Atividade Avaliativa 3 - Redes Neurais Artificiais

[1]Chrystian Arriel Amaral
[2]Henrique Silva Rabelo

[1]Turma 10A do curso de Ciência da Computação, Universidade Federal de Lavras, C.P. 3037, 37200–000, Lavras, MG, Brasil.
[1]Turma 14A do curso de Sistemas de Informação, Universidade Federal de Lavras, C.P. 3037, 37200–000, Lavras, MG, Brasil.

28 de fevereiro de 2023

1 Introdução

Com o objetivo de colocar em prática a teoria das Redes Neurais Artificiais (RNA), esse trabalho propõe modelar e implementar uma solução em Python, a qual vai reproduzir uma RNA previamente definida. Com essa mesma solução, vamos treinar e obter uma predição com base em um conjunto de dados, também previamente definidos. Por fim vamos apresentar os resultados positivos obtidos e discutir possíveis melhorias.

2 Desenvolvimento

2.1 Função de Simulação

Tem como principal objetivo calcular o valor dos neurônios presentes na rede. Sua função, é receber a entrada X no neurônio de entrada e fazer os cálculos necessários levando em conta os pesos das arestas e também considerando a função de ativação de cada neurônio.

Para sua execução, são necessários 3 parâmetros:
x - valor de entrada na rede neural
weights - pesos das arestas usados para os cálculos
bias - usado para ajustar a saída junto da soma ponderada das entradas para o neurônio

```
def simulate(x, weights, bias)
```

Como retorno da função, teremos uma lista com o valor de todos os 4 neurônios da nossa rede neural. É importante resaltar que esses valores calculados já passaram pela sua função de ativação: f2(x) = relu(x), e f1(x) = f3(x) = f4(x) = sigmoidal(x).

```
return [neuron_1, neuron_2, neuron_3, neuron_4]
```

2.2 Função de Calculo dos Deltas

Regra Delta - implementa um gradiente descendente em cima erro para funções de ativação lineares.

A função é responsável pelo cálculo do erro de cada neurônio individualmente. Essa apuração é feita com base no valor que foi calculado pela rede com a entrada X que resultou na saída Y comparado com o valor esperado presente na amostra $sample_y$.

Para obter o valor de Y, foi utilizada a função *simulate*(), a partir disso, com os valores de Y para aquela entrada podemos calcular o erro geral da simulação e consequentemente o delta para neurônio.

Para o uso da função:
Y - lista com valores dos neurônios calculados
$sample_y$ - valor final esperado presente na amostra de treinamento
train_weights - valor dos pesos das arestas

```
def calculate_deltas(y, sample_y, train_weights)
```

Inicialmente será calculado o delta do neurônio de saída, que seria

```
delta_saida = calculate_error(y[NEURON_1_INDEX],sample_y) *
    sigmoid_derivate(y[NEURON_1_INDEX])
```

A partir disso, os demais deltas serão calculados com base no delta de saída, pesos e derivada da função de ativação de cada neurônio. Exemplo:

```
delta_neuron_3 = delta_neuron_1 * weigths_1[1] * sigmoid_derivate(y[NEURON_3_INDEX])
```

Tabela 1: Sample 2

x	y
0.0	1.0
0.5	0.1

Tabela 2: Pesos predefinidos

3	0	0	0
−4	1	0	0
−1	−3	0	0
0	0	2	−10

Ao final da execução, teremos como retorno a lista de deltas que serão usados posteriormente para a atualização de pesos,bias e consequentemente, treinamento da rede neural.

```
return [delta_neuron_1, delta_neuron_2, delta_neuron_3, delta_neuron_4]
```

2.3 Função de Atualização dos Pesos

A função realiza a atualização dos pesos com base nos deltas calculados e nos resultados de cada neurônio. O calculo para cada peso é

```
peso = deltas[i] * Y[i] * learning_rate
```

sendo i o índice do neurônio e *learningrate* a taxa de aprendizagem. Para os pesos entre as entradas e os neurônios da primeira camada, temos o seguinte cálculo

```
peso = deltas[i] * X * learning_rate
```

2.4 Função de Atualização dos Bias

Semelhante a atualização dos pesos, o bias são atualizados com base nos deltas. Temos como o calculo o seguinte

```
bias_neuron = bias_neuron - deltas[i] * learning_rate
```

sendo i o índice do neurônio e *learningrate* a taxa de aprendizagem.

2.5 Função de Treinamento

Tem a função de treinar em loop a rede neural. A cada execução, serão calculados os valores com a partir de uma entrada e com base na saída os valores dos pesos e bias serão atualizados.

Dessa forma, a cada iteração, assume-se que com bases na atualização do pesos e bias, os valores obtidos na simulação se aproximem das amostras e o percentual de erro diminua.

Por fim, ao final do loop, teremos os pesos e o bias calculados e aproximados do ideal.

3 Resultados

Os resultados foram obtidos com o *notebook python* em anexo a esse relatório.

3.1 Conjunto menor e função test

Realizamos um teste da função **test**, a qual utiliza a rede neural para obter o valor de saida y para uma entrada x. Nesse caso usamos o conjunto de dados da tabela 1. Como os pesos usamos a matriz predefinida, representada na tabela 2, onde as colunas representam os neurônios 1, 2 ,3 e 4, respectivamente e as linhas os neurônios 2, 3, 4 e 5, respectivamente. Os resultados obtidos da rede neural não foram iguais aos corretos. A figura 2 contém o gráfico dos resultados. Esse resultado era esperado, dado que não houve treinamento da rede neural, algo que não seria eficiente com poucos dados.

A figura 1 contém o gráfico de comparação dos dados esperados com os obtidos pela predição.

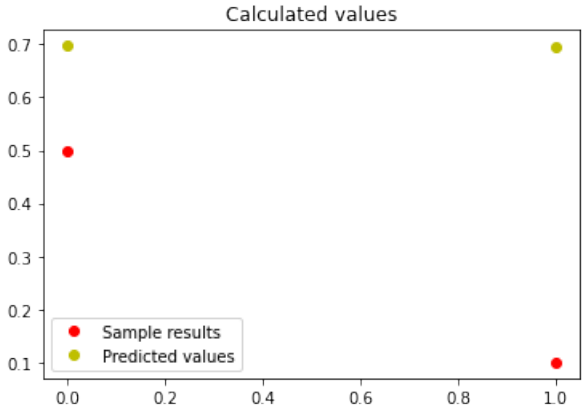

Figura 1: Resultados da predição do conjunto Sample 2 comparados com os corretos.

3.1.1 Conjunto maior e treinamento da RNA

Para o treinamento da RNA, foi utilizado um conjunto maior de dados (tabela 1), onde rodamos 40 mil gerações, com uma taxa de aprendizagem de 0.0045. Os pesos iniciais foram os mesmos citados anteriormente (tabela 2). Como resultado obtivemos uma média de erro de 2.88819, aproximadamente.

Para definir o valor da taxa de aprendizagem, foi necessario calibrar, testando diferentes valores ate obter um bom resultado.

A tabela 4 demonstra os valores obtidos por fim ao realizar o teste com a rede neural já treinada.

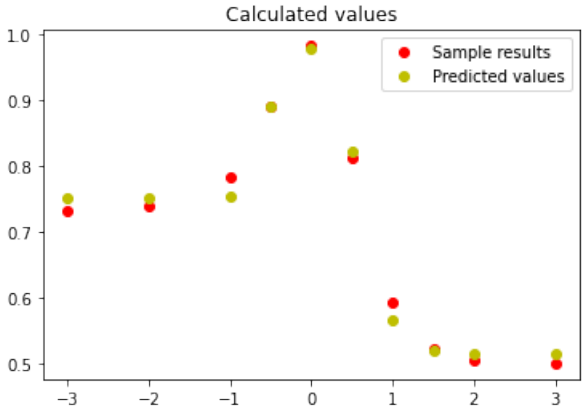

Figura 2: Resultados da predição do conjunto Sample 1 comparados com os corretos.

4 Conclusão

Foi possível concluir que o treinamento da rede neural, com um conjunto maior de dados, torna o resultado melhor.

Tabela 3: Sample 1

x	**y**
-3.0	0.73212
-2.0	0.7379
-1.0	0.7838
-0.5	0.8903
0.0	0.9820
0.5	0.8114
1.0	0.5937
1.5	0.5219
2.0	0.5049
3.0	0.5002

Tabela 4: Resultados para o conjunto Sample 1

x	$calculated_y$	$sample_y$
-3.0	0.7508941092263605	0.73212
-2.0	0.7508772773249369	0.7379
-1.0	0.7525099013677723	0.7838
-0.5	0.889983833577383	0.8903
0.0	0.9785956271602927	0.982
0.5	0.8229242508433527	0.8114
1.0	0.5646628755454731	0.5937
1.5	0.5193178159532837	0.5219
2.0	0.5146261940782666	0.5049
3.0	0.514116934179838	0.5002