NN2

L. Tostrams (s4386167) R. van Kemenade (s0839795)

February 2015

Exercise 1

a

We have two input variables X1 and X2 and they have corresponding weights W1 and W2. Both weights are 0.5 and X1 and X2 are binary. You multiply X1 and X2 with their respective weights and sum the result, this gives you X. As activation function we have :

$$y(X-1) = \begin{cases} 1, & \text{if } X \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

If one of the inputs is 1 and the other 0 this results in a 0. And like in an and gate if both inputs are 1 then the result will also be 1.

b

We know that the decision boundary is $W^t x = 0$ and this means that $W1 * X1 + ... + Wn * Xn = 0$ This is equal to the inproduct being zero. The formula for the angle between two vectors is: But if the top half (the inproduct) is zero, then the results will be 0 and consequently the angle will be 90 degrees.

Exercise 2

a

$E^n(w) = \frac{1}{2}(y^n - t^n)^2$ is a composition of 2 functions: $E^n(z) = \frac{1}{2}(z)^2$ and $z(w) = y^n - t^n$. To calculate $\frac{\partial E^n}{\partial w_i}$ we can use the chain rule, so that:

$\frac{\partial E^n}{\partial w_i} = \frac{\partial E^n}{\partial z} * \frac{\partial z}{\partial w_i}$

then, calculate:

$\frac{\partial E^n}{\partial z} = z = y^n - t^n$

$\frac{\partial z}{\partial w_i} = \frac{\partial y^n}{\partial w_i}$

and so:

$\frac{\partial E^n}{\partial w_i} = (y^n - t^n) * \frac{\partial y^n}{\partial w_i}$

b

We have: $y^n = f(a^n)$ and $a^n = w^T * x^n$, so

$\frac{\partial y^n}{\partial w_i}$ is equivalent to $\frac{\partial f(a^n)}{\partial w_i}$.

Then, with the chain rule:

$\frac{\partial f(a^n)}{\partial w_i} = \frac{\partial f(a^n)}{\partial a^n} * \frac{\partial a^n}{\partial w_i}$

and then

$a^n = w_1 * x_1^n + ... + w_i * x_i^n + ... + w_m * x_m^n$

$\frac{\partial a^n}{\partial w_i} = x_i^n$

so:

$\frac{\partial f(a^n)}{\partial w_i} = \frac{\partial f(a^n)}{\partial a^n} * x_i^n$

and with $f(a^n) = y^n$ that becomes

$\frac{\partial y^n}{\partial w_i} = \frac{\partial f(a^n)}{\partial a^n} * x_i^n$

c

If we use a linear activation function, $f(a^n) = a^n$, and then $\frac{\partial f(a^n)}{\partial a^n}$ will evaluate to 1. Then the partial derivative reduces to:

$\frac{\partial E(w)}{\partial w_i} = \sum_n (y^n - t^n) x_i^n$

This is related to the perceptron convergence procedure, because this partial derivative is almost equal to the way in which the weightvector w changes.

Exercise 3

a

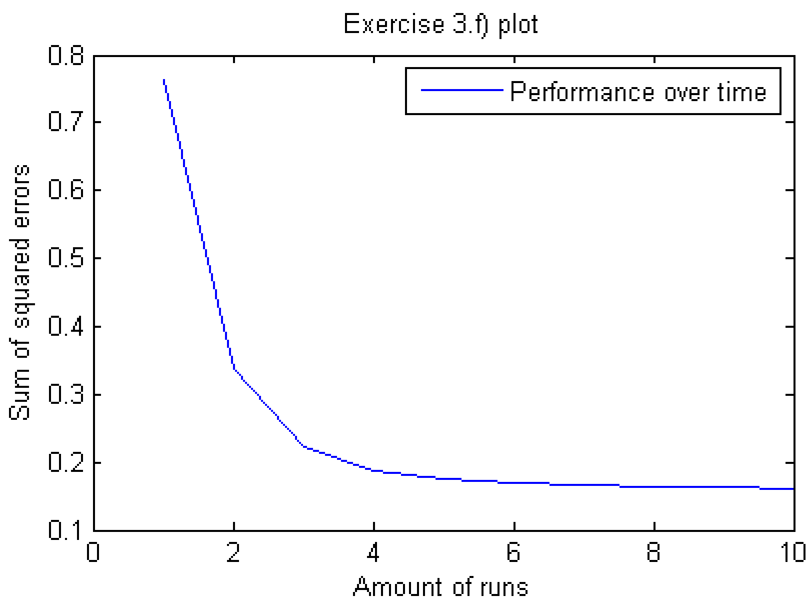
```
load or.mat
whos
```

b,c,d,e

See Matlab code

f

It makes sense that the output is not binary, because there is no activation function and treshold for the output. The performance of the perceptron improves significantly:



Matlab code perceptron.m

```
% perceptron.m Lisa Tostrams s4386167 Rob van Kemenade s0839795
% Exercise 2

load or.mat

[weights outputs performance] = slp(X,Y, 0.1, 10);
plot(performance)
xlabel('Amount of runs');
ylabel('Sum of squared errors');
legend('Performance over time');
title('Exercise 3.f) plot');
```

Matlab code slp.m

```
% slp.m Lisa Tostrams s4386167 Rob van Kemenade s0839795
function [weights outputs performance] = slp(X, Y, alpha, nepochs)
%X is the data, Y the output labels, alpha the learning rate, and nepochs
%the number of epochs it should run. This function will train the weights
%of a perceptron, and return this in the matrix 'weights'.
k = size(X, 1);
n = size(X, 2);

weights = rand(1, k) * 0.1;                    %initialize random weights to start
performance = zeros(1,nepochs);
```

```
outputs = zeros(nepochs,n);
P = randperm(n);                                % randomize the order of the input
for e = 1:nepochs
    SSE = 0;
    for i = 1:n
        pattern = P(i);                         %pick an input pair
        onePattern = X(:,pattern);
        activation = weights * onePattern;      %compute the output
        outputs(e,i) = activation;              %store output
        target = Y(:,pattern);
        errorTerm = (target - activation);      %calculate error
        SSE = SSE + (errorTerm^2);              %calculate squared sum of errors
        deltaWeights = errorTerm * onePattern' * alpha;  %calculate how weights
        weights = weights + deltaWeights;       % update weights
    end
    performance(e) = 0.5*SSE;                   %store performance
end

end
```