SFWR ENG 4X03 Assignment 4

1.a)

function netbp(points,labels,neurons,learning\_rate,niter,file)

%NETBP Uses backpropagation to train a network

%%%%%%% DATA %%%%%%%%%%%

x1 = points(1,:);

x2 = points(2,:);

% Initialize weights and biases

rng(5000);

W2 = 0.5\*randn(neurons(1),2); W3 = 0.5\*randn(neurons(2),neurons(1)); W4 = 0.5\*randn(neurons(3),neurons(2));

b2 = 0.5\*randn(neurons(1),1); b3 = 0.5\*randn(neurons(2),1); b4 = 0.5\*randn(neurons(3),1);

% Forward and Back propagate

savecost = zeros(niter,1); % value of cost function at each iteration

for counter = 1:niter

k = randi(14); % choose a training point at random

x = [x1(k); x2(k)];

% Forward pass

a2 = activate(x,W2,b2);

a3 = activate(a2,W3,b3);

a4 = activate(a3,W4,b4);

% Backward pass

delta4 = a4.\*(1-a4).\*(a4-labels(:,k));

delta3 = a3.\*(1-a3).\*(W4'\*delta4);

delta2 = a2.\*(1-a2).\*(W3'\*delta3);

% Gradient step

W2 = W2 - learning\_rate\*delta2\*x';

W3 = W3 - learning\_rate\*delta3\*a2';

W4 = W4 - learning\_rate\*delta4\*a3';

b2 = b2 - learning\_rate\*delta2;

b3 = b3 - learning\_rate\*delta3;

b4 = b4 - learning\_rate\*delta4;

% Monitor progress

newcost = cost(W2,W3,W4,b2,b3,b4);

%fprintf("newcost = %f\n",newcost); % display cost to screen

savecost(counter) = newcost;

end

% Show decay of cost function

save costvec

semilogy([1:1e4:niter],savecost(1:1e4:niter))

function costval = cost(W2,W3,W4,b2,b3,b4)

costvec = zeros(10,1);

for i = 1:10

x =[x1(i);x2(i)];

a2 = activate(x,W2,b2);

a3 = activate(a2,W3,b3);

a4 = activate(a3,W4,b4);

costvec(i) = norm(labels(:,i) - a4,2);

end

costval = norm(costvec,2)^2;

end % of nested function

save(file,'W2','W3','W4','b2','b3','b4','savecost','learning\_rate');

end

b)

function category = classifypoints(file,points)

load(file);

% Forward pass

a2 = activate(points,W2,b2);

a3 = activate(a2,W3,b3);

a4 = activate(a3,W4,b4);

% Categorizing output of point

category = a4(1,:) >= a4(2,:);

end

c) Larger values of learning rates seem to faster result in an accurate answer and more neurons result in more complex mapping. The best combination that I was able to run is a learning rate of 1 and a neuron configuration of [7 50 2]

2.

Bisection

f = @(x) x - exp(2-sqrt(x)); f(x) = 1.877322

g = @(x) x\*sin(x^2)-1; g(x) = 4.368127

Newton

h = @(x) x^3-2\*x-5; h(x) = 2.094551

g = @(x) x\*sin(x^2)-1 g(x) = 3.930742

fzero f(x) = 1.877322

fzero g(x) = 4.368127

fzero h(x) = 2.094551

Newton’s method for g(x) has a significantly larger error

3.