Answers to Homework # 2

15-496/782: Introduction to Artificial Neural Networks

Dave Touretzky, Spring 2004

Problem 1. Consider a multilayer perceptron whose hidden units use x^5 and whose output units use $\cos(2x)$ as the transfer function, rather than the usual sigmoid or tanh. Using the chain rule, starting from $\partial E/\partial y_k$, derive the formulas for the weight updates Δw_{jk} and Δw_{ij}. Your final formulas should be purely algebraic, i.e., they should not contain partial derivatives.

Answer: the original derivation of the backpropagation learning rule can be copied very closely; only the nonlinear transfer function and its derivative change.

$$\frac{\partial E}{\partial y_k} = (y_k - d_k) \tag{1}$$

$$\frac{\partial E}{\partial net_k} = \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial net_k} = (y_k - d_k) \cdot -2\sin(2net_k) \tag{2}$$

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}} = -\eta \frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{jk}} = -\eta \cdot (y_k - d_k) \cdot -2\sin(2net_k) \cdot y_j \tag{3}$$

$$\frac{\partial E}{\partial y_j} = \sum_k \left(\frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j} \right) = \sum_k \left(\frac{\partial E}{\partial net_k} w_{jk} \right) \tag{4}$$

$$\frac{\partial E}{\partial net_j} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} = \frac{\partial E}{\partial y_j} \cdot 5net_j^4 \tag{5}$$

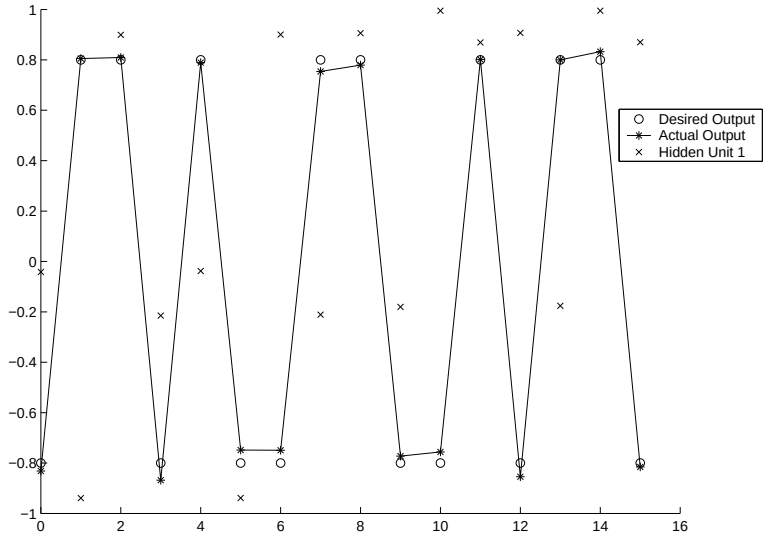
$$\begin{aligned} \Delta w_{ij} &= -\eta \tfrac{\partial E}{\partial w_{ij}} \\ &= -\eta \tfrac{\partial E}{\partial net_j} \cdot \tfrac{\partial net_j}{\partial w_{ij}} \\ &= -\eta \cdot \left(\textstyle\sum_k [(y_k - d_k) \cdot -2\sin(2net_k) \cdot w_{jk}]\right) \cdot 5net_j^4 \cdot y_i \end{aligned} \tag{6}$$

Problem 2a Write code to train a backprop network to solve the four-bit parity problem. Use targets of $+0.8$ and -0.8. Use a small learning rate, around 0.04, and no momentum.

Answer: The `bpxor` demo is easily modified to handle the four-input parity problem. The code appears on the last page.

Problem 2b. Plot the set of 16 output values, `Result2`, as a line with 16 points numbered 0–15. Update this plot every 10 epochs. Also plot the desired outputs as a set of points (not a line); use a different symbol and a different color. This will help you monitor the network's progress as it tries to get all the patterns to come out right. Note: in order to superimpose two plots, you will need to use the command `hold on`.

Answer: The answers to parts b and c are presented in part in the following plot of the network output values, the desired outputs and the outputs of one hidden unit.



Problem 2c. Try to analyze what the hidden unit is doing.

Answer: The input patterns, numbered 0 to 15 in the plot, have binary encodings of form DCBA, where D is the '8' bit, C the '4' bit, B the '2' bit, and A the '1' bit. Thus, for example, input pattern "5" has the encoding DCBA=0101. The weights between the input layer and the first hidden unit have values $Weights1(1,:) = [-0.0419, 1.5472, 0.0042, 1.5111, -1.6876]$. (Remember that the first number is the bias connection; its value is -0.0419.) The C unit's weight is near zero, so the C input is ignored by this hidden unit. The B and D units have positive weights of around 1.5, but the A unit has a negative weight of around -1.6. So either the B or the D input is enough to activate the hidden unit as long as the A input is off, but if the A input is on, both the B and D inputs must be active to bring the hidden unit's activation above zero. We can summarize the activation conditions for this unit as: $[(B \vee D) \wedge \bar{A}] \ \vee \ (B \wedge D)$.

Problem 2d. (i) How many epochs does it take to learn this problem using 5 hidden units? Run your network ten times and record the number of epochs each time, and the mean and standard deviation for the ten runs. (Use Matlab's `mean` and `std` functions.) (ii) Now run the same experiment using only 4 hidden units. (iii) Now try training with 20 hidden units. (iv) How about 100 hidden units? (v) Can the network ever solve the problem with only three hidden units? Use a learning rate of 0.015 for this experiment. (vi) Two hidden units are not enough to solve four-bit parity. But how well can the network do, i.e., how many of the 16 inputs cases can it get right, where "right" just means the output has the correct sign?

Answer:

		Trial Number											
	NHIDDENS	1	2	3	4	5	6	7	8	9	10	mean	std dev
(i)	5	1320	1170	1050	1140	1300	820	1740	1270	790	1390	1199	278
(ii)	4	1540	7980	1170	1020	1230	2580	2080	6850	6360	2450	3326	2661
(iii)	20	600	720	1230	690	730	860	1170	750	810	680	824	211
(iv)	100	1620	1650	2390	2080	810	770	2240	1170	1490	1850	1607	562

(v) A network with three hidden units has great difficulty learning this task, although some people have reported success if they train for 30,000 epochs.

Alternate problem (v): Suppose you wanted to hand-design a network with four hidden units to compute four-bit parity. Explain how to do this by using hidden units to test the number of 1 bits in the input.

Answer: Let all four hidden units have weights of $+1$ from each of the input units. Give the first hidden unit a bias connection of -0.2; this hidden until will only have a positive output if at least one input is on. Give the second hidden unit a bias of -1.2; it will have a positive output only if at least two inputs are on. Give the third hidden unit a bias connection of $-2; 2$, and the fourth hidden unit a bias connection of -3.2. Give the output unit a bias of -0.2, and assign its connections from the four hidden units weights of $+1$, -1, $+1$, and -1. The output unit will have positive activation only if an odd number of inputs are active.

(vi) A network with 2 hidden units can get 15 of the inputs correct. Training usually results in 7 positive and 4 negative targets being met almost exactly, with another 4 negative outputs resulting in some intermediate non-zero value (ie. -0.5) with the correct sign. This is also the value incorrecly learned for the remaining positive target. Or vice versa (exchange positive with negative in this paragraph.)

```
% Four-input parity problem.
%
% David S. Touretzky.  February, 2002.

Patterns = zeros(4,16);
Patterns(4,[ 2  4  6  8 10 12 14 16]) = 1;
Patterns(3,[ 3  4  7  8 11 12 15 16]) = 1;
Patterns(2,[ 5  6  7  8 13 14 15 16]) = 1;
Patterns(1,[ 9 10 11 12 13 14 15 16]) = 1;
Desired = mod(sum(Patterns,1),2)*1.6 - 0.8;

[NINPUTS,NPATS] = size(Patterns);
NHIDDENS = 5;
[NOUTPUTS,NPATS] = size(Desired);

LearnRate = 0.04;
Momentum = 0;
DerivIncr = 0.02;
deltaW1 = 0;
deltaW2 = 0;

Inputs1 = [ones(1,NPATS); Patterns];

Weights1 = rand(NHIDDENS,1+NINPUTS)-0.5;
Weights2 = rand(NOUTPUTS,1+NHIDDENS)-0.5;

TSS_Limit = 0.02;

for epoch = 1:10000
  bp_innerloop

  if rem(epoch,10) ==0
    fprintf('Epoch %4d:  Error = %f\n',epoch,TSS);
    if TSS < TSS_Limit, break, end
    clf, whitebg(gcf,[0 0 0]), hold on
    axis([0 16 -1 1])
    plot(0:NPATS-1,Desired,'o')
    plot(0:NPATS-1,Result2,'g*-')
    plot(0:NPATS-1,Result1(1,:),'mx')
    drawnow
  end
end

legend('Desired Output','Actual Output','Hidden Unit 1');
```