Lista 2: Dropout e Keras

César A. Galvão - 190011572

1 Questão 1

1.1 Item a)

Altere seu código da Lista 1 (ou, se preferir, os códigos disponibilizados como gabarito) para implementar a técnica dropout na camada de entrada e na camada intermediária. Use $p = 0,6$, onde p representa a probabilidade de inclusão de cada neurônio. Atenção: neste item, não é preciso calcular o custo da rede no conjunto de validação!

A cada nova iteração do algoritmo de otimização, a rede neural corrente gera estimativas pontuais aleatórias para as observações do conjunto de treinamento. Essas estimativas, por sua vez, são usadas para calcular o custo no conjunto de treinamento e atualizar os pesos da rede.

Reporte o menor custo observado durante o treinamento e salve os respectivos pesos para responder os demais itens da Questão 1.

A seguir é utilizado o código do gabarito da Lista 1, acrescido de uma alteração na função de *backpropagation* para incluir o dropout às unidades x_1, x_2, h_1 e h_2. O dropout é implementado por meio de uma máscara binária que é aplicada a cada unidade com probabilidade $p = 0,6$, gerada a cada iteração com o uso de `rbinom()`. A máscara é aplicada matricialmente a $\mathbf{X}$ e $\mathbf{h}$.

```
# funcoes de apoio ----
sigmoide <- function(x) {
  return(1/(1+exp(-x)))
}

derivada_sigmoide <- function(x) {
  return(exp(-x)/((1+exp(-x))^2))
}

mse_cost <- function(y_true, y_hat) {
  return(mean((y_true - y_hat)^2))
}

# forward propagation ----
feed_forward <- 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)
}

feed_forward_drop <- function(theta, x, drop_rate = 0.6) {
# Transformação de x para um formato de matriz
  ifelse(is.double(x), x <- as.matrix(x), x <- t(as.matrix(x)))

  #gera mascaras
  mask <- replicate(dim(x)[2],rbinom(4, 1, drop_rate))
  #aplica mascaras
  x <- x * mask[1:2,]

  # 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)

  #aplica mascaras
  h <- h * mask[3:4,]

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


feed_forward_scale <- function(theta, x, scale = 0.6) {
# Transformação de x para um formato de matriz
  ifelse(is.double(x), x <- as.matrix(x), x <- t(as.matrix(x)))

  # #gera mascaras
  # mask <- replicate(dim(x)[2],rbinom(4, 1, drop_rate))
  # #aplica mascaras
```

```r
  # x <- x * mask[1:2,]

  # Extração dos parâmetros
  W1 <- matrix(data = theta[1:4], nrow = 2)
  W2 <- matrix(data = theta[5:6], nrow = 2)
  scaled_W1 <- (scale) * W1
  scaled_W2 <- (scale) * W2
  b1 <- theta[7:8]
  b2 <- theta[9]
  # Camada escondida
  a <- matrix(data = rep(b1, ncol(x)), nrow = 2) + W1 %*% x
  h <- sigmoide(a)

  # #aplica mascaras
  # h <- h * mask[3:4,]

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



# back propagation ----
# para as epochs, é necessário rodar um for
back_prop_drop <- function(theta, x, y, drop_rate = 0.6){

  ### Primeiro, deve-se realizar o forward propagation
  ifelse(is.double(x), x <- as.matrix(x), x <- t(as.matrix(x)))

    #gera mascaras
    mask <- replicate(dim(x)[2],rbinom(4, 1, drop_rate))
    #aplica mascaras
    x <- x * mask[1:2,]

  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)

  #aplica mascaras
  h <- h * mask[3:4,]

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

  ### Em seguida, passamos para a implementação do back propagation
  ## Camada final: k = 2
```

```r
  # 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(
    list(
      vetor_grad = vetor_grad,
      mse_cost = mse_cost(y, y_hat))
    )
}


back_prop_weight <- function(theta, x, y, drop_rate = 0.6){

  ### Primeiro, deve-se realizar o forward propagation
  ifelse(is.double(x), x <- as.matrix(x), x <- t(as.matrix(x)))

    # #gera mascaras
    # mask <- replicate(dim(x)[2],rbinom(4, 1, drop_rate))
    # #aplica mascaras
    # x <- x * mask[1:2,]

  # criar matrizes do WSIR
  W1 <- matrix(data = theta[1:4], nrow = 2)
  W2 <- matrix(data = theta[5:6], nrow = 2)
  scaled_W1 <- (drop_rate) * W1
  scaled_W2 <- (drop_rate) * W2

  b1 <- theta[7:8]
  b2 <- theta[9]
```

```
  # A agora é multiplicado pelo WS W1
  a <- matrix(data = rep(b1, ncol(x)), nrow = 2) + scaled_W1 %*% x
  h <- sigmoide(a)

  #aplica mascaras
    # h <- h * mask[3:4,]

  # gera yhat agora multiplicado pelo WS W2
  y_hat <- as.double(b2 + t(scaled_W2) %*% h)

  g <- -2*(y - y_hat)/length(y)
  grad_b2 <- sum(g)
  grad_W2 <- g %*% t(h)
  g <- W2 %*% g
  g <- g * derivada_sigmoide(a)
  grad_b1 <- rowSums(g)
  grad_W1 <- g %*% t(x)
  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(
    list(
      vetor_grad = vetor_grad,
      mse_cost = mse_cost(y, y_hat))
    )
}
```

A seguir são gerados os mesmos dados da lista 1.

```
# semente aleatoria indicada
set.seed(1.2024)

### Gerando dados "observados"
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))

# dados particionados conforme a lista 1
treino <- dados[1:80000, ]
val <- dados[80001:90000, ]
```

```r
teste <- dados[90001:nrow(dados), ]

# particoes de x
x_treino <- treino %>%
  select(x1.obs, x2.obs)
x_val <- val %>%
  select(x1.obs, x2.obs)
x_teste <- teste %>%
  select(x1.obs, x2.obs)

# particoes de y
y_treino <- treino$y
y_val <- val$y
y_teste <- teste$y
```

A seguir, calculamos o custo no conjunto de treinamento e registramos os valores do gradiente nas épocas. A inicialização é a mesma, em $\boldsymbol{\theta} = (0, \ldots, 0)$, com taxa de aprendizagem $\epsilon = 0.1$ e 100 iterações.

```r
epsilon <- 0.1
M <- 100

#lista de theta para receber os valores
theta_est <- list()
# Theta inicial
theta_est[[1]] <- rep(0, 9)
# inicializacao do vetor de custo
custo_treino <- numeric(M)


# Execução
for(i in 1:M) {
# Cálculo dos gradientes dos parâmetros
  grad <- back_prop_drop(theta = theta_est[[i]], x = x_treino, y = y_treino,
↪  drop_rate = 0.6)
  # Cálculo do custo de treino
  custo_treino[i] <- grad$mse_cost
  # Atualização dos parâmetros
  theta_est[[i+1]] <- theta_est[[i]] - epsilon*grad$vetor_grad
}

best_epoch <- which(custo_treino == min(custo_treino))
min_cost <- min(custo_treino)
best_grad <- theta_est[[best_epoch]]
```

O menor custo observado durante o treinamento foi de 165.7767 na época 92, conforme a Figura 1.

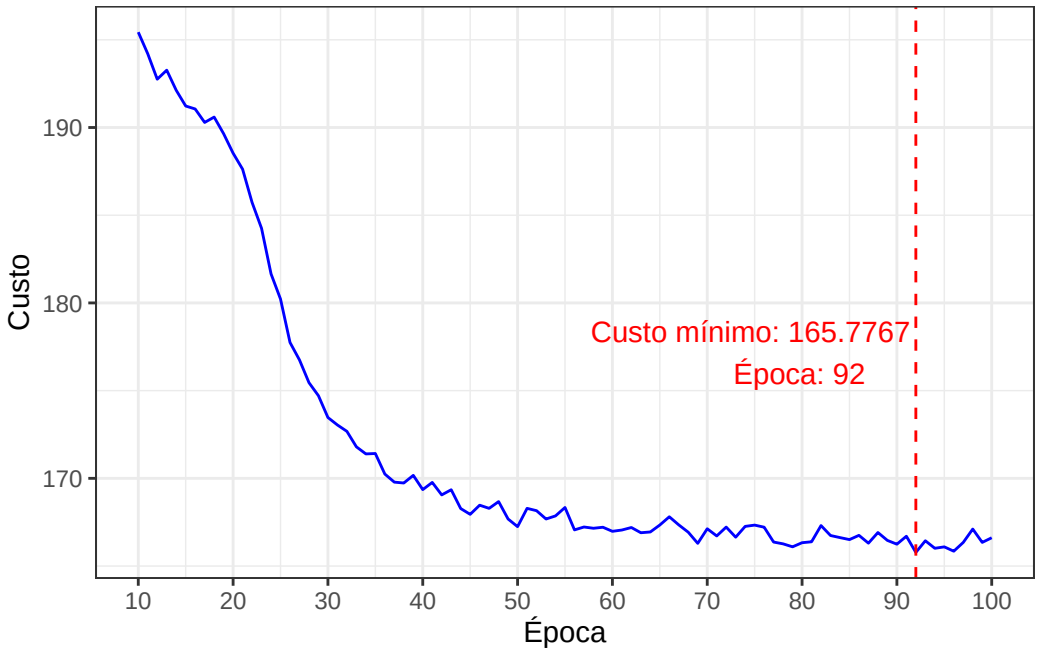

Figura 1: Custo no conjunto de treinamento

O vetor gradiente obtido na melhor época é

```
       w1        w2        w3        w4        w5        w6        b1        b2
 1.313836  1.316983 -5.631248 -5.639386  8.835846  8.874542 -2.348411 -2.356765
       b3
18.454321
```

1.2 Item b)

Considerando os pesos obtidos na Seção 1.1, para a primeira observação do conjunto de teste, gere 200 previsões $(\hat{y}_{1,1}, \ldots, \hat{y}_{1,200})$, uma para cada sub-rede amostrada aleatoriamente. Use as previsões para construir uma estimativa pontual e um intervalo de confiança para y_1 . Veja a Figura 7.6 do livro Deep Learning. Note que com esse procedimento, não é preciso assumir normalidade para os erros, como fizemos na Lista 1.

A seguir, são geradas 200 previsões para a primeira observação do conjunto de teste, com os pesos obtidos na Seção 1.1. A previsão pontual é a média das previsões e o intervalo de confiança é obtido empiricamente pelos quantis das previsões.

```r
n_pred <- 200
predictions <- numeric(n_pred)

for(i in 1:n_pred) {
  predictions[i] <- feed_forward_drop(theta = best_grad, x = x_teste[1,])
}
```

A estimativa pontual e o intervalo de confiança são dados a seguir:

```r
mean_pred <- mean(predictions)
quantile(predictions, c(0.025, 0.975))
```

```
    2.5%    97.5%
18.45432 24.18317
```

As previsões geradas estão representadas na Figura 2. A linha vermelha representa a média e as linhas azuis representam os limites do intervalo de confiança.

```r
data_plot_q1b <- tibble(
  pred = predictions
)

ggplot(data_plot_q1b) +
  geom_histogram(aes(x = pred), bins = 30) +
  geom_vline(xintercept = mean_pred, linetype = "dashed", color = "red") +
  geom_vline(xintercept = quantile(predictions, 0.025), linetype = "dashed", color =
  ↪  "blue") +
  geom_vline(xintercept = quantile(predictions, 0.975), linetype = "dashed", color =
  ↪  "blue") +
  scale_y_continuous(expand = c(0, 0)) +
  labs(
    x = "Previsões",
    y = "Frequência"
  ) +
  theme_bw()
```

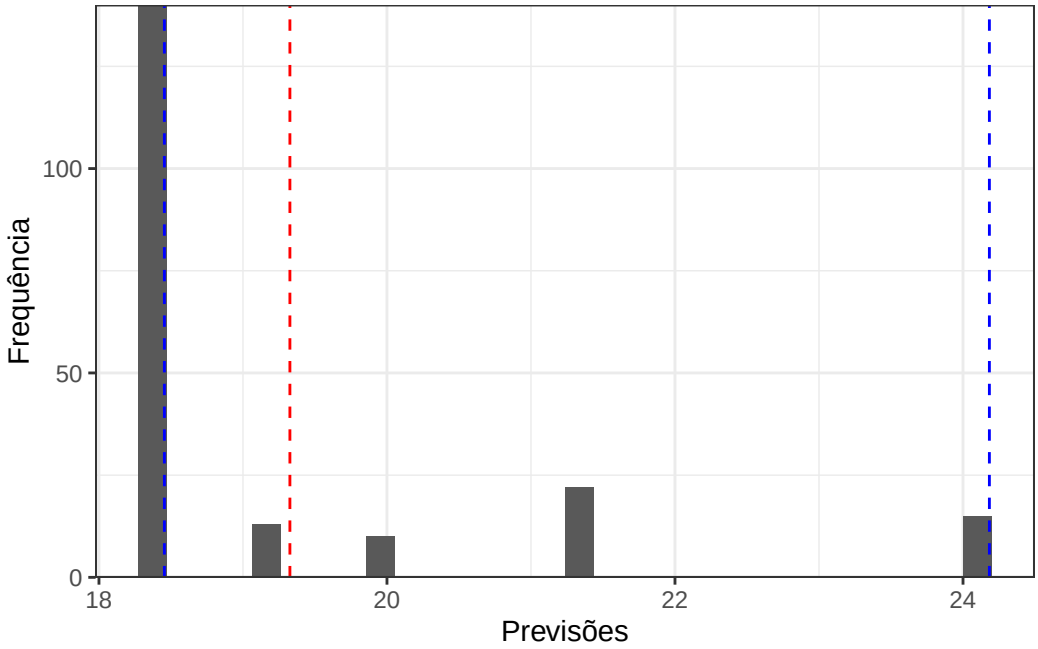

Figura 2: Previsões para a primeira observação do conjunto de teste

1.3 Item c)

Repita o item b) para gerar estimativas pontuais para cada observação do conjunto de testes.

Usando o mesmo vetor gradiente obtido na Seção 1.1, são geradas previsões para cada observação do conjunto de teste.

```
#ajuste do tamanho das predições
n_pred <- 200

#monta uma lista de previsões para cada observação
pred_matrix_drop <- replicate(n = nrow(x_teste), expr = rep(0, n_pred), simplify =
↪  FALSE)

for(j in 1:nrow(x_teste)) { # 1:10000 elementos do conjunto de teste
  for(i in 1:n_pred) { # 1:200 predições para cada elemento do conjunto de teste
    pred_matrix_drop[[j]][i] <- feed_forward_drop(theta = best_grad, x =
↪  x_teste[j,])
  }
}

means_q1c <- sapply(pred_matrix_drop, mean)
```

1.4 Item d)

Use a regra weight scaling inference rule (página 263 do livro Deep Learning) para gerar novas estimativas para as observações do conjunto de testes. Qual dos procedimentos (o do item c) ou o utilizado neste item) produziu melhores resultados? Considerando o tempo computacional de cada um, qual você escolheria nessa aplicação?

Primeiro são obtidas as previsões usando WSIR. A diferença entre esse método e o anterior é

$$\begin{aligned} \mathbf{W}_{(.)}^{\text{WSIR}} &= p\mathbf{W}_{(.)} \\ \mathbf{A} &= \mathbf{b}_1 + \mathbf{W}_{(.)}^{\text{WSIR}}\mathbf{X} \\ \hat{\mathbf{y}} &= \mathbf{H}\mathbf{W}_2^{\text{WSIR}} + b_3 \end{aligned}$$

Em seguida, recalibramos a rede com a mesma inicialização, mas usando WSIR. Em seguida, geramos novamente 200 previsões para cada observação do conjunto de teste com o melhor $\boldsymbol{\theta}_{\text{WSIR}}$.

```
epsilon <- 0.1
M <- 100

#lista de theta para receber os valores
theta_est_WSIR <- list()
# Theta inicial
theta_est_WSIR[[1]] <- rep(0, 9)
# inicializacao do vetor de custo
custo_treino_WSIR <- numeric(M)


# Execução
for(i in 1:M) {
# Cálculo dos gradientes dos parâmetros
  grad <- back_prop_weight(theta = theta_est_WSIR[[i]], x = x_treino, y = y_treino)
  # Cálculo do custo de treino
  custo_treino_WSIR[i] <- grad$mse_cost
  # Atualização dos parâmetros
  theta_est_WSIR[[i+1]] <- theta_est_WSIR[[i]] - epsilon*grad$vetor_grad
}

best_epoch_WSIR <- which(custo_treino_WSIR == min(custo_treino_WSIR))
min_cost_WSIR <- min(custo_treino_WSIR)
best_grad_WSIR <- theta_est_WSIR[[best_epoch_WSIR]]

#ajuste do tamanho das predições
```

```r
n_pred <- 200

#monta uma lista de previsões para cada observação
pred_matrix_WSIR <- replicate(n = nrow(x_teste), expr = rep(0, n_pred), simplify =
↪  FALSE)

for(j in 1:nrow(x_teste)) { # 1:10000 elementos do conjunto de teste
  for(i in 1:n_pred) { # 1:200 predições para cada elemento do conjunto de teste
    pred_matrix_WSIR[[j]][i] <- feed_forward_scale(theta = best_grad_WSIR, x =
↪  x_teste[j,])
  }
}

means_q1d <- sapply(pred_matrix_WSIR, mean)
```

Com base estritamente no MSE, o método drop-out produziu melhores resultados, como mostrado na Tabela 1.

Tabela 1: MSE no conjunto de teste

Metodo	MSE
Dropout	143.48
WSIR	214.25

2 Questão 2

A questão 2 foi realizada no Google Collab. A maior parte dos resultados desta seção foram obtidos a partir de uma sessão na plataforma.

2.1 Item a)

Ajuste a rede neural especificada na Lista 1 usando o *Keras*. Compare com sua implementação (Lista 1, item e) quanto ao tempo computacional e ao custo obtido no conjunto de validação. Use o mesmo algoritmo de otimização (*full gradient descent*) e ponto de partida.

Primeiro foram instalados os pacotes necessários:

```r
install.packages("pacman")
pacman::p_load("tidyverse", "keras", "tensorflow", "reticulate", "rsample")
```

Os dados utilizados foram iguais aos utilizados na sessão local:

```
### Configurações iniciais ----
# semente aleatoria indicada
set.seed(1.2024)

### Gerando dados "observados"
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))

# dados particionados conforme a lista 1
treino <- dados[1:80000, ]
val <- dados[80001:90000, ]
teste <- dados[90001:nrow(dados), ]

# particoes de x
x_treino <- treino %>%
  select(x1.obs, x2.obs) %>%
  as.matrix()
x_val <- val %>%
  select(x1.obs, x2.obs) %>%
  as.matrix()
x_teste <- teste %>%
  select(x1.obs, x2.obs) %>%
  as.matrix()

# particoes de y
y_treino <- treino$y
y_val <- val$y
y_teste <- teste$y
```

Em seguida, o modelo é configurado. O full gradient descent, de acordo com a documentação, é obtido com as configurações `lr = 0.1, momentum = 0, weight_decay = FALSE)` do otimizador. A função de ativação é definida como sigmóide e `kernel_initializer='zeros', bias_initializer='zeros'` indicam a inicialização dos pesos e dos viéses como zero. Finalmente, a métrica da função perda é definida como MSE.

```
# definicao do modelo
model <- keras_model_sequential() %>%
  layer_dense(units = 2, input_shape = c(2), activation = 'sigmoid',
  ↪  kernel_initializer='zeros', bias_initializer='zeros') %>%
  layer_dense(units = 1)

# compilar o modelo com full gradient descent
model %>% compile(
  optimizer = optimizer_sgd(lr = 0.1, momentum = 0, weight_decay = FALSE),  #
  ↪  Gradient descent with learning rate 0.01
```

```
  loss = 'mse',
  metrics = 'mse'
)
```

O modelo é entao treinado com 100 épocas e o *batch* é definido como o tamanho total do banco de dados. As partições específicas de treino e validação são definidas.

```
history <- model %>% fit(
  x = x_treino,
  y = y_treino,
  epochs = 100,
  batch_size = nrow(x_treino),
  validation_data = list(x_val, y_val),
  verbose = 2
)
```

Finalmente, obtém-se os pesos e o custo associado a esse modelo no conjunto de validação.

```
evaluation <- model %>% evaluate(x_val, y_val, verbose = 0)

mse_loss <- evaluation[['loss']]

weights_model1 <- model %>% get_weights()
```

O tempo de treino foi de 11 segundos e o menor custo observado foi de 142.8076477. O tempo de treino é muito próximo ao item e) da lista 1, assim como o custo. Os pesos obtidos foram:

```
[[1]]
           [,1]       [,2]
[1,] -0.8902205 -0.9162572
[2,] -2.0196459 -2.0407729

[[2]]
[1] 2.463278 2.528810

[[3]]
         [,1]
[1,] 8.927702
[2,] 9.146015

[[4]]
[1] 8.982577
```

2.2 Item b)

Ajuste a rede neural mais precisa (medida pelo MSE calculado sobre o conjunto de validação) que conseguir, com a arquitetura que quiser. Use todos os artifícios de regularização que desejar (*weight decay*, *Bagging*, *droupout*, *Early stopping*). Reporte a precisão obtida para essa rede no conjunto de validação.

Optou-se por uma arquitetura de rede com 3 camadas intermediárias *fully connected* com 128 unidades cada, função de ativação ReLU e inicialização aleatória dos pesos. A regularização *dropout* foi aplicada com probabilidade de 0,5. O otimizador escolhido foi o *Adam* com taxa de aprendizado de 0,01 e *minibatch* de tamanho 32 para 100 épocas. O modelo foi treinado com 1000 épocas e *batch* de tamanho 32.

```
model2 <- keras_model_sequential() %>%
  layer_dense(units = 128, input_shape = c(2), activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 1)

model2 %>% compile(
  optimizer = optimizer_adam(lr = 0.01),
  loss = 'mse',
  metrics = 'mse'
)

history2 <- model2 %>% fit(
  x = x_treino,
  y = y_treino,
  epochs = 100,
  batch_size = 32,
  validation_data = list(x_val, y_val),
  verbose = 2
)
```

O menor custo obtido no conjunto de validação, conforme indicado no bloco a seguir, foi de 5.2935634. O tempo de treino dessa rede foi de 14 minutos.

```
evaluation2 <- model2 %>% evaluate(x_val, y_val, verbose = 0)

mse_loss2 <- evaluation2[['loss']]

weights_model2 <- model2 %>% get_weights()
```

2.3 Item c)

Refaça o item h) da Lista 1 para essa nova rede. Comente os resultados.

Este mesmo gráfico, para a lista anterior, apresentava concentrações em valores extremos e com variância diferente em diferentes posições de y. Em comparação, a Figura 3 apresenta uma distribuição mais homocedástica e com valores mais próximos de y, evidenciado pela nuvem de pontos muito próximos à reta $y = \hat{y}$. É possível que a rede neural tenha sido capaz de capturar a relação entre $\mathbf{x}$ e y de forma mais precisa.

```
tibble(
  yhat = nova_y_val,
  y = y_val) %>%
  ggplot(aes(x = y, y = yhat)) +
  geom_point(alpha = 0.3)+
  geom_abline(intercept = 0, slope = 1, color = "red")+
  xlab(TeX("$y$")) + ylab(TeX("$\\hat{y}$"))+
  theme_bw()
  theme(axis.title.y = element_text(angle = 0, vjust = 0.5))
```

```
List of 1
 $ axis.title.y:List of 11
  ..$ family       : NULL
  ..$ face         : NULL
  ..$ colour       : NULL
  ..$ size         : NULL
  ..$ hjust        : NULL
  ..$ vjust        : num 0.5
  ..$ angle        : num 0
  ..$ lineheight   : NULL
  ..$ margin       : NULL
  ..$ debug        : NULL
  ..$ inherit.blank: logi FALSE
  ..- attr(*, "class")= chr [1:2] "element_text" "element"
 - attr(*, "class")= chr [1:2] "theme" "gg"
 - attr(*, "complete")= logi FALSE
 - attr(*, "validate")= logi TRUE
```

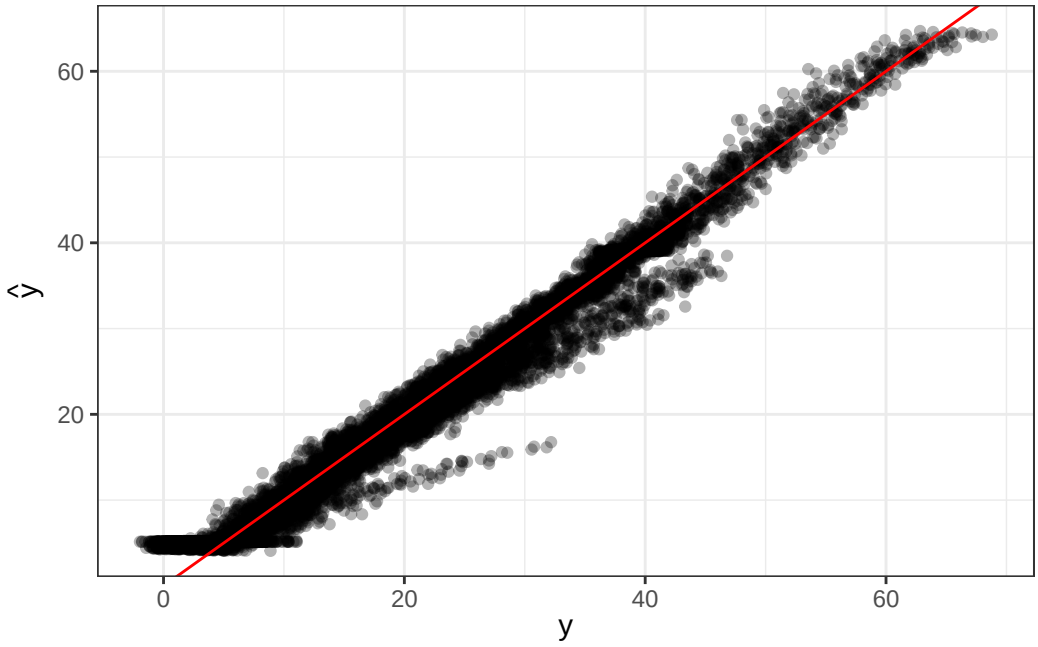

Figura 3: Previsões para o conjunto de validação com a rede neural do item 2b.

2.4 Item d)

Use a função de previsão do Keras para prever o valor da variável resposta $\hat{y} = f(x_1 = 1, x_2 = 1; \theta)$, para θ) definido de acordo com a rede ajustada. (Veja o item a) da Lista 1).

O valor obtido é de 14.141758. Em comparação, na lista 1 item a foi obtido $\hat{y} = 0{,}215$.

```
newdata <- data.frame(x1 = 1, x2 = 1) %>% as.matrix()

yhat_d <- model2 %>% predict(newdata)

yhat_d
```

2.5 Item e)

Neste exemplo meramente didático, conhecemos a superfície que estamos estimando. Apresente, lado a lado, a Figura 1 da Lista 1 e a superfície estimada pela sua rede neural. Para tanto, basta trocar a variável mu pelos valores preditos pela rede. Comente os resultados.

Na Figura 4 é exibido o gráfico da lista 1 e a superfície estimada pela rede neural. Evidentemente a superfície de resposta não é muito interpretável, exceto por uma possível homogeneidade de respostas. A representação interna da rede, no entanto, parece capturar a bem a relação entre x_1, x_2 e y, de modo que pode ser apenas um padrão não interpretável para o leitor.

```
# figura da lista 1
n <- 100
x1 <- seq(-3, 3, length.out=n)
x2 <- seq(-3, 3, length.out=n)

dados.superficie <- as_tibble(expand.grid(x1, x2)) %>%
  rename_all(~ c("x1", "x2")) %>%
  mutate(mu = nova_y_val)

plot_superficie<- ggplot(dados.superficie, aes(x=x1, y=x2)) +
  geom_point(aes(colour=mu), size=2, shape=15) +
  coord_cartesian(expand=F) +
  scale_colour_gradient2(low="red", mid="white", high="blue",# midpoint=0,
                         name=TeX("$\\hat{Y}|X_1, X_2$")) +
  xlab(TeX("$X_1$")) + ylab(TeX("$X_2$")) +
  theme(legend.position = "bottom",
        axis.title.y = element_text(angle = 0, vjust = 0.5))

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",
        axis.title.y = element_text(angle = 0, vjust = 0.5))

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

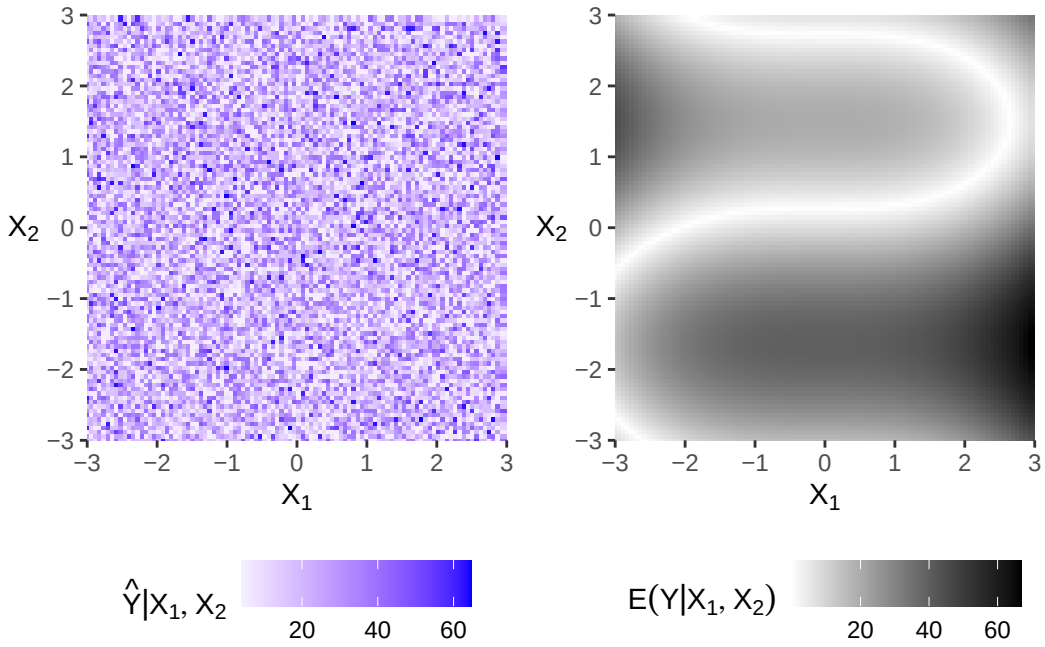

Figura 4: Comparação entre a superfície estimada pela rede neural e a superfície real.

Para facilitar a interpretação da figura anterior, apresenta-se a Figura 5 em que é exibida a superfície dos resíduos. É notável que a maior partedos resíduos parece estar muito próxima de zero, o que corrobora a hipótese de que a rede neural foi capaz de capturar a relação desejada.

```
dados.residuos <- as_tibble(expand.grid(x1, x2)) %>%
  rename_all(~ c("x1", "x2")) %>%
  mutate(mu = nova_y_val-y_val)

ggplot(dados.residuos, aes(x=x1, y=x2)) +
  geom_point(aes(colour=mu), size=2, shape=15) +
  coord_cartesian(expand=F) +
  scale_colour_gradient2(low="red", mid="white", high="blue", midpoint=0,
                         name=TeX("$\\hat{e}|X_1, X_2$")) +
  xlab(TeX("$X_1$")) + ylab(TeX("$X_2$")) +
  theme(legend.position = "bottom",
        axis.title.y = element_text(angle = 0, vjust = 0.5))
```

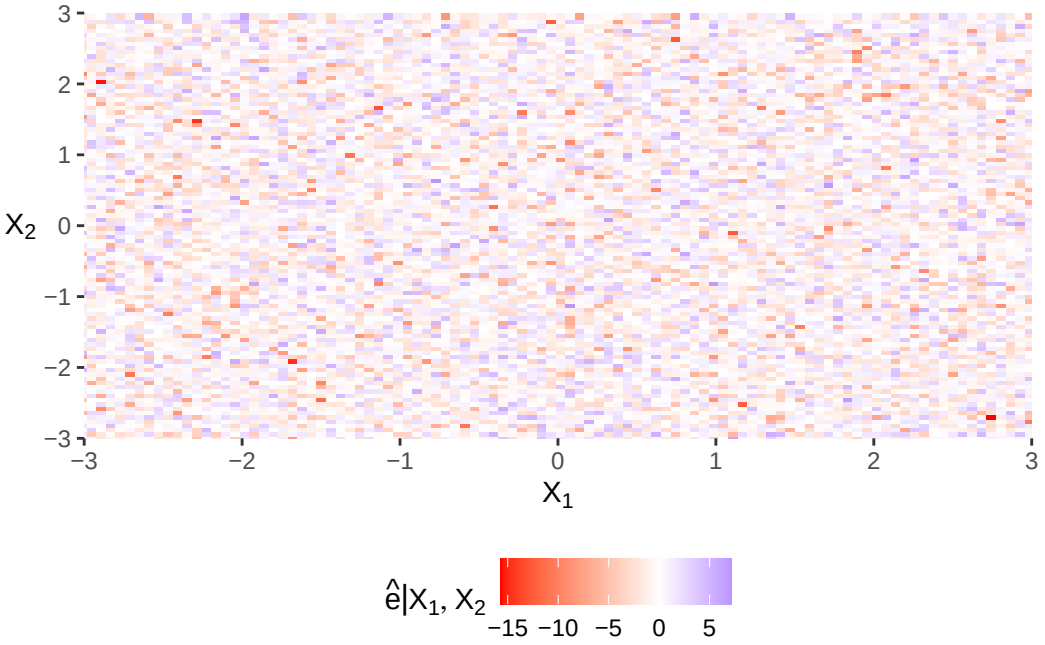

Figura 5: Superfície de resíduos estimada pela rede neural.

2.6 Item f)

Construa uma nova rede, agora ajustada sobre os valores previstos (ao invés dos valores observados de y) para cada observação dos conjuntos de treinamento e validação. Use a arquitetura mais parcimoniosa que conseguir, sem comprometer substancialmente o poder de previsão da rede (quando comparada à obtida no item 2b). Cite um possível uso para essa nova rede.

Primeiro são gerados os valores previstos para cada observação dos conjuntos de treinamento e validação.

```
novo_y_treino <- model2 %>% predict(x_treino)
nova_y_val <- model2 %>% predict(x_val)
```

Em seguida um novo modelo é treinado, parcimonioso em relação ao anterior. A mudança ocorre somente na quantidade de unidades nas camadas intermediárias, que agora é de 30 em cada uma das 3 camadas.

```
model3 <- keras_model_sequential() %>%
  layer_dense(units = 30, input_shape = c(2), activation = 'relu') %>%
```

```
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 30, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 30, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 1)

model3 %>% compile(
  optimizer = optimizer_adam(lr = 0.01),
  loss = 'mse',
  metrics = 'mse'
)

history3 <- model3 %>% fit(
  x = x_treino,
  y = novo_y_treino,
  epochs = 100,
  batch_size = 32,
  validation_data = list(x_val, nova_y_val),
  verbose = 2
)

evaluation3 <- model3 %>% evaluate(x_val, nova_y_val, verbose = 0)
mse_loss3 <- evaluation3[['loss']]
```

O tempo de treino foi de 12 minutos e o menor custo observado foi de 31.8574009. É possível que o as duas etapas de treinamento façam a função de suavização de ruídos.