EPC8: Redes com função de base radial

Guilherme Rocha Gonçalves[1], Rodolfo Coelho Dalapicola[1]

Universidade de São Paulo
São Carlos, Brasil

1 Introdução

1.1 Contextualização

Oitava atividade do estudo sobre Redes Neurais Artificiais desenvolvida durante o curso da disciplina SEL5712 Redes Neurais Artificiais, oferecida no primeiro semestre do ano de 2019 para os alunos do programa de mestrado em Engenharia Elétrica e oferecido pelo Professor Dr. Ivan Nunes da Silva.

1.2 Objetivo

Esta atividade têm como objetivo a criação de uma rede neural Perceptron Multicamada para solução de um problema deprevisão de funções, porém, a primeira camada da rede contém uma função de base radial. O trabalho desenvolverá toda a etapa de treinamento *backpropagation* e também do treinamento usando o *k-means*, além de usar a rede para calcular a saída de um subconjunto de teste.

2 Materiais e Métodos

2.1 Problema

Neste exercício proposto, é preciso determinar a quantidade de gasolina a ser injetado pelo sistema de injeção eletrônica em um veículo através de três variáveis de entrada. As topologias usadas serão redes com cinco, dez e quinze neurônios na primeira camada e um na saída. Assim, será possível comparar o desempenho de cada uma das redes. Será usada uma função de base radial na primeira camada e a função logística na camada de saída.

2.2 Bases de treino e de teste

O treinamento da rede neural tem dois estágios: o primeiro, que envolve o treinamento dos neurônios da camada intermediária e um segundo, onde ocorre o ajuste de pesos da camada de saída.

Na primeira etapa ocorre um treinamento auto organizado (não supervisionado) usando o *k-means*. Nessa etapa são calculadas as distâncias entre as amostras e os neurônios da primeira camada. A cada iteração do *k-means*, as coordenadas dos neurônios se aproximam cada vez mais do subconjunto de dados (*cluster*) com mais amostras próximas. A variância da vizinhança do neurônio entra na equação radial, assim como sua "posição"(em duas dimensões pode-se falar em coordenadas, mas nada impede que se existam n dimensões).

A segunda etapa cuida do ajuste dos pesos da segunda camada, usando a *Regra Delta Generalizada*, comuns aos PMCs.

A função de ativação da entrada é uma gaussiana definida conforme o material da disciplina (slides RNA da aula 8) e função de saída é a função logística.

As bases de treino e de teste foram fornecidas pelo EPC8 e consiste de 150 dados de treinamento e 15 dados de teste.

2.3 Linguagem

Os códigos deste trabalho foram feitos usando *python*. Nenhuma biblioteca de Redes Neurais foram usadas, apenas bibliotecas para manipulação de matrizes e visualização de gráficos. O código está no Apêndice e também pode ser acessado através do link https://github.com/grgoncal/SEL5712—Redes-Neurais-Artificiais.

3 Resultados e discussão

3.1 K-means

Foram feitos três treinamentos para cada uma das topologias propostas pelo EPC8. Para cada um deles, foi registrado o número de épocas e o Erro Quadrático Médio final do treinamento. Os resultados estão resumidos na Tabela 1

Treinamento	Rede 1		Rede 2		Rede 3	
	EQM	Épocas	EQM	Épocas	EQM	Épocas
1	0.169	651	0.251	2933	0.2627	2710
2	0.169	353	0.249	2986	0.2645	2739
3	0.178	279	0.247	2971	0.2684	2576

Tabela 1: Treinamento da primeira camada

3.2 Casos de teste

Em seguida, foi realizado o treinamento da camada de saída usando o *backpropagation* com uma taxa de aprendizagem de 0.01 e uma precisão de 0.0000001.

A rede treinada calculou a saída de cada um dos treinamentos de cada topologia para 15 casos de testes. A saída dos treinamentos pode ser vista na Tabela 2.

					Rede 1			Rede 2			Rede 3		
Amostra	x1	x2	x3	d	y(T1)	y(T2)	y(T3)	y(T1)	y(T2)	y(T3)	y(T1)	y(T2)	y(T3)
01	0.5102	0.7464	0.0860	0.5965	0.6213	0.6215	0.6219	0.5782	0.5782	0.5782	0.5882	0.5882	0.5882
02	0.8401	0.4490	0.2719	0.6790	0.7486	0.7490	0.7501	0.6751	0.6751	0.6751	0.6350	0.6350	0.6350
03	0.1283	0.1882	0.7253	0.4662	0.4366	0.4365	0.4357	0.4784	0.4784	0.4784	0.5035	0.5035	0.5035
04	0.2299	0.1524	0.7353	0.5012	0.4442	0.4442	0.4438	0.4977	0.4977	0.4977	0.5198	0.5198	0.5200
05	0.3209	0.6229	0.5233	0.6810	0.7241	0.7246	0.7264	0.6543	0.6543	0.6543	0.6660	0.6660	0.6660
06	0.8203	0.0682	0.4260	0.5643	0.5919	0.5919	0.5929	0.5329	0.5329	0.5329	0.5327	0.5327	0.5328
07	0.3471	0.8889	0.1564	0.5875	0.5432	0.5433	0.5437	0.5806	0.5806	0.5806	0.6013	0.6012	0.6016
08	0.5762	0.8292	0.4116	0.7853	0.8272	0.8276	0.8286	0.7781	0.7781	0.7781	0.7554	0.7554	0.7554
09	0.9053	0.6245	0.5264	0.8506	0.8214	0.8217	0.8224	0.8799	0.8799	0.8799	0.8252	0.8252	0.8252
10	0.8149	0.0396	0.6227	0.6165	0.5904	0.5904	0.5910	0.5908	0.5908	0.5908	0.6793	0.6793	0.6794
11	0.1016	0.6382	0.3173	0.4957	0.4524	0.4528	0.4546	0.5020	0.5020	0.5020	0.5258	0.5258	0.5256
12	0.9108	0.2139	0.4641	0.6625	0.6519	0.6521	0.6532	0.6182	0.6182	0.6182	0.5765	0.5765	0.5764
13	0.2245	0.0971	0.6136	0.4402	0.3664	0.3666	0.3665	0.4324	0.4324	0.4324	0.4681	0.4681	0.4681
14	0.6423	0.3229	0.8567	0.7663	0.7265	0.7263	0.7262	0.7669	0.7669	0.7669	0.7621	0.7621	0.7621
15	0.5252	0.6529	0.5729	0.7893	0.8603	0.8606	0.8617	0.8237	0.8237	0.8237	0.7870	0.7870	0.7869

Tabela 2: Resultados da rede

O desvio médio de cada uma das topologias foi de 7%, 2.7% e 4.87% para 5, 10 e 15 neurônios na primeira camada, respectivamente. A variância de cada uma das topologias foi de aproximadamente 2.3%, 1.7% e 1.19% novamente para 5, 10 e 15 neurônios na primeira camada.

A segunda rede, com 10 neurônios teve um desvio menor que as demais topologias, e deveria ser a topologia escolhida para uma aplicação prática, devido ao seu erro baixo.

Nota-se também que o número de neurônios na primeira camada tem um papel muito importatne na precisão da rede, podendo ser insuficientes, como no caso de 5 neurônios na primeira camada, ou em um número exagerado, como no caso de 15 neurônios. Em ambos os casos a precisão foi menor do que com 10 neurônios, mostrando que uma aproximação com esse número de neurônios para esse problema é mais adequada.

3.3 Erro Quadrático Médio

Para o melhor treinamento de cada topologia foi gerado um gráfico do comportamento do erro quadrático médio. Os resultados podem ser vistos nas Figuras 1, 2 e 3.

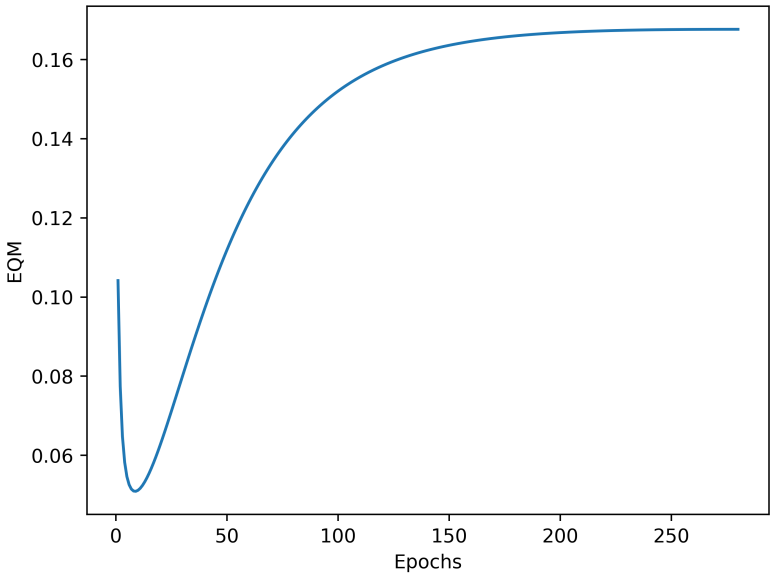

Figura 1: Erro Quadrático Médio do melhor treinamento da Rede 1.

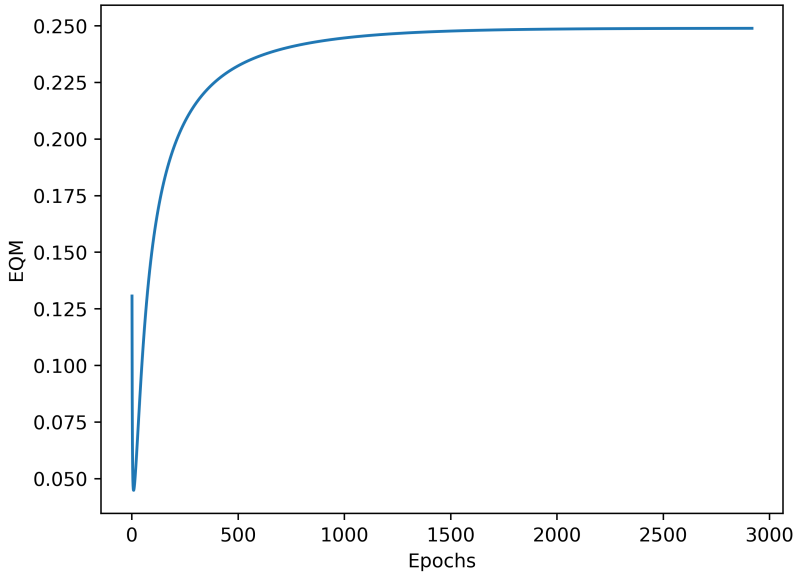

Figura 2: Erro Quadrático Médio do melhor treinamento da Rede 2.

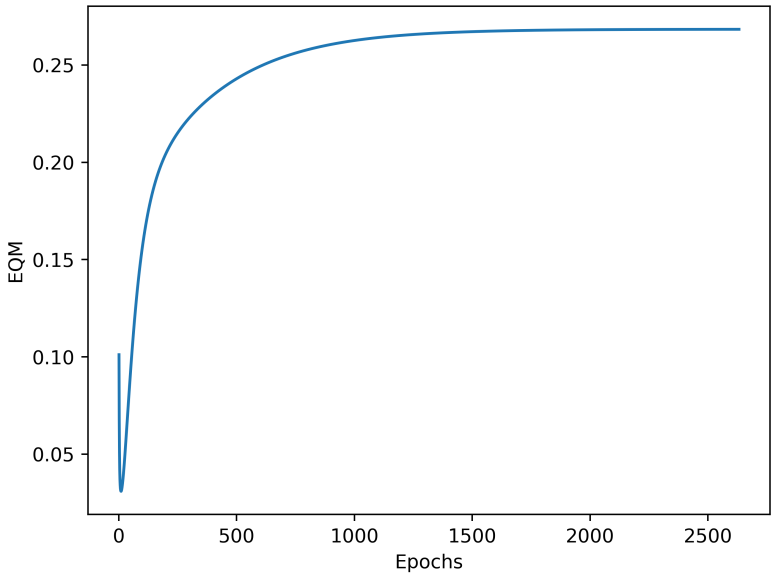

Figura 3: Erro Quadrático Médio do melhor treinamento da Rede 3.

4 Conclusão

As redes neurais com funções de base radial são muito versáteis e reduzem muito a quantidade de neurônios necessária para classificar ou realizar determinada tarefa. Neste EPC, foram testadas três topologias diferentes para a solução de um problema de engenharia.

As três topologias tiveram seus desvios padrões e variância comparados e a rede que teve o melhor desempenho foi a segunda. Isso pode significar que 10 neurônios são capazes de oferecer uma boa solução para este problema em específico. Mais ou menos que 10 neurônios obtiveram resultados piores ou insatisfatórios.

Foi possível com a topologia de 10 neurônios uma taxa de acerto de 1.7% com uma base de teste de 150 amostras.

Novamente, é importante que se treine a última camada da rede mais de uma vez para tentar encontrar a solução mais otimizada para o problema. Já em relação ao k-means, não existe um motivo para se treinar várias vezes, uma vez que o resultado sempre é idêntico.

O método de aprendizagem não supervisionado do *k-means* para a primeira camada funcionou bem e conseguiu separar as amostras em clusters bem organizados. Em relação a camada de saída, foi usado o *backpropagation*, que novamente foi capaz de suprir as necessidades da rede.

5 Apêndice

5.1 Código fonte

O código fonte dos algoritmos desenvolvidos está disponível no repositório público https://github.com/grgoncal/SEL5712—Redes-Neurais-Artificiais para livre acesso.

```
# IMPORTS ------------------------------------------------------------
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import sys

#//////////////////////////////////////////////////////////////////////
# CONSTANTS ///////////////////////////////////////////////////////////
#//////////////////////////////////////////////////////////////////////

error_tolerance = 0.0000001
learning_rate = 0.01
number_neurons_1st_layer = 5
number_neurons_2nd_layer = 1

#//////////////////////////////////////////////////////////////////////
# K-MEANS /////////////////////////////////////////////////////////////
#//////////////////////////////////////////////////////////////////////

def kMeans(train, w1, number_neurons_1st_layer):

    groups = []
    last_groups = []
    for i in range(number_neurons_1st_layer):
        last_groups.append([1])
        groups.append([])


    while(last_groups != groups):
```

```
31
32          last_groups = groups
33          groups = []
34          for i in range(number_neurons_1st_layer):
35              groups.append([])
36
37          euclidian_distance = pd.DataFrame(np.zeros((len(train),
    number_neurons_1st_layer)))
38
39          for i in range(len(train)):
40              for j in range(number_neurons_1st_layer):
41                  for column in range(len(train.columns) - 1):
42                      euclidian_distance.iloc[i,j] += math.pow(train.iloc[i,
    column] - w1.iloc[j,column] , 2)
43                  euclidian_distance.iloc[i,j] = math.sqrt(euclidian_distance.
    iloc[i,j])
44              groups[euclidian_distance.iloc[i,:].idxmin()].append(i)
45
46          w1 = pd.DataFrame(np.zeros((number_neurons_1st_layer,3)))
47
48          for i in range(number_neurons_1st_layer):
49              for j in groups[i]:
50                  for n in range(len(train.columns) - 1):
51                      w1.iloc[i,n] += train.iloc[j,n]/len(groups[i])
52
53
54      variance = []
55      for i in range(number_neurons_1st_layer):
56          variance.append(0)
57
58      for i in range(number_neurons_1st_layer):
59          for j in groups[i]:
60              for column in range(len(train.columns) - 1):
61                  variance[i] += math.pow(train.iloc[j,column] - w1.iloc[i,column
    ], 2)
62          variance[i] = variance[i]/len(groups[i])
63
64      return w1, variance
65
66
67  #////////////////////////////////////////////////////////////////
68  # OUTPUT LAYER /////////////////////////////////////////////////
69  #////////////////////////////////////////////////////////////////
70
71  def outputLayer(train, w1, w2, variance, testNumber, number_neurons_1st_layer):
72      chart = [[],[]]
73
74      d = train.iloc[:,3]
75      x = train.iloc[:,:-1]
76
77      z = pd.DataFrame(np.zeros((len(train), number_neurons_1st_layer + 1)))
78      for i in range(len(z)):
79          z.iloc[i,number_neurons_1st_layer] -= 1
80
81      for sample in range(len(train)):
82          for neuron in range(number_neurons_1st_layer):
```

```
83              for column in range(len(x.columns)):
84                  z.iloc[sample,neuron] += math.pow(x.iloc[sample,column] - w1.
    iloc[neuron,column], 2)
85              z.iloc[sample,neuron] = math.exp(-z.iloc[sample,neuron]/(2*variance
    [neuron]))
86
87      epoch = 0
88      Eqm = 0
89      lEqm = 1
90      while(abs(Eqm - lEqm) > error_tolerance and epoch < 5000):
91          lEqm = Eqm
92
93          # SECOND LAYER WEIGHTS
94          for i in range(len(train)):
95              y = logistic((z.iloc[i,:] * w2.iloc[:,0]).sum())
96
97              for j in range(number_neurons_1st_layer + 1):
98                  gradient = (d.iloc[i] - y) * dlogistic((z.iloc[i,:]).sum())
99                  w2.iloc[j,0] += learning_rate * gradient * z.iloc[i,j]
100
101         # QUADRATIC ERROR
102         Eqm = 0
103         for i in range(len(train)):
104             y = (z.iloc[i,:] * w2.iloc[:,0]).sum()
105             Eqm += math.pow(d.iloc[i] - y ,2)/2
106         Eqm = Eqm/len(train)
107
108         chart[0].append(epoch)
109         chart[1].append(Eqm)
110
111         epoch += 1
112         print str(Eqm) + " " + str(abs(Eqm - lEqm))
113
114     plt.figure(1)
115     plt.plot(chart[0][1:], chart[1][1:])
116     plt.xlabel('Epochs')
117     plt.ylabel('EQM')
118     plt.savefig("./" + str(number_neurons_1st_layer) + "_test_number_" + str(
    testNumber) + ".png", dpi = 500)
119     plt.close()
120
121     return w2, epoch, Eqm
122
123 #////////////////////////////////////////////////////////////////
124 # LOGISTIC FUNCTION ///////////////////////////////////////////////
125 #////////////////////////////////////////////////////////////////
126
127 def logistic(x):
128     return 1 / (1 + math.exp(-x))
129
130 #////////////////////////////////////////////////////////////////
131 # DERIVATIVE LOGISTIC FUNCTION ////////////////////////////////////
132 #////////////////////////////////////////////////////////////////
133
134 def dlogistic(x):
135     return logistic(x) * (1 - logistic(x))
```

```
136
137 #////////////////////////////////////////////////////////////////
138 # TEST SAMPLES ////////////////////////////////////////////////////
139 #////////////////////////////////////////////////////////////////
140
141 def runTest(test,w1,w2, variance, number_neurons_1st_layer):
142     x = test.iloc[:,:]
143
144     z = pd.DataFrame(np.zeros((len(test), number_neurons_1st_layer + 1)))
145     for i in range(len(z)):
146         z.iloc[i,number_neurons_1st_layer] -= 1
147
148     for sample in range(len(test)):
149         for neuron in range(number_neurons_1st_layer):
150             for column in range(len(x.columns) - 1):
151                 z.iloc[sample,neuron] += math.pow(x.iloc[sample,column] - w1.
    iloc[neuron,column], 2)
152             z.iloc[sample,neuron] = math.exp(-z.iloc[sample,neuron]/(2*variance
    [neuron]))
153
154
155     out = []
156
157     for i in range(len(test)):
158         y = logistic((z.iloc[i,:] * w2.iloc[:,0]).sum())
159         out.append(y)
160
161     return out
162
163 #////////////////////////////////////////////////////////////////
164 # MAIN ////////////////////////////////////////////////////////////
165 #////////////////////////////////////////////////////////////////
166
167 # TRAIN DATASET ----------------------------------------------------
168 train = pd.read_csv('./train.csv', header = None)
169 test = pd.read_csv('./test.csv', header = None)
170
171 results = []
172
173 for number_neurons in range(5,20,5):
174     number_neurons_1st_layer = number_neurons
175     for test_number in range(1,4):
176         w1 = pd.DataFrame(train.iloc[:number_neurons_1st_layer,:-1])
177         w2 = pd.DataFrame(np.random.rand(number_neurons_1st_layer + 1,
    number_neurons_2nd_layer))
178
179         # CALCULATE CLUSTERS
180         w1, variance = kMeans(train, w1,number_neurons_1st_layer)
181         w2, epoch, eqm = outputLayer(train,w1,w2,variance,test_number,
    number_neurons_1st_layer)
182
183         index_results = (number_neurons/5) * 3 - (4 - test_number)
184         results.append(runTest(test, w1, w2, variance, number_neurons_1st_layer
    ))
185         print "END TRAINING " + str(test_number) + " NUMBER OF EPOCHS: " + str(
    epoch) + " EQM: " + str(eqm)
```

```
186         print str(results[index_results])
187
188 print "[RESUTLADOS]
    ////////////////////////////////////////////////////////////"
189
190 print str(results)
```

Listing 1.1: Python code