Eric Azevedo de Oliveira
Disciplina: Inteligência Artificial
DATA: 27/05/2022

1 PERCEPTRON

1.1 Explicação

Antes da própria explicação do Percptron, devo explicar oque seria um neurônio proposto por "McCulloch-Pitt", que basicamente ele é um conjunto de sinapses, iguais os nossos neurônio em si, mas que só fazem apenas duas coisas, somar os sinais de entrada, e realizar uma função para mandar esse sinal já somado para a saída. Essa função, realizadas apos o somatório, é chamada de Duncao de ativação, que serve só para restringir a amplitude da saída do neurônio, deixando assim a saída ser de 0 ou 1. A função de ativação é super simples, pois, para restringir essa amplitude ela utiliza a ideologia de mais próximo de si.
Pôs finalização da explicação do neurônio em si, irei dar continuação a explicação do Perceptron, basicamente ele é composto pelo neurônio referido acima, com a função linear, e com o Aprendizado Supervisionado. Basicamente esse era o Percptron proposto, mas ainda faltava a fórmula de Adaptar o Peso dos vetores de entrada, com isso se deu origem a fórmula:

$$w(n-1) = w(n) + N[d(n) - y(n)]x(n)$$

Onde: w(n) = peso
d(n) = Resposta desejada
X(n) = Vetor de entrada
b(n) = Bias (Irei explicar sua função)
y(n) = Resposta real
N = Taxa de aprendizado(Irei aprofundar sobre isso)

1.2 Explicação Algotitmo

Apos breve explicação do seu conjunto em si, irei explicar como o algoritmo em si, funciona. Inicialmente devemos entender que o Percptron só funciona com funções lineares com isso portas logicas como a Xor não ira funcionar nele(irei explicar o motivo). Com isso iremos entrar com a porta logica And para seu meu valor X e entrar com sua resposta que sera meu valor Y, e definindo a taxa de aprendizado como 0,03, com isso iremos montar a estrutura do

Percptron , colocando inicialmente os pesos dos vetores como randômicos, e iremos iniciar a Rede neural. Os pesos irão entrar e passar pelos vetores chegando assim no neurônio, onde ele ira pegar o valor de entrada e somá-los com o valor do peso que foi atribuído aleatoriamente para o vetor, com isso iremos aplicar a função de ativação, que nos dirá se a saída é 0 ou 1, e apos isso iremos comparar a saída com a saída esperada, se a saída for igual à saída esperada não atualiza os pesos e colocamos outra entrada para passar pelo mesmo processo, mas se a saída for diferente iramos aplicar a fórmula de alteração dos pesos e com isso colocar a nova entrada com o reajuste já feitos, e iremos fazer isso ate uma quantia de épocas informadas na iniciação do programa, ou ate que nosso programa para de errar continuamente.
Nesse processo do algoritmo, existe uma entrada chamada Bias que sempre tera o peso de 1, utilizamos essa entrada, pois, se em um programa nosso resultado der sempre 0, a fórmula da de ajustar os pesos sempre iram se multiplicar por 0, com isso nunca ira ocorrer uma atualização dos pesos, por isso que utilizamos o bias, pois, ela forca que mesmo a entrada seja 0 que nosso peso ao ser errado tenha uma mudança.

Além disso, a pre determinação da taxa de aprendizado, é algo fundamental para o funcionamento rápido ou demorado de nossa rede, pois, ela e implícita em nossa fórmula de ajuste dos pesos, e se for um número muito alto, nossa Rede pode se super ajustar fazendo assim passar do ponto que queremos, ou demorar muito a chegar no ponto que queremos.

1.3 Codigo

O Código feito se encontra em:
(https://github.com/Quebec-Eric/4Periodo/blob/master/IA/Lista7/Pers.ipynb)
Basicamente, os paramentos passados são a matriz que já esta, contido o Bias, o vetor com os resultados que procuro., a taxa de aprendizado que coloquei como 0,5, e a quantidade de épocas que defini como 50. Pôs o envio desses paramentos, zero os valores dos pesos, crio um vetor de saída e um vetor de erros, e início meu trinamento. Ele vai somar todos os valores de peso com a entrada e fazer a função de ativação, que ira decidir se o somatório ira para 0 ou para 1, e com isso, faco a atualização te todos os vetores pesos com a fórmula. Apos isso eu pego todos os erros e faco erros ao quadrado. Apos de N vezes rodando a função me retorna um array dos erros, que ao printar percebo se ele consegue aprender fazendo que em um momento os erros em questões seja 0.0 , ou se não consegui aprender sendo que a saída desse vetor ira ser diferente que 0.0.

1.4 Função booleana AND

Como A função AND é uma função linear o Percptron consegue resolver como mostra as imagens a seguir:

```
print(FazerPErc(x, y, 0.5, 50)[1])
✓ 0.3s
[0.5, 1.5, 1.5, 1.0, 0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Entrada função And:

```
# entrada  AND
x = [[1., 0., 0.],
     [1., 0., 1.],
     [1., 1., 0.],
     [1., 1., 1.]]
```

Como é possível notar a porta logica AND , e'uma função linear , onde o é possível por meio do Perceptron achar o resultado esperado.Que apos 5 interações foi possível aprender e com isso ela começou a parar de errar.

1.5 Função booleana OR

Como A função OR é uma função linear o Percptron consegue resolver como mostra as imagens a seguir:

```
print(FazerPErc(xOR, yor, 0.5, 50)[1])
✓ 0.1s
[0.5, 1.0, 0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Entrada Função OR:

```
xOR= [[1., 0., 0.],
      [1., 0., 1.],
      [1., 1., 0.],
      [1., 1., 1.]]
[9] ✓ 0.4s
```

Como é possível notar a porta logica OR , e'uma função linear , onde o é possível por meio do Perceptron achar o resultado esperado.Que apos 3 interações foi possível aprender e com isso ela começou a parar de errar.

1.6 Função booleana XOR

Já a função boleana Xor , o Percptron não ira conseguir resolver, pois, nesse caso a função não e linear,se tivéssemos mais um neurônio seria possível resolver esse problema que sera resolvido no Backpropagation. Pois, quando iremos realizar uma função não linear, sua demonstração no gráfico, não e possível sera uma camada da outra com apenas uma linha, mas sim com duas, e assim em diante.
Como mostra o exemplo a seguir o Percptron não consegue realizar o aprendizado:

```
print(FazerPErc(xor, yx, 0.5, 50)[1])
✓ 0.7s
[1.0, 1.5, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0]
```

Entrada Função XOR:

```
xor =[[1., 0., 0.],
      [1., 0., 1.],
      [1., 1., 0.],
      [1., 1., 1.]]
✓ 0.1s
```

Com a analise , percebemos que um neoronio sozinho nao consegue aprender , e com isso ele erra para sempre mostrando 2.2 .....

2 BACKPROPAGATION

2.1 Explicação

Backpropagation já é uma melhoria do Percptron, pois ele alem de resolver problemas lineares, ele resolve não lineares. Ele é um algorítimo que tem multi-Camadas mais difundido, se baseia no aprendizado Supervisionado com Correção de erros.
Ele tem a mesma ideia na fase de propagação que o Percptron , seus vetores têm pesos e com isso ao chegar no neurônio ele ira fazer a soma do peso com a entrada, mas ao invés de fazer a função de ativação iremos fazer a função sigmóidal:

$$\frac{1}{1 + e^{(-av)}}$$

No qual av é o somatório de todos os (pesos) mais entradas que chegam naquela neurônio.
Depois de realizar a sgmoidal esse resultado ira para a próximo neurônio e com isso o peso do vetor ira ser somado com o resultado na sgmoidal e novamente ira ser aplicado outra sgmoidal, e esse passa ira chegar, ate a saída.
Chegando na saída iremos analisar se o resultado da saída foi esperado ou não, se foi esperado,

só iremos iniciar novamente com outro input, mas se não foi esperado iremos fazer a propagação para trás. Como não sabemos quem errou temos re calcular o erro de todos e ajustar todos os vetores presentes. Com isso não iremos utilizar a diferença absoluta, mas sim o erro médio quadrático:

$$MSE = \sum (Si - Oi)^2)$$

E se pertencer à camada interna irei utilizar a fórmula.

$$\delta = f'(net)x \sum \delta w$$

Onde :
Delta = Erro da camada posterior.
W= Peso das Conexões.
Com isso, o erro das camadas posteriores estão em um total vínculo com as camadas anteriores.
E depois de calcular todos os erros da Rede iremos ajustar os pesos que a fórmula é;

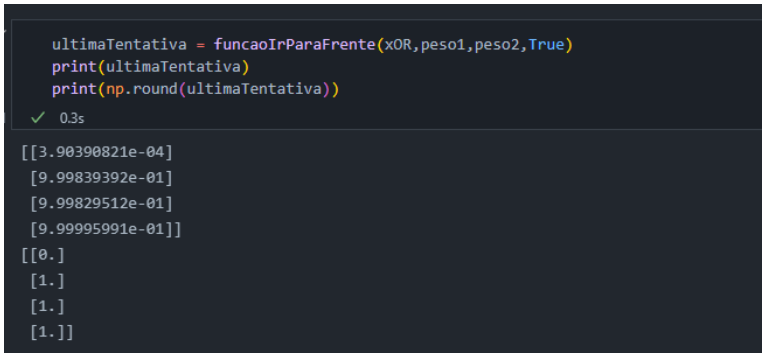
$$w(t+1) = w(t) + ta * Xi(t) * erroj(t)$$

Onde:
w = Peso entre um neurônio i e o j Erroj = indica o erro associado ao I-ésimo neurônio.
Xi indica a entrada recebida.
E depois disso, o algorítimo ira passar por épocas ate começas a acertar, ou terminar a quantidade de épocas que foi passado.

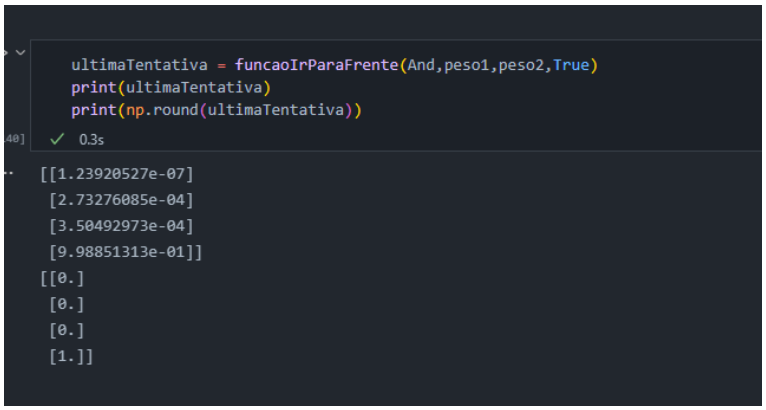
2.2 Codigo

O Código pode ser encontrado em:
(https://github.com/Quebec-Eric/4Periodo/blob/master/IA/Lista7/BackP.ipynb)
Inicialmente, crio a matriz com as portas logicas, e com o bias já incluso, depois faco outro array com os valores respostas desejados. Com isso crio os vetores com pesos randômicos micos e defino a taxa de atualização quando o tempo de eras.
Com isso partindo do ponto do tempo de eras nosso programa inicia com a função de Ir para frente, que faz todos os cálculos e somatórios necessitários para chegar a resposta no final, apos essa resposta irei fazer a volta , contanto os erros e fazendo as somas. Apos a ida e volta atualizo os pesos com a função sgmoidal , e os contabilizo. Com isso apos todo o processo envio novamente a entrada para a função e como todos os pesos já estão atualizados corretamente, ele ira me retornar a resposta correta.

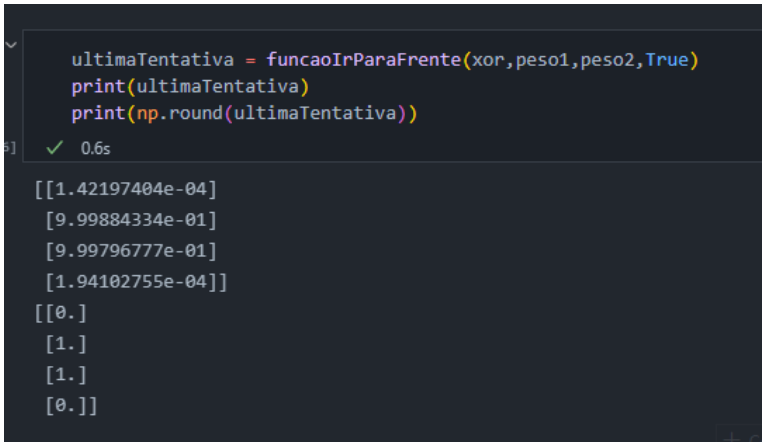
2.3 Função booleana And



2.4 Função booleana OR



2.5 Função booleana XOR



Podemos observar, que a função consegue resolver problemas não lineares, mostrando em si os resultados.