Reporte práctica 12

Red neuronal

A 31 de Octubre de 2017

Práctica

En esta práctica se analizó un código de red neuronal, la cual sirve para imitar el aprendizaje de alguna máquina para reconocer dígitos.

Simulación y Resultados

En la práctica se paralelizó el código original, solo se paralelizaron las partes donde convenía.

Para la paralelización se activaron las librerías *doParallel* y la librería *foreach* las cuales permiten la paralelización del código, y se creó un *cluster* con dos núcleos para la ejecución del código.

```
1. suppressMessages(library(doParallel))
2. suppressMessages(library(foreach))
3. registerDoParallel(makeCluster(2))
```

A continuación se muestran las funciones realizadas para la paralelización del *for* de prueba.

```
1. neu <- function(){
2. #for (t in 1:t1) { # prueba
3.   d <- sample(0:tope, 1)
4.   pixeles <- runif(dim) < modelos[d + 1,] # fila 1 contiene el cero,
   etc.
5.   correcto <- binario(d, n)
6.   salida <- rep(FALSE, n)
7.   for (i in 1:n) { # paralelizar
8.     w <- neuronas[i,]
9.     deseada <- correcto[i]
10.          resultado <- sum(w * pixeles) >= 0
11.          salida[i] <- resultado
12.        }
13.        r <- min(decimal(salida, n), k) # todos los no-existentes van
   al final
14.        return(r == d)
15.      }
16.
17.
18.      contadores <- foreach(t = 1:t1, .combine = c) %dopar% neu()
19.      stopImplicitCluster()
20.      con <- (sum(contadores)/t1)*100
```

En esta parte del código se creó la función *obj* y para realizar los cálculos y la evaluación de los resultados, esta función tiene su *foreach* para guardar los resultados.

Para obtener un diagrama caja-bigotes del tiempo de ejecución de los códigos original y modificado contra el tiempo, se utilizó la función *sys.time()* para guardar el tiempo que dura la ejecución de los dos códigos y se usó el código siguiente:

```
1. datos <- data.frame()
2.
3. for(i in 1:3){
4.   for (t1 in seq(500, 1000, 100)) {
5.     source('C:/Users/Pablo/Desktop/sim/p12/p12_3mod1.R',
   encoding = 'UTF-8')
6.     to <- cbind("O", t1, t)
7.     source('C:/Users/Pablo/Desktop/sim/p12/p12_3mod2.R',
   encoding = 'UTF-8')
8.     tm <- cbind("M", t1, t)
9.     datos <- rbind(datos, to, tm)
10.        }
11.      }
12.
13.      save.image(file = "con.RData")
14.
15.      names(datos) <- c("Tipo","t1", "Tiempo")
16.      datos$t1 <- as.numeric(levels(datos$t1))[datos$t1]
17.      datos$Tiempo <- as.numeric(levels(datos$Tiempo))[datos$Tiempo]
18.      datos$Tipo <- as.factor(datos$Tipo)
19.      png("tiempos2.png", width = 600, height = 800, pointsize = 15)
20.      boxplot(Tiempo ~ Tipo * t1, data = datos, col = ("grey"),
   border = c("orange", "green"), xlab = "Pruebas", ylab = "Tiempo (s)")
21.      legend("topleft", inset = 0.02, c("C. Original", "C.
   Modificado"), fill = c("orange", "green"), horiz=TRUE, cex=0.8,
   box.lty = 0)
22.      graphics.off
```

En donde se utilizaron dos *data.frame* para guardar los resultados, un *for* para poder hacer varias ejecuciones de los códigos, los cuáles fueron añadidos al código con la función *source*, y para generar el diagrama caja-bigotes se utilizó el comando *boxplot*.

En la figura 1 se muestra el diagrama caja-bigotes del tiempo de ejecución de los códigos original y modificado, en éste se puede observar la diferencia del tiempo de ejecución para distintos valores de *t1*, estos se usaron para distinguir cuando el cambio el tiempo de ejecución es significativo. Se puede observar además que el cambio en el tiempo de ejecución es significativo a partir de 500 pruebas donde el tiempo de ejecución para el código paralelizado es menor que el tiempo de ejecución del código original.

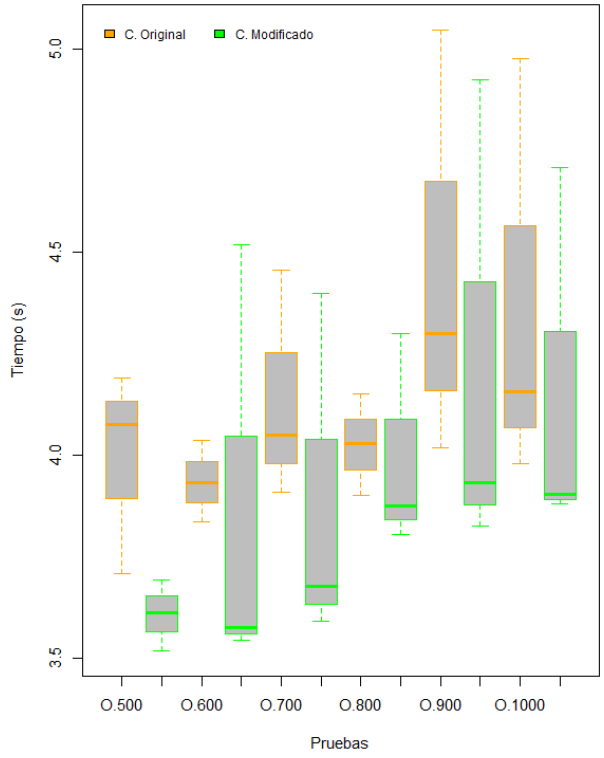

Fig. 1. Diagrama caja-bigotes de tiempo de ejecución de códigos original y modificado.

A continuación se presenta el código utilizado para obtener el porcentaje de resultados correctos de los códigos y su comparación.

```
1. datos <- data.frame()
2.
3. for(i in 1:3){
4.   for (t1 in seq(500, 1000, 100)) {
5.     source('C:/Users/Pablo/Desktop/sim/p12/p12_3mod1.R',
   encoding = 'UTF-8')
6.     to <- cbind("O", t1, con)
7.     source('C:/Users/Pablo/Desktop/sim/p12/p12_3mod2.R',
   encoding = 'UTF-8')
8.     tm <- cbind("M", t1, con)
9.     datos <- rbind(datos, to, tm)
10.        }
11.      }
12.
13.      save.image(file = "con.RData")
14.
15.      names(datos) <- c("Tipo","t1", "Porcentajes")
16.      datos$t1 <- as.numeric(levels(datos$t1))[datos$t1]
17.      datos$Porcentajes <- as.numeric(levels(datos$Porcentajes))[datos$
   Porcentajes]
```

```
18.       datos$Tipo <- as.factor(datos$Tipo)
19.       png("tiempos2.png", width = 600, height = 800, pointsize = 15)
20.       boxplot(Tiempo ~ Tipo * t1, data = datos, col = ("grey"),
    border = c("orange", "green"), xlab = "Pruebas", ylab = "Tiempo (s)")
21.       legend("topleft", inset = 0.02, c("C. Original", "C.
    Modificado"), fill = c("orange", "green"), horiz=TRUE, cex=0.8,
    box.lty = 0)
22.       graphics.off
```

Este código es similar al código donde se graficaron los diagramas caja-bigotes. En éste se realizaron pequeñas modificaciones para poder graficar los porcentajes de los resultados correctos.

En la figura 2 se muestran los diagramas caja-bigote para diferentes pruebas en función del porcentaje del número de resultados correctos. En estos diagramas se puede observar como los porcentajes de los resultados correctos son mayores para las pruebas con menores valores y que al aumentar el número de pruebas el porcentaje de resultados correctos para la prueba paralelizada incrementa en un punto en el cual ya se vuelve significativo el incremento del porcentaje de resultados correctos. Esto se puede deber a la existencia de un mayor número de pruebas los que hace que el código aprenda y se vulva más fácil identificar los dígitos correctos o cometer menos errores.

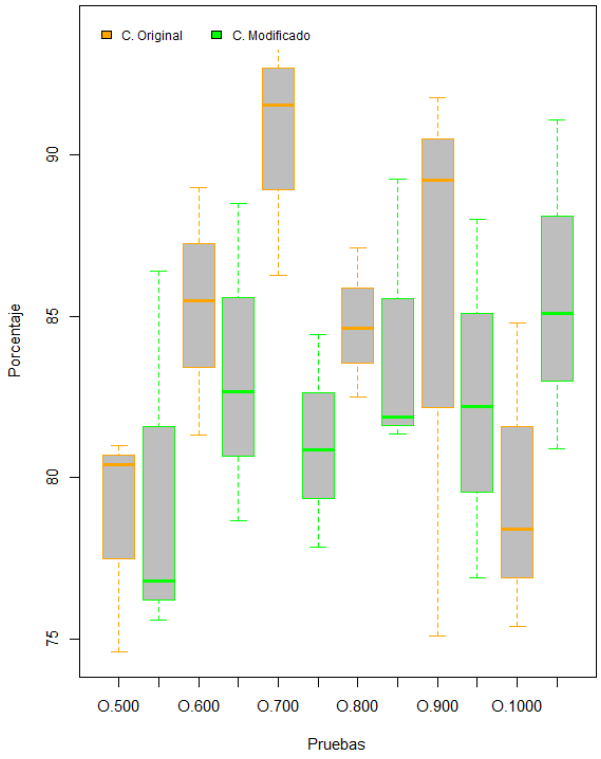

Fig. 2. Diagramas caja-bigote de pruebas contra el porcentaje de resultados correctos.

Conclusiones

Se puede concluir que la paralelización del código ayuda a disminuir los tiempos de ejecución del mismo, sobre todo al incrementar el número de pruebas, y para obtener un cambio significativo en los tiempos de ejecución el número de pruebas debe ser mayor a 500. Muchas veces la paralelización al no ser ejecutada correctamente puede perjudicar los tiempos de ejecución. También se puede concluir que el número de pruebas ayuda corregir los errores en la red neuronal.

Reto 1

El primer reto consiste en estudiar el desempeño de la red neuronal para los dígitos en función de las probabilidades asignadas a la generación de los dígitos

A continuación se muestra el código que se utilizó para realizar el reto 1.

```
setwd("C:/Users/Pablo/Desktop/sim/p12")
t1 <- 300
resultados <- data.frame()

suppressMessages(library(doParallel))
suppressMessages(library(foreach))
registerDoParallel(makeCluster(2))

for (negro in seq(0.05, 0.80, 0.20)){
        for (gris in seq(0.05, 0.80, 0.20)){
          for (blanco in seq(0.05, 0.80, 0.20)){

            source('C:/Users/Pablo/Desktop/sim/p12/p12_3mcalor.R',
   encoding = 'UTF-8')

            probabilidad <- cbind(negro, gris, blanco, con)
            print(probabilidad)
            resultados<-rbind(resultados, probabilidad)
          }
        }
      }
      stopImplicitCluster()

      save.image(file = "mcalor.RData")

      colnames(resultados) <- c("Negro","Gris","Blanco","Porcentaje")

      library(ggplot2)
      ggplot(resultados, aes(Blanco, Gris)) +
        geom_raster(aes(fill = Porcentaje)) +
        scale_fill_gradient(low = "yellow", high = "orange")
      ggsave("calorgb.png")

      ggplot(resultados, aes(Negro, Gris)) +
        geom_raster(aes(fill = Porcentaje)) +
        scale_fill_gradient(low = "yellow", high = "orange")
      ggsave("calorgn.png")
```

```
37.
38.       ggplot(resultados, aes(Negro, Blanco)) +
39.         geom_raster(aes(fill = Porcentaje)) +
40.         scale_fill_gradient(low = "yellow", high = "orange")
41.       ggsave("calornb.png")
```

En esta parte de código una vector *gsetwd* para seleccionar el lugar donde se guardarán los datos de nuestro *data.frame* llamado *resultados*. Se creó una varaiable *t1* para poder utilizar el numero de pruebas dentro de este código, posteriormente se activaron las librerías *doParallel* y *foreach* para evitar problemas con la paralelización del código original. Del código original se eliminaron las probabilidades de los colores de los pixeles y se agregaron en este código para variar las las probabilidades. Se utilizó un comando *source* para llamar el código paralelizado y hacer los cálculos de la red neuronal. Para la graficación se utilizó el paquete *ggplot2* para generar un mapa de calor de las distintas probabilidades.[1]

En la figura 3 se muestra el mapa de calor de las probabilidades gris y blanco donde se observa que el porcentaje de resultados correctos es menor al 10 %. Esto se debe al incremento de las probabilidades de pixeles de color gris donde este aumento en la probabilidad de tener pixeles de color gris aumenta las probabilidades de tener errores en el código que genera los dígitos.

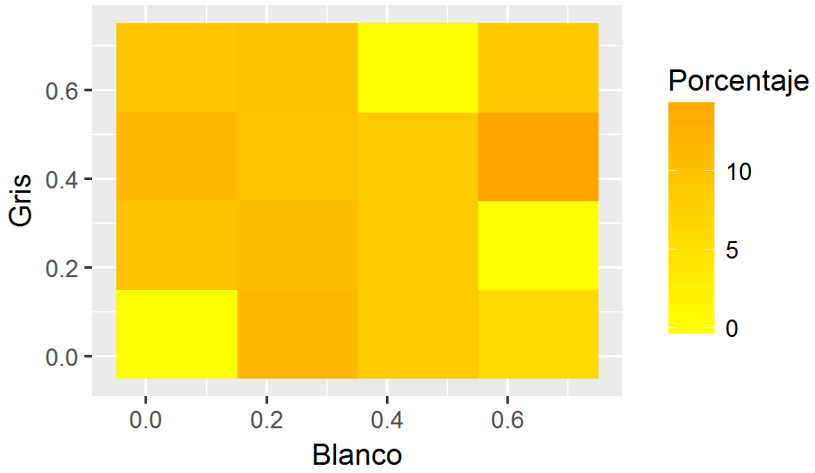

Fig. 3. Mapa de calor para las probabilidades de gris y blanco.

En la figura 4 se muestra el mapa de calor de las probabilidades gris y negro, aquí se observa el incremento del porcentaje de los resultados correctos ya que al disminuir la probabilidad de obtener pixeles de color gris, debido a esto la red neuronal puede identificar más fácilmente los dígitos del código binario, y esto se representa en el mapa de calor donde las zonas oscuras representan las probabilidades de obtener resultados correctos en función de las probabilidades de pixeles de color negro y gris.

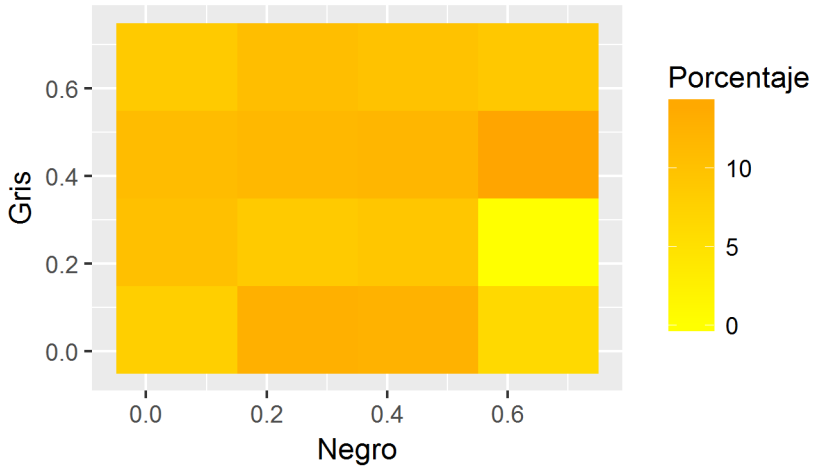

Fig. 4. Mapa de calor para las probabilidades de gris y negro.

En la figura 5 se muestra el mapa de calor para las probabilidades de color blanco y negro donde se observa que las zonas más obscuras representan los porcentajes de resultados correctos donde se observa que los porcentajes son mayores al 5% pero que aún con el cambio de probabilidad del color de los píxeles de color blanco y negro ya que se pueden obtener porcentajes nulos de resultados correctos.

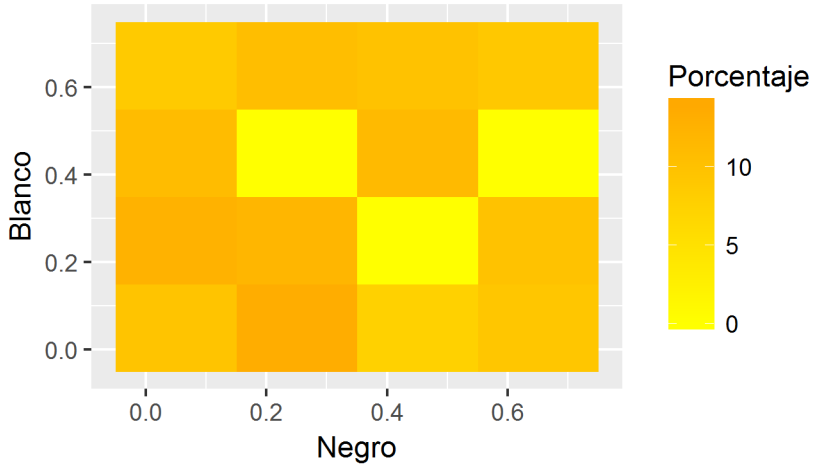

Fig. 5. Mapa de calor para las probabilidades de blanco y negro.

Conclusiones Reto 1

El uso de mapas de calor sirve para estudiar el efecto sistemático del cambio de probabilidades en la red neuronal, son de gran importancia ya que permiten conocer los cambios significativos en los resultados correctos y su porcentaje, así como las zonas en las cuales se obtienen mejores resultados con la identificación de los cambios de color. También ayuda a mejorar la selección del

cambio de las probabilidades. El cambio más significativo encontrado visualmente está en el mapa de calor donde se modifican las probabilidades de los pixeles de color gris y negro ya que esto permite a la red neuronal cometer menos errores en la selección de los dígitos.

Referencias

1) https://plot.ly/r/heatmaps/

2) http://elisa.dyndns-web.com/teaching/comp/par/p7.html