Resolución de un problema de clasificación con aprendizaje profundo utilizando un subconjunto del conjunto MNIST

Carlos García Gutiérrez (UO139393)

Introducción

La ejecución de esta práctica consta de tres partes:
- Cargar en memoria los datos a utilizar
- Crear, entrenar y comparar los resultados de varias configuraciones de redes densas
- Crear, entrenar y comparar los resultados de varias configuraciones de redes convolucionales

Carga de datos en memoria

```
library(keras)
```

Obtenemos el dataset MNIST

```
mnist <- dataset_mnist()
```

Definimos una semilla con los dígitos del DNI y generamos una secuencia aleatoria con un tamaño de la mitad del de la lista de imágenes/etiquetas

```
set.seed(53540153)
sample_array <- sample.int(nrow(mnist$train$x), size = floor(.10 * nrow(mnist$train$x)))
#PONER AL 50% ANTES DE ENTREGAR!!!
```

Obtenemos la mitad de las imágenes/etiquetas para entrenar
El conjunto de test es el completo

```
train_images <- mnist$train$x[sample_array,,]
train_labels <- mnist$train$y[sample_array]
test_images <- mnist$test$x
test_labels <- mnist$test$y
```

Se reordenan los datos para poder ser usados como entrada de las redes neuronales y se escalan los valores RGB de las imágenes para que estén en el intervalo [0, 1], asimismo se transforman los las etiquetas a valores binarios, según el dígito que representen

```
train_images <- array_reshape(train_images, c(nrow(train_images), 28 * 28))
train_images <- train_images / 255
test_images <- array_reshape(test_images, c(nrow(test_images), 28 * 28))
test_images <- test_images / 255
train_labels <- to_categorical(train_labels)
test_labels <- to_categorical(test_labels)
```

Redes neuronales densas Vamos a crear tres redes neuronales densas: con dos capas (entrada y salida), con tres capas

(igual que la anteriore pero con una capa oculta) y con cuatros capas (igual que la primera pero con dos capas ocultas)

Las capa de entrada contiene una neurona por cada pixel (28 x 28) y estas se activan utilizando la función ReLU, que es la adecuada para la escala de grises de las imágenes

La capa de salida tiene 10 neuronas, que son las necesarias para las 10 categorías a considerar (valores del 0 al 9), en este caso la función de activación es la adecuada para problemas de clasificación múltiple de una etiqueta (como es nuestro caso)

Evidentemente, para un problema tan sencillo, no sería necesarias redes con tantas capas, pero se ha decidido utilizar estas configuraciones para ilustrar el problema del sobreajuste e intentar minimizarlo posteriormente mediante la regularización

Para todas las redes, la función a minimizar es "categorical_crossentropy", que es la adecuada para para problemas de clasificación múltiple de una etiqueta. La optimización estará basada en el descenso del gradiente utilizando solo un conjunto de los pesos, según lo visto en clase

```
dense_network_2layers <- keras_model_sequential() %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(28 * 28)) %>%
  layer_dense(units = 10, activation = "softmax")

dense_network_2layers %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)

summary(dense_network_2layers)
```

```
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## dense (Dense)                (None, 512)               401920
## _________________________________________________________________________
## dense_1 (Dense)              (None, 10)                5130
## =========================================================================
## Total params: 407,050
## Trainable params: 407,050
## Non-trainable params: 0
## _________________________________________________________________________
```

```
dense_network_3layers <- keras_model_sequential() %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(28 * 28)) %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")

dense_network_3layers %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)

summary(dense_network_3layers)
```

```
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_2 (Dense)              (None, 512)               401920
## _________________________________________________________________
## dense_3 (Dense)              (None, 512)               262656
## _________________________________________________________________
## dense_4 (Dense)              (None, 10)                5130
## =================================================================
## Total params: 669,706
## Trainable params: 669,706
## Non-trainable params: 0
## _________________________________________________________________
```

```
dense_network_4layers <- keras_model_sequential() %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(28 * 28)) %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")

dense_network_4layers %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)

summary(dense_network_4layers)
```

```
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_5 (Dense)              (None, 512)               401920
## _________________________________________________________________
## dense_6 (Dense)              (None, 512)               262656
## _________________________________________________________________
## dense_7 (Dense)              (None, 512)               262656
## _________________________________________________________________
## dense_8 (Dense)              (None, 10)                5130
## =================================================================
## Total params: 932,362
## Trainable params: 932,362
## Non-trainable params: 0
## _________________________________________________________________
```

Se realiza el entrenamiento de las redes (utilizando cinco iteraciones)

```
dense_network_2layers %>% fit(train_images, train_labels, epochs = 5, batch_size = 128)
dense_network_3layers %>% fit(train_images, train_labels, epochs = 5, batch_size = 128)
dense_network_4layers %>% fit(train_images, train_labels, epochs = 5, batch_size = 128)
```

Se obtienen y se muestran los resultados

```
metrics_dense_network_2_layers <- dense_network_2layers %>% evaluate(test_images, test_labels, verbose =
metrics_dense_network_3_layers <- dense_network_3layers %>% evaluate(test_images, test_labels, verbose =
metrics_dense_network_4_layers <- dense_network_4layers %>% evaluate(test_images, test_labels, verbose =
metrics_dense_network_2_layers
```

```
## $loss
## [1] 0.2367687
##
## $acc
## [1] 0.928
```

```
metrics_dense_network_3_layers
```

```
## $loss
## [1] 0.1778803
##
## $acc
## [1] 0.9458
```

```
metrics_dense_network_4_layers
```

```
## $loss
## [1] 0.208638
##
## $acc
## [1] 0.9435
```

Se puede observar que añadir una capa oculta a la red mejora los resultados, pero añadir una segunda capa oculta los vuelve a empeorar; esto era de esperar ya que las redes más profundas producen un sobreajuste al modelo