EPC1: Perceptron

Guilherme Rocha Gonçalves[1]

Pós Graduação em Engenharia de Produção
Universidade de São Paulo
São Carlos, Brasil

1 Introdução

1.1 Contextualização

Primeira 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 para solução de um problema de engenharia. Além disso, este trabalho pode ser considerado como um complemento ao que foi lecionado em sala de aula.

2 Materiais e Métodos

2.1 Problema

Uma rede neural artificial precisa ser treinada para automatizar a classificação de duas classes de pureza de um determinado óleo.

2.2 Bases de treino e de teste

O treinamento da rede neural será feito usando a *Regra de Aprendizado de Hebb*. A Regra de Hebb busca corrigir os pesos da rede neural quando a saída obtida difere da esperada. Esse ajuste é feito proporcionalmente ao valor das entradas. Esse tipo de método admite infinitas soluções, uma vez que qualquer variação nos pesos iniciais ou na taxa de aprendizagem pode gerar uma solução válida diferente. O Perceptron é usado para classificação de classes linearmente separáveis. Os dados de treino e de testes são fornecidos pelo EPC2.

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 Treinamento da Rede ADELINE

Foram feitos 5 treinamentos para a rede. O vetor de pesos foi iniciado com valores aleatórios entre 0 e 1, a taxa de aprendizagem foi definida como 0.01. O resultado dos treinamentos estão documentado na Tabela 1.

Treino	w0	w1	w2	w3	w0	w1	w2	w3	Épocas
1	0.90518106	0.35501475	0.20326804	0.90178048	1.04518106	0.29796875	0.12178004	-0.22154552	2
2	0.55919909	0.76994198	0.8311337	0.96034159	0.71919909	0.70461998	0.7413117	-0.28979041	2
3	0.72717551	0.79284309	0.7107744	0.44688658	0.80717551	0.78800509	0.6635764	-0.12909142	2
4	0.26593594	0.10589005	0.46855095	0.24276428	0.32593594	0.10619005	0.43481295	-0.16668372	2
5	0.64827633	0.80172618	0.3532264	0.3293457	0.70827633	0.74426018	0.3340524	-0.1637783	2

Tabela 1: Treinamento

3.2 Perceptron em casos de testes

Foram fornecidos alguns objetos de teste para que a rede possa classificar após cada iteração de treinamento. Os resultados foram iguais para todos os casos, como era esperado. A Tabela 2 mostra os resultados.

Amostra	x1	x2	x3	y(1)	y(2)	y(3)	y(4)	y(5)
1	-0.3565	0.062	5.9891	-1	-1	-1	-1	-1
2	-0.7842	1.1267	5.5912	-1	-1	-1	-1	-1
3	0.3012	0.5611	5.8234	-1	-1	-1	-1	-1
4	0.7757	1.0648	8.0677	-1	-1	-1	-1	-1
5	0.157	0.8028	6.304	-1	-1	-1	-1	-1
6	-0.7014	1.0316	3.6005	-1	-1	-1	-1	-1
7	0.3748	0.1536	6.1537	-1	-1	-1	-1	-1
8	-0.692	0.9404	4.4058	-1	-1	-1	-1	-1
9	-1.397	0.7141	4.9263	-1	-1	-1	-1	-1
10	-1.8842	-0.2805	1.2548	-1	-1	-1	-1	-1

Tabela 2: Resultado dos testes.

3.3 Diferença de épocas no Perceptron

Diferente do ADALINE, o perceptron permite que existam infinitas soluções. Portanto, dependendo dos parâmetros estáticos, tais como a taxa de aprendizagem e os valores iniciais do vetor de pesos, o número de épocas pode variar.

Se a taxa de aprendizagem for muito baixa, pode ser que mais épocas sejam necessárias para uma solução ser encontrada. Caso a taxa de aprendizagem for muito alta, o algoritmo pode ficar instável, aumentando em alguns casos o número de épocas ou na pior das hipóteses, impossibilitando a convergência. No algoritmo desenvolvido para esse trabalho, o número de épocas aumentou quando a taxa de aprendizagem foi alterada para 0.001, chegando em até 8 épocas, em alguns casos.

4 Conclusão

A grande desvantagem do perceptron é exigir que as classes a serem classificadas sejam obrigatóriamente linearmente separáveis. Caso contrário, a rede não converge.

Além dessa limitação, foi verificado pelo proponente a influência da taxa de aprendizagem na convergência e também a importância de se limitar o número de épocas, para eventuais erros no código ou mesmo para evitar *loops* infinitos causados pela não-convergência.

O Perceptron é um ótimo inicio de rede para a disciplina e apesar de simples, pode resolver muitos problemas.

5 Apêndice

5.1 Código fonte

O código fonte dos algoritmos desenvolvidos está disponível no repositório público [1] para livre acesso.

```
# IMPORTS ----------------
import pandas as pd
import numpy as np

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

learningRate = 100
maxEpochs = 1000
errTol = 0
trainNumber = 5

#////////////////////////
# TRAIN /////////////////
#////////////////////////

def train(x, d, w):
    epoch = 0
    err = 1
    print ("\n[W INITIAL]  " + str(w.T.values))

    while(epoch < maxEpochs and err > errTol):
        err = 0
        for i in range(0,x.shape[0]):
            u = w.iloc[:,0] * x.iloc[i,:]
            u = u.sum()
            if u >= 0:
                y = 1
            else:
                y = -1

            if (y - d.iloc[i]) == 2:
                w = (w.T + learningRate * (d.iloc[i] - y) * x.
    iloc[i,:]).T
                err += 1

        # print err
        epoch += 1

    print ("[W FINAL]  " + str(w.T.values))
    print ("[EPOCHS]  " + str(epoch) + "\n")
    return w

#////////////////////////
# TEST /////////////////
#////////////////////////

def test(w):
    x = pd.read_csv('./test.csv', header = None)
```

```
50      for name in reversed(x.columns.values):
51          x.rename(columns = {name : name + 1}, inplace = True)
52      x.insert(loc = 0, column=0, value = np.full((x.shape[0],1)
        , -1))
53
54      y = []
55
56      for i in range(0, x.shape[0]):
57          u = w.iloc[:,0] * x.iloc[i,:]
58          u = u.sum()
59          if u >= 0:
60              y.append(1)
61          else:
62              y.append(-1)
63
64      return y
65
66  #////////////////////////
67  # MAIN ///////////////////
68  #////////////////////////
69
70  # TRAIN DATASET ----------------
71  data = pd.read_csv('./train.csv', header = None)
72
73  # GET INPUTS, OUTPUTS AND WEIGTHS ----------------
74  x = data.iloc[:,1:(data.shape[1] - 1)]
              # GET INPUTS
75  x.insert(loc = 0, column=0, value = np.full((x.shape[0],1),
        -1))     # ADD -1 INPUT
76  d = data.iloc[:,(data.shape[1] - 1)]
              # GET OUTPUTS
77
78  for i in range(1, trainNumber + 1):
79      w = pd.DataFrame(np.random.rand((data.shape[1] - 1)))
                  # GENERATE WEIGHTS
80      print("[TRAINING NUMBER " + str(i) + "] ---------")
81      y = test(train(x, d, w))
82      print("[RESULT OF TRAINING NUMBER " + str(i) + "] " + str(
        y) + "\n")
```

Listing 1.1: Python code

Referências Bibliográficas

[1] Gonçalves, G.R.: Github: Sel5712 - redes neurais artificiais. github.com/grgoncal/SEL5712—Redes-Neurais-Artificiais, acessado: 2019-04-05