Relatório de Redes Neurais

Alunos: Anabelle Elizabeth Araujo de Souza, Gabriel Luiz dos Santos Silva e Jules Severo Barcos

Prof. Me. Alexandre Garcia de Oliveira

Santos, 02 de junho de 2023

CONTROLE DE VERSÃO			
Autor	**Versão**	**Data**	**Descrição**
Anabelle Souza	1.0	01/06/2023	Edição do documento
Jules Severo	2.0	01/06/2023	Edição do documento
Gabriel L. S. Silva	3.0	01/06/2023	Edição do documento

Sumário

1 Definição de Redes Neurais

Redes neurais são modelos computacionais que se baseiam no funcionamento do cérebro humano. Elas são desenvolvidas para resolver problemas complexos de aprendizado e identificação de padrões. Esses modelos consistem em unidades de processamento chamadas de neurônios artificiais, que são conectados uns aos outros por meio de conexões ponderadas.Cada neurônio artificial recebe um conjunto de entradas, que são multiplicadas pelos valores dos pesos correspondentes. Essas entradas são processadas por meio de uma função de ativação não-linear, que introduz não-linearidade nos resultados dos neurônios. A saída de um neurônio pode ser enviada como entrada para outros neurônios, formando assim uma rede de neurônios interligados.As redes neurais se destacam por sua capacidade de aprender e generalizar a partir de exemplos fornecidos. Durante o processo de treinamento, os pesos das conexões entre os neurônios são ajustados de forma iterativa. Isso é feito utilizando algoritmos de otimização que minimizam a diferença entre a saída esperada e a saída real da rede neural. Esse ajuste de pesos permite que a rede neural reconheça padrões e tome decisões com base nos dados de entrada.

2 Redes Neurais: Peso x Bias

Em redes neurais, pesos (weights) e bias são dois componentes fundamentais para o funcionamento e aprendizado do modelo. Eles são usados para ajustar e controlar a atividade dos neurônios artificiais em cada camada da rede.

Pesos (weights): Os pesos são valores associados às conexões entre os neurônios em uma rede neural. Cada conexão entre dois neurônios é representada por um peso, que indica a força ou a importância daquela conexão. Os pesos são ajustados durante o treinamento da rede neural, buscando encontrar a melhor configuração que permita que a rede aprenda a mapear corretamente os dados de entrada para as saídas desejadas. Os pesos determinam como os sinais são propagados pela rede, sendo multiplicados pelas entradas correspondentes e somados para produzir a saída de cada neurônio.

Bias (viés): O bias, também conhecido como termo de viés, é um parâmetro adicional em cada neurônio, utilizado para introduzir um deslocamento na função de ativação. O bias permite que a rede neural ajuste a curva de ativação, controlando o ponto de partida ou o limiar a partir do qual a ativação ocorre. Ele adiciona um valor constante à soma ponderada dos sinais de entrada, antes de passá-los pela função de ativação. O bias é ajustado juntamente com os pesos durante o treinamento para otimizar o desempenho da rede. .

3 Descrição de Classificação do Data Set Câncer

Estamos utilizando o conjunto de dados sobre o câncer de mama, na qual possuem 5 colunas com as informações dos pacientes e 1 coluna com o diagnóstico. Este conjunto de dados de câncer de mama foi obtido nos Hospitais da Universidade de Wisconsin, Madison, do Dr. William H. Wolberg, onde pode ser feito o download pelo Kaggle.

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
50	11.760	21.60	74.72	427.9	0.08637	1
51	13.640	16.34	87.21	571.8	0.07685	1
52	11.940	18.24	75.71	437.6	0.08261	1
53	18.220	18.70	120.30	1033.0	0.11480	0
54	15.100	22.02	97.26	712.8	0.09056	0
55	11.520	18.75	73.34	409.0	0.09524	1
56	19.210	18.57	125.50	1152.0	0.10530	0
57	14.710	21.59	95.55	656.9	0.11370	0
58	13.050	19.31	82.61	527.2	0.08060	1
59	8.618	11.79	54.34	224.5	0.09752	1
60	10.170	14.88	64.55	311.9	0.11340	1

Figura 1: Dados

https://www.kaggle.com/datasets/merishnasuwal/breast-cancer-prediction-dataset.
Nosso objetivo é classificar com base nas colunas: "mean$_r$*adius*","*mean$_t$exture*","*mean$_p$erimeter*","
se uma pessoa possue ou não câncer

4 Função do Erro

```
-- Função para calcular o erro quadrático médio entre as saídas da rede e os valores reais
err :: [Double] -> [Dados] -> Double
err ws dados =
  let n = fromIntegral (length dados)
   in sum [((predict ws d) - y) ** 2 | d@(_, _, _, _, _, y) <- dados] / n
```

Figura 2: Função erro

Com o objetivo de prever e classificar se um paciente possue ou não câncer, utilizaremos uma rede neural com 5 entradas de dados e 1 neurônio e 2 saídas, na qual retornará o diagnóstico positivo ou negativo.

Para isso implementamos a função de ativação sigmoide e sua derivada na qual retornara o diagnóstico:

```haskell
-- Função de ativação sigmóide
sigma :: Double -> Double
sigma x = 1 / (1 + exp (-x))
```

Foi implementada a função predict na qual também está incluso a função sigmoide. O predict recebe os 5 valores enviados pelo e realiza a operação com os pesos calculados pelo gradiente descendente. No final a a função classifica o diagnóstico

```haskell
-- Função para calcular a saída da rede neural para uma entrada de
predict :: [Double] -> Dados -> Double
predict ws (x1, x2, x3, x4, x5, _) =
  let w1 = ws !! 0
      w2 = ws !! 1
      w3 = ws !! 2
      w4 = ws !! 3
      w5 = ws !! 4
      b = ws !! 5
   in sigma (w1 * x1 + w2 * x2 + w3 * x3 + w4 * x4 + w5 * x5 + b)
```

A função err e calcula o erro quadrático médio entre as previsões geradas por um modelo e os valores reais

```haskell
-- Função para calcular o erro quadrático médio entre as saídas da rede e o
err :: [Double] -> [Dados] -> Double
err ws dados =
  let n = fromIntegral (length dados)
   in sum [((predict ws d) - y) ** 2 | d@(_, _, _, _, _, y) <- dados] / n
```

E o grádiente possue 3 funções: A primeira função, gradiente, calcula o gradiente do erro quadrático médio em relação aos pesos e ao viés da rede neural. Isso significa que a função calcula quanto o erro mudaria se cada peso ou viés fosse alterado um pouco. Essas informações são usadas para atualizar os pesos e o viés de forma a reduzir o erro.

A segunda função, atualizaPesos, usa as informações do gradiente calculadas pela função gradiente para atualizar os pesos e o viés da rede neural. Ela faz isso subtraindo uma fração do gradiente dos pesos e do viés atuais. A fração é determinada pela taxa de aprendizado lr, que é um parâmetro que controla a rapidez com que a rede neural aprende.

A terceira função, update, usa a função atualizaPesos para treinar a rede neural por um número fixo de épocas. Uma época é uma passagem completa pelos dados de treinamento. A cada época, a função update chama a função atualizaPesos para atualizar os pesos e o viés da rede neural e, em seguida, repete esse processo até que o número especificado de épocas tenha sido concluído.

1. Derivada da função de ativação sigmoide:

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

2. Derivada parcial em relação a w_1:

$$\frac{\partial E}{\partial w_1} = \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{w}, \mathbf{x}_i) - y_i) \cdot \sigma'(f(\mathbf{w}, \mathbf{x}_i)) \cdot x_{i1}$$

3. Derivada parcial em relação a w_2:

$$\frac{\partial E}{\partial w_2} = \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{w}, \mathbf{x}_i) - y_i) \cdot \sigma'(f(\mathbf{w}, \mathbf{x}_i)) \cdot x_{i2}$$

4. Derivada parcial em relação a w_3:

$$\frac{\partial E}{\partial w_3} = \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{w}, \mathbf{x}_i) - y_i) \cdot \sigma'(f(\mathbf{w}, \mathbf{x}_i)) \cdot x_{i3}$$

5. Derivada parcial em relação a w_4:

$$\frac{\partial E}{\partial w_4} = \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{w}, \mathbf{x}_i) - y_i) \cdot \sigma'(f(\mathbf{w}, \mathbf{x}_i)) \cdot x_{i4}$$

6. Derivada parcial em relação a w_5:

$$\frac{\partial E}{\partial w_5} = \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{w}, \mathbf{x}_i) - y_i) \cdot \sigma'(f(\mathbf{w}, \mathbf{x}_i)) \cdot x_{i5}$$

7. Derivada parcial em relação a b:

$$\frac{\partial E}{\partial b} = \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{w}, \mathbf{x}_i) - y_i) \cdot \sigma'(f(\mathbf{w}, \mathbf{x}_i))$$

- E é o erro quadrático médio,
- $\mathbf{w} = [w_1, w_2, w_3, w_4, w_5]$ é o vetor de pesos,
- $\mathbf{x}_i = [x_{i1}, x_{i2}, x_{i3}, x_{i4}, x_{i5}]$ é o vetor de entrada dos dados de treinamento,
- y_i é o valor de saída real correspondente aos dados de treinamento,
- $f(\mathbf{w}, \mathbf{x}_i)$ é a saída da rede neural para os dados de

5 Sigmóide

```haskell
-- Função de ativação sigmóide
sigma :: Double -> Double
sigma x = 1 / (1 + exp (-x))
```

6 Gradiente

```haskell
-- Função para calcular o gradiente do erro quadrático médio em relação aos pesos e ao viés da rede neural
gradiente :: [Double] -> [Dados] -> [Double]
gradiente ws dados =
  let n = fromIntegral (length dados)
      dedw1 = sum [2 * (predict ws d - y) * dsigma (predict ws d) * x1 | d@(x1, _, _, _, _, y) <- dados] / n
      dedw2 = sum [2 * (predict ws d - y) * dsigma (predict ws d) * x2 | d@(_, x2, _, _, _, y) <- dados] / n
      dedw3 = sum [2 * (predict ws d - y) * dsigma (predict ws d) * x3 | d@(_, _, x3, _, _, y) <- dados] / n
      dedw4 = sum [2 * (predict ws d - y) * dsigma (predict ws d) * x4 | d@(_, _, _, x4, _, y) <- dados] / n
      dedw5 = sum [2 * (predict ws d - y) * dsigma (predict ws d) * x5 | d@(_, _, _, _, x5, y) <- dados] / n
      dedb = sum [2 * (predict ws d - y) * dsigma (predict ws d) | d@(_, _, _, _, _, y) <- dados] / n
   in [dedw1, dedw2, dedw3, dedw4, dedw5, dedb]

atualiza :: Double -> [Double] -> [Double] -> [Double]
atualiza _ [] _ = []
atualiza lr (w : ws) (g : gs) = (w - lr * g) : atualiza lr ws gs

atualizaPesos :: Double -> [Double] -> [Dados] -> [Double]
atualizaPesos lr ws dados =
  let grad = gradiente ws dados
   in atualiza lr ws grad
```

7 Parâmetros

```haskell
main :: IO ()
main = do
  let passos = 1000 --
      lr = 0.01
      pesosinicio = [3.2831774297364507,2.0683092898755584,13.187685985405743,-1.8089015482574466,1.0111544562453125,0.314267956279973]
 -- Valores iniciais dos pesos (w1, w2, w3, w4, w5, b)
      pesosFinais = update passos lr pesosinicio dadosTreino
  putStrLn "\n\nPesos finais:"
  print pesosFinais
  putStrLn "\nErro quadrático médio final:"
  print (err pesosFinais dadosTreino)
  putStrLn "\nPrevisões:"
  let dadosTeste = (5, 15, 24, 235, 360, 100) -- onde fica 999 pode ser qualquer valor que o algoritmo vai prever
  let previsoes = predict pesosFinais dadosTeste
  print previsoes
```

Figura 3: Learning Rate e Vetor Inicial

8 Resultados e Discussões

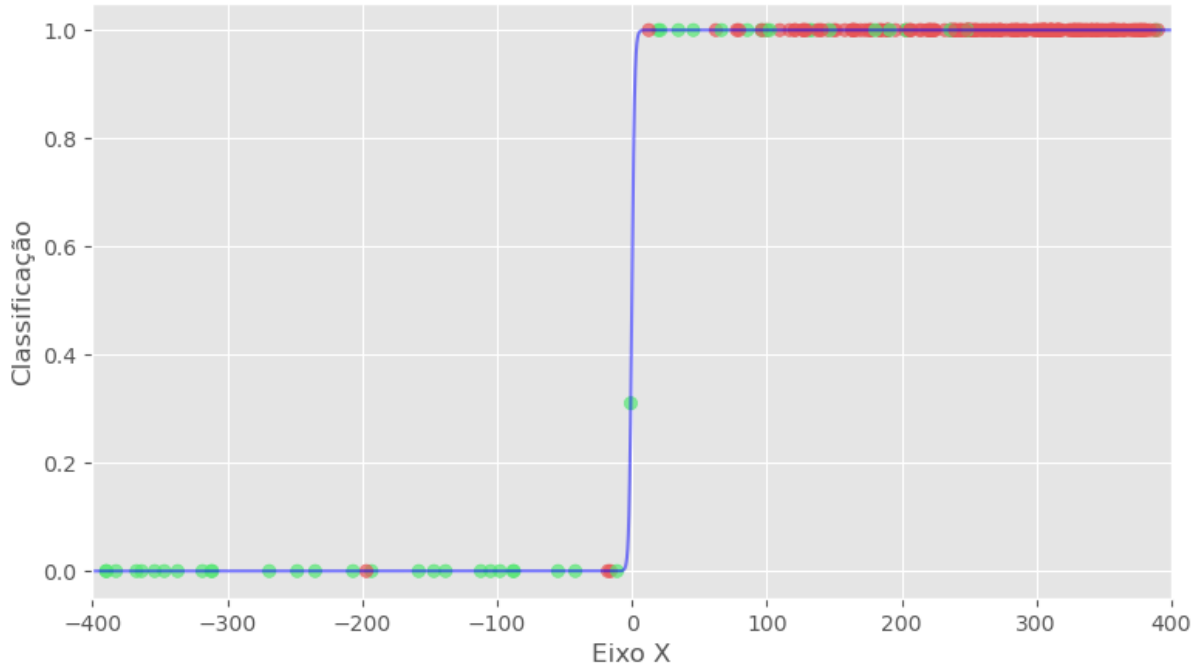
Nosso algoritmo de Redes Neurais verifica através de Pesos e Bias, o diagnóstico de "Câncer"e "Não Câncer". Para isso, quando a previsão do algoritmo de classificação for igual a 1, temos um resultado como "Câncer". Caso seja 0 , o diagnóstico é dado como "Nâo Câncer"

```haskell
main :: IO ()
main = do
  let passos = 1000 --
      lr = 0.01
      pesosinicio = [3.2831774297364507
,2.0683092898755584,13.187685985405743,-1.8089015482574466,1.0111544562453125,0.314267956279973]
 -- Valores iniciais dos pesos (w1, w2, w3, w4, w5, b)
      pesosFinais = update passos lr pesosinicio dadosTreino
  putStrLn "\n\nPesos finais:"
  print pesosFinais
  putStrLn "\nErro quadrático médio final:"
  print (err pesosFinais dadosTreino)
  putStrLn "\nPrevisões:"
  let dadosTeste = (5, 15, 24, 235, 360, 100) -- onde fica 999 pode ser qualquer valor que o
algoritmo vai prever
  let previsoes = predict pesosFinais dadosTeste
  print previsoes
```

```
GHCi, version 9.0.2: https://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/runner/Neurais/.ghci
[1 of 1] Compiling Main             ( Main.hs, interpreted )
Ok, one module loaded.
ghci>

Pesos finais:
[6.5144357836441955,6.151439911806097,34.514309903812155,174.17636939371772,1.030726927
2263158,0.5045847314617005]

Erro quadrático médio final:
0.4866666666666667

Previsões:
1.0
ghci>
```

```haskell
main :: IO ()
main = do
  let passos = 100 --
      lr = 0.01
      pesosinicio =
[3.2831774297364507,2.0683092898755584,13.187685985405743,-1.8089015482574466,1.0111544562453125
,0.314267956279973] -- Valores iniciais dos pesos (w1, w2, w3, w4, w5, b)
      pesosFinais = update passos lr pesosinicio dadosTreino
  putStrLn "\n\nPesos finais:"
  print pesosFinais
  putStrLn "\nErro quadrático médio final:"
  print (err pesosFinais dadosTreino)
  putStrLn "\nPrevisões:"
  let dadosTeste = (15.5,4.1,87.8, 95.9,198.8, 999) -- onde fica 999 pode ser qualquer valor
que o algoritmo vai prever
  let previsoes = predict pesosFinais dadosTeste
  print previsoes
```

```
Pesos finais:
[8.088376393281118,10.805808684062503,41.91671652265235,-16.738740680867462,1.067400581
1537148,0.9974211692399431]

Erro quadrático médio final:
0.5133333333333333

Previsões:
0.0
```



A acurácia pode ser vista no Notebook, na qual é 0.96 sobre os dados de testes (dados que não foram postos para a rede treinar)

9 Github

AnabelleSouza/Redes Neurais • gabrielluizone/Hasketafell