CMSC 475/675: Introduction to Neural Networks

Review for Exam 2 (Chapters 5, 6, 7)

1. Competitive Learning Networks (CLN)

- Purpose: self-organizing to form pattern clusters/classes based on similarities.
- Architecture: competitive output nodes (WTA, Mexican hat, Maxnet, Hamming nets)
  - external judge
  - lateral inhibition (explicit and implicit)
- Simple competitive learning (unsupervised)
  - both training examples (samples) and weight vectors are normalized.
  - two phase process (competition phase and reward phase)
  - learning rules (moving **winner's** weight vector toward input training vector)
    $Dw_j = \alpha(i_l - w_j)$ or $Dw_j = \alpha \cdot i_l$ where $\boldsymbol{i_l}$ is the current input vector and $\boldsymbol{w_j}$ is the winner's weight vector
  - learning algorithm
  - w_j is trained to represent a class of patterns (close to the centroid of that class).
- Advantages and problems
  - unsupervised
  - simple (less time consuming)
  - number of output nodes and the initial values of weights affects the learning results (and thus the classification quality), over- and under-classification
  - stuck vectors and unstuck

2. Self-Organizing Map (SOM)

- Motivation: from random map to topographic map
  - what is topographic map
  - biological motivations
- SOM data processing
  - network architecture: two layers
  - output nodes have neighborhood relations
  - lateral interaction among neighbors (depending on radius/distance function D)
- SOM learning
  - weight update rule (differs from competitive learning when $D > \mathbf{0}$)
  - learning algorithm (winner and its neighbors move their weight vectors toward training input)
  - illustrating SOM on a two dimensional plane
    - plot output nodes (weights as the coordinates)
    - links connecting neighboring nodes
- Applications
  - TSP (how and why)

3. Counter Propagation Networks (CPN)

- Purpose: fast and coarse approximation of vector mapping $y = \phi(x)$
- Architecture (forward only CPN):
  - three layers (input, hidden, and output)
  - hidden layer is competitive (WTA) for classification/clustering
- CPN learning (two phases). For the winning hidden node z_j

- phase 1: weights from input to hidden (w_j) are trained by *competitive learning* to become the representative vector of a cluster of input vectors.
- phase 2: weights from hidden to output (v_j)are trained by *delta rule* to become an average output of $y = \phi(x)$ for all input $\boldsymbol{x}$ in cluster j.
- learning algorithm

- Works like table lookup (but for multi-dimensional input space)
- Full CPN (bi-directional) (only if an inverse mapping $x = \phi^{-1}(y)$ exists)

7. Adaptive Resonance Theory (ART)

- Motivation: varying number of classes, incremental learning
  - how to determine when a new class needs to be created
  - how to add a new class without damaging/destroying existing classes
- ART1 model (for binary vectors)
  - architecture:
    - bottom up weights b_{ij} and topdown weights t_{ji}
    - vigilance ρ for similarity comparison
  - operation: cycle of three phases
    - *recognition (recall) phase*: competitively determine the winner j^* (at output layer) using b_{ij} as its class representative.
    - *comparison (verification) phase*: determine if the input resonates with (sufficiently similar to) class j^* using t_{ji} .
    - *weight update (adaptation) phase*: based on similarity comparison, either add a new class node or update weights (both b_{ij} and t_{ji}) of the winning node
    - vigilance ρ
  - classification as search
- ART1 learning/adaptation
  - weight update rules:

$$b_{j^*,l}(\text{new}) = \frac{s_l^*}{0.5 + \sum_{i=1}^{n} s_i^*} = \frac{t_{l,j^*}(\text{old})x_l}{0.5 + \sum_{l=1}^{n} t_{l,j^*}(\text{old})x_l} \qquad t_{j^*,l}(\text{new}) = s_l^* = t_{j^*,l}(\text{old}) \cdot x_l$$

  - learning when search is successful: only winning node j^* updates its b_{j^*} and t_{j^*} .
  - when search fails: treat $\boldsymbol{x}$ as an outlier (discard it) or create a new class (add a new output node) for $\boldsymbol{x}$
- Properties of ART1 and comparison to competitive learning networks

7. Principle Component Analysis (PCA) Networks

- What is PCA: a statistical procedure that transform a given set of input vectors to reduce their dimensionality while minimizing the information loss
- Information loss: $||x - x'|| = ||x - W^T W x||$, where $y = Wx$, $x' = W^T y$
- Pseudo-inverse matrix as the transformation matrix for minimizing information loss
- PCA net to approximate PCA:
  - two layer architecture (x and y)
  - learning to find W to minimize $\sum_l || x_l - x_l' || = \sum_l || x_l - W^T W x_l || = \sum_l || x_l - W^T y_l ||$
  - error-driven: for a given x, 1) compute y using W; 2) compute x' using W^T

3) $DW = \eta_l (y_l x_l^T - y_l y_l^T W) = \eta_l y_l (x_l^T - y_l^T W) = \eta_l y_l (x_l^T - W^T y_l)$

6. Associative Models

- Simple AM
  Associative memory (AM) (content-addressable/associative recall; pattern correction/completion)
  Network architecture: single layer or two layers
  Auto- and hetero-association
- Hebbian rule: $Dw_{j,k} = i_{p,k} \cdot d_{p,j}$
  - Correlation matrix: $w_{j,k} = \sum_{P=1}^{P} i_{p,k} d_{p,j}$
  - Principal and cross-talk term: $W \cdot i_l = d_l \cdot \|i_l\|^2 + \sum_{p \neq l} d_p \cdot i_p^T \cdot i_l$
- Delta rule: $Dw_{jk} = \eta(d_{p,j} - S(y_{p,j})) \cdot S'(y_{p,j}) \cdot i_{p,k}$, derived following gradient descent to minimize total error
- Storage capacity: up to *n* - 1 mutually orthogonal patterns of dimension *n*
- Iterative autoassociative memory
  - Motives (comparing with non-iterative recall)
  - Using the output of the current iteration as input of the next iteration (stop when a state repeats)
  - Dynamic system (stable states, attractors, genuine and spurious memories)
- Discrete Hopfield model for auto-associative memory
  - Network architecture (single-layer, fully connected, recurrent)
  - Weight matrix for Hopfield model (symmetric with zero diagonal elements)

$$w_{jk} = \begin{cases} \sum_{p=1}^{P} i_{p,k} \cdot i_{p,j} & \text{if } k \neq j \\ 0 & \text{otherwise} \end{cases}$$

  - Net input and node function: $net_k = \sum_j w_{kj} \cdot x_j + \theta_k$, $x_k = \mathrm{sgn}(net_k)$
  - Recall procedure (iterative until stabilized)
  - Stability of dynamic systems
    - Ideas of Lyapunov function/energy function (monotonically non-increasing and bounded from below)
    - Convergence of Hopfield AM: its an energy function
    - $E = -0.5 \sum_{i \neq j} \sum_j x_i x_j w_{ij} - \sum_i \theta_i y_i$
  - Storage capacity of Hopfield AM ($P \approx n/(2\log_2 n)$).
- Bidirectional AM (BAM)
  - Architecture: two layers of non-linear units
  - Weight matrix: $W_{n \times m} = \sum_{p=1}^{P} s^T(p) \cdot t(p)$
  - Recall: bi-directional (from ***x*** to ***y*** and ***y*** to ***x***); recurrent
  - Analysis
  - Energy function: $L = -XWY^T = -\sum_{j=1}^{m} \sum_{i=1}^{n} x_i w_{ij} y_j$
  - Storage capacity: $P = O(\max(n, m))$

7. Continuous Hopfield Models

- Architecture:
  - fully connected (thus recurrent) with $\boldsymbol{w_{ij}} = \boldsymbol{w_{ji}}$ and $\boldsymbol{w_{ii}} = 0$
  - net input to: same as in DHM
    internal activation u_i: $du_i(t)/dt = net_i(t)$ (approximated as $u_i(new) = u_i(old) + \delta \cdot net_i$)

output: $x_i = f(u_i)$ where ***f***(.) is a sigmoid function

- Convergence
  - energy function $E = -0.5\sum_{ij} x_i w_{ij} x_j - \sum_i \theta_i x_i$
  - $\dot{E} \le 0$ (why) so ***E*** is a Lyapunov function
  - during computation, all x_i 's change along the gradient descent of ***E***.
- Hopfield model for optimization (TSP)
  - energy function (penalty for constraint violation)
  - weights (derived from the energy function)
  - local optima
- General approach for formulating combinatorial optimization problems in NN
  - Represent problem space as NN state space
  - Define an energy function for the state space: feasible solutions as local minimum energy space, optimal solutions as global minimum energy space
  - Find weights that let the system moves along the energy reduction trajectory

8. Simulated Annealing (SA)

- Why need SA (overcome local minima for gradient descent methods)
- Basic ideas of SA
  - Gradual cooling from a high T to a very low T
  - Adding noise
  - System reaches thermal equilibrium at each T
- Boltzmann-Gibbs distribution in statistical mechanics
  - States and its associated energy

    $P_\alpha = \frac{1}{z} e^{-\beta E_\alpha}$, where $z = \sum_\alpha e^{-\beta E_\alpha}$ is the normalization factor so $\sum_r P_\alpha = 1$

    $P_\alpha / P_\beta = e^{-E_\alpha/T} / e^{-E_\beta/T} = e^{-(E_\alpha - E_\beta)/T} = e^{-DE/T}$
- Change state in SA (stochastically)
  - probability of changing from s_α to s_β (Metropolis method):

$$P(s_\alpha \to s_\beta) = \begin{cases} 1 & \text{if } (E_\beta - E_\alpha) < 0 \\ e^{-(E_\beta - E_\alpha)/T} & \text{otherwise} \end{cases}$$

  - probability of setting x_i to 1 (another criterion commonly used in NN):

$$P_i = \frac{e^{-E_a/T}}{e^{-E_a/T} + e^{-E_b/T}} = \frac{1}{1 + e^{-(E_b - E_a)/T}}.$$

- Cooling schedule
  - $T(k) = T(0)/\log(1+k)$
  - $T(k+1) = T(k) \cdot \beta$
  - annealing schedule (cooling schedule plus number of iteration at each temperature)
- SA algorithm
- Advantages and problems
  - escape from local minimum
  - very general
  - slow

9. Boltzmann Machine (BM) = discrete HM + hidden nodes + SA

- BM architecture
  - visible and hidden units
  - energy function (similar to HM)
- BM computing algorithm (SA)
- BM learning
  - what is to be learned (probability distribution of visible vectors in the training set)
  - free run and clamped run
  - learning to maximize the similarity between two distributions $P^+(Va)$ and $P^-(Va)$
  - learning takes gradient descent approach to minimize

$$G = \sum_a P^+(V_a) \ln \frac{P^+(V_a)}{P^-(V_a)}$$

  - the learning rule $Dw_{ij} = -\mu(p_{ij}^+ - p_{ij}^-)$ (meaning of p_{ij}^+ and p_{ij}^-, and how to compute them)
  - learning algorithm
- Advantages and problems
  - higher representational power
  - learning probability distribution
  - extremely slow

10. Evolutionary Computing

- Biological evolution:
  - Reproduction (cross-over, mutation)
  - Selection (survival of the fittest)
- Computational model (genetic algorithm)
  - Fitness function
  - Randomness (stochastic process)
  - termination
- Advantages and problems
  - General optimization method (can find global optimal solution in theory)
  - Very expensive (large population size, running for many iterations)

11. Reinforcement Learning

- Basic ideas of RL:
  - Definition
  - Differences from supervised and unsupervised learning
- Issues
  - Distribution of credit and blame
  - Random search

Note: neural network models and variations covered in the class but not listed in this document will not be tested in Exam 2.