), pattern will be stable!

- **Storage Capacity:**

  Upper Bound: \( p_{\text{max}} < 0.138k \)

  Bound depend on the amount of misclassification that is permitted, and on assumptions about the correlation of the target patterns.
The Energy of a Network State

- Energy of a the Hopfield Net (Hopfield 1982)
  \[ H = -\frac{1}{2} \sum_{i \neq j} w_{ij} x_i x_j \]

- Lyapunov Stability
  - If the energy of a system always decreases over time, the system must end up in a stable state
  - Stored patterns are thus minima in the energy landscape
  - The problem of proving stability: is it possible to find an appropriate energy function? This is a piece of art!
Energy of a Hopfield Network

- Stability Proof

Let: \( \tilde{x}_k = g(\sum_j w_{kj} x_j) \) be the updated value of one unit

The energy only changes if \( \tilde{x}_k \neq x_k \)

\[
\tilde{H} - H = -\frac{1}{2} \sum_{i \neq j} w_{ij} \tilde{x}_i x_j - \frac{1}{2} \sum_{i \neq j} w_{ij} x_i x_j
\]

\[
= -\frac{1}{2} \sum_{i \neq j \text{ and } i,j \neq k} w_{ij} x_i x_j - \frac{1}{2} \sum_j w_{kj} \tilde{x}_k x_j - \frac{1}{2} \sum_j w_{ik} x_i \tilde{x}_k - \frac{1}{2} \sum_{i \neq j} w_{ij} x_i x_j
\]

\[
= 2 \sum_j w_{kj} x_k x_j \quad (\text{here we used the fact that } \tilde{x}_i = -x_i)
\]

\[
= 2 x_k \sum_j w_{kj} x_j = 2 x_k \sum_j w_{kj} x_j = 2 x_k \tilde{x}_k c \quad (\text{where } c \text{ is a constant } > 0)
\]

This expression is < 0!
Instead of deterministic updating of the units, they are updated probabilistically:

\[ p(x_i = 1) = \frac{1}{1 + \exp(-\beta \sum_j x_j w_{ij})} \]

+ \( \beta \) plays the role of an inverse temperature.

How to perform an update?
- Pick a state \( x_i \)
- Calculate \( p(x_i) \)
- Generate the state \( x_i \) according to this probability

Why stochastic updating?
- The Hopfield net has spurious minima in the energy landscapes (ghost memory), and the stochasticity will push the network out of these local minima.

When the network is stochastic, what is the energy distribution over the (complete) network states?
Parallel to Physics: The Boltzmann Distribution

- Consider a physical system with a set of states $\alpha$, each of which has a total energy $H(\alpha)$ (the energy of the entire macro state)
- If the system is at absolute temperature $T>0$, the state (and thus the energy) will fluctuate over time, i.e., it becomes a random variable
- At thermal equilibrium, the probability distribution over the system’s states is:

$$P_\alpha = \frac{1}{Z} \exp\left(-\frac{H_\alpha}{k_b T}\right)$$

Boltzmann-Gibbs Distribution

$$Z = \sum_{\alpha} \exp\left(-\frac{H_\alpha}{k_b T}\right)$$
(called the Partition Function)
Parallels to Physics: The Helmholtz Free Energy

- Consider a physical system with a set of states \( \alpha \), each of which has a total energy \( H(\alpha) \) (the energy of the entire macro state).
- If the system is at absolute temperature \( T>0 \), the state (and thus the energy) will fluctuate over time, i.e., it becomes a random variable.
- The Helmholtz Free Energy of a system is defined as:
  \[
  F = \sum_{\alpha} P(\alpha) H(\alpha) + T \sum_{\alpha} P(\alpha) \ln P(\alpha) = \overline{H} - TS
  \]
  \[
  = \text{Average Energy - Entropy}
  \]
- Note: The probability distribution minimizing \( F \) is the Boltzmann distribution.
- Dynamic systems with stochastic updating minimize \( F \).
Boltzmann Machines

- Boltzmann machines are stochastic Hopfield nets with hidden units
- Boltzmann machines were the first to introduce hidden units
- Boltzmann machines can perform supervised and unsupervised learning by directly modeling the probability distributions of the data vectors
- Boltzmann machines can model more complex statistical dependencies between input (and output) data
Boltzmann Machine Learning

- Cost function: Kullback-Leibler distance (or cross-entropy), i.e., the network tries to model the probability distribution of the given data:

\[ J = - \sum Q_\alpha \ln \frac{P_\alpha}{Q_\alpha} = \sum Q_\alpha \ln \frac{Q_\alpha}{P_\alpha} \]

where \( Q_\alpha \) is the distribution of the target data, and \( P_\alpha \) is the distribution generated by the network at the "visible" units.

- Minimization by gradient descent

\[ \Delta w_{ij} = -\eta \frac{\partial J}{\partial w_{ij}} = \eta \sum Q_\alpha \frac{\partial P_\alpha}{P_\alpha} \frac{\partial P_\alpha}{\partial w_{ij}} \]
Boltzmann Machine Learning

- Taking the derivative

Notation: $\alpha$ denotes the visible states
$\lambda$ denotes the hidden states

$$P_{\alpha\lambda} = \frac{\exp(-\beta H_{\alpha\lambda})}{Z}$$
where
$$H_{\alpha\lambda} = -\frac{1}{2} \sum_{ij} w_{ij} x_i x_j$$
and
$$Z = \sum_{\alpha\lambda} \exp(-\beta H_{\alpha\lambda})$$

$$P_\alpha = \sum_\lambda P_{\alpha\lambda} = \sum_\lambda \frac{\exp(-\beta H_{\alpha\lambda})}{Z} = \frac{\exp\left(\frac{\beta}{2} \sum_{ij} w_{ij} x_i x_j\right)}{Z}$$

$$\frac{\partial P_\alpha}{\partial w_{ij}} = \frac{\beta}{2} \frac{\sum_\lambda \exp(-\beta H_{\alpha\lambda}) x_i x_j}{Z} - \frac{\left(\sum_\lambda \exp(-\beta H_{\alpha\lambda})\right)\left(\frac{\beta}{2} \sum_{\alpha\lambda} \exp(-\beta H_{\alpha\lambda}) x_i x_j\right)}{Z^2}$$

$$= \frac{\beta}{2} \left(\sum_\lambda P_{\alpha\lambda} x_i x_j - P_\alpha \sum_{\alpha\lambda} P_{\alpha\lambda} x_i x_j\right)$$

$$= \frac{\beta}{2} \left(\sum_\lambda P_{\alpha\lambda} x_i x_j - P_\alpha E\{x_i x_j\}\right)$$
Boltzmann Machine Learning

- Weight update:

\[ \Delta w_{ij} = -\eta \frac{\partial J}{\partial w_{ij}} = \eta \sum_{\alpha} \frac{Q_{\alpha}}{P_{\alpha}} \frac{\partial P_{\alpha}}{\partial w_{ij}} \]

\[ = \eta \frac{\beta}{2} \sum_{\alpha} \frac{Q_{\alpha}}{P_{\alpha}} \left( \sum_{\lambda} P_{\alpha \lambda} x_i x_j - P_{\alpha} E\{x_i x_j\} \right) \]

\[ = \eta \frac{\beta}{2} \sum_{\alpha} \left( \sum_{\lambda} \frac{Q_{\alpha}}{P_{\alpha}} P_{\alpha \lambda} x_i x_j - Q_{\alpha} E\{x_i x_j\} \right) \]

\[ = \eta \frac{\beta}{2} \sum_{\alpha} \left( \sum_{\lambda} Q_{\alpha} P_{\lambda \alpha} x_i x_j - Q_{\alpha} E\{x_i x_j\} \right) \]

\[ = \eta \frac{\beta}{2} \left( \sum_{\alpha \lambda} Q_{\alpha} P_{\lambda \alpha} x_i x_j - \sum_{\alpha} Q_{\alpha} E\{x_i x_j\} \right) \]

\[ = \eta \frac{\beta}{2} \left( \sum_{\alpha \lambda} Q_{\alpha} P_{\lambda \alpha} x_i x_j - E\{x_i x_j\} \right) \]

\[ = \eta \frac{\beta}{2} \left( E\{x_i x_j\}_{\text{clamped}} - E\{x_i x_j\}_{\text{free}} \right) \]
Boltzmann Machine Learning

- The Boltzmann Machine Learning Algorithm:
  - Initialize network with random weights
  - Clamped phase:
    - For all training patterns:
      - Fix the visible units to the training pattern
      - Let system run; anneal temperature from high to low
      - After system is in thermal equilibrium, compute average $x_i x_j$ over a time window
        - The average over all averages for each pattern is the clamped expectation
  - Free phase:
    - Randomly initialize all system’s states at high temperature
    - Let system run; anneal temperature from high to low
    - After system is in thermal equilibrium, compute average $x_i x_j$ over a time window
      - The average over all averages of the random trials is the free expectation
  - Update weights, and run the entire algorithm until convergence
    - Note that this is awfully slow – some speed-ups exist
Summary

- Associative memory is addressed as recurrent networks.
- Hopfield networks are simple (no-hidden state) associative memory networks, easy to train.
- Boltzmann machines are complex associative memory system, essentially modeling probability distributions. While elegant, training is done by sampling and very slow.