Fundamentos de Redes Neurais

Henrique Nazário Rocha
Acadêmico do curso de Engenharia Elétrica
Universidade Tecnológica Federal do Paraná
nazario.utfpr@gmail.com

I. Introdução.

Um neurônio artificial é uma unidade de processamento de informação. Sua Inspiração é derivada do estudo de nosso sistema nervoso, no caso, sobre a propagação e geração de pulsos elétricos pela membrana celular dos neurônios biológicos. A representação do mesmo é dada por um modelo matemático simplificado do neurônio biológico, onde este possui pesos sinápticos ajustáveis com o intuito de replicar a plasticidade sináptica. O neurônio artificial contém funções de receber, processar e transmitir sinais, sendo capazes de reconhecer padrões de dados através do processo de treinamento e realizar generalizações baseada no aprendizado adquirido.

Este trabalho tem como objetivo descrever a atividade realizada no exercício 1 da matéria fundamentos de redes neurais.

II. Perceptron.

O Modelo de rede neural Perceptron foi desenvolvido por Frank Rosenblatt, onde foi inspirado em trabalhos anteriores de Warren McCulloch e Walter Pitts, onde o Perceptron se baseava no funcionamento da retina. O Perceptron permite uma compreensão de forma bem simples e clara do funcionamento de uma rede neural.

O Perceptron é formado apenas de um neurônio artificial, este único neurônio recebe todos os sinais de entrada e possui apenas uma saída.

Na Figura 1 temos a representação de um Perceptron por blocos, é possível notar as entradas do neurônio $\{x_1, x_2, x_3, x_n\}$, também na Figura temos os pesos sinápticos ou sinapses do neurônio artificial $\{w_1, w_2, w_3, w_n\}$, o peso sináptico w_0, é referente ao limiar de ativação, temos um somador onde seria o corpo celular de nosso neurônio, denotado por $\sum$, seguindo o fluxo da informação temos o potencial de ativação onde é denotado por u, e uma função de ativação $g(.)$ e a saída do neurônio y.

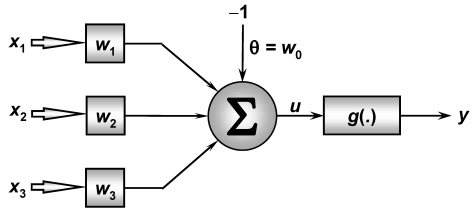

Fig. 1. Perceptron.

Para realização da atividade foi necessário montar o algorítimo Perceptron, para a modelagem computacional foi empregado a biblioteca *Python-Numpy*, onde esta biblioteca implementa suporte para trabalhar com *arrays* e matrizes multidimensionais, possuindo uma larga coleção de funções matemáticas para trabalhar com estas estruturas.

```
import numpy as np

class Perceptron:

  '''
  Objeto Perceptron, um único neurônio
  para realizar operações linearmente separáveis
  '''
  __bias = -1
  def __init__(self, features, interactions = 100,
               learning_rate = 0.01):

    '''
    Método construtor da classe Perceptron,
    Default - interações = 100 e
    função de ativação degrau.
    '''

    self.learning_rate = learning_rate
    self.weight = np.random.random_sample(features+1)
    self.interactions = interactions
    self.MSE = []

  def forward(self, x):
    u = np.dot(x, self.weight)
    return np.heaviside(u,0.)

  def train(self, xtrain, ytrain):
    xtrain = np.insert(xtrain, 0,
                       Perceptron.__bias, axis=1)
    self.interaction = 0
    escape = False
    error = np.zeros(ytrain.shape[0])
    cont = 0

    while (escape == False):
      escape = True
      cont = 0
      self.interaction += 1

      for inputs, label in zip(xtrain, ytrain):
          yPredict = self.forward(inputs)
          error[cont] = (label - yPredict)
          cont+=1

          if(yPredict!=label):
              escape = False
              self.weight = self.weight + (
            self.learning_rate * (label - yPredict) * inputs)

      if self.interaction >= self.interactions:
        escape = True
      self.MSE.append(np.square(error).mean())
    print('Quantidade de Interações: '+ str(
                                  self.interaction))

  def test(self, x, y):
    x = np.insert(x, 0, Perceptron.__bias, axis=1)
    yPredict = self.forward(x)
    error = (y - yPredict)
    print('MSE: '+str(np.square(error).mean())+ '%')
```

III. Adaline.

A rede Adaline *ADAptive LINear Element* foi a primeira rede rede neural a ser efetivamente utilizada pela indústria, com um algoritmo de otimização regra delta. (WIDROW; HOFF, 1960) Assim como a arquitetura perceptron, a topologia de rede neural Adaline também possui apenas um neurônio artificial,acaba sendo uma adaptação do modelo Perceptron, mas acaba se diferenciando-se pelo método de treinamento, que tem o objetivo mensurar o erro gerado pela rede, para o ajuste dos pesos sinápticos da mesma (da Silva; SPATTI; FLAUZINO, 2010).

Na Figura 2 temos a representação de um Adaline, é possível notar as entradas do neurônio $\{x_1, x_2, x_3, x_n\}$, também na Figura temos os pesos sinápticos ou sinapses do neurônio artificial $\{w_1, w_2, w_3, w_n\}$, o peso sináptico w_0, é referente ao limiar de ativação, temos um somador onde seria o corpo celular de nosso neurônio, denotado por $\sum$, seguindo o fluxo da informação temos o potencial de ativação onde é denotado por u, o bloco associado que recebe o potencial vindo do somatório, e também recebe o sinal de inferência para a saída desejada da rede, e uma função de ativação $g(.)$, e a saída do neurônio y.

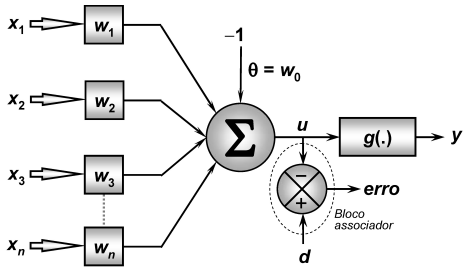

Fig. 2. Adaline.

Para realização da atividade foi necessário montar o algorítimo assim como o Perceptron, como já comentado a rede Adaline acaba sendo muito parecida com a topologia mencionada anteriormente, isso facilita para a modelagem computacional, onde foi-se utilizado a própria rede Perceptron com algumas adaptações para obter a rede Adaline.

```
class Adaline(Perceptron):

  '''
    Objeto Adaline, um único neurônio
    para realizar operações
    linearmente separáveis, tem como
    herança a classe Perceptron
    '''

  def __init__(self, features, interactions = 100,
         learning_rate = 0.01, precision = 10e-6):

      super().__init__(features, interactions,
                learning_rate)
      self.precision = precision

  def forward(self, x):
    u = np.dot(x, self.weight)
    return u

  def train(self, xtrain, ytrain):
    xtrain = np.insert(xtrain, 0, -1, axis=1)
    self.interaction = 0
    escape = False
    error = np.zeros(ytrain.shape[0])
    cont = 0
    mse = 1
    while (escape == False):
      escape = True
      cont = 0
      self.interaction += 1

      for inputs, label in zip(xtrain, ytrain):

          yPredict = self.forward(inputs)
          error[cont] = (label - yPredict)
          cont+=1
          self.weight = self.weight + (self.learning_rate *
                        (label - yPredict) * inputs)

      self.MSE.append(np.square(error).mean())

      if(np.absolute(self.MSE[-1] - mse) > self.precision):
          mse = self.MSE[-1]
          escape = False


      if self.interaction >= self.interactions:
        escape = True

    print('Quantidade de Interações: '+ str(
                                  self.interaction))
```

IV. Resultados

A. Portas Lógicas

Circuitos digitais trabalham em dois níveis de ação diferentes, valor falso ou verdadeiro, baixo ou alto, ligado ou desligado. As portas lógicas são um sistema matemático que define informações como valores numéricos. As portas lógicas são blocos de construção básicas, onde os sinais de entrada e os sinais de saída de uma porta lógica pode ser expressa por uma tabela de verdade.

Com o objetivo de testar os modelos de neurônios artificiais elaborados, é utilizado as tabelas verdade de três portas lógicas como conjunto de treinamento, sendo elas: *And, Or, Xor*. As duas arquiteturas já descritas, devem realizar a classificação dos dados das portas lógicas.

X	Y	AND	OR	XOR
0	0	A	A	A
0	1	A	B	B
1	0	A	B	B
1	1	B	B	A

TABLE I
AND, OR, XOR

Para cada caso, foi determinado um critério de parada, para a topologia Perceptron, foi adotado um limite de iterações, onde o máximo de iterações de treinamento que a arquitetura realizava para ajustar os pesos sinápticos era constante.

Para o conjunto de entrada de dados do tipo *AND*, o neurônio artificial Perceptron alcançou o resultado esperado, classificando assim todos os dados de forma correta. No treinamento para o grupo de dados do tipo *OR*, o Perceptron apresentou uma solução mais demorada, tendo assim maior número de iterações, mas foi possível realizar a classificação de todo conjunto de dados. Para base de dados da porta lógica *XOR* o Perceptron não conseguiu realizar a classificação, teve oscilações no erro *MSE*, de forma que para este conjunto de dados não foi realizável a classificação dos mesmos.

Para o neurônio Adaline, foi adotado um critério de parada do tipo precisão, onde era comparado o valor do erro atual com o anterior, até que essa diferença entre o erro *MSE* fosse menor que a precisão requerida, além do limite de iterações.

Para o conjunto de entrada de dados do tipo *AND*, o neurônio artificial Adaline alcançou o resultado esperado, classificando assim todos os dados de forma correta. No treinamento para o grupo de dados do tipo *OR*, o neurônio apresentou uma solução ótima para o problema de classificar. Para base de dados da porta lógica *XOR*, assim como o Perceptron, o neurônio Adaline não conseguiu realizar a classificação.

O resultado da relação de erro *MSE* e iterações é apresentado em forma de um gráfico. Figuras 3, 4, 5

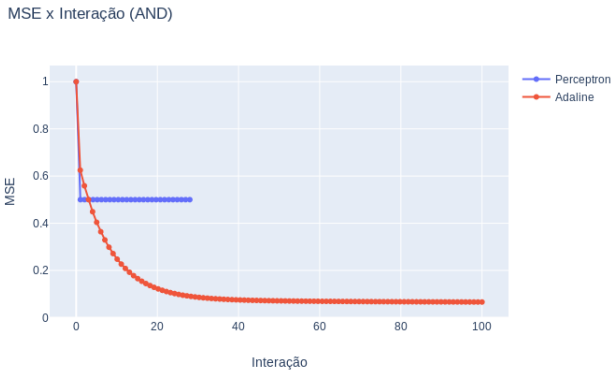

Fig. 3. AND.

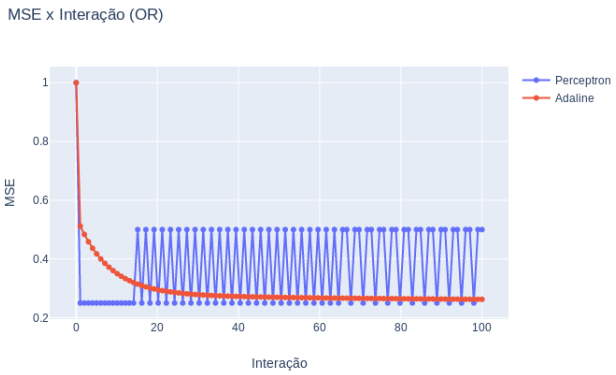

Fig. 4. OR.

B. *Projeto Prático*

A resolução computacional do projeto prático do livro de Silva, Spatti e Flausino "Redes Neurais Artificiais para engenharia e ciências aplicadas" foi realizada utilizando as mesmas arquiteturas anteriores.

V. Conclusões.

O presente relatório apresentou a aplicação de Neurônios Artificiais para classificação de padrões.

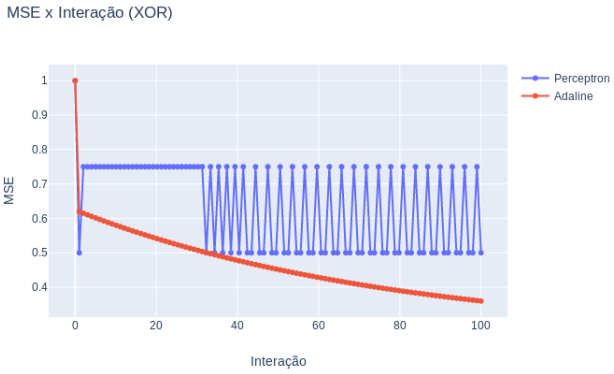

Fig. 5. XOR.

Treinamento	Valor dos Pesos Iniciais			
	w0	w1	w2	w3
T1	[0.31230848]	[0.61742236]	[0.32748302]	[0.39090078]
T2	[0.02781297]	[0.28402763]	[0.01231194]	[0.83570639]
T3	[0.37551201]	[0.18399216]	[0.23021536]	[0.20074041]
T4	[0.96905253]	[0.94319487]	[0.68331309]	[0.38303926]
T5	[0.46132795]	[0.62664875]	[0.5495158]	[0.95065024]

Valor dos Pesos Finais				Número de Iterações
w0	w1	w2	w3	
[0.25983077]	[0.9254249]	[1.23235365]	[-1.33262818]	897
[0.23680309]	[0.76721462]	[1.1787362]	[-1.14092289]	863
[0.25194143]	[0.86616376]	[1.21460049]	[-1.26405977]	870
[0.24974571]	[0.86324602]	[1.20795253]	[-1.25285994]	869
[0.24355503]	[0.82009574]	[1.19363304]	[-1.20092242]	883

Amostra	x1	x2	x3	T1	T2	T3	T4	T5
1	-1.147	2.242	7.2435	0	0	0	0	0
2	-7.970	8.795	3.8762	0	0	0	0	0
3	-10.625	6.366	2.4707	0	0	0	0	0
4	5.307	1.285	5.6883	0	0	0	0	0
5	-12.200	7.777	1.7252	0	0	0	0	0
6	3.957	1.076	5.6623	0	0	0	0	0
7	-1.013	5.989	7.1812	0	0	0	0	0
8	2.4482	9.455	11.2095	1	1	1	1	1
9	2.0149	6.192	10.9263	0	0	0	0	0

Fig. 6. Tabela 1 Resultados.

Os resultados mostram que o Perceptron, considerado o neurônio artificial mais famoso para fins didáticos, teve um desempenho ótimo de percentual de classificação, com exceção para o conjunto de dados *XOR*. Ainda assim foi possível notar que o neurônio Adaline exibiu resultados ótimos, e até melhor em relação Perceptron, mas ainda assim, não foi possível realizar a classificação de dados *XOR* usando está topologia.

References

[1] WIDROW; HOFF, 1960

[2] DA SILVA, I.N., SPATTI D., E FLAUZINO, R. (2010). Redes neurais artificiais para engenharia e ciências aplicadas: curso prático, Artliber Editora Ltda, São Paulo, SP, Brasil.

[3] HAYKIN, S. (1999) NEURAL NETWORKS: A Comprehensive Foundation. Prentice Hall, Upper Saddle River.