Universidade de Brasília

DEPARTAMENTO DE ESTATÍSTICA

25 de maio de 2023

Lista 2: *Dropout* e *Keras*.

Prof. Guilherme Rodrigues

Redes Neurais Profundas

Tópicos especiais em Estatística 1

Nesta lista utilizaremos o pacote computacional *Keras* (ou outro de sua preferência – *PyTorch*, *TensorFlow*, *Theano*, *H2O*, *Caffe*) para ajustar redes neurais profundas. Considere os dados e os modelos descritos na Lista 1 para responder os itens a seguir.

Questão 1)

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. Reporte o custo dessa rede.

Inicialmente, vamos adaptar as funções do gabarito da Lista 1 para para implementar o dropout.

```
### DADOS ----
rm(list=ls())
set.seed(1.2023)
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))


# FUNÇÕES AUXILIARES ----

sigmoide <- function(x) { return(1/(1+exp(-x)))
}

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

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

# SEPARAÇÃO TREINO E TESTE ----

corte <- 80000
treino <- dados[1:corte,]
teste <- dados[(corte+1):nrow(dados),]
x_treino <- treino %>%
  select(x1.obs, x2.obs)
x_teste  <- teste %>%
  select(x1.obs, x2.obs)
y_treino <- treino$y
y_teste  <- teste$y

# ADAPTAÇÃO DO BACK- PROPAGATION DO GABARITO DA LISTA 1



foward_prop <- function(theta, x, m1=1, m2=1, scaling=1){
   # os parâmetrsos tem a possibilidade de inlcuir mascaras geradas anteriormente e de fazer weight


  theta <- theta * scaling #aplica o scaling, que é por defaul 1 (sem weight scaling)
  # feed-foward
  x <- t(as.matrix(x))

  W1 <- matrix(data = theta[1:4], nrow = 2)
```

```r
  W2 <- matrix(data = theta[5:6], nrow = 2)
  b1 <- theta[7:8]
  b2 <- theta[9]

  # possibilidade de inlcuir mascara da camada de entrada.
  x <- m1 * x
  a <- matrix(data = rep(b1, ncol(x)), nrow = 2) + W1 %*% x
  h <- sigmoide(a)

   # possibilidade de inlcuir mascara passada como parametero ma camada de intermediaria

  h <- m2 * h
  y_hat <- as.double(b2 + t(W2) %*% h)

  return(y_hat)
}

back_prop <- function(theta, x, y){

  # feed-foward
  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]

  # mascara da camada de entrada.
  m1 <- matrix(rbinom(n=(2*ncol(x)), size=1, prob = 0.6),
               nrow = nrow(x) ,ncol = ncol(x))
  x <- m1 * x
  a <- matrix(data = rep(b1, ncol(x)), nrow = 2) + W1 %*% x
  h <- sigmoide(a)

  # mascara da camada intermediaria
  m2 <- matrix(rbinom(n=(2*ncol(x)), size=1, prob = 0.6), nrow = nrow(x) ,ncol = ncol(x))
  h <- m2 * h
  y_hat <- as.double(b2 + t(W2) %*% h)


  # derivadas do back-propagation
  g <- -2*(y - y_hat)/length(y)
  grad_b2 <- sum(g)
  grad_W2 <- g %*% t(h)

  g <- W2 %*% g * m2  # mudança em relação ao original

  g <- g * derivada_sigmoide(a)
  grad_b1 <- rowSums(g)
  grad_W1 <- g %*% t(x)
  g <- W1 %*% g


  # saída
  vetor_grad <- c(grad_W1, grad_W2, grad_b1, grad_b2)
  names(vetor_grad) <- c(paste0("w", 1:6), paste0("b", 1:3))
  grad <- list(
```

```
    grad=vetor_grad,
    y_hat=y_hat) # para usar a previsão dessa rede (como se fosse 1 simulacao de MC)

  return(grad)
}


# Alguns parametros ----

# Taxa de aprendizagem
epsilon <- 0.1
# Número de iterações
M <- 100
theta_est <- list()
# Theta inicial
theta_est[[1]] <- rep(0, 9)
custo_treino <- custo_teste <- numeric(M)
```

As 100 iterações são calculadas a seguir. Deixei como comentário a possibilidade de incluir um mini-batch.

```
# Execução


#batch <- 800 # Se quisesse fazer por mini-batch

for(i in 1:M) {

  # Se quisesse fazer por mini-batch

  #x_batch <- x_treino[((i-1) * batch  + 1) : (i * batch ),] # se quisesse fazer por mini-batch
  #y_batch <- y_treino[((i-1) * batch  + 1) : (i * batch )]

  #x_batch <- x_treino %>% sample_n(800)
  #y_batch <- y_treino %>% sample(800)

  # Cálculo dos gradientes dos parâmetros

  grad <- back_prop(theta = theta_est[[i]],
                    x = x_treino , y = y_treino)
  custo_treino[i] <- mse_cost(y_treino, grad$y_hat)  # está usando o y_hat de  1 simulação de MC
  theta_est[[i+1]] <- theta_est[[i]] - epsilon*grad$grad

}


# pegar a iteração que aconteceu isso
min_custo_treino <- min(custo_treino)



qual1 <- which(custo_treino==min(custo_treino))
pesos <- theta_est[[qual1]]
```

Estou assumindo que o que a questão está pedindo é para monitorar o MSE na própria implementação. Que pelo que entendi seria o equivalente a estar fazendo uma previsão com 1 amostra de monte-carlo. Pelo que entendi na aula essa questão não necessitava de implementar toda a situação de comparar treino e teste etc.

b) Considerando os pesos obtidos em a), 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.

Pela implementação acima, em que se monitorou o MSE do treino, o menor MSE ocorreu na iteração 86 e os pesos foram:

```
pesos
```

```
##        w1        w2        w3        w4        w5        w6        b1        b2
##  1.143933  1.165732 -4.681277 -4.673066  9.337880  9.352390 -2.672990 -2.671047
##        b3
## 18.617916
```

Nota-se que houve uma pequena quebra de simetria, mas os pesos ainda estão em "pares".

Por algum motivo, nesse caso eu achei melhor gerar as máscaras previamente. Também fiquei na dúvida se deveria ter salvado as mascaras usadas na questão anterior para utilizar aqui novamente. Mas eu só as gerei aleatoriamente mesmo de novo. E então forneci essas mascaras como parâmetros na função do foward_prop.

```
# funcao para gerar as mascaras
gera_mascaras <- function(x){
  x <- t(x)
  linhas <- nrow(x)
  colunas <- ncol(x)
  M1 <- matrix(rbinom(n=(2*ncol(x)), size=1, prob = 0.6),
               nrow = linhas ,ncol = colunas)
  M2 <- matrix(rbinom(n=(2*ncol(x)), size=1, prob = 0.6),
               nrow = linhas ,ncol = colunas)
 rbind(M1, M2)
}


mascaras_uma_obs <- map(1:200, ~{
  gera_mascaras(x_teste[1,]) #apenas a primeira observação
})



uma_obs <- map_dbl(1:200, ~{

  foward_prop(theta=theta_est[[qual1]],
              x=x_teste[1,],
              m1=mascaras_uma_obs[[.]][1:2,],
              m2=mascaras_uma_obs[[.]][3:4,])
  } )


# calculando as medidas
media <- mean(uma_obs)
q1 <-quantile(uma_obs, c(0.025, 0.975))[1]
q3 <-quantile(uma_obs, c(0.025, 0.975))[2]
```

Para essa observação, a estimativa pontual foi de 19.6980011 e o IC foi de (18.6179157 ; 21.7326415)

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

Utilizando a função de feed-foward gerada anteriormente, gera-se as 200 mascaras, que são passadad

como parâmetros para a função do feed-foward.

```
teste_obs <- map(1:200, ~{
  gera_mascaras(x_teste)
})

todas <- map_dfc(1:200, ~{
prev <- foward_prop(theta=theta_est[[qual1]],  x=x_teste,
             m1=mascaras_uma_obs[[.]][1:2,],
             m2=mascaras_uma_obs[[.]][3:4,])})

IC_treino <- todas %>%
  rowwise() %>%
  summarise(
    q1=quantile(c_across(`...1`:`...200`), 0.025), # os nomes das colunas ficaram esquisitos
    media=mean(c_across(`...1`:`...200`)),
    q3=quantile(c_across(`...1`:`...200`), 0.975))
```

```
mse_MC <- IC_treino %>%
  mutate(obs=y_teste,
         dif=(obs-media)^2) %>%
  pull(dif) %>%
  mean()
```

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?

No caso das previsões geradas com as amostras de Monte Carlo o MSE foi de 141.2953345.

No caso do MSE utilizando o weight-scaling , esse valor é calculado uma única vez:

```
# Usando as fu
mse_WS <- mse_cost(y_teste, foward_prop(theta=pesos, x=x_teste, scaling=0.6))
```

O resultado é de 165.6979446.

Questão 2)

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 teste. Use o mesmo algoritmo de otimização (*full gradient descent*) e ponto de partida.

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.

Considerando a rede ajustada no item b), responda os itens a seguir.

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

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).

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.

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.