Prof. Dr. L. Paditz, 14.02.2019

HTW Dresden

paditz@htw-dresden.de

Kontrolle der txt-Datei und Aufsplittung deren Inhalte,

Verarbeitung einer txt-Datei mit Text- und Dateninhalten

==========

```
import numpy as np
DATA_FILE =
'/home/parallels/DATA_DIR/birth_life_2010.txt'
text = open(DATA_FILE, 'r').readlines()[1:]
print(text)
data = [line[:-1].split('\t') for line in text]
print(data)
births = [float(line[1]) for line in data]
```

```
print(births)

lifes = [float(line[2]) for line in data]

print(lifes)

data = list(zip(births, lifes))

print(data)

n_samples = len(data)

print(n_samples)

data = np.asarray(data, dtype=np.float32)

print(data)
```

Verarbeitung einer txt-Datei mit Text- und Dateninhalten:

==========

https://gaussian37.github.io/ML-Code-Linear-Regression-from-Text-with-Tensorflow/

Skript zum Kopieren in das Terminal(Linux):

```
python3

""" Solution for simple linear regression example using
placeholders
Created by Chip Huyen (chiphuyen@cs.stanford.edu)
CS20: "TensorFlow for Deep Learning Research"
cs20.stanford.edu
Lecture 03
"""
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
import time

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

DATA_FILE =
```

```
'/home/parallels/DATA_DIR/birth_life_2010.txt'


# Step 1: read in data from the .txt file


def huber_loss(labels, predictions, delta=14.0):
    residual = tf.abs(labels - predictions)
    def f1(): return 0.5 * tf.square(residual)
    def f2(): return delta * residual - 0.5 *
tf.square(delta)
    return tf.cond(residual < delta, f1, f2)


# Read in birth_life_2010.txt and return: data in the
form of NumPy array n_samples: number of samples


# readline from 1 row (except 0 row : category)
text = open(DATA_FILE, 'r').readlines()[1:]
# Split each line with '\t'
data = [line[:-1].split('\t') for line in text]
```

```
# Select the column 1 of birth
births = [float(line[1]) for line in data]
# Select the column 2 of lifes
lifes = [float(line[2]) for line in data]
# Zip birth & lifes
data = list(zip(births, lifes))
# The number of samples
n_samples = len(data)
# Transform data type from list to np.array
data = np.asarray(data, dtype=np.float32)
# return data, n_samples


# Step 2: create placeholders for X (birth rate) and Y
(life expectancy)
X = tf.placeholder(tf.float32, name='X')
Y = tf.placeholder(tf.float32, name='Y')


# Step 3: create weight and bias, initialized to 0
```

```
# w = tf.get_variable('weights',
initializer=tf.constant(0.0)) old single weight
w = tf.get_variable('weights_1',
initializer=tf.constant(0.0))
u = tf.get_variable('weights_2',
initializer=tf.constant(0.0))
b = tf.get_variable('bias', initializer=tf.constant(0.0))

# Step 4: build model to predict Y
Y_predicted = w * X + b  #linear
#Y_predicted = w * X * X + X * u + b  #quadratic
#Y_predicted = w  # test of nonsense

# Step 5: use the squared error as the loss function
# you can use either mean squared error or Huber loss
loss = tf.reduce_mean(tf.square(Y - Y_predicted,
name='loss'))
#loss = huber_loss(Y, Y_predicted)
```

```
# Step 6: using gradient descent with learning rate of
0.0001 to minimize loss
optimizer =
tf.train.GradientDescentOptimizer(learning_rate=
0.0001).minimize(loss)

start = time.time()
writer = tf.summary.FileWriter('./graphs/linear_reg',
tf.get_default_graph())
with tf.Session() as sess:
    # Step 7: initialize the necessary variables, in this
case, w,u and b
    sess.run(tf.global_variables_initializer())
    # Step 8: train the model for 1001 epochs
    for i in range(1001):
        total_loss = 0
        for x, y in data:
```

```
                # Session execute optimizer and fetch
values of loss
                _, _loss = sess.run([optimizer, loss],
feed_dict={X: x, Y:y})
                total_loss += _loss
            if i%100 == 0:
                print('Epoch {0}: {1}'.format(i,
total_loss/n_samples))
    # close the writer when you're done using it
    writer.close()
    # Step 9: output the values of w,u and b
    w_out, u_out, b_out = sess.run([w, u, b])

print('Took: %f seconds' %(time.time() - start))
print(f'w = {w_out}')
print(f'u = {u_out}')
print(f'b = {b_out}')
```

```
# plot the results
plt.plot(data[:,0], data[:,1], 'bo', label='Real data')
plt.plot(data[:,0], data[:,0] * w_out  + b_out, 'r',
label='Predicted data')
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.
>>>
>>> """ Solution for simple linear regression example
using placeholders
```

```
... Created by Chip Huyen
(chiphuyen@cs.stanford.edu)
... CS20
... cs20.stanford.edu
... Lecture 03
... """
' Solution for simple linear regression example using
placeholders\nCreated by Chip Huyen
(chiphuyen@cs.stanford.edu)
\nCS20\ncs20.stanford.edu\nLecture 03\n'
>>> import os
>>> os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
>>> import time
>>>
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> import tensorflow as tf
>>>
```

```
>>> DATA_FILE =
'/home/parallels/DATA_DIR/birth_life_2010.txt'
>>>
>>> # Step 1: read in data from the .txt file
...
>>> def huber_loss(labels, predictions, delta=14.0):
...       residual = tf.abs(labels - predictions)
...       def f1(): return 0.5 * tf.square(residual)
...       def f2(): return delta * residual - 0.5 *
tf.square(delta)
...       return tf.cond(residual < delta, f1, f2)
...
>>> # Read in birth_life_2010.txt and return: data in
the form of NumPy array n_samples: number of
samples
...
... # readline from 1 row (except 0 row : category)
... text = open(DATA_FILE, 'r').readlines()[1:]
```

```
>>> # Split each line with '\t'
... data = [line[:-1].split('\t') for line in text]
>>> # Select the column 1 of birth
... births = [float(line[1]) for line in data]
>>> # Select the column 2 of lifes
... lifes = [float(line[2]) for line in data]
>>> # Zip birth & lifes
... data=list(zip(births,lifes))
>>> # The number of samples
... n_samples = len(data)
>>> # Transform data type from list to np.array
... data = np.asarray(data, dtype=np.float32)
>>> # return data, n_samples
...
>>> # Step 2: create placeholders for X (birth rate)
and Y (life expectancy)
... X = tf.placeholder(tf.float32, name='X')
>>> Y = tf.placeholder(tf.float32, name='Y')
```

```
>>>
>>> # Step 3: create weight and bias, initialized to 0
... # w = tf.get_variable('weights',
initializer=tf.constant(0.0)) old single weight
... w = tf.get_variable('weights_1',
initializer=tf.constant(0.0))
>>> u = tf.get_variable('weights_2',
initializer=tf.constant(0.0))
>>> b = tf.get_variable('bias',
initializer=tf.constant(0.0))
>>>
>>> # Step 4: build model to predict Y
... Y_predicted = w * X + b  #linear
>>> #Y_predicted = w * X * X + X * u + b
#quadratic
... #Y_predicted = w  # test of nonsense
...
>>> # Step 5: use the squared error as the loss
```

```
function
... # you can use either mean squared error or Huber
loss
... loss = tf.reduce_mean(tf.square(Y - Y_predicted,
name='loss'))
>>> #loss = utils.huber_loss(Y, Y_predicted)
...
>>> # Step 6: using gradient descent with learning rate
of 0.0001 to minimize loss
... optimizer =
tf.train.GradientDescentOptimizer(learning_rate=
0.0001).minimize(loss)
>>>
>>> start = time.time()
>>> writer =
tf.summary.FileWriter('./graphs/linear_reg',
tf.get_default_graph())
>>> with tf.Session() as sess:
```

```
...     # Step 7: initialize the necessary variables, in
this case, w and b
...     sess.run(tf.global_variables_initializer())
...     # Step 8: train the model for 1001 epochs
...     for i in range(1001):
...         total_loss = 0
...         for x, y in data:
...             # Session execute optimizer and
fetch values of loss
...             _, _loss = sess.run([optimizer,
loss], feed_dict={X: x, Y:y})
...             total_loss += _loss
...         if i%100 == 0:
...             print('Epoch {0}: {1}'.format(i,
total_loss/n_samples))
...     # close the writer when you're done using it
...     writer.close()
...     # Step 9: output the values of w and b
```

```
...       w_out, b_out = sess.run([w, b])

...

Epoch 0: 3833.7294941952355

Epoch 100: 345.25299883872196

Epoch 200: 106.44657805792399

Epoch 300: 48.67149240462826

Epoch 400: 34.716749463281225

Epoch 500: 31.35716862543241

Epoch 600: 30.55451585000479

Epoch 700: 30.365150467117967

Epoch 800: 30.322084076785924

Epoch 900: 30.312888509673556

Epoch 1000: 30.311351597927338

>>> print('Took: %f seconds' %(time.time() - start))

Took: 99.697321 seconds

>>> print(f'w = {w_out}')

w = -5.769937992095947

>>> print(f'b = {b_out}')
```

```
b = 85.66625213623047
>>>
>>> # plot the results
... plt.plot(data[:,0], data[:,1], 'bo', label='Real
data')
[<matplotlib.lines.Line2D object at 0x7f41a00dd160>]
>>> plt.plot(data[:,0], data[:,0] * w_out + b_out,
'r', label='Predicted data')
[<matplotlib.lines.Line2D object at 0x7f41a1a644a8>]
>>> plt.legend()
<matplotlib.legend.Legend object at 0x7f41a00dd9e8>
>>> plt.show()
```

Anlage:

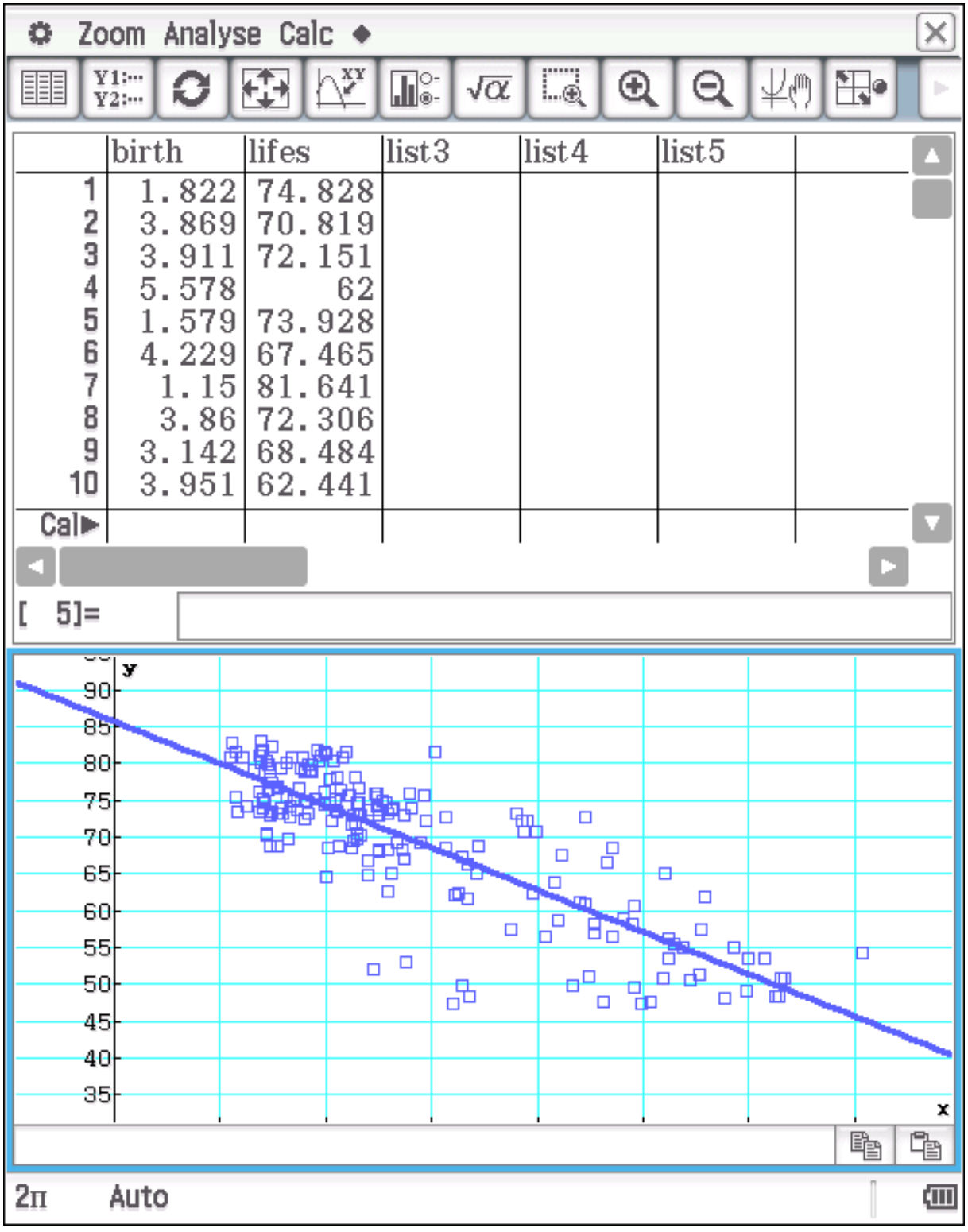
Bild mit linearer Regression

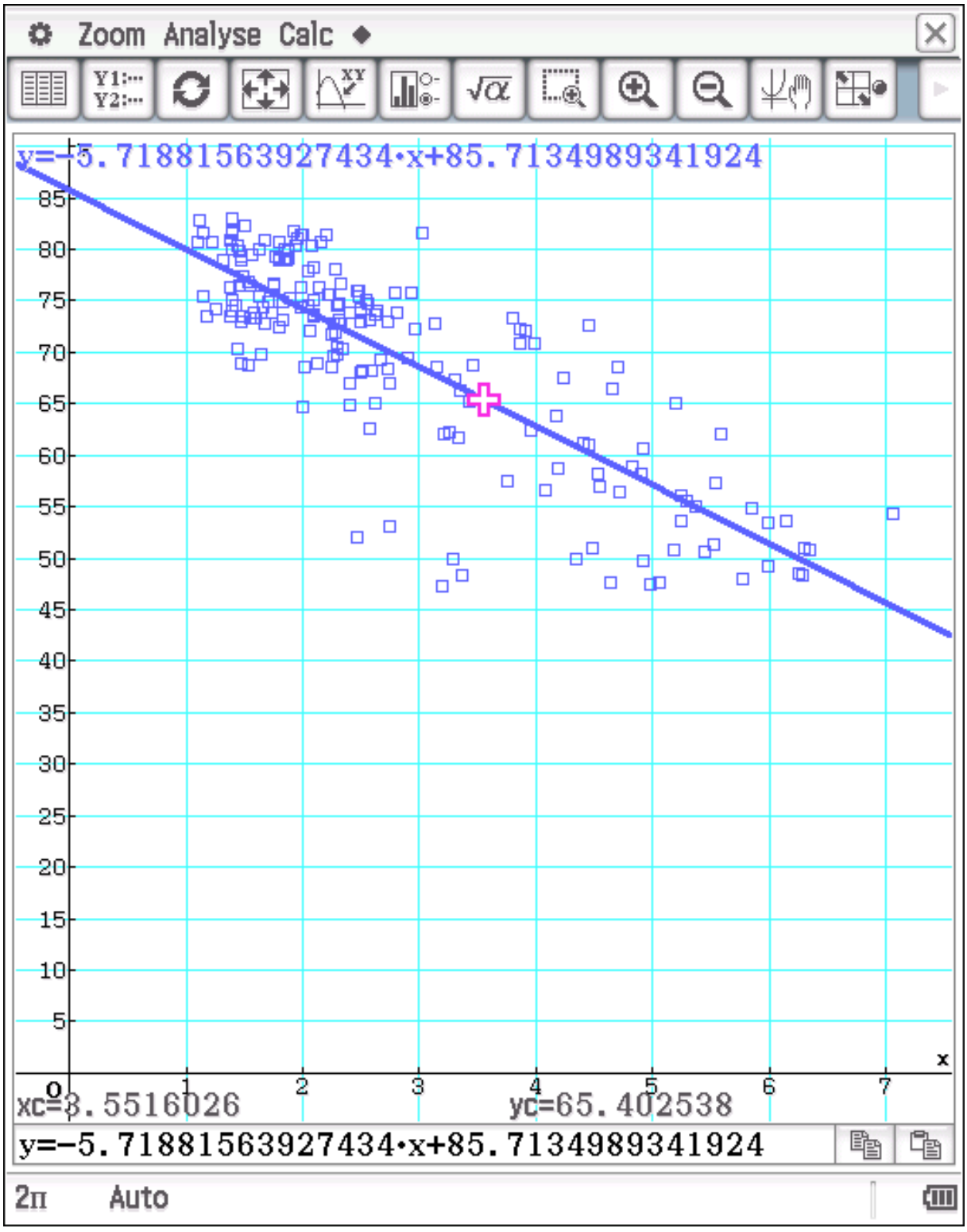
Download für dieses Dokument:

www.informatik.htw-dresden.de/

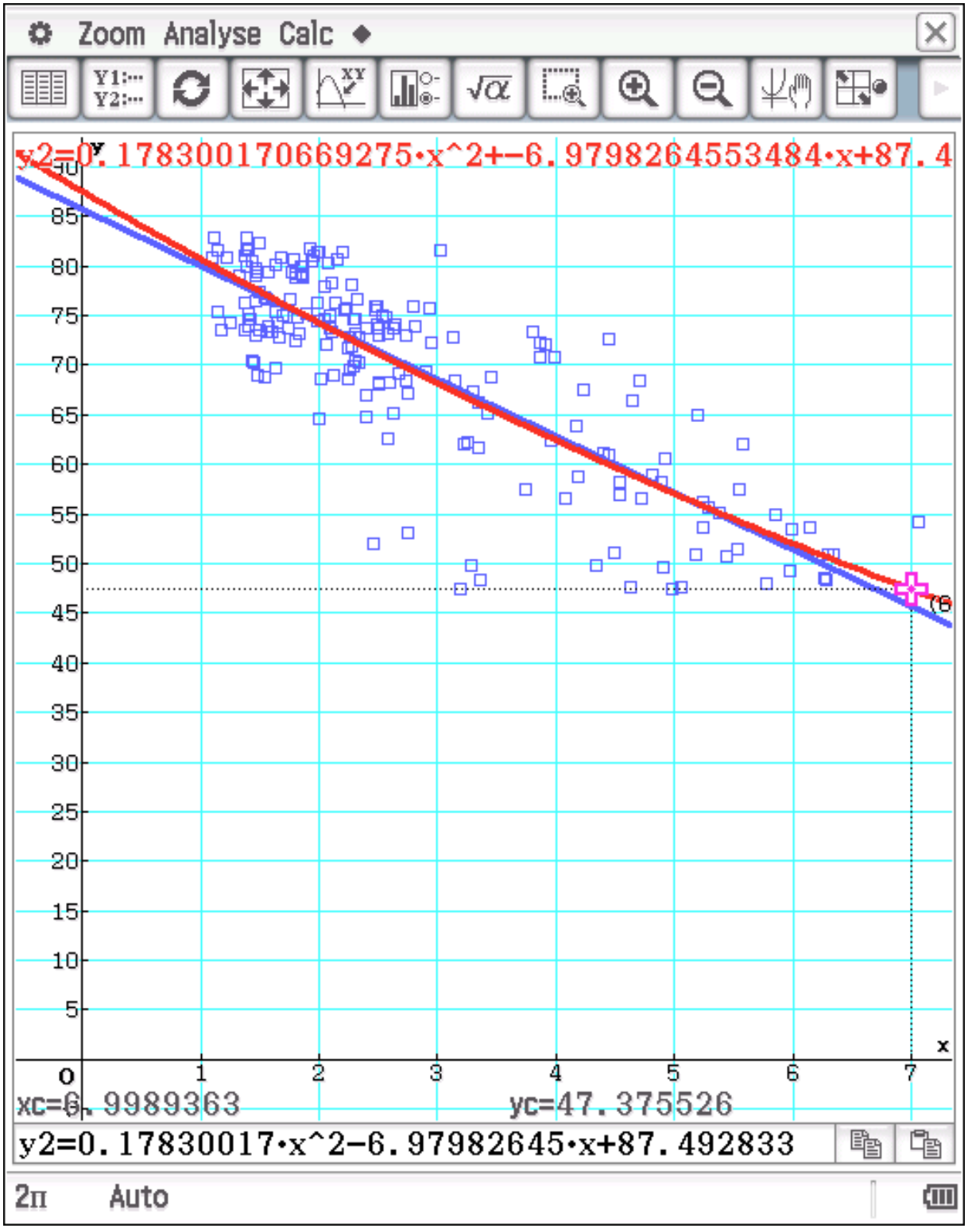
~paditz/Tensorflow-Ue07.pdf

Lineare Regression, optimale Lösung




Zoom Analyse Calc
y=−5.718815639274 34·x+85.7134989341924
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
1
2
3
4
5
6
7
x
xc=3.5516026
yc=65.402538
y=−5.718815639274 34·x+85.7134989341924
2π
Auto

Mit quadratischer Regression:



Ergebnis mit Tensorflow: Abbruch nach 1001 Schritten

y = w * x + b mit

w = -5.769937992095947

b = 85.66625213623047

optimale Lösung: mit ClassPad

y = -5.71881563927434 * x + 85.7134989341924

Grafik mit tensorflow: