Prof. Dr. L. Paditz, 11.03.2019

HTW Dresden,

paditz@htw-dresden.de

Zusammenhang zwischen dem

Grillenzirpen(Chirps per 15 sec)

und der Temperatur(Temp in Farenheit)

Grille (engl. cricket)

==========

Quelle:

https://mathbits.com/MathBits/TISection/Statistics2/

linearREAL.htm

The following data shows the relationship between chirps per second of a **striped ground cricket** (gestreifte Bodengrille) and the corresponding ground temperature. There seems to be some confusion relating to the

"units" used in Pierce's data. According to the text, **The Song of Insects** by **George W. Pierce, 1948, page 20,** the information and unit labeling, as stated at this site, correctly reflects his findings.

https://de.wikipedia.org/wiki/George_W._Pierce

http://www.chbeck.de/11253545

Leseprobe: "Vom Grillen-Thermometer"

https://beckassets.blob.core.windows.net/product/other/11253545/leseprobe_wer%20falsch%20rechnet,%20den%20bestraft%20das%20leben.pdf

Quelle:

https://de.wikipedia.org/wiki/Dolbearsches_Gesetz

Das **Dolbearsche Gesetz** (Dolbear's Law) beschreibt den Zusammenhang zwischen der Temperatur und der

Zirprate einer fast überall in den USA vorkommenden Grille, der zu den Blütengrillen gehörenden Oecanthus fultoni (Snowy Tree Cricket), die deshalb auch als **Thermometergrille** (Thermometer Cricket) bezeichnet wird.

Das "Gesetz" wurde vom Physiker und Erfinder Amos Emerson Dolbear (1837-1910) formuliert und 1897 publiziert.

In der heute verwendeten präzisierten und vereinfachten Form muss man lediglich 13 Sekunden lang zählen, wie oft das Insekt zirpt. Addiert man zu dieser Zahl 40, dann erhält man die Temperatur am Standort der Grille in Grad Fahrenheit.

Quelle:

http://jidan.sinkpoint.com/tf-linearregress/

und

https://

github.com/pestoo0221/tesnforflow_lineargression/

blob/master/linearregression_picker_withsummary.py

data:

https://www.mathbits.com/MathBits/TISection/

Statistics2/linearREAL.htm

20.0 88.6

16.0 71.6

19.8 93.3

18.4 84.3

17.1 80.6

15.5 75.2

14.7 69.7

15.7 71.6

15.4 69.4

16.3 **83.3**

15.0 79.6

17.2 82.6

16.0 80.6

17.0 83.5

14.4 76.3

Teilweise geringfügige Abweichungen in den Originaldaten.

Quelle:

https://stackoverflow.com/questions/50197560/tensorflow-are-all-the-basic-operations-in-python-overidden-in-tensorflow

Daten:

	Chirps	Temp
0	20.0	88.6
1	16.0	71.6
2	19.8	93.3
3	18.4	84.3
4	17.1	80.6
5	15.5	75.2
6	14.7	69.7

7 17.1 82.0

8 15.4 69.4

9 16.2 83.3

10 15.0 78.6

11 17.2 82.6

12 16.0 80.6

13 17.0 83.5

14 14.1 76.3

==========

vollständiger Datensatz:

https://github.com/santipuch590/deeplearning-

playground/blob/master/data/**PierceCricketData.**csv

==========

x_data:={20.,16.,19.8,18.4,17.1,15.5,14.7,17.

{20,16,19.8,18.4,17.1,15.5,14.7,17.1,15.4,16

y_data:={88.6,71.6,93.3,84.3,80.6,75.2,69.7,8

{88.6,71.6,93.3,84.3,80.6,75.2,69.7,82,69.4,

Informativ: Umrechnung in °C

c_data:=fRound((y_data-32)*5/9,1)

{31.4,22,34.1,29.1,27,24,20.9,27.8,20.8,28.5

stop

LinearReg x_data,y_data

done

DispStat

done

==========

Lineare Regression

y=a·x+b

a = 3.2441657

b = 26.012043

r = 0.8364793

r^2 = 0.6996976

MSe = 14.591217

n:=15

15

Normierung mit n:

loss=14.591217*$\frac{n-2}{n}$

loss=12.6457214

Tensorflow:

a=3.244018, b=26.014519, MSe=14.591213

a=3.2440054, b=26.014523, loss=12.645723

12.645723*$\frac{n}{n-2}$

14.59121885

==========

STAT-Menü 

Skript:

```
python3

import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

DATA_FILE =
"/home/parallels/DATA_DIR/PierceCricketData.csv"
# Step 1: read in data from the .csv file

df = pd.read_csv(DATA_FILE)
df.head()
print(df)
x_data, y_data =
(df["Chirps"].values,df["Temp"].values)
print(x_data)
print(y_data)

x = tf.placeholder(tf.float32, shape=x_data.shape)
y = tf.placeholder(tf.float32, shape=y_data.shape)
m = tf.Variable(3.0, name='m')
c = tf.Variable(26.0, name='c')

Y_pred = m*x+c
# nf = 0.1 Dämpfung für loss
nf = 1.0
print("nf:", nf)
```

```
n = x_data.shape[0]
L = (Y_pred*nf-y*nf)**2
loss = (1/(n-2))*tf.reduce_sum(L)

# loss =
tf.reduce_mean(tf.squared_difference(Y_pred,y))
learning_rate = 0.001
optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimi▶

session = tf.Session()
session.run(tf.global_variables_initializer())

convergenceTolerance = 0.000001
previous_m = np.inf
previous_c = np.inf

steps = {}
steps['m'] = []
steps['c'] = []

losses=[]

for k in range(1000):
    _ ,_m, _c, _l =
session.run([optimizer,m,c,loss], feed_dict={x:
x_data, y: y_data})
    steps['m'].append(_m)
```

```
        steps['c'].append(_c)
        losses.append(_l)
        if (np.abs(previous_m - _m) <=
convergenceTolerance) and (np.abs(previous_c - _c)
<= convergenceTolerance):
            print("Finished by Convergence Criterion")
            print([_])
            print(k)
            print([_m, _c, _l])
            break
        previous_m = _m,
        previous_c = _c,

print(losses)

# plot the results
plt.plot(x_data, y_data, 'bo', label='Real data')
plt.plot(x_data, x_data * _m  + _c, 'r',
label='Predicted data')
# label the axis
plt.xlabel("# Chirps per 15 sec")
plt.ylabel("Temp in Farenheit")
plt.legend()
plt.show()
```

Rechnerprotokoll:

```
parallels@parallels-Parallels-Virtual-Platform:~$ python3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
```

```
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for
more information.
>>>
>>> import tensorflow as tf
>>> import numpy as np
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>>
>>> DATA_FILE =
"/home/parallels/DATA_DIR/PierceCricketData.csv"
>>> # Step 1: read in data from the .csv file
...
>>> df = pd.read_csv(DATA_FILE)
>>> df.head()
   Chirps  Temp
0    20.0  88.6
1    16.0  71.6
2    19.8  93.3
3    18.4  84.3
4    17.1  80.6
>>> print(df)
    Chirps  Temp
0     20.0  88.6
1     16.0  71.6
2     19.8  93.3
3     18.4  84.3
4     17.1  80.6
5     15.5  75.2
```

```
6      14.7  69.7
7      17.1  82.0
8      15.4  69.4
9      16.2  83.3
10     15.0  78.6
11     17.2  82.6
12     16.0  80.6
13     17.0  83.5
14     14.1  76.3
>>> x_data, y_data =
(df["Chirps"].values,df["Temp"].values)
>>> print(x_data)
[20.   16.   19.8 18.4 17.1 15.5 14.7 17.1 15.4
16.2 15.   17.2 16.   17.  14.1]
>>> print(y_data)
[88.6 71.6 93.3 84.3 80.6 75.2 69.7 82.   69.4
83.3 78.6 82.6 80.6 83.5 76.3]
>>>
>>> x = tf.placeholder(tf.float32,
shape=x_data.shape)
>>> y = tf.placeholder(tf.float32,
shape=y_data.shape)
>>> m = tf.Variable(3.0, name='m')
>>> c = tf.Variable(26.0, name='c')
>>>
>>> Y_pred=m*x+c
>>> # nf = 0.1 Dämpfung für loss
... nf=1
>>> print("nf:", nf)
```

```
nf: 1
>>>
>>> n = x_data.shape[0]
>>> L = (Y_pred*nf-y*nf)**2
>>> loss = (1/(n-2))*tf.reduce_sum(L)
>>>
>>> # loss =
tf.reduce_mean(tf.squared_difference(Y_pred,y))
... learning_rate = 0.001
>>> optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimi▶
>>>
>>> session = tf.Session()
>>> session.run(tf.global_variables_initializer())
>>>
>>> convergenceTolerance = 0.000001
>>> previous_m = np.inf
>>> previous_c = np.inf
>>>
>>> steps={}
>>> steps['m'] = []
>>> steps['c'] = []
>>>
>>> losses=[]
>>>
>>> for k in range(1000):
...       _ ,_m, _c, _l =
session.run([optimizer,m,c,loss], feed_dict={x:
x_data, y: y_data})
```

```
...         steps['m'].append(_m)
...         steps['c'].append(_c)
...         losses.append(_l)
...         if (np.abs(previous_m - _m) <=
convergenceTolerance) and (np.abs(previous_c - _c)
<= convergenceTolerance):
...             print("Finished by Convergence
Criterion")
...             print([_])
...             print(k)
...             print([_m, _c, _l])
...             break
...         previous_m = _m,
...         previous_c = _c,
...
Finished by Convergence Criterion
[None]
12
[3.244018, 26.014519, 14.591213]
>>> print(losses)
[33.928455, 16.997967, 14.890769, 14.628501,
14.595857, 14.591796, 14.591286, 14.591236,
14.591222, 14.591215, 14.591225, 14.591221,
14.591213]
>>>
>>> # plot the results
... plt.plot(x_data, y_data, 'bo', label='Real data')
[<matplotlib.lines.Line2D object at 0x7f611b9a5fd0>]
>>> plt.plot(x_data, x_data * _m  + _c, 'r',
```

```
label='Predicted data')
[<matplotlib.lines.Line2D object at 0x7f613b7fa048>]
>>> # label the axis
... plt.xlabel("# Chirps per 15 sec")
Text(0.5, 0, '# Chirps per 15 sec')
>>> plt.ylabel("Temp in Farenheit")
Text(0, 0.5, 'Temp in Farenheit')
>>> plt.legend()
<matplotlib.legend.Legend object at 0x7f611b9b06d8>
>>> plt.show()
```

==========

andere Quelle zu **PierceCricketData.**csv:

https://www.snip2code.com/Embed/3214782/Linear-Regression-Example

Skript:

```
python3

import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

DATA_FILE =
"/home/parallels/DATA_DIR/PierceCricketData.csv"
# Step 1: read in data from the .csv file
```

```
df = pd.read_csv(DATA_FILE)

df.head()
print(df)

x_data, y_data =
(df["Chirps"].values,df["Temp"].values)
print(type(y_data[0]))
print(type(y_data))

chirps = tf.placeholder(tf.float32,
shape=(x_data.size), name="chirps")
temperatureActual = tf.placeholder(tf.float32,
shape=(y_data.size), name="temperatureActual") #
yActual

slope = tf.Variable(3.0)
intercept = tf.Variable(26.0)
temperaturePrediction =
tf.add(tf.multiply(slope,chirps),intercept) #
yPrediction

# nf = 0.1
nf = 1.0
print("nf:", nf)
loss = tf.reduce_mean(tf.squared_difference(
temperatureActual*nf, temperaturePrediction*nf))
optimizer = tf.train.GradientDescentOptimizer(0.001)
```

```
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

convergenceTolerance = 0.00000001
previousSlope = np.inf
previousIntercept = np.inf
for epoch in range(100):
    _, slopeVal, interceptVal, lossVal =
sess.run([train, slope, intercept, loss],
feed_dict={chirps: x_data,
          temperatureActual: y_data})
    print(slopeVal, interceptVal, lossVal)
    if (np.abs(previousIntercept - interceptVal) <=
convergenceTolerance):
          break
    if (np.abs(previousSlope - slopeVal) <=
convergenceTolerance):
          break
    previousSlope = slopeVal
    previousIntercept = interceptVal


# plot the results
plt.plot(x_data, y_data, 'bo', label='Real data')
plt.plot(x_data, x_data * slopeVal  + interceptVal,
'r', label='Predicted data')
```

```
# label the axis
plt.xlabel("# Chirps per 15 sec")
plt.ylabel("Temp in Farenheit")
plt.legend()
plt.show()
```

Rechnerprotokoll:

```
parallels@parallels-Parallels-Virtual-Platform:~$ python3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for
more information.
>>>
>>> import tensorflow as tf
>>> import numpy as np
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>>
>>> DATA_FILE =
"/home/parallels/DATA_DIR/PierceCricketData.csv"
>>> # Step 1: read in data from the .csv file
...
>>> df = pd.read_csv(DATA_FILE)
>>>
>>> df.head()
   Chirps  Temp
0    20.0  88.6
1    16.0  71.6
```

```
2     19.8  93.3
3     18.4  84.3
4     17.1  80.6
>>> print(df)
     Chirps  Temp
0      20.0  88.6
1      16.0  71.6
2      19.8  93.3
3      18.4  84.3
4      17.1  80.6
5      15.5  75.2
6      14.7  69.7
7      17.1  82.0
8      15.4  69.4
9      16.2  83.3
10     15.0  78.6
11     17.2  82.6
12     16.0  80.6
13     17.0  83.5
14     14.1  76.3
>>>
... x_data, y_data =
(df["Chirps"].values,df["Temp"].values)
>>> print(type(y_data[0]))
<class 'numpy.float64'>
>>> print(type(y_data))
<class 'numpy.ndarray'>
>>>
... chirps = tf.placeholder(tf.float32,
```

```
shape=(x_data.size), name="chirps")
>>> temperatureActual = tf.placeholder(tf.float32,
shape=(y_data.size), name="temperatureActual") #
yActual
>>>
... slope = tf.Variable(3.0)
>>> intercept = tf.Variable(26.0)
>>> temperaturePrediction =
tf.add(tf.multiply(slope,chirps),intercept) #
yPrediction
>>>
... # nf = 0.1
... nf=1
>>> print("nf:", nf)
nf: 1
>>> loss = tf.reduce_mean(tf.squared_difference(
temperatureActual*nf, temperaturePrediction*nf))
>>> optimizer =
tf.train.GradientDescentOptimizer(0.001)
>>>
... train = optimizer.minimize(loss)
>>>
... init = tf.global_variables_initializer()
>>> sess = tf.Session()
>>> sess.run(init)
>>>
... convergenceTolerance = 0.00000001
>>> previousSlope = np.inf
>>> previousIntercept = np.inf
```

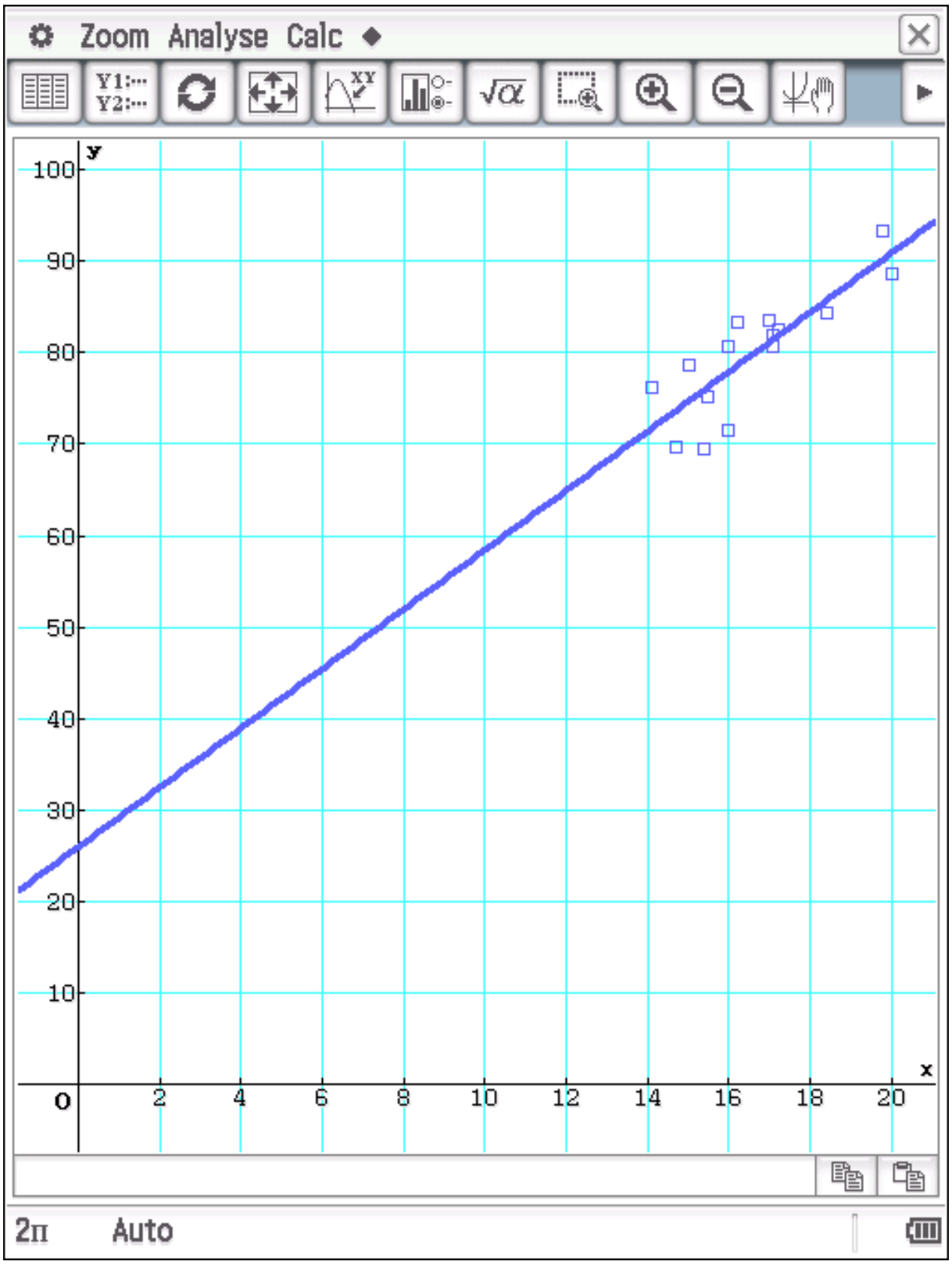
```
>>> for epoch in range(100):
...         _, slopeVal, interceptVal, lossVal =
sess.run([train, slope, intercept, loss],
feed_dict={chirps: x_data,
...                 temperatureActual: y_data})
...         print(slopeVal, interceptVal, lossVal)
...         if (np.abs(previousIntercept - interceptVal)
<= convergenceTolerance):
...                 break
...         if (np.abs(previousSlope - slopeVal) <=
convergenceTolerance):
...                 break
...         previousSlope = slopeVal
...         previousIntercept = interceptVal
...
3.1368732 26.008146 29.40466
3.1969724 26.011723 15.876786
3.223361 26.013292 13.268667
3.234948 26.013983 12.765824
3.2400355 26.014286 12.668881
3.2422693 26.01442 12.650176
3.2432501 26.014479 12.6465845
3.243681 26.014503 12.645894
3.24387 26.014515 12.645759
3.243953 26.01452 12.645727
3.2439895 26.014523 12.645732
3.2440054 26.014523 12.645723
>>>
>>> # plot the results
```

```
... plt.plot(x_data, y_data, 'bo', label='Real data')
[<matplotlib.lines.Line2D object at 0x7fdf808f2eb8>]
>>> plt.plot(x_data, x_data * slopeVal  +
interceptVal, 'r', label='Predicted data')
[<matplotlib.lines.Line2D object at 0x7fdf886958d0>]
>>> # label the axis
... plt.xlabel("# Chirps per 15 sec")
Text(0.5, 0, '# Chirps per 15 sec')
>>> plt.ylabel("Temp in Farenheit")
Text(0, 0.5, 'Temp in Farenheit')
>>> plt.legend()
<matplotlib.legend.Legend object at 0x7fdf8087c5c0>
>>> plt.show()
```

Download für dieses Dokument:

www.informatik.htw-dresden.de/
~paditz/Tensorflow-Ue14.pdf

Lineare Regression Uebung14: mit ClassPad



Lineare Regression Uebung14: mit Tensorflow