R Lab: Fundamentals of Neural Networks

In this lab, we discuss some of the fundamentals of neural networks using Keras and TensorFlow. The deliverable for this lab is a Word or PDF file containing responses to the exercises at the end of the lab. The deliverable should be submitted through Canvas.

Load Packages and Virtual Environment

We first start off with loading the R packages that we will be using throughout the lab. These packages include reticulate, tensorflow, and keras. The first package, reticulate, is needed to access the virtual environment (which contains the Python interpreter) we created in the first lab. The last two packages, tensorflow and keras, provide the R interfaces for the TensorFlow and Keras Python packages, respectively. The function 'use_virtualenv' is used to open the virtual environment we created in the first lab containing the Python interpreter.

```r
library(reticulate)
library(tensorflow)
library(keras)

use_virtualenv("my_tf_workspace", required = TRUE)
```

Back to Our First Keras Example

We return to our Keras example from the first lab, which uses the pre-installed mtcars data set to build a neural network model for predicting the miles per gallon of a vehicle, given its number of cylinders, displacement, and horsepower.

As in the first lab, we load the mtcars data set into the current environment and specify the data frame that contains the features (i.e., the independent variables) to be used for predicting the label (i.e., the dependent variable). We also convert this data frame to an array (also called a tensor), which is the data structure that Keras and TensorFlow functions expect. Finally, we specify the vector (which is also an array/tensor) of labels for each observation.

```r
mtcars <- mtcars

mtcars_x <- mtcars[, c("cyl", "disp", "hp")]
mtcars_x <- array(data = unlist(mtcars_x),
                  dim = c(32, 3),
                  dimnames = list(rownames(mtcars_x),
                                  colnames(mtcars_x)))

mtcars_y <- mtcars[, "mpg"]
```

We start off with the same architecture of the neural network model used in the first lab, which is a perceptron that contains only an input layer and an output layer. Although a perceptron is the simplest architecture for a neural network, it teaches us two important aspects of defining neural network architectures in Keras: 1) the input layer is not explicitly defined - the 'input_shape' argument of the first layer is used to specify the shape of the input, and 2) the last layer we define is the output layer, so the output of the last layer must correspond to the output we expect for our particular problem.

In our case, the input is three numerical values: the number of cylinders, displacement, and horsepower of the vehicle, so we specify 'input_shape=3'. Furthermore, we are predicting the miles per gallon of a vehicle, which is one numerical value. Thus, the last layer should only have one unit and use the 'linear' activation function.

```
nn_model <- keras_model_sequential() %>%
  layer_dense(units = 1, input_shap = 3, activation = "linear")
```

After specifying the architecture of the neural network, we can view its structure, which shows the shape of the output of each layer, as well as the number of weights ('Param #').

```
nn_model
```

```
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## ================================================================================
##  dense (Dense)                      (None, 1)                        4
## ================================================================================
## Total params: 4
## Trainable params: 4
## Non-trainable params: 0
## ________________________________________________________________________________
```

The number of weights in the only layer defined (which is the output layer of the perceptron) is 4. These weights correspond to the connections between the input layer with three units (remember, 'input_shape=3') and the output layer with one unit: three weights for the connections and one weight for the bias (3+1=4).

Let's add a hidden layer to the architecture of the neural network and use it to redefine 'nn_model'. This hidden layer has two units and uses the 'relu' activation function. Note, this activation function adds non-linearity to the model! Also note, this hidden layer is now the first layer defined in our Keras model, so we specify the 'input_shape' in it.

```
nn_model <- keras_model_sequential() %>%
  layer_dense(units = 2, input_shape = 3, activation = "relu") %>%
  layer_dense(units = 1, activation = "linear")
```

As for the perceptron, we can view its structure and the number of weights.

```
nn_model
```

```
## Model: "sequential_1"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## ================================================================================
##  dense_2 (Dense)                    (None, 2)                        8
##  dense_1 (Dense)                    (None, 1)                        3
## ================================================================================
## Total params: 11
## Trainable params: 11
## Non-trainable params: 0
## ________________________________________________________________________________
```

The number of weights in the first layer defined is now 8. These weights correspond to the connections between the input layer with three units and the hidden layer with two units: 3x2=6 weights for the connections and two weights for the bias $(6 + 2 = 8)$.

Having specified the architecture of the neural network model, we now specify the optimization algorithm and loss that will be used to find the best values of the weights based on 'mtcars_x' and 'mtcars_y'. The purpose of the optimizer and loss function can be seen in Figure 1.9 of our textbook "Deep Learning with R". In particular, the loss function provides a measure of how well the predictions for the observations in 'mtcars_x' given by the neural network model match the true labels in 'mtcars_y'. The optimization algorithm uses this measurement to decide how to change the values of the weights to achieve better performance in terms of loss. In addition to the loss and optimizer, we can also select the metrics to output during the model-building process. Mean absolute error is a common choice for the regression metric, which is in the same measurement unit as the output. You may be wondering: why don't we use the same metric (i.e., 'mean_absolute_error') for the loss. Well, we need the loss to be differentiable; this will become clearer throughout the textbook readings. Once the loss, optimization algorithm, and metrics are specified, we compile the model using the 'compile' function.

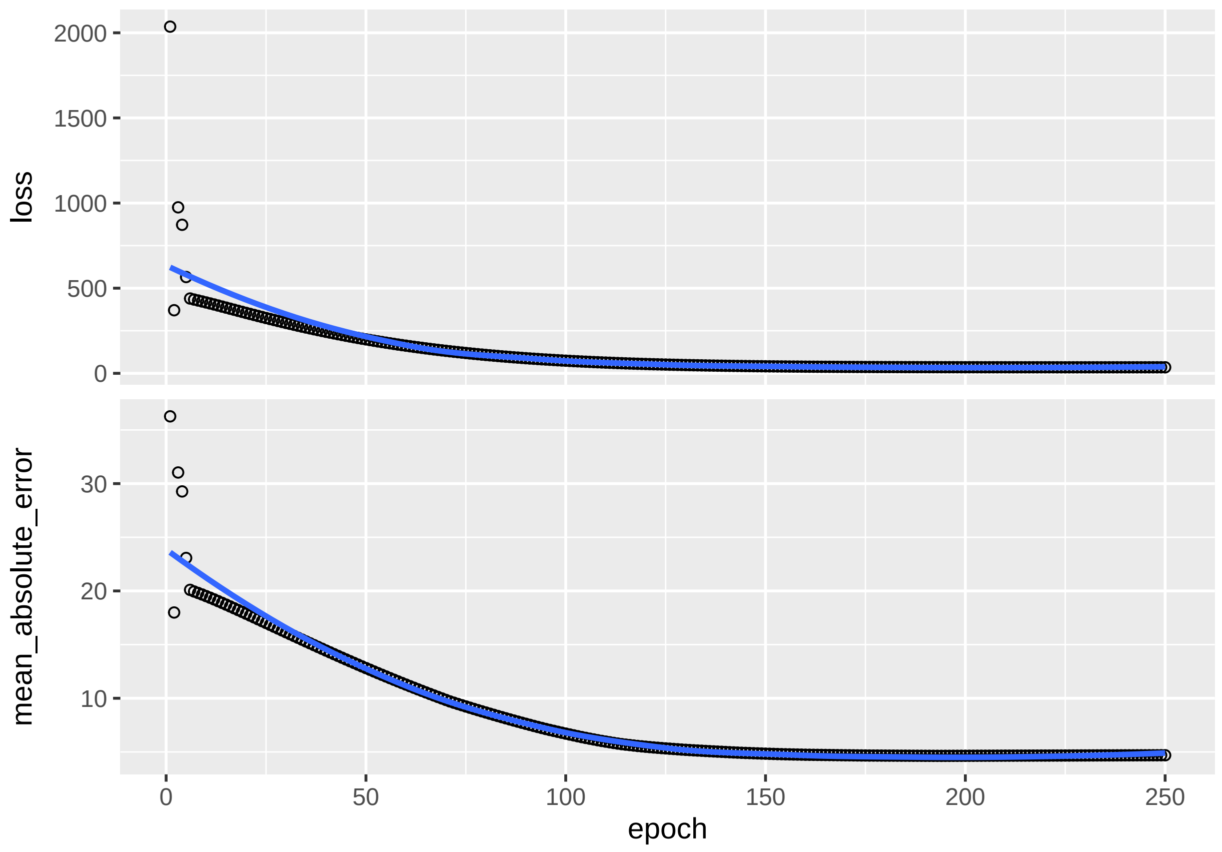
```
nn_model %>% compile(optimizer = optimizer_adam(learning_rate = 0.2),
                     loss = "mean_squared_error",
                     metrics = "mean_absolute_error")
```

Now, we are ready to train the neural network model on the 'mtcars_x' and 'mtcars_y' data using the 'fit' function.

```
nn_model_training <- nn_model %>% fit(x = mtcars_x,
                                      y = mtcars_y,
                                      epochs = 250,
                                      verbose = FALSE)
```

Plotting the 'nn_model_training' variable shows how the training process proceeded in terms of the loss ('mean_squared_error') and the metric ('mean_absolute_error'). Note that both of these values continue to decrease as the learning process proceeds. These curves are called the learning curves. Note, the learning curves in your output may be different than the learning curves shown in the output below. This is normal, since there are random initializations and values that are used in several Keras functions.

```
plot(nn_model_training)
```



We can obtain the fitted weights using the Keras 'get_weights' function. Note, as for the learning curves above, the weights in your output may also be different than the weights shown in the output below.

```
get_weights(nn_model)
```

```
## [[1]]
##           [,1]       [,2]
## [1,] -1.294023 -0.4214823
## [2,] -1.479654 -0.5245135
## [3,] -0.935578  0.2305309
##
## [[2]]
## [1] -2.125417  0.000000
##
## [[3]]
##             [,1]
## [1,] -0.14516281
## [2,] -0.02217257
##
## [[4]]
## [1] 19.91286
```

We can also make predictions for new vehicles using the 'predict' function.

```
prediction <- predict(nn_model, array(c(6, 350, 125), dim = c(1, 3)))

prediction
```

```
##          [,1]
## [1,] 19.91286
```

This vehicle has a predicted miles per gallon of 19.9128647.

Exercises

1) Using the neural network above, what is the predicted miles per gallon of a vehicle with 8 cylinders, a displacement of 250, and a horsepower of 200?
2) What loss function did we specify when building the neural network model above?
3) Define the rank of a tensor. What is the rank of a matrix (remember, a matrix is a tensor of a particular rank)?
4) The data we will be working with throughout the course, including in this lab, is vector data. What is the tensor rank of vector data?
5) Consider the following tensor of ones: array(1, dim = c(500, 256, 256, 3)). What is its rank? What is its shape? What is the dimension of the second axis? If this tensor represented a collection of color images, how many images are there?