

Utilizando redes neurais para soluções lineares

Resolvendo funções Lógicas com Perceptron

Adhonay Junior Silva

Pontifícia Universidade Católica de Minas Gerais

Belo Horizonte, Minas Gerais

adhonay.silva@sga.pucminas.br

ABSTRACT

Veremos como a rede neural Perceptron se comporta em relação a problema linearmente separáveis como as portas AND e OR e não separáveis no caso da porta XOR.

1 INTRODUÇÃO

Rede Neural Artificial (RNA) é um modelo matemático-computacional baseado no sistema nervoso central dos seres vivos. Este modelo, que possui a capacidade de adquirir conhecimento, é composto por um conjunto de neurônios artificiais que estão interligados entre si e juntos formam a sinapse. O conhecimento de uma RNA é obtido através de um treinamento realizado a partir de uma base de dados. Neste processo os neurônios artificiais realizam cálculos a partir da entrada e se atualizam acerca da informação a cada iteração, que também é conhecida como época. No documento será tratado como a RNA Perceptron se comporta com as funções lógicas AND, OR e XOR.

2 PERCEPTRON

O Perceptron foi criado em 1958 por Rosenblatt, sendo a forma mais simples da representação de uma rede neural artificial, uma vez que é constituída de uma única camada neural e de um único neurônio. A figura 1 ilustra a rede Perceptron de única camada, onde podemos ter N entradas, mas apenas uma única saída com um valor de 0 e 1 ou de -1 e 1, dependendo do problema.

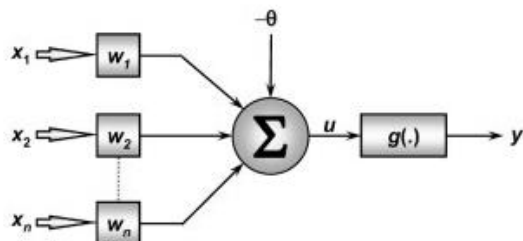
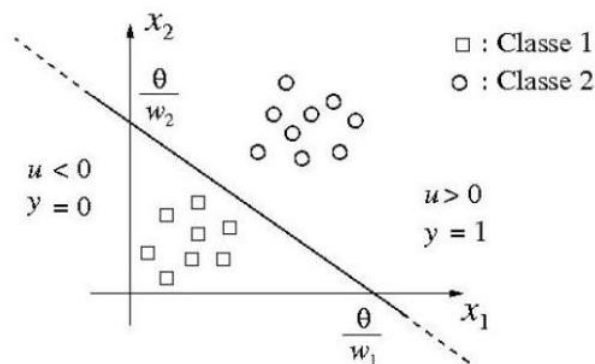


Figura 1: Rede Perceptron de uma única camada.

Os sinais de entrada $\{ X1, X2, \dots, Xn \}$: são os sinais externos normalmente em valor real ou binário. Pesos sinápticos $\{ W1, W2, \dots, Wn \}$: São valores aleatórios para balancear o valor de cada entrada da rede, esses valores são aprendidos durante o treinamento. Combinador linear $\{ \Sigma \}$: Utiliza todos sinais de entrada que foram balanceados pelos respectivos pesos como forma de produzir um potencial de ativação. Limiar de ativação $\{ \theta \}$: Especifica qual será o limite apropriado para que o resultado produzido pelo combinador linear possa gerar um valor inicial de ativação. Potencial de ativação $\{ u \}$: É o resultado obtido pela subtração do valor produzido entre o combinador linear e o limiar de ativação, caso o valor seja positivo, $u \geq 0$ então a saída é um provável valor correto, caso contrário esta errado. Função de ativação $\{ g \}$: Tende a limitar a saída de um neurônio em um intervalo valores no nosso caso entre 0 e 1. Sinal de saída $\{ y \}$: É o resultado final da rede.



Porta OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

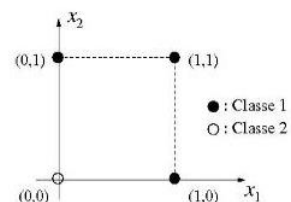


Figura 2: Função porta logica OR.

O Perceptron de uma única camada é utilizado para dividir duas classes linearmente separáveis como ilustra a figura 2, uma porta logica OR pode ser dividida por uma única reta separando os valores que contem 1 em alguma instancia em relação aos que possuem somente 0. Mas problemas para classes não linearmente separáveis como ilustrado pela figura 3 e apresentando erro, pois é impossível resolver esse problema dividindo em duas classes utilizando apenas uma reta, precisaria de pelo menos duas, que poderia ser facilmente resolvido com o Perceptron multicamadas, que trata esses tipos de resoluções não lineares.

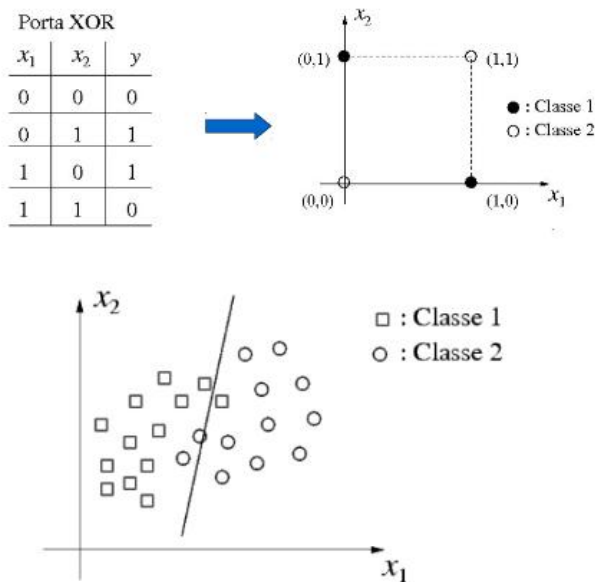


Figura 3: Função porta logica XOR.

O funcionamento do Perceptron ocorre da seguinte forma, os sinais de entradas X_i recebem seus respectivos valores e possuem pesos específicos W_i , que irão ser somados \sum , caso esta soma atinja o limiar de ativação então um sinal de saída (u) é gerado que será usado pela função de ativação, no caso do trabalho, função degrau (0 ou 1) quando se trata de um problema simples. Por ser um modelo computacional que se baseia no uso de um único neurônio o Perceptron simples está sujeito a muitos erros quando se tratar de sistemas linearmente não separáveis ou com muitas entradas, na resolução da porta XOR veremos como a saída é diferente da esperada. E mesmo para sistemas linearmente separáveis que existe solução, o valor dos pesos de ajustes W_i podem ter várias instancias dependendo de como será traçada esta reta de separação.

3 METODOLOGIA

3.1 Ambiente

Os testes foram realizados em uma máquina cujo sistema operacional é Windows 8.1, processador da mesma e um

core i5-4590 @ 3.30GHz possuindo 8GB de memória ram com uma placa dedicada NVIDIA GTX 970 com 1GB de armazenamento.

3.2 Algoritmo

Desenvolvido na linguagem programação Python 3.6 o algoritmo recebe um arquivo.txt com determinadas entradas que contém os bits que serão utilizados nas operações lógicas e pedindo para escolher qual operação desejada (AND, OR ou XOR), a taxa de aprendizagem esta sedada no próprio algoritmo, podendo mudar a mesma apenas trocando o valor da variável "TaxaAprendizado" o número de iterações segue o mesmo raciocínio.

A função de treinamento da rede recebe como parâmetros o vetor de aprendizado que é gerado a partir da quantidade de bits na entrada no arquivo de texto, mas para isso o usuário escolher qual ação vai realizar, pois dependendo dela influencia como o vetor será montado para gerar o vetor de saída correspondente ao da porta logica escolhida. Recebe o vetor de pesos(random), o vetor de entrada pelo arquivo e a taxa de aprendizagem que pode ser alterada, o valor do bias é setado como 1.

```
def FuncaoNTreinamento(VetorAprendizado, VetorPesos,
TaxaAprendizado, VetorEntrada):
    for i in range(epoca):
        for x in range(0, len(VetorAprendizado)):
            Resultado += (VetorAprendizado[x][i] * VetorPesos[i])
            + (ValorBias * VetorPesos[i])
            teste = 1 if Resultado >= 0 else 0
            if(teste != VetorAprendizado[x][i]):
                VetorPesos = AlteraPesoN(x, teste, TaxaAprendizado,
VetorPesos, VetorAprendizado)
            Resultado += (VetorEntrada[x] * VetorPesos[x])
            teste = 1 if Resultado >= 0 else 0
        return teste
```

O processo de treinamento tem como objetivo calibrar os pesos de modo iterativo, partindo de valores aleatórios. A taxa de aprendizagem {TaxaAprendizado} (um valor entre 0 e 1) diz o quão rápido a rede chega ao seu processo de classificação, um valor muito pequena ou muito alto pode levar para valores fora do ajuste ou retardar o aprendizado influenciando na resolução da solução.

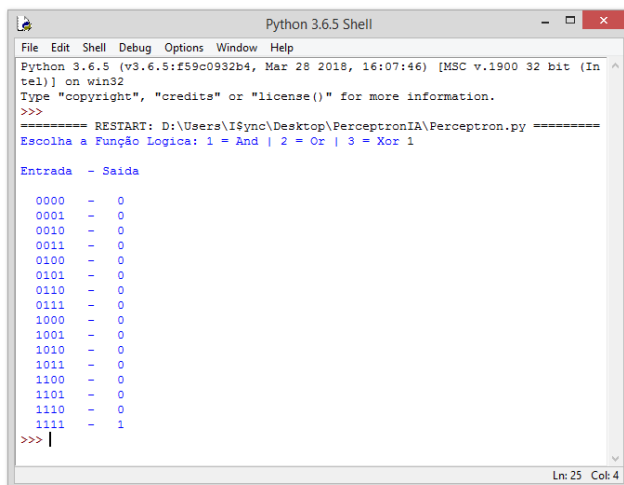
```
def AlteraPesoN(x, teste, TaxaAprendizado, VetorPesos,
VetorAprendizado):
    for y in range(0, len(VetorPesos) - 1):
        VetorPesos[y] = VetorPesos[y] + (TaxaAprendizado *
(VetorAprendizado[x][y] - teste) * VetorAprendizado[x][y])
        VetorPesos[x] = VetorPesos[x] + (TaxaAprendizado *
(VetorAprendizado[x][y] - teste) * ValorBias)
```

Assim, a função alteraPesoN onde o vetor contendo os pesos de um passo de iteração será o resultado da soma de si mesmo no passo anterior com o produto das seguintes parcelas: a taxa de aprendizagem, a amostra de aprendizagem desse passo e a diferença entre o valor desejado, saída "correta" onde essa diferença é chamada de erro de saída: se for diferente de zero, é aplicada a correção.

3.3 Testes

Foi utilizado o Python 3.6.5 para execução dos seguintes testes, foram testadas as 3 portas lógicas com tamanho diferente de entrada em relação a XOR e com variação na taxa de aprendizado das portas AND e OR e analisando o comportamento do Perceptron em relação elas.

Porta lógica AND (E) (também é chamada de conjunção lógica) é uma operação lógica em dois operandos que resulta em um valor lógico verdadeiro somente se todos os operados têm um valor verdadeiro[1]. Como ilustra a figura 4 podemos notar que o único valor de saída é igual 1 que retrata a entrada 1111, equivalente a multiplicação pois $4 \times 1 = 1$ sendo as outras saídas iguais a zero, pelo fato de conter algum bit 0 na entrada, a saída do Perceptron foi satisfeita usando uma taxa de aprendizado de 0.6, conseguiu resolver o problema linear de forma correta.



```

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\Users\ISync\Desktop\PerceptronIA\Perceptron.py =====
Escolha a Função Lógica: 1 = And | 2 = Or | 3 = Xor 1

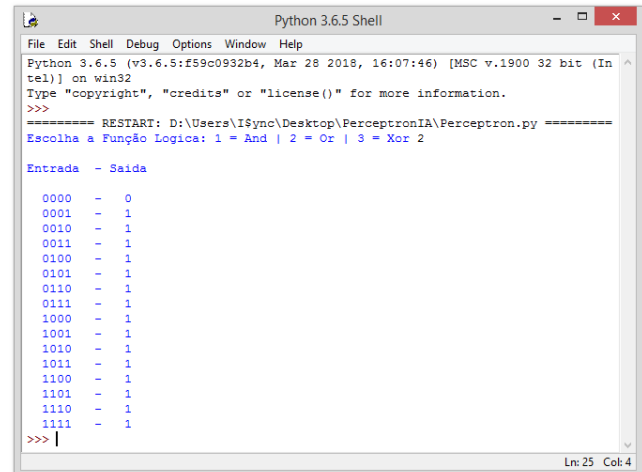
Entrada - Saída
0000 - 0
0001 - 0
0010 - 0
0011 - 0
0100 - 0
0101 - 0
0110 - 0
0111 - 0
1000 - 0
1001 - 0
1010 - 0
1011 - 0
1100 - 0
1101 - 0
1110 - 0
1111 - 1
>>>

```

Figura 4: Execução Perceptron porta logica AND.

Enquanto a porta lógica OR (OU), também é chamada de disjunção lógica, é uma operação lógica entre dois ou mais operandos que resulta em um valor lógico falso se, e somente se, todos os operandos tiverem um valor falso[1]. A figura 5 ilustra exatamente esse caso, onde existe apenas um valor que não é falso no caso da saída 0, onde sua primeira entrada 0000 e nenhum valor falso, sendo a única de saída zero, a taxa de

aprendizado utilizada foi de 0.4, novamente como se trata de um problema linearmente separável o Perceptron obteve a solução correta pra saída.



```

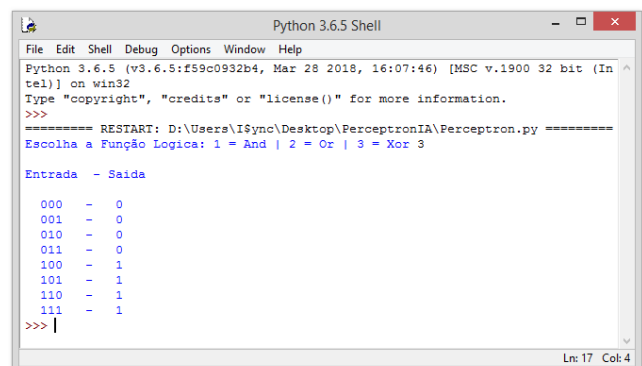
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\Users\ISync\Desktop\PerceptronIA\Perceptron.py =====
Escolha a Função Lógica: 1 = And | 2 = Or | 3 = Xor 2

Entrada - Saída
0000 - 0
0001 - 1
0010 - 1
0011 - 1
0100 - 1
0101 - 1
0110 - 1
0111 - 1
1000 - 1
1001 - 1
1010 - 1
1011 - 1
1100 - 1
1101 - 1
1110 - 1
1111 - 1
>>>

```

Figura 5: Execução Perceptron porta logica AND.

Entretanto a porta XOR(Ou exclusivo ou disjunção exclusiva), é uma operação lógica entre dois operandos que resulta em um valor lógico verdadeiro se e somente se o número de operandos com valor verdadeiro for ímpar. Considerando o valor 0 como falso e 1 como verdadeiro, podemos notar que o Perceptron não conseguiu a solução ideal para o problema, a taxa de aprendizado foi de 0.6 e como o problema não é linearmente separável ele não foi capaz de acertar todo o problema, dividindo em duas classes valores pertencentes de uma com a outra e assim não garantindo a solução, podemos notar que ele gera a saída errada para as entradas 001 e 010 que deveriam ter saída 1, para as entradas 101 e 110 que tiveram saída igual a 1 e deveria ser 0.



```

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\Users\ISync\Desktop\PerceptronIA\Perceptron.py =====
Escolha a Função Lógica: 1 = And | 2 = Or | 3 = Xor 3

Entrada - Saída
000 - 0
001 - 0
010 - 0
011 - 0
100 - 1
101 - 1
110 - 1
111 - 1
>>>

```

Figura 6: Execução perceptron porta logica XOR.

4 CONCLUSÃO

Após a execução dos testes, notasse que realmente o Perceptron com apenas uma camada não consegue ter uma resposta eficiente para separar as classes dos problemas

lógicos testados, mas foi capaz de solucionar os linearmente separáveis com eficiência, em relação a taxa de aprendizado e alterando o número de iterações para os problemas lineares em algumas iterações o Perceptron não foi capaz de solucionar o problema, isso deve-se ao fato da reta que divide os resultados da OR(função lógica que foi testada com uma taxa de 0.1 e com 5 iterações) estar mais distante da origem assim precisando de mais iterações ou de uma taxa de aprendizado maior. Mas colocando uma taxa satisfatória para os problemas lineares junto com iterações o Perceptron com uma camada não terá problema em dar a saída correta.

REFERÊNCIAS

- [1] Silva, Ivan N.; Spatti, Danilo H.; Flauzino, Rogério A; Redes Neurais Artificiais para engenharia e ciências aplicadas, Artliber, 2010.
- [2] TOCCI, Ronald J.; WIDMER, Neal S.; MOSS, Gregory L. Sistemas digitais: princípios e aplicações. 10. ed. São Paulo: Pearson Prentice Hall, c2007.
- [3] NOBRE, Cristiane Neri. Redes neurais: Algoritmo do Perceptron. PUC-MG: [s.n.], 2018. 83 p.
- [4] JAIN, A. K, MAO, J., MOHIUDDIN, K.M. Artificial neural networks: a tutorial. IEEE Computer, v. 29, n. 3, p. 56-63, 1996.