Lista 2: Dropout e Keras

César A. Galvão - 190011572

Questão 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 $\mathbf{h}$.

```
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))
}

# para as epochs, é necessário rodar um for
back_prop_drop <- function(theta, x, y, p_masks = 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, p_masks))
    #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
  # 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))
    )
}
```

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, ]
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.

```
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, p_masks = 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.

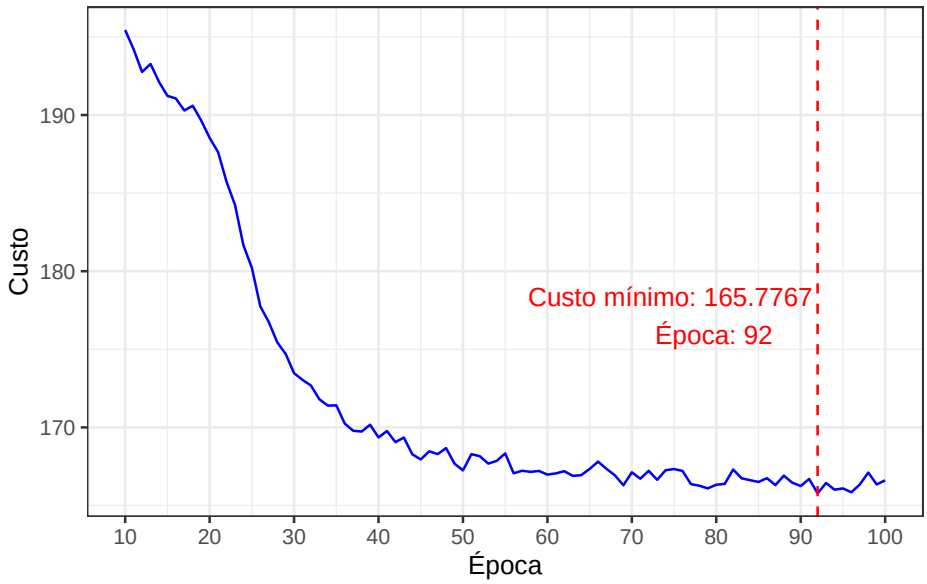

Figura 1: Custo no conjunto de treinamento

O vetor gradiente obtido na melhor época é

```
best_grad
```

```
       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
```