Universidade de São Paulo

SCC0270- Introdução a Redes Neurais

Projeto No.1

Pedro Morello Abbud

Número USP 8058718

Disciplina minsitrada por
Profa. Dra. Roseli Aparecida Francelin Romero

24 de Agosto de 2017

1 Introdução

Pretende-se neste documento explicar como foi implementado o perceptron de uma única camada, Adaline (Adaptive Linear Neuron), em python.

2 Implementação

Foi construída uma classe *Perceptron* que possuí os seguintes métodos: *read_samples*,*train*, *test*, *print_stats*. O funcionamento destas funções estão explicadas nas subseções a seguir.

2.1 read_samples

```
17      def read_samples(self,dir):
18          """Le todos os arquivos contidos na pasta dir, e armazena os valores
            ↪  esperados em xmatrix e sua estrututra de dados em xmatrix"""
19          expected=[]
20          xmatrix=[]
21          for root,dirs,files in os.walk(dir):
22              for file in files:
23                  with open(os.path.join(root,file),"r") as auto:
24                      expected.append(int(auto.readline().strip('\n')))
25                      a=[]
26                      for line in auto:
27                          a.append([int(n) for n in line.strip('\n').split(' ')])
28                      xmatrix.append(a)
29          return np.asarray(xmatrix),expected
```

Listing 1: Código da função *read_samples*

A função percorre o diretório relativo que foi especificado pela varíavel *dir* e salva em memória os resultados esperados de cada amostra de treinamento e os valores que definem a amostra. A formatação de cada amostra de treinamento e teste é composto pela primeira linha de header que é o resultado esperado e as demais linhas formam uma matriz 5x5 de números (-1 e 1) separados por espaços.

2.2 train

```
31      def train(self):
32          """Treina o perceptron"""
33          stout=[]
34          while self.expected!=stout:
35              self.loop=self.loop+1
36              stout=[]
37              for matriz in self.xmatrix:
38                  stout.append(np.vdot(matriz,self.weights)+self.bias)
39                  stout= [1 if a>0   else -1 for a in stout ]
40                  for i,calculated in enumerate(stout):
41                      if(calculated!=self.expected[i]):
42                          erro=self.expected[i]-calculated
43                          self.bias=erro*self.eta+self.bias
44                          for j,weight_line in enumerate(self.weights):
45                              for k,weight in enumerate(weight_line):
46                                  self.weights[j][k]=weight+erro*self.eta*self.xmatrix[i][j][k]
```

Listing 2: Código da função *train*

A função principal do código; é a função que é responsável pelo treinamento do perceptron. É a implementação praticamente literal do algoritmo da Regra Delta (LMS):

1. Iniciar os pesos sinápticos com valores randômicos pequenos ou iguais a zero.
2. Aplicar um padr'ao com seu respectivo valor esperado (t_j) e verificar a saída da rede (y_j).
3. Calcular o erro na saída: $E_j = t_j - y_j$
4. Se $E_j = 0$, volte ao passo 2. Caso contrário se $E_j \neq 0$, atualizar os pesos: $\Delta w_{ij} = \eta x_i E_j$
5. Volte ao passo 2.

2.3 test

```
49      def test(self,dir):
50          """Testa o perceptron com os arquivos contidos na pasta dir"""
51          matrix_train,expected_train= self.read_samples(dir)
52          stout=[]
53          for matriz in matrix_train:
54              stout.append(np.vdot(matriz,self.weights)+self.bias)
55              stout= [1 if a>0   else -1 for a in stout ]
56
57          print("Resultado esperado do teste:")
58          print(expected_train)
59          print("Resultado obtido no teste:")
60          print(stout)
61          return
```

Listing 3: Código da função *test*

Função responsável por validar a implementação do perceptron. Busca através da função *read_samples* todos os arquivos contidos na pasta especificada pelo atributo *dir* e os classifica conforme os pesos previamente calculados. A função então mostra na tela quais eram os valores esperados e quais foram os valores antecipados por ele.

2.4 print_stats

```
63      def print_stats(self):
64          """Mostra na tela informações relevantes do objeto """
65          print("==== Informações deste Perpectron ==== ")
66          print("Bias:")
67          print(self.bias)
68          print("Pesos:")
69          print(self.weights)
70          print("Loops até convergência:")
71          print(self.loop)
```

Listing 4: Código da função *print_stats*.

Função responsável por expor atributos da classe que foram calculados após o treinamento.

3 Resultados

Foi implementado com sucesso o Perceptron Adeline. O programa consegue classificar entradas de duas classes corretamente, desde que estas sejam linearmente separáveis e existe uma superfície de decisão com forma de hiperplano que separe as duas classes. Abaixo temos o output ao chamar as funções *test* e *print_stats*:

```
Resultado esperado do teste:
[1, 1, -1, -1]
Resultado obtido no teste:
[1, 1, -1, -1]
==== Informações deste Perpectron ====
Bias:
0.06
Pesos:
[[1 0 0 0 1]
 [1 0 0 0 1]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 1 0 0]]
Loops até convergência:
2
```

Listing 5: Output do programa perceptron.py.