3a Questão:

1. Curva de erro durante o aprendizado (no mínimo 50 ciclos)

ANEXO - Redes Neurais:

Perceptron:

**public** **class** Perceptron {

**private** **double**[] w;

**private** **double** alfa;

**private** **int** NP;

**public** Perceptron(**double** alfa, **int** NP,**double**[] ws) {

**this**.w = ws;

**this**.alfa = alfa;

**this**.NP = NP;

}

**public** **void** training(**double**[][] patterns, **double**[] d,**int** noPatterns){

**boolean** stopCodition = **false**;

**double** net = 0;

**double** y = 0;

**double** e;

**int** noAdjust = 0;

**double**[] x;

**int** cycle = 0;

**while**(!stopCodition){

noAdjust = 0;

**for**(**int** data=0;data < noPatterns;data++){

x = patterns[data];

net = 0;

**for**(**int** j=0; j <= NP ;j++){

net += w[j]\* x[j];

}

y = ( net >= 0 ) ? 1 : 0;

e = d[data] - y;

**if**( e != 0){

**for**(**int** j=0;j <= NP ;j++){

w[j] = w[j] + alfa \* e \* x[j];

}

}**else**{

noAdjust++;

}

}

//Parou de aprender

**if**(noAdjust==noPatterns){

stopCodition = **true**;

}

cycle++;

System.*out*.println("Cycle = " + cycle);

}

System.*out*.println("Finalized in cycle " + cycle);

}

**public** **double** evaluateResult(**double**[] dataInput, **int** numPatterns) {

**double** net = 0;

**double** y = 0;

**double**[] x;

x = dataInput;

net = 0;

**for**(**int** j=0; j <= numPatterns ;j++){

net += w[j]\* x[j];

}

y = ( net >= 0 ) ? 1 : 0;

**return** y;

}

}

Adaline:

**public** **class** Adaline {

**private** **double**[][] w;

**private** **double** alfa;

**private** **int** NP;

**private** **int** NN;

**private** **double**[] EMQ;

**public** Adaline(**double** alfa, **int** NP,**int** NN, **double**[][] w0) {

**this**.w = w0;

**this**.alfa = alfa;

**this**.NP = NP;

**this**.NN = NN;

}

**public** **double**[] getOutput(**double**[] x){

**double** net = 0.0;

**double**[] y = **new** **double**[NN];

**for**(**int** neuron=0; neuron < NN ;neuron++){

net = 0;

**for**(**int** j=0; j<= NP; j++){

net += w[neuron][j] \* x[j];

}

y[neuron] = 1.0/(1.0 + Math.*exp*(-net));

}

**return** y;

}

**public** **void** training(**double**[][] patterns, **double**[][] d,**int** numPatterns, **int** maxCycles){

**double** net = 0;

**double** y = 0;

**double** e;

**double**[] x;

**int** cycle = 0;

EMQ = **new** **double**[maxCycles];

**while**( cycle < maxCycles){

**for**(**int** data=0;data < numPatterns;data++){

x = patterns[data];

**for**(**int** neuron=0;neuron < NN;neuron++){

net = 0;

**for**(**int** j=0; j <= NP ;j++){

net += w[neuron][j]\* x[j];

}

//funcao sigmoide logistica

y = 1.0/(1.0+Math.*exp*(-net));

System.*out*.println("y= " + y);

e = d[data][neuron] - y;

EMQ[cycle] += Math.*pow*(e, 2.0);

**for**(**int** j=0;j <= NP ;j++){

w[neuron][j] = w[neuron][j] + alfa \* e \* x[j] \* y \* (1-y);

System.*out*.println("w" +neuron+""+j + "= " + w[neuron][j]);

}

}

}

EMQ[cycle] = (EMQ[cycle] / numPatterns);

System.*out*.println("Cycle = " + cycle + " EMQ = " + EMQ[cycle] + "\n");

cycle++;

}

System.*out*.println("Finalized in " + cycle + " cycles");

}

}