Computational Methods in Cognitive Science (BCS/CSC 512)

Homework #2

Question 1

Recall the exclusive-or (xor) task:

Pattern	x1	x2	y*
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

Using the backpropagation algorithm, train a feedforward network with two input units, four hidden units, and one output unit to perform this task. The hidden and output units should use the logistic activation function. The network should be trained to maximize a Bernoulli likelihood function. All units should have a bias weight. Set the learning rate to 1.0 (or perhaps something smaller; do not use momentum). Present the four training patterns in a random order (use on-line learning) and let the simulation run for 2500 epochs. First, initialize the network's weights to zero and train the network. What happens? Why? Next, initialize the network's weights to small random values (perhaps in the range of [-0.1, 0.1]). What happens? Write down each unit's weights and also the activation of each unit in response to each pattern. Explain in words how the network has learned to solve the problem (that is, try to interpret the hidden units as performing logical predicates such as OR, AND, NOT, NOR (NOT OR), NAND (NOT AND)). Run the network several more times. Does it always discover the same solution? Why or why not?

Question 2

Here are four training patterns:

Pattern	x1	x2	x3	x4	y1*	y2*	y3*	y4*
1	1	0	0	0	1	0	0	0
2	0	1	0	0	0	1	0	0
3	0	0	1	0	0	0	1	0
4	0	0	0	1	0	0	0	1

Using the backpropagation algorithm, train a feedforward network with four input units, two hidden units, and four output units to perform this task. Let the hidden units use the logistic activation function, and the output units use the softmax activation function. The network should maximize a multinomial likelihood function. [Hint: Make sure that you understand the derivative of the softmax function, and how to implement this derivative when updating an output unit that uses this activation function. In previous years, several students struggled with this part.] All units should have a bias weight. Set the learning rate to 1.0 (or perhaps something smaller; do not use momentum). Initialize the network's weights to small random values. Present the four training patterns in a random order (use on-line learning) and let the simulation run for 2500 epochs. What happens? Write down each unit's weights and also the activation of each unit in response to each pattern. Explain in words how the network has learned to solve the problem (again, try to interpret the hidden units in terms of logical predicates). Run the network several more times. Does it always discover the same solution? Why or why not? Now train a network that has four hidden units in exactly the same way. Write down each unit's weights and also the activation of each unit in response to each pattern. Explain in words how the network has learned to solve the problem. Are any of the hidden units redundant (that is, are any of the hidden units computing the same function)? Next train a network that has eight hidden units in exactly the same way. Are you still able to easily figure out how the network has learned to solve the problem? If so, please explain. If not, why not?

Question 3

Using the backpropagation algorithm, train a network to approximate the function $y^* = \sin(x) + \epsilon$ where ϵ is a random sample from a Normal distribution whose mean is 0.0 and whose standard deviation is 0.2. Generate two data sets, one for training and one for testing (each data set should have about 1000 items). The value of x should range between $-\pi$ and π. The network should have 1 input unit, 10 hidden units, and 1 output unit. The hidden units should use a logistic activation function, and the output unit should use a linear activation function. The network should maximize a Normal likelihood function. All units should have a bias weight. Set the learning rate to a small value (try 0.001; do not use momentum). Initialize the network's weights to small random values. Present the training patterns in a random order (use on-line learning) and let the simulation run for 2500 epochs. What happens? After every 20th epoch, evaluate the network's performance on the data items from the test set (i.e., compute the appropriate log likelihood function using the test set data items). Present a graph in which you report these log likelihood values. Repeat this set of simulations but with a network that has 30 hidden units. What happens? Do you see any evidence that the network is overfitting the training data?