CMSC 491N/691N **Introduction to Neural Networks** **Spring 2001**

Review 2 (Chapters. 4, 5, 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)
  - external judge
  - lateral inhibition (explicit and implicit)
- Learning (unsupervised and incremental)
  - 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)

    $\Delta \boldsymbol{w}_j = a(\boldsymbol{x} - \boldsymbol{w}_j)$ or $\Delta \boldsymbol{w}_j = a \cdot \boldsymbol{x}$ where $\boldsymbol{x}$ is the current input vector
  - learning algorithm
  - $\boldsymbol{w}_j$ is trained to represent 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)
  - stuck vectors and unsticking

2. Kohonen 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
- SOM learning
  - weight update rule (differs from competitive learning when $\boldsymbol{R} > \boldsymbol{0}$)
  - 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 $\boldsymbol{y} = f(\boldsymbol{x})$
- 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 $\boldsymbol{z}_j$

- phase 1: v_j (weights from input to hidden) is trained by competitive learning to become the representative vector of a cluster of input vectors.
- phase 2: u_j (weights from hidden to output) is trained by delta rule to become an average output of $y = f(x)$ for all input 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 $x = f^{-1}(y)$ exists)

4. Adaptive Resonance Theory (ART)

- Motivation: stability-elasticity dilemma in neural network models
  - 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: **F1(a), F1(b), F2, G1, G2, R,**
    bottom up weights b_{ij} and topdown weights t_{ji} between **F1(b)** and **F2**
  - operation: cycle of two phases
    - *recognition (recall) phase*:
      competitively determine the winner J (at **F2**) with t_J as its class representative.
    - *comparison (verification) phase*:
      determine if the input resonates with (sufficiently similar to) class J
    - vigilance r
  - classification as search
- ART1 learning/adaptation
  - weight update rules:
    $$b_{ij}(new) = \frac{L \cdot x_i}{L - 1 + |x|}, \qquad t_{ji} = x_i$$
  - learning when search is successful: only winning node J updates its $b_{.J}$ and $t_{J.}$.
  - when search fails: treat x as an outlier (discard it) or create a new class (add a node on **F2**) for x
  - learning algorithm
- Properties of ART1 and comparison to competitive learning networks

5. Continuous Hopfield model

- Architecture:
  - fully connected (thus recurrent) with $w_{ij} = w_{ji}$ and $w_{ii} = 0$
  - input to node i: $in_i = \sum_j w_{ij} \cdot v_j + q_i$
    internal activation u_i: $du_i / dt = in_i$ (approximated as $u_i(new) = u_i(old) + d \ \ in_i$)
    output: $v_i = g(u_i)$ where $g(.)$ is a sigmoid function
- Convergence
  - energy function $E = -0.5 \sum_{ij} v_i w_{ij} v_j + \sum_i q_i v_i$
  - $\dot{E} \le 0$ (why) so E is a Lyapunov function
  - during computation, all v_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 constraint satisfaction optimization problems

6. 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

    $P_a = \frac{1}{z} e^{-b E_a}$, where $z = \sum_a e^{-b E_a}$ is the normalization factor so $\sum_r P_a = 1$

    $P_a / P_b = e^{-E_a/T} / e^{-E_b/T} = e^{-(E_a - E_b)/T} = e^{-\Delta E/T}$
- Change state in SA (stochastically)
  - probability of changing from s_a to s_b (Metropolis method):

    $$P(s_a \to s_b) = \begin{cases} 1 & \text{if } (E_b - E_a) < 0 \\ e^{-(E_b - E_a)/T} & \text{otherwise} \end{cases}$$
  - 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
  - $T(k) = T(0)/\log(1+k)$ (Cauchy machine, with longer tail)
  - $T(k) = T(0)/k$, or $T(k+1) = T(k) \cdot b$
  - annealing schedule (cooling schedule plus number of iteration at each temperature)
- SA algorithm
- Advantages and problems
  - escape from local minimum
  - very general
  - slow

7. 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 take gradient descent approach to minimize

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

  - the learning rule $\Delta w_{ij} = -m(p_{ij}^+ - p_{ij}^-)$ (meaning of p_{ij}^+ and p_{ij}^-)
  - learning algorithm
- Advantages and problems
  - higher representational power
  - learning probability distribution
  - extremely slow

8. Basic Ideas of Some Other Neural Network Models

- Reinforcement learning (RL)
  - general ideas of RL (reward and penalty)
  - *ARP* (associative reward-and-penalty) algorithm for NN
    - stochastic units (for random search)
    - desired output induced by reward signal
- Recurrent BP (RBP)
  - generalization of BP to recurrent networks
  - Hopfield units
  - gradient descent to minimize error E (how to obtain E: $E = 0.5\sum_k (t_k - y_k^\infty)^2$, where y_k^∞ is computed by relaxing the original network to equilibrium)
  - transposed network, driven by error, computes weight updates by relaxing it to equilibrium.
  - weight update process for RBP
- Networks of Radial Basis Functions (RBF)
  - A better function approximator
  - unit of RBF (e.g., normalized Gaussian unit), receptive field of a unit
  - architecture and computation (compare to CPN: hidden nodes are not WTA)
  - learning (competitive for hidden units; LMS for output units)
  - compare with BP and CPN
- Probabilistic Neural Networks (PNN)
  - purpose: NN realization of Bayesian decision rule for classification
  - network structure: layers of pattern units and summation/class units
  - learning is not to minimize the error but to obtain probability density function