Redes Neurais Artificiais

(Prof. Ivan Nunes da Silva)

EPC-8

Luiz Felipe Machado Votto

A quantidade de gasolina {*y*} a ser injetada por um sistema de injeção eletrônica de combustível para veículos automotores pode ser computada em tempo-real em função de três grandezas {*x*1, *x*2, *x*3}. Devido à complexidade inerente do processo, configurado como um sistema não-linear, pretende-se utilizar uma rede neural artificial para o mapeamento entre as entradas e a saída do processo.

Sabe-se que para efetuar o respectivo mapeamento (problema de aproximação funcional), duas potenciais arquiteturas podem ser aplicadas, a saber, o perceptron multicamadas ou a RBF. Dado que a equipe de engenheiros e cientistas já realizou o mapeamento do problema por meio do perceptron multicamadas, o objetivo agora é treinar uma RBF a fim de que os resultados fornecidos por ambas as arquiteturas possam ser contrastados.

Assim, efetue o treinamento de uma RBF com o objetivo de computar a quantidade de gasolina {*y*} a ser injetada pelo sistema de injeção eletrônica em função das variáveis {*x*1 , *x*2 , *x*3}. A topologia da rede RBF está ilustrada na figura abaixo.

1

*x*1

*x*2

2

*y*

1

:

*x*3

N1

As topologias candidatas de RBF para serem aplicadas no mapeamento do problema acima são especificadas como se segue:

**Rede 1** 🡪 RBF com N1 = 05

**Rede 2** 🡪 RBF com N1 = 10

**Rede 3** 🡪 RBF com N1 = 15

Utilizando os dados de treinamento apresentados no Anexo, execute o treinamento das redes RBF conforme as topologias definidas acima. Para tanto, faça as seguintes atividades:

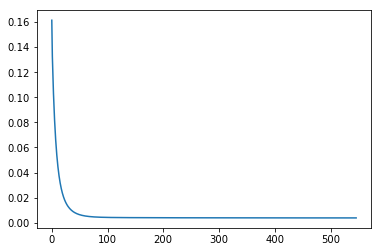
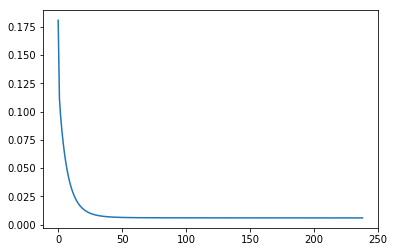
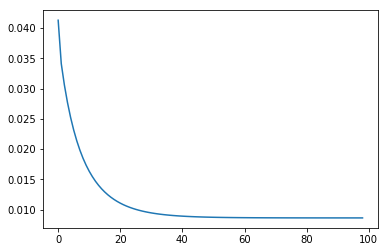
1. Execute 3 treinamentos para cada topologia de rede RBF definida anteriormente, inicializando a matriz de pesos da camada de saída com valores aleatórios entre 0 e 1. Se for o caso, reinicie o gerador de números aleatórios em cada treinamento de tal forma que os elementos das matrizes de pesos iniciais não sejam os mesmos. Utilize uma taxa de aprendizado η = 0.01 e precisão ε = 10-7.
2. Registre os resultados finais desses 3 treinamentos, para cada uma das três topologias de rede, na tabela a seguir:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Treinamento | **Rede 1** | | **Rede 2** | | **Rede 3** | |
| EQM | Épocas | EQM | Épocas | EQM | Épocas |
| 1o (T1) | 0,0086 | 120 | 0,0059 | 239 | 0,0038 | 479 |
| 2o (T2) | 0,0086 | 85 | 0,0059 | 361 | 0,0038 | 547 |
| 3o (T3) | 0,0086 | 99 | 0,0059 | 356 | 0,0038 | 236 |

1. Para todos os treinamentos efetuados no item 2, faça a validação da rede em relação aos valores desejados apresentados na tabela abaixo. Forneça para cada treinamento o erro relativo médio (%) entre os valores desejados e os valores fornecidos pela rede em relação a todos os padrões de teste. Obtenha também a respectiva variância (%).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | |  | |  | |  |  | | | **Rede 1** | | | | | **Rede 2** | | | | | | | **Rede 3** | | | | | |
| Amostra | *x*1 | | *x*2 | | *x*3 | | *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 | | 0.6081 | | | 0.6084 | | | 0.6082 | | 0.5878 | | | 0.5878 | | 0.5878 | | 0.5938 | | | 0.5915 | | 0.5923 | |
| 02 | 0.8401 | | 0.4490 | | 0.2719 | | 0.6790 | | 0.7361 | | | 0.7355 | | | 0.7363 | | 0.6758 | | | 0.6758 | | 0.6758 | | 0.634 | | | 0.6371 | | 0.6349 | |
| 03 | 0.1283 | | 0.1882 | | 0.7253 | | 0.4662 | | 0.4364 | | | 0.4365 | | | 0.4371 | | 0.4881 | | | 0.4881 | | 0.4881 | | 0.509 | | | 0.5089 | | 0.5071 | |
| 04 | 0.2299 | | 0.1524 | | 0.7353 | | 0.5012 | | 0.4431 | | | 0.4429 | | | 0.4438 | | 0.5033 | | | 0.5033 | | 0.5033 | | 0.5137 | | | 0.5212 | | 0.517 | |
| 05 | 0.3209 | | 0.6229 | | 0.5233 | | 0.6810 | | 0.7018 | | | 0.6996 | | | 0.702 | | 0.6446 | | | 0.6446 | | 0.6446 | | 0.6649 | | | 0.6643 | | 0.6648 | |
| 06 | 0.8203 | | 0.0682 | | 0.4260 | | 0.5643 | | 0.596 | | | 0.5963 | | | 0.595 | | 0.5291 | | | 0.5291 | | 0.5291 | | 0.5187 | | | 0.5274 | | 0.5224 | |
| 07 | 0.3471 | | 0.8889 | | 0.1564 | | 0.5875 | | 0.5434 | | | 0.5438 | | | 0.5431 | | 0.5828 | | | 0.5828 | | 0.5828 | | 0.5763 | | | 0.5967 | | 0.5882 | |
| 08 | 0.5762 | | 0.8292 | | 0.4116 | | 0.7853 | | 0.8284 | | | 0.8269 | | | 0.8292 | | 0.7573 | | | 0.7573 | | 0.7573 | | 0.7419 | | | 0.7348 | | 0.738 | |
| 09 | 0.9053 | | 0.6245 | | 0.5264 | | 0.8506 | | 0.8244 | | | 0.8237 | | | 0.825 | | 0.8903 | | | 0.8903 | | 0.8903 | | 0.8205 | | | 0.8165 | | 0.8177 | |
| 10 | 0.8149 | | 0.0396 | | 0.6227 | | 0.6165 | | 0.5946 | | | 0.5951 | | | 0.5936 | | 0.5997 | | | 0.5998 | | 0.5997 | | 0.6775 | | | 0.6816 | | 0.6789 | |
| 11 | 0.1016 | | 0.6382 | | 0.3173 | | 0.4957 | | 0.4633 | | | 0.4625 | | | 0.463 | | 0.5019 | | | 0.5019 | | 0.5019 | | 0.538 | | | 0.5309 | | 0.5333 | |
| 12 | 0.9108 | | 0.2139 | | 0.4641 | | 0.6625 | | 0.6502 | | | 0.6502 | | | 0.6493 | | 0.6178 | | | 0.6178 | | 0.6178 | | 0.5778 | | | 0.5771 | | 0.5762 | |
| 13 | 0.2245 | | 0.0971 | | 0.6136 | | 0.4402 | | 0.3715 | | | 0.3711 | | | 0.3724 | | 0.4372 | | | 0.4372 | | 0.4372 | | 0.4695 | | | 0.4703 | | 0.4656 | |
| 14 | 0.6423 | | 0.3229 | | 0.8567 | | 0.7663 | | 0.7149 | | | 0.7149 | | | 0.7144 | | 0.754 | | | 0.754 | | 0.754 | | 0.74 | | | 0.7457 | | 0.7432 | |
| 15 | 0.5252 | | 0.6529 | | 0.5729 | | 0.7893 | | 0.8824 | | | 0.88 | | | 0.8829 | | 0.7982 | | | 0.7982 | | 0.7982 | | 0.7655 | | | 0.7568 | | 0.7604 | |
| Erro Relativo Médio (%): | | | 0.0058 | | | | | | | | | 0.0063 | | | 0.0058 | | 0.0076 | | | 0.0076 | | 0.0076 | | 0.0094 | | | 0.0081 | | 0.0095 |
| Variância (%): | | | 0.0021 | | | | | | | | | 0.002 | | | 0.0021 | | 0.0005 | | | 0.0005 | | 0.0005 | | 0.0016 | | | 0.0016 | | 0.0016 |

1. Para cada uma das topologias apresentadas na tabela acima, considerando ainda o melhor treinamento {T1, T2 ou T3} realizado em cada uma delas, trace o gráfico dos valores de erro quadrático médio (EQM) em função de cada época de treinamento. Imprima os três gráficos numa mesma folha de modo não superpostos.



1. Baseado nas análises dos itens acima, indique qual das topologias candidatas {Rede 1, Rede 2 ou Rede 3} e com que qual configuração final de treinamento {T1, T2 ou T3} seria a mais adequada para este problema.

Tomando o erro quadrático médio em vista, a topologia da Rede 3 é mais recomendada, e, com base na etapa de avaliação, a T2 desta topologia pode ser considerada a mais adequada.

ANEXO: Código Python da classe RBF usada para solução do problema.

import matplotlib.pyplot as plt

import numpy as np

from operator import itemgetter

import random

import utils

class RBF:

def \_\_init\_\_(self, specs=[2, 2, 1], learning\_rate=.01, epsilon=1E-7,

act\_func=(lambda x: x), act\_prime=(lambda x: 1)):

self.specs = specs

self.learning\_rate = learning\_rate

self.epsilon = epsilon

self.act\_func = act\_func

self.act\_prime = act\_prime

self.weights = []

# Entry layer has no weights

self.weights.append([])

self.weights.append([])

for j in range(self.specs[1]):

middle\_weight = [random.uniform(-1, 1) for i in range(self.specs[0])]

self.weights[1].append(middle\_weight)

self.weights.append([])

for j in range(self.specs[2]):

out\_weight = [random.uniform(-1, 1) for i in range(self.specs[1] + 1)]

self.weights[2].append(out\_weight)

self.variances = [1 for neuron in range(self.specs[1])]

def \_\_call\_\_(self, sample):

middle\_output = [-1]

for j in range(self.specs[1]):

middle\_output.append(self.\_make\_pseudo\_sample(sample, j))

out\_input = []

for j in range(self.specs[2]):

out\_input.append(np.dot(self.weights[2][j], middle\_output))

return self.act\_func(out\_input)

def \_dist(self, a, b):

a\_ = np.array(a)

b\_ = np.array(b)

return np.linalg.norm(b\_ - a\_)

def \_k\_means(self):

for neuron in range(self.specs[1]):

self.weights[1][neuron] = self.samples[neuron][:]

last\_groups = []

groups = [[] for neuron in range(self.specs[1])]

while last\_groups != groups:

last\_groups = groups

groups = [[] for neuron in range(self.specs[1])]

for k in range(len(self.samples)):

dists = []

for neuron in range(self.specs[1]):

dists.append(self.\_dist(self.weights[1][neuron],

self.samples[k]))

min\_index = min(enumerate(dists), key=itemgetter(1))[0]

groups[min\_index].append(k)

for neuron in range(self.specs[1]):

for synapse in range(self.specs[0]):

self.weights[1][neuron][synapse] = 0

for k in groups[neuron]:

inc = self.samples[k][synapse]

self.weights[1][neuron][synapse] += inc

self.weights[1][neuron] = np.array(self.weights[1][neuron])

self.weights[1][neuron] \*= 1 / len(groups[neuron])

for neuron in range(self.specs[1]):

self.variances[neuron] = 0

for k in groups[neuron]:

for syn in range(self.specs[0]):

inc = (self.samples[k][syn] \

- self.weights[1][neuron][syn]) \*\* 2

self.variances[neuron] += inc

self.variances[neuron] \*= 1 / len(groups[neuron])

def \_make\_pseudo\_sample(self, sample, neuron):

g = 1

for i in range(len(sample)):

g \*= utils.gauss(sample[i],

self.weights[1][neuron][i],

self.variances[neuron])

return g

def \_make\_pseudo\_sample\_set(self):

pseudo\_samples = []

for k in range(len(self.samples)):

pseudo\_samples.append([-1])

for neuron in range(self.specs[1]):

pseudo\_samples[k].append(

self.\_make\_pseudo\_sample(self.samples[k], neuron))

return pseudo\_samples

def train(self, samples, desired, log=False):

self.samples = samples

self.desired = desired

self.\_k\_means()

pseudo\_samples = self.\_make\_pseudo\_sample\_set()

iteration = 0

error = self.epsilon + 1

last\_error = 0

errors = []

while np.abs(error - last\_error) > self.epsilon:

last\_error = error

inputs = []

for k in range(len(self.samples)):

inputs.append([])

for neuron in range(self.specs[2]):

inputs[k].append(np.dot(self.weights[2][neuron],

pseudo\_samples[k]))

outputs = [self.act\_func(x) for x in inputs]

for j in range(self.specs[2]):

delta = (self.desired[k][j] - outputs[k][j]) \

\* self.act\_prime(inputs[k][j])

for i in range(self.specs[1] + 1):

self.weights[2][j][i] += self.learning\_rate \

\* delta \

\* pseudo\_samples[k][i]

error = 0

for k in range(len(self.samples)):

for neuron in range(self.specs[2]):

inp = np.dot(self.weights[2][neuron],

pseudo\_samples[k])

output = self.act\_func(inp)

error += (desired[k][neuron] - output) \*\* 2

error /= len(self.samples)

errors.append(error)

if log:

if not iteration % 72:

print(iteration, ': ', error)

iteration += 1

if log:

print('Network trained in %s iterations.' % iteration)

print('With error: ', error)

plt.plot(errors)

plt.show()