Unit 5 Lecture 1: Deep Learning Preliminaries

November 15, 2022

In this R demo, we'll get warmed up for deep learning by fitting a multi-class logistic regression to the MNIST handwritten digit data. The inputs are 28 pixel by 28 pixel images of handwritten digits (a total of 784 pixels), and the output is one of the ten categories 0, 1,. . . , 9.

First let's load some libraries:

```
library(keras)     # for deep learning
library(stat471)   # for deep learning helper functions
library(tidyverse) # for everything else
```

Next let's load the MNIST data and do some reshaping and rescaling:

```
# load the data
mnist <- dataset_mnist()
```

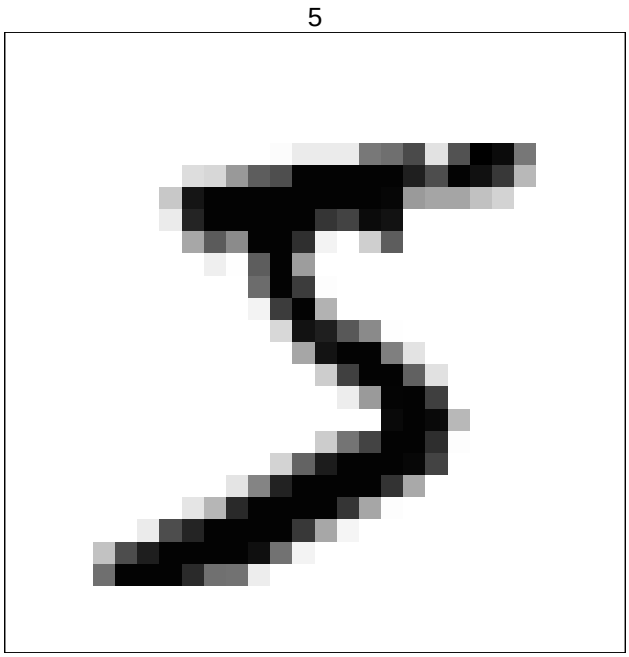
```
## Loaded Tensorflow version 2.9.2
```

```
# extract information about the images
num_train_images <- dim(mnist$train$x)[1]    # number of training images
num_test_images <- dim(mnist$test$x)[1]      # number of test images
img_rows <- dim(mnist$train$x)[2]            # rows per image
img_cols <- dim(mnist$train$x)[3]            # columns per image
num_pixels <- img_rows * img_cols            # pixels per image
num_classes <- length(unique(mnist$train$y)) # number of image classes
max_intensity <- 255                         # max pixel intensity

# normalize and reshape the images
x_train <- array_reshape(
  mnist$train$x / max_intensity,
  c(num_train_images, img_rows, img_cols, 1)
)
x_test <- array_reshape(
  mnist$test$x / max_intensity,
  c(num_test_images, img_rows, img_cols, 1)
)

# extract the responses from the training and test data
g_train <- mnist$train$y
g_test <- mnist$test$y

# recode response labels using "one-hot" representation
y_train <- to_categorical(g_train, num_classes)
y_test <- to_categorical(g_test, num_classes)
```

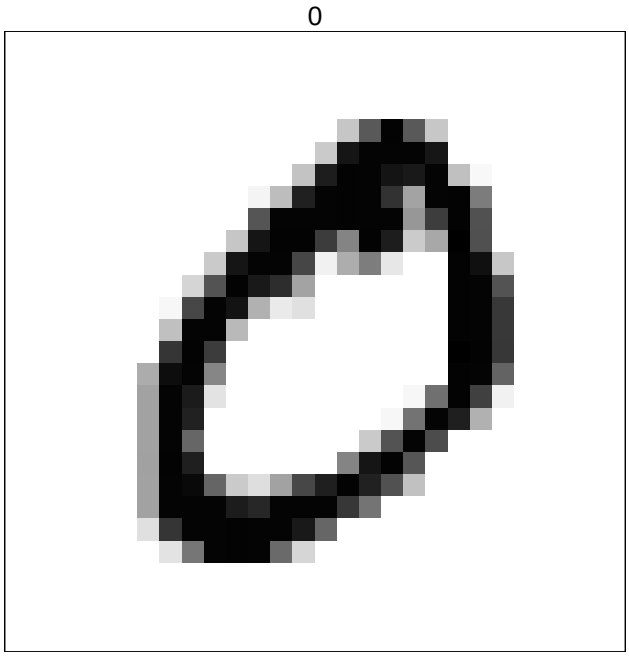
Let's plot a few of the digits:

```
# plot a few of the digits
p1 <- plot_grayscale(x_train[1, , , ], g_train[1])
```

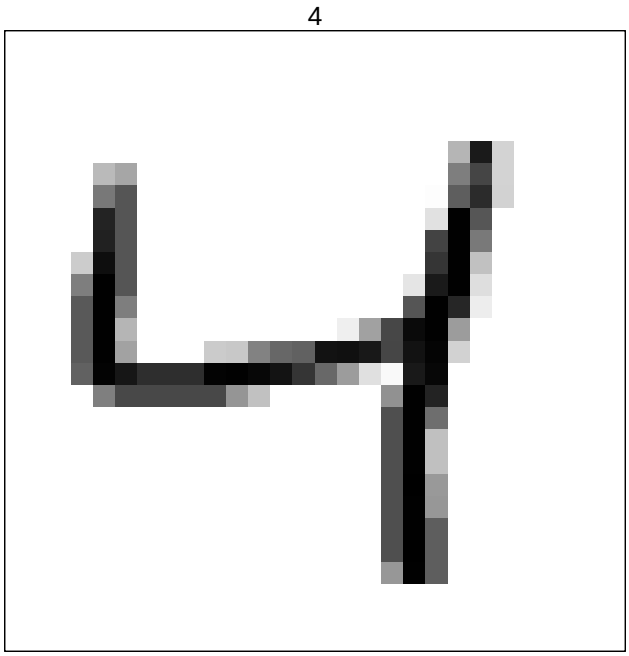
```
plot(p1)
```




```
p2 <- plot_grayscale(x_train[2, , , ], g_train[2])
plot(p2)
```




```
p3 <- plot_grayscale(x_train[3, , , ], g_train[3])
plot(p3)
```



Now, we can define the class of model we want to train (multi-class logistic model):

```
model_lr <- keras_model_sequential() %>%
  layer_flatten(
    input_shape =          # flatten the images
      c(img_rows, img_cols, 1)
  ) %>%
  layer_dense(
    units = num_classes,   # number of outputs
    activation = "softmax" # type of activation function
  )
```

We can get a summary of this model as follows:

```
summary(model_lr)
```

```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  flatten (Flatten)                  (None, 784)                     0
##  dense (Dense)                      (None, 10)                      7850
## =========================================================================
## Total params: 7,850
## Trainable params: 7,850
## Non-trainable params: 0
## _________________________________________________________________________
```

Now we need to *compile* the model, which adds information to the object about which loss we want to use, which way we want to optimize the loss, and how we will evaluate validation error:

```
model_lr %>%                           # note: modifying model_lr "in place"
  compile(
    loss = "categorical_crossentropy",  # which loss to use
    optimizer = optimizer_rmsprop(),    # how to optimize the loss
```

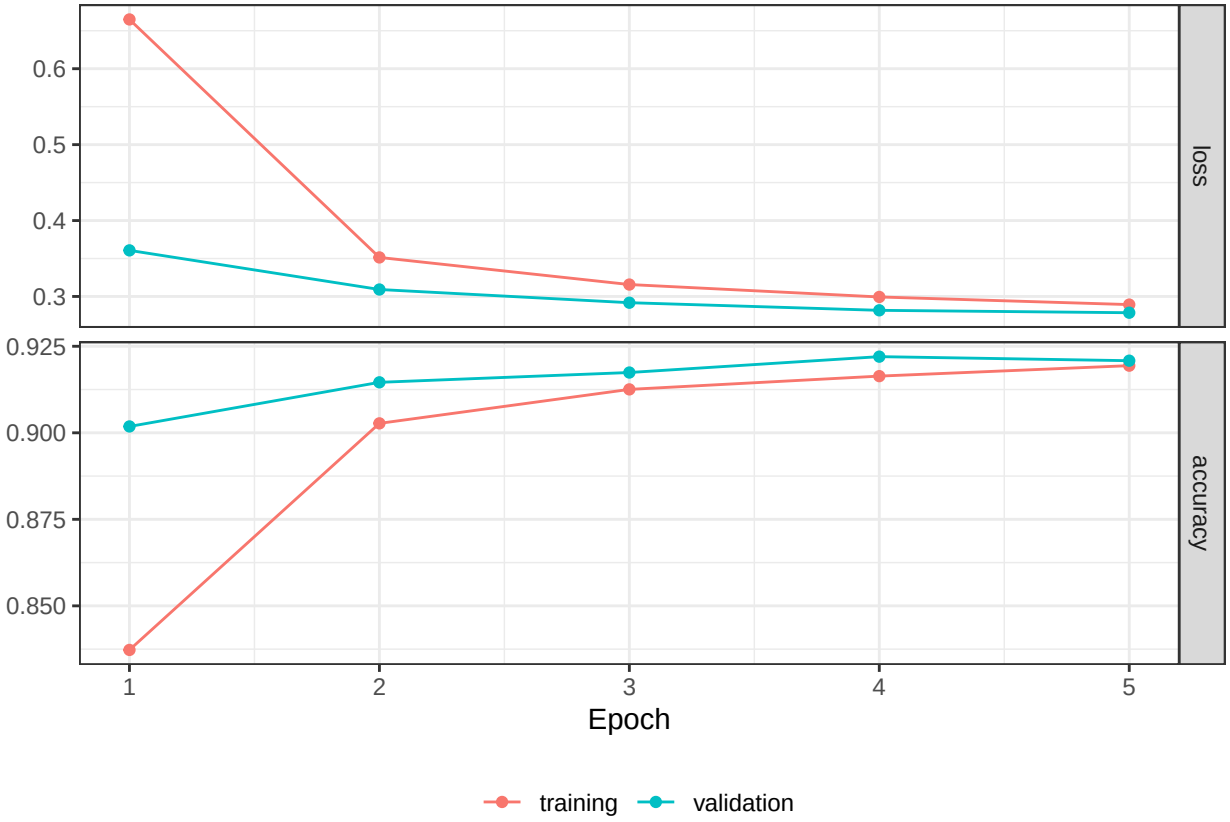
```
    metrics = c("accuracy")             # how to evaluate the fit
  )
```

Finally, we can train the model! Let's use a small number of epochs (gradient steps) so the run time is manageable.

```
model_lr %>%
  fit(
    x_train,                  # supply training features
    y_train,                  # supply training responses
    epochs = 5,               # an epoch is a gradient step
    batch_size = 128,         # we will learn about batches in Lecture 2
    validation_split = 0.2    # use 20% of the training data for validation
  )
```

The field `model_lr$history$history` contains the progress during training, and can be plotted via

```
# plot the history
plot_model_history(model_lr$history$history)
```



We can get the fitted probabilities using the `predict()` function, and extract the classes with highest predicted probability using `k_argmax()`

```
# get fitted probabilities
model_lr %>%
  predict(x_test) %>%
  head()
```

```
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,] 6.489578e-06 8.842616e-10 9.889039e-06 1.964700e-03 2.161905e-06
```

```
## [2,] 6.594752e-04 5.643314e-06 9.924999e-01 2.541800e-04 7.060835e-11
## [3,] 1.330151e-05 9.676043e-01 1.366966e-02 4.375928e-03 3.928219e-04
## [4,] 9.994317e-01 2.122651e-12 8.378268e-05 5.450644e-06 1.099157e-07
## [5,] 6.481899e-04 4.542796e-07 4.989367e-03 1.322164e-04 9.114672e-01
## [6,] 1.031944e-06 9.835294e-01 3.405710e-03 2.569160e-03 5.640183e-05
##              [,6]         [,7]         [,8]         [,9]        [,10]
## [1,] 1.608025e-05 2.192252e-10 9.968402e-01 0.0000172654 1.143324e-03
## [2,] 1.623251e-03 4.818834e-03 6.780349e-14 0.0001386782 1.097427e-10
## [3,] 1.025434e-03 2.417184e-03 1.326759e-03 0.0085058352 6.687793e-04
## [4,] 2.314859e-04 1.224714e-04 4.260944e-05 0.0000464606 3.593506e-05
## [5,] 2.498665e-04 2.098440e-03 4.392126e-03 0.0124051590 6.361686e-02
## [6,] 1.146011e-04 5.028083e-05 3.631603e-03 0.0054497407 1.192118e-03
```

```
# get predicted classes
predicted_classes <- model_lr %>%
  predict(x_test) %>%
  k_argmax() %>%
  as.integer()
head(predicted_classes)
```

```
## [1] 7 2 1 0 4 1
```

We can extract the misclassification error / accuracy manually:

```
# misclassification error
mean(predicted_classes != g_test)
```

```
## [1] 0.0797
```

```
# accuracy
mean(predicted_classes == g_test)
```

```
## [1] 0.9203
```

Or we can use a shortcut and call `evaluate`:

```
evaluate(model_lr, x_test, y_test, verbose = FALSE)
```

```
##      loss  accuracy
## 0.2803867 0.9203000
```

Finally, let's take a look at one of the misclassified digits:

```
misclassifications <- which(predicted_classes != g_test)
idx <- misclassifications[1]
plot_grayscale(x_test[idx, , , ]) +
  ggtitle(sprintf(
    "Predicted class %d; True class %d",
    predicted_classes[idx],
    g_test[idx]
  ))
```

Predicted class 6; True class 5