ML0120EN-1.2-Review-LinearRegressionwithTensorFlow

January 18, 2020

LINEAR REGRESSION WITH TENSORFLOW

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

Table of Contents

Linear Regression

Linear Regression with TensorFlow

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.

Let's first import the required packages:

[1]:
```
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)
```

Let's define the independent variable:

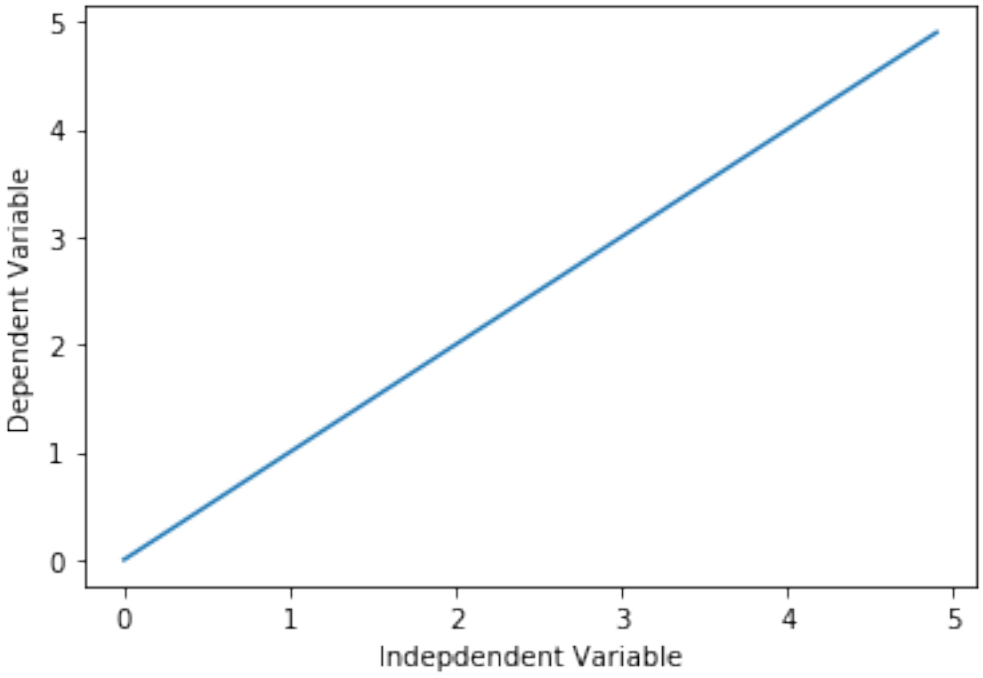
[2]:
```
X = np.arange(0.0, 5.0, 0.1)
X
```

[2]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
       1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5,
       2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8,
       3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9])
```

[3]:
```
##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:

They are also used to describe properties of different materials:

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.

[4]:
```
!wget -O FuelConsumption.csv https://s3-api.us-geo.objectstorage.softlayer.net/
↪cf-courses-data/CognitiveClass/ML0101ENv3/labs/FuelConsumptionCo2.csv
```

```
--2020-01-18 22:05:26--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-
courses-data/CognitiveClass/ML0101ENv3/labs/FuelConsumptionCo2.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-
geo.objectstorage.softlayer.net)… 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-
geo.objectstorage.softlayer.net)|67.228.254.196|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 72629 (71K) [text/csv]
Saving to: 'FuelConsumption.csv'

FuelConsumption.csv 100%[===================>]  70.93K  --.-KB/s    in 0.06s

2020-01-18 22:05:26 (1.09 MB/s) - 'FuelConsumption.csv' saved [72629/72629]
```

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

[5]:
```
df = pd.read_csv("FuelConsumption.csv")

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

[5]:

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	\
0	2014	ACURA	ILX	COMPACT	2.0	4	
1	2014	ACURA	ILX	COMPACT	2.4	4	
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	

	TRANSMISSION	FUELTYPE	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	\
0	AS5	Z	9.9	6.7	
1	M6	Z	11.2	7.7	
2	AV7	Z	6.0	5.8	
3	AS6	Z	12.7	9.1	
4	AS6	Z	12.1	8.7	

	FUELCONSUMPTION_COMB	FUELCONSUMPTION_COMB_MPG	CO2EMISSIONS
0	8.5	33	196
1	9.6	29	221
2	5.9	48	136
3	11.1	25	255
4	10.6	27	244

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:

[6]:
```
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:

[7]:
```
a = tf.Variable(20.0)
b = tf.Variable(30.2)
y = a * train_x + b
```

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.

[8]:
```
loss = tf.reduce_mean(tf.square(y - train_y))
```

Then, we define the optimizer method. The gradient Descent optimizer takes in parameter: learning rate, which corresponds to the speed with which the optimizer should learn; there are pros and cons for increasing the learning-rate parameter, with a high learning rate the training model converges quickly, but there is a risk that a high learning rate causes instability and the model will not converge. Please feel free to make changes to learning parameter and check its effect. On the other hand decreasing the learning rate might reduce the convergence speed, but it would increase the chance of converging to a solution. You should note that the solution might not be a global optimal solution as there is a chance that the optimizer will get stuck in a local optimal solution. Please review other material for further information on the optimization. Here we will use a simple gradient descent with a learning rate of 0.05:

[9]:
```
optimizer = tf.train.GradientDescentOptimizer(0.05)
```

Now we will define the training method of our graph, what method we will use for minimize the loss? We will use the .minimize() which will minimize the error function of our optimizer, resulting in a better model.

[10]:
```
train = optimizer.minimize(loss)
```

Don't forget to initialize the variables before executing a graph:

[11]:
```
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
```

Now we are ready to start the optimization and run the graph:

[12]:
```
loss_values = []
train_data = []
for step in range(100):
    _, loss_val, a_val, b_val = sess.run([train, loss, a, b])
    loss_values.append(loss_val)
    if step % 5 == 0:
        print(step, loss_val, a_val, b_val)
        train_data.append([a_val, b_val])
```

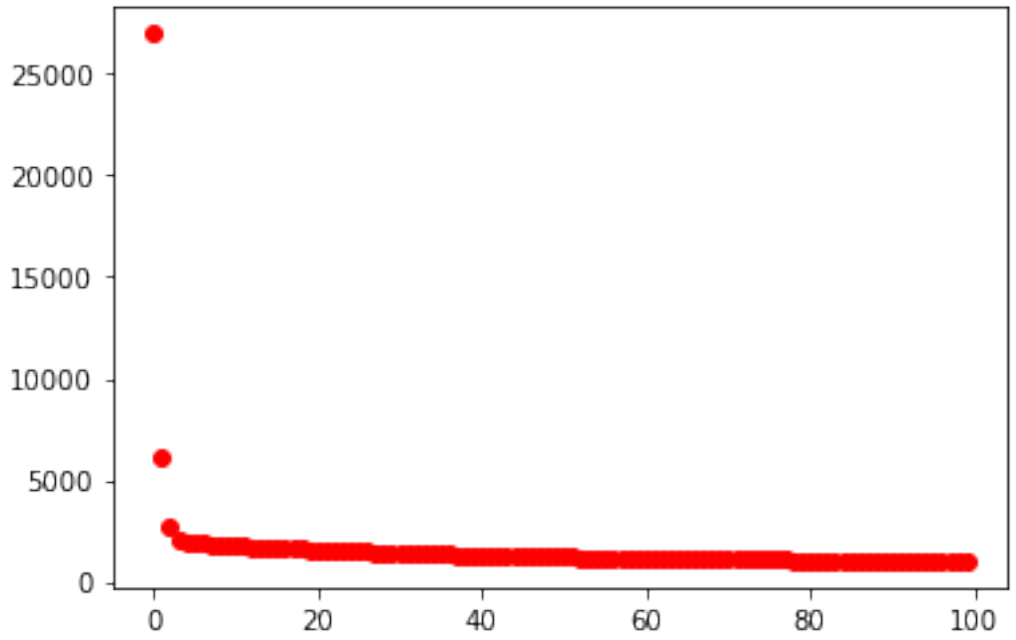
```
0 26992.594 77.07106 46.110275
5 1891.7205 58.84462 47.59573
10 1762.7241 57.65104 53.019833
15 1653.5897 56.36652 58.023922
20 1559.0441 55.172844 62.68204
25 1477.1372 54.061794 67.01765
```

```
30 1406.179 53.027664 71.05309
35 1344.7057 52.065136 74.809135
40 1291.4506 51.169243 78.30512
45 1245.3145 50.33538 81.559074
50 1205.3451 49.55925 84.58775
55 1170.7189 48.83685 87.40674
60 1140.7214 48.164467 90.03055
65 1114.734 47.53864 92.472694
70 1092.2203 46.956135 94.74576
75 1072.7163 46.413967 96.86146
80 1055.8193 45.909332 98.83067
85 1041.1812 45.439632 100.66355
90 1028.4996 45.002453 102.36953
95 1017.5135 44.595547 103.95739
```

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

[13]:
```
plt.plot(loss_values, 'ro')
```

[13]:
```
[<matplotlib.lines.Line2D at 0x7f8a92329588>]
```



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

[14]:
```
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)
```

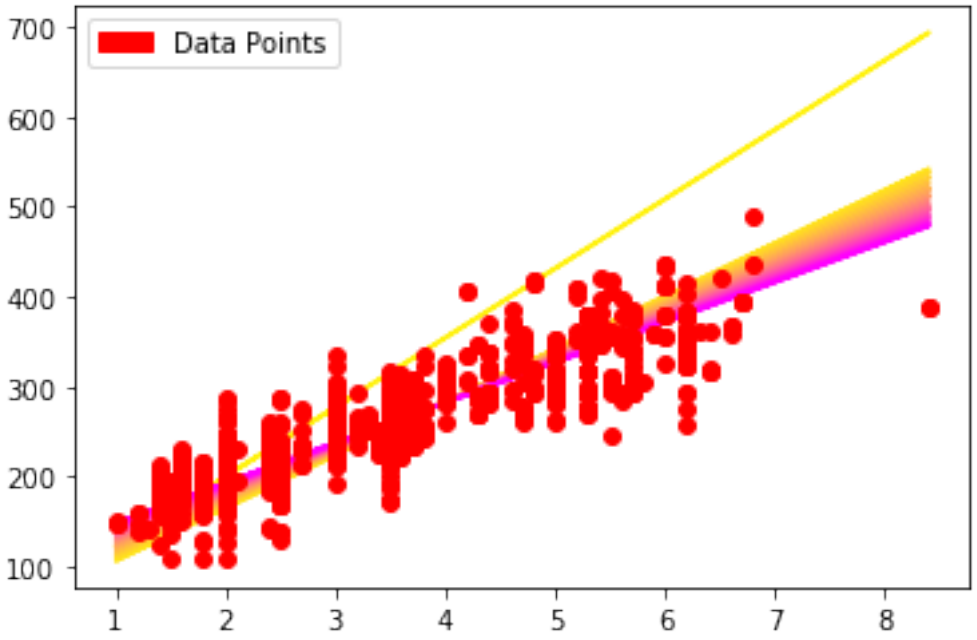
```
    if cb > 1.0: cb = 1.0
    if cg < 0.0: cg = 0.0
    [a, b] = f
    f_y = np.vectorize(lambda x: a*x + b)(train_x)
    line = plt.plot(train_x, f_y)
    plt.setp(line, color=(cr,cg,cb))

plt.plot(train_x, train_y, 'ro')


green_line = mpatches.Patch(color='red', label='Data Points')

plt.legend(handles=[green_line])

plt.show()
```



0.1 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.

0.1.1 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 Saeed Aghabozorgi , Rafael Belo Da Silva

This tutorial was inspired by the documentation of TensorFlow : https://www.tensorflow.org/versions/r0.9/get_started/index.html

Copyright © 2018 Cognitive Class. This notebook and its source code are released under the terms of the MIT License.