CENTRO UNIVERSITÁRIO
UNI METROCAMP

Inteligência Artificial

Prof Eduardo Nunes

Aluno: Marcelo Pedroni da Silva

RA: 202051855029

Projeto Algoritmo Genético – AV2 (2,5 pontos)

Rede Neural Artificial

1 – Rodar o algoritmo com 100 ciclos de treinamento e taxa de aprendizagem de 0,5. Anotar os valores finais de treinamento, tanto os valores dos pesos, quanto o resultado obtido após treinamento.

Setando as configs:

R:

```
def sigmoidDerivada(sig):
    return sig * (1 - sig)


entradas = np.array([[0,0],
                     [0,1],
                     [1,0],
                     [1,1]])

saidas = np.array([[1],[0],[0],[1]])

pesos1 = np.array([[-0.424, -0.740, -0.961], # pesos de p1 a p6
                   [0.358, -0.577, -0.469]])

pesos2 = np.array([[-0.017], [-0.893], [0.148]])


ciclos_treinamento = 100
taxaAprendizagem = 0.5
```

Resultados do primeiro ciclo:

```
(base) marcelo@marcelo-Inspiron-5437:~/
 multicamada.py
camada saida
[[0.40588573]
 [0.43187857]
 [0.43678536]
 [0.45801216]]
Erro: 0.5011915107628695
```

Resultados do centésimo ciclo:

```
camada saida
[[0.51140096]
 [0.50648187]
 [0.49603823]
 [0.49725456]]
Erro: 0.49846614492331476
Peso 1 [[-0.40820879 -0.60088421 -1.08712791]
 [ 0.36567553 -0.36756142 -0.69515176]]
Peso 2[[ 0.16322582]
 [-0.6213635 ]
 [ 0.55130519]]
(base) marcelo@marcelo-Inspiron-5437:~/Documentos/Unimetroca
```

2- Rodar o algoritmo com 100 ciclos de treinamento e taxa de aprendizagem de 5. Comparar com os valores obtidos no item 1. Analisar se o resultado melhorou e explicar o porquê.

R:

Setando as configs:

```python
def sigmoidDerivada(sig):
    return sig * (1 - sig)



entradas = np.array([[0,0],
                     [0,1],
                     [1,0],
                     [1,1]])

saidas = np.array([[1],[0],[0],[1]])

pesos1 = np.array([[-0.424, -0.740, -0.961], # pesos de p1 a p6
                   [0.358, -0.577, -0.469]])

pesos2 = np.array([[-0.017], [-0.893], [0.148]])



clos_treinamento = 100
taxaAprendizagem = 5
```

Resultados do primeiro ciclo:

```
(base) marcelo@marcelo-Inspiron-5437:~/Do
 multicamada.py
camada saida
[[0.40588573]
 [0.43187857]
 [0.43678536]
 [0.45801216]]
Erro: 0.5011915107628695
```

Resultados do centésimo ciclo:

```
camada saida
[[0.83620324]
 [0.31676754]
 [0.35746976]
 [0.5428979 ]]
Erro: 0.323784042285435
Peso 1 [[-0.05840052 -1.09879523 -5.02488119]
 [ 1.52801949 -0.52212543 -4.85553906]]
Peso 2[[ 1.2345353 ]
 [-4.94984991]
 [ 6.98039781]]
(base) marcelo@marcelo-Inspiron-5437:~/Document
```

Comentários:

Claramente podemos observar que na segunda tentativa, ao aumentarmos para 5 a taxa de aprendizagem, os resultados foram mais acurados, embora não tenham alcançado o objetivo de treinar a rede até não haver erros.

O erro, ao final, é de 0.323 (arredondando) contra 0.498 do primeiro exercício.

Os valores que deveriam se aproximar de [1, 0, 0, 1] foram de:

Ex. 1: [0.511, 0.506, 0.496, 0.497]

Ex. 2: [0.836, 0.316, 0.357, 0.542]

Tais dados nos revelam que os valores do Ex. 2 estão mais próximos ao objetivo do que o 1, em concordância com o erro, que está mais próximo ao objetivo (0).

3 – Podendo mudar a quantidade de ciclos de treinamento e/ou a taxa de aprendizagem procure obter melhores resultados e justifique suas mudanças.

R:

Creio ser uma questão de ajuste fino entre ambas as características.

Para este exemplo em questão não são necessárias tantas interações, e um step de uma ou duas ordens de grandeza menor já bastam.

Para isso realizei diversos testes e encontrei a sintonia em:

ciclos de treinamento: 10000

taxa de aprendizagem: 10

Obtendo os resultados:

```
camada saida
[[0.99285427]
 [0.00934538]
 [0.01110844]
 [0.99113244]]
Erro: 0.009116777192094914
Peso 1 [[-6.78631733 -1.546246   -8.86948037]
 [13.2106618   0.91511932 -7.5637139 ]]
Peso 2[[ 13.65067985]
 [-25.66435281]
 [ 21.88191955]]
(base) marcelo@marcelo-Inspiron-5437:~/Documentos/Unimetrocam
```

Notei que ao aumentar a taxa de aprendizagem, principalmente o último valor que deveria se aproximar de 1 não convergia tão bem, ao aumentar para 20 a taxa de aprendizagem, o valor convergiu para 0.4999.

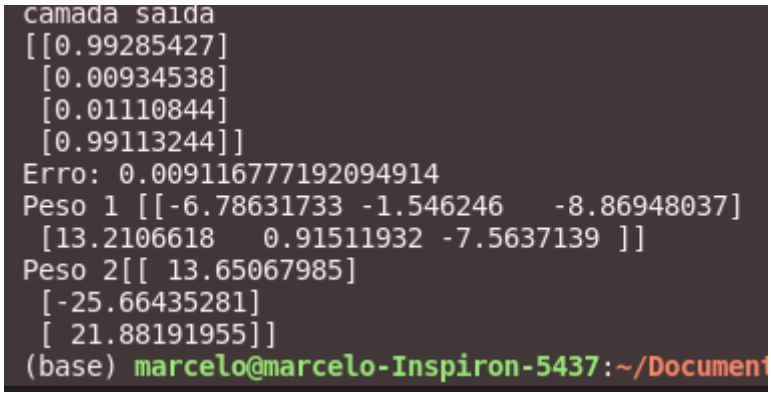
No exercício 2, pudemos notar que diminuir a taxa também não funciona bem (quando mantêm-se o ciclo de treinamento constante).

Assim, restava testar outros valores de ciclo de treinamento.

Ao extrapolar para valores de 1 ou 2 ordens de grandeza maiores, os resultados alcançados são pouco melhores, que não justificam o gasto computacional envolvido.

Ao simular ciclo de treinamento de 100000 (uma ordem de grandeza a mais) obtive resultados de erro melhores em termos de apenas 0,006363199 em relação ao ciclo de treinamento de 1000.

Quanto aos valores obtidos, foram praticamente os mesmos.



Logo, conclui-se que se desconsiderarmos o gasto computacional para executar com maiores precisões, os resultados obtidos com ciclo de treinamento = 100000 e taxa de aprendizado de 10 seria mais apropriado, porém em questões realistas, o gasto computacional de fato não compensa a mínima diferença percebida, assim uma ótima escolha seria “setar” ciclo de treinamento em 10000 e taxa de aprendizado em 10.