Result

Ruiqi Wang

2024-05-22

```
library(keras)
mnist <- dataset_mnist()
train_images <- mnist$train$x
train_labels <- mnist$train$y
test_images <- mnist$test$x
test_labels <- mnist$test$y
str(train_images)
```

```
##  int [1:60000, 1:28, 1:28] 0 0 0 0 0 0 0 0 0 0 ...
```

```
# int [1:60000, 1:28, 1:28] 0 0 0 0 0 0 0 0 0 0 ...
str(test_labels)
```

```
##  int [1:10000(1d)] 7 2 1 0 4 1 4 9 5 9 ...
```

```
# int [1:10000(1d)] 7 2 1 0 4 1 4 9 5 9 ...
train_images <- array_reshape(train_images, c(60000, 28 * 28))
train_images <- train_images / 255
test_images <- array_reshape(test_images, c(10000, 28 * 28))
test_images <- test_images / 255
train_labels <- to_categorical(train_labels)
test_labels <- to_categorical(test_labels)
```

```
network <- keras_model_sequential() %>%
layer_dense(units = 512, activation = "relu", input_shape = c(28 * 28)) %>%
layer_dense(units = 10, activation = "softmax")
network %>% compile(
optimizer = "rmsprop",
loss = "categorical_crossentropy",
metrics = c("accuracy")
)
network %>% fit(train_images, train_labels, epochs = 5, batch_size = 128)
```

```
## Epoch 1/5
## 469/469 - 4s - loss: 0.3817 - accuracy: 0.8893 - 4s/epoch - 8ms/step
## Epoch 2/5
## 469/469 - 3s - loss: 0.3204 - accuracy: 0.9099 - 3s/epoch - 7ms/step
## Epoch 3/5
## 469/469 - 3s - loss: 0.3228 - accuracy: 0.9113 - 3s/epoch - 7ms/step
## Epoch 4/5
## 469/469 - 3s - loss: 0.3325 - accuracy: 0.9102 - 3s/epoch - 7ms/step
## Epoch 5/5
## 469/469 - 3s - loss: 0.3457 - accuracy: 0.9093 - 3s/epoch - 7ms/step
```

```
network_1 <- keras_model_sequential() %>%
layer_dense(units = 512, activation = "relu", input_shape = c(28 * 28)) %>%
layer_dense(units = 10, activation = "softmax")
network_1 %>% compile(
optimizer = "rmsprop",
loss = "categorical_crossentropy",
metrics = c("accuracy")
)
network_1 %>% fit(train_images, train_labels, epochs = 5, batch_size = 64)
```

```
## Epoch 1/5
## 938/938 - 7s - loss: 0.3785 - accuracy: 0.8926 - 7s/epoch - 8ms/step
## Epoch 2/5
## 938/938 - 7s - loss: 0.3450 - accuracy: 0.9064 - 7s/epoch - 7ms/step
## Epoch 3/5
## 938/938 - 7s - loss: 0.3658 - accuracy: 0.9050 - 7s/epoch - 7ms/step
## Epoch 4/5
## 938/938 - 7s - loss: 0.3950 - accuracy: 0.9020 - 7s/epoch - 7ms/step
## Epoch 5/5
## 938/938 - 7s - loss: 0.4291 - accuracy: 0.8990 - 7s/epoch - 7ms/step
```

Here the batch size decreases from 128 to 64. The time is longer than before. Also, the best accuracy is drop from 0.91 to 0.9.

```
metrics <- network %>% evaluate(test_images, test_labels, verbose = 0)
metrics
```

```
##     loss accuracy
## 0.351831 0.914200
```

```
network
```

```
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                      Param #
## ================================================================================
##  dense_1 (Dense)                    (None, 512)                       401920
##  dense (Dense)                      (None, 10)                        5130
## ================================================================================
## Total params: 407050 (1.55 MB)
## Trainable params: 407050 (1.55 MB)
## Non-trainable params: 0 (0.00 Byte)
## ________________________________________________________________________________
```

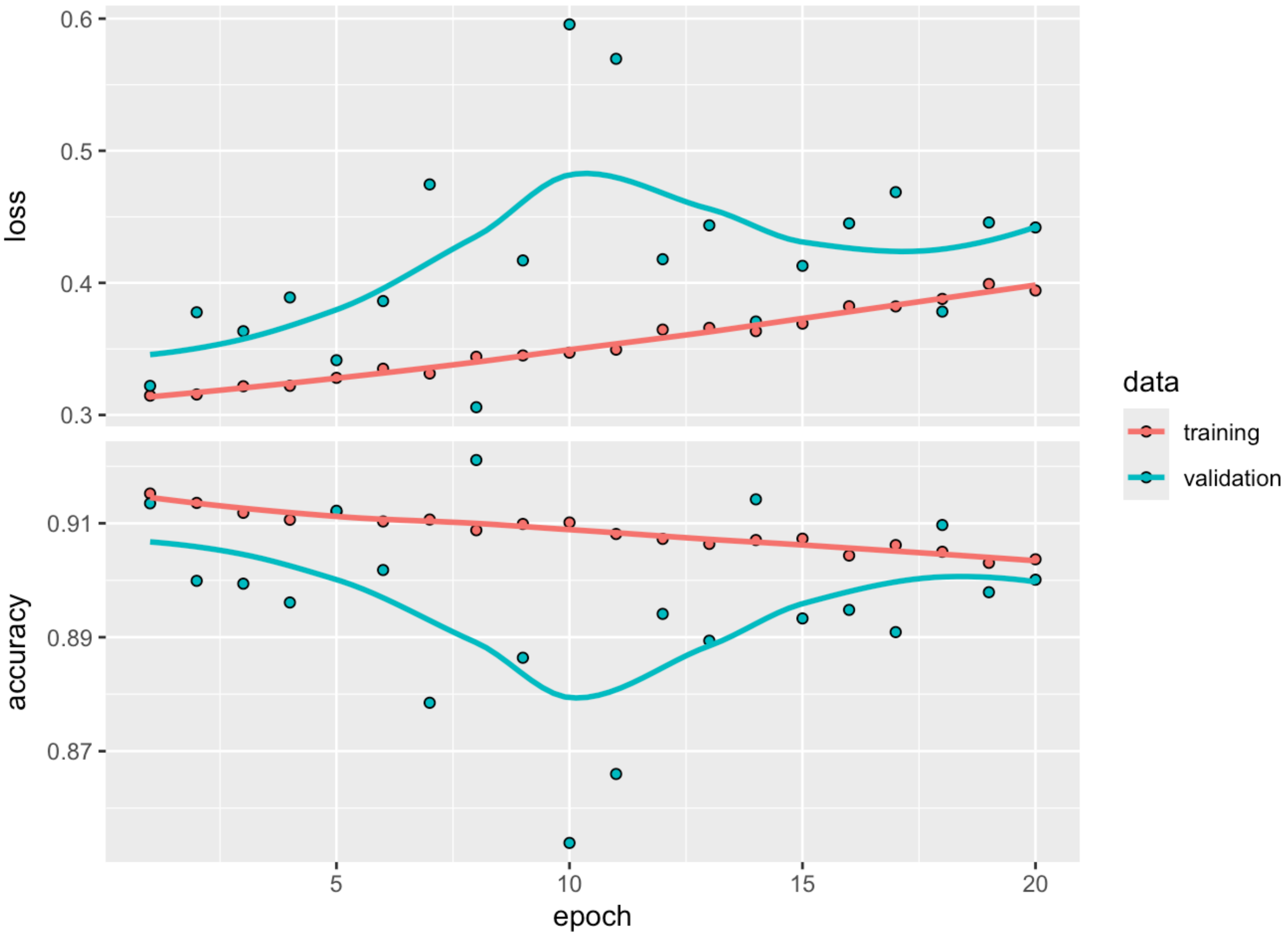
```
#Setting aside a validation set:
val_indices <- 1:10000
x_val <- train_images[val_indices,]
partial_x_train <- train_images[-val_indices,]
y_val <- train_labels[val_indices,]
partial_y_train <- train_labels[-val_indices,]
history <- network %>% fit(
partial_x_train,
partial_y_train,
epochs = 20,
batch_size = 512,
validation_data = list(x_val, y_val)
)
```

```
## Epoch 1/20
## 98/98 - 1s - loss: 0.3146 - accuracy: 0.9152 - val_loss: 0.3220 - val_accuracy: 0.
9135 - 938ms/epoch - 10ms/step
## Epoch 2/20
## 98/98 - 1s - loss: 0.3155 - accuracy: 0.9136 - val_loss: 0.3777 - val_accuracy: 0.
8999 - 826ms/epoch - 8ms/step
## Epoch 3/20
## 98/98 - 1s - loss: 0.3217 - accuracy: 0.9118 - val_loss: 0.3634 - val_accuracy: 0.
8994 - 832ms/epoch - 8ms/step
## Epoch 4/20
## 98/98 - 1s - loss: 0.3221 - accuracy: 0.9106 - val_loss: 0.3890 - val_accuracy: 0.
8961 - 827ms/epoch - 8ms/step
## Epoch 5/20
## 98/98 - 1s - loss: 0.3281 - accuracy: 0.9120 - val_loss: 0.3414 - val_accuracy: 0.
9122 - 822ms/epoch - 8ms/step
## Epoch 6/20
## 98/98 - 1s - loss: 0.3350 - accuracy: 0.9103 - val_loss: 0.3862 - val_accuracy: 0.
9018 - 819ms/epoch - 8ms/step
## Epoch 7/20
## 98/98 - 1s - loss: 0.3314 - accuracy: 0.9107 - val_loss: 0.4745 - val_accuracy: 0.
8785 - 820ms/epoch - 8ms/step
## Epoch 8/20
## 98/98 - 1s - loss: 0.3441 - accuracy: 0.9088 - val_loss: 0.3059 - val_accuracy: 0.
9211 - 826ms/epoch - 8ms/step
## Epoch 9/20
## 98/98 - 1s - loss: 0.3451 - accuracy: 0.9099 - val_loss: 0.4170 - val_accuracy: 0.
8864 - 831ms/epoch - 8ms/step
## Epoch 10/20
## 98/98 - 1s - loss: 0.3472 - accuracy: 0.9101 - val_loss: 0.5957 - val_accuracy: 0.
8539 - 828ms/epoch - 8ms/step
## Epoch 11/20
## 98/98 - 1s - loss: 0.3494 - accuracy: 0.9081 - val_loss: 0.5696 - val_accuracy: 0.
8660 - 831ms/epoch - 8ms/step
## Epoch 12/20
## 98/98 - 1s - loss: 0.3646 - accuracy: 0.9073 - val_loss: 0.4179 - val_accuracy: 0.
8941 - 841ms/epoch - 9ms/step
## Epoch 13/20
## 98/98 - 1s - loss: 0.3659 - accuracy: 0.9064 - val_loss: 0.4436 - val_accuracy: 0.
8894 - 830ms/epoch - 8ms/step
## Epoch 14/20
## 98/98 - 1s - loss: 0.3634 - accuracy: 0.9071 - val_loss: 0.3707 - val_accuracy: 0.
9142 - 829ms/epoch - 8ms/step
## Epoch 15/20
## 98/98 - 1s - loss: 0.3691 - accuracy: 0.9073 - val_loss: 0.4129 - val_accuracy: 0.
8933 - 823ms/epoch - 8ms/step
## Epoch 16/20
## 98/98 - 1s - loss: 0.3823 - accuracy: 0.9043 - val_loss: 0.4451 - val_accuracy: 0.
8948 - 825ms/epoch - 8ms/step
## Epoch 17/20
## 98/98 - 1s - loss: 0.3822 - accuracy: 0.9062 - val_loss: 0.4687 - val_accuracy: 0.
8909 - 825ms/epoch - 8ms/step
## Epoch 18/20
## 98/98 - 1s - loss: 0.3879 - accuracy: 0.9050 - val_loss: 0.3783 - val_accuracy: 0.
9097 - 821ms/epoch - 8ms/step
## Epoch 19/20
## 98/98 - 1s - loss: 0.3992 - accuracy: 0.9031 - val_loss: 0.4457 - val_accuracy: 0.
8979 - 825ms/epoch - 8ms/step
## Epoch 20/20
## 98/98 - 1s - loss: 0.3942 - accuracy: 0.9037 - val_loss: 0.4419 - val_accuracy: 0.
9001 - 824ms/epoch - 8ms/step
```

```
str(history)
```

```
## List of 2
##  $ params :List of 3
##   ..$ verbose: int 2
##   ..$ epochs : int 20
##   ..$ steps  : int 98
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.315 0.315 0.322 0.322 0.328 ...
##   ..$ accuracy    : num [1:20] 0.915 0.914 0.912 0.911 0.912 ...
##   ..$ val_loss    : num [1:20] 0.322 0.378 0.363 0.389 0.341 ...
##   ..$ val_accuracy: num [1:20] 0.914 0.9 0.899 0.896 0.912 ...
##  - attr(*, "class")= chr "keras_training_history"
```

```
plot(history)
```



```r
library(keras)
model <- keras_model_sequential() %>%
layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu",
input_shape = c(28, 28, 1)) %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
layer_max_pooling_2d(pool_size = c(2, 2)) %>%
layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
layer_flatten() %>%
layer_dense(units = 64, activation = "relu") %>%
layer_dense(units = 10, activation = "softmax")
model
```

```
## Model: "sequential_2"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  conv2d_2 (Conv2D)                  (None, 26, 26, 32)              320
##  max_pooling2d_1 (MaxPooling2D)     (None, 13, 13, 32)              0
##  conv2d_1 (Conv2D)                  (None, 11, 11, 64)              18496
##  max_pooling2d (MaxPooling2D)       (None, 5, 5, 64)                0
##  conv2d (Conv2D)                    (None, 3, 3, 64)                36928
##  flatten (Flatten)                  (None, 576)                     0
##  dense_5 (Dense)                    (None, 64)                      36928
##  dense_4 (Dense)                    (None, 10)                      650
## ================================================================================
## Total params: 93322 (364.54 KB)
## Trainable params: 93322 (364.54 KB)
## Non-trainable params: 0 (0.00 Byte)
## ________________________________________________________________________________
```

```
library(keras)
model_1 <- keras_model_sequential() %>%
    layer_conv_2d(filters = 32, kernel_size = c(5, 5), activation = "relu", input_shape = c(28, 28, 1), padding = "same") %>%
    layer_max_pooling_2d(pool_size = c(2, 2)) %>%
    layer_conv_2d(filters = 64, kernel_size = c(5, 5), activation = "relu", padding = "same") %>%
    layer_max_pooling_2d(pool_size = c(2, 2)) %>%
    layer_conv_2d(filters = 64, kernel_size = c(5, 5), activation = "relu", padding = "same") %>%
    layer_flatten() %>%
    layer_dense(units = 64, activation = "relu") %>%
    layer_dense(units = 10, activation = "softmax")

model_1
```

```
## Model: "sequential_3"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## ================================================================================
##  conv2d_5 (Conv2D)                  (None, 28, 28, 32)               832
##  max_pooling2d_3 (MaxPooling2D)     (None, 14, 14, 32)               0
##  conv2d_4 (Conv2D)                  (None, 14, 14, 64)               51264
##  max_pooling2d_2 (MaxPooling2D)     (None, 7, 7, 64)                 0
##  conv2d_3 (Conv2D)                  (None, 7, 7, 64)                 102464
##  flatten_1 (Flatten)                (None, 3136)                     0
##  dense_7 (Dense)                    (None, 64)                       200768
##  dense_6 (Dense)                    (None, 10)                       650
## ================================================================================
## Total params: 355978 (1.36 MB)
## Trainable params: 355978 (1.36 MB)
## Non-trainable params: 0 (0.00 Byte)
## ________________________________________________________________________________
```

Here we can see the kernel size increases from (3,3) to (5,5). The Param # in result table increases a lot. The same situation happens in total params and trainable params.

```
mnist <- dataset_mnist()
c(c(train_images, train_labels), c(test_images, test_labels)) %<-% mnist
train_images <- array_reshape(train_images, c(60000, 28, 28, 1))
train_images <- train_images / 255
test_images <- array_reshape(test_images, c(10000, 28, 28, 1))
test_images <- test_images / 255
train_labels <- to_categorical(train_labels)
test_labels <- to_categorical(test_labels)
model %>% compile(
optimizer = "rmsprop",
loss = "categorical_crossentropy",
metrics = c("accuracy")
)
model %>% fit(
train_images, train_labels,
epochs = 5, batch_size=64
)
```

```
## Epoch 1/5
## 938/938 - 10s - loss: 0.1865 - accuracy: 0.9406 - 10s/epoch - 11ms/step
## Epoch 2/5
## 938/938 - 9s - loss: 0.0557 - accuracy: 0.9828 - 9s/epoch - 10ms/step
## Epoch 3/5
## 938/938 - 9s - loss: 0.0417 - accuracy: 0.9872 - 9s/epoch - 10ms/step
## Epoch 4/5
## 938/938 - 9s - loss: 0.0330 - accuracy: 0.9902 - 9s/epoch - 10ms/step
## Epoch 5/5
## 938/938 - 9s - loss: 0.0278 - accuracy: 0.9923 - 9s/epoch - 10ms/step
```

```
results <- model %>% evaluate(test_images, test_labels)
```

```
## 313/313 - 2s - loss: 0.0351 - accuracy: 0.9896 - 2s/epoch - 6ms/step
```

```
results
```

```
##       loss   accuracy
## 0.03505512 0.98960000
```