R Lab: Installing Keras and TensorFlow

In this lab, the steps for installing Keras and TensorFlow, the two machine learning environments that will be used throughout the course, are provided. The deliverable for this lab is a Word or PDF file containing your responses to the exercises at the end of the lab. The deliverable should be submitted through Canvas.

There are two options for using Keras and TensorFlow in this course. The first option is to install both environments locally on your machine. The second option is to use the shared workspace for the course on Posit (formerly RStudio) Cloud, which already has Keras and TensorFlow installed and loaded. The preferred method is the first (to install locally on your machine), since you can continue using them after the course has ended.

Option 1: Install and Use on Local Machine

Run the following code in RStudio to install Keras and TensorFlow on your local machine.

```
# download Rtools from https://cran.r-project.org/bin/windows/Rtools/
# download Python 3.10 from https://www.python.org/downloads/release/python-3109/
# select Windows installer (64-bit)
install.packages("reticulate")
install.packages("keras")
install.packages("tensorflow")

library(reticulate)
# for the 'python' argument in the following function virtualenv_create(), specify the
# path to python.exe that was installed above
# on a Windows machine, the path to python.exe should look similar to the path below
virtualenv_create("my_tf_workspace",
    python = 'C:\\Users\\peter\\AppData\\Local\\Programs\\Python\\Python310\\python.exe')

library(tensorflow)
install_tensorflow(envname = "my_tf_workspace", version = "2.9-cpu")
```

To ensure that TensorFlow has been successfully installed, run the following code. Note, any time you want to use Keras on your local machine, the virtual environment created above ('my_tf_workspace') needs to be specified and used, as in the third line of the following code. Also, the 'reticulate' and 'tensorflow' packages need to be loaded again, since the R session was restarted.

```
library(reticulate)
library(tensorflow)
use_virtualenv("my_tf_workspace")
tf$constant("Hello Tensorflow!")
```

```
## tf.Tensor(b'Hello Tensorflow!', shape=(), dtype=string)
```

Keras and TensorFlow have now been successfully installed. Take a screenshot of the output of the code directly above and submit it for Exercise 2 below.

Option 2: Use Shared Workspace on Posit Cloud

If Option 1 above does not work, then another option is to use the shared workspace on Posit Cloud. Send an email to let me know that Option 1 did not work on your local machine and I will send an invite to use the shared workspace. Once you are logged in to the shared workspace on Posit Cloud, a new RStudio project can be created by clicking the 'New Project' drop-down menu in the top right-hand corner and selecting 'New RStudio Project'.

Once a new RStudio project has been created, run the following code. Note, any time you want to use Keras and TensorFlow in the shared workspace, the virtual environment 'my_tf_workspace' needs to be loaded and used, as in the third line of the following code.

```
library(reticulate)
library(tensorflow)
use_virtualenv("my_tf_workspace", required = TRUE)
```

```
tf$constant("Hello Tensorflow!")
```

```
## tf.Tensor(b'Hello Tensorflow!', shape=(), dtype=string)
```

Take a screenshot of the output of the code directly above and submit it for Exercise 2 below.

First Keras Example

Now that Keras and TensorFlow are available, we can build our first neural network model. This example uses the pre-installed mtcars data set to build a neural network model for predicting the miles per gallon of a car, given its number of cylinders, displacement, and horsepower.

We first load the R 'keras' library and the mtcars data set into the current environment. We then 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). Note, Keras and TensorFlow functions cannot take data frames as input, so we need to convert the data frame to an array, which are structures that will be discussed in more detail throughout the course. Finally, we specify the vector of labels for each observation.

```
library(keras)
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 are now ready to specify the architecture of the neural network model, which will only include an input layer and an output layer (i.e., the neural network will be a perceptron). Note, we do not explicitly specify the input layer - we only need to specify the argument 'input_shape' for the first layer. Also, since we are not using any hidden layers, the first (and only) layer is also the output layer. In general, the last layer specified for the architecture of a neural network in Keras is the output layer.

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

After specifying the architecture of the neural network, we then specify the optimization algorithm, the learning rate, and the loss function, and then fit the neural network using the data in 'mtcars_x' and 'mtcars_y'. Also, the 'epochs' argument will be discussed in more detail throughout the course, so it is okay to disregard for now.

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

nn_model_training <- nn_model %>% fit(x = mtcars_x,
                                      y = mtcars_y,
                                      epochs = 10000,
                                      verbose = FALSE)
```

We can obtain the fitted weights using the Keras 'get_weights' function. Note, the weights in your output may be slightly different than the weights shown in the output below. This is normal, since there are random initializations and values that are used in several Keras functions.

```
get_weights(nn_model)
```

```
## [[1]]
##              [,1]
## [1,] -1.22742236
## [2,] -0.01883977
## [3,] -0.01468102
##
## [[2]]
## [1] 34.18492
```

The neural network model that we trained above is a linear regression model written in perceptron form, as discussed in the first live session. We can check this by comparing the weights from the trained neural network model above to the coefficients calculated using the 'lm' function, which is the base R function for implementing linear regression.

```
lr_model <- lm(mpg ~ cyl + disp + hp, data = mtcars)
lr_model$coefficients
```

```
## (Intercept)         cyl        disp          hp
## 34.18491917 -1.22741994 -0.01883809 -0.01467933
```

The weights from the neural network model and the coefficients from the linear regression model are indeed the same!

Exercises

1) What is the main difference between supervised and unsupervised learning? Was the type of learning performed in the 'First Keras Example' section supervised or unsupervised learning? Why?
2) Take a screenshot of the output of 'tf$constant("Hello Tensorflow!")' as described above.
3) Briefly describe the fields of machine learning and deep learning, as well as the main difference(s) between both fields.
4) What type of transformation is performed by each layer in a neural network? What is the purpose of the loss function in training a neural network?