Prof. Dr. L. Paditz, 19.03.2019

HTW Dresden,

paditz@htw-dresden.de

Zusammenhang zwischen dem

Grillenzirpen(Chirps per 15 sec)

und der Temperatur(Temp in Farenheit)

Grille (engl. cricket)

=====================================

neue 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

PierceCricketData.csv saved data from

http://mathbits.com/MathBits/TISection/

Statistics2/linearREAL.htm

Sehr schön gestaltete Web-Seite!

Teilweise geringfügige Abweichungen in den Originaldaten.

Skript:

```
python3
# !/usr/bin/env python
# -*- coding: utf-8 -*-
# Jidan Zhong
# 2017- Jan-31
### Linear regression


import io
import tensorflow as tf
import numpy as np
```

```
import pandas as pd

import matplotlib.patches as mpatches

import matplotlib.pyplot as plt

# %matplotlib inline



plt.rcParams['figure.figsize'] = (10,6)



def save_plot():

    """Save a pyplot plot to buffer."""

    # plt.figure()

    # plt.plot(x, y)

    # plt.title("test")

    buf = io.BytesIO()

    plt.savefig(buf, format='png')

    buf.seek(0)

    image = tf.image.decode_png(buf.getvalue(),
channels=4)

    # Add the batch dimension
```

```
    image = tf.expand_dims(image, 0)

    # Add image summary

    summary_op = tf.image.summary("plot", image)

    # return buf

    return summary_op


def variable_summaries(var):

    ######################ADDDDDED########

    """Attach a lot of summaries to a Tensor (for

TensorBoard visualization)."""

    with tf.name_scope('summaries'):

        mean = tf.reduce_mean(var)

        tf.summary.scalar('mean', mean)

    with tf.name_scope('stddev'):

        stddev =

tf.sqrt(tf.reduce_mean(tf.square(var - mean)))

        tf.summary.scalar('stddev', stddev)

        tf.summary.scalar('max',
```

```
tf.reduce_max(var))

          tf.summary.scalar('min',

tf.reduce_min(var))

          tf.summary.histogram('histogram', var)


## load data and explore the data


# PierceCricketData.csv saved data from

http://mathbits.com/MathBits/TISection/Statistics2/line▶

# df =

pd.read_csv("/home/parallels/DATA_DIR/PierceCricketI▶

= None)


df =

pd.read_csv("/home/parallels/DATA_DIR/PierceCricketI▶

df.head()

# x_data, y_data = (df[0], df[1]) - fehlerhaft!
```

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

# plot data to explore

plt.figure()
plt.plot(x_data, y_data, 'ro')
plt.xlabel("# Chirps per 15 sec")
plt.ylabel("Temp in Farenheit")
# plt.show()

# ############################1

# normalize the data so the performance is better
x_data_n, y_data_n = ((x_data -
x_data.mean())/x_data.std(),(y_data -
```

```
y_data.mean())/y_data.std() )

# plot data to explore


plt.figure()

plt.plot(x_data_n, y_data_n, 'bo')

plt.xlabel("# Chirps per 15 sec")

plt.ylabel("Temp in Farenheit")

# plt.show()


# ###########################2


steps = {}




with tf.Graph().as_default():

    with tf.name_scope('input'):

        ## preparing linear model

        Xsize= x_data.size
```

```
        Ysize= y_data.size
        X = tf.placeholder(tf.float32, shape=(Xsize),
name='x-input')
        Y = tf.placeholder(tf.float32,shape=(Ysize),
name='y-input')
    with tf.name_scope('model'):
        with tf.name_scope('weights'):
            W = tf.Variable(3.0, name='weight') #
weight
            tf.summary.scalar('weights', W)
            # variable_summaries(W)
        with tf.name_scope('bias'):
            B = tf.Variable(1.0, name = 'bias') #
bias
            tf.summary.scalar('bias', B)
            # variable_summaries(B)
        with tf.name_scope('linear_model'):
            # construct a model
```

```
            y = X * W + B  # or # Y =
tf.add(tf.mul(W, X), B)
    with tf.name_scope('loss'):
        # seting up the loss function
        loss =
tf.reduce_mean(tf.squared_difference(y,Y))
        # This is wrong: loss =
tf.reduce_mean(tf.square(y-Y))
        tf.summary.scalar('loss', loss)
        # tf.scalar_summary('loss', loss)
        # variable_summaries(loss)
    with tf.name_scope('train'):
        optimizer =
tf.train.GradientDescentOptimizer(0.1).minimize(loss)
    # ##################################3
    # Merge all the summaries and write them out to
/tmp/mnist_logs (by default)
    merged = tf.summary.merge_all()
```

```
    # saver = tf.train.Saver()

    # start = time.time()

    init = tf.global_variables_initializer()

    #pred = sess.run(y, feed_dict={X:x_data})

    with

tf.Session(config=tf.ConfigProto(allow_soft_placement=T▶

log_device_placement=True)) as sess:

        # the first one means if the device doesnt

exist, it can automatically appoint an existing device;

        # 2nd means it will show the log infor for

parameters and operations are on which device

        # ##-1-

        # train_writer =

tf.summary.FileWriter('/home/parallels/DATA_DIR/test▶

sess.graph)

        sess.run(init)

        convergenceTolerance = 0.0001

        previous_w = np.inf
```

```
        previous_b = np.inf
        # steps = {}
        steps['w'] = []
        steps['b'] = []
        losses=[]
        for k in range(1000):
            yPred, _, weight, bias, ls, summary =
sess.run([y,optimizer,W,B,loss, merged],
feed_dict={X:np.asarray(x_data_n),
Y:np.asarray(y_data_n)})
            # ##-2-
            # train_writer.add_summary(summary,
k)
            print("step: %d, loss: %f" %(k,ls))
            steps['w'].append(weight)
            steps['b'].append(bias)
            losses.append(ls)
            if (np.abs(previous_w - weight) or
```

```
np.abs(previous_b - bias) ) <= convergenceTolerance :
                print("Finished by Convergence
Criterion")
                print(k)
                print(ls)
                break
            previous_w = weight
            previous_b = bias
        print("In the model, Final W: %f, Final B:
%f" %(weight, bias))
        # model without normalizing data will be :
        # y = X * (W * y_data.std() /
x_data.std()) + (y_data.std() * b  + y_data.mean()
- x_data.mean() * W * y_data.std() /
x_data.std())
        print("W for original data: %f, B for original
data: %f" %(weight* y_data.std()/ x_data.std(),
y_data.std() * bias  + y_data.mean() -
```

```
x_data.mean() * weight * y_data.std() /
x_data.std()))
        # final step show the graph
        plt.figure()
        plt.plot(x_data_n, yPred)
        plt.plot(x_data_n, y_data_n, 'ro')
        plt.xlabel("# Chirps per 15 sec normalized
")
        plt.ylabel("Temp in Farenheit normalized")
        # save the figure to buffer
        # summary_op = save_plot()
        # summary1 = sess.run(summary_op)
        # ##-3-
        # train_writer.add_summary(summary1)
        ########## Plot the figures for self
exploration
        plt.figure(1)
        plt.subplot(221)
```

```
plt.plot(x_data_n, yPred)
plt.plot(x_data_n, y_data_n, 'ro')
# label the axis
plt.xlabel("# Chirps per 15 sec normalized")
plt.ylabel("Temp in Farenheit normalized")
# plt.show()
# print the loss function
plt.subplot(223)
plt.plot(range(k+1), losses)
plt.xlabel("step")
plt.ylabel("loss")
# plt.show()
plt.subplot(224)
plt.plot(x_data, yPred * y_data.std() +
y_data.mean() )
plt.plot(x_data, y_data, 'ro')
# label the axis
plt.xlabel("# Chirps per 15 sec")
```

```
plt.ylabel("Temp in Farenheit")
# plt.show()
# print the step changes
plt.subplot(222)
# ##-4-
# converter = plt.colors
cr, cg, cb = (0.0, 1.0, 0.0)
for f in range(k):
    cb +=1.0 / k
    cg -=1.0 / k
    cr +=1.0 / k / 2
#############3
    if cb > 1.0: cb = 1.0
    if cg < 0.0: cg = 0.0
    if cr > 1.0: cr = 0.5
    a = steps['w'][f]
    b = steps['b'][f]
    f_y =  np.vectorize(lambda x: a * x +
```

b) (x_data_n)

```
            line = plt.plot(x_data_n, f_y)
            plt.setp(line, color=(cr,cg,cb))
            plt.plot(x_data_n, y_data_n,'ro')
            plt.xlabel("# Chirps per 15 sec
normalized")
            plt.ylabel("Temp in Farenheit
normalized")
    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.
>>> # !/usr/bin/env python
```

```
... # -*- coding: utf-8 -*-
... # Jidan Zhong
... # 2017- Jan-31
... ### Linear regression
...
>>> import io
>>> import tensorflow as tf
>>> import numpy as np
>>> import pandas as pd
>>> import matplotlib.patches as mpatches
>>> import matplotlib.pyplot as plt
>>> # %matplotlib inline
...
>>> plt.rcParams['figure.figsize'] = (10,6)
>>>
>>> def save_plot():
...     """Save a pyplot plot to buffer."""
...     # plt.figure()
```

```
...        # plt.plot(x, y)

...        # plt.title("test")

...        buf = io.BytesIO()

...        plt.savefig(buf, format='png')

...        buf.seek(0)

...        image = tf.image.decode_png(buf.getvalue(),

channels=4)

...        # Add the batch dimension

...        image = tf.expand_dims(image, 0)

...        # Add image summary

...        summary_op = tf.image.summary("plot",

image)

...        # return buf

...        return summary_op

...

>>> def variable_summaries(var):

...

##############################################
```

```
...     """Attach a lot of summaries to a Tensor
(for TensorBoard visualization)."""
...     with tf.name_scope('summaries'):
...         mean = tf.reduce_mean(var)
...         tf.summary.scalar('mean', mean)
...     with tf.name_scope('stddev'):
...         stddev =
tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
...         tf.summary.scalar('stddev', stddev)
...         tf.summary.scalar('max',
tf.reduce_max(var))
...         tf.summary.scalar('min',
tf.reduce_min(var))
...         tf.summary.histogram('histogram', var)
...
>>> ## load data and explore the data
...
>>> # PierceCricketData.csv saved data from
```

```
http://mathbits.com/MathBits/TISection/Statistics2/line▶

... # df =

pd.read_csv("/home/parallels/DATA_DIR/PierceCricketI▶

= None)

...

>>> df =

pd.read_csv("/home/parallels/DATA_DIR/PierceCricketI▶

>>> 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

>>> # x_data, y_data = (df[0], df[1]) - fehlerhaft!

...

>>> 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]
>>>
>>> # plot data to explore
...
>>> plt.figure()
<Figure size 1000x600 with 0 Axes>
>>> plt.plot(x_data, y_data, 'ro')
[<matplotlib.lines.Line2D object at 0x7fd58eb4c240>]
>>> 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.show()

...

>>> # ############################1

...

>>> # normalize the data so the performance is better

... x_data_n, y_data_n = ((x_data -

x_data.mean())/x_data.std(),(y_data -

y_data.mean())/y_data.std() )

>>> # plot data to explore

...

>>> plt.figure()

<Figure size 1000x600 with 0 Axes>

>>> plt.plot(x_data_n, y_data_n, 'bo')

[<matplotlib.lines.Line2D object at 0x7fd58eb13400>]

>>> 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.show()

...

>>> # ############################2

...

>>> steps={}

>>> with tf.Graph().as_default():

...     with tf.name_scope('input'):

...         ## preparing linear model

...         Xsize= x_data.size

...         Ysize= y_data.size

...         X = tf.placeholder(tf.float32,

shape=(Xsize), name='x-input')

...         Y =

tf.placeholder(tf.float32,shape=(Ysize),

name='y-input')

...     with tf.name_scope('model'):

...         with tf.name_scope('weights'):
```

```
...                W = tf.Variable(3.0,
name='weight') # weight
...                tf.summary.scalar('weights', W)
...                # variable_summaries(W)
...            with tf.name_scope('bias'):
...                B = tf.Variable(1.0, name =
'bias') # bias
...                tf.summary.scalar('bias', B)
...                # variable_summaries(B)
...            with tf.name_scope('linear_model'):
...                # construct a model
...                y = X * W + B  # or # Y =
tf.add(tf.mul(W, X), B)
...        with tf.name_scope('loss'):
...            # seting up the loss function
...            loss =
tf.reduce_mean(tf.squared_difference(y,Y))
...            # This is wrong: loss =
```

```
tf.reduce_mean(tf.square(y-Y))
...             tf.summary.scalar('loss', loss)
...             # tf.scalar_summary('loss', loss)
...             # variable_summaries(loss)
...         with tf.name_scope('train'):
...             optimizer =
tf.train.GradientDescentOptimizer(0.1).minimize(loss)
...         # ###############################3
...         # Merge all the summaries and write them
out to /tmp/mnist_logs (by default)
...         merged = tf.summary.merge_all()
...         # saver = tf.train.Saver()
...         # start = time.time()
...         init = tf.global_variables_initializer()
...         #pred = sess.run(y, feed_dict={X:x_data})
...         with
tf.Session(config=tf.ConfigProto(allow_soft_placement=T▶
log_device_placement=True)) as sess:
```

```
...             # the first one means if the device
doesnt exist, it can automatically appoint an existing
device;
...             # 2nd means it will show the log infor
for parameters and operations are on which device
...             # ##-1-
...             # train_writer =
tf.summary.FileWriter('/home/parallels/DATA_DIR/test▶
sess.graph)
...             sess.run(init)
...             convergenceTolerance = 0.0001
...             previous_w = np.inf
...             previous_b = np.inf
...             # steps = {}
...             steps['w'] = []
...             steps['b'] = []
...             losses=[]
...             for k in range(1000):
```

```
...                 yPred, _, weight, bias, ls,
summary = sess.run([y,optimizer,W,B,loss, merged],
feed_dict={X:np.asarray(x_data_n),
Y:np.asarray(y_data_n)})
...                 # ##-2-
...                 #
train_writer.add_summary(summary, k)
...                 print("step: %d, loss: %f" %(k,ls))
...                 steps['w'].append(weight)
...                 steps['b'].append(bias)
...                 losses.append(ls)
...                 if (np.abs(previous_w - weight) or
np.abs(previous_b - bias) ) <= convergenceTolerance :
...                     print("Finished by Convergence
Criterion")
...                     print(k)
...                     print(ls)
...                     break
```

```
...                     previous_w = weight
...                     previous_b = bias
...                 print("In the model, Final W: %f, Final
B: %f" %(weight, bias))
...                 # model without normalizing data will be
:
...                 # y = X * (W * y_data.std() /
x_data.std()) + (y_data.std() * b  + y_data.mean()
- x_data.mean() * W * y_data.std() /
x_data.std())
...                 print("W for original data: %f, B for
original data: %f" %(weight* y_data.std()/
x_data.std(), y_data.std() * bias  + y_data.mean()
- x_data.mean() * weight * y_data.std() /
x_data.std()))
...                 # final step show the graph
...                 plt.figure()
...                 plt.plot(x_data_n, yPred)
```

```
...             plt.plot(x_data_n, y_data_n, 'ro')
...             plt.xlabel("# Chirps per 15 sec
normalized ")
...             plt.ylabel("Temp in Farenheit
normalized")
...             # save the figure to buffer
...             # summary_op = save_plot()
...             # summary1 = sess.run(summary_op)
...             # ##-3-
...             # train_writer.add_summary(summary1)
...             ########## Plot the figures for self
exploration
...             plt.figure(1)
...             plt.subplot(221)
...             plt.plot(x_data_n, yPred)
...             plt.plot(x_data_n, y_data_n, 'ro')
...             # label the axis
...             plt.xlabel("# Chirps per 15 sec
```

```
normalized")
...           plt.ylabel("Temp in Farenheit
normalized")
...           # plt.show()
...           # print the loss function
...           plt.subplot(223)
...           plt.plot(range(k+1), losses)
...           plt.xlabel("step")
...           plt.ylabel("loss")
...           # plt.show()
...           plt.subplot(224)
...           plt.plot(x_data, yPred * y_data.std() +
y_data.mean() )
...           plt.plot(x_data, y_data, 'ro')
...           # label the axis
...           plt.xlabel("# Chirps per 15 sec")
...           plt.ylabel("Temp in Farenheit")
...           # plt.show()
```

```
...            # print the step changes
...            plt.subplot(222)
...            # ##-4-
...            # converter = plt.colors
...            cr, cg, cb = (0.0, 1.0, 0.0)
...            for f in range(k):
...                cb +=1.0 / k
...                cg -=1.0 / k
...                cr +=1.0 / k / 2
#############3
...                if cb > 1.0: cb = 1.0
...                if cg < 0.0: cg = 0.0
...                if cr > 1.0: cr = 0.5
...                a = steps['w'][f]
...                b = steps['b'][f]
...                f_y =  np.vectorize(lambda x: a *
x + b) (x_data_n)
...                line = plt.plot(x_data_n, f_y)
```

```
...                    plt.setp(line, color=(cr,cg,cb))

...                    plt.plot(x_data_n, y_data_n,'ro')

...                    plt.xlabel("# Chirps per 15 sec
normalized")

...                    plt.ylabel("Temp in Farenheit
normalized")

...          plt.show()

...

<tf.Tensor 'model/weights/weights:0' shape=()
dtype=string>

<tf.Tensor 'model/bias/bias_1:0' shape=()
dtype=string>

<tf.Tensor 'loss/loss:0' shape=() dtype=string>

...

step: 0, loss: 5.981124

step: 1, loss: 3.936028

step: 2, loss: 2.627167

step: 3, loss: 1.789496
```

step: 4, loss: 1.253386

step: 5, loss: 0.910276

step: 6, loss: 0.690685

step: 7, loss: 0.550147

step: 8, loss: 0.460203

step: 9, loss: 0.402639

step: 10, loss: 0.365798

step: 11, loss: 0.342219

step: 12, loss: 0.327129

step: 13, loss: 0.317472

step: 14, loss: 0.311291

step: 15, loss: 0.307335

step: 16, loss: 0.304803

step: 17, loss: 0.303183

step: 18, loss: 0.302146

step: 19, loss: 0.301482

step: 20, loss: 0.301057

step: 21, loss: 0.300786

step: 22, loss: 0.300612

step: 23, loss: 0.300500

step: 24, loss: 0.300429

step: 25, loss: 0.300384

step: 26, loss: 0.300354

step: 27, loss: 0.300336

step: 28, loss: 0.300324

step: 29, loss: 0.300316

step: 30, loss: 0.300311

step: 31, loss: 0.300308

step: 32, loss: 0.300306

step: 33, loss: 0.300305

step: 34, loss: 0.300304

step: 35, loss: 0.300303

step: 36, loss: 0.300303

step: 37, loss: 0.300303

step: 38, loss: 0.300303

Finished by Convergence Criterion

38

0.30030262

In the model, Final W: 0.836839, Final B: 0.000166

W for original data: 3.245560, B for original data:

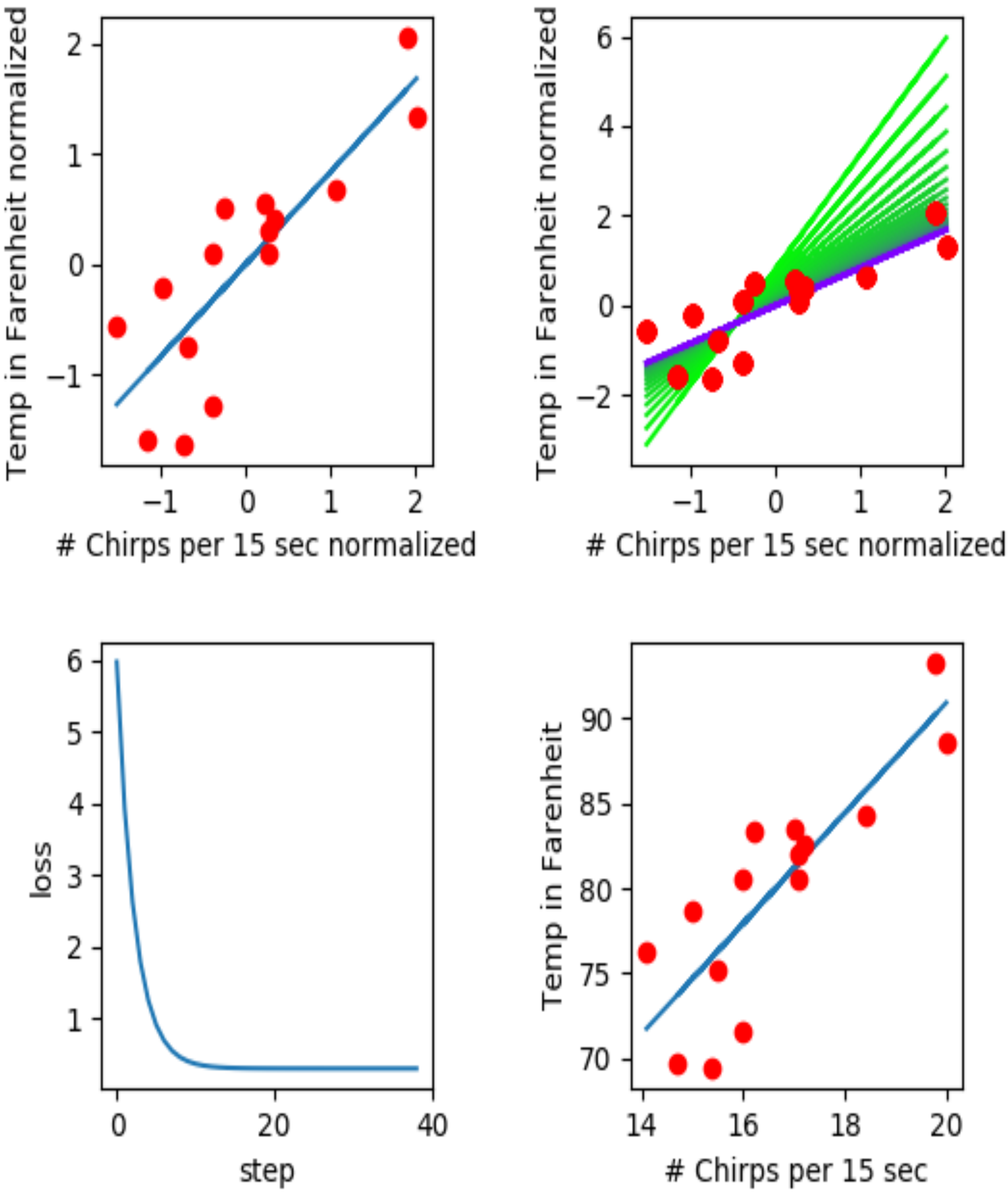
25.989934

Download für dieses Dokument:

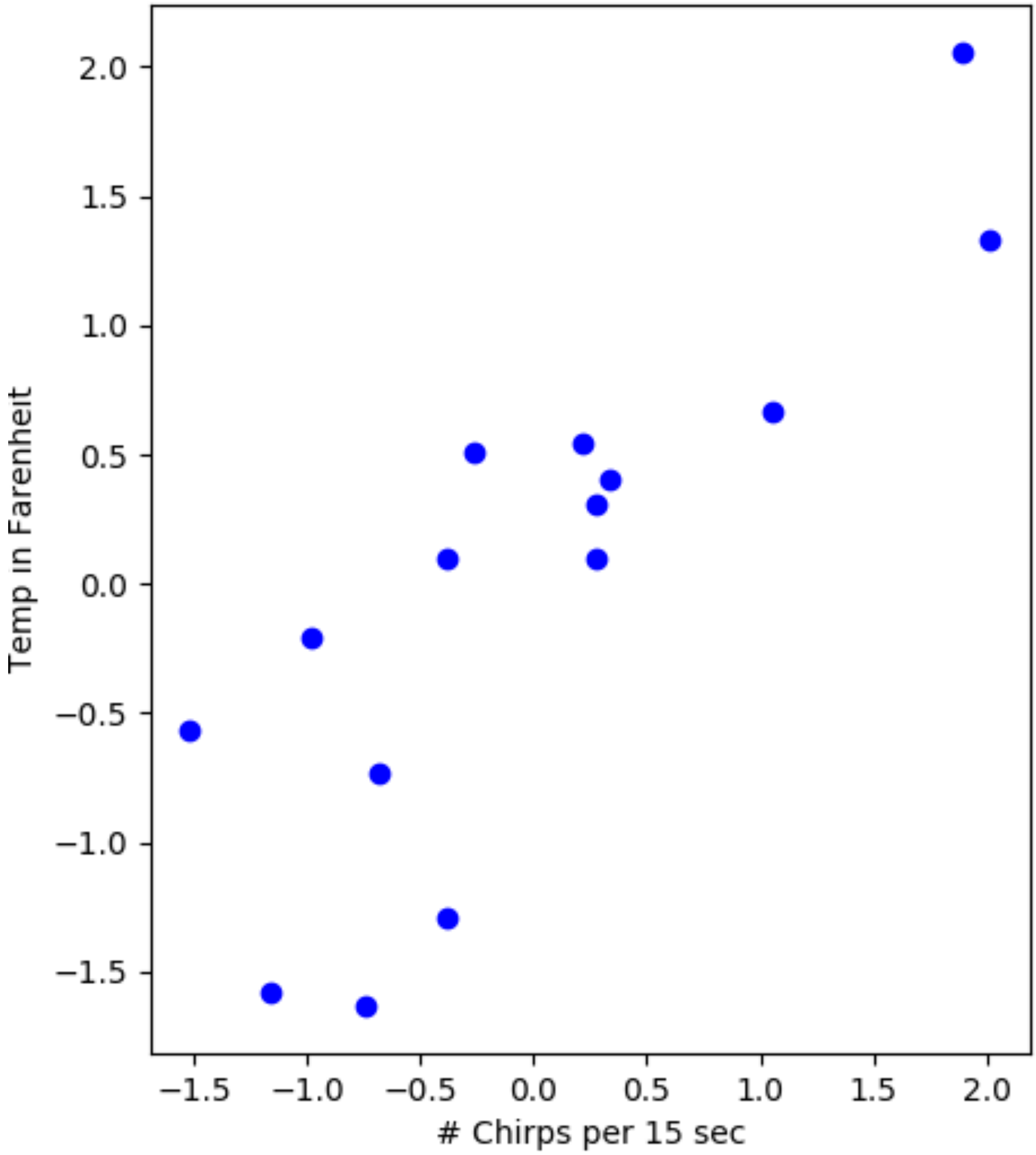
www.informatik.htw-dresden.de/

~paditz/Tensorflow-Ue16.pdf

Lineare Regression



Einzelbilder: normalisierte Daten



Einzelbilder: normalisierte Daten