Práctica 12

Red neuronal

Introducción

En esta práctica se simula una red neuronal, esta red neuronal se compone de perceptrones los cuales después de un entrenamiento previo, pueden evaluar una situación y dar un resultado que puede ser correcto o incorrecto dependiendo del nivel de entrenamiento que se le aplique. En este caso entrenaremos a nuestros perceptrones para que reconozcan diez dígitos (0-9). La entrada que recibirá será por medio de una imagen la cual tendrá el digito en cuestión.

Objetivos

1. Paralelizar el código original y estudiar los efectos en los tiempos de ejecución de la red neuronal.
2. Realizar un estudio sistemático del desempeño de la red neuronal para los diez dígitos modificando las tres probabilidades asignadas a la generación de los mismos.

Simulación y Resultados

Para el primer objetivo se identificaron las etapas del código que se repetían muchas veces y que son independientes entre sí, para poder paralelizar. La etapa elegida en esta ocasión fue la etapa de prueba donde los perceptrones se encargan de evaluar un digito y regresar otro digito que puede ser o no el correcto. Para llevar a cabo esta paralelización primero se declaró una función *prueba* y se modificó la parte del código original, específicamente la parte de la matriz *contadores*, ya que al momento de paralelizar es mala idea que todos los núcleos puedan modificar la misma variable al mismo tiempo. En lugar de esto lo que regresa nuestra función es un *TRUE* si nuestro perceptrón acertó y un *FALSE* si falló con respecto al digito que recibió como entrada. Posteriormente con ayuda del paquete *doparallel* y la instrucción *foreach* se realizó una paralelización con 300 dígitos a evaluar donde para cada una se almacenó la respuesta de nuestra función en un vector y finalmente se calculó el porcentaje de aciertos de nuestra simulación. El código que se utilizó para llevar a cabo la paralelización se muestra a continuación:

```
1. prueba<-function(){
2.   d <- sample(0:tope, 1)
3.   pixeles <- runif(dim) < modelos[d + 1,] # fila 1 contiene el
   cero, etc.
4.   correcto <- binario(d, n)
5.   salida <- rep(FALSE, n)
6.   for (i in 1:n) {
7.     w <- neuronas[i,]
8.     deseada <- correcto[i]
9.     resultado <- sum(w * pixeles) >= 0
10.          salida[i] <- resultado
11.       }
```

```
12.         r <- min(decimal(salida, n), k) # todos los no-existentes
   van al final
13.         if (r==d){
14.            return(TRUE)
15.          }
16.       }
17.       aciertos<-foreach(t=1:a, .combine = c) %dopar% prueba()
18.       stopImplicitCluster()
19.       AciertosP<-round((sum(aciertos)/a)*100,2)
20.       print(paste("Porcentaje de Aciertos P",AciertosP,"%"))
```

Para corroborar si nuestra paralelización fue exitosa se corrieron ambos programas utilizando la instrucción *source* y con ayuda de dos *for*, el primero para variar la cantidad de dígitos de prueba *a* y el segundo la cantidad de réplicas *r*. El código para realizar dicha simulación se muestra a continuación:

```
1. for (a in seq(500,2000,500)){
2.      for(r in 1:20){
3.
4.      source('~/GitHub/SimulacionComputacional/P12/original.R')
5.      Toriginal=cbind(a,r,"o",Tiempo,AciertosP)
6.
7.
8.      source('~/GitHub/SimulacionComputacional/P12/P12p.R')
9.      Tparalelo=cbind(a,r,"p",Tiempo,AciertosP)
10.           Resultados=rbind(Resultados,Toriginal,Tparalelo)
11.           }
12.         }
```

La figura 1 muestra la simulación realizada para un valor de dígitos prueba *a* (200, 400, 600, 800 y 1000) y un número de réplicas *r* igual a 10. Donde se observa que los tiempos son muy parecidos para el programa paralelizado y original, aunque se observa que a mayor número de dígitos el programa paralelizado tiene un mejor tiempo de ejecución.

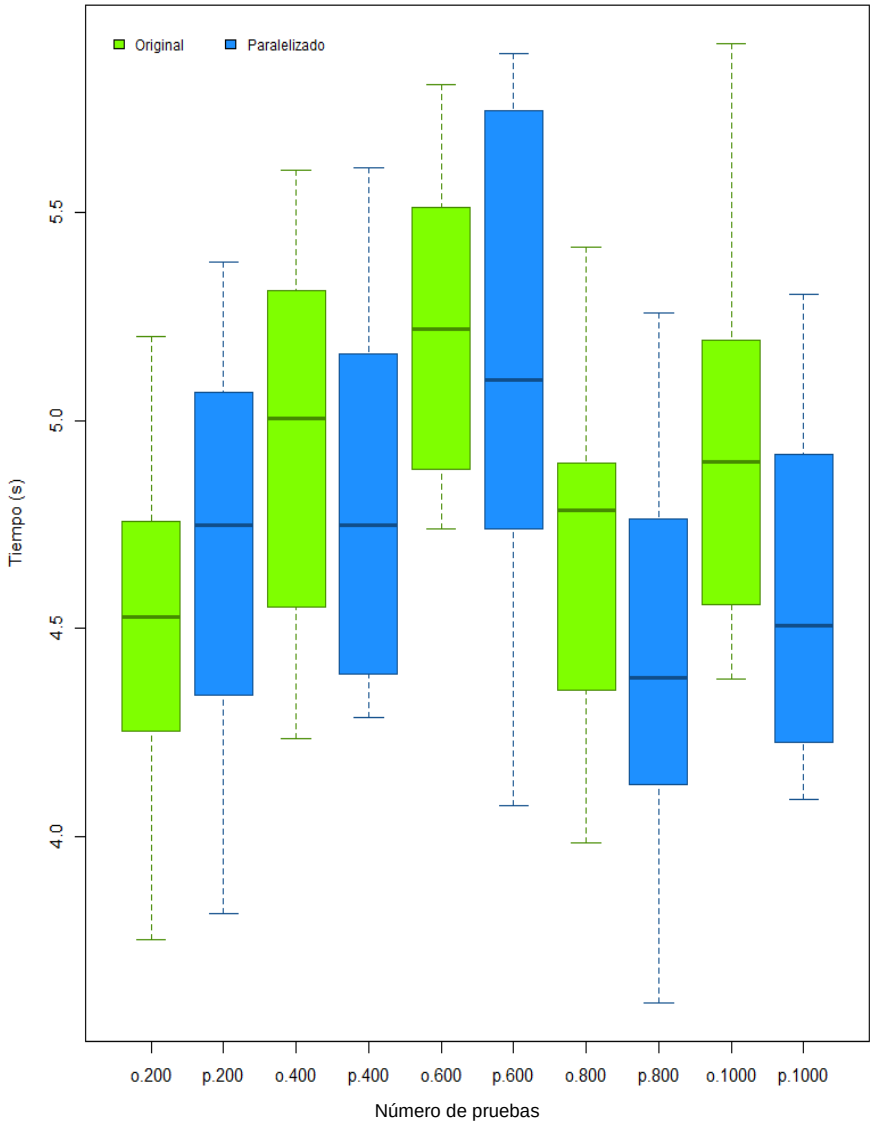


Figura 1. Gráfico de caja-bigotes comparación de tiempos de ejecución.

Para apreciar de mejor manera el cambio en el tiempo entre ambos programas, se decidió aumentar la cantidad de dígitos de prueba *a* (500, 1000, 1500, 2000) con 10 réplicas, los resultados obtenidos se muestran en la figura 2 donde se observa que a mayores valores de dígitos prueba el programa paralelizado brilla más.

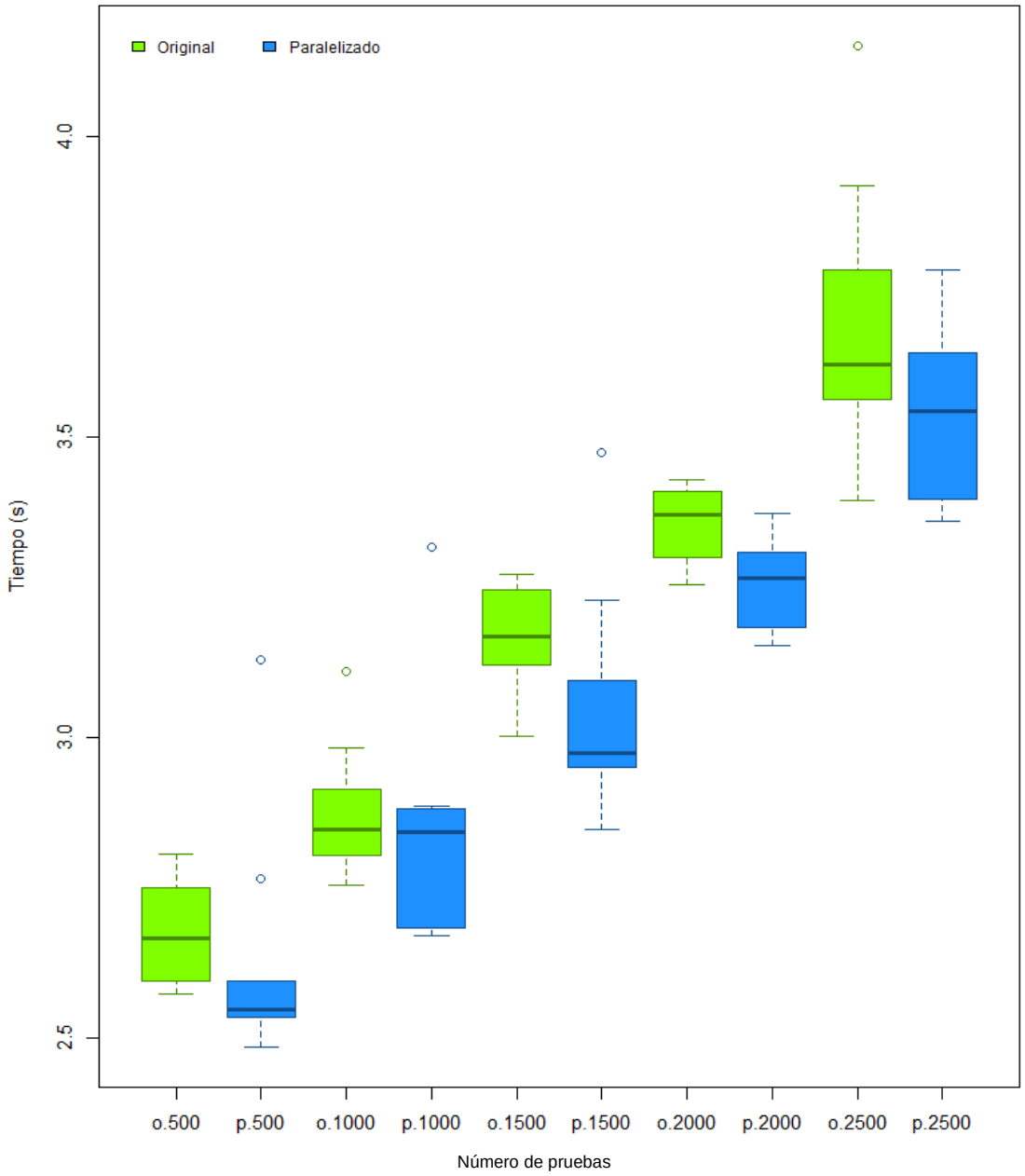


Figura 2. Gráfico de caja-bigotes comparación de tiempos de ejecución.

Además se analizó la distribución de los porcentajes obtenidos con ambos programas por medio de diagramas de caja-bigotes mostrados en la figura 3 donde se observa que los porcentajes de acierto para ambos programas tienen valores muy parecidos y la diferencia no es significativa. Por lo tanto, nuestro programa paralelizado hace lo mismo que el programa original.

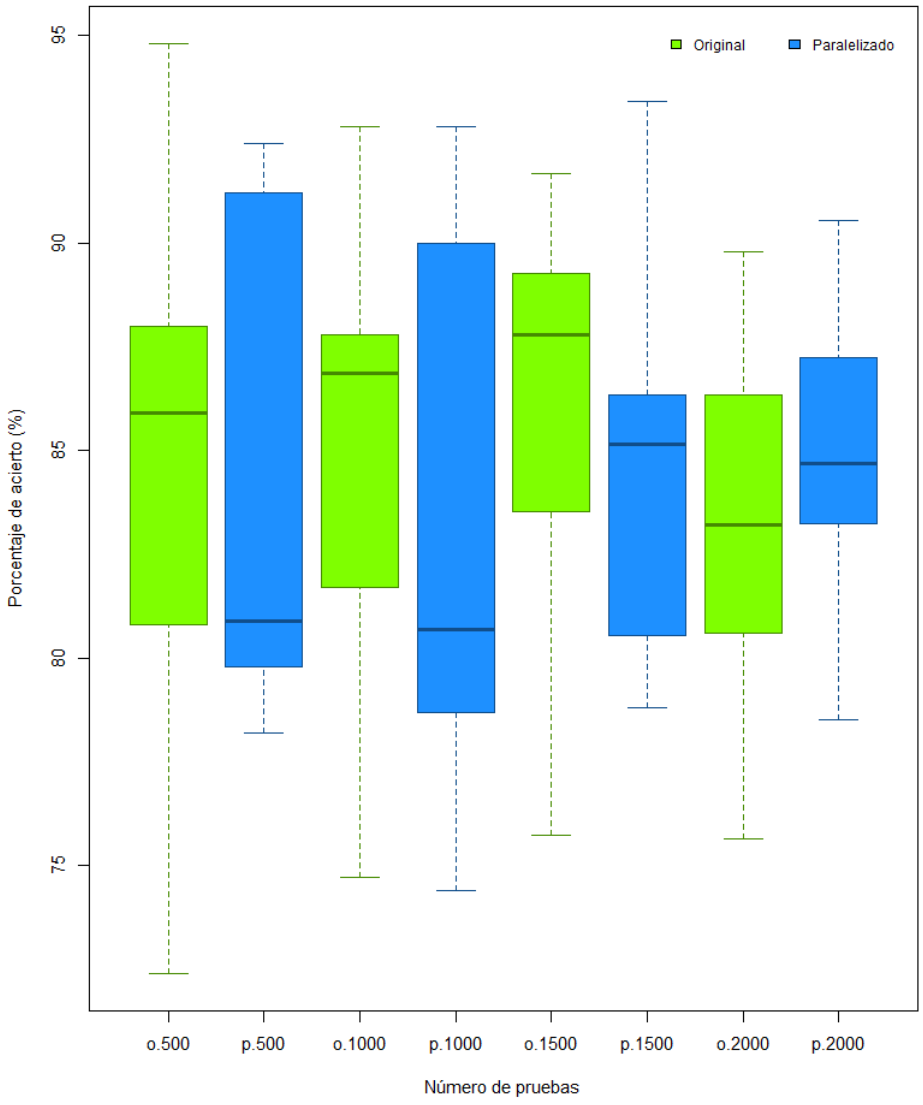


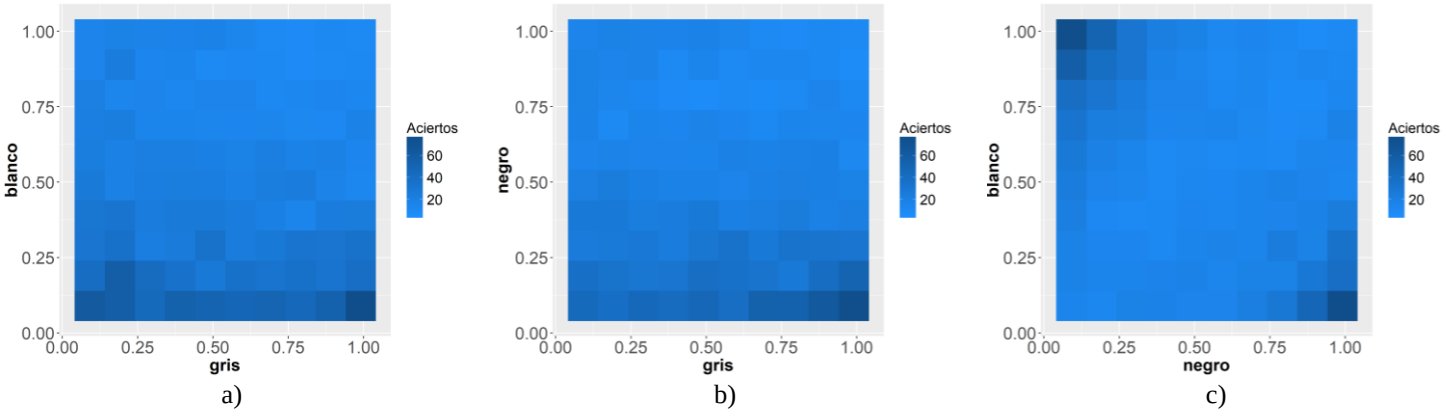
Figura 3. Gráfico de caja-bigotes comparación de porcentaje de aciertos para ambos programas.

Reto 1

Para cumplir con el objetivo del primer reto se realizó un estudio sistemático del desempeño de la red neuronal para los diez dígitos modificando las tres probabilidades asignadas a la generación de los mismos. Para lograr esto se declararon las variables *negro, gris y blanco* y se controlaron por medio de tres *for*, donde se tomaron valores de 0.09 a 0.99 con un aumento de 0.1 en cada una de las variables, esto con el fin de lograr un gran número de combinaciones y determinar la contribución de cada una en el desempeño de la red. Se guardaron los porcentajes obtenidos para cada una de las combinaciones en el data frame *Resultados* donde con el paquete *ggplot2* se graficó un mapa de calor con la instrucción *geom_raster* que se muestra en la figura 4. El código para realizar el reto 1 se muestra a continuación:

```
a=200
for (negro in seq(0.09,0.99,0.1))
  for (gris in seq(0.09,0.99,0.1))
    for (blanco in seq(0.09,0.99,0.1)){

    source('~/GitHub/SimulacionComputacional/P12/P12p.R')
    Tparalelo=cbind(a,"p",AciertosP,negro,gris,blanco)
    print(Tparalelo)
    Resultados=rbind(Resultados,Tparalelo)
      }

      library(ggplot2)
      ggplot(Resultados, aes(negro, blanco)) +
        geom_raster(aes(fill=Paciertos)) +
        scale_fill_gradient(low="dodgerblue", high="dodgerblue4")+
        coord_fixed(ratio = 1)
      ggsave(filename = "n-b.png")

      ggplot(Resultados, aes(gris, blanco)) +
        geom_raster(aes(fill=Paciertos)) +
        scale_fill_gradient(low="dodgerblue", high="dodgerblue4")+
        coord_fixed(ratio = 1)
      ggsave(filename = "g-b.png")

      ggplot(Resultados, aes(gris, negro)) +
        geom_raster(aes(fill=Paciertos)) +
        scale_fill_gradient(low="dodgerblue", high="dodgerblue4")+
        coord_fixed(ratio = 1)
      ggsave(filename = "g-n.png")
```

En la figura 4 se observan los mapas de calor con combinaciones de dos colores a diferentes probabilidades, en la figura 4a se observa que a probabilidades pequeñas del color blanco y en todo el rango de probabilidades del gris se obtiene un mejor porcentaje de acierto donde los mejores se encuentran en los extremos de la probabilidad del color gris, en el figura 4b podemos observar un comportamiento similar a la figura anterior donde para probabilidades de color negro bajas y en probabilidades de color gris alta tenemos el mejor porcentaje de aciertos. Podemos concluir de estas dos figuras que la probabilidad del color gris es la que tiene una mayor contribución en el porcentaje de aciertos. Finalmente en la figura 4c muestra la relación que existe entre las probabilidades de los colores blanco y negro donde se observa que cuando el color blanco tiene grandes probabilidades y el negro bajas, se obtienen los mejores porcentajes de acierto y viceversa.



a) b) c)

Figura 4. Mapa de calor comparativo de porcentaje de aciertos en función de dos colores a) blanco y gris, b) negro y gris, c) blanco y negro.