Week1_HomeworkSimonsen

Steven Simonsen

2024-08-30

```
#install.packages("reticulate")
#install.packages("keras")
#install.packages("tensorflow")

library(reticulate)

virtualenv_create("my_tf_workspace",
python = 'C:\\Users\\steve\\AppData\\Local\\Programs\\Python\\Python312\\python.exe')
```

```
## virtualenv: my_tf_workspace
```

```
library(tensorflow)
install_tensorflow(envname = "my_tf_workspace", version = "2.17.0-cpu") #I used
```

```
## Using virtual environment "my_tf_workspace" ...
```

```
## + "C:/Users/steve/OneDrive/Documents/.virtualenvs/my_tf_workspace/Scripts/python.exe" -m pip install
```

```
##
## Installation complete.
```

```
#an updated version since I already had python 3.12 installed.

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

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

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

#First Keras Example

```
library(keras3) #Given I had to use an updated version of tensorflow, I also
```

```
##
## Attaching package: 'keras3'
```

```
## The following objects are masked from 'package:tensorflow':
##
##     set_random_seed, shape
```

```
#had to use keras3 instead of keras to avoid positional argument errors.

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"]


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


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)

get_weights(nn_model)
```

```
## [[1]]
##              [,1]
## [1,] -1.22742271
## [2,] -0.01883848
## [3,] -0.01467975
##
## [[2]]
## [1] 34.18493
```

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

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

#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?

The main difference between supervised and unsupervised learning is that supervised learning has a dependent variable, or label, that that the model is predicting. Supervised learning can include both regression and classification learning.

Unsupervised learning does not have a label, or an expected result. Rather, unsupervised learning identifies relationships and trends within the various data points.

The first Keras example is supervised learning, because we have a label, mpg, we are trying to predict based on various predictors. Given that the output is numeric, the model works to solve a regression problem.

2) Take a screenshot of the output of 'tf$constant("Hello Tensorflow!")' as described above.

Here is the output from the console: > library(reticulate) > library(tensorflow) > use_virtualenv("my_tf_workspace") > tf$constant("Hello Tensorflow!") tf.Tensor(b'Hello Tensorflow!', shape=(), dtype=string)

3) Briefly describe the fields of machine learning and deep learning, as well as the main difference(s) between both fields.

Machine learning and deep learning are different, yet related fields. Machine learning is the study of getting machines to learn from data. Deep learning extends the boundaries of machine learning by creating neural networks with many layers. Both fields are a part of Artificial Intelligence, which is the study of getting machines to perform human tasks.

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?

In the initial input layer, transformations are not typically applied, as this is the data from the originating data set. After that, the data is passed to a single, or various hidden layers within the model. Within the hidden layers, both linear and non-linear transformations occur. Both the use of linear and non-linear transformations help to produce better representations of the data within each hidden layer. In other words, this allows the network to learn and represent complex relationships within the data, and each hidden layer can use a different activation function to represent the data. Finally, the output layer produces the final prediction or output. Here, either linear or non-linear (ex: softmax) transformations are performed, and a final activation function is utilized to predict the final output.

The purpose of the loss function is to quantify the error identified between the predicted outputs and the actual output values. The loss function also influences the optimization, or the weights provided within the model at each hidden layer to minimize the loss. There are different loss algorithms that can be utilized, and each influences the way the model minimizes losses in different ways.