Unit 5 Lecture 1: Deep Learning Preliminaries

November 18, 2021

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(tidyverse) # for everything else
```

Let's also load some helper functions written for this class:

```
source("../../functions/deep_learning_helpers.R")
```

Next let's load the MNIST handwritten digit data:

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

```
## Loaded Tensorflow version 2.5.0
```

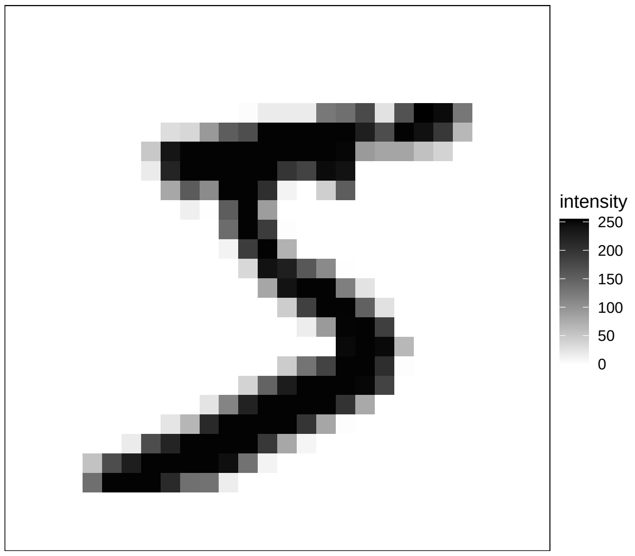
```
# extra train and test data
x_train <- mnist$train$x
g_train <- mnist$train$y
x_test <- mnist$test$x
g_test <- mnist$test$y

# examine dimensions
dim(x_train)
```

```
## [1] 60000    28    28
```

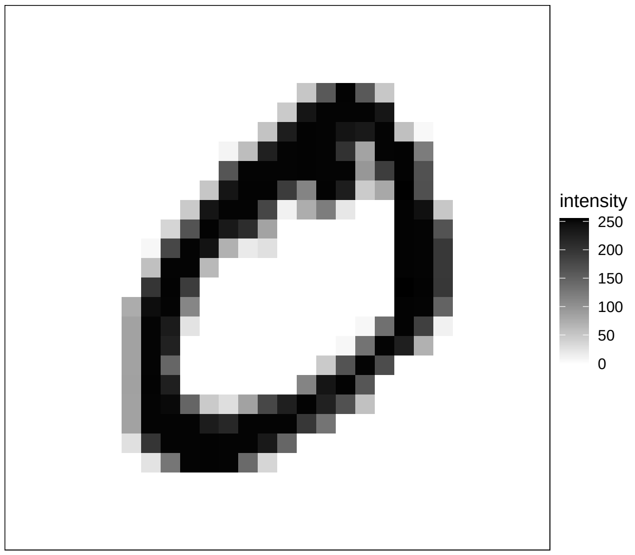
```
dim(x_test)
```

```
## [1] 10000    28    28
```

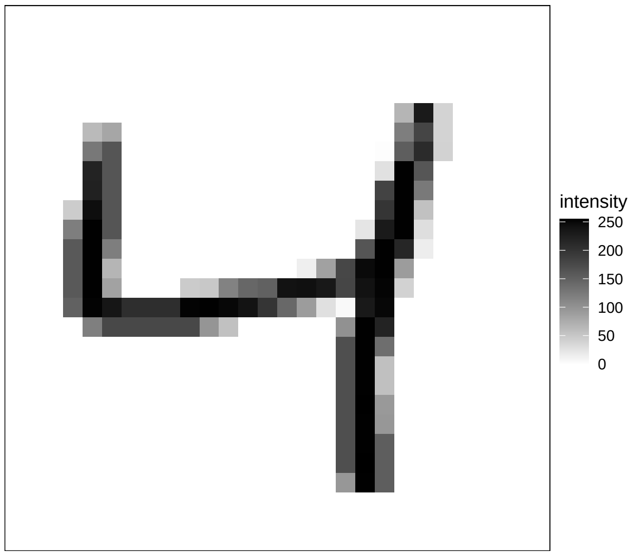
```
# plot a few of the digits
p1 = plot_grayscale(x_train[1,,])
plot(p1)
```



```r
p2 = plot_grayscale(x_train[2,,])
plot(p2)
```



```r
p3 = plot_grayscale(x_train[3,,])
plot(p3)
```



Now we *flatten* the images into vectors of length 784:

```
# define some problem parameters
num_pixels = dim(x_train)[2]*dim(x_train)[3]
num_classes = 10
max_intensity = 255

# flatten training and testing data
x_train <- array_reshape(x_train, c(nrow(x_train), num_pixels))
x_test <- array_reshape(x_test, c(nrow(x_test), num_pixels))

# rescale pixel intensities to the unit interval
x_train <- x_train / max_intensity
x_test <- x_test / max_intensity

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

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

```
modellr <- keras_model_sequential() %>%
  layer_dense(input_shape = num_pixels,  # number of initial inputs
              units = num_classes,       # number of outputs
              activation = "softmax")    # type of activation function
```

We can get a summary of this model as follows:

```
summary(modellr)
```

```
## Model: "sequential"
## ________________________________________________________________________________
## Layer (type)                      Output Shape                    Param #
## ================================================================================
## 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:

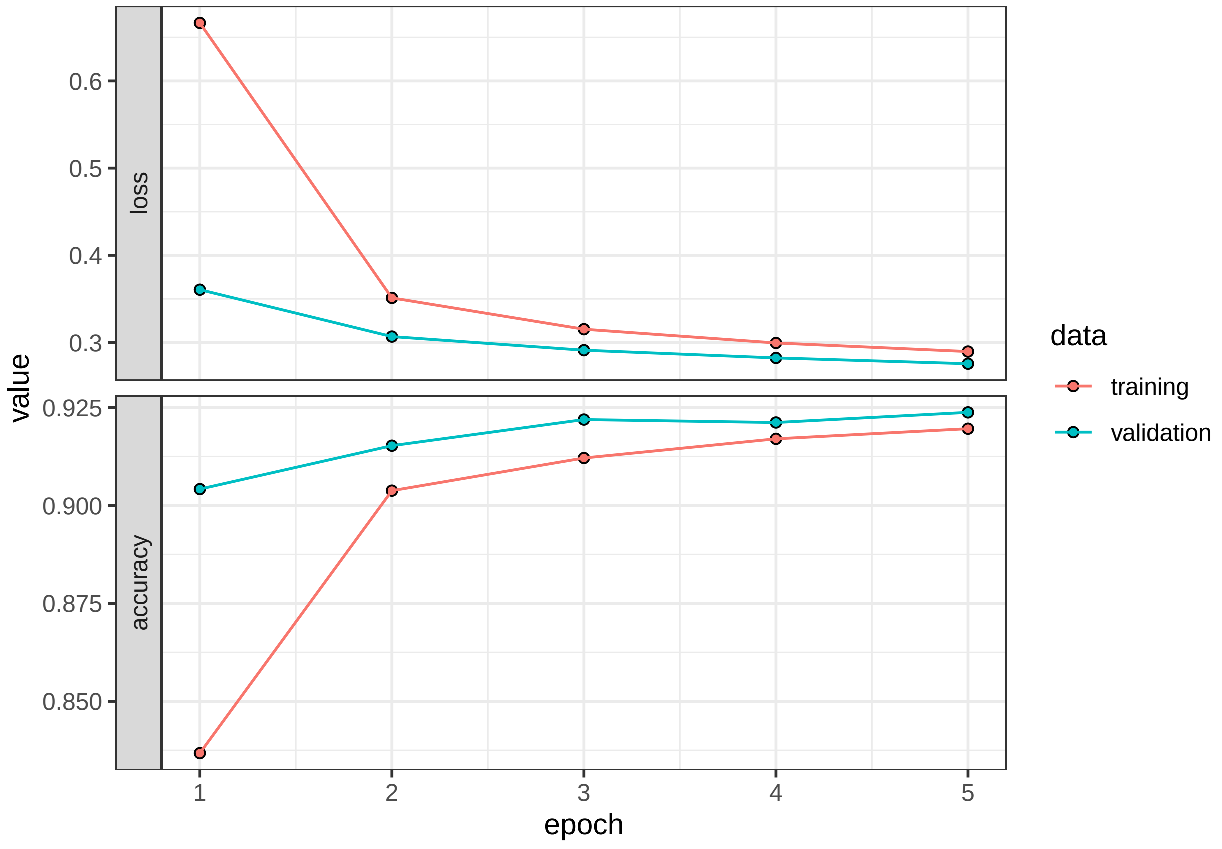
```r
modellr %>%                                       # note: modifying modellr in place
  compile(loss = "categorical_crossentropy",      # which loss to use
          optimizer = optimizer_rmsprop(),        # how to optimize the loss
          metrics = c("accuracy"))                # how to evaluate the fit
```

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

```r
history = modellr %>%
  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 `history` object contains the progress during training, and can be plotted via

```r
# plot the history
plot(history) + geom_line() + theme_bw()
```



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

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

```
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,] 9.082902e-06 6.544177e-10 8.647398e-06 2.242724e-03 1.809079e-06
## [2,] 4.665967e-04 6.477154e-06 9.880613e-01 5.512805e-04 8.326603e-11
## [3,] 7.702241e-06 9.740097e-01 1.138925e-02 3.181868e-03 3.603188e-04
## [4,] 9.991684e-01 2.810844e-12 6.874798e-05 1.406577e-05 1.530030e-07
## [5,] 7.889891e-04 5.521042e-07 4.804351e-03 1.298161e-04 9.104623e-01
## [6,] 7.182104e-07 9.865361e-01 3.357757e-03 2.281207e-03 6.697823e-05
##              [,6]         [,7]         [,8]         [,9]        [,10]
## [1,] 1.935877e-05 2.465952e-10 9.966018e-01 2.331144e-05 1.093332e-03
## [2,] 1.578730e-03 9.206007e-03 5.347885e-14 1.296008e-04 7.256735e-11
## [3,] 8.184360e-04 2.044107e-03 7.425547e-04 7.008764e-03 4.371786e-04
## [4,] 3.731773e-04 2.194146e-04 5.696304e-05 6.574884e-05 3.342310e-05
## [5,] 3.797947e-04 3.611429e-03 6.442858e-03 1.295052e-02 6.042946e-02
## [6,] 8.854746e-05 4.801165e-05 2.060872e-03 4.767334e-03 7.925237e-04
```

```r
# get predicted classes
predicted_classes = modellr %>% 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:

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

```
## [1] 0.0779
```

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

```
## [1] 0.9221
```

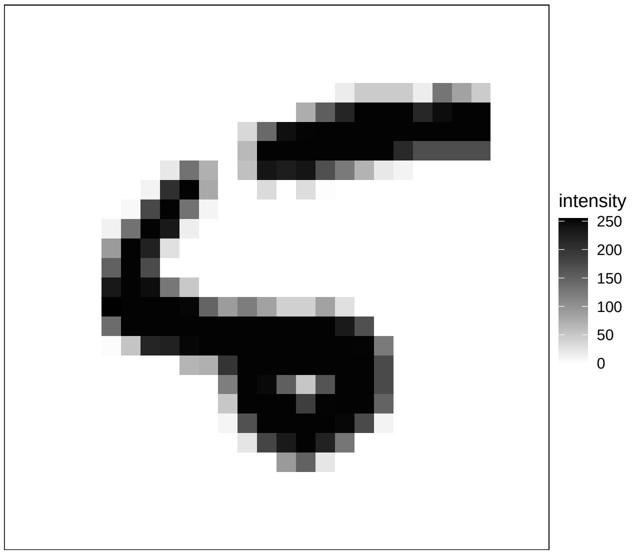
Or we can use a shortcut and call `evaluate`:

```r
evaluate(modellr, x_test, y_test, verbose = FALSE)
```

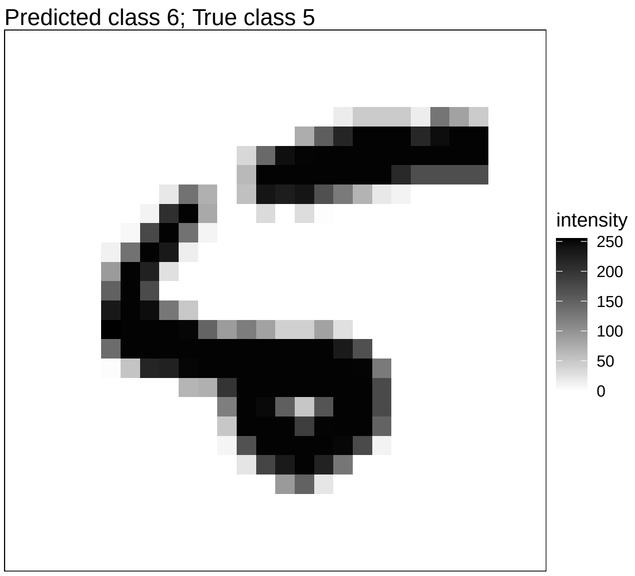
```
##      loss  accuracy
## 0.2794005 0.9221000
```

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

```r
misclassifications = which(predicted_classes != g_test)
idx = misclassifications[1]
plot_grayscale(mnist$test$x[idx,,])
```



```r
plot_grayscale(mnist$test$x[idx,,]) +
  ggtitle(sprintf("Predicted class %d; True class %d",
                  predicted_classes[idx],
                  g_test[idx]))
```


Predicted class 6; True class 5
intensity