University of Ljubljana, Faculty of Electrical Engineering

Intelligent Decision Support Systems

Report Homework 4

Neural Networks and PSO learning

Ilaria, Anna Lauriola

May 3, 2019

1. Introduction

1.1 Problem Description

Within this assignment we will build system for breast cancer classification. In the data.mat there are values for certain characteristics obtained from breast scans. Based on these values one can classify cancer as benign or malign. The first column in the data matrix denotes the tumor (2-malign, 4-benign).

2. Implementation and Methods

We used Matlab to implement NN and PSO methods in order to solve the problem. We took inspiration from the slides of chapter 10 of Intelligent decision support systems and wrote e solved the problem by hand before writing the Matlab code.

2.1 Artiftcial Neural Network

2.1.1 Basic Theoretical Background

[1].The study of Artificial Neural Networks (ANN) was inspired by attempts to simulate biological neural systems. The human brain consists primarily of nerve cells called neurons, linked together with other neurons via strands of fiber called axons. Axons are used to transmit nerve impulses from one neuron to another whenever the neurons are stimulated. A neuron is connected to the axons of other neurons via dendrites, which are extensions from the cell body of the neuron. The contact point between a dendrite and an axon is called a synapse. Neurologists have discovered that the human brain learns by changing the strength of the synaptic connection between neurons upon repeated stimulation by the same impulse. Analogous to human brain structure, an ANN is composed of an interconnected assembly of nodes and directed links.
The network may contain several intermediary layers between its input and output layers. Such intermediary layers are called hidden layers and the nodes embedded in these layers are called hidden nodes. The resulting structure is known as a multilayer neural network. In a feed-forward neural network, the nodes in one layer are connected only to the nodes in the next layer. The number of neurons in the hidden layer depends on a problem and is set by the user.
The input neurons have no transfer functions and weights – are direct feed-trough.
The output of the neuron is the value of function f(u). The value of u is a linear combination of neuron's inputs (x1*w1i + x2*w2i+...+xN*wNi+wi0).
The learning procedure is a procedure that optimizes the weights so that for the given pairs of inputs and outputs the output of the neural network is the same (or almost the same) as the given outputs.
The network may use types of activation functions. Examples of other activation functions include linear, sigmoid (logistic),sign function and hyperbolic tangent functions. These

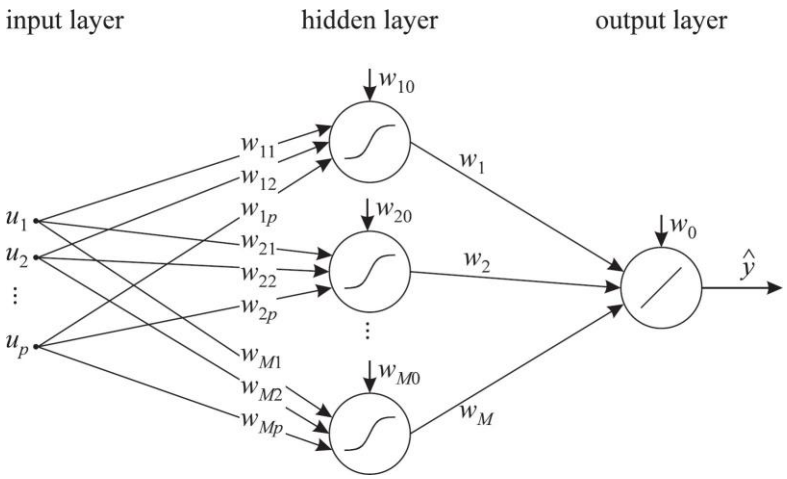

Figure 1: **Multilayer perceptron network**

activation functions allow the hidden and output nodes to produce output values that are nonlinear in their input parameters.

2.1.2 Matlab Code

1) The first thing to do is split the data into learning and test data set.

```
load('data.mat');
T = data(:,1);

preclass = cell(2,1) ;
class = [2 4] ;
fori = 1:2
   preclass{i} = data(T==class(i),:) ;
end

%TrainingData
train_class_2 = preclass{1}(1:148,:);
train_class_4 = preclass{2}(1:256,:);

%MatrixofInputs
x = [train_class_2(:,2:31); train_class_4(:,2:31)]';
x = normalize(x,'range');

%Matrix/VectorofTargetOutputs.
t = [train_class_2(:,1); train_class_4(:,1)]';
t = normalize(t,'range');

%TestingData
test_class_2 = preclass{1}(149:212,:);
```

```
test_class_4 = preclass{2}(257:357,:);
test_x = [test_class_2(:,2:31); test_class_4(:,2:31)]';
test_x = normalize(test_x,'range');
test_t = [test_class_2(:,1); test_class_4(:,1)]';
test_t = normalize(test_t,'range');
```

2) Preparing and Traing NN.

The neural network structure is already implemented in Matlab. The neural network is generated as: **net = newff(x,t,n,'tansig','tansig')**;

With this the feedforward neural network is created with tansig transfer functions in the hidden and output layer. The parameter n defines the number of neurons in the hidden layer. The x is the matrix of inputs and t is the matrix/vector of target outputs.

Vector t and matrix x are row based (each variable is represented in a row not in a column). The initialization of NN initializes the weights and sets the proper dimensions of input and output layer.

```
%%PreparingandTraingNN.
n = 2;%theNumberofNeuronsintheHiddenLayeror[32]for2...
    hiddenlayerswith3and2neuronsineachlayer.
%Theneuralnetworkisgeneratedas:
net = newff(x,t,2,{'tansig','tansig'});%Activationfunctions:tansig...
    (Sigmoidfun)orpurelin.
%weneedtocomparetheresultsifweusepurelininsteadoftansig...
    because
%weareusingtansigfunction

%%TheNNisinitializedwith:
net = init(net);
net = configure(net,x,t);
%view(net);#itshowsthenet

%NNtraining:
net = train(net,x,t);
```

3) Calculate the MSE of model with both training and test data

```
%%CalculatetheMSEofmodelwithbothtrainingandtestdata

%Theoutputofneuralnetworkwiththetestingdata.
est_t = net(x);

%TheMSEcriterioncanbecalculatedas:
mse_calc = sum((est_t - t).^2) / length(est_t);
figure()
plotconfusion(t,est_t)
title("Training Confusion Matrix")

%CalculatetheMSEofmodelwithtestingdata
```

```
13
14 test_error = test_t - net(test_x);
15 test_mse_calc = sum((test_error).^2) / length(test_t);
16 figure()
17 plotconfusion(test_t,net(test_x));
18 title("Test Confusion Matrix")
```

2.2 Particle Swarm Optimization

2.2.1 Basic Theoretical Background

[2].Particle swarm optimization (PSO) algorithm is a stochastic optimization technique based on swarm, which was proposed by Eberhart and Kennedy (1995) and Kennedy and Eber- hart(1995). PSO algorithm simulates animal's social behavior, including insects, herds, birds and shes. These swarms conform a cooperative way to nd food, and each member in the swarms keeps changing the search pattern according to the learning experiences of its own and other members.Main design idea of the PSO algorithm is closely related to two researches: one is evolutionary algorithm, just like evolutionary algorithm; PSO also uses a swarm mode which makes it to simultaneously search large region in the solution space of the optimized objective function. The other is artificial life, namely it studies the artificial systems with life characteristics.
In studying the behavior of social animals with the artificial life theory, for how to construct the swarm artificial life systems with cooperative behavior by computer, Millonas proposed the basic principles (van den Bergh 2001):

- **Proximity**: the swarm should be able to carry out simple space and time computations.
- **Quality**: the swarm should be able to sense the quality change in the environment and response it.
- **Diverse response**: the swarm should not limit its way to get the resources in a narrow scope.
- **Stability**: the swarm should not change its behavior model with every environmental change
- **Adaptability**: the swarm should change its behavior model when this change is worthy.

[3]. The PSO algorithm maintains multiple potential solutions at one time. During each iteration of the algorithm, each solution is evaluated by an objective function to determine its fitness. Each solution is represented by a particle in the fitness landscape (search space). The particles "fly" or "swarm" through the search space to find the maximum value returned by the objective function.
Each particle maintains:

- Position in the search space (solution and fitness)
- Velocity
- Individual best position

In addition, the swarm maintains its global best position.
The PSO algorithm consists of just three steps:

- Evaluate fitness of each particle
- Update individual and global bests
- Update velocity and position of each particle

These steps are repeated until some stopping condition is met.

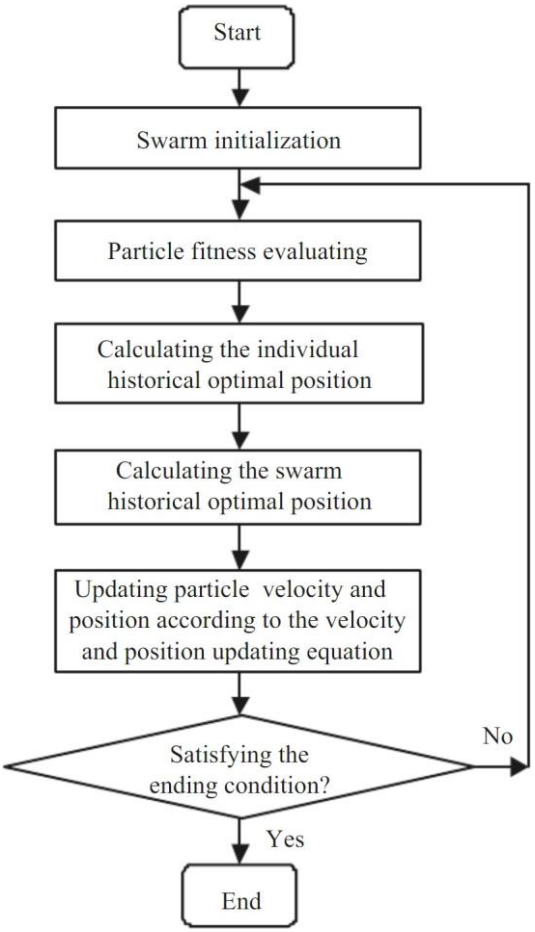

Figure 2: **Flowchart of the particle swarm optimization algorithm**

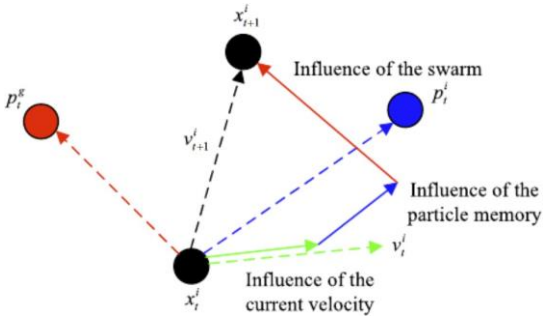

Figure 3: **Iteration scheme of the particles**

2.2.2 Matlab Code

1) PSO Algorithm

```
function[x, err] = PSO(CostFunction, nVar)
%CostFunction-MSEcriterion
%nVar=NumberofDecisionVariables.

VarSize = [1 nVar];%SizeofDecisionVariablesMatrix

VarMin = -7;%LowerBoundofVariables
VarMax = 7;%UpperBoundofVariables

%%PSOParameters
MaxIter = 100;%MaximumNumberofIterations
nPop = 50;%Populationsize(SwarmSize)

%PSOParameters
w = 0.7;%InertiaWeight
c1 = 2.0;%PersonalLearningCoefficient
c2 = 2.0;%GlobalLearningCoefficient

%VelocityLimits
VelMax = 0.1*(VarMax - VarMin);
VelMin = -VelMax;

%%Initializtion
init_particle.Position = [];
init_particle.Cost = [];
init_particle.Velocity = [];
init_particle.Best.Position = [];
init_particle.Best.Cost = [];

particle = repmat(init_particle, nPop, 1);

GlobalBest.Cost = inf;

fori = 1:nPop
```

```
35      %InitializePosition
36      particle(i).Position = unifrnd(VarMin, VarMax, VarSize);
37
38      %InitializeVelocity
39      particle(i).Velocity = zeros(VarSize);
40
41      %Evaluation
42      particle(i).Cost = CostFunction(particle(i).Position);
43
44      %UpdatePersonalBest
45      particle(i).Best.Position = particle(i).Position;
46      particle(i).Best.Cost = particle(i).Cost;
47
48      %UpdateGlobalbest
49      ifparticle(i).Best.Cost < GlobalBest.Cost
50          GlobalBest = particle(i).Best;
51      end
52  end
53
54  BestCost = zeros(MaxIter, 1);
55
56  %%PSOAlgorithm
57   forit = 1:MaxIter
58      fori = 1:nPop
59          %UpdateVelocity
60          particle(i).Velocity = w*particle(i).Velocity...
61              + c1*rand(VarSize).*(particle(i).Best.Position -...
                    particle(i).Position)...
62              + c2*rand(VarSize).*(GlobalBest.Position -...
                    particle(i).Position);
63
64          %ApplyVelocityLimits
65          particle(i).Velocity = max(particle(i).Velocity, VelMin);
66          particle(i).Velocity = min(particle(i).Velocity, VelMax);
67
68          %UpdatePosition
69          particle(i).Position = particle(i).Position +...
                particle(i).Velocity;
70
71          %VelocityMirrorEffect
72          IsOutside=(particle(i).Position < VarMin |...
                particle(i).Position > VarMax);
73          particle(i).Velocity(IsOutside) =...
                -particle(i).Velocity(IsOutside);
74
75          %ApplyPositionLimits
76          particle(i).Position = max(particle(i).Position, VarMin);
77          particle(i).Position = min(particle(i).Position, VarMax);
78
79          %Evaluation
80          particle(i).Cost = CostFunction(particle(i).Position);
81
82          %UpdatePersonalBest
```

```
        ifparticle(i).Cost < particle(i).Best.Cost
            particle(i).Best.Position = particle(i).Position;
            particle(i).Best.Cost = particle(i).Cost;

            %UpdateGlobaleBest
            ifparticle(i).Best.Cost < GlobalBest.Cost
                GlobalBest = particle(i).Best;
            end
        end

    end

    BestCost(it) = GlobalBest.Cost;

    disp(['Iteration'num2str(it)':BestCost='...
        num2str(BestCost(it))]);

end

BestSol = GlobalBest;
x = BestSol.Position';
err = BestSol.Cost;

%%Results
figure;
semilogy(BestCost,'LineWidth', 2);
xlabel('Iteration');
ylabel('BestCost');
grid on;
```

2) MSE Function

```
functionMSE calc = MSE( wb, net, inputs, targets)

%wbistheweightsandbiasesrowvector
%Itmustbetransposedwhentransferringtheweightsandbiasesto...
    thenetworknet.

 net = setwb(net, wb');

 yest = net(inputs);

 MSE_calc = sum((yest-targets).^2) / length(yest);
```

3) Apply PSO to NN

```
clear all;
clc;
%formatcompact;

%%Datapreparation:Splitintotraindataandtestdata.
load('data.mat');
```

```
T = data(:,1);

preclass = cell(2,1) ;
class = [2 4] ;
fori = 1:2
     preclass{i} = data(T==class(i),:) ;
end

%TrainingData
 train_class_2 = preclass{1}(1:148,:);
 train_class_4 = preclass{2}(1:256,:);

%MatrixofInputs
inputs = [train_class_2(:,2:31); train_class_4(:,2:31)]';
inputs = normalize(inputs,'range');

%Matrix/VectorofTargetOutputs.
targets = [train_class_2(:,1); train_class_4(:,1)]';
targets = normalize(targets,'range');

%TestingData
 test_class_2 = preclass{1}(149:212,:);
 test_class_4 = preclass{2}(257:357,:);
test_x = [test_class_2(:,2:31); test_class_4(:,2:31)]';
test_x = normalize(test_x,'range');
test_t = [test_class_2(:,1); test_class_4(:,1)]';
test_t = normalize(test_t,'range');

[numVar, numSamp] = size(inputs);
 %%InitializetheNN
 %%Numberofneurons
n = 2;

%Theneuralnetworkisgeneratedas:
net = newff(inputs,targets,2,{'tansig','tansig'});%Activation...
      functions:tansig(Sigmoidfun)orpurelin.

%TheNNisinitializedwith:
%net=init(net);
net = configure(net,inputs,targets);
view(net);

%%ApplyPSOtoNN-Training
%calculatesMSE
h = @(wb) MSE(wb, net, inputs, targets);%MSE-costfunction...
      independentonweight-bias.

%runningtheparticleswarmoptimizationalgorithmwithdesiredoptions
[sol, err_ga] = PSO(h, (numVar+1)*n +n + 1);
net = setwb(net, sol');
%
%errorMSEPSOoptimizedNN
error = targets - net(inputs);
```

```
MSE_calc = sum((error).^2) / length(targets)
figure()
plotconfusion(targets,net(inputs))
title("Training Confusion Matrix PSO")

%%Testing
test_error = test_t - net(test_x);
test_MSE = sum((test_error).^2) / length(test_t)

figure()
plotconfusion(test_t,net(test_x));
title("Test Confusion Matrix PSO")
```

3. Discussion

3.1 Neural Network

Running the Matlab code for NN using the tan-Sigmoid activation function, we got the Best Validation Performance plot.

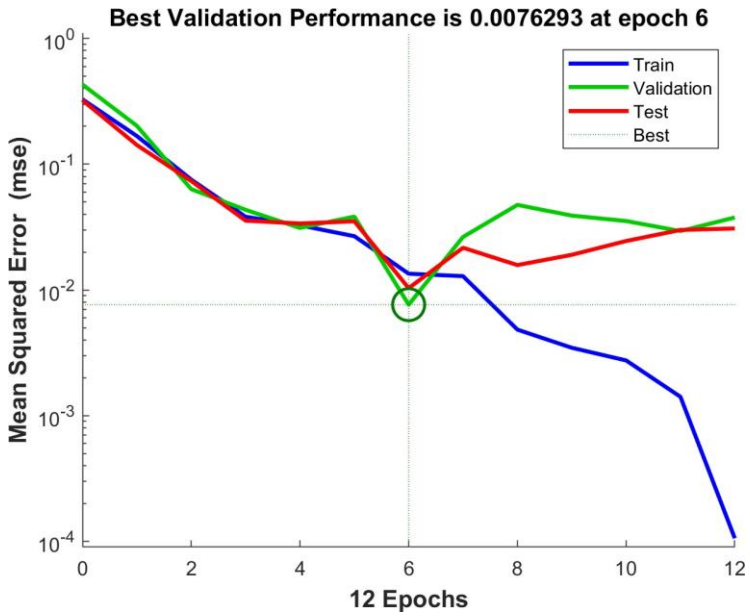

Figure 4: **Best Validation Performance**

Generally, the error reduces after more epochs of training, but might start to increase on the validation data set as the network starts overfitting the training data. In the default setup, the training stops after six consecutive increases in validation error, and the best performance is taken from the epoch with the lowest validation error.

We also plotted the Confusion Matrix for both training and test set in ftgure5and ftgure6.

On the confusion matrix plot, the rows correspond to the predicted class (Output Class) and the columns correspond to the true class (Target Class). The diagonal cells correspond to observations that are correctly classified. The off-diagonal cells correspond to incorrectly classified observations. Both the number of observations and the percentage of the total number of observations are shown in each cell. The column on the far right of the plot shows the percentages of all the examples predicted to belong to each class that are correctly and incorrectly classified. These metrics are often called the precision (or positive predictive value) and false discovery rate, respectively. The row at the bottom of the plot shows the percentages of all the examples belonging to each class that are correctly and incorrectly classified. These metrics are often called the recall (or true positive rate) and false negative rate, respectively. The cell in the bottom right of the plot shows the overall accuracy.

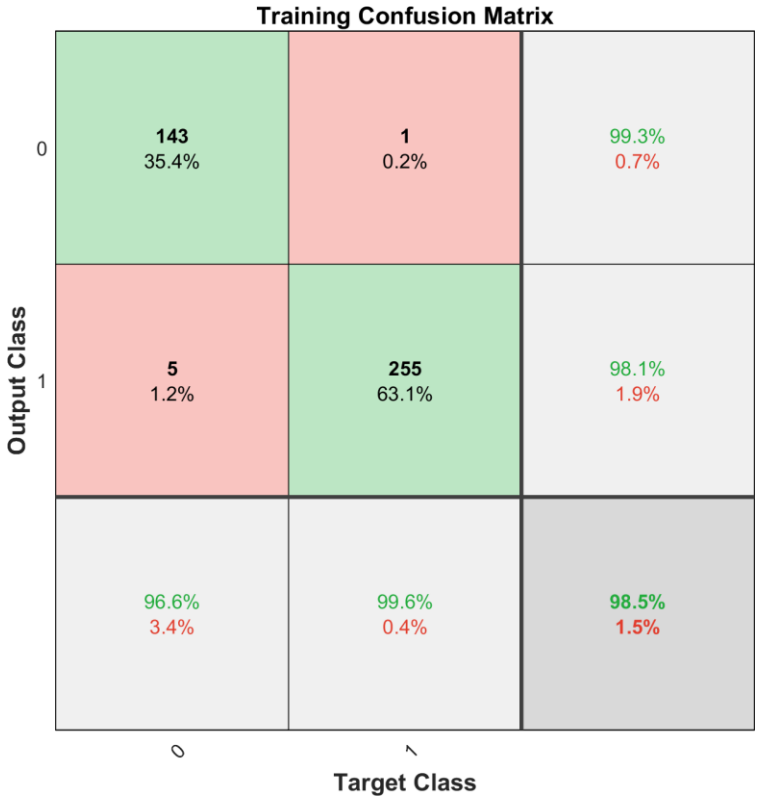

Figure 5: **NN Training Confusion Matrix**

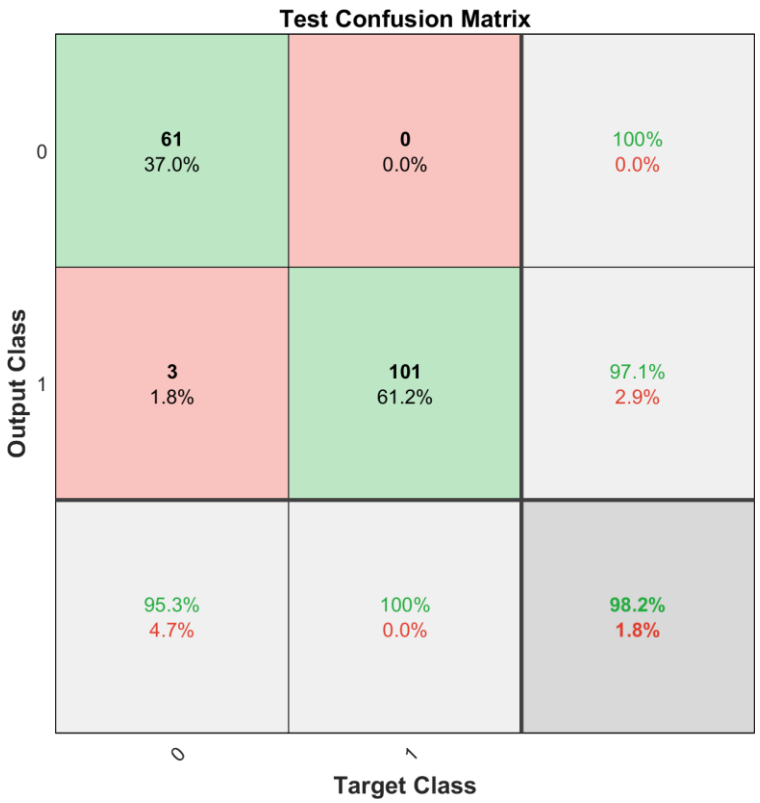

Figure 6: **NN Test Confusion Matrix**

We repeated the same measurements using purelin (linear function) as activation function and see the matrices in figure7 and figure8.

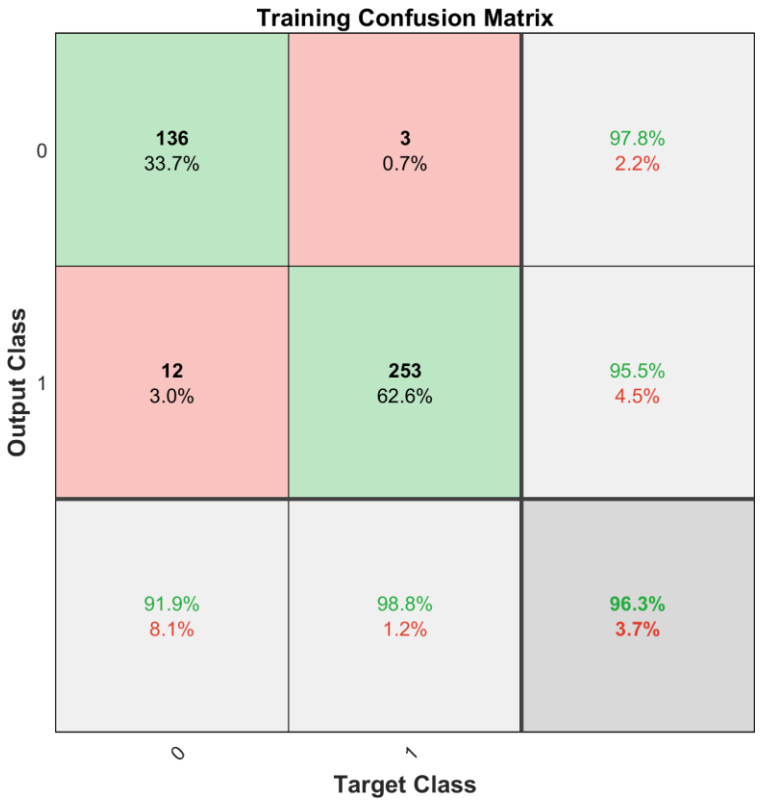

Figure 7: **NN Training Confusion Matrix with purelin activation function**

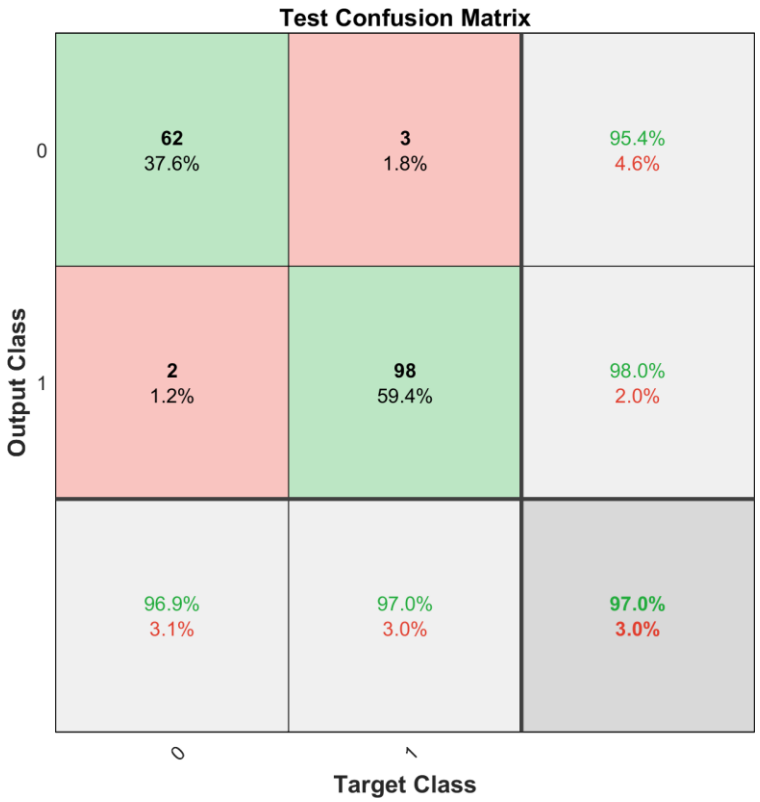

Figure 8: **NN Test Confusion Matrix with purelin activation function**

We can see that, even if the precision and the recall are lower than the ones we got with tansig activation function, the values are still quite good.

3.1.1 PSO

Running the PSO algorithm with weight = 1 we got the following confusion matrices in figure9 and figure10.

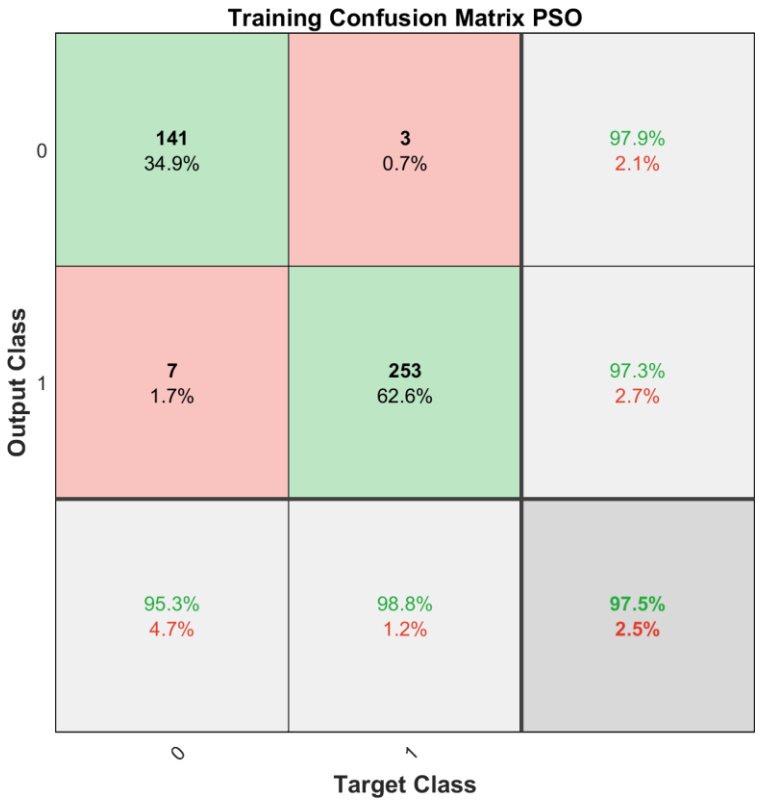

Figure 9: **PSO Training Confusion Matrix**

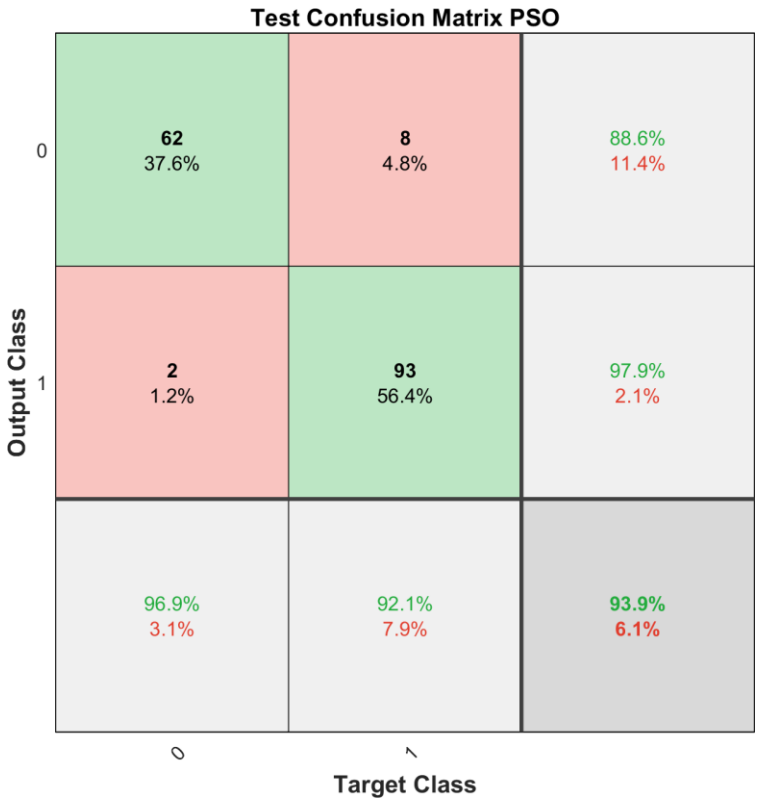

Figure 10: **PSO Test Confusion Matrix**

If we set the inertial weight = 0.70 and we perform the weighted PSO we got the matrices in figure11 and and figure12

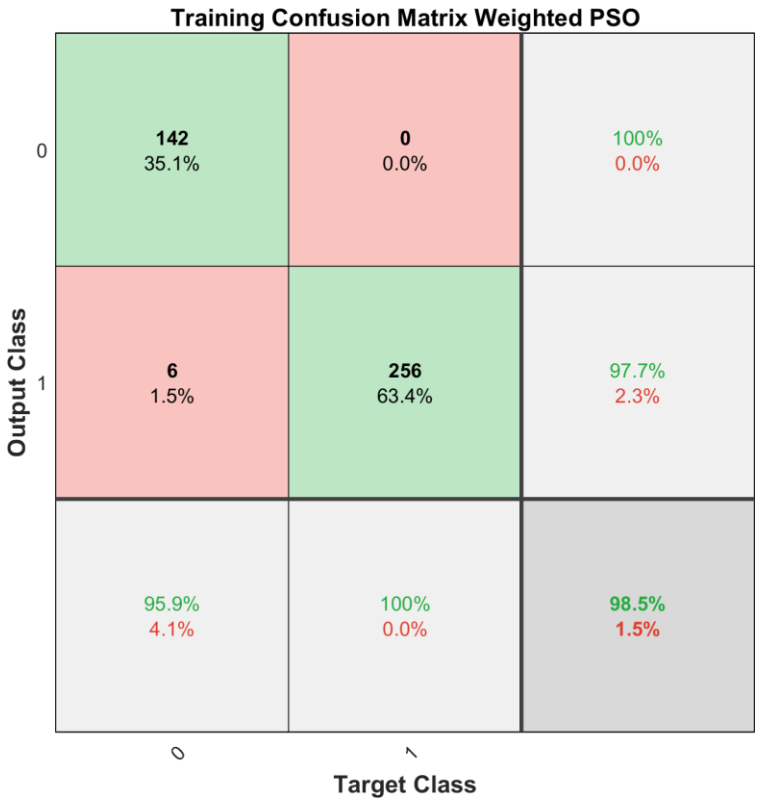

Figure 11: **Weighted PSO Test Confusion Matrix**

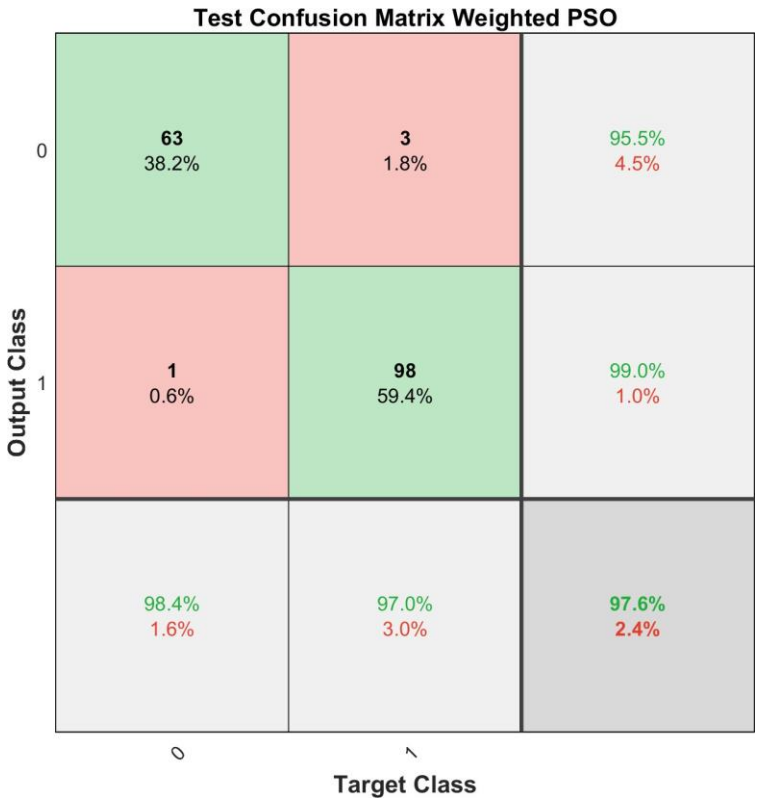

Figure 12: **Weighted PSO Test Confusion Matrix**

As we can see, the values are better in the Weighted PSO case.

4. Conclusion

After doing all these experiments we can compare the accuracy values in the three cases putting them all together in a table to better see the differences from the three methods used. We use in all three cases the tansig activation function. Comparisons are shown in figure13

	Training	Test
NN Matlab Toolbox	98.5%	98.2%
PSO	97.5%	93.9%
Weighted - PSO	98.5%	97.6%

Figure 13: **Comparison Table**

References

[1]. Pang-Ning Tan. (2013).Introduction to Data Mining.Pearson.PDF.

[2].Wang, Dongshu & Tan, Dapei & Liu, Lei. (2017). Particle swarm optimization algorithm: an overview. Soft Computing. 10.1007/s00500-016-2474-6.

[2]. Blondin James. Particle Swarm Optimization Applications in Parameterization of Classifiers. Armstrong Atlantic State University. PDF.