Homework # 1 Answers

15-496/782: Introduction to Artificial Neural Networks

Dave Touretzky, Spring 2004

Problem 1. Using inequalities expressed in terms of the weights w_0, w_1, and w_2, prove that the perceptron cannot solve the following classification problem:

$$Patterns = \begin{bmatrix} 2 & 6 \\ 1 & 3 \\ 3 & 9 \end{bmatrix} \qquad Desired = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Answer: the proof is similar to the proof that perceptrons cannot do XOR. Assume some set of weights exists that performs the task correctly. Then:

1.	$w_0 + 2 \cdot w_1 + 6 \cdot w_2 \leq 0$	by pattern 1
2.	$w_0 + w_1 + 3 \cdot w_2 > 0$	by pattern 2
3.	$w_0 + 3 \cdot w_1 + 9 \cdot w_2 > 0$	by pattern 3
4.	$2 \cdot w_0 + 4 \cdot w_1 + 12 \cdot w_2 > 0$	by pattern 2 + pattern 3
5.	$2 \cdot w_0 + 4 \cdot w_1 + 12 \cdot w_2 \leq 0$	2 × pattern 1

Lines 4 and 5 are contradictory.

Problem 2. The proof of the perceptron convergence theorem states that if $\hat{w}$ is a weight vector that correctly classifies all patterns, and $\overline{w}^{(\tau)}$ is the weight vector at step τ of the algorithm, then $\hat{w} \cdot \overline{w}^{(\tau)}$ increases at each step. Modify the `perceptron` program to demonstrate this by displaying the value at each step. Turn in your source code and a sample run.

Answer: We are given the values for `slope` and `yint`, which define the decision boundary. For points in class 1 we have

$$x_2 - \texttt{slope} \cdot x_1 - \texttt{yint} > 0$$

The perceptron computes

$$w_0 + w_1 x_1 + w_2 x_2 > 0$$

From this last equation we can calculate the y-intercept, i.e., the value of x_2 when $x_1 = 0$, as $w_0/w_2 = -x_2 = -\texttt{yint}$. The slope of the decision boundary is $-w_1/w_2$. We can choose any value for w_2 and calculate the remaining weights. If we choose $w_2 = 1$ we get a correct weight vector of $[w_0\ w_1\ w_2] = [$`-yint -slope` $1]$.

The code on the following page displays both the dot product and the angle in degrees between the correct weight vector and the current weight estimate. You will note that the dot product increases with every iteration, as the Perceptron Convergence Theorem requires. The angle does not not always decrease, but the general trend is downward.

```
% perceptron - Perceptron demo with random training set.

% Calculate and plot the training set.
NPATS = 10+floor(rand(1)*30);
Patterns = rand(2,NPATS)*2-1;
slope = log(rand(1)*10);
yint = rand(1)*2-1;
Desired = Patterns(2,:) - Patterns(1,:)*slope - yint  > 0;

%Calculate the correct weight vector
correctweights = [-yint -slope 1];

PlotPats(Patterns,Desired)

Inputs = [ones(1,NPATS); Patterns];
Weights = [0 0 0];

for i = 1:50
  % Calculate and display the dot product
  dotprod = Weights*correctweights';
  angl = acos(dotprod/max(1e-10,(norm(Weights)*norm(correctweights))))/pi*180;
  fprintf('%2d. dot=%7.4f, ang=%7.4f   Weights = ',i,dotprod,angl);
  disp(Weights);

  Result = (Weights * Inputs) > 0;
  if Result == Desired, break, end
  Weights = Weights + (Desired-Result) * Inputs';

  PlotBoundary(Weights,i,0)
  pause(1)
end

PlotBoundary(Weights,i,1)
```

```
>> percepans
 1. dot= 0.0000, ang=90.0000   Weights =      0      0      0
 2. dot= 6.3510, ang=70.3604   Weights =     8.0000   -1.8914    0.9488
 3. dot=17.6129, ang=42.7304   Weights =    -6.0000   -8.5063    1.3795
 4. dot=18.8872, ang=12.3858   Weights =    -1.0000   -7.8111    3.1145
 5. dot=20.1615, ang=21.5806   Weights =     4.0000   -7.1158    4.8496
 6. dot=25.1815, ang=30.1488   Weights =    -5.0000  -11.3636    2.9161
 7. dot=26.4557, ang= 5.4394   Weights =          0  -10.6683    4.6512
 8. dot=26.6684, ang= 8.4328   Weights =     2.0000   -9.7617    6.3326
 9. dot=26.8576, ang= 4.9860   Weights =     1.0000  -10.0758    6.0721
```

Problem 3. Run the `bowl` demo with learning rates of 0.01, 0.1, 0.142, and 0.15. Hand in a printout of each run. What can you say about the model's behavior at each learning rate?

Answer: See Figure 1 for the results of running `bowl` with learing rates of 0.01, 0.1, 0.142, and 0.15. With learning rate = 0.01, the algorithm is very slow to converge and the changes in the weights, at each iteration, are small. Using a learning rate of 0.1, the convergence was much faster (requiring fewer iterations) than with a rate of 0.01, but otherwise the behavior was similar. With a learning rate of 0.142, the sucessive weight vectors oscillated due to an overcorrecting of the error, but eventually settled to the optimal solution (bottom of the bowl). The algorithm is unstable with a learning rate of 0.15. Each iteration of the algorithm drives the weights further away from the optimal solution.

Problem 4. Consider the function $f(x, y) = \exp\left(-(0.6x - 0.7)^2 - (y - 0.4)^2\right)$. (a) Train an LMS network to approximate this function over the unit square ($0 \leq x \leq 1$, $0 \leq y \leq 1$). (b) What is the shape of the approximation function? (c) By looking at the weights of your trained network, you can see the first degree polynomial that the neural network has devised to approximate f. Write down this polynomial, and hand it in along with the code you wrote to solve this problem.

Answer: The approximation function is the plane: $z = 0.3488x - 0.1368y + 0.6431$. Figure 2 shows the function $f(x, y)$ and its linear approximation. The code is presented on a subsequent page.

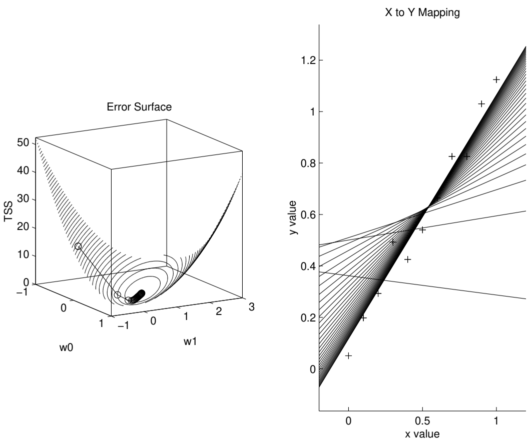
Problem 5. How much information does it take to describe a two-input perceptron? The classical description uses a vector of three real-valued parameters: $\vec{w} = \langle w_0, w_1, w_2 \rangle$. But the perceptron's decision boundary is a line, which can be uniquely specified with just two parameters, e.g., slope and intercept.

Jack says: "I claim a perceptron can be described with less information than three real numbers. Here's how I would do it with just two real values: set $s_0 = w_0/w_2$, and $s_1 = w_1/w_2$. From the description $\langle s_0, s_1 \rangle$, I can construct a weight vector $\langle s_0, s_1, 1 \rangle$ that behaves exactly the same as $\vec{w}$ for all inputs."

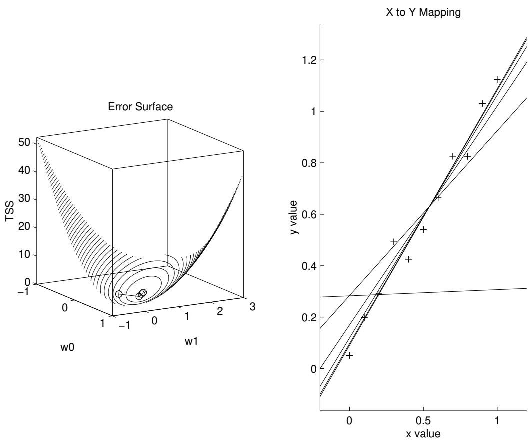
Jill replies: "I claim a perceptron requires more than just two real numbers to describe. Consider the case where w_2 is negative. What will your approach do?"

(a) Whose claim is correct, and why? (b) How much information does it really take to correctly describe a two-input perceptron? (Don't worry about the case where $w_2 = 0$.)

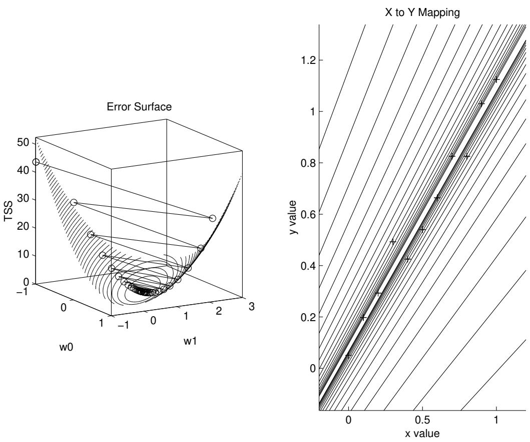
Answer. Jill's claim is correct. Jack has the right idea initially when he says that three real numbers is overkill; the description of the line is overdetermined since w_0, w_1 and w_2 are not independent. But in settling on just two real numbers he is throwing away too much information, as Jill points out. If w_2 is negative, division through by w_2 requires us to exchange $\leq$ for $>$ and vice versa, otherwise the resulting perceptron will misclassify every training point. So it takes two real numbers plus a single bit to describe a two-input perceptron: two real numbers to describe the line, as Jack proposes, and one bit to specify which side of the line is 1 and which is 0.



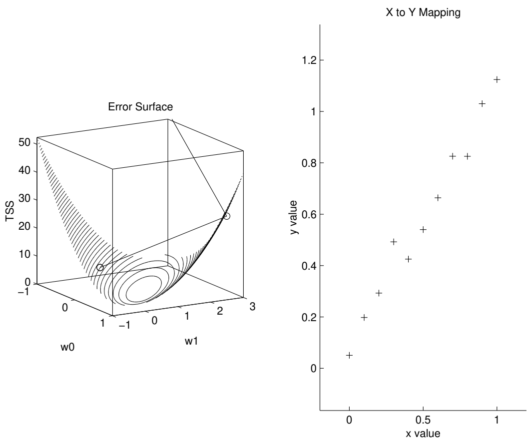
(a) Learning rate = 0.01



(b) Learning rate = 0.1



(c) Learning rate = 0.142



(d) Learning rate = 0.15

Figure 1: Printouts for each learning rate for Problem 3.

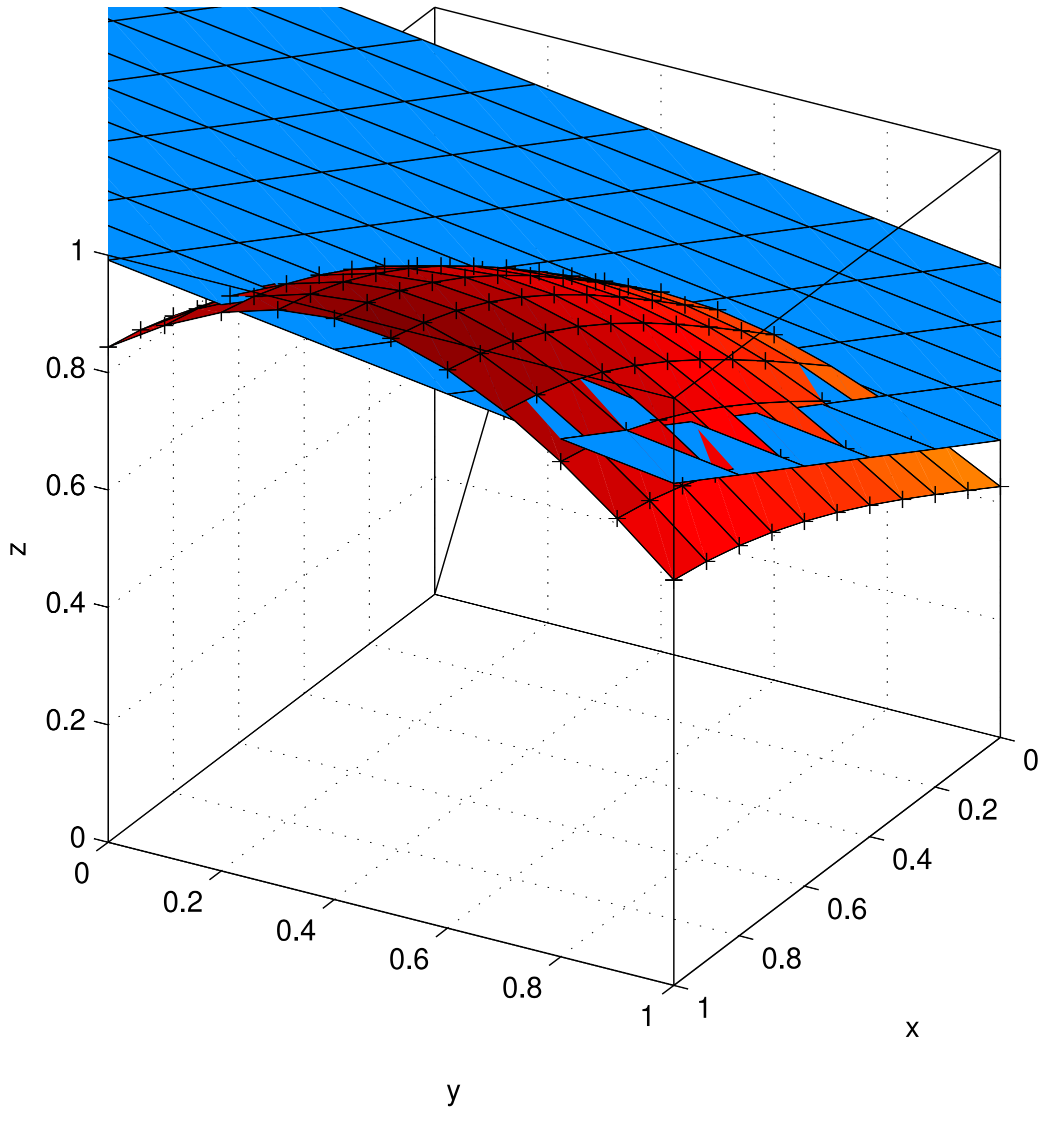


Figure 2: The function $f(x, y)$ and its linear approximation

```
% Answer to homework 1, problem 4.

pts = 0:0.1:1;
[x,y] = meshgrid(pts);
z = exp(-((0.6*x-0.7).^2+(y-0.4).^2));

Patterns = [x(:) y(:)];
Desired = z(:);
NPATS   = size(Patterns,1);
NINPUTS = size(Patterns,2);
NUNITS  = size(Desired,2);

Inputs = [ones(NPATS,1) Patterns];
InitWeights = rand(NUNITS,1+NINPUTS) - 0.5;
Weights = InitWeights;

PlotPats3D(Patterns,Desired)
surf(x,y,z)
axis([0 1 0 1 0 1]), view(10,25)

LearnRate = 0.01;
TSS = Inf;

for epoch = 0:2000,
  if rem(epoch,5) == 0
    PlotBoundarySurface(Weights)
    pause(0.25)
  end

  NetIn = Inputs * Weights';
  Result = NetIn;
  Error = Result - Desired;
  OldTSS = TSS;
  TSS = sum(sum(Error.^2));
  fprintf('%2d.  TSS=%8.5f,    Weights =',epoch,TSS);
  disp(Weights);

  if abs(TSS-OldTSS) < 0.00001
    PlotBoundarySurface(Weights)
    break;
  end;

  dW = - (Error' * Inputs);
  Weights = Weights + LearnRate * dW;
end
```