Alexei Alves de Souza - 398611
AP1 - ICA

->>>> Projeto 5.9

1) Execute o treinamento da rede Perceptron, por meio do algoritmo de aprendizagem backpropagation convencional, inicializando-se as matrizes de pesos com valores aleatórios apropriados. Utilize a função de ativação logística (sigmóide) para todos os neurônios, com taxa de aprendizado {h} de 0,1 e precisão {ε} de 10-6.

```
MLP = MLPClassifier(hidden_layer_sizes=15,activation = 'logistic', tol=1e-6,learning_rate_init=0.1,
                    max_iter=100000,momentum=0.0, solver='sgd')
```

```
start_time1 = time.time()
#------------------------
MLP.fit(train,ltrain)
#------------------------
end_time1 = time.time()

t_exec1 = end_time1 - start_time1
```

```
print('N de iteracoes p/ convergencia:\n',MLP.n_iter_,' iteracoes')
```

```
N de iteracoes p/ convergencia:
 17560  iteracoes
```

2) Efetue, em seguida, o treinamento da rede Perceptron por meio do algoritmo de aprendizagem backpropagation com momentum, utilizando as mesmas matrizes de pesos iniciais que foram usadas no item anterior. Adote também a função de ativação logística (sigmóide) para todos os neurônios, com taxa de aprendizado {h} de 0,1, fator de momentum {a} de 0,9 e precisão {ε} de 10-6.

momentum = 0.9, setamos o valor do momentum na inicialização para 0.9 ¶

```
MLP_2 = MLPClassifier(hidden_layer_sizes=15,activation = 'logistic',tol=1e-6,learning_rate_init=0.1,
                      max_iter=10000,momentum = 0.9,solver='sgd')
```

```
start_time2 = time.time()
#------------------------
MLP_2.fit(train,ltrain)
#------------------------
end_time2 = time.time()

t_exec2 = end_time2 - start_time2
```

```
print('N de iteracoes p/ convergencia:\n',MLP_2.n_iter_,' iteracoes')
```

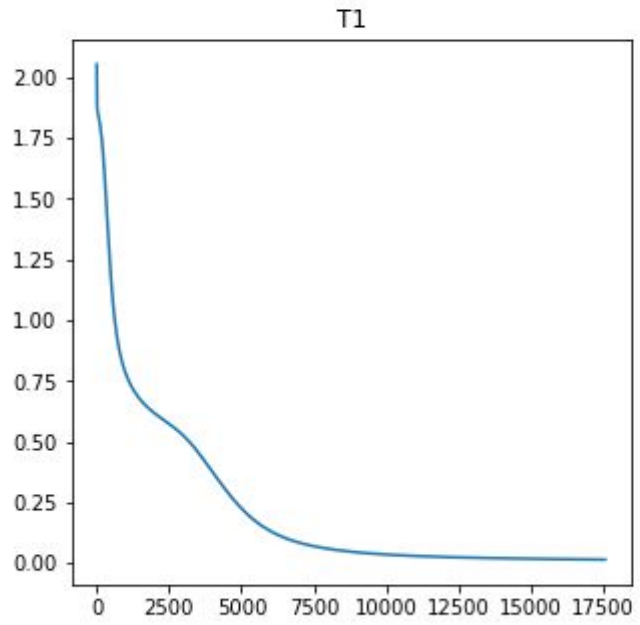
```
N de iteracoes p/ convergencia:
 3938  iteracoes
```

3) Para os dois treinamentos realizados nos itens anteriores, trace os respectivos gráficos dos valores de erro quadrático médio {EM} em função de cada época de treinamento. Imprima os dois gráficos numa mesma página de modo não

superpostos. Meça também o tempo de processamento envolvido com cada treinamento.

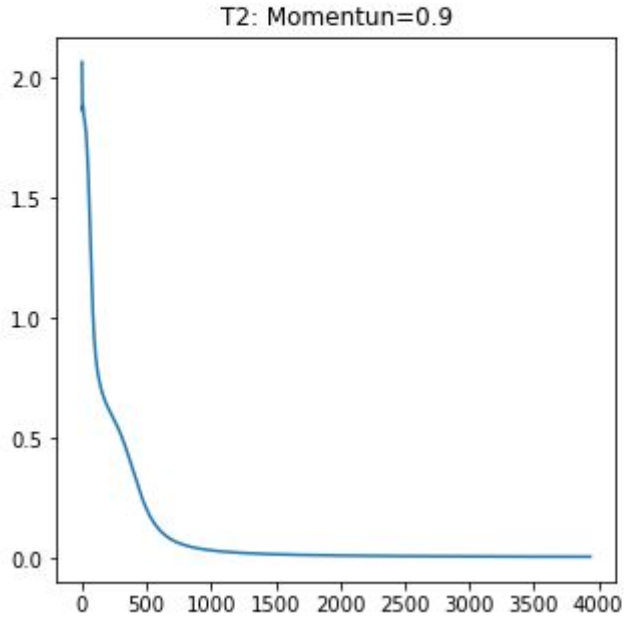
```
plt.figure(figsize = (5,5))
plt.plot(niter,loss)
plt.title('T1')
print('Tempo de execucao:', np.round(t_exec1,3),'segundos')
```

```
Tempo de execucao: 5.143 segundos
```



```
plt.figure(figsize = (5,5))
plt.plot(niter2,loss2)
plt.title('T2: Momentun=0.9')
print('Tempo de execucao:', np.round(t_exec2,3), 'segundos')
```

```
Tempo de execucao: 1.128 segundos
```



4) Dado que o problema se configura como um típico processo de classificação de padrões, implemente então a rotina que faz o pós-processamento das saídas fornecidas pela rede (valores reais) para números inteiros. Como sugestão, adote o critério de arredondamento simétrico, isto é:

Nesse codigo é recebido um vetor ou array de entradas, sendo este redimensionando para uma lista para análise elemento por elemento e classificação, e no final redimensional para a dimensão original.

```
def myPredict(inputs):
    outputs = []
    #Obtem as dimensoes
    x= inputs.shape[0]
    y= inputs.shape[1]

    #Redimensiona para analise elemento por elemento
    new_inputs = inputs.copy().reshape((x*y,))
    for i in range(len(new_inputs)):
        if new_inputs[i] >= 0.5:
            new_inputs[i] = 1
        else :
            new_inputs[i] = 0

    #Redimensiona o vetor para as dimensoes originais
    new_inputs = new_inputs.reshape((x,y))
    return new_inputs
```

5) Faça a validação da rede aplicando o conjunto de teste fornecido na tabela 5.5. Forneça a taxa de acertos (%) entre os valores desejados frente àquelas respostas fornecidas pela rede (após o pós-processamento) em relação a todos os padrões de teste.

```
: print(Mytable)
```

```
+--------------------------------+------------+------------+
|            x1 x2 x3            |   yreal    |  ypredito  |
+--------------------------------+------------+------------+
| [0.1187 0.2568 0.314  0.3037] | [1. 0. 0.] | [1. 0. 0.] |
| [0.726  0.75   0.7007 0.4953] | [0. 0. 1.] | [0. 0. 1.] |
| [0.7325 0.4761 0.3888 0.5683] | [0. 1. 0.] | [0. 1. 0.] |
| [0.5682 0.5683 0.5054 0.4426] | [0. 1. 0.] | [0. 1. 0.] |
+--------------------------------+------------+------------+
```

```
print('Precisao:\n',MLP_2.score(test,ltest)*100,'%')
```

```
Precisao:
 100.0 %
```