Colton Cunov
Artificial Neural Networks, Oct 2020

Homework 3

1. Convolutional Networks

2. Fully connected Autoencoder

3. Restricted Boltzmann Machine

4. Tic Tac Toe

5. Appendices (code)

a. Convolutional Networks

b. Fully connected Autoencoder

c. Restricted Boltzmann Machine

d. Tic Tac Toe

1. Convolutional networks (2020)

The below tables were calculated from a confusion matrix of each dataset.

Network 1

Digit	Train % Error	Validation % Error	Test % Error
0	0.06%	0.14%	0.11%
1	0.09%	0.14%	0.11%
2	0.10%	0.12%	0.13%
3	0.12%	0.30%	0.09%
4	0.08%	0.15%	0.08%
5	0.12%	0.20%	0.14%
6	0.05%	0.14%	0.19%
7	0.11%	0.11%	0.18%
8	0.15%	0.29%	0.23%
9	0.16%	0.28%	0.25%
Total	**1.04%**	**1.87%**	**1.51%**

The hardest digits for this network to classify are 8 and 9, together accounting for almost a third of the errors across all datasets. The easiest are 0 and 4, together accounting for only 14%.

Network 2

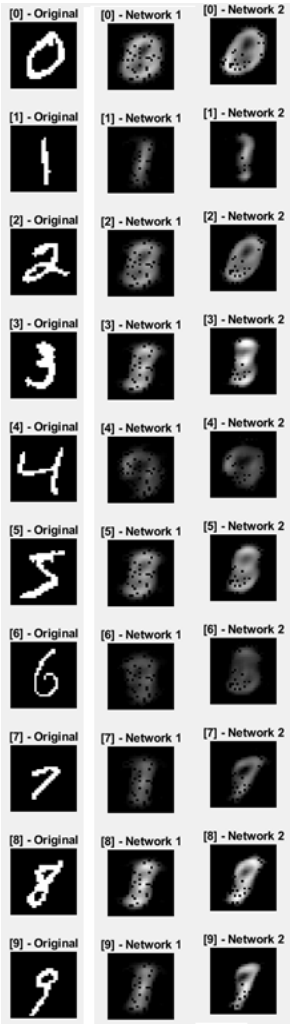
Digit	Train % Error	Validation % Error	Test % Error
0	0.06%	0.09%	0.03%
1	0.08%	0.11%	0.04%
2	0.12%	0.16%	0.12%
3	0.13%	0.15%	0.15%
4	0.11%	0.12%	0.08%
5	0.08%	0.10%	0.12%
6	0.05%	0.12%	0.18%
7	0.10%	0.12%	0.16%
8	0.13%	0.22%	0.10%
9	0.13%	0.20%	0.24%
Total	**0.99%**	**1.39%**	**1.22%**

The hardest digits for this network to classify are 8 and 9 again, together accounting for 28% of the total errors. The easiest are 0 and 1, together accounting for only 11%.

This network beats the previous network in all datasets, and didn't take any longer to train. This suggests that additional layers can compensate for fewer epochs in regards to classification accuracy and training time.

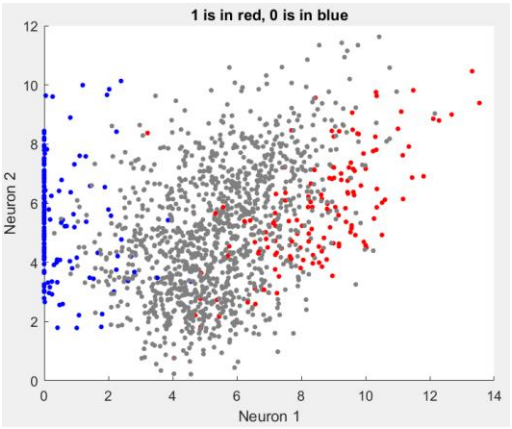
2. Fully connected Autoencoder

Network 1 reproduces about two of them convincingly (0 and 1, the easiest) and Network 2 reproduces about four convincingly (0, 1, 3, and 7).



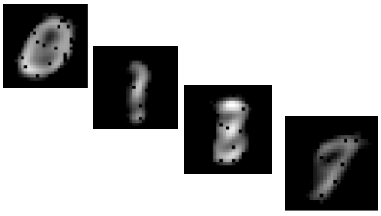
Yes! There is a pattern. It separates them linearly. I could use SVM to get a y=mx+b line that discriminates the two.

I chose three random points from each cluster on the scatter plot and fed them into the decoder. They were all reproduced correctly.



For autoencoder 2, one neuron lights up more for each convincingly reproduced digit. That is to say, one neuron will have a significantly higher value than the others and which neuron that is dictates what the decoder will reproduce. The other digits will have random values with no one neuron dominating. Examples are shown below.

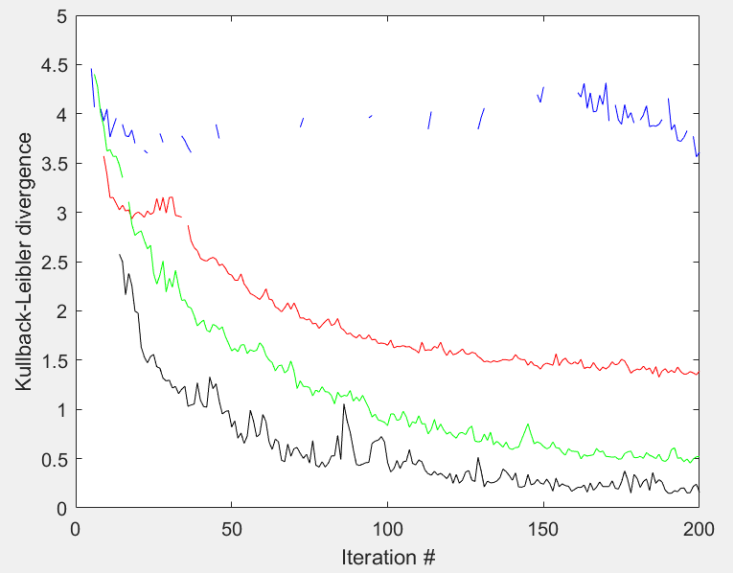
Digit	Neuron 1	Neuron 2	Neuron 3	Neuron 4
0	**22.2246**	13.4539	16.5791	9.5972
1	4.7702	**29.407**	3.3738	2.6155
3	10.6605	3.2841	**19.6941**	2.6546
7	4.9645	5.3134	2.9449	**14.935**



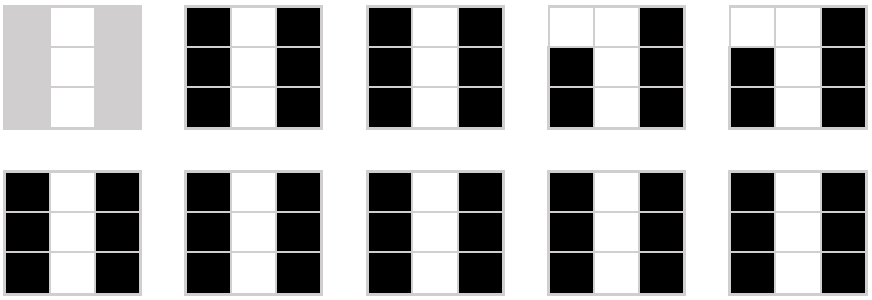
3. Restricted Boltzmann machine

The figure to the right shows the KL-divergence for the **two**, **four**, **eight**, and **sixteen** hidden neuron cases.

To estimate the Boltzmann distributions, I sampled 20k randomly generated patterns.



For a sample pattern, where the 2nd column is -1 and the other values are 0, the first 10 iterations are shown below. The network converges to a stored pattern in the 1st iteration, but made an error in the 3rd iteration which was not corrected until the 5th. I tried a few other ones that all converged after the 1st iteration, but this one was interesting.



4. Tic tac toe (2020)

I wasn't able to finish this one by the deadline. What is done:

- CheckForWin.m
  - Check if a player won
- AddToQTable.m
  - Append if a state is new
- InitializeRewardTable.m
  - Optimistic initialization
- Pseudo code of the main script
- MakeMove.m is 80% done
  - Choose random with probability = epsilon, otherwise choose best from Q Table

What is left to be done:

- UpdateQTable.m
  - According to 11.19

What is done is shown starting on page 14.

5. Appendices

5.1 Code for Convolutional Networks

Network 1

```
[xTrain, tTrain, xValid, tValid, xTest, tTest] = LoadMNIST(3);

momentum = .9;
maxEpochs = 60;
miniBatchSize = 8192;
initialLearningRate = 0.001;
validationPatience = 5;
validationFrequency = 30;

options = trainingOptions('sgdm','Momentum',momentum,'MaxEpochs',maxEpochs,...
    'MiniBatchSize',miniBatchSize,'InitialLearnRate',initialLearningRate,...

'ValidationPatience',validationPatience,'ValidationFrequency',validationFrequency,...
    'Shuffle','every-epoch','ValidationData',{xValid,tValid});

layers = [
    imageInputLayer([28 28 1],"Name","imageinput")
    convolution2dLayer([5
5],20,"Name","conv","Padding",1,"WeightsInitializer","narrow-normal")
    reluLayer("Name","relu_1")
    maxPooling2dLayer([2 2],"Name","maxpool","Stride",1,"padding",0)
    fullyConnectedLayer(100,"Name","fc_1","WeightsInitializer","narrow-normal")
    reluLayer("Name","relu_2")
    fullyConnectedLayer(10,"Name","fc_2","WeightsInitializer","narrow-normal")
    softmaxLayer("Name","softmax")
    classificationLayer("Name","classoutput")];

net = trainNetwork(xTrain,tTrain,layers,options);

tTrainPred = classify(net,xTrain);
tValPred = classify(net,xValid);
tTestPred = classify(net,xTest);

trainConfusion = confusionmat(tTrain,tTrainPred);
validConfusion = confusionmat(tValid,tValPred);
testConfusion = confusionmat(tTest,tTestPred);
diagTrain = diag(trainConfusion);
diagValid = diag(validConfusion);
diagTest = diag(testConfusion);
for i = 1:10
    iTrainError(i) = (sum(trainConfusion(i,:)) - diagTrain(i)) /
sum(trainConfusion(:));
    iValidError(i) = (sum(validConfusion(i,:)) - diagValid(i)) /
sum(validConfusion(:));
    iTestError(i) = (sum(testConfusion(i,:)) - diagTest(i)) / sum(testConfusion(:));
end
trainError = 1 - sum(tTrain==tTrainPred) / length(tTrain);
validError = 1 - sum(tValid==tValPred) / length(tValid);
testError = 1 - sum(tTest==tTestPred) / length(tTest);
```

Network 2

```
[xTrain, tTrain, xValid, tValid, xTest, tTest] = LoadMNIST(3);

momentum = .9;
maxEpochs = 30;
miniBatchSize = 8192;
initialLearningRate = 0.01;
validationPatience = 5;
validationFrequency = 30;
options = trainingOptions('sgdm','Momentum',momentum,'MaxEpochs',maxEpochs,...
    'MiniBatchSize',miniBatchSize,'InitialLearnRate',initialLearningRate,...

'ValidationPatience',validationPatience,'ValidationFrequency',validationFrequency,...
    'Shuffle','every-epoch','ValidationData',{xValid,tValid});

layers = [
    imageInputLayer([28 28 1],"Name","imageinput")
    convolution2dLayer([3
3],20,"Name","conv_1","Padding",1,"Stride",1,"WeightsInitializer","narrow-normal")
    batchNormalizationLayer("Name","batchnorm_1")
    reluLayer("Name","relu_1")
    maxPooling2dLayer([2 2],"Name","maxpool_1","Stride",2,"Padding",0)
    convolution2dLayer([3
3],30,"Name","conv_2","Padding",1,"Stride",1,"WeightsInitializer","narrow-normal")
    batchNormalizationLayer("Name","batchnorm_2")
    reluLayer("Name","relu_2")
    maxPooling2dLayer([2 2],"Name","maxpool_2","Stride",2,"Padding",0)
    convolution2dLayer([3
3],50,"Name","conv_3","Padding",1,"Stride",1,"WeightsInitializer","narrow-normal")
    batchNormalizationLayer("Name","batchnorm_3")
    reluLayer("Name","relu_3")
    fullyConnectedLayer(10,"Name","fc","WeightsInitializer","narrow-normal")
    softmaxLayer("Name","softmax")
    classificationLayer("Name","classoutput")];

net = trainNetwork(xTrain,tTrain,layers,options);

tTrainPred = classify(net,xTrain);
tValPred = classify(net,xValid);
tTestPred = classify(net,xTest);

trainConfusion = confusionmat(tTrain,tTrainPred);
validConfusion = confusionmat(tValid,tValPred);
testConfusion = confusionmat(tTest,tTestPred);
diagTrain = diag(trainConfusion);
diagValid = diag(validConfusion);
diagTest = diag(testConfusion);
for i = 1:10
    iTrainError(i) = (sum(trainConfusion(i,:)) - diagTrain(i)) /
sum(trainConfusion(:));
    iValidError(i) = (sum(validConfusion(i,:)) - diagValid(i)) /
sum(validConfusion(:));
    iTestError(i) = (sum(testConfusion(i,:)) - diagTest(i)) / sum(testConfusion(:));
end
trainError = 1 - sum(tTrain==tTrainPred) / length(tTrain);
validError = 1 - sum(tValid==tValPred) / length(tValid);
testError = 1 - sum(tTest==tTestPred) / length(tTest);
```

5.2 Code for Fully Connected Autoencoder

```
[xTrain, tTrain, xValid, tValid, xTest, tTest] = LoadMNIST(3);
xTrainAdj = reshape(xTrain,784,[]) / 255;
xValidAdj = reshape(xValid,784,[]) / 255;
xTestAdj = reshape(xTest,784,[]) / 255;

miniBatchSize = 8192;
initialLearningRate = 0.001;
shuffle = 'every-epoch';
maxEpochs = 800;
executionEnvironment = 'gpu';

options = trainingOptions('adam','MaxEpochs',maxEpochs,...
    'MiniBatchSize',miniBatchSize,'InitialLearnRate',initialLearningRate,...

'Shuffle',shuffle,'ValidationData',{xValidAdj,xValidAdj},'ExecutionEnvironment',execut
ionEnvironment);

layers1 = [
    sequenceInputLayer(784,"Name","sequence","Normalization","rescale-zero-one")
    fullyConnectedLayer(50,"Name","fc_1","WeightsInitializer","glorot")
    reluLayer("Name","relu_1")
    fullyConnectedLayer(2,"Name","fc_2","WeightsInitializer","glorot")
    reluLayer("Name","relu_2")
    fullyConnectedLayer(784,"Name","fc_3","WeightsInitializer","glorot")
    reluLayer("Name","relu_3")
    regressionLayer("Name","regressionoutput")];

layers2 = [
    sequenceInputLayer(784,"Name","sequence","Normalization","rescale-zero-one")
    fullyConnectedLayer(50,"Name","fc_1","WeightsInitializer","glorot")
    reluLayer("Name","relu_1")
    fullyConnectedLayer(4,"Name","fc_2","WeightsInitializer","glorot")
    reluLayer("Name","relu_2")
    fullyConnectedLayer(784,"Name","fc_3","WeightsInitializer","glorot")
    reluLayer("Name","relu_3")
    regressionLayer("Name","regressionoutput")];

net1 = trainNetwork(xTrainAdj,xTrainAdj,layers1,options);
save net1
net2 = trainNetwork(xTrainAdj,xTrainAdj,layers2,options);
save net2

foundTen = 0;
found = cell(10,1);
i=1;
while foundTen ~= 10
    digit = tTrain(i);
    if isempty(found{digit})
        found{digit} = xTrainAdj(:,i);
        foundTen = foundTen + 1;
    end
    i = i + 1;
end
digits = found;

net1output = cell(10,1);
net2output = cell(10,1);
digi = 0;
for i=1:3:30
    net1output{digi+1} = reshape(predict(net1,digits{digi+1}),28,28);
    net2output{digi+1} = reshape(predict(net2,digits{digi+1}),28,28);
```

```
    subplot(10,3,i)
    h = imshow(mat2gray(reshape(digits{digi+1},28,28)),'border','tight');
    title(strcat('[',num2str(digi),'] - Original'));
    subplot(10,3,i+1)
    h = imshow(net1output{digi+1},'border','tight');
    title(strcat('[',num2str(digi),'] - Network 1'));
    subplot(10,3,i+2)
    h = imshow(net2output{digi+1},'border','tight');
    title(strcat('[',num2str(digi),'] - Network 2'));
    digi = digi + 1;
end

layers_encode1(1) = net1.Layers(1);
layers_encode1(2) = net1.Layers(2);
layers_encode1(3) = net1.Layers(3);
layers_encode1(4) = net1.Layers(4);
layers_encode1(5) = net1.Layers(5);
layers_encode1(6) = regressionLayer("Name","regressionoutput");
net1_encode = assembleNetwork(layers_encode1);

layers_decode1(1) = sequenceInputLayer(2);
layers_decode1(2) = net1.Layers(6);
layers_decode1(3) = net1.Layers(7);
layers_decode1(4) = net1.Layers(8);
net1_decode = assembleNetwork(layers_decode1);

layers_encode2(1) = net2.Layers(1);
layers_encode2(2) = net2.Layers(2);
layers_encode2(3) = net2.Layers(3);
layers_encode2(4) = net2.Layers(4);
layers_encode2(5) = net2.Layers(5);
layers_encode2(6) = regressionLayer("Name","regressionoutput");
net2_encode = assembleNetwork(layers_encode2);

layers_decode2(1) = sequenceInputLayer(4);
layers_decode2(2) = net2.Layers(6);
layers_decode2(3) = net2.Layers(7);
layers_decode2(4) = net2.Layers(8);
net2_decode = assembleNetwork(layers_decode2);

scat = [];
colorMap = [];
predictEncode1 = predict(net1_encode,xTrainAdj(:,:));
for i = 1:1500
    scat = [scat; double(tTrain(i))-1 double(predictEncode1(:,i))'];
    if ismember(double(tTrain(i)), [1 2])
        if double(tTrain(i)) == 1
            colorMap = [colorMap; [1 0 0]];
        else
            colorMap = [colorMap; [0 0 1]];
        end
    else
        colorMap = [colorMap; [.5 .5 .5]];
    end
end
scatter(scat(:,2),scat(:,3),24* ones(2, 1),colorMap,'filled')
xlabel('Neuron 1')
ylabel('Neuron 2')
title('1 is in red, 0 is in blue')


zeros = [[10,6];[8,4];[11,9]];
ones = [[1,4];[.5,8];[3,9]];
```

```
tmp = 0;
for i = 1:2:6
    tmp = tmp + 1;
    subplot(6,2,i)
    imshow(reshape(predict(net1_decode,zeros(tmp,:)'),28,28));
    subplot(6,2,i+1)
    imshow(reshape(predict(net1_decode,ones(tmp,:)'),28,28));
end


zero = [22.2246,13.4539,16.5791,9.5972];
one = [4.7702,29.407,3.3738,2.6155];
three = [10.6605,3.2841,19.6941,2.6536];
seven = [4.9645,5.3134,2.9499,14.935];
subplot(3,2,1)
imshow(reshape(predict(net2_decode,zero),28,28))
imshow(reshape(predict(net2_decode,one),28,28))
imshow(reshape(predict(net2_decode,three),28,28))
imshow(reshape(predict(net2_decode,seven),28,28))
```

5.3 Code for Restricted Boltzmann Machine

```
function DKL = CalculateDKL(PB)
    DKL = 0;
    for mu = 1:14
        DKL = DKL + (1/14)*log((1/14)/PB(mu));
    end
end


function PB = EstimateBoltzmannDistn(W,biasV,biasH,dat)
    PB = zeros(1,14);
    numHiddenNeurons = size(W,1);
    numVisibleNeurons = size(W,2);
    for T=1:20000
        randPattern = randi([0 1],9,1);
        randPattern(randPattern==0) = -1;
        [v0, vK, bH0, bHK] = IterateDynamics(randPattern,W,biasV,biasH,12);

        for mu = 1:14
            if isequal(dat(mu,:),vK')
                PB(mu) = PB(mu) + 1;
            end
        end
    end
    PB = PB / 20000;
end


function p = GetProbabilityFromB(b)
    p = 1/(1+exp(-2*b));
end


function biasH = InitializeBiases(numNeurons)
    biasH = -1 + 2*rand(1,numNeurons);
end


function theta = InitializeThresholds(numHidden)
theta = -1 + 2*rand(1,numHidden);
end


function W = InitializeWeights(numVisible,numHidden)
    W = zeros(numVisible,numHidden);
    for i=1:numVisible
        for j=1:numHidden
            W(i,j) = -1 + 2*rand;
        end
    end
    W = W';
end


function [v0, vK, bH0, bHK] = IterateDynamics(pattern,W,biasV,biasH,K)
    numHiddenNeurons = size(W,1);
    numVisibleNeurons = size(W,2);
    v = pattern;
    v=[0 -1 0 0 -1  0 0 -1 0 ]';
    v0 = v;
    bH = W * v - biasH';
    bH0 = bH;
```

```
    h = [];
    for i = 1:numHiddenNeurons
        if rand < GetProbabilityFromB(bH(i))
            h(i) = 1;
        else
            h(i) = -1;
        end
    end
    h = h';
    vPrint=[];
    for t = 1:K
        bV = (h' * W - biasV)';
        vPrint = [vPrint v];
        for j = 1:numVisibleNeurons
            if rand < GetProbabilityFromB(bV(j))
                v(j) = 1;
            else
                v(j) = -1;
            end
        end
        bH = W * v - biasH';
        for i = 1:numHiddenNeurons
            if rand < GetProbabilityFromB(bH(i))
                h(i) = 1;
            else
                h(i) = -1;
            end
        end
    end
    bHK = bH;
    vK = v;
end


clear all
dat = LoadData();
numVisibleNeurons = 9;
numPatterns = 14;
learningRate = .01;
K = 100;
numIterations = 200;
DKL = zeros(4,numIterations);

iHiddenNeuron = 1;
for numHiddenNeurons=[2 4 8 16]
    W = InitializeWeights(numVisibleNeurons,numHiddenNeurons);
    biasH = InitializeBiases(numHiddenNeurons);
    biasV = InitializeBiases(numVisibleNeurons);
    for T = 1:numIterations
        tmp = randperm(numPatterns);
        p0 = randi(numPatterns);
        p0 = tmp(1:p0);
        for mu = p0

            pattern = dat(mu,:)';
            [v0, vK, bH0, bHK] = IterateDynamics(pattern,W,biasV,biasH,K);

            tmp1 = tanh(bH0) * v0';
            tmp2 = tanh(bHK) * vK';
            deltaW{mu} = learningRate * (tmp1 - tmp2);
            deltaBiasH{mu} = -learningRate * (tanh(bH0)-tanh(bHK))';
            deltaBiasV{mu} = -learningRate * (v0 - vK);
        end
```

```
        for mu = p0
            W = W + deltaW{mu};
            biasH = biasH + deltaBiasH{mu};
            biasV = biasV + deltaBiasV{mu};
        end
        PB = EstimateBoltzmannDistn(W,biasV,biasH,dat);
        DKL(iHiddenNeuron,T) = CalculateDKL(PB);

plot(1:numIterations,DKL(1,:),'b',1:numIterations,DKL(2,:),'r',1:numIterations,DKL(3,:
),'g',1:numIterations,DKL(4,:),'k');
        disp(strcat(num2str(iHiddenNeuron),'-',num2str(T),'---
',num2str(DKL(iHiddenNeuron,T))));
    end
    iHiddenNeuron = iHiddenNeuron + 1;
end
```

5.4 Code for Tic Tac Toe

```
function qTable = AddToQTable(qTable,boardState,initialEstimatedVal)
    append = true;
    for i = 1:size(qTable,2)
        if isequal(qTable{1,i} , boardState)
            append = false;
            break;
        end
    end
    if append == true
        qTable{1,size(qTable,2) + 1} = boardState;
        qTable{2,size(qTable,2)} =
InitializeRewardTable(boardState,initialEstimatedVal);
    end
end


function win = CheckForWin(boardState,player)
    win = 0;
    if boardState(1:3) == player
        win = 1;
    elseif boardState(4:6) == player
        win = 1;
    elseif boardState(7:9) == player
        win = 1;
    elseif boardState(1) == player && boardState(4) == player && boardState(7) ==
player
        win = 1;
    elseif boardState(2) == player && boardState(5) == player && boardState(8) ==
player
        win = 1;
    elseif boardState(3) == player && boardState(6) == player && boardState(9) ==
player
        win = 1;
    elseif boardState(1) == player && boardState(5) == player && boardState(9) ==
player
        win = 1;
    elseif boardState(3) == player && boardState(5) == player && boardState(7) ==
player
        win = 1;
    end
end

function rewardTable = InitializeRewardTable(boardState,initialEstimatedVal)
    rewardTable = ones(1,9) * initialEstimatedVal;
    for i = 1:9
        if boardState(i) ~= 0
            rewardTable(i) = NaN;
        end
    end
end


function [boardState, iQTable] = MakeMove(qTable,iQTable,boardState,player,epsilon)
    tmpTable = qTable{2,iQTable};
    if rand < epsilon
        validChoice = 0;
        while validChoice == 0
            choice = randi(length(tmpTable));
            if ~isnan(tmpTable(choice))
                boardState(choice) = player;
                break
```

```
            end
        end
    else
        [maxVal, choice] = max(tmpTable);
        boardState(choice) = player;
    end
end

clear all
initialBoardState = [0,0,0,0,0,0,0,0,0];
epsilon0 = .5;
epsilonFin = .02;
playerO = 1;
playerX = -1;
initialEstimatedVal = .7;

playerOWins = 0;
playerXWins = 0;
draws = 0;

qTableX = cell(2,1); % first row is board, 2nd is expected values\
qTableO = cell(2,1);
boardState = initialBoardState;
qTableX{1,1} = initialBoardState;
qTableX{2,1} = initialEstimatedVal * ones(1,9);
qTableO{1,1} = initialBoardState;
qTableO{2,1} = initialEstimatedVal * ones(1,9);


for iGame = 1:100000
    % initialize game
    gameOnGoing = 1;
    boardState = initialBoardState;
    iQTableO = 1; iQTableX = 1;
    while gameOnGoing == 1
        [boardState iQTableO] = MakeMove(qTableO,iQTableO,boardState,playerO);
        gameOnGoing = CheckForWin(boardState,playerO);
        if gameOnGoing == 0
            qTableO = UpdateQTable(qTableO);
            qTableX = UpdateQTable(qTableX);
        else
            boardState = MakeMove(boardState,playerX);
            gameOnGoing = CheckForWin(boardState,playerX);
            if gameOnGoing == 0
                qTableO = UPdateQTable(qTableO);
                qTableX = UpdateQTable(qTableX);
            end
        end
    end
end
```