deep_learning_example-1

Ashish Dutt

February 7, 2018

The following example is taken from the website (https://keras.rstudio.com/)

```
# clean the workspace
rm(list = ls())
#install.packages("keras", dependencies = TRUE)
library(keras)
# Read in MNIST data
mnist <- dataset_mnist()

x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y
```

The x data is a 3-d array (images,width,height) of grayscale values . To prepare the data for training we convert the 3-d arrays into matrices by reshaping width and height into a single dimension (28x28 images are flattened into length 784 vectors). Then, we convert the grayscale values from integers ranging between 0 to 255 into floating point values ranging between 0 and 1:

```
# reshape
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
# rescale
x_train <- x_train / 255
x_test <- x_test / 255
```

Note that we use the `array_reshape()` function rather than the dim<-() function to reshape the array. This is so that the data is re-interpreted using row-major semantics (as opposed to R's default column-major semantics), which is in turn compatible with the way that the numerical libraries called by Keras interpret array dimensions.

The y data is an integer vector with values ranging from 0 to 9. To prepare this data for training we one-hot encode the vectors into binary class matrices using the Keras `to_categorical()` function

```
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)
```

Defining the Model

The core data structure of Keras is a model, a way to organize layers. The simplest type of model is the Sequential model, a linear stack of layers.

We begin by creating a sequential model and then adding layers using the pipe (%>%) operator:

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu', input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
```

```
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = 'softmax')
```

The input_shape argument to the first layer specifies the shape of the input data (a length 784 numeric vector representing a grayscale image). The final layer outputs a length 10 numeric vector (probabilities for each digit) using a softmax activation function.

Use the `summary()` function to print the details of the model:

```
# Print a summary of the model
summary(model)
```

```
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## dense_1 (Dense)              (None, 256)               200960
## _________________________________________________________________________
## dropout_1 (Dropout)          (None, 256)               0
## _________________________________________________________________________
## dense_2 (Dense)              (None, 128)               32896
## _________________________________________________________________________
## dropout_2 (Dropout)          (None, 128)               0
## _________________________________________________________________________
## dense_3 (Dense)              (None, 10)                1290
## =========================================================================
## Total params: 235,146
## Trainable params: 235,146
## Non-trainable params: 0
## _________________________________________________________________________
```

```
# Get model configuration
get_config(model)
```

```
## [{'class_name': 'Dense', 'config': {'name': 'dense_1', 'trainable': True, 'batch_input_shape': (None
```

```
# Get layer configuration
get_layer(model, index = 1)
```

```
## <keras.layers.core.Dense>
```

```
# List the model's layers
model$layers
```

```
## [[1]]
## <keras.layers.core.Dense>
##
## [[2]]
## <keras.layers.core.Dropout>
##
## [[3]]
## <keras.layers.core.Dense>
##
## [[4]]
## <keras.layers.core.Dropout>
##
## [[5]]
## <keras.layers.core.Dense>
```

```
# List the input tensors
model$inputs
```

```
## [[1]]
## Tensor("dense_1_input:0", shape=(?, 784), dtype=float32)
```

```
# List the output tensors
model$outputs
```

```
## [[1]]
## Tensor("dense_3/Softmax:0", shape=(?, 10), dtype=float32)
```

Compile the Model

```
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)
```

Training and Evaluation

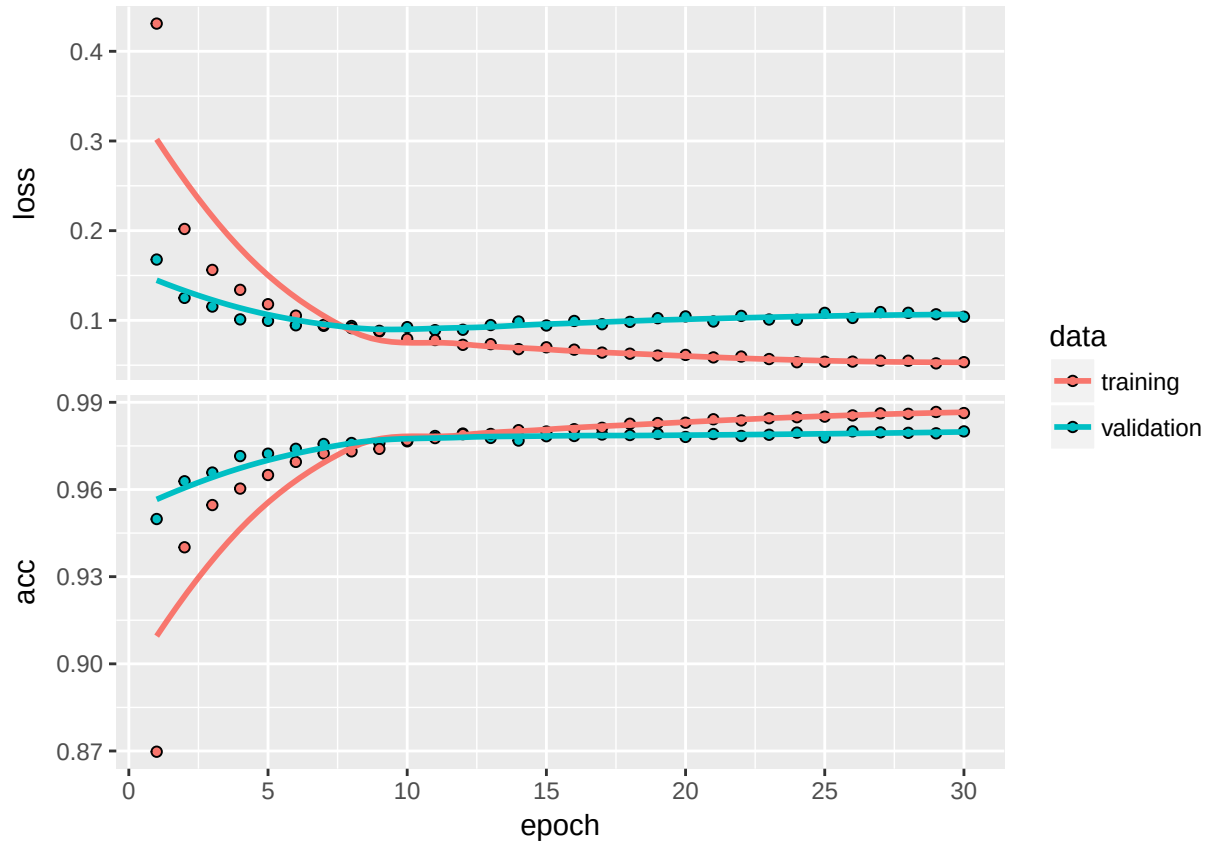
Use the fit() function to train the model for 30 epochs using batches of 128 images:

```
history <- model %>% fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2
)
```

Plot the Model

The `history` object returned by `fit()` includes loss and accuracy metrics which we can plot:

```
plot(history)
```



Evaluating model performance Evaluate the model's performance on the test data:

```
model %>% evaluate(x_test, y_test)
```

```
## $loss
## [1] 0.1085
##
## $acc
## [1] 0.9813
```

Generate predictions on new data:

```
model %>% predict_classes(x_test)
```

```
##   [1] 7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1 3 1 3 4 7
##  [36] 2 7 1 2 1 1 7 4 2 3 5 1 2 4 4 6 3 5 5 6 0 4 1 9 5 7 8 9 3 7 4 6 4 3 0
##  [71] 7 0 2 9 1 7 3 2 9 7 7 6 2 7 8 4 7 3 6 1 3 6 9 3 1 4 1 7 6 9
##  [ reached getOption("max.print") -- omitted 9900 entries ]
```