Universidade de Brasília

DEPARTAMENTO DE ESTATÍSTICA

9 de abril de 2024

Lista 1: Ajustando uma RNA "no braço".

Aluno: William Edward Rappel de Amorim

Prof. Guilherme Rodrigues

Introdução a Redes Neurais Profundas

Tópicos em Estatística 1

(A) **As questões deverão ser respondidas em um único relatório *PDF* ou *html*, produzido usando as funcionalidades do *Rmarkdown* ou outra ferramenta equivalente.**

(B) **O aluno poderá consultar materiais relevantes disponíveis na internet, tais como livros, *blogs* e artigos.**

(C) **O trabalho é individual. Suspeitas de plágio e compartilhamento de soluções serão tratadas com rigor.**

(D) **Os códigos *R* utilizados devem ser disponibilizados na integra, seja no corpo do texto ou como anexo.**

(E) **O aluno deverá enviar o trabalho até a data especificada na plataforma Aprender 3.**

(F) **O trabalho será avaliado considerando o nível de qualidade do relatório, o que inclui a precisão das respostas, a pertinência das soluções encontradas, a formatação adotada, dentre outros aspectos correlatos.**

(G) **Escreva seu código com esmero, evitando operações redundantes, comentando os resultados e usando as melhores práticas em programação.**

Considere um processo gerador de dados da forma

$$\begin{aligned} Y &\sim N(\mu, \sigma^2 = 1) \\ \mu &= |X_1^3 - 30 \text{ sen}(X_2) + 10| \\ X_j &\sim \text{Uniforme}(-3, 3), \quad j = 1, 2. \end{aligned}$$

Neste modelo (que iremos considerar como o "**modelo real**"), a esperança condicional de Y é dada por $E(Y|X_1, X_2) = |X_1^3 - 30 \text{ sen}(X_2) + 10|$. A superfície tridimensional $(E(Y|X_1, X_2), X_1, X_2)$ está representada em duas dimensões cartesianas na Figura 1.

O código a seguir simula $m = 100.000$ observações desse processo.

```
### 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))
```

Nesta lista estamos interessados em estimar o modelo acima usando uma rede neural simples, ajustada sobre os dados simulados. Precisamente, queremos construir uma rede neural com apenas uma camada escondida contendo dois neurônios.

Matematicamente, a rede é descrita pelas seguintes equações:

$$\begin{aligned} h_1 &= \phi(x_1 w_1 + x_2 w_3 + b_1) = \phi(a_1) \\ h_2 &= \phi(x_1 w_2 + x_2 w_4 + b_2) = \phi(a_2) \\ \hat{y} &= h_1 w_5 + h_2 w_6 + b_3, \end{aligned}$$

onde $\phi(x) = \frac{1}{1+e^{-x}}$ representa a função de ativação logística (sigmoide).

Adotaremos como função de custo o erro quadrático médio, expresso por

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

onde x_{ji} representa a j-ésima covariável (*feature*) da i-ésima observação, $\boldsymbol{\theta} = (w_1, \ldots, w_6, b_1, b_2, b_3)$ é o vetor de pesos (parâmetros) e, pela definição da rede,

$$f(x_{1i}, x_{2i}; \boldsymbol{\theta}) = \hat{y}_i = \phi(x_{1i} w_1 + x_{2i} w_3 + b_1) w_5 + \phi(x_{1i} w_2 + x_{2i} w_4 + b_2) w_6 + b_3.$$

Uma representação gráfica da rede está apresentada na Figura 2.

Em notação matricial, a rede neural pode ser descrita por

$$\begin{aligned} \boldsymbol{a} &= \boldsymbol{W}^{(1)\top} \boldsymbol{x} + \boldsymbol{b}^{(1)} \\ \boldsymbol{h} &= \phi(\boldsymbol{a}) \\ \hat{y} &= \boldsymbol{W}^{(2)\top} \boldsymbol{h} + b_3, \end{aligned}$$

onde

$$\boldsymbol{W}^{(1)} = \begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix}, \quad \boldsymbol{W}^{(2)} = \begin{pmatrix} w_5 \\ w_6 \end{pmatrix}, \quad \boldsymbol{b}^{(1)} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}, \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \boldsymbol{h} = \begin{pmatrix} h_1 \\ h_2 \end{pmatrix}, \quad \boldsymbol{a} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}.$$

```r
### Figura 1: Gerando o gráfico da superfície
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))

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("$X_2$"))
```

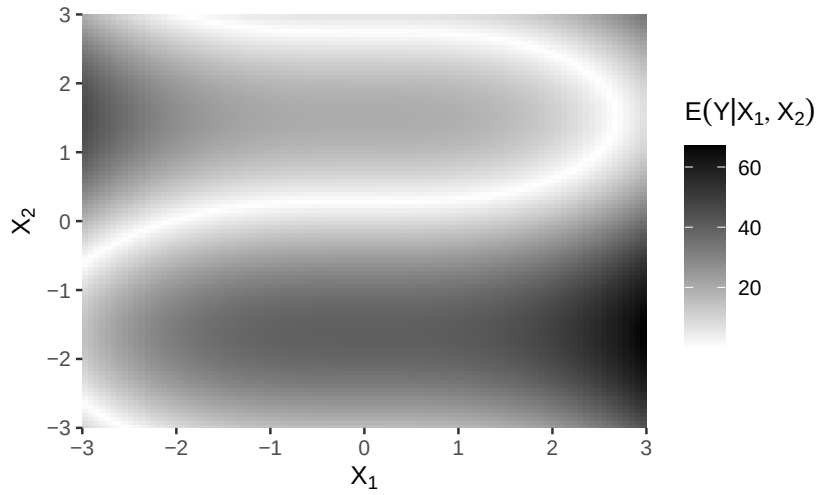


Figura 1: Gráfico da superfície do valor esperado da variável resposta Y em função das variáveis de entrada X_1 e X_2.

```r
### Figura 2: Arquitetura da RNA.
par(mar=c(0, 0, 0, 0))
wts_in <- rep(1, 9)
struct <- c(2, 2, 1) # dois inputs, dois neurônios escondidos e um otput
plotnet(wts_in, struct = struct,
       x_names="", y_names="",
       node_labs=F, rel_rsc=.7)
aux <- list(
 x=c(-.8, -.8, 0, 0, .8, rep(-.55, 4), -.12, -0.06, .38, .38, .7),
 y=c(.73, .28, .73, .28, .5, .78, .68, .48, .32, .88, .5, .68, .44, .7),
 rotulo=c("x_1", "x_2", "h_1", "h_2", "\\hat{y}", paste0("w_", 1:4),
          "b_1", "b_2", "w_5", "w_6", "b_3")
)
walk(transpose(aux), ~ text(.$x, .$y,
                            TeX(str_c("$", .$rotulo, "$")), cex=.8))
```

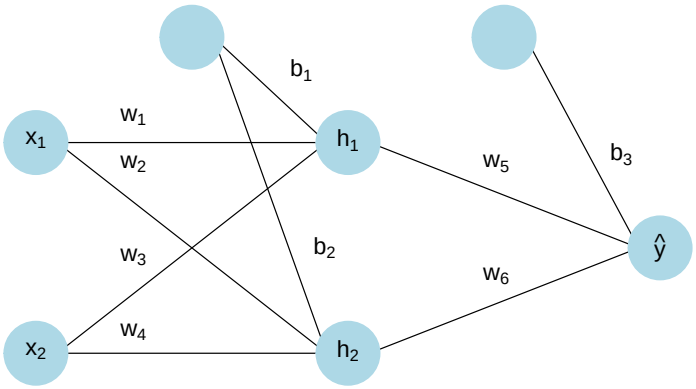


Figura 2: Arquitetura da rede neural artificial. Adotamos função de ativação sigmoide e linear nas camadas escondidas e de saída, respectivamente.

Considerando as informações acima, responda os itens a seguir.

a) Crie uma função computacional para calcular o valor previsto 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 $\boldsymbol{x} = (1, 1)$. Dica: veja o Algoritmo 6.3 do livro Deep Learning.

Resolução:

Primeiro, criamos uma função para calcular a sigmoide, para um dado valor x.

```
# Função Sigmoide
sigmoide <- function(x) {
  return(1/(1+exp(-x)))
}
```

Em seguida, criamos uma função que irá realizar o *forward propagation*, para um vetor $\boldsymbol{\theta}$ com os parâmetros e um objeto $\boldsymbol{x}$ com as variáveis explicativas.

```
# Função para realizar o forward propagation, dados theta e x
forward_prop <- function(theta, x) {
  # Transformação de x para um formato de matriz
  ifelse(is.double(x), x <- as.matrix(x), x <- t(as.matrix(x)))

  # Extração dos parâmetros
  W1 <- matrix(data = theta[1:4], nrow = 2)
  W2 <- matrix(data = theta[5:6], nrow = 2)
  b1 <- theta[7:8]
  b2 <- theta[9]

  # Camada escondida
  a <- matrix(data = rep(b1, ncol(x)), nrow = 2) + W1 %*% x
  h <- sigmoide(a)

  # Previsão
  y_hat <- as.double(b2 + t(W2) %*% h)

  return(y_hat)
}
```

Por último, aplicamos essa função para $\boldsymbol{\theta} = (0.1, \ldots, 0.1)$ e $\boldsymbol{x} = (1, 1)$, obtendo $\hat{y} = 0,215$.

```r
# Aplicando a função para os valores de theta e x especificados no enunciado
forward_prop(theta = rep(.1, 9), x = c(1, 1))
```

```
## [1] 0.2148885
```

b) Crie uma rotina computacional para calcular a função de custo $J(\boldsymbol{\theta})$. 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 $\boldsymbol{\theta} = (0.1, \ldots, 0.1)$?

Resolução:

Primeiro, criamos uma função para calcular a função de custo, dados os valores observados ($\boldsymbol{y}$) e os valores previstos ($\hat{\boldsymbol{y}}$).

```r
# Função para calcular a função de custo
mse_cost <- function(y_true, y_hat) {
  return(mean((y_true - y_hat)^2))
}
```

Depois, dividimos o conjunto de dados em treinamento e teste, separando também as variáveis explicativas e a resposta.

```r
# Divisão dos dados em treinamento, validação e teste
treino   <- dados[1:80000, ]
val      <- dados[80001:90000, ]
teste    <- dados[90001:nrow(dados), ]
x_treino <- treino %>%
  select(x1.obs, x2.obs)
x_val <- val %>%
  select(x1.obs, x2.obs)
x_teste  <- teste %>%
  select(x1.obs, x2.obs)
y_treino <- treino$y
y_val <- val$y
y_teste  <- teste$y

# Cálculo do custo da rede no banco de teste para theta especificado no enunciado
y_hat <- forward_prop(theta = rep(.1, 9), x = x_teste)
(mse_teste <- mse_cost(y_teste, y_hat))
```

```
## [1] 663.1286
```

Por último, realizamos o *forward propagation* para obter $\hat{\boldsymbol{y}}$ e calculamos o custo da rede no banco de teste para $\boldsymbol{\theta} = (0.1, \ldots, 0.1)$ e obtemos $J(\boldsymbol{\theta}) = 663.13$.

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

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

Resolução:

Pela regra da cadeia, tem-se que

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \left(\frac{\partial J}{\partial w_1}, \ldots, \frac{\partial J}{\partial b_3} \right) = \sum_{i=1}^{n_{treino}} \frac{\partial J(\hat{y}_i)}{\partial \hat{y}_i} \nabla_{\boldsymbol{\theta}} \hat{y}_i.$$

Dessa forma, o primeiro passo é calcular a derivada parcial da função de custo $J(\hat{y}_i)$ em relação a um dado valor previsto $\hat{y}_i$:

$$\frac{\partial J(\hat{y}_i)}{\partial \hat{y}_i} = \frac{1}{m} 2(y_i - \hat{y}_i)(-1) = -\frac{2}{m}(y_i - \hat{y}_i).$$

Em seguida, calculamos as derivadas parciais de $\hat{y}_i$ em relação a cada um dos parâmetros. Neste processo, a derivada da função de ativação $\phi(x)$ será dada por $\phi'(x) = \frac{\exp(-x)}{(1+\exp(-x))^2}$. De início, esse processo será realizado de forma individual para cada parâmetro:

$$\begin{aligned}
\frac{\partial \hat{y}_i}{\partial w_1} &= \frac{\partial \hat{y}_i}{\partial h_1}\frac{\partial h_1}{\partial w_1} = w_5\phi'(a_1)x_{1i} = w_5\phi'(x_{1i}w_1 + x_{2i}w_3 + b_1)x_{1i},\\
\frac{\partial \hat{y}_i}{\partial w_2} &= \frac{\partial \hat{y}_i}{\partial h_2}\frac{\partial h_2}{\partial w_2} = w_6\phi'(a_2)x_{1i} = w_6\phi'(x_{1i}w_2 + x_{2i}w_4 + b_2)x_{1i},\\
\frac{\partial \hat{y}_i}{\partial w_3} &= \frac{\partial \hat{y}_i}{\partial h_1}\frac{\partial h_1}{\partial w_3} = w_5\phi'(a_1)x_{2i} = w_5\phi'(x_{1i}w_1 + x_{2i}w_3 + b_1)x_{2i},\\
\frac{\partial \hat{y}_i}{\partial w_4} &= \frac{\partial \hat{y}_i}{\partial h_2}\frac{\partial h_2}{\partial w_4} = w_6\phi'(a_2)x_{2i} = w_6\phi'(x_{1i}w_2 + x_{2i}w_4 + b_2)x_{1i},\\
\frac{\partial \hat{y}_i}{\partial w_5} &= h_1 = \phi(a_1) = \phi(x_{1i}w_1 + x_{2i}w_3 + b_1),\\
\frac{\partial \hat{y}_i}{\partial w_6} &= h_2 = \phi(a_2) = \phi(x_{1i}w_2 + x_{2i}w_4 + b_2),\\
\frac{\partial \hat{y}_i}{\partial b_1} &= \frac{\partial \hat{y}_i}{\partial h_1}\frac{\partial h_1}{\partial b_1} = w_5\phi'(a_1) = w_5\phi'(x_{1i}w_1 + x_{2i}w_3 + b_1),\\
\frac{\partial \hat{y}_i}{\partial b_2} &= \frac{\partial \hat{y}_i}{\partial h_2}\frac{\partial h_2}{\partial b_2} = w_6\phi'(a_2) = w_6\phi'(x_{1i}w_2 + x_{2i}w_4 + b_2),\\
\frac{\partial \hat{y}_i}{\partial b_3} &= 1.
\end{aligned}$$

Agora, o procedimento será realizado de forma matricial, pois isto facilitará a implementação do *back-propagation*. Tem-se que

$$\begin{aligned}
\nabla_{\boldsymbol{W}^{(1)}}\hat{y}_i &= (\boldsymbol{W}^{(2)} \odot \phi'(\boldsymbol{a}))\boldsymbol{x}^\top = \left(\begin{pmatrix} w_5 \\ w_6 \end{pmatrix} \odot \begin{pmatrix} \phi'(a_1) \\ \phi'(a_2) \end{pmatrix}\right)\begin{pmatrix} x_{1i} & x_{2i} \end{pmatrix} = \begin{pmatrix} w_5\phi'(a_1)x_{1i} & w_5\phi'(a_1)x_{2i} \\ w_6\phi'(a_2)x_{1i} & w_6\phi'(a_2)x_{2i} \end{pmatrix},\\
\nabla_{\boldsymbol{W}^{(2)}}\hat{y}_i &= \boldsymbol{h}^\top = \phi(\boldsymbol{a}^\top) = \begin{pmatrix} h_1 & h_2 \end{pmatrix} = \begin{pmatrix} \phi(a_1) & \phi(a_2) \end{pmatrix} = \begin{pmatrix} \phi(x_{1i}w_1 + x_{2i}w_3 + b_1) & \phi(x_{1i}w_2 + x_{2i}w_4 + b_2) \end{pmatrix},\\
\nabla_{\boldsymbol{b}^{(1)}}\hat{y}_i &= \boldsymbol{W}^{(2)} \odot \phi'(\boldsymbol{a}) = \begin{pmatrix} w_5 \\ w_6 \end{pmatrix} \odot \begin{pmatrix} \phi'(a_1) \\ \phi'(a_2) \end{pmatrix} = \begin{pmatrix} w_5\phi'(a_1) \\ w_6\phi'(a_2) \end{pmatrix} = \begin{pmatrix} w_5\phi'(x_{1i}w_1 + x_{2i}w_3 + b_1) \\ w_6\phi'(x_{1i}w_2 + x_{2i}w_4 + b_2) \end{pmatrix},\\
\frac{\partial \hat{y}_i}{\partial b_3} &= 1.
\end{aligned}$$

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

Resolução:

Primeiro, criamos uma função para calcular a derivada da sigmoide, para um dado valor de x. A derivada da sigmoide é dada por $\phi'(x) = \frac{\exp(-x)}{(1+\exp(-x))^2}$. O código R abaixo faz a implementação dessa função.

```r
# Função Derivada da Sigmoide
derivada_sigmoide <- function(x) {
  return(exp(-x)/((1+exp(-x))^2))
}
```

Em seguida, passamos para a criação da função que implementará o *back-propagation*, dados $\boldsymbol{\theta}$, $\boldsymbol{x}$ e $\boldsymbol{y}$. Essa rotina computacional fornecerá como saída o gradiente definido no item c), ou seja, a saída será um vetor com 9 valores, cada valor representando a derivada parcial relativa a um dos parâmetros (w_1, w_2, ..., w_6, b_1, b_2, b_3).

```
# Função para realizar o back-propagation para um vetor theta, uma dada matriz design
# e as observações
back_prop <- function(theta, x, y){
  ### Primeiro, deve-se realizar o forward propagation
  ifelse(is.double(x), x <- as.matrix(x), x <- t(as.matrix(x)))
  W1 <- matrix(data = theta[1:4], nrow = 2)
  W2 <- matrix(data = theta[5:6], nrow = 2)
  b1 <- theta[7:8]
  b2 <- theta[9]
  a <- matrix(data = rep(b1, ncol(x)), nrow = 2) + W1 %*% x
  h <- sigmoide(a)
  y_hat <- as.double(b2 + t(W2) %*% h)

  ### Em seguida, passamos para a implementação do back propagation
  ## Camada final: k = 2
  # Primeiro, calculamos o gradiente da função de custo em relação ao valor previsto
  g <- -2*(y - y_hat)/length(y)
  # Como a última camada possui função de ativação linear, g já é o gradiente em
  # relação ao valor pré-ativação da última camada
  # Obtemos o gradiente em relação ao termo de viés
  grad_b2 <- sum(g)
  # Calculamos o gradiente em relação aos pesos
  grad_W2 <- g %*% t(h)
  # Atualizamos o valor de g
  g <- W2 %*% g
  ## Camada escondida: k = 1
  # Calculamos o gradiente em relação ao valores de ativação
  g <- g * derivada_sigmoide(a)
  # Obtemos o gradiente em relação ao termo de viés
  grad_b1 <- rowSums(g)
  # Calculamos o gradiente em relação aos pesos
  grad_W1 <- g %*% t(x)
  # Atualizamos o valor de g
  g <- W1 %*% g

  ### Final
  # Criamos um vetor com os gradientes de cada parâmetro
  vetor_grad <- c(grad_W1, grad_W2, grad_b1, grad_b2)
  names(vetor_grad) <- c(paste0("w", 1:6), paste0("b", 1:3))
  return(vetor_grad)
}
```

Por último, aplicamos o *back propagation* ao conjunto de dados de treino, para $\boldsymbol{\theta} = (0.1, \ldots, 0.1)$.

```
# Aplicando a função no banco de treinamento e theta especificado no enunciado
back_prop(theta = rep(.1, 9), x = x_treino, y = y_treino)
```

```
##          w1          w2          w3          w4          w5          w6
##  -0.1767887  -0.1767887   0.6458047   0.6458047 -22.3246918 -22.3246918
##          b1          b2          b3
##  -1.0732662  -1.0732662 -43.4113383
```

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 $\boldsymbol{\theta} = (0, \ldots, 0)$, 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.

Resolução:

Antes de iniciar o processo iterativo, devemos especificar os valores utilizados e criar os objetos que receberão os resultados. Primeiro, especificamos a taxa de aprendizagem e o número de iterações.

```
# Taxa de aprendizagem
epsilon <- 0.1

# Número de iterações
M <- 100
```

Depois, criamos uma lista que receberá os parâmetros estimados em cada iteração. Como são 9 parâmetros, cada elemento da lista corresponderá a um vetor de 9 valores. Inicializamos o primeiro elemento da lista com os valores especificados no enunciado: $\boldsymbol{\theta} = (0, \ldots, 0)$.

```
# Lista para receber os parâmetros estimados em cada iteração
theta_est <- list()

# Theta inicial
theta_est[[1]] <- rep(0, 9)
```

Em seguida, criamos dois vetores numéricos (preenchidos com 0) de comprimento 100, que receberão os custos de treino e validação em cada iteração.

```
# Vetores para receber os custos de, respectivamente, treino e validação em cada iteração
custo_treino <- custo_val <- numeric(M)
```

Finalmente, passamos para a implementação do método do gradiente, para encontrar os parâmetros que minimizam a função de custo no banco de validação. Para isso, em cada iteração, aplicamos o *back-propagation* no conjunto de treinamento, depois avaliamos os custos de treino e validação, atualizamos os valores dos parâmetros utilizando o gradiente calculado e passamos para a próxima iteração.

```
# Execução
for(i in 1:M) {
  # Cálculo dos gradientes dos parâmetros
  grad <- back_prop(theta = theta_est[[i]], x = x_treino, y = y_treino)

  # Cálculo do custo de treino
  custo_treino[i] <- mse_cost(y_treino, forward_prop(theta_est[[i]], x_treino))

  # Cálculo do custo de validação
  custo_val[i] <- mse_cost(y_val, forward_prop(theta_est[[i]], x_val))

  # Atualização dos parâmetros
  theta_est[[i+1]] <- theta_est[[i]] - epsilon*grad
}
```

Após este procedimento, o menor custo no banco de treino e sua respectiva iteração foi de:

```
min_custo_treino <- min(custo_treino)
round(min_custo_treino, 3); which.min(custo_treino)
```

```
## [1] 144.823
```

```
## [1] 19
```

O menor custo no banco de validação e sua respectiva iteração foi de:

```
min_custo_val <- min(custo_val)
round(min_custo_val, 3); which.min(custo_val)
```

```
## [1] 144.491
```

```
## [1] 21
```

O vetor de parâmetros estimado é aquele que minimizou o custo no banco de validação, ou seja:

```
min_theta <- unlist(theta_est[which.min(custo_val)])
round(min_theta, 3)
```

```
##     w1     w2     w3     w4     w5     w6     b1     b2     b3
## -1.132 -1.132 -2.255 -2.255  8.328  8.328  2.056  2.056 11.286
```

A partir dos resultados acima, percebe-se que o custo mínimo no banco de validação foi muito próximo ao custo mínimo no banco de treinamento, o que é um bom sinal. Passando para os parâmetros, percebe-se que a estimativas dos seguintes pares de parâmetros foram iguais: w_1 e w_2, w_3 e w_4, w_5 e w_6, b_1 e b_2. Além disso, o viés de maior magnitude foi encontrado na camada de saída, e os maiores pesos foram atribuídos às representações da camada escondida, e não da camada inicial (variáveis explicativas).

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

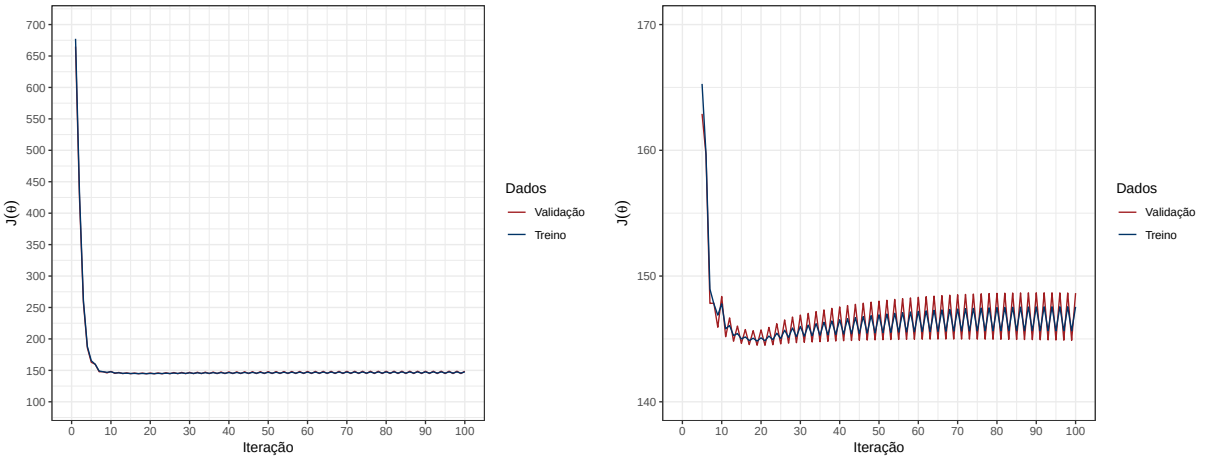
Resolução:

Abaixo, está o código que elabora o gráfico solicitado. Ao lado, está apresentado um segundo gráfico, que é idêntico ao primeiro, porém mostra apenas a partir da 5ª iteração e com *zoom*, para facilitar a visualização das curvas.

```
reshape2::melt(data.frame(val = custo_val,
                          treino = custo_treino,
                          iter = 1:M), id = "iter") %>%
  ggplot(aes(x = iter, y = value, colour = variable)) +
  geom_line() +
  labs(x = "Iteração", y = TeX("$J(\\theta)$")) +
  scale_colour_manual("Dados", values = c("#A11D21", "#003366"),
                      labels = c("Validação", "Treino")) +
  scale_x_continuous(limits = c(0,100), breaks = seq(0,100,10)) +
  scale_y_continuous(limits = c(100,700), breaks = seq(100,700,50)) +
  theme_bw()
reshape2::melt(data.frame(val = custo_val[5:M],
                          treino = custo_treino[5:M],
                          iter = 5:M), id = "iter") %>%
  ggplot(aes(x = iter, y = value, colour = variable)) +
  geom_line() +
  labs(x = "Iteração", y = TeX("$J(\\theta)$")) +
  scale_colour_manual("Dados", values = c("#A11D21", "#003366"),
                      labels = c("Validação", "Treino")) +
  scale_x_continuous(limits = c(0,100), breaks = seq(0,100,10)) +
  scale_y_continuous(limits = c(140,170), breaks = seq(140,170,10)) +
  theme_bw()
```

A partir da observação do gráfico, fica evidente que os custos de treino e teste sempre estiveram muito próximos, o que evidencia que o nosso modelo não sofreu o processo de *overfitting*. Apesar de estarem próximos, o custo da validação sempre esteve um pouco abaixo do custo de treino. Além disso, é perceptível que o maior decréscimo ocorre nas primeiras iterações. A partir da 7ª iteração, ambos os custos já atingem valores abaixo de 150 e estabilizam na 10ª iteração, permanecendo aproximadamente no mesmo patamar até a última iteração.

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?



(a) Completo

(b) A partir da 5ª iteração e com zoom

Figura 3: Custos de treinamento e validação para 100 iterações.

Resolução:

Primeiro, calculamos os valores previstos e os resíduos da rede no banco de teste.

```
# Valores previstos no conjunto de teste
y_hat <- forward_prop(theta = min_theta, x = x_teste)

# Resíduos
res_teste <- y_teste - y_hat
```

Em seguida, construímos o gráfico dos valores previstos em função de X_1 e X_2.

```
# Gráfico dos valores previstos em relação às features
cbind(y_hat, x_teste) %>%
  ggplot(aes(x = x1.obs, y = x2.obs)) +
  geom_point(aes(colour = y_hat), size = 2, shape = 15) +
  coord_cartesian(expand = F) +
  scale_colour_gradientn(colours = rev(heat.colors(5)),
                         name = TeX("$\\hat{Y}(X_1, X_2)$")) +
  labs(x = TeX("X_1"), y = TeX("X_2")) +
  scale_x_continuous(limits = c(-3,3), breaks = seq(-3,3)) +
  scale_y_continuous(limits = c(-3,3), breaks = seq(-3,3)) +
  theme_bw()
```

No gráfico, fica evidente que, a medida que X_1 e X_2 aumentam simultaneamente, o valor previsto diminui. Como o valor previsto é uma tentativa de estimar a esperança, é desejável que os valores previstos estejam próximos do valor verdadeiro da esperança. Porém, ao comparar o gráfico acima com a Figura 2 (superfície $(E(Y|X_1, X_2), X_1, X_2)$, vemos que os dois são muito diferentes, ou seja, as previsões são ruins. Isto será confirmado ao analisar o gráfico abaixo, que ilustra o comportamento dos resíduos em função de X_1 e X_2.

```
# Gráfico dos resíduos em relação às features
cbind(res_teste, x_teste) %>%
  ggplot(aes(x = x1.obs, y = x2.obs)) +
  geom_point(aes(colour = res_teste), size = 2, shape = 15) +
  coord_cartesian(expand = F) +
  scale_colour_gradient2(low = "black", high = "#A11D21", name = "Resíduos") +
  labs(x = TeX("X_1"), y = TeX("X_2")) +
  scale_x_continuous(limits = c(-3,3), breaks = seq(-3,3)) +
  scale_y_continuous(limits = c(-3,3), breaks = seq(-3,3)) +
  theme_bw()
```

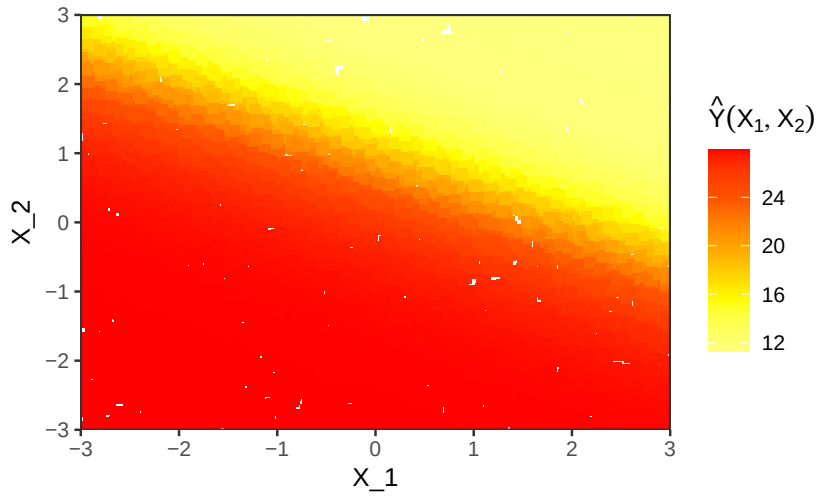


Figura 4: Gráfico da superfície do valor previsto ($\hat{Y}$) em função das variáveis de entrada X_1 e X_2.

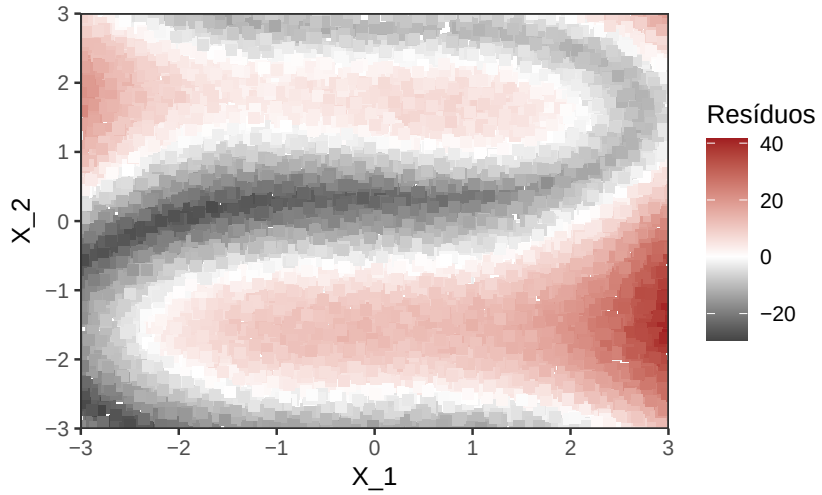


Figura 5: Gráfico da superfície dos resíduos em função das variáveis de entrada X_1 e X_2.

Nas regiões mais claras, os resíduos apresentam valores de menor magnitude, ou seja, a previsões estão mais próximas dos valores reais. Porém, existem regiões em que os resíduos assumem valores elevados, por exemplo, na região em que X_1 está próximo de 3 e X_2 entre 0 e -2. Além desta região, ao longo da curva em preto que parece um "S" invertido, os resíduos também assumem valores de magnitude elevada, só que de sinal oposto. Nessas regiões, o modelo é claramente viesado.

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.

Resolução:

Abaixo, está apresentado o gráfico do valor observado (eixo Y) em função do valor previsto (eixo X).

```
data.frame(y_hat, y_teste) %>%
  ggplot(aes(x = y_hat, y = y_teste)) +
  geom_point(size = 1, alpha = .1) +
  geom_abline(colour = "#A11D21") +
  scale_x_continuous(limits = c(0, 60)) +
  labs(x = TeX("$\\hat{Y}$"), y = "Y") +
  theme_bw()
```

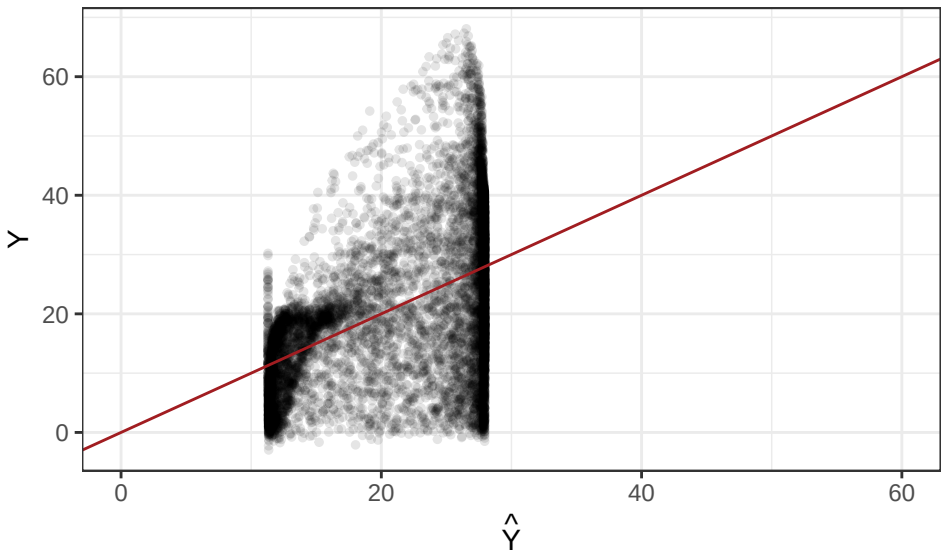


Figura 6: Dados de Teste - Previsto x Observado.

A partir da observação do gráfico, fica evidente que as previsões da rede estiveram em um intervalo muito mais restrito do que o intervalo dos valores observados. Enquanto os valores reais variaram de aproximadamente -2 até um pouco mais de 67, os valores previstos estiveram restritos ao intervalo $(11, 28)$. Com isso, muitas das previsões foram completamente equivocadas e muito distantes do valor observado.

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).

Resolução:

Primeiro, devemos inicializar os objetos que serão utilizados na execução do processo iterativo.

```
# Número de observações
k <- 300

# Matriz para armazenar os gradientes para cada tamanho de amostra
grad_k <- matrix(NA, nrow = k, ncol = 9)
colnames(grad_k) <- c(paste0("w", 1:6), paste0("b", 1:3))
```

Em seguida, passamos para a implementação do gradiente, apenas para as k-primeiras observações, com $k = 1, \ldots, 300$.

```
# Execução
for(i in 1:k) {
  grad_k[i,] <- back_prop(theta = rep(.1, 9), x = dados[1:i, 1:2], y = dados$y[1:i])
}
```

Agora, construímos o gráfico que ilustra o primeiro elemento do gradiente em função do número de amostras.

```
# Gráfico
linha <- back_prop(theta = rep(.1, 9), x = x_treino, y = y_treino)[1]
data.frame(k = 1:k, grad_w1 = grad_k[,1]) %>%
  ggplot(aes(x = k, y = grad_w1)) +
```

```
  geom_point(size = 1, colour = "#003366") +
  geom_hline(yintercept = linha, colour = "#A11D21") +
  labs(x = "Número de amostras (k)", y = TeX("$\\frac{\\partial J}{\\partial w_1}$")) +
  scale_x_continuous(breaks = seq(0,300,50)) +
  theme_bw()
```

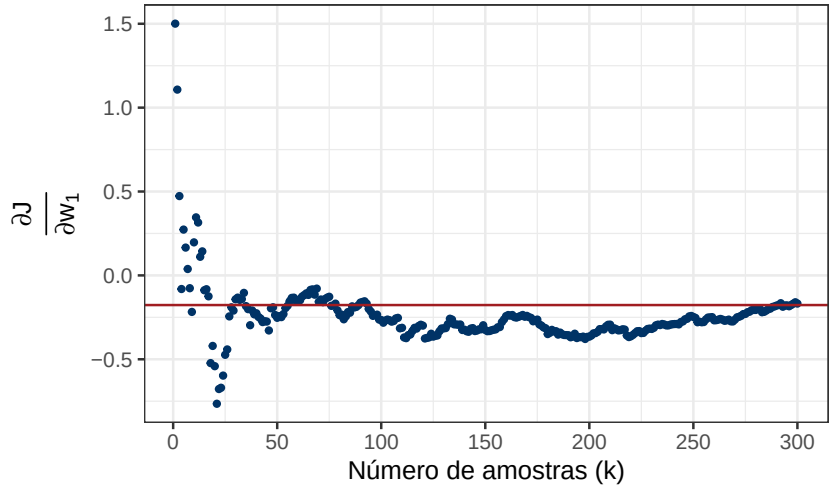


Figura 7: Primeiro elemento do gradiente em função do número de amostras.

Ao visualizar o gráfico acima, percebe-se que, para um número pequeno de amostras, o valor do gradiente varia bastante. Porém, a partir de 100 observações, o valor se torna aproximadamente o mesmo para qualquer que seja o k, considerando k no intervalo $[100, 300]$. Além disso, o gradiente parece convergir para o valor obtido no item d) (linha vermelha horizontal), que foi obtido ao utilizar o conjunto de treinamento inteiro ($k = 80000$). Agora, iremos comparar o tempo de execução para $k = 300$ e $k = 100000$.

```
# Usando microbenchmark para comparar k = 300 e k = 100000
microbenchmark('Se k = 300:' = back_prop(rep(.1, 9), x_treino[1:300,],
                                         y_treino[1:300]),
               'Se k = 100000:' = back_prop(rep(.1, 9), dados[, 1:2], dados$y))
```

```
## Unit: microseconds
##             expr       min      lq      mean    median         uq       max
##     Se k = 300:   284.201   314.0   425.092   429.201   475.6015   893.801
##  Se k = 100000: 42599.702 46257.2 50441.771 49735.801 53349.1010 72543.501
##  neval
##    100
##    100
```

Os resultados acima evidenciam que, ao utilizar apenas as primeiras 300 observações ao invés do conjunto de dados inteiro, a execução é, em média, mais de 125 vezes mais rápida. Como foi visto que, a partir de um certo valor de k, o gradiente é aproximadamente o mesmo e que a execução com menos observações é bem mais rápida do que com o conjunto de dados inteiro, podemos acelerar a execução do item e) a partir da seleção de um valor k que seja suficiente para obter um gradiente próximo do que seria obtido com todo o conjunto de dados de treino, e utilizar apenas as primeiras k observações de treino para realizar o *back propagation*.

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

$$Y_i \sim N(\beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i}, \sigma^2)$$

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.

Resolução:

Primeiro, ajustamos os dois modelos lineares e obtemos as previsões para o conjunto de teste.

```
# Modelo linear 1
mod1 <- lm(y ~ x1.obs + x2.obs, treino)
mod1_hat <- predict(mod1, newdata = x_teste)

# Modelo linear 2
mod2 <- lm(y ~ x1.obs + x2.obs + I(x1.obs^2) + I(x2.obs^2) + x1.obs:x2.obs, treino)
mod2_hat <- predict(mod2, newdata = x_teste)
```

Em seguida, calculamos o erro quadrático médio no conjunto de teste desses dois modelos e da rede neural.

```
## Erro quadrático médio no conjunto de teste
# Modelo linear 1
(mod1_mse <- mse_cost(y_teste, mod1_hat))
```

```
## [1] 134.5856
```

```
# Modelo linear 2
(mod2_mse <- mse_cost(y_teste, mod2_hat))
```

```
## [1] 93.62245
```

```
# Rede neural ajustada
(nn_mse <- mse_cost(y_teste, y_hat))
```

```
## [1] 140.5994
```

Os valores mostram que o modelo menos preciso é a rede neural, seguida pelo modelo linear apenas com as covariáveis e, como melhor modelo, temos o modelo linear com as covariáveis, termos quadráticos e de interação linear. Para previsão, eu usaria o modelo que obteve o menor erro quadrático médio no conjunto de teste, pois ele é o modelo mais preciso para realizar previsões de valores ainda não observados, isto é, eu escolheria o **modelo linear 2**.

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?

Resolução:

Para obter o efeito no valor esperado da variável resposta causado por um aumento de uma unidade da covariável x_1, basta derivar, para cada modelo, a equação da esperança estimada da variável resposta ($\hat{y}_i$) em relação a x_{1i}. Para o modelo linear 1, temos que $\frac{\partial \hat{y}_i}{\partial x_{1i}} = \hat{\beta}_1$. Com isso, a partir do ajuste do modelo no conjunto de treino, temos que, ao aumentar x_{1i} em 1 unidade, o valor esperado da resposta aumenta em 1,196 unidade.

Para o modelo linear 2, temos que $\frac{\partial \hat{y}_i}{\partial x_{1i}} = \hat{\beta}_1 + 2\hat{\beta}_3 x_{1i} + \hat{\beta}_5 x_{2i}$. Substituindo pelas estimativas obtidas ao ajustar o modelo no conjunto de treinamento, obtemos $1,212 + 1,387x_{1i} - 2,112x_{2i}$. Ou seja, o aumento esperado vai depender do nível da própria variável x_{1i} e também do nível da outra covariável (x_{2i}).

Por último, para a rede neural, temos que $\frac{\partial \hat{y}_i}{\partial x_{1i}} = \phi'(x_{1i}w_1 + x_{2i}w_3 + b_1)w_1w_5 + \phi'(x_{1i}w_2 + x_{2i}w_4 + b_2)w_2w_6$. Ao substituir os valores estimados ao ajustar a rede no conjunto de treino, obtemos $\phi'(-1,192x_{1i} - 2,295x_{2i} + 2,180)(-1,192)(8,273) + \phi'(-1,192x_{1i} - 2,295x_{2i} + 2,180)(-1,192)(8,273) = -19,728\phi'(-1,192x_{1i} - 2,295x_{2i} + 2,180)$. Logo, o aumento esperado irá depender do nível das duas covariáveis, além de ser necessário aplicar a função derivada da sigmoide.

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, $\frac{y_i - \hat{y}}{\hat{\sigma}} \sim N(0,1)$, 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")`.

Resolução:

A primeira etapa é calcular os intervalos dos modelos lineares, utilizando a função `predict(mod, interval="prediction")`.

```
# Intervalos de confiança do modelo linear 1
mod1_ci <- predict(mod1, newdata = x_teste, interval = "prediction")
mod1_linf <- mod1_ci[,2]
mod1_lsup <- mod1_ci[,3]

# Intervalos de confiança do modelo linear 2
mod2_ci <- predict(mod2, newdata = x_teste, interval = "prediction")
mod2_linf <- mod2_ci[,2]
mod2_lsup <- mod2_ci[,3]
```

Em seguida, calculamos os intervalos da rede neural. O enunciado evidencia que, a partir de uma estimativa do desvio padrão dos resíduos, é possível obter um intervalo de confiança normal utilizando a rede. Para estimar esse desvio padrão, não podemos utilizar o conjunto de teste, pois esses valores não foram observados. Por isso, a estimativa que utilizaremos para o desvio padrão será a raiz quadrada do erro quadrático médio no conjunto de **treinamento**. Assim, obtemos os intervalos para a rede neural.

```
# Intervalos de predição da rede neural
y_hat_treino <- forward_prop(theta = min_theta, x = x_treino)
mse_treino <- mse_cost(y_treino, y_hat_treino)
nn_linf <- y_hat - qnorm(.975)*sqrt(mse_treino)
nn_lsup <- y_hat + qnorm(.975)*sqrt(mse_treino)
```

Por último, calculamos e comparamos a cobertura das previsões intervalares de cada modelo no conjunto de teste.

```
## Percentual de cobertura no conjunto de teste
# Modelo linear 1
(mod1_cob <- mean(mod1_linf < y_teste & y_teste < mod1_lsup))
```

```
## [1] 0.9588
```

```
# Modelo linear 2
(mod2_cob <- mean(mod2_linf < y_teste & y_teste < mod2_lsup))
```

```
## [1] 0.9668
```

```
# Rede neural ajustada
(nn_cob <- mean(nn_linf < y_teste & y_teste < nn_lsup))
```

```
## [1] 0.9456
```

Os 3 modelos apresentaram uma cobertura próxima de 95%, o que é um resultado razoável, pois esta era a cobertura desejada. O modelo que mais se aproximou do valor desejado foi o modelo linear 1. Ele foi seguido pela rede, com cobertura menos que 1% distante do valor objetivo. O modelo com maior cobertura foi também aquele que esteve mais distante da cobertura desejada: o segundo modelo linear obteve cobertura de quase 97%.

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

Resolução:

O gráfico a seguir apresenta a identificação dos pontos que estavam contidos nos respectivos intervalos de confiança do **modelo linear 1**. Apesar da cobertura observada *global* estar próxima à nominal (95%),

quando analisada localmente, a cobertura empírica é muito diferente, a depender da região do espaço das observações.

```
# Gráfico

x_teste %>%
  mutate(y = y_teste,
         mod1_linf = mod1_linf,
         mod1_lsup = mod1_lsup,
         acertou = ifelse(mod1_linf < y_teste & y_teste < mod1_lsup,
                          "Acertou",
                          "Errou")) %>%
  ggplot(aes(x=x1.obs, y=x2.obs, color = acertou)) +
  geom_point(size = .5) +
  xlab(TeX("$x_1$")) +
  ylab(TeX("$x_2$")) +
  scale_color_manual(
    name = 'Intervalo 95%',
    values = c("darkgreen", "tomato"),
    labels = c("Acertou", "Errou")
  ) +
  theme_bw()
```

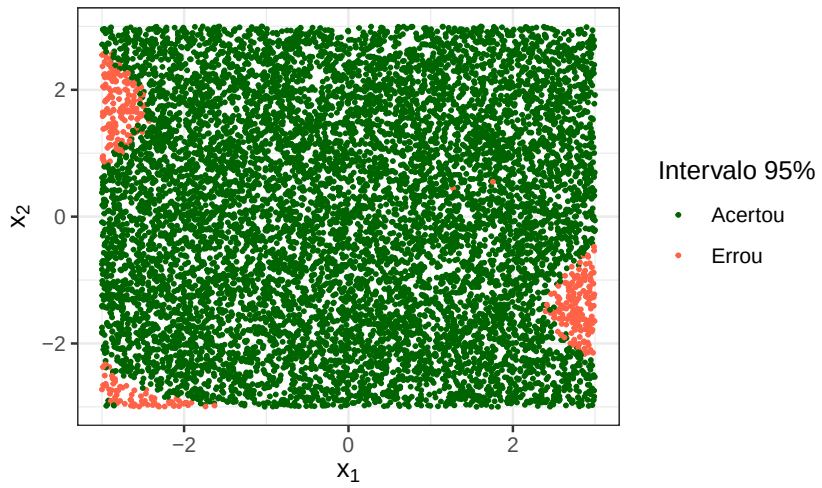


Figura 8: Identificação de erros e acertos dos intervalos de confiança feitos pelo Modelo Linear 1 para o conjunto de teste.