function TMLMVN.GetWeights(Kind: TLerning): Boolean;

Label

lProcessEnd,

lNextstep;

var

i, j, k, kk, LV, LV1, IC, Number\_Layers, Number\_Out\_Layer: Integer;

Number\_Neurons, Number\_Neurons\_1 : array of Integer;

/ / Will be stored in Number\_Neurons number of neurons at each level of the network,  
 / / But in Number\_Neurons\_1 same amount kmenshennye 1

Norma\_Layer : array of Extended;

Tmp\_Sum, Tmp, Tmp1, global\_error: TComplexExt;

Exit, LocalExit: Boolean;

a\_num, vvv, vvv1, SecCount2: Integer; // Number of sectors

Temp\_Norma, Temp\_Norma1, Temp\_WSum\_Norma, Temp\_RMSE: Extended;

LEARNING, tmpCriteria : Boolean;

TrueVector, EstVector: Array Of Double;

TrueVectorIJ, EstVectorIJ: Array Of Array Of Integer;

MinErrorRateA: Array of Double;

begin

Application.ProcessMessages;

Result := false;

fNextStep := False;

if \_LocalLength > \_Length then \_LocalLength := \_Length;

if odd(Sectors.Count) // calculation of the number equal to half the number of sectors

then SecCount2:=(Sectors.Count+1) div 2

else SecCount2:=Sectors.Count div 2;

SectorSize:= 2\*pi/Sectors.Count; // Determinization of the angular size of one sector

try

\_ErrorRate := 100; // 100% at the begining

\_MinErrorRate := 100; // 100% at the begining

\_RMSE:=0; // 0 at the begining

Temp\_RMSE:=0; // 0 at the begining

\_MinRMSE := 100000.0; // The initial value \_MinRMSE = 1000000

ExtremalExit := False;

// The total number of levels equal to the number of hidden levels + 2 (+ input and output)

Number\_Layers := HiddenLayers+2;

Number\_Out\_Layer := HiddenLayers+1; // index of output level (the numbering begins at 0!)

// count neuron quantity of each level and store in array Number\_Neurons

SetLength(Number\_Neurons, HiddenLayers+2);

SetLength(Number\_Neurons\_1, HiddenLayers+2);

for i:=0 to HiddenLayers+1 do

begin

Number\_Neurons[i] := System.Length(Neurons[i]);

Number\_Neurons\_1[i] := Number\_Neurons[i] - 1;

end;

SetLength(Norma\_Layer, Number\_Layers);

// nuzhno vystavit korrektniy ClearenceInterval dlja kazhdogo neirona

// nuzhno vystavit correction ClearenceInterval dlja kazhdogo neuron

// na vyodnom urovne

// na vyodnom urovne

for i:=0 to Number\_Neurons\_1[Number\_Out\_Layer] do

if Neurons[HiddenLayers+1][i].ClearenceInterval<0 then

Neurons[HiddenLayers+1][i].ClearenceInterval := ClearenceInterval;

// Вариант подсчета нормирующих множителей, равных для нейронов данного уровня

// 1/(число всех нейронов на предшествующем уровне + 1) (1-данный нейрон)

Norma\_Layer[0]:=1; // Нулевому уровню ничего не предшествует

for i:= 1 to HiddenLayers+1 do

Norma\_Layer[i] := 1/(Number\_Neurons[i-1]+1);

SetLength(TrueVector, Number\_Neurons[Number\_Out\_Layer]);

SetLength(EstVector, Number\_Neurons[Number\_Out\_Layer]);

SetLength(TrueVectorIJ, Number\_Neurons[Number\_Out\_Layer], \_LocalLength);

SetLength(EstVectorIJ, Number\_Neurons[Number\_Out\_Layer], \_LocalLength);

SetLength(MinErrorRateA, Number\_Neurons[Number\_Out\_Layer]);

for i:=0 to System.Length(MinErrorRateA)-1 do MinErrorRateA[i] := 0;

\_Iteration := 0;

repeat

// Если остановка обучения осуществляется по проценту оставшихся нербученными наборов или

// по RMSE, то перед итерацией обучения вычисляется текцшая ошибка в смысле процента наборов,

// на которых нужно проводить обучение или RMSE. Обучение останавливается, если ошибка ниже

// соответствующего заданного порога

if (StoppingCriteria=sc\_ErrorRate) or (StoppingCriteria=sc\_RMSE) then

begin

IndexLine := 0;

Temp\_RMSE:=0;

while IndexLine<=\_LocalLength-1 do

begin

// Application.ProcessMessages;

CurrentNumber := IndexLine;

if ExtremalExit then goto lProcessEnd;

// for 0 level

for i:=0 to 0 do // zaglushka

for j:=0 to Number\_Neurons\_1[i] do

begin

{ считаем взвешенные сумму }

Tmp\_Sum.Re := Neurons[i][j].Weights[0].Re;

Tmp\_sum.Im := Neurons[i][j].Weights[0].Im;

LV := System.Length(Neurons[i][j].Weights)-1; // общее число весов у нейрона - 1

LV1 := LV-1; // вычитаем еще 1, чтобы вести счет от 0 в цикле

for k:=0 to LV1 do

begin

if InputsType = it\_Discrete then

tmp := Mult( Neurons[i][j].Weights[k+1],

Sectors.CurentSector[Variables[Order[IndexLine]].Vector[LV-1-k]])

else

tmp := Mult( Neurons[i][j].Weights[k+1],

InputsTableContinious[Order[IndexLine]][k]);

Tmp\_sum := Add(Tmp\_Sum, Tmp);

end;

Temp\_Norma := sqrt(sqr(Tmp\_Sum.Re)+sqr(Tmp\_Sum.Im));

Neurons[i][j].FSF := Tmp\_Sum;

if Temp\_Norma<>0 then

Neurons[i][j].W\_Sum := Mult(Tmp\_Sum,1/Temp\_Norma)

else

begin

Neurons[i][j].W\_Sum.Re := 1;

Neurons[i][j].W\_Sum.Im := 0;

end;

// Для нейронов входного уровня выход равен нормированному значению взвешенной суммы,

// т.е., для этих нейронов выход не выравнивается на границу сектора, а просто приводится

// к точке на единичной окружности путем нормировки взвешенной суммы на ее модуль

case KindOfOutput of

out\_Continious: Neurons[i][j].Output := Neurons[i][j].W\_Sum;

out\_Descrete: Neurons[i][j].Output := Sectors.CurentSector[C\_sign(Tmp\_Sum, Sectors.Count, SectorSize)];

end;

end;

// for hidden + last levels

for i:=1 to Number\_Out\_Layer do

for j:=0 to Number\_Neurons\_1[i] do

begin

{ считаем взвешенные сумму }

Tmp\_Sum.Re := Neurons[i][j].Weights[0].Re;

Tmp\_sum.Im := Neurons[i][j].Weights[0].Im;

LV := System.Length(Neurons[i][j].Weights)-1; // общее число весов у нейрона - 1

LV1 := LV-1; // вычитаем еще 1, чтобы вести счет от 0 в цикле

for k:=0 to LV1 do

begin

// внимание тут меняю Output на W\_Sum ----- Почему?

// Дима, почему? В случае дискретных выходов на входном и скрытых уровнях

// это будет неправильно, нужно именно Output

tmp := Mult( Neurons[i][j].Weights[k+1],

Neurons[i-1][k].Output{W\_Sum});

Tmp\_sum := Add(Tmp\_Sum, Tmp);

end;

Temp\_Norma := sqrt(sqr(Tmp\_Sum.Re)+sqr(Tmp\_Sum.Im));

Neurons[i][j].FSF := Tmp\_Sum;

if Temp\_Norma<>0 then

Neurons[i][j].W\_Sum := Mult(Tmp\_Sum,1/Temp\_Norma)

else

begin

Neurons[i][j].W\_Sum.Re := 1;

Neurons[i][j].W\_Sum.Im := 0;

end;

// Для нейронов скрытых уровней выход равен нормированному значению взвешенной суммы,

// т.е., для этих нейронов выход не выравнивается на границу сектора, а просто приводится

// к точке на единичной окружности путем нормировки взвешенной суммы на ее модуль

// Для нейрона выходного уровня выход же, наоборот, выравнивается

// на границу соответствующего сектора

if (i <> Number\_Out\_Layer) then

case KindOfOutput of

out\_Continious: Neurons[i][j].Output := Neurons[i][j].W\_Sum;

out\_Descrete: Neurons[i][j].Output := Sectors.CurentSector[C\_sign(Tmp\_Sum, Sectors.Count, SectorSize)];

end

else

begin

Neurons[i][j].Output := Sectors.CurentSector[C\_sign(Tmp\_Sum, Sectors.Count, SectorSize)];

TrueVectorIJ[j][Order[IndexLine]] := Neurons[i][j].Vector[Order[IndexLine]];

// EstVectorIJ[j][Order[IndexLine]] := C\_sign(Neurons[i][j].W\_Sum, Sectors.Count);

// Temp\_Norma1 := TrueVectorIJ[j][Order[IndexLine]]+Round(Neurons[i][j].ClearenceInterval) mod (Sectors.Count-1);

// определяем, правильный ли выход у данного выходного нейрона

if (StoppingCriteria=sc\_ErrorRate) // StoppingCriteria=sc\_RMSE,

// ветка для sc\_RMSE чуть ниже в else

then

begin

if (abs(TrueVectorIJ[j][Order[IndexLine]] - C\_sign(Neurons[i][j].W\_Sum, Sectors.Count, SectorSize)) <= Neurons[i][j].ClearenceInterval)

//or

// (abs(Temp\_Norma1 - C\_sign(Neurons[i][j].W\_Sum, Sectors.Count)) <= Neurons[i][j].ClearenceInterval)

then

EstVectorIJ[j][Order[IndexLine]] := 1 // correct value

else

EstVectorIJ[j][Order[IndexLine]] := 0; // error

end

else // StoppingCriteria=sc\_RMSE

// приращение квадратической ошибки

begin

vvv := abs(TrueVectorIJ[j][Order[IndexLine]]-C\_sign(Neurons[i][j].W\_Sum, Sectors.Count, SectorSize) );

if (vvv > SecCount2) // если текушая разность между ожидаемым и реальным выходом > половины числв секторов,

then vvv:=Sectors.Count-vvv; // то значит "прочкочили" 0-2pi и нужно привести рахность по mod

Temp\_RMSE:= Temp\_RMSE+sqr(vvv); // наращивание квадратической ошибки

end

end;

end;

inc(IndexLine);

end;

// определение ошибки (% или RMSE), минимальной ошибки и того, нужно ли обучаться

if (StoppingCriteria=sc\_ErrorRate) // StoppingCriteria=sc\_RMSE,

// ветка для sc\_RMSE чуть ниже в else

then

begin

Exit := True;

// Приращение vvv- счетчика числа наборов, на которых выход неправильный хотя бы

// для одного из выходных нейронов

vvv := 0;

for j:=0 to \_LocalLength-1 do // цикл по наборам

for i:=0 to Number\_Neurons\_1[Number\_Out\_Layer] do // цикл по выходным нейронам

if (EstVectorIJ[i][j]=0) // Если хотя бы для одного из выходных нейронов

then // выход не был равен ожидаемому, значит, на данном наборе

begin // происходило обучение и увеличиваем vvv (число ошибок)

Inc(vvv); // после чего выходим из цикла по выходным нейронам

break // и переходим к следующему набору

end;

\_ErrorRate := 100\*vvv/\_LocalLength; // Ошибка в процентах числа наборов на которых

// происходило обучение

\_MinErrorRate := min(\_MinErrorRate,\_ErrorRate); // вычисление текущей минимальной ошибки

Exit := Exit and (\_MinErrorRate <= ClearenceInterval); // Выход, если ошибка в процентах

// меньше заданной

if Exit then begin Result := true; goto lProcessEnd; end; // Выход

end

else // StoppingCriteria=sc\_RMSE

begin

Exit := True;

Temp\_RMSE:=Temp\_RMSE/Number\_Out\_Layer; // усреднение квадратической ошибки по количеству выходных нейронов

\_RMSE:=sqrt(Temp\_RMSE/\_LocalLength); // вычисление среднеквадратической ошибки

\_MinRMSE := min(\_MinRMSE,\_RMSE); // вычисление текущего минимального RMSE

Exit := Exit and (\_MinRMSE <= ClearenceInterval); // Выход, если ошибка по RMSE

// меньше заданной

if Exit then begin Result := true; goto lProcessEnd; end; // Выход

end

end; // конец if StoppingCriteria = sc\_ErrorRate or StoppingCriteria = sc\_RMSE then

Exit := True; // флаг выхода

Inc(\_Iteration);

IndexLine := 0;

\_NofN :=0; // \_Nofn - счетчик несовпадений на данной итерации

while IndexLine<=\_LocalLength-1 do

begin

// Application.ProcessMessages;

CurrentNumber := IndexLine;

if ExtremalExit then goto lProcessEnd;

if fNextStep then

begin

// Application.ProcessMessages;

inc(IndexLine);

fNextStep := False;

end;

// for 0 level

for i:=0 to 0 do // zaglushka

for j:=0 to Number\_Neurons\_1[i] do

begin

{ считаем взвешенные сумму }

Tmp\_Sum.Re := Neurons[i][j].Weights[0].Re;

Tmp\_sum.Im := Neurons[i][j].Weights[0].Im;

LV := System.Length(Neurons[i][j].Weights)-1; // общее число весов у нейрона - 1

LV1 := LV-1; // вычитаем еще 1, чтобы вести счет от 0 в цикле

for k:=0 to LV1 do

begin

if InputsType = it\_Discrete then

tmp := Mult( Neurons[i][j].Weights[k+1],

Sectors.CurentSector[Variables[Order[IndexLine]].Vector[LV-1-k]])

else

tmp := Mult( Neurons[i][j].Weights[k+1],

InputsTableContinious[Order[IndexLine]][k]);

Tmp\_sum := Add(Tmp\_Sum, Tmp);

end;

Temp\_Norma := sqrt(sqr(Tmp\_Sum.Re)+sqr(Tmp\_Sum.Im));

Neurons[i][j].FSF := Tmp\_Sum;

if Temp\_Norma<>0 then

Neurons[i][j].W\_Sum := Mult(Tmp\_Sum,1/Temp\_Norma)

else

begin

Neurons[i][j].W\_Sum.Re := 1;

Neurons[i][j].W\_Sum.Im := 0;

end;

// Для нейронов входного уровня выход равен нормированному значению взвешенной суммы,

// т.е., для этих нейронов выход не выравнивается на границу сектора, а просто приводится

// к точке на единичной окружности путем нормировки взвешенной суммы на ее модуль

case KindOfOutput of

out\_Continious: Neurons[i][j].Output := Neurons[i][j].W\_Sum;

out\_Descrete: Neurons[i][j].Output := Sectors.CurentSector[C\_sign(Tmp\_Sum, Sectors.Count, SectorSize)];

end;

end;

// for hidden + last levels

for i:=1 to Number\_Out\_Layer do

for j:=0 to Number\_Neurons\_1[i] do

begin

{ считаем взвешенные сумму }

Tmp\_Sum.Re := Neurons[i][j].Weights[0].Re;

Tmp\_sum.Im := Neurons[i][j].Weights[0].Im;

LV := System.Length(Neurons[i][j].Weights)-1; // общее число весов у нейрона - 1

LV1 := LV-1; // вычитаем еще 1, чтобы вести счет от 0 в цикле

for k:=0 to LV1 do

begin

// внимание тут меняю Output на W\_Sum ----- Почему?

// Дима, почему? В случае дискретных выходов на входном и скрытых уровнях

// это будет неправильно, нужно именно Output

tmp := Mult( Neurons[i][j].Weights[k+1],

Neurons[i-1][k].Output{W\_Sum});

Tmp\_sum := Add(Tmp\_Sum, Tmp);

end;

Temp\_Norma := sqrt(sqr(Tmp\_Sum.Re)+sqr(Tmp\_Sum.Im));

Neurons[i][j].FSF := Tmp\_Sum;

if Temp\_Norma<>0 then

Neurons[i][j].W\_Sum := Mult(Tmp\_Sum,1/Temp\_Norma)

else

begin

Neurons[i][j].W\_Sum.Re := 1;

Neurons[i][j].W\_Sum.Im := 0;

end;

// Для нейронов скрытых уровней выход равен нормированному значению взвешенной суммы,

// т.е., для этих нейронов выход не выравнивается на границу сектора, а просто приводится

// к точке на единичной окружности путем нормировки взвешенной суммы на ее модуль

// Для нейрона выходного уровня выход же, наоборот, выравнивается

// на границу соответствующего сектора

if (i <> Number\_Out\_Layer)

then

case KindOfOutput of

out\_Continious: Neurons[i][j].Output := Neurons[i][j].W\_Sum;

out\_Descrete: Neurons[i][j].Output := Sectors.CurentSector[C\_sign(Tmp\_Sum, Sectors.Count, SectorSize)];

end

else

begin

Neurons[i][j].Output := Sectors.CurentSector[C\_sign(Tmp\_Sum, Sectors.Count, SectorSize)];

TrueVector[j] := Neurons[i][j].Vector[Order[IndexLine]];

EstVector[j] := C\_sign(Neurons[i][j].W\_Sum, Sectors.Count, SectorSize);

end;

end;

{if InputsType=it\_Discrete then

//begin if CriteriaFunction(TrueVector,EstVector,StoppingCriteria)<=ClearenceInterval

// then LEARNING := False else LEARNING := True; end

LEARNING := not (CriteriaFunction(TrueVector,EstVector,StoppingCriteria)<=ClearenceInterval)

else

begin}

LEARNING := True; // Проверяем, нужно ли обучаться - если хоть на одном выходном нейроне

// ожидаемый выход отличается от реального больше, чем на MAX (clearence для данного нейрона)

for i:=0 to System.Length(Neurons[HiddenLayers+1])-1 do

// LEARNING := LEARNING and (CriteriaFunction(TrueVector[i],EstVector[i],sc\_MAX {StoppingCriteria})<=Neurons[HiddenLayers+1][i].ClearenceInterval);

LEARNING := LEARNING and (abs(TrueVector[i]-EstVector[i])<=Neurons[HiddenLayers+1][i].ClearenceInterval);

LEARNING := not LEARNING;

// end;

if LEARNING then

begin

// считаем ошибки

Exit := False; // флаг выхода. уже не выходим

Inc(\_NofN); // приращение количества несовпадений

// for the last level

for i:= Number\_Out\_Layer to Number\_Out\_Layer do // заглушка

for j:=0 to Number\_Neurons\_1[i] do

begin

// !!!!!!!!!!

// В следующей строке ОЧЕНЬ ВАЖНАЯ ПОПРАВКА: при вычислении ошибки выходного нейрона вычитать будем из

// ожидаемого значения нормированное значение реальной взвешенной суммы, т.е., не тот дискретный корень

// из единицы, которому равен выход выходного нейрона, а "непрерывный" корень из единицы, равный

// нормированному значению взвешенной суммы

tmp := Mult( Neurons[i][j].W\_Sum, -1);

global\_error := Add( Sectors.CurentSector[Neurons[i][j].Vector[Order[IndexLine]]], tmp);

// Следующие две строки - вычисление величины, обратной модулю взвешенной суммы на выходном нейроне

// и домножение ошибки выходного нейроона (глобальной ошибки сети) на эту величину.

// Включать эти две строки следует только для чисто многозначных функций !!!

// Для булевых функций их нужно отключать !!!

// Вычисление 1/|z|, где z - взвешенная сумма на выходном нейроне:

// Temp\_Norma:=1/sqrt(sqr(Neurons[HiddenLayers+1][0].FSF.Re)+sqr(Neurons[HiddenLayers+1][0].FSF.Im));

// Домножение ошибки на 1/|z|, где z - взвешенная сумма на выходном нейроне:

// if (Temp\_Norma > 1.0) then

// global\_error := Mult( global\_error, Temp\_Norma );

// Домножение ошибки на 1/(число всех нейронов на предшествующих уровнях)

global\_error := Mult( global\_error, Norma\_Layer[Number\_Out\_Layer] );

// global\_error := Mult( global\_error, 1/System.Length(Neurons[i]) );

// Окончательная ошибка на выходном нейроне:

Neurons[i][j].LocalError := global\_error;// last level error

end;

// Если уж нормировать веса, то лучше это делать перед и обучением и вычислением ошибок,

// т.е., именно в этом месте. Но с другой стороны от нормировки можно отказаться, поскольку

// существенным для нас является только то, что модуль выхода нейрона равен единице, т.е., выход

// всегда находится на единичной окружности (а он именно там и находится). То, чтобы модуль

// взвешенной суммы был равен единице - необязательное условие.

// А далее идет уже непосредственно вычисление ошибок для всех уровней, кроме выходного (для него вычеслено раньше)

for i:=HiddenLayers downto 0 do

for k:=0 to Number\_Neurons\_1[i] do

begin

Neurons[i][k].LocalError.Re := 0;

Neurons[i][k].LocalError.Im := 0;

// Вариант Backpropagation № 1 - распрстранение ошибок с нейронов предыдущего уровня

// с умножением на соответствующие обратные веса

for j:=0 to Number\_Neurons\_1[i+1] do

begin

// Здесь важное дополнение -

// при вычислении ошибок нейронов i-того уровня мы будем проводить домножение не на

// веса соответствующих нейронов (i+1)-го уровня, а на числа, сопряженные этим весам,

// причем нормированные на квадрат своего собственного модуля (обратные веса),

// отсюда и следующие три строки кода

Temp\_Norma:= (sqr(Neurons[i+1][j].Weights[k+1].Re)+sqr(Neurons[i+1][j].Weights[k+1].Im));

Tmp := Conjugated(Neurons[i+1][j].Weights[k+1]);

Tmp := Mult(Tmp,1/Temp\_Norma);

// Tmp := Mult( Tmp, 1/System.Length(Neurons[i+1]) );

Neurons[i][k].LocalError :=

Add( Neurons[i][k].LocalError,

Mult( (Neurons[i+1][j].LocalError), Tmp )

);

end;

// ВНИМАНИЕ ! В следующих двух строчках важное нововведение - окончательная ошибка для всех нейронов скрытых и

// входного уровней должна домножаться на 1/|z|, где z - текущее

// значение взвешенной суммы !!!

Temp\_Norma:=1/sqrt(sqr(Neurons[i][k].FSF.Re)+sqr(Neurons[i][k].FSF.Im)); {1/|z| - величина, обратная модулю взвегенной суммы на данном нейроне}

Neurons[i][k].LocalError := Mult(Neurons[i][k].LocalError, Temp\_Norma ); {Домножение ошибки данного нейрона на 1/|z|}

Neurons[i][k].LocalError := Mult(Neurons[i][k].LocalError, Norma\_Layer[i]) {Домножение ошибки на 1/(число всех нейронов на предшествующем уровне) }

// ???? чет непонятное уже. нудно будеть проверить

end;

// обучаем. for 0 level

IC := Number\_Neurons\_1[0];

for j:=0 to IC do

begin

Temp\_Norma := 1/(Neurons[0][j].Norma);

// ВНИМАНИЕ ! В следующих двух строчках важное нововведение - окончательная ошибка для всех нейронов скрытых и

// входного уровней должна домножаться на 1/|z|, где z - текущее

// значение взвешенной суммы !!!

LV := System.Length(Neurons[0][j].Weights)-1;

Temp\_Norma := Temp\_Norma/(LV+1); // нормировка на (число входов данного нейрона +1)

Tmp := Mult( Neurons[0][j].LocalError, Temp\_Norma); // нормировка ошибки

Neurons[0][j].Weights[0] :=

Add( Neurons[0][j].Weights[0], Tmp);

LV1:=LV-1;

for k:=0 to LV1 do

begin

if InputsType=it\_Discrete then

Tmp := Mult( Neurons[0][j].LocalError,

Conjugated(Sectors.CurentSector[Variables[Order[IndexLine]].Vector[LV-1-k]]))

else

Tmp := Mult( Neurons[0][j].LocalError,

Conjugated(InputsTableContinious[Order[IndexLine]][k]));

Tmp := Mult(Tmp, Temp\_Norma);

Neurons[0][j].Weights[k+1] :=

Add( Neurons[0][j].Weights[k+1], Tmp);

end;

end;

//-------------------------------------------------------------------

// Пересчитываем взвешенные суммы после коррекции весов для нулевого уровня

// for 0 level

for i:=0 to 0 do // zaglushka

for j:=0 to Number\_Neurons\_1[i] do

begin

{ считаем взвешенные сумму }

Tmp\_Sum.Re := Neurons[i][j].Weights[0].Re;

Tmp\_sum.Im := Neurons[i][j].Weights[0].Im;

LV := System.Length(Neurons[i][j].Weights)-1;

for k:=0 to LV-1 do

begin

if InputsType=it\_Discrete then

tmp :=

Mult( Neurons[i][j].Weights[k+1],

Sectors.CurentSector[Variables[Order[IndexLine]].Vector[LV-1-k]])

else

tmp :=

Mult( Neurons[i][j].Weights[k+1],

InputsTableContinious[Order[IndexLine]][k]);

Tmp\_sum := Add(Tmp\_Sum, Tmp);

end;

Temp\_Norma := sqrt(sqr(Tmp\_Sum.Re)+sqr(Tmp\_Sum.Im));

if Temp\_Norma<>0 then

Neurons[i][j].W\_Sum := Mult(Tmp\_Sum,1/Temp\_Norma)

else

begin

Neurons[i][j].W\_Sum.Re := 1;

Neurons[i][j].W\_Sum.Im := 0;

end;

case KindOfOutput of

out\_Continious: Neurons[i][j].Output := Neurons[i][j].W\_Sum;

out\_Descrete: Neurons[i][j].Output := Sectors.CurentSector[C\_sign(Tmp\_Sum, Sectors.Count, SectorSize)];

end;

end;

// Конец пересчета взвешенных сумм для нулевого уровня

//-------------------------------------------------------------------

// for ... other levels

for i:=1 to HiddenLayers+1 do

for k:=0 to System.Length(Neurons[i])-1 do

begin

Temp\_Norma := 1/(Neurons[i][k].Norma);

// Следующая строчка - это, чтобы можно было задавать другой нормировочный множитель,

// отличный от общего, для выходного нейрона сети,

// либо, если просто Temp\_Norma:= 1/(LV+1), то нормировочный множитель для любого нейрона равен

// 1/(число входов+1) - стандартная величина

LV := System.Length(Neurons[i][k].Weights)-1;

Temp\_Norma := Temp\_Norma/(LV+1); // нормировка на (число входов данного нейрона +1)

Tmp := Mult( Neurons[i][k].LocalError, Temp\_Norma); // нормировка ошибки

Neurons[i][k].Weights[0] :=

Add( Neurons[i][k].Weights[0], Tmp);

LV1:=LV-1;

for j:=0 to LV1 do

begin

Tmp := Mult( Neurons[i][k].LocalError, Temp\_Norma);

// обратить внимание какой Output. заменил на W\_Sum

Tmp := Mult( Tmp, Conjugated(Neurons[i-1][j].Output));

Neurons[i][k].Weights[j+1] :=

Add( Neurons[i][k].Weights[j+1], Tmp);

end;

//-----------------------------------

// Пересчет взвешенных сумм для нейронов только что обученного уровня (i-того), кроме последнего

if (i<>Hiddenlayers+1) then

begin

{ считаем взвешенные сумму }

Tmp\_Sum.Re := Neurons[i][k].Weights[0].Re;

Tmp\_sum.Im := Neurons[i][k].Weights[0].Im;

LV := System.Length(Neurons[i][k].Weights)-1; // общее число весов у нейрона - 1

LV1 := LV-1; // вычитаем еще 1, чтобы вести счет от 0 в цикле

for j:=0 to LV1 do

begin

tmp :=

Mult( Neurons[i][k].Weights[j+1],

Neurons[i-1][j].Output);

Tmp\_sum := Add(Tmp\_Sum, Tmp);

end;

Temp\_Norma := sqrt(sqr(Tmp\_Sum.Re)+sqr(Tmp\_Sum.Im));

Neurons[i][k].FSF := Tmp\_Sum;

if Temp\_Norma<>0 then

Neurons[i][k].W\_Sum := Mult(Tmp\_Sum,1/Temp\_Norma)

else

begin

Neurons[i][k].W\_Sum.Re := 1;

Neurons[i][k].W\_Sum.Im := 0;

end;

case KindOfOutput of

out\_Continious: Neurons[i][k].Output := Neurons[i][k].W\_Sum;

out\_Descrete: Neurons[i][k].Output := Sectors.CurentSector[C\_sign(Tmp\_Sum, Sectors.Count, SectorSize)];

end;

end;

// Конец пересчета взвешенных сумм

//-----------------------------------

end;

// Ну, со следующей строчкой понятно - можно включать, можно выключать. В принципе, когда она выключена,

// обучение идет быстрее, во всяком случае для булевого случая

inc(IndexLine);

end // конец if LEARNING then

else

inc(IndexLine);

// Application.ProcessMessages;

end; //~ end of while IndexLine<=\_LocalLength-1 do

// Текущая минимальная ошибка (по числу несовпадений) вычисляется и выводится в MLUBN в TFMLUBN.ExecuteTimer

(\*

if StoppingCriteria=sc\_ErrorRate then // Если остановка по проценту ошибок, то

begin // вычисляем процент и сравниваем с заданным ClearenceInterval

\_ErrorRate := 100\*\_NofN/\_LocalLength; // Ошибка в процентах числа наборов на которых

// происходило обучение

\_MinErrorRate := min(\_MinErrorRate,\_ErrorRate); // минимальная ошибка в % =min(старая минимальная ошибка%, текущая ошибка%)

Exit := (\_MinErrorRate <= ClearenceInterval); // Выход, если ошибка в процентах

// меньше заданной

end;

\*)

// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

// тут нужно вставить код для досчета rmse/mean/max

// если меньше положенного то конец обучения

// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Application.ProcessMessages;

until Exit;

Result := true;

lProcessEnd:;

Except

MessageDlg('Error in learn processing, please ckeck entered parameters and try again.', mtError, [mbOK], 0);

end

end;