Lista 1: Ajustando uma RNA 'no braço'

César A. Galvão - 190011572

Para essa lista, é considerada a seguinte arquitetura de uma rede neural *feed-forward*:

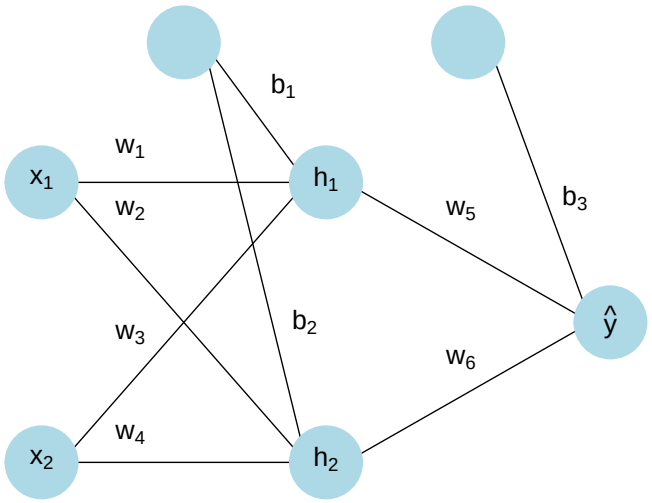

Figura 1: Arquitetura da rede neural artificial. Adotamos função de ativação sigmoide e linear nas camadas escondidas e de saída, respectivamente.

Questão 1

Item a

Crie uma função computacional para calcular o valor previso da variável resposta $\hat{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ em função de $\boldsymbol{x}$ e $\boldsymbol{\theta}$. Use a função para calcular $\hat{y}$ para $\boldsymbol{\theta} = (0.1, \ldots, 0.1)$ e $\boldsymbol{x} = (1, 1)$.

Implementa-se de forma matricial. A função não é adaptativa ao tamanho da rede e exige que o usuário forneça uma lista $\boldsymbol{\theta}$ com os seguintes elementos:

1. $W^{(1)}$ - matriz 2×2 de pesos da camada de entrada $\begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix}$. Cada linha deve representar os pesos de cada neurônio para o neurônio subsequente, i.e. w_{ij} representa o peso do neurônio de entrada i para o próximo neurônio j.
2. $\boldsymbol{b}^{(1)}$ - vetor de viés da camada de entrada $\begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$.
3. $W^{(2)}$ - matriz 2×1 de pesos da camada de saída $\begin{pmatrix} w_5 \\ w_6 \end{pmatrix}$.
4. $\boldsymbol{b}^{(2)}$ - vetor elemento de viés da camada de saída $\begin{pmatrix} b_3 \end{pmatrix}$.

Como função de ativação foi escolhida a função sigmóide denotada por $\phi = \frac{1}{1+e^{-x}}$. A função de previsão é dada por:

$$\hat{y} = W^{(2)\top} \phi(W^{(1)\top} \boldsymbol{x} + \boldsymbol{b}^{(1)}) + \boldsymbol{b}^{(2)}$$

ou

$$\begin{aligned} \boldsymbol{a} &= W^{(1)\top} \boldsymbol{x} + \boldsymbol{b^{(1)}} \\ \boldsymbol{h} &= \phi(\boldsymbol{a}) \\ \hat{y} &= W^{(2)\top} \boldsymbol{h} + b^{(2)} \end{aligned}$$

```
# Função de ativação
phi <- function(x) {1/(1 + exp(-x))}

# Função de previsão
ffwd <- function(x, theta) {

  if(any(dim(theta[[1]]) != c(2, 2))){
    stop("Feed Forward: O primeiro elemento de theta deve ser uma
    ↪  matriz de pesos 2x2 para a aplicação nos dados")
  } else if(length(theta[[2]]) != 2){
    stop("Feed Forward: O segundo elemento de theta deve ser um
    ↪  vetor viés de tamanho 2 para somar aos dados")
  } else if(any(dim(theta[[3]]) != c(2, 1))){
    stop("Feed Forward: O terceiro elemento de theta deve ser uma
    ↪  matriz de pesos 2x1 para aplicação na única camada h")
  } else if(length(theta[[4]]) != 1){
    stop("Feed Forward: O quarto elemento de theta deve ser um
    ↪  vetor viés de tamanho 1 para soma na única camada h")
  } else if(!is.vector(x) & length(x) != 2){
    stop("Feed Forward: x deve ser um vetor de tamanho 2")
  }
```

```
  x <- as.vector(x)

  W1 <- theta[[1]]
  b1 <- theta[[2]]
  W2 <- theta[[3]]
  b2 <- theta[[4]]

  a <- (t(W1)%*%x)+b1
  h <- phi(a)
  yhat <- (t(W2)%*%h)+b2

  return( # separacao dos elementos de saída para usar no
  ↪  backpropagation
    list(yhat = as.double(yhat),
         hidden = h,
         pre_activation = a)
  )
}
```

Agora vamos calcular $\hat{y}$ para $\boldsymbol{\theta} = (0.1, \ldots, 0.1)$ e $\boldsymbol{x} = (1, 1)$.

```
x <- c(1,1)

theta <- list(
  M1 = matrix(c(0.1), nrow = 2, ncol = 2),
  b12 = c(0.1, 0.1),
  M2 = matrix(c(0.1), nrow = 2, ncol = 1),
  b3 = c(0.1)
)

ffwd(x, theta)
```

Obtemos $\hat{y} = 0.2148885$.

Item b

Crie uma rotina computacional para calcular a função de custo $J(\theta)$. Em seguida, divida o conjunto de dados observados de modo que as **primeiras** 80.000 amostras componham o conjunto de **treinamento**, as próximas 10.000 o de **validação**, e as **últimas** 10.000 o de **teste**. Qual é o custo da rede no **conjunto de teste** quando $\boldsymbol{\theta} = (0.1, ..., 0.1)$?

Primeiro são gerados os dados conforme as instruções da lista:

```
### Gerando dados "observados"
set.seed(1.2024)
m.obs <- 100000
dados <- tibble(x1.obs=runif(m.obs, -3, 3),
                x2.obs=runif(m.obs, -3, 3)) %>%
         mutate(mu=abs(x1.obs^3 - 30*sin(x2.obs) + 10),
                y=rnorm(m.obs, mean=mu, sd=1))
```

Depois, implementamos a função de custo $J(\theta)$, que é dada por

$$\begin{aligned} J(\boldsymbol{\theta}) =& \frac{1}{m}\sum_{i=1}^{m} L\left(f(x_{1i}, x_{2i}; \boldsymbol{\theta}), y_i\right) \\ =& \frac{1}{m}\sum_{i=1}^{m} \left(f(x_{1i}, x_{2i}; \boldsymbol{\theta}) - y_i\right)^2 \\ =& \frac{1}{m}\sum_{i=1}^{m} \left(\hat{y}_i - y_i\right)^2, \end{aligned}$$

onde m é o número de observações.

```
J.Loss <- function(dados, theta, target){

  if(!is.data.frame(dados) | !is_tibble(dados)){
    stop("Loss: Dados deve ser uma dataframe ou tibble.")
  } else if (dim(dados)[2] != 2){
    stop("Loss: Matriz de dados deve ter 2 colunas.")
  } else if (!is.list(theta)){
    stop("Loss: Theta deve ser uma lista com pesos e viéses.")
  } else if (!is.numeric(target)){
    stop("Loss: Target deve ser um vetor numérico.")
  }

# aloca memoria para o tamanho dos dados
  yhat <- double(nrow(dados))

# aloca memoria para os valores pre ativação dos 'a'
  a <- matrix(nrow = nrow(dados), ncol = 2, dimnames = list(NULL,
↪  c("a1", "a2")))


# transpoe os dados para termos acesso aos vetores de x que serao
↪  passados para a primeira camada da rede
```

```
  tdados <- t(as.matrix(dados))

  for(i in 1:ncol(tdados)){
    yhat[i] <- ffwd(tdados[,i], theta)$yhat
    a[i,] <- ffwd(tdados[,i], theta)$pre_activation
  }

  return(
    list(
      loss = mean((target - yhat)^2), #média ja entrega soma/m
      yhat = yhat,
      pre_activation = a,
      hidden = matrix(c(phi(a[,1]), phi(a[,2])), ncol = 2)
    )
  )
}
```

Em seguida, separamos o nosso conjunto de dados em treinamento, validação e teste.

```
dados_train <- dados[1:80000,]
dados_valid <- dados[80001:90000,]
dados_test <- dados[90001:nrow(dados),]
```

Finalmente, executamos a função de *feed-forward* nos dados gerados para calcularmos $\hat{\boldsymbol{y}}$ e em seguida calculamos o custo da rede com $\boldsymbol{\theta} = (0.1, \ldots, 0.1)$.

```
# transformamos os dados em matriz
x_test <- dados_test %>%
  select(x1.obs, x2.obs)

# separamos o target
y_test <- dados_test %>%
  pull(y)

# theta já foi gerado e será reaproveitado
J.Loss(x_test, theta, y_test)$loss
```

Obtemos um custo de 663.1286383.

Item c

Use a regra da cadeia para encontrar expressões algébricas para o vetor gradiente

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \left(\frac{\partial J}{\partial w_1}, \ldots, \frac{\partial J}{\partial b_3} \right).$$

Desejamos $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ tal que

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} (y - \hat{y}_i)^2 \right] \\
&= \frac{2}{m} \sum_{i=1}^{m} (y - \hat{y}_i) \nabla_{\boldsymbol{\theta}} (y - \hat{y}_i), \quad \text{pois } \hat{y}_i = f(x_{1i}, x_{2i}; \boldsymbol{\theta}) \\
&= \frac{2}{m} \sum_{i=1}^{m} (y - \hat{y}_i)(-1) \nabla_{\boldsymbol{\theta}} \hat{y}_i \qquad (1)
\end{aligned}$$

Resolvemos o gradiente, considerando $\phi(x) = \sigma(x) = \frac{1}{1+e^{-x}}$,

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \hat{y}_i &= \nabla_{\boldsymbol{\theta}} \left[h_{1i} w_5 + h_{2i} w_6 + b_3 \right] \\
&= \nabla_{\boldsymbol{\theta}} \left[\phi(a_{1i}) w_5 + \phi(a_{2i}) w_6 + b_3 \right] \\
&= \nabla_{\boldsymbol{\theta}} \left[\phi(x_{1i} w_1 + x_{2i} w_3 + b_1) w_5 + \phi(x_{1i} w_2 + x_{2i} w_4 + b_2) w_6 + b_3 \right].
\end{aligned}$$

É imediato que

$$\frac{\partial \hat{y}_i}{\partial b_3} = 1, \quad \frac{\partial \hat{y}_i}{\partial w_5} = h_{1i}, \quad \frac{\partial \hat{y}_i}{\partial w_6} = h_{2i}.$$

Para $b_j, j \in \{1, 2\}$ resolvemos de forma análoga:

$$\begin{aligned}
\frac{\partial \hat{y}_i}{\partial b_1} &= w_5 \frac{\partial h_1}{\partial b_1} = w_5 \frac{\partial}{\partial b_1} \phi(x_{1i} w_1 + x_{2i} w_3 + b_1) \\
&= w_5 \frac{\partial}{\partial b_1} \left(\frac{1}{1 + e^{-(x_{1i} w_1 + x_{2i} w_3 + b_1)}} \right) \\
&= w_5 \frac{(-1) \cdot \frac{\partial}{\partial b_1} \left(1 + e^{-(x_{1i} w_1 + x_{2i} w_3 + b_1)} \right)}{\left(1 + e^{-(x_{1i} w_1 + x_{2i} w_3 + b_1)} \right)^2}
\end{aligned}$$

e

$$\frac{\partial}{\partial b_1}\left(1 + e^{x_{1i}w_1 + x_{2i}w_3 + b_1}\right) = 1 \cdot e^{x_{1i}w_1 + x_{2i}w_3 + b_1}$$
$$= e^{x_{1i}w_1 + x_{2i}w_3 + b_1} = e^{a_1}.$$

Portanto,

$$\frac{\partial \hat{y}_i}{\partial b_1} = w_5 \frac{-e^{a_1}}{(1 + e^{a_1})^2}, \quad \text{e} \quad \frac{\partial \hat{y}_i}{\partial b_2} = w_6 \frac{-e^{a_2}}{(1 + e^{a_2})^2}.$$

Para $w_j, j \in \{1, 2, 3, 4\}$,

$$\begin{aligned}
\frac{\partial \hat{y}_i}{\partial w_1} &= w_5 \frac{\partial h_1}{\partial w_1} = w_5 \frac{-x_{1i}\, e^{a_1}}{(1 + e^{a_1})^2} \\
\frac{\partial \hat{y}_i}{\partial w_2} &= w_6 \frac{-x_{1i}\, e^{a_2}}{(1 + e^{a_2})^2} \\
\frac{\partial \hat{y}_i}{\partial w_3} &= w_5 \frac{-x_{2i}\, e^{a_1}}{(1 + e^{a_1})^2} \\
\frac{\partial \hat{y}_i}{\partial w_4} &= w_6 \frac{-x_{2i}\, e^{a_2}}{(1 + e^{a_2})^2}.
\end{aligned}$$

Finalmente, substituimos os as componentes na equação (1) e explicitamos cada componente do gradiente[1]:

[1] Essa notação poderia ser escrita de forma mais elegante e sucinta. No entanto, dessa forma facilita a comparação entre a expressão analítica e a implementação computacional pelo leitor.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial J}{\partial w_1} \\ \frac{\partial J}{\partial w_2} \\ \frac{\partial J}{\partial w_3} \\ \frac{\partial J}{\partial w_4} \\ \frac{\partial J}{\partial w_5} \\ \frac{\partial J}{\partial w_6} \\ \frac{\partial J}{\partial b_1} \\ \frac{\partial J}{\partial b_2} \\ \frac{\partial J}{\partial b_3} \end{pmatrix} = \begin{pmatrix} \frac{2}{m}\sum_{i=1}^{m}(y-\hat{y}_i)\, w_5\, x_{1i}\, \frac{e^{a_{1i}}}{(1+e^{a_{1i}})^2} \\ \frac{2}{m}\sum_{i=1}^{m}(y-\hat{y}_i)\, w_6\, x_{1i}\, \frac{e^{a_{2i}}}{(1+e^{a_{2i}})^2} \\ \frac{2}{m}\sum_{i=1}^{m}(y-\hat{y}_i)\, w_5\, x_{2i}\, \frac{e^{a_{1i}}}{(1+e^{a_{1i}})^2} \\ \frac{2}{m}\sum_{i=1}^{m}(y-\hat{y}_i)\, w_6\, x_{2i}\, \frac{e^{a_{2i}}}{(1+e^{a_{2i}})^2} \\ \frac{2}{m}\sum_{i=1}^{m}(y-\hat{y}_i)\,(-h_{1i}) \\ \frac{2}{m}\sum_{i=1}^{m}(y-\hat{y}_i)\,(-h_{2i}) \\ \frac{2}{m}\sum_{i=1}^{m}(y-\hat{y}_i)\, w_5\, \frac{e^{a_{1i}}}{(1+e^{a_{1i}})^2} \\ \frac{2}{m}\sum_{i=1}^{m}(y-\hat{y}_i)\, w_6\, \frac{e^{a_{2i}}}{(1+e^{a_{2i}})^2} \\ \frac{2}{m}\sum_{i=1}^{m}(y-\hat{y}_i)(-1) \end{pmatrix} \tag{2}$$

Item d

Crie uma função computacional que receba como entrada o vetor θ, uma matriz design (x) e as respectivas observações (y) e forneça, como saída, o gradiente definido no item c). Apresente o resultado da função aplicada sobre o banco de treinamento, quando $\theta = (0.1, \ldots, 0.1)$. Atenção: implemente o algoritmo *back-propagation* (Algoritmo 6.4 do livro Deep Learning) para evitar realizar a mesma operação múltiplas vezes.

Primeiramente, implementamos o gradiente encontrado em (2):

```
gradiente <- function(dados, theta, target){

  if(!is.list(theta)){
    stop("Gradiente: Theta deve ser uma lista com pesos e
    ↪  viéses.")
```

```
} else if(!is.data.frame(dados) | !is_tibble(dados)){
  stop("Gradiente: Dados deve ser uma dataframe ou tibble.")
} else if(!is.numeric(target)){
  stop("Gradiente: Target deve ser um vetor numérico.")
}

loss_results <- J.Loss(dados, theta, target)

# vetor de yhat
yhat <- loss_results$yhat

# pre_activation da J.Loss retorna o vetor de 'a'
a <- loss_results$pre_activation
#transformacao em nos neuronios pre-ativacao
a<- exp(a)/(1+ exp(a))^2

# diferenca entre (target) y e yhat
diff <- (target-yhat)

# dados como uma matriz
dados <- as.matrix(dados)

g_w1 <- 2*mean(diff*theta[[3]][1]*dados[,1]*a[,1]) #x1, a1, w5
g_w2 <- 2*mean(diff*theta[[3]][2]*dados[,1]*a[,2]) #x1, a2, w6
g_w3 <- 2*mean(diff*theta[[3]][1]*dados[,2]*a[,1]) #x2, a1, w5
g_w4 <- 2*mean(diff*theta[[3]][2]*dados[,2]*a[,2]) #x2, a2, w6
g_w5 <- 2*mean(diff*(-loss_results$hidden[,1])) #h1
g_w6 <- 2*mean(diff*(-loss_results$hidden[,2])) #h2
g_b1 <- 2*mean(diff*theta[[3]][1]*a[,1]) #a1, w5
g_b2 <- 2*mean(diff*theta[[3]][2]*a[,2]) #a2, w6
g_b3 <- 2*mean(diff*(-1))

grad <- c(g_w1, g_w2, g_w3, g_w4, g_w5, g_w6, g_b1, g_b2, g_b3)
names(grad) <- c("w1", "w2", "w3", "w4", "w5", "w6", "b1",
↪  "b2", "b3")



return(
  list(grad = grad,
     loss_results = list(
       loss = loss_results$loss,
       yhat = loss_results$yhat,
       hidden = loss_results$hidden,
       pre_activation = loss_results$pre_activation
```

```
        )
      )
    )
}
```

É apresentado a seguir o resultado da função aplicada sobre o banco de treinamento, quando $\theta = (0.1, \ldots, 0.1)$:

```
gradiente(dados_train[,c(1,2)], theta, dados_train$y)$grad %>%
  as_tibble() %>%
  summarise(Componente = c("w1", "w2", "w3", "w4", "w5", "w6",
  ↪  "b1", "b2", "b3"),
         Derivada = .$value) %>%
  knitr::kable()
```

Componente	Derivada
w1	0.1767887
w2	0.1767887
w3	-0.6458047
w4	-0.6458047
w5	-22.3246918
w6	-22.3246918
b1	1.0732662
b2	1.0732662
b3	-43.4113383

Item e

Aplique o método do gradiente para encontrar os parâmetros que minimizam a função de custo no **banco de validação**. Inicie o algoritmo no ponto $\theta = (0, \ldots, 0)$, use taxa de aprendizagem $\epsilon = 0.1$ e rode o algoritmo por 100 iterações. Reporte o menor custo obtido e indique em qual iteração ele foi observado. Apresente também o vetor de pesos estimado e comente o resultado.

Para a *backpropagation* foi implementada a função a seguir que funciona em um de dois modos: por número de iterações ou por critério de aceitação. Neste caso, deve ser fornecido um limiar do valor da função de custo para que o algoritmo pare.

```
backpropagation <- function(dados, theta, target, learning_rate =
↪  NULL, epochs = NULL, acceptance = NULL){
```

```
  if(!is.list(theta)){
    stop("Back propagation: Theta deve ser uma lista com pesos e
    ↪  viéses.")
  } else if(!is.data.frame(dados) | !is_tibble(dados)){
    stop("Back propagation: Dados deve ser uma dataframe ou
    ↪  tibble.")
  } else if(!is.numeric(target)){
    stop("Back propagation: Target deve ser um vetor numérico.")
  } else if ((!is.null(epochs) & !is.null(acceptance)) |
↪  (is.null(epochs) & is.null(acceptance))){
    stop("Back propagation: Escolha se devem ser realizadas
    ↪  iterações programadas ou se o critério de parada será
    ↪  definido por um limite.")
  } else if (!is.numeric(learning_rate) & !is.null(learning_rate)
↪  & learning_rate > 0){
    stop("Back propagation: Learning rate deve ser um número real
    ↪  positivo.")
  } else if (!is.integer(epochs) & !is.null(epochs)){
    stop("Back propagation: Epochs deve ser um número inteiro
    ↪  positivo.")
  }


# Se for definido por learning rate
 if(!is.null(learning_rate)){
  loss_history <- numeric(epochs)
# initiate loop
  for(i in 1:epochs){
    #1 forward computation and gradient
    grad_results <- gradiente(dados, theta, target)
    # disponivel:
    ## gradiente.
    ## loss results: yhat, hidden, pre_activation
    loss_history[i] <- grad_results$loss_results$loss
    #2 back propagate the gradient
    # novos pesos = theta - learning rate * gradiente

    gradient_list <- list(
      W1_update = matrix(grad_results$grad[1:4], nrow = 2, byrow
      ↪  = TRUE),
      b12_update = grad_results$grad[7:8],
      W2_update = matrix(grad_results$grad[5:6], nrow = 2, byrow
      ↪  = TRUE),
      b3_update = grad_results$grad[9]
    )
```

```
   theta <- list( #usaremos o mesmo nome para atualizar no loop
     W1 = theta[[1]] - learning_rate*gradient_list$W1_update,
     b12 = theta[[2]] - learning_rate*gradient_list$b12_update,
     W2 = theta[[3]] - learning_rate*gradient_list$W2_update,
     b3 = theta[[4]] - learning_rate*gradient_list$b3_update
   )
 }
 # outputs: pesos finais e histórico de loss
 return(list(
   theta = theta,
   loss_history = loss_history
   )
 )
} else if (!is.null(acceptance)){
  # Se for definido por acceptance

  i <- 1
  loss_history <- Inf

  while(loss_history[i] > acceptance){
    #1 forward computation and gradient
   grad_results <- gradiente(dados, theta, target)
   # disponivel:
   ## gradiente.
   ## loss results: yhat, hidden, pre_activation
   loss_history[i] <- grad_results$loss_results$loss
   #2 back propagate the gradient
   # novos pesos = theta - learning rate * gradiente

   gradient_list <- list(
     W1_update = matrix(grad_results$grad[1:4], nrow = 2, byrow
     ↪  = TRUE),
     b12_update = grad_results$grad[7:8],
     W2_update = matrix(grad_results$grad[5:6], nrow = 2, byrow
     ↪  = TRUE),
     b3_update = grad_results$grad[9]
   )

   theta <- list( #usaremos o mesmo nome para atualizar no loop
     W1 = theta[[1]] - learning_rate*gradient_list$W1_update,
     b12 = theta[[2]] - learning_rate*gradient_list$b12_update,
     W2 = theta[[3]] - learning_rate*gradient_list$W2_update,
     b3 = theta[[4]] - learning_rate*gradient_list$b3_update
   )
```

```
    i <- i+1
   }
   return(list(
    theta = theta,
    loss_history = loss_history,
    iterations = i-1
    )
   )
 }
}
```

A seguir é apresentado o resultado da função aplicada sobre o banco de validação, quando $\theta = (0, \ldots, 0)$, com taxa de aprendizagem $\epsilon = 0.1$ e 100 iterações.

```
theta <- list(
  W1 = matrix(rep(0, 4), nrow = 2, byrow = TRUE),
  b12 = c(0, 0),
  W2 = matrix(rep(0, 2), nrow = 2, byrow = TRUE),
  b3 = 0
)

learning_rate <-  0.1

epochs <- 100L

y <- dados_valid$y
x <- dados_valid[,c(1,2)]

back_validacao <- backpropagation(x, theta, y, learning_rate,
↪  epochs)
```

O menor custo obtido foi 164.8767649 e foi observado na iteração 25.

Um gráfico do histórico de custo é apresentado a seguir:

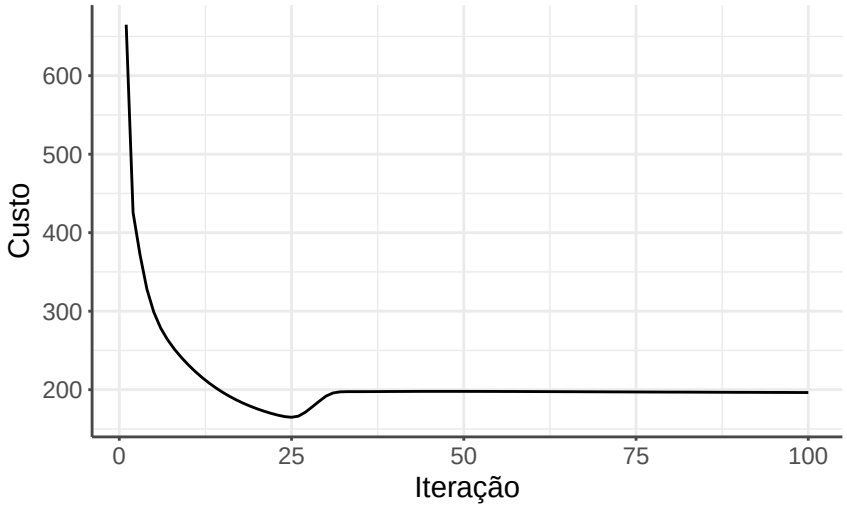

Figura 2: Histórico de custo para o banco de validação.

O vetor de pesos estimado foi:

$$W^{(1)} = \begin{pmatrix} -2.1967097 & -2.1967097 \\ 1.0577389 & 1.0577389 \end{pmatrix}, \quad \boldsymbol{b}^{(1)} = \begin{pmatrix} -6.8399939 \\ -6.8399939 \end{pmatrix},$$

$$W^{(2)} = \begin{pmatrix} -1.0835418 \\ -1.0835418 \end{pmatrix}, \quad \boldsymbol{b}^{(2)} = 21.8534499.$$

Interessantemente, os pesos de cada variavel para a camada seguinte, assim como os viéses, são idênticos. Além disso, os pesos aplicados nos neurônios da camada escondida também são idênticos. Isso pode ser um indicativo de que ambas as variáveis explicativas possuem importância igual para a previsão da variável resposta. Ao mesmo tempo, ambas X_1 e X_2 têm o mesmo processo gerador, então não parece ser surpreendente que os pesos sejam iguais.

Finalmente, o gráfico indica que possivelmente o algoritmo estacionou em um mínimo local de valor superior ao mínimo encontrado na iteração de número 25.

Item f

Lições aprendidas

É muito positivo construir funções que retornam listas com os resultados de cada etapa do algoritmo. Isso facilita a depuração e a compreensão do código.

Além disso, é importante testar cada função separadamente para garantir que ela está retornando o resultado esperado.

Frequentemente quando avançava um item, percebia que poderia incrementar funções anteriores que já retornavam internamente vetores que seriam usados futuramente. Dessa forma, é possível utilizar listas para fazer referencias fáceis a componentes de cada etapa e evitar repetições desnecessárias de cálculos, reduzindo o custo computacional da lista como um todo.