IBM Developer
SKILLS NETWORK

LINEAR REGRESSION WITH TENSORFLOW

LINEAR REGRESSION WITH TENSORFLOW

Objective for this Notebook

1. What is Linear Regression
2. Linear Regression with TensorFlow.

In this notebook we will overview the implementation of Linear Regression with TensorFlow

Table of Contents

Linear Regression

Defining a linear regression in simple terms, is the approximation of a linear model used to describe the relationship between two or more variables. In a simple linear regression there are two variables, the dependent variable, which can be seen as the "state" or "final goal" that we study and try to predict, and the independent variables, also known as explanatory variables, which can be seen as the "causes" of the "states".

When more than one independent variable is present the process is called multiple linear regression.
When multiple dependent variables are predicted the process is known as multivariate linear regression.

The equation of a simple linear model is

$$Y = aX + b$$

Where Y is the dependent variable and X is the independent variable, and a and b being the parameters we adjust. a is known as "slope" or "gradient" and b is the "intercept". You can interpret this equation as Y being a function of X, or Y being dependent on X.

If you plot the model, you will see it is a line, and by adjusting the "slope" parameter you will change the angle between the line and the independent variable axis, and the "intercept parameter" will affect where it crosses the dependent variable's axis.

We begin by installing TensorFlow version 2.2.0 and its required prerequistes.

[ ]:
```
!pip install grpcio==1.24.3
!pip install tensorflow==2.2.0
```

Restart kernel for latest version of TensorFlow to be activated

Next, let's first import the required packages:

[ ]:
```
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import numpy as np
import tensorflow as tf
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (10, 6)
```

[ ]:
```
if not tf.__version__ == '2.2.0':
    print(tf.__version__)
    raise ValueError('please upgrade to TensorFlow 2.2.0, or restart your Kernel (Kernel->Restart & Clear Output)')
```

IMPORTANT! => Please restart the kernel by clicking on "Kernel"->"Restart and Clear Outout" and wait until all output disapears. Then your changes are beeing picked up

Let's define the independent variable:

```
X = np.arange(0.0, 5.0, 0.1)
X
```

```
##You can adjust the slope and intercept to verify the changes in the graph
a = 1
b = 0

Y= a * X + b

plt.plot(X, Y)
plt.ylabel('Dependent Variable')
plt.xlabel('Indepdendent Variable')
plt.show()
```

OK... but how can we see this concept of linear relations with a more meaningful point of view?

Simple linear relations were used to try to describe and quantify many observable physical phenomena, the easiest to understand are speed and distance traveled:

$$DistanceTraveled = Speed$$
$$timesTime + InitialDistance$$

$$Speed = Acceleration$$
$$timesTime + InitialSpeed$$

They are also used to describe properties of different materials:

$$Force = Deformation$$
$$timesStiffness$$

$$HeatTransfered = TemperatureDifference$$
$$timesThermalConductivity$$

$$ElectricalTension(Voltage) = ElectricalCurrent$$
$$timesResistance$$

$$Mass = Volume$$
$$timesDensity$$

When we perform an experiment and gather the data, or if we already have a dataset and we want to perform a linear regression, what we will do is adjust a simple linear model to the dataset, we adjust the "slope" and "intercept" parameters to the data the best way possible, because the closer the model comes to describing each ocurrence, the better it will be at representing them.

So how is this "regression" performed?

Linear Regression with TensorFlow

A simple example of a linear function can help us understand the basic mechanism behind TensorFlow.
For the first part we will use a sample dataset, and then we'll use TensorFlow to adjust and get the right parameters. We download a dataset that is related to fuel consumption and Carbon dioxide emission of cars.

```
!wget -O FuelConsumption.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/FuelConsumptionCo2.csv
```

Understanding the Data

`FuelConsumption.csv`:

We have downloaded a fuel consumption dataset, `FuelConsumption.csv`, which contains model-specific fuel consumption ratings and estimated carbon dioxide emissions for new light-duty vehicles for retail sale in Canada. Dataset source

- **MODELYEAR** e.g. 2014
- **MAKE** e.g. Acura
- **MODEL** e.g. ILX
- **VEHICLE CLASS** e.g. SUV
- **ENGINE SIZE** e.g. 4.7
- **CYLINDERS** e.g 6
- **TRANSMISSION** e.g. A6
- **FUEL CONSUMPTION in CITY(L/100 km)** e.g. 9.9
- **FUEL CONSUMPTION in HWY (L/100 km)** e.g. 8.9
- **FUEL CONSUMPTION COMB (L/100 km)** e.g. 9.2
- **CO2 EMISSIONS (g/km)** e.g. 182 --> low --> 0


```
df = pd.read_csv("FuelConsumption.csv")

# take a look at the dataset
df.head()
```

Lets say we want to use linear regression to predict Co2Emission of cars based on their engine size. So, lets define X and Y value for the linear regression, that is, train_x and train_y:

```
train_x = np.asanyarray(df[['ENGINESIZE']])
train_y = np.asanyarray(df[['CO2EMISSIONS']])
```

First, we initialize the variables a and b, with any random guess, and then we define the linear function:

```
a = tf.Variable(20.0)
b = tf.Variable(30.2)


def h(x):
   y = a*x + b
   return y
```

Now, we are going to define a loss function for our regression, so we can train our model to better fit our data. In a linear regression, we minimize the squared error of the difference between the predicted values(obtained from the equation) and the target values (the data that we have). In other words we want to minimize the square of the predicted values minus the target value.

So we define the equation to be minimized as loss.

To find value of our loss, we use tf.reduce_mean(). This function finds the mean of a multidimensional tensor, and the result can have a different dimension.

```
def loss_object(y,train_y) :
    return tf.reduce_mean(tf.square(y - train_y))
    # Below is a predefined method offered by TensorFlow to calculate loss function
    #loss_object = tf.keras.losses.MeanSquaredLogarithmicError()
```

Now we are ready to start training and run the graph. We use GradientTape to calculate gradients:

```
learning_rate = 0.01
train_data = []
loss_values =[]
# steps of looping through all your data to update the parameters
training_epochs = 200

# train model
for epoch in range(training_epochs):
    with tf.GradientTape() as tape:
        y_predicted = h(train_x)
        loss_value = loss_object(train_y,y_predicted)
        loss_values.append(loss_value)

        # get gradients
        gradients = tape.gradient(loss_value, [b,a])

        # compute and adjust weights
        b.assign_sub(gradients[0]*learning_rate)
        a.assign_sub(gradients[1]*learning_rate)
        if epoch % 5 == 0:
            train_data.append([a.numpy(), b.numpy()])
```

Lets plot the loss values to see how it has changed during the training:

```
plt.plot(loss_values, 'ro')
```

Lets visualize how the coefficient and intercept of line has changed to fit the data:

```
cr, cg, cb = (1.0, 1.0, 0.0)
for f in train_data:
    cb += 1.0 / len(train_data)
    cg -= 1.0 / len(train_data)
    if cb > 1.0: cb = 1.0
    if cg < 0.0: cg = 0.0
    [a, b] = f
    f_y = np.vectorize(lambda x: a*x + b)(train_x)
    line = plt.plot(train_x, f_y)
    plt.setp(line, color=(cr,cg,cb))

plt.plot(train_x, train_y, 'ro')
green_line = mpatches.Patch(color='red', label='Data Points')

plt.legend(handles=[green_line])

plt.show()
```

Want to learn more?

Running deep learning programs usually needs a high performance platform. **PowerAI** speeds up deep learning and AI. Built on IBM's Power Systems, **PowerAI** is a scalable software platform that accelerates deep learning and AI with blazing performance for individual users or enterprises. The **PowerAI** platform supports popular machine learning libraries and dependencies including TensorFlow, Caffe, Torch, and Theano. You can use PowerAI on IMB Cloud.

Also, you can use **Watson Studio** to run these notebooks faster with bigger datasets.**Watson Studio** is IBM's leading cloud solution for data scientists, built by data scientists. With Jupyter notebooks, RStudio, Apache Spark and popular libraries pre-packaged in the cloud, **Watson Studio** enables data scientists to collaborate on their projects without having to install anything. Join the fast-growing community of **Watson Studio** users today with a free account at Watson Studio.This is the end of this lesson. Thank you for reading this notebook, and good luck on your studies.

Thanks for completing this lesson!

If you are familiar with some of these methods and concepts, this tutorial might have been boring for you, but it is important to get used to the TensorFlow mechanics, and feel familiar and comfortable using it, so you can build more complex algorithms in it.

Created by Romeo Kienzler , Saeed Aghabozorgi , Rafael Belo Da Silva

Updated to TF 2.X by Samaya Madhavan

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-09-21	2.0	Srishti	Migrated Lab to Markdown and added to course repo in GitLab

---

Si ynb