CMSC 475/675: Introduction to Neural Networks
Review for Exam 1 (Chapters 1, 2, 3, 4)

1. **Basics**
   - Comparison between human brain and von Neumann architecture
   - Processing units/nodes (input/output/hidden)
   - Activation/node functions (threshold/step, linear-threshold, sigmoid, RBF)
   - Network architecture (feed-forward/recurrent nets, layered)
   - Connection and weights (excitatory, inhibitory)
   - Types of learning (supervised/unsupervised), Hebbian rule,
     - Training samples
     - Overtraining/overfitting problem, cross-validation test

2. **Single Layer networks** (Perceptron, Adaline, and the delta rule)
   - **Architecture**
   - Decision boundary and the problem of linear separability \( w_0 + \sum_{i=1}^{n} x_i w_i = 0 \)
   - **Perceptron**
     - Learning rule (only when \( x_p \neq d_p \)): \( \Delta w_k = \eta \cdot \text{class}(i_p) \cdot i_p \)
     - Perceptron convergence theorem
   - **Delta learning rule** and Adaline
     - Error driven: \( E = (d_p - net_p)^2 = (d_p - \sum_k w_k i_{k,p})^2 \) or \( E = \sum_{p=1}^{P} (d_p - net_p)^2 = \sum_{p=1}^{P} (d_p - \sum_i w_i i_{i,p})^2 \)
     - Learning rule (delta rule): \( \Delta w_k = \eta \cdot (d_p - net_p) \cdot i_{k,p} \) for each training sample \( (i_p, d_p) \)
     - Gradient descent approach in deriving delta learning rule
       \[ \Delta w_k \propto -\partial E / \partial w_k = -2(d_p - net_p) i_{k,p} \]
     - Local minimum error for gradient descent approach

3. **Backpropagation (BP) Networks**
   - Multi-layer feed-forward architecture with at least one layer of hidden nodes of non-linear and differentiable activation functions
   - **Motivation to have non-linear hidden nodes** (representational power). Why non-linear?
   - Feed forward computing
   - **BP learning**
     - Training samples: \( (i_p, d_p) \)
     - Obtain errors at output layer (feed-forward phase): \( \delta_k = (d_k - o_k) S'(net_k^{(2)}) \)
     - Obtain errors at hidden layer (error backpropagation phase): \( \mu_j = (\sum \delta_k w_k^{(2,1)}) \cdot S'(net_j^{(1)}) \)
     - Weight update: \( \Delta w_{k,j}^{(2,1)} = \eta \cdot \delta_k x_j^{(1)}, \Delta w_{j}^{(1,0)} = \eta \cdot \mu_j x_i \)
     - Why BP learning works (gradient descent to minimize error): \( \Delta w_k = -\eta \cdot \partial E / \partial w_k \)
     - Learning procedure (batch and sequential modes)
     - In what sense BP learning generalizes the delta rule
   - Issues of practical concerns
     - Bias, error bound, training data, initial weights, number and size of hidden layers;
     - Learning rate (momentum, adaptive rate)
   - **Advantages and problems with BP learning**
     - Powerful (general function approximator); easy to use; wide applicability; good generalization
– Local minima; overfitting; parameters may be hard to determine; network paralysis; long learning time, black-box; hard to accommodate new samples (non-incremental learning)

• Variations of BP nets
  – Momentum term
  – Adaptive learning rate
  – Quickprop

4. Other Multilayer Nets with Supervised Learning

• Adaptive multilayer nets
  – Why smaller net (with smaller # of hidden nodes) are often preferred
  – Finding “optimal” network size: pruning and growing hidden nodes

• Cascade net (basic ideas):
  – When and how to add a new hidden node
  – What weights are to be trained when a new node is added, and how they are trained

• Prediction networks:
  – BP nets for prediction
  – Recurrent nets: unfolding vs gradient descent

• NN of radial basis function (RBF)
  – Definition of RBF, examples of RBF (especially Gaussian function)
  – Advantages of RBF wrt sigmoid functions
  – RBF for function approximation

• Polynomial networks

Types of questions that may appear on Exam 1:

• True/False
  – Backpropagation learning is guaranteed to converge.

• Definitions
  – Recurrent networks.

• Short questions (conceptual)
  – What are the major differences between human brain and Von Neumann machine?

• Longer questions
  – What is the overfitting problem in BP learning? What can you suggest to ease this problem?

• Apply some NN model to a small concrete problem
  – Construct a neural network with one hidden node and one output node to solve the XOR problem. The network should be feedforward but not necessarily layered.