Tarea: clasificación multinomial en keras

Preparación de datos

```
library(tidyverse)
library(keras)
use_session_with_seed(421) # ayuda a reproducibilidad, pero más lento
set.seed(112)
zip_train <- read_csv("../datos/zip-train.csv")
zip_test <- read_csv("../datos/zip-test.csv")
zip_train_x <- zip_train %>% select(-X1) %>% as.matrix
zip_test_x <- zip_test %>% select(-V1) %>% as.matrix
zip_train_y <- zip_train$X1
zip_test_y <- zip_test$V1
```

```
# Convertir a categóricas (hace one-hot encoding)
num_clases <- 10
y_train <- to_categorical(zip_train_y, num_clases)
y_test <- to_categorical(zip_test_y, num_clases)
```

Definir modelo

Definimos regresión logística multinomial

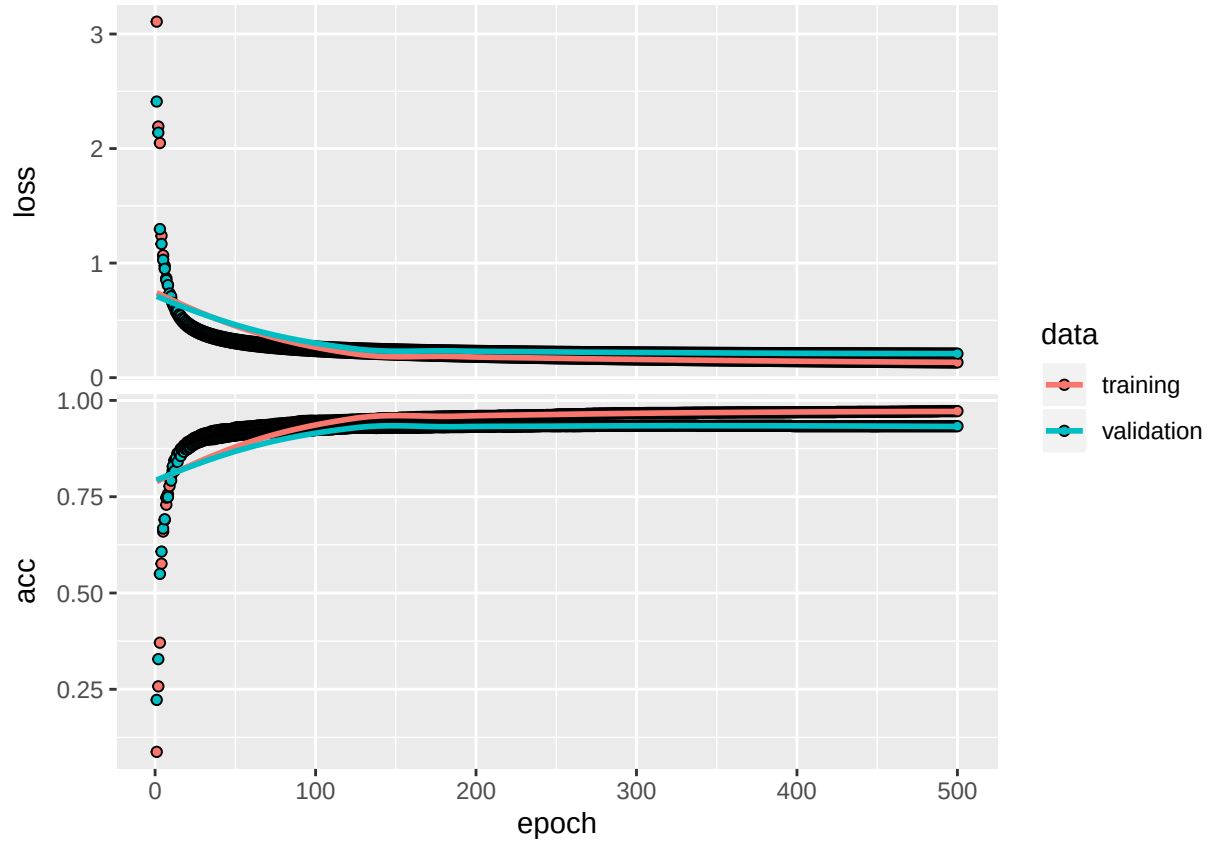
```
model <- keras_model_sequential() %>%
  layer_dense(units = num_clases, activation = "softmax")
```

Compilamos y corremos:

```
model %>% compile(
  loss = loss_categorical_crossentropy,
  optimizer = optimizer_sgd(lr = 0.2),
  metrics = c('accuracy') # ver tasa de correctos
)
# Entrenar con descenso en gradiente
history <- model %>% fit(
  zip_train_x, y_train,
  batch_size = nrow(zip_train_x),
  validation_split = 0.5,
  verbose = 0,
  epochs = 500)
```

Puedes revisar la convergencia graficando o examinando *history*:

```
plot(history)
```



Evalúa con la muestra de prueba (recuerda que entropía cruzada es lo mismo que devianza promedio, pero sin el factor de 2):

```
scores <- model %>% evaluate(
  zip_test_x, y_test, verbose = 0
)

# Output metrics
cat('Prueba - pérdida (entropía cruzada):', scores[[1]], '\n')
```

```
## Prueba - pérdida (entropía cruzada): 0.362849
```

```
cat('Prueba - tasa de correctos :', scores[[2]], '\n')
```

```
## Prueba - tasa de correctos : 0.9033383
```

Coeficientes del modelo

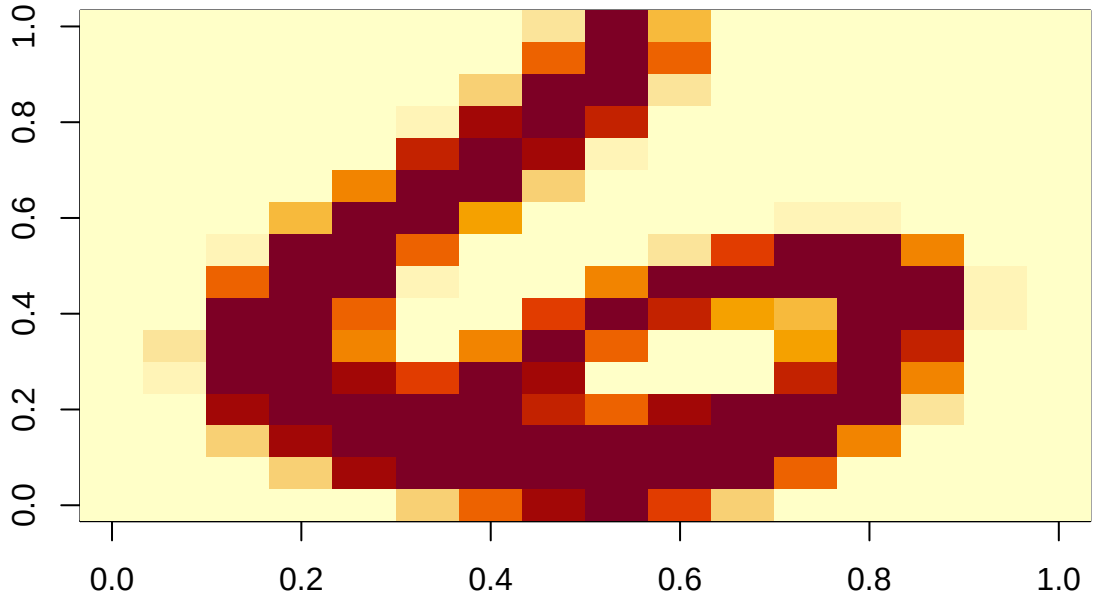
```
coeficientes <- get_weights(model)[[1]]
ordenadas <- get_weights(model)[[2]]
```

Pregunta 1 ¿De qué tamaño es la matriz de coeficientes? Explica la dimensión en términos de los datos (entradas y variable a predecir)

Pregunta 2 Grafica los coeficientes asociados al dígito 1. ¿Cuáles son los dígitos que tienen valores positivos grandes (aumentan la probabilidad de que el dígito sea 1)?

Por ejemplo, para graficar un dígito, hacemos:

```
graficar_vector <- function(x){
    mat <- matrix(as.numeric(x), nrow = 16)[1:16, 16:1]
    image(mat)
}
graficar_vector(zip_train_x[1, ])
```



Aplica la misma función para examinar los coeficientes asociados a la clase del dígito 1, por ejemplo:

Predicciones

Puedes producir predicciones probabilísticas con Keras de la siguiente forma. Para los casos 1 y 30 de prueba, por ejemplo:

```
probas <- predict(model, x = zip_test_x[c(1,30), ,drop = FALSE]) %>% t
probas %>% round(3)
```

```
##        [,1]  [,2]
##  [1,] 0.000 0.030
##  [2,] 0.000 0.000
##  [3,] 0.000 0.000
##  [4,] 0.000 0.740
##  [5,] 0.013 0.001
##  [6,] 0.000 0.053
##  [7,] 0.000 0.000
##  [8,] 0.006 0.000
##  [9,] 0.002 0.173
## [10,] 0.979 0.003
```

Pregunta 3 reconstruye estas probabilidades usando la matriz de coeficientes y ordenadas (interceptos) en el paso anterior. Tip: multiplica coeficientes por los datos de entrada, suma las ordenadas y aplica la función softmax (aplica exponencial y normaliza):

¿Qué pasa si corremos más iteraciones?

Ahora corremos más iteraciones:

```
model %>% fit(
  zip_train_x, y_train,
  batch_size = nrow(zip_train_x),
  epochs = 6000,
  validation_split = 0.5,
  verbose = 0
)
scores <- model %>% evaluate(
  zip_test_x, y_test
)

# Output metrics
cat('Prueba - pérdida (entropía cruzada):', scores[[1]], '\n')
```

```
## Prueba - pérdida (entropía cruzada): 0.4664774
```

```
cat('Prueba - tasa de correctos :', scores[[2]], '\n')
```

```
## Prueba - tasa de correctos : 0.8998505
```

Pregunta 4: ¿Qué modelo ajustado es mejor? Explica por qué.