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*:

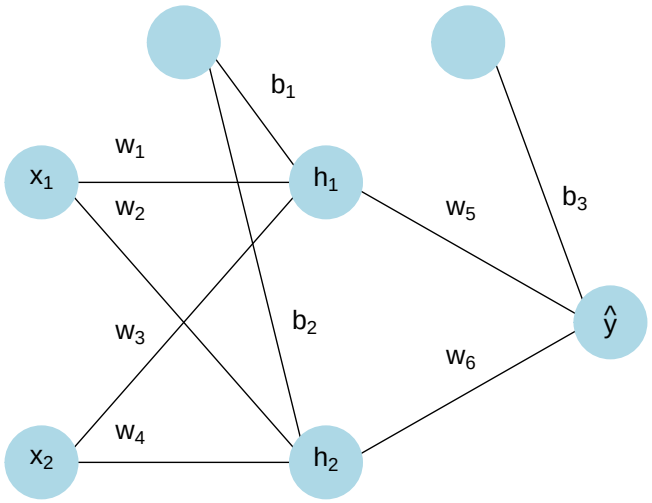

Figura 1: Arquitetura da rede neural artificial. Adotamos função de ativação sigmoide e linear nas camadas escondidas e de saída, respectivamente.

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.data.frame(x) | !tibble::is_tibble(x) & dim(x)[2]
↪  != 2){
    stop("Feed Forward: x deve ser uma dataframe ou tibble com 2
    ↪  colunas")
```

```
  }

  x <- as.matrix(x)

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

  a <- (t(W1)%*%t(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 <- data.frame(x1 = 1, x2 = 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.

```
# VERIFICAR A CONTA AQUI NO FINAL E PASSAR PARA MATRICIAL

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

  if(!is.data.frame(dados) | !tibble::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.")
  }

# transpoe os dados para termos acesso aos vetores de x que serao
↪  passados para a primeira camada da rede
  #tdados <- t(as.matrix(dados))

    yhat <- ffwd(dados, theta)$yhat
    a <- t(ffwd(dados, theta)$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

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

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

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

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.

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

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) | !tibble::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 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((-1)*diff*theta[[3]][1]*dados[,1]*a[,1]) #x1,
↪  a1, w5
  g_w2 <- 2*mean((-1)*diff*theta[[3]][2]*dados[,1]*a[,2]) #x1,
↪  a2, w6
  g_w3 <- 2*mean((-1)*diff*theta[[3]][1]*dados[,2]*a[,1]) #x2,
↪  a1, w5
  g_w4 <- 2*mean((-1)*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((-1)*diff*theta[[3]][1]*a[,1]) #a1, w5
  g_b2 <- 2*mean((-1)*diff*theta[[3]][2]*a[,2]) #a2, w6
  g_b3 <- 2*mean(diff*(-1)) #b3

  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,
        diff = diff
      )
    )
  )
}
```

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

```
grad_item_d <- gradiente(dados_train[,c(1,2)], theta,
↪  dados_train$y)$grad

grad_item_d %>%
  as_tibble() %>%
  reframe(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){

  if(!is.list(theta)){
    stop("Back propagation: Theta deve ser uma lista com pesos e
    ↪  viéses.")
  } else if(!is.data.frame(dados) | !tibble::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.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.")
  }

# initiate history and best theta
loss_history <- numeric(epochs)
best_theta <- theta

# initiate loop
  for(i in 1:epochs){
    grad_results <- gradiente(dados, theta, target)
    # fill in history
    loss_history[i] <- grad_results$loss_results$loss
    # fill in gradient list for update
    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]
    )
    # best theta condition
    if (i >= 2){
      if(loss_history[i] < loss_history[i-1]){
          best_theta <- theta
      }
```

```
    }
    # update theta with learning rate
    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,
    best_theta = best_theta
    )
  )
}
```

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 144.1480361 e foi observado na iteração 16.

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 para a iteração de menor valor da perda foi:

$$\hat{W}^{(1)} = \begin{pmatrix} -1.6958498 & -1.6958498 \\ -2.6882879 & -2.6882879 \end{pmatrix}, \quad \hat{\boldsymbol{b}}^{(1)} = \begin{pmatrix} 3.4379156 \\ 3.4379156 \end{pmatrix},$$

$$\hat{W}^{(2)} = \begin{pmatrix} 8.7439654 \\ 8.7439654 \end{pmatrix}, \quad \hat{\boldsymbol{b}}^{(2)} = 9.7284693.$$

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 16.

Item f

Apresente o gráfico do custo no conjunto de treinamento e no de validação (uma linha para cada) em função do número da interação do processo de otimização. Comente os resultados.

A seguir é construido o gráfico. Novamemente, trata-se de duas bases de dados relativamente simples, com um mesmo processo gerador dos dados e uma quantidade razoável de pontos, então é esperado que o custo de treinamento e validação sejam semelhantes.

Apesar disso, é possível notar que, na base de treinamento, o algoritmo escapa do mínimo identificado ligeiramente mais rápido, possivelmente devido à maior disponibilidade de dados. Neste caso, a base de treinamento é oito vezes maior que a base de validação.

```
# hiperparametros e base de treinamento
learning_rate <-  0.1
epochs <- 100L
y <- dados_train$y
x <- dados_train[,c(1,2)]

comparacao_treino_validacao <- tibble(
  epoch = rep(1:epochs, 2),
  base = rep(c("Validação", "Treinamento"), each = epochs),
  # back_validacao foi gerada no item e
  loss = c(back_validacao$loss_history, backpropagation(x, theta,
  ↪  y, learning_rate, epochs)$loss_history)
)

comparacao_treino_validacao %>%
  filter(loss < 170) %>%
  ggplot(aes(epoch, loss, group = base, color = base))+
  geom_line(alpha = .5)+
  labs(x = "Iteração",
       y = "Custo",
       color = "")+
  theme_bw()+
    theme(panel.border = element_blank(), # Remove all panel
    ↪  borders
        axis.line = element_line(color = "#474747"), # Add axis
        ↪  lines
        axis.ticks.x = element_line(color = "#474747"), # X axis
        ↪  ticks
        axis.ticks.y = element_line(color = "#474747"), # Y axis
        ↪  ticks
        axis.ticks.length = unit(0.05, "cm"), # Control tick
        ↪  length
        legend.position = "bottom")
```

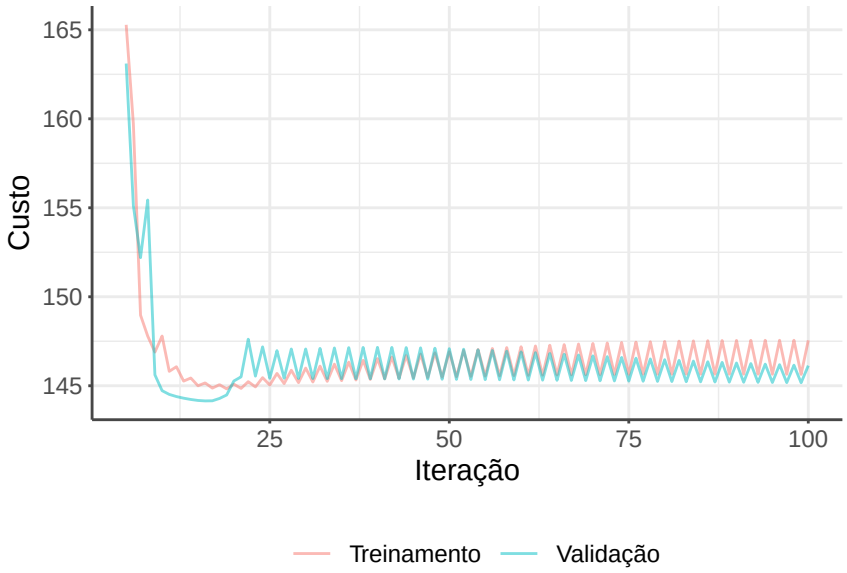

Figura 3: Histórico de custo para o banco de treinamento e validação.

Item g

Calcule os valores previstos ($\hat{y}_i$) e os resíduos ($y_i \hat{y}_i$) da rede no conjunto de teste e represente-os graficamente em função de x_1 e x_2. Dica: tome como base o código usado para a visualização da superfície $(E(Y|X_1, X_2), X_1, X_2)$. Altere o gradiente de cores e, se necessário, use pontos semi-transparentes. Analise o desempenho da rede nas diferentes regiões do plano. Há locais onde o modelo é claramente viesado ou menos acurado?

```
#calcular o back propagation e usar o melhor theta do conjunto de
↪  teste

# usamos o conjunto de teste
theta <- back_validacao$theta

resultados_gradiente <- gradiente(dados_test[,c(1,2)], theta,
↪  dados_test$y)

residuos = resultados_gradiente$loss_results$diff

# usamos o conjunto de teste
dados_graf_residuos <- dados_test %>%
  select(x1.obs, x2.obs, y) %>% #seleciona variaveis de interesse
  mutate(residuos = residuos,
```

```
          yhat = resultados_gradiente$loss_results$yhat
          )

# base do gráfico
plot_residuos <- ggplot(dados_graf_residuos, aes(x=x1.obs,
↪  y=x2.obs)) +
  geom_point(aes(colour=residuos), size=2, shape=15, alpha = .3)
  ↪  +
  coord_cartesian(expand=F) +
  scale_colour_gradient(low="white",
    high="black",
    name="Resíduos") +
  xlab(TeX("$X_1$")) + ylab(TeX("$X_2$"))+
  theme(legend.position = "bottom")

#gráfico da esperança
n <- 100
x1 <- seq(-3, 3, length.out=n)
x2 <- seq(-3, 3, length.out=n)
dados.grid <- as_tibble(expand.grid(x1, x2)) %>%
  rename_all(~ c("x1", "x2")) %>%
  mutate(mu=abs(x1^3 - 30*sin(x2) + 10))

plot_esperanca <- ggplot(dados.grid, aes(x=x1, y=x2)) +
  geom_point(aes(colour=mu), size=2, shape=15) +
  coord_cartesian(expand=F) +
  scale_colour_gradient(low="white",
  high="black",
  name=TeX("$E(Y|X_1, X_2)$")) +
  xlab(TeX("$X_1$")) + ylab(TeX("$X_2$"))+
  theme(legend.position = "bottom")
```

O gráfico dos resíduos utilizando o gradiente implementado e o gráfico da esperança são apresentados a seguir. Interessantemente, o resíduo parece ser parece ser maior em regiões em que a esperança $E(Y|X_1, X_2)$ é maior e menor onde a esperança é menor.

```
plot_grid(plot_residuos, plot_esperanca, ncol=2)
```

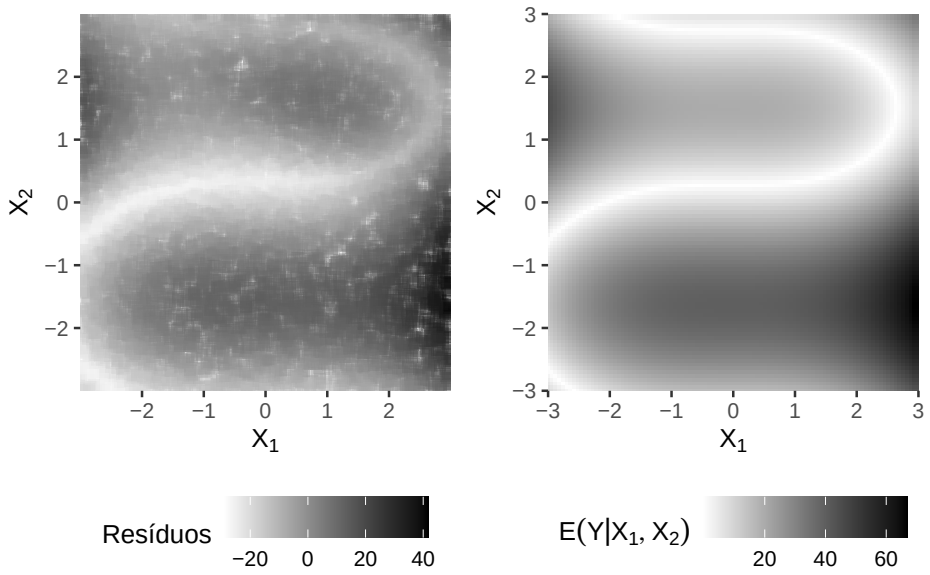

Figura 4: Resíduos da rede em função de X_1 e X_2.

Item h

Faça um gráfico do valor observado y_i em função do valor esperado $\hat{y}_i = E(Y_i|x_{1i}, x_{2i})$ para cada observação do conjunto de teste. Interprete o resultado.

No gráfico abaixo é possível observar que os valores de $\hat{y}$ tendem a se concentrar nas extremidades, parece haver uma sobreposição de comportamentos na transição entre os valores extremos do valor esperado.

Enquanto parece haver uma grande dispersão em toda a região observada, os valores entre os extremos de $\hat{y}$ parecem ter menor variância e estão mais concentrados em torno do valor esperado — apesar de haver pontos bem distantes. Pode-se dizer que o modelo é mais viesado para valores extremos de $\hat{y}$.

```
dados_graf_residuos %>%
  ggplot(aes(yhat, y))+
  geom_point(alpha = 0.3)+
  geom_abline(slope = 1, intercept = 0, color = "red")+
  theme_bw()+
  labs(x = TeX("$\\hat{y}_i$"), y = TeX("$y_i$"))+
  theme(panel.border = element_blank(), # Remove all panel
  ↪  borders
    axis.line = element_line(color = "#474747"), # Add axis lines
    axis.ticks.x = element_line(color = "#474747"), # X axis
    ↪  ticks
```

```
    axis.ticks.y = element_line(color = "#474747"), # Y axis
    ↪  ticks
    axis.ticks.length = unit(0.05, "cm"), # Control tick length
    legend.position = "bottom",
    axis.title.y = element_text(angle = 0, vjust = 0.5))
```

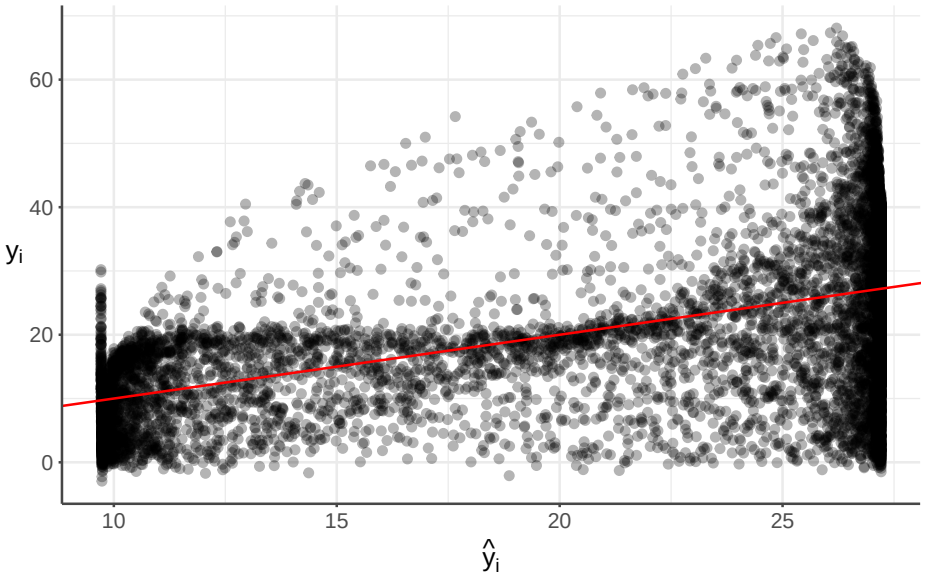

Figura 5

Item i

Para cada $k = 1, \ldots, 300$, recalcule o gradiente obtido no item d) usando apenas as k-primeiras observações do banco de dados original. Novamente, use $\boldsymbol{\theta} = (0.1, \ldots, 0.1)$. Apresente um gráfico com o valor do primeiro elemento do gradiente — isso é, a derivada parcial $\frac{\partial J}{\partial w_1}$ — em função do número de amostras k. Como referência, adicione uma linha horizontal vermelha indicando o valor obtido em d). Em seguida, use a função `microbenchmark` para comparar o tempo de cálculo do gradiente para $k = 300$ e $k = 100000$. Explique de que maneira os resultados dessa análise podem ser usados para acelerar a execução do item e).

Primeiro são obtidos os elementos $\frac{\partial J}{\partial w_1}$ para as amostras de tamanho 1 a 300:

```
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)
)

k_w1 <- function(dados, theta, i){
  base <- dados[1:i,c(1, 2)] #dados com linha 1 ate i
  y <- dados$y[1:i] #target com linhas 1 até i
  return(gradiente(base, theta, y)$grad[1]) #retorna w1
}

valores_w1 <- tibble(
  k = 1:300,
  w1 = map_dbl(k, ~ k_w1(dados, theta, .))
)
```

O gráfico do valor de w_1 calculado em função do tamanho da amostra é exibido a seguir. Como é de se esperar, o valor de w_1 tende a se aproximar do valor obtido no item d) à medida que o tamanho da amostra aumenta.

```
ggplot(valores_w1, aes(k, w1, group = 1))+
  geom_line()+
  geom_hline(yintercept = grad_item_d["w1"], color = "red")+
  theme_bw()+
  labs(x = "Tamanho da amostra", y = TeX("$w_1$"))+
  theme(panel.border = element_blank(), # Remove all panel
  ↪  borders
    axis.line = element_line(color = "#474747"), # Add axis lines
    axis.ticks.x = element_line(color = "#474747"), # X axis
    ↪  ticks
    axis.ticks.y = element_line(color = "#474747"), # Y axis
    ↪  ticks
    axis.ticks.length = unit(0.05, "cm"), # Control tick length
    legend.position = "bottom",
    axis.title.y = element_text(angle = 0, vjust = 0.5))+
    annotate("text", x = 200, y = 0,
           label = paste0("w1 = ",round(grad_item_d["w1"],2)),
           hjust = 1, vjust = 0, size = 4, color = "red")
```

Figura 6

```
res <- microbenchmark::microbenchmark(
  k_300 = k_w1(dados, theta, 300),
  k_100000 = k_w1(dados, theta, 100000),
  times = 100
)
```

Como esses resultados podem ser usados para acelerar a execução do item e)?

Item j

Ajuste sobre o conjunto de treinamento um modelo linear normal (modelo linear 1)

$$Y_i = N\left(\beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i}, \sigma\right)$$

usando a função `lm` do pacote `R` (ou outra equivalente). Em seguida, inclua na lista de covariáveis termos quadráticos e de interação linear. Isso é, assuma que no modelo linear 2,

$$E(Y|x_1, x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 x_2.$$

Compare o erro quadrático médio no conjunto de teste dos dois modelos lineares acima com o da rede neural ajustada anteriormente. Qual dos 3 modelos você usaria para previsão? Justifique sua resposta.

Os modelos lineares são ajustados a seguir:

```
MSE1 <- lm(y ~ x1.obs + x2.obs, data = dados_train) %>%
  anova() %>%
  pull(`Mean Sq`) %>%
  min()

MSE2 <- lm(y ~ x1.obs*x2.obs + I(x1.obs^2) + I(x2.obs^2), data =
↪  dados_train) %>%
  anova() %>%
  pull(`Mean Sq`) %>%
  min()
```

As perdas quadráticas são expostas na tabela a seguir:

```
tibble(
  Modelo = c("Rede Neural", "Modelo Linear 1", "Modelo Linear
  ↪  2"),
  MSE = c(min(back_validacao$loss_history), MSE1, MSE2)
) %>%
  mutate(MSE = round(MSE, 2)) %>%
  knitr::kable()
```

Modelo	MSE
Rede Neural	144.15
Modelo Linear 1	138.71
Modelo Linear 2	94.76

Pautado exclusivamente pelas perdas quadráticas, o modelo que deveria ser escolhido deveria ser o Modelo Linear 2, que inclui interações e termos quadráticos. No entanto, é importante considerar também a capacidade de generalização do modelo. Uma avaliação desses modelos considerando outras bases de dados para teste seria necessária para determinar o melhor desempenho.

Item k

Para cada modelo ajustado (os dois lineares e a rede neural), descreva o efeito no valor esperado da variável resposta causado por um aumento de uma unidade

da covariável x_1?

Para o Modelo 1, o incremento no valor esperado da variável resposta em função de x_1 é representado apenas por β_1. Para o Modelo 2, o incremento unitário no valor esperado da variável resposta é representado por $\frac{d\hat{y}}{d\,x_1} = \beta_1 + 2\beta_3 x_1 + \beta_5 x_2$. Para a rede neural, o incremento unitário no valor esperado da variável resposta é representado pelos pesos de $\boldsymbol{\theta}$ associados a x_1, ou seja, w_1 e w_2.

Temos portanto, como variação no valor esperado da variável resposta em função do incremento unitário de x_1:

o efeito de x1 na rede neural é a soma dos pesos w1 e w2? e como faz no modelo 2?

- Modelo 1: 1.2031906
- Modelo 2: 1.1912685 + 1.4082196x_1 + -2.09555 x_2
- Rede Neural: -3.3916996

Item l

Novamente, para cada um dos 3 modelos em estudo, calcule o percentual de vezes que o intervalo de confiança de 95% (para uma nova observação!) capturou o valor de y_i . Considere apenas os dados do conjunto de teste. No caso da rede neural, assuma que, aproximadamente, $\frac{y_i - \hat{y}_i}{\hat{\sigma}} N(0,1)$, onde $\hat{\sigma}$ representa a raiz do erro quadrático médio da rede. Comente os resultados. Dica: para os modelos lineares, use a função `predict(mod, interval="prediction")`.

Item m

Para o modelo linear 1, faça um gráfico de disperção entre x_1 e x_2 , onde cada ponto corresponde a uma observação do conjunto de teste. Identifique os pontos que estavam contidos nos respectivos intervalos e confianças utiliizando a cor verde. Para os demais pontos, use vermelho. Comente o resultado.

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.

A utilização de `cache: true` na seção YAML do documento é muito útil para economizar tempo para renderizar versões incrementadas do documento, especialmente no caso de termos que executar várias vezes cálculos custosos como o back propagation — Isso foi escrito quando eu usava o processo de *back-propagation* dentro de um `for`.

O tempo de processamento foi reduzido de 10 minutos a 5s usando contas matriciais. No entanto, tive muitos problemas na hora de converter o código. Uma revisão bem detalhada teve que ser feita para identificar pequenos problemas de matrizes transpostas.