Faculdade de Tecnologia da Baixada Santista

Curso Superior de Tecnologia em Ciência de Dados

Redes Neurais

Gabriela Duarte Maciel

Santos - SP
19 de maio de 2023.

Descrição

Neste projeto, uma rede neural é implementada com o objetivo de classificar números do conjunto inteiro como pares ou ímpares. Lembrando que é possível fazer a classificações de números negativos, pois o resultado de mod será zero ou um, levando em consideração o um negativo, pois caso um número seja negativo, ainda sim é possível classifica-lo. Ademais, o intuito desse modelo é explorar a capacidade da rede neural em aprender padrões e tomar decisões com base nos dados de entrada.

Para construir o gráfico de dados utilizado na rede neural, foram considerados os valores no intervalo de 0 a 1000 do conjunto dos números naturais. Cada número foi associado a uma classe, indicando se ele é par ou ímpar. A partir dessa associação, a rede neural é treinada para aprender a classificar corretamente novos números que lhe forem apresentados.

Classificação

A rede neural implementada tem o objetivo de classificar se um número do conjunto natural é par ou ímpar. A rede foi treinada utilizando os valores do conjunto natural de 0 a 1000 como dados de entrada. A classificação realizada pela rede neural é baseada em dois pesos (w1 e w2) e dois viés (b1 e b2) que foram ajustados durante o treinamento. A rede utiliza uma função de ativação, no caso a função sigmoide, para transformar a combinação linear dos pesos e entradas em uma probabilidade, que representa a chance de o número ser par ou ímpar.

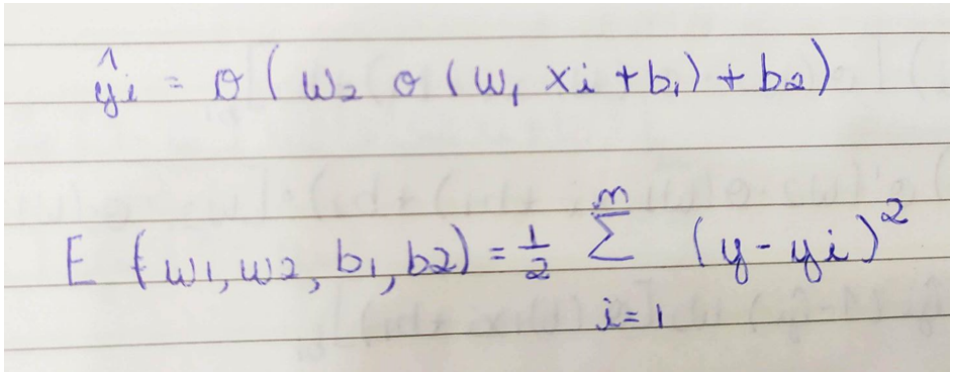
Ao fornecer um número como entrada para a rede neural, ela realiza uma série de cálculos para obter a probabilidade de ser par ou ímpar. Em seguida, é aplicado um limite de decisão de 0.5, para determinar a classificação final. Se a probabilidade for maior que 0.5 o número é classificado como par; caso contrário, é classificado como ímpar.

Função do erro

A função é responsável por ajustar os valores dos parâmetros (xt e yt), atualizando-os na direção do gradiente negativo, até que a diferença entre os valores atualizados e os valores anteriores, representada pelo erro, seja menor que um limite pré-definido que é a tolerância.

A função de erro é calculada usando a fórmula da distância entre os pontos (xnovo, ynovo) e (xt, yt) no espaço de parâmetros. Essa fórmula é utilizada para medir a diferença entre os valores antigos e atualizados dos parâmetros, representando a magnitude da atualização realizada em cada iteração do algoritmo.

Ao comparar o valor do erro com o limite de tolerância, a função descent decide se deve continuar o processo de otimização ou se já atingiu um ponto de convergência próximo o suficiente. Caso o erro seja menor que a tolerância, significa que os parâmetros foram otimizados o suficiente e a função retorna os valores atualizados dos parâmetros, juntamente com o número de iterações realizadas.



Função do erro

```haskell
descent :: Double -> Double -> Double -> Double -> Int -> (Double, Double, Int)
descent lr xt yt err i
  | err < tol = (xt, yt, i)
  | otherwise =
      let dfdx = fst (grad xt yt)
          dfdy = snd (grad xt yt)
          xnovo = xt - lr * dfdx
          ynovo = yt - lr * dfdy
          errnovo = sqrt ((xnovo - xt) ** 2 + (ynovo - yt) ** 2)
       in descent lr xnovo ynovo errnovo (i + 1)
```

Função do erro

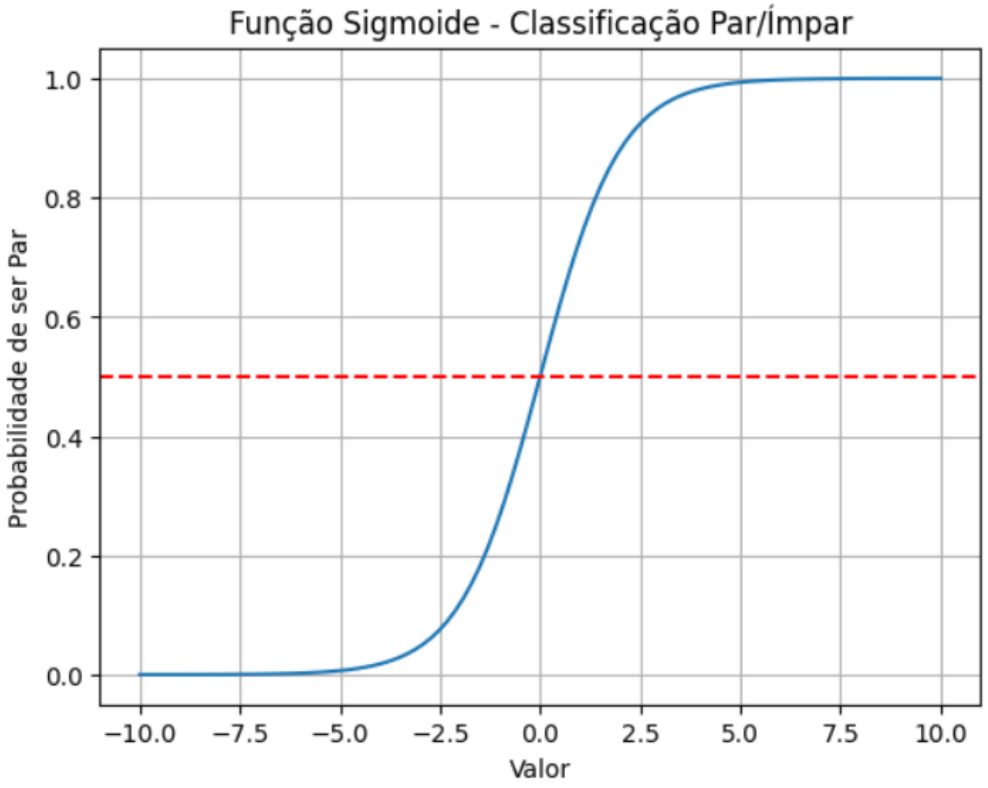
Parametros

Os parametros utilizados nesta rede neural foram learning rate, com o valor de (10 ** (-6)) e vetor inicial, com os valores de ([18.43878139117114, 18.2805394706888,-9.372209396659654,-8.398746300630092])

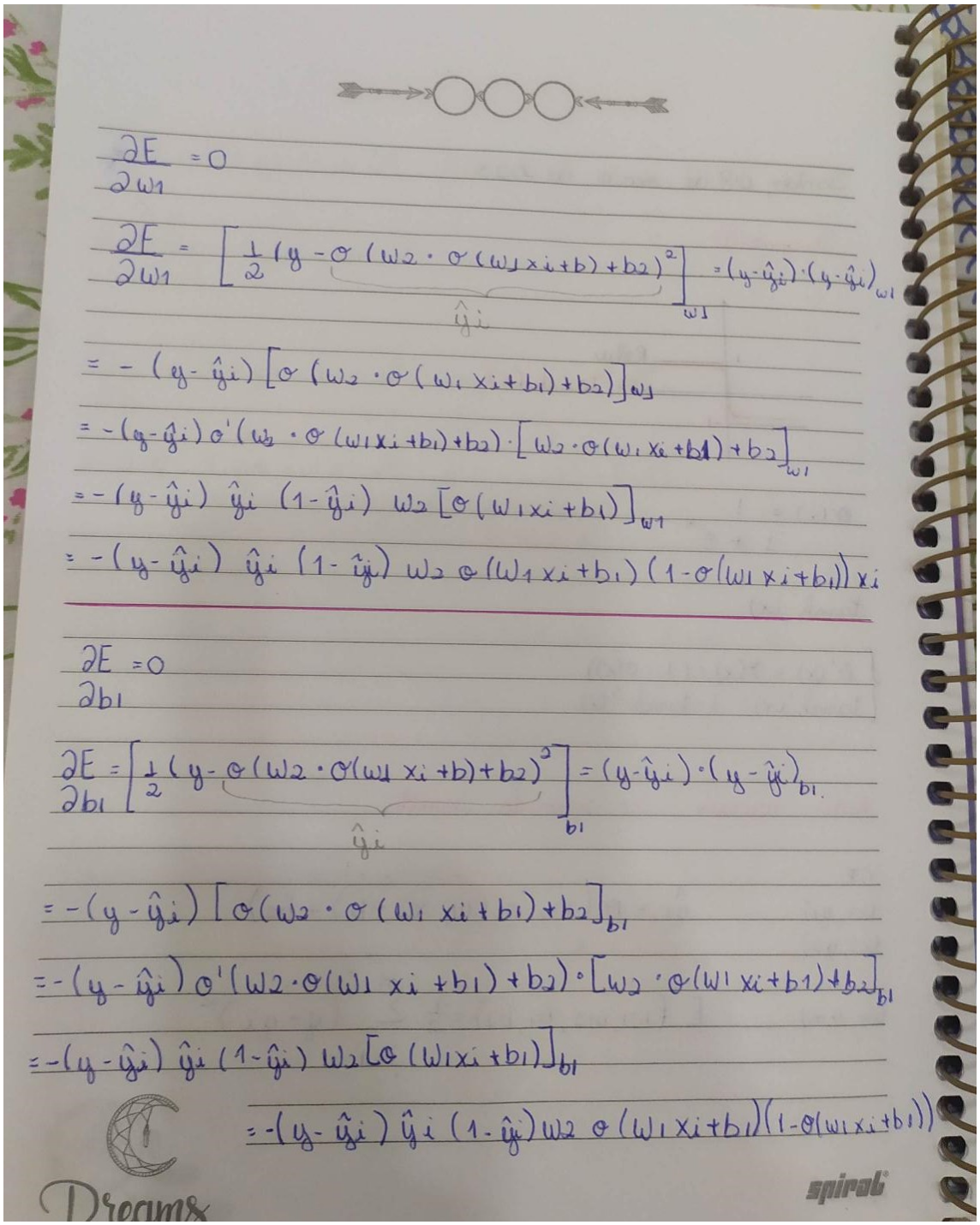
Sigmoide e Gradiente

A função sigmoide é utilizada para introduzir não-linearidades nas camadas da rede neural, permitindo que o modelo aprenda a representar relações complexas entre os dados de entrada e a saída desejada. A função sigmoide retorna valores entre 0(impar) e 1(par), segundo a classificação realizada nesta rede. Nesse caso se a predição for maior ou igual a 0.5, a rede já considera o número como par, se for menor ele sera um número impar.

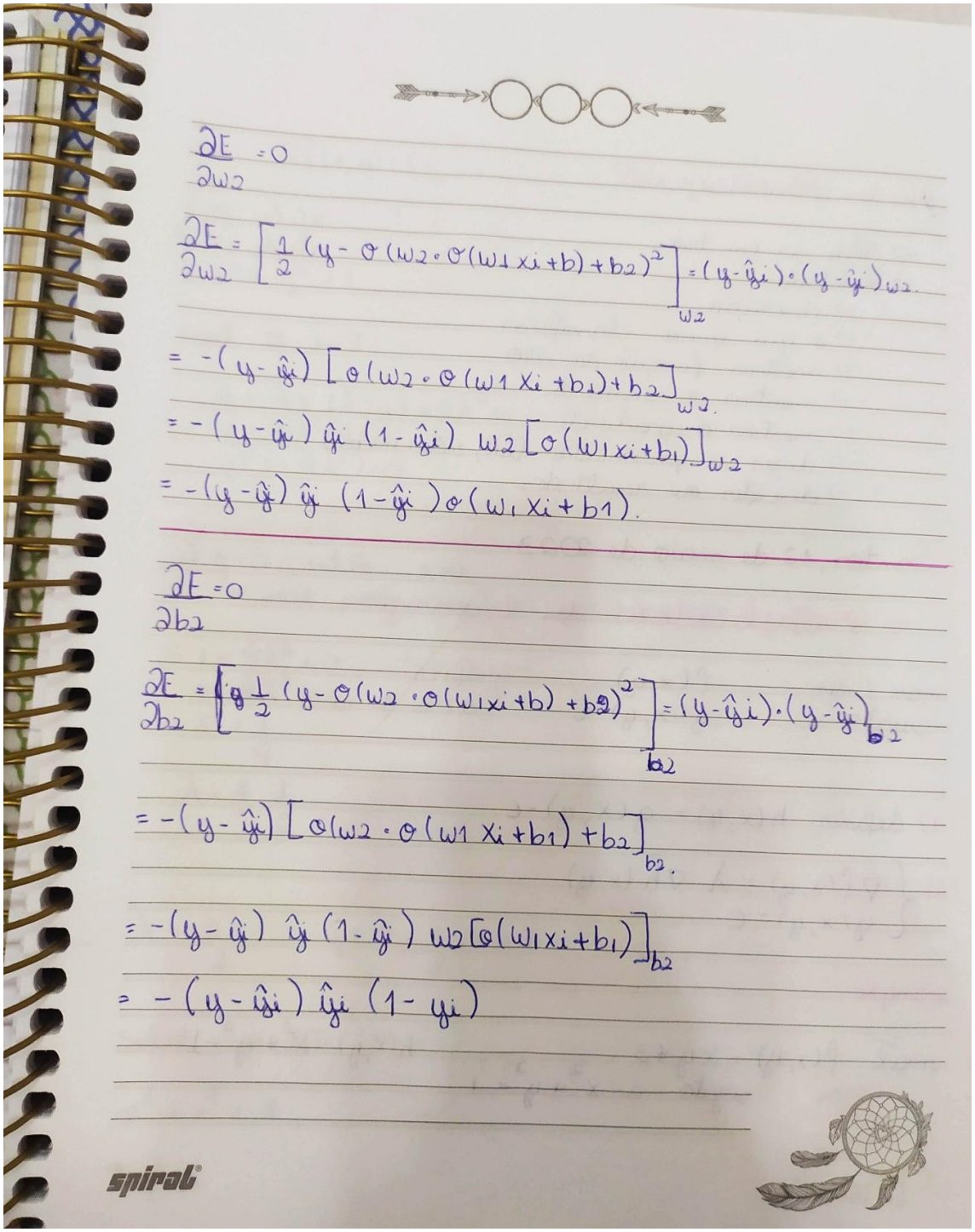
Além da função sigmoide, outros conceitos importantes estão envolvidos na implementação dessa rede neural, como o cálculo do gradiente, que é utilizado no treinamento da rede por meio do algoritmo de gradiente descendente. Esse algoritmo ajusta os pesos e os viéses das conexões entre as unidades da rede, buscando minimizar o erro entre as previsões da rede e as saídas desejadas. Para calcular gradiente, foi utilizado as derivadas de cada ponto da função de erro, (w1,b1,w2,b2).



Função sigmoide



Derivadas dos pontos



Continuação das derivadas dos pontos

Código de Redes Neurais em Haskell

Este é um algoritmo desenvolvido em Haskell, nele é descrito desde seu início, os calculadas das derivadas, logo após é escrito a função de erro, é passada a tolerância, cálulo da predição, função sigmoide e uma função com os valores para serem classificados. No fim é mostrado no console o resultado de um número aleatório, calculado a sua predição e sua classificação. E também uma função onde testa um conjunto de dados através de um laço de repetição-for. Para melhor vizualização foi plotado em um gráfico de dispersão, juntamente com uma linha bem na divisão da predição dos dados.

```haskell
module Main where
import Control.Monad (forM_)

--Derivadas
grad :: Double -> Double -> (Double, Double)
grad w b =
  let dedw = 2 * ((5 - w - b) * (-1) + (7 - 2 * w - b) * (-2) + (9 - 3 * w - b) * (-3) + (10 - 4 * w - b)
* (-4) + (12 - 5 * w - b) * (-5) + (15 - 6 * w - b) * (-6))
      dedb = -2 * ((5 - w - b) + (7 - 2 * w - b) + (9 - 3 * w - b) + (10 - 4 * w - b) + (12 - 5 * w - b) +
(15 - 6 * w - b))
   in (dedw, dedb)

-- Função do Erro
descent :: Double -> Double -> Double -> Double -> Int -> (Double, Double, Int)
descent lr xt yt err i
  | err < tol = (xt, yt, i)
  | otherwise =
      let dfdx = fst (grad xt yt)
          dfdy = snd (grad xt yt)
          xnovo = xt - lr * dfdx
          ynovo = yt - lr * dfdy
          errnovo = sqrt ((xnovo - xt) ** 2 + (ynovo - yt) ** 2)
       in descent lr xnovo ynovo errnovo (i + 1)

predict :: Double -> [Double] -> Double
predict xi xs =
  let w1 = xs !! 0
      w2 = xs !! 1
```

Código em haskell.

```haskell
      w2 = xs !! 1
      b1 = xs !! 2
      b2 = xs !! 3
  in sigma (w2 * sigma (w1 * xi + b1) + b2)


-- Função sigmoide
sigma :: Double -> Double
sigma x = 1 / (1 + exp (-x))

func :: Int -> Double
func n
  | n >= 500 = 0.5
  | otherwise = 0.5

neural :: [(Double, Double)] -> [Double] -> [Double]
neural ts xs =
  let w1 = xs !! 0
      w2 = xs !! 1
      b1 = xs !! 2
      b2 = xs !! 3
      f xi = sigma (w2 * sigma (w1 * xi + b1) + b2)
      dedw1 =
        sum
          [ -(yi - f xi)
              * f xi
              * (1 - f xi)
```

Continuação do código em haskell.

```haskell
              * w2
              * sigma (w1 * xi + b1)
              * (1 - sigma (w1 * xi + b1))
              * xi
          | (xi, yi) <- ts
          ]
      dedw2 =
        sum
          [ -(yi - f xi)
              * f xi
              * (1 - f xi)
              * sigma (w1 * xi + b1)
          | (xi, yi) <- ts
          ]
      dedb1 =
        sum
          [ -(yi - f xi)
              * f xi
              * (1 - f xi)
              * w2
              * sigma (w1 * xi + b1)
              * (1 - sigma (w1 * xi + b1))
          | (xi, yi) <- ts
          ]
      dedb2 =
        sum
          [ -(yi - f xi)
              * f xi
```

Continuação do código em haskell.

```
                * f xi
                * (1 - f xi)
              | (xi, yi) <- ts
            ]
    in [dedw1, dedw2, dedb1, dedb2]

descentV :: ([Double] -> [Double]) -> Double -> Int -> Double -> [Double] -> ([Double], Int)
descentV grad lr i err xts
  | err < tol = (xts, i)
  | otherwise =
      let dfdxs = grad xts
          xsnovo = [xt - lr * grad | (xt, grad) <- zip xts dfdxs]
          errnovo = sum [(xnovo - xt) ** 2 | (xnovo, xt) <- zip xsnovo xts] in descentV grad lr (i + 1)
errnovo xsnovo

--Tolerância
tol :: Double
tol = 10 ** (-6)

parOuImpar :: Int -> String
parOuImpar n =
-- mod/2 == 0 logo 0 == 1, logo impar
    if n `mod` 2 == 0 then "par" else "impar"
main = do
    putStrLn "Digite um valor: "
    input <- getLine
    let value = read input :: Double
        tc = [(fromIntegral n / 1000, func n) | n <- [1 .. 1000]]
```

Continuação do código em haskell.

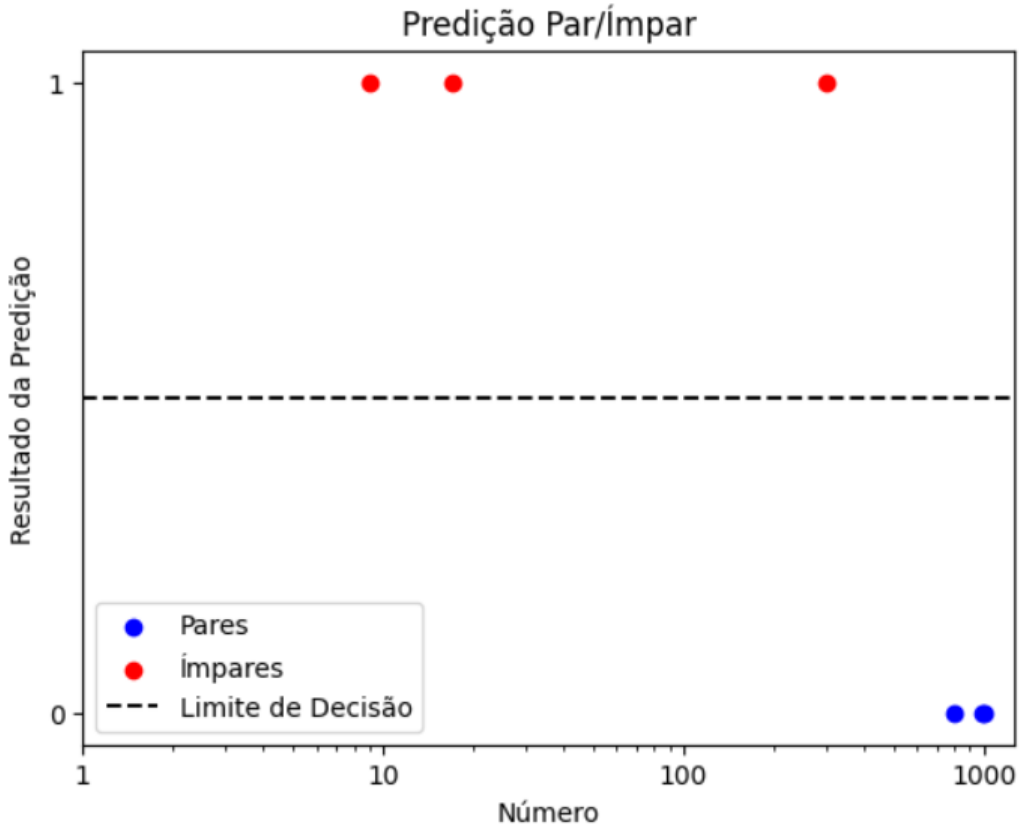
```
        tc = [(fromIntegral n / 1000, func n) | n <- [1 .. 1000]]
        --Valores Iniciais
        p = descentV (neural tc) 0.01 0 9999 [18.438781391117114, 18.28050394706888, -9.372209396659654,
-8.398746300630092]
        fp = fst p
    putStrLn "Descent"
    print p
    putStrLn $ "Predição para " ++ input
    putStrLn $ if odd (round value) then "impar" else "par"
    print $ predict (value / 1000) fp

    --Função para imprimir os valores e suas classificações(Impar-Par)
    let numbers = [988, 299, 800, 9, 17, 1000]
    forM_ numbers $ \num -> do
      putStrLn (show num)
      if predict (fromIntegral num / 1000) fp >= 0.5
        then putStrLn ("O número " ++ show num ++ " é par")
        else putStrLn ("O número " ++ show num ++ " é ímpar")
```

Continuação do código em haskell.

```
Digite um valor:
90
Descent
([10.720757575003384,15.36017365021452,-15.19387052756457,-1.6344205394757443e-2],1114)
Predição para 90
par
0.49591657900031805
988
O número 988 é par
299
O número 299 é ímpar
800
O número 800 é par
9
O número 9 é ímpar
17
O número 17 é ímpar
1000
O número 1000 é par
```

Resultado no console.



Classificação do conjunto de números do laço-for do console.

Conclusão

Portanto, o algoritmo apresentado utiliza a técnica de rede neural para realizar a classificação de números como pares ou ímpares. Através do treinamento realizado com a função, o modelo aprende os pesos e bias que são utilizados pela função preditiva. E também com o gradiente descendente, o modelo realiza ajustes nesses valores para melhorar a precisão de sua classificação.Ao analisar o comportamento desta rede neural e os gráficos que o algoritmo gera, podemos considerar que, se um número retornar uma predição menor que 0.5 ele é classificado como impar e caso a predição seja maior ou igual a 0.5 ele é considerado como par.

Link do Github:

https://github.com/gabiduarte435/RedesNeurais$_{G}abrielaDuarte$