Homework 9

Steven Black

4/16/2019

```r
library(keras)
library(ggplot2)
```

IS 677: Introduction to Data Science Spring 2019

Homework Assignment 9 (Due: April 21, 2019, midnight EST)

Use the CIFAR10 dataset that comes with Keras. It has 50,000 training and 10,000 testing images of 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The image size is 32*32*3. The 3 indicates that these are color images.

1. Load the data into a variable called cifar. (5 points)

```r
cifar <- dataset_cifar10()
```

2. Load the first 1000 `train_images` and the corresponding `train_labels`, first 500 test_images and the corresponding test_labels into appropriate variables. (5 points)

```r
train_images <- cifar$train$x[1:1000, 1:32, 1:32, 1:3]
train_labels <- cifar$train$y[1:1000]

test_images <- cifar$test$x[1:500, 1:32, 1:32, 1:3]
test_labels <- cifar$test$y[1:500]
```

3. Create one-hot encoding for the labels for both train and test labels. (5 points)

```r
train_labels_categorical <- to_categorical(train_labels)
test_labels_categorical <- to_categorical(test_labels)
```

4. Reshape and scale the training and test images using image data generators of batch sizes 20. Use the test data for validation. (15 points)

```r
batch_size <- 20

train_datagen <- image_data_generator(rescale = 1/255)
validation_datagen <- image_data_generator(rescale = 1/255)

train_generator <- flow_images_from_data(train_images,
  y = train_labels_categorical,
  generator = train_datagen,
  batch_size = 32)

validation_generator <- flow_images_from_data(test_images,
  y = test_labels_categorical,
  generator = validation_datagen,
  batch_size = 20)
```

5. Create a sequential Keras model similar to listing 5.5 with:

- three conv2d layers with filter sizes of 32, 64, and 128,

- max__pooloing layers of pool size (2,2) after each conv layer,
- and a dropout layer with 25% dropouts after the max_pooling layers
- and 50% dropouts after the pre-final dense layer.
- Note that the last layer is dense with 10 units and softmax activation. (25 points)

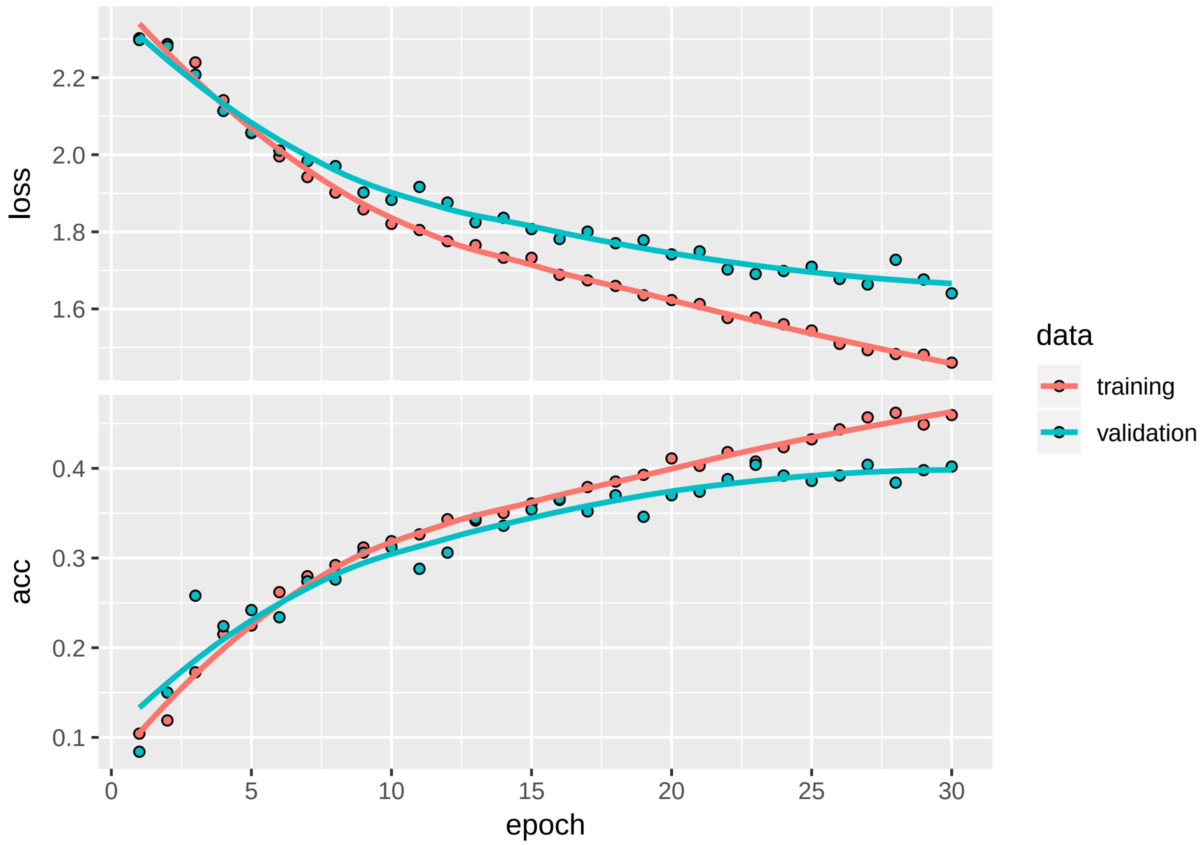
```
build_model <- function() {
  model <- keras_model_sequential() %>%
    layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu",
                  input_shape = c(32, 32, 3)) %>%              # 1/3 conv2d layers with filters = 32
    layer_max_pooling_2d(pool_size = c(2, 2)) %>%              # 1/3 max_pooling w pool_size=(2,2) aft
    layer_dropout(rate = 0.25) %>%                             # 1/3 dropout layer w rate=25% after ma
    layer_conv_2d(filters = 64, kernel_size = c(3, 3),
                  activation = "relu") %>%                     # 2/3 conv2d layers with filters = 64
    layer_max_pooling_2d(pool_size = c(2, 2)) %>%              # 2/3 max_pooling w pool_size=(2,2) aft
    layer_dropout(rate = 0.25) %>%                             # 2/3 dropout layer w rate=25% after ma
    layer_conv_2d(filters = 128, kernel_size = c(3, 3),
                  activation = "relu") %>%                     # 3/3 conv2d layers with filters = 128
    layer_max_pooling_2d(pool_size = c(2, 2)) %>%              # 3/3 max_pooling w pool_size=(2,2) aft
    layer_dropout(rate = 0.25) %>%                             # 3/3 dropout layer w rate=25% after ma
    layer_flatten() %>%
    layer_dense(units = 512, activation = "relu") %>%        # pre-final dense layer
    layer_dropout(rate = 0.50) %>%                           # 50% dropouts after the pre-final dense la
    layer_dense(units = 10, activation = "softmax")          # last layer is dense with 10 units and sof

  return(model)
}

model <- build_model()
```

6. Compile the model with `categorical_crossentropy` as the loss function and `optimizer_rmsprop` with 0.01% learning rate (lr=0.0001). (5 points)

```
model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_rmsprop(lr = 1e-4),
  metrics = c("acc")
)
```

7. Fit the model using `fit_generator()` with 30 epochs. Use the test data generator for the validation data. Store the results in a variable called history. (5 points)

```
epochs <- 30

history <- model %>% fit_generator(
  train_generator,
  steps_per_epoch = 100,
  epochs = epochs,
  validation_data = validation_generator,
  validation_steps = 50)

plot(history)
```

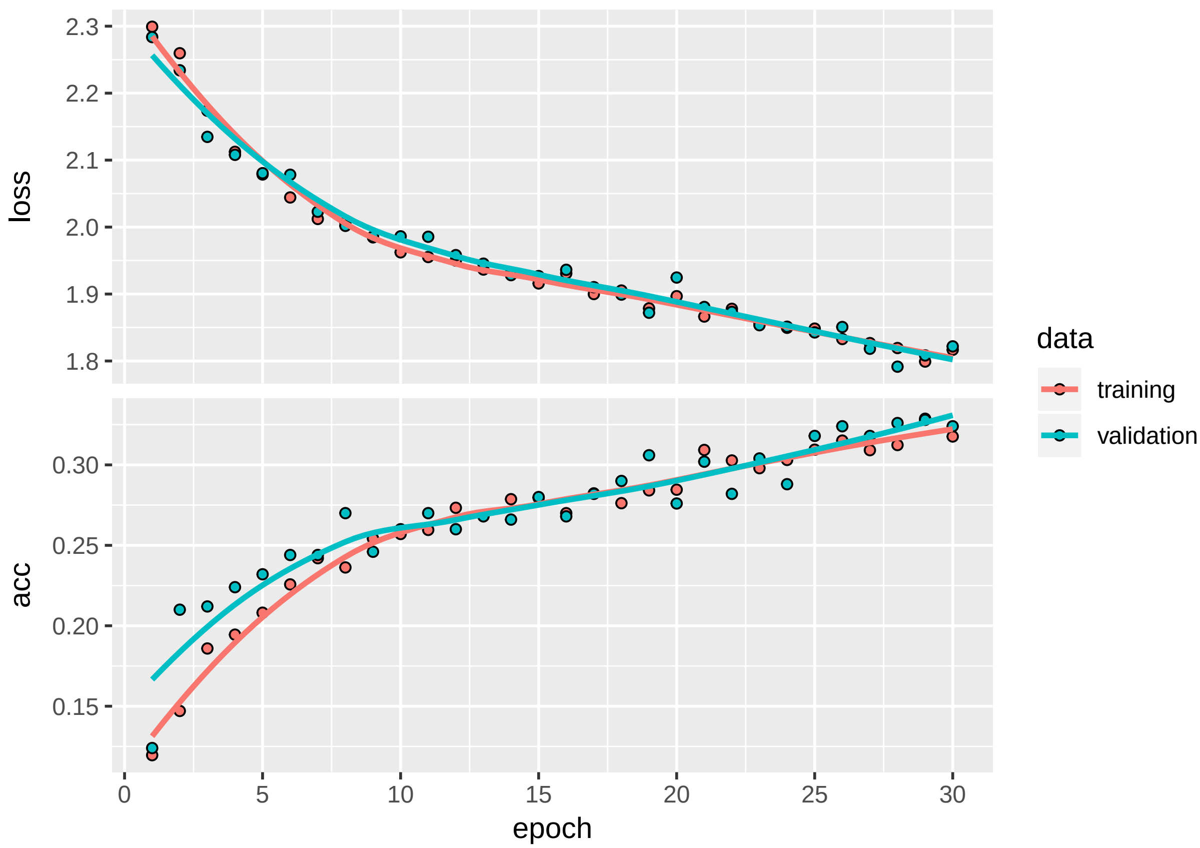


8. Clear the session. Redo steps 5-6. Create a data generator with data augmentation with the parameters shown in listing 5.14 for datagen. Fit the model with the data generator. (25 points)

```
compile_and_fit_model <- function(model, train_generator, validation_generator) {
    model %>% compile(
    loss = "categorical_crossentropy",
    optimizer = optimizer_rmsprop(lr = 1e-4),
    metrics = c("acc")
  )

  history <- model %>% fit_generator(
    train_generator,
    steps_per_epoch = 100,
    epochs = 30,
    validation_data = validation_generator,
    validation_steps = 50
  )

  return(history)
}
```

```
k_clear_session()

datagen <- image_data_generator(
  rescale = 1/255,
  rotation_range = 40,
  width_shift_range = 0.2,
  height_shift_range = 0.2,
```

```
  shear_range = 0.2,
  zoom_range = 0.2,
  horizontal_flip = TRUE
)

test_datagen <- image_data_generator(rescale = 1/255)

train_generator <- flow_images_from_data(train_images,
  y = train_labels_categorical,
  generator = datagen,
  batch_size = 32)

validation_generator <- flow_images_from_data(test_images,
  y = test_labels_categorical,
  generator = test_datagen,
  batch_size = 20)

model <- build_model()

history <- compile_and_fit_model(model, train_generator, validation_generator)

plot(history)
```



9. Explain the results. (10 points)

There is a lot more noise in the most recent training history. Perhaps all the engineering we did in the `datagen` introduced noise into the training data. Noisey training data would explain noisey results.