Prof. Dr. L. Paditz, 14.02.2019

HTW Dresden,

paditz@htw-dresden.de

Mathematik-Grundlagen als eActivty mit ClassPad

quadrat. Regression in TensorFlow

==

Datenquelle:

https://college.cengage.com/mathematics/brase/

understandable_statistics/7e/students/datasets/

slr/frames/slr05.html

Feuer und Diebstähle in Chicago

In den folgenden 42 Datenpaaren bedeuten:

X = Anzahl Feuer pro 1000 Wohneinheiten

Y = Anzahl Diebstähle pro 1000 Einwohner

innerhalb derselben Postleitzahl im Chicagoer

Metro–Bereich

X_liste:={6.2,9.5,10.5,7.7,8.6,34.1,11,6.9,7.3,▶

{6.2,9.5,10.5,7.7,8.6,34.1,11,6.9,7.3,15.1,2▶

Y_liste:={29,44,36,37,53,68,75,18,31,25,34,14,▶

{29,44,36,37,53,68,75,18,31,25,34,14,11,11,2▶

dim(X_liste)=dim(Y_liste)

42=42

X_liste und Y_liste sind verbundene Datenlisten

STAT

Regression mit ClassPad:

LinearReg X_liste,Y_liste,1,y1

done

DispStat

done

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

Lineare Regression

y=a•x+b

a = 1.313456

b = 16.995157

r = 0.5511213

r^2 = 0.3037347

MSe = 378.86116

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

$$\mathbf{MSe}=\frac{1}{n-2}\sum_{k=1}^{n}\left((Y_k-(w*X_k+b))^2\right)\rightarrow \text{min!}$$

(Bem.: Normierung mit n-2)
stop

QuadReg X_liste, Y_liste, 1, y2

done

DispStat

done

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

Quadratische Reg.

$y=a\cdot x^2+b\cdot x+c$

a = 0.0519528

b = -0.604882

c = 28.234184

r^2 = 0.3585458

MSe = 357.98629

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

stop

Ansatz:

$Y = w*X^2 + u*X + b$

gesucht sind die Skalen-Parameter w, u und b für eine optimale Modellanpassung.

Wir verwenden den mittleren quadratischen Fehler (MSe) als **Verlustfunktion.**

Kommen wir zum Programm:

$$\textbf{MSe}=\frac{1}{n-3}\sum_{k=1}^{n}\left((Y_k-(w\cdot X^2{}_k+u*X_k+b))^2\right)\rightarrow \min!$$

Sei (ohne Vorfaktor $\frac{1}{n-3}$)

$$f(w,u,b)=\sum_{k=1}^{n}\left((Y_k-(w\cdot X^2{}_k+u*X_k+b))^2\right)$$

notwendige Bedingung für min!:

$$grad(f)=\begin{bmatrix}\frac{d}{dw}(f)\\ \frac{d}{du}(f)\\ \frac{d}{db}(f)\end{bmatrix}=\begin{bmatrix}0\\0\\0\end{bmatrix}$$

$$\frac{d}{dw}(f)=2\sum_{k=1}^{n}((Y_k-(w\cdot X^2{}_k+u*X_k+b))\cdot(-X^2{}_k))=0$$

$$\frac{d}{du}(f)=2\sum_{k=1}^{n}((Y_k-(w\cdot X^2{}_k+u*X_k+b))\cdot(-X_k))=0$$

$$\frac{d}{db}(f)=2\sum_{k=1}^{n}((Y_k-(w\cdot X^2{}_k+u*X_k+b))\cdot(-b))=0$$

weiter

sum((Y_liste−w•X_liste^2−u•X_liste−b)•X_liste^2)=0

−1576.09•(b+39.7•u+1576.09•w−147)−1310.44•(b+3▶

simplify(ans)➔Equ1

−1E-4•(105985900•b+2760225150•u+8.341208253E+1▶

sum((Y_liste−w•X_liste^2−u•X_liste−b)•X_liste)=0

−39.7•(b+39.7•u+1576.09•w−147)−36.2•(b+36.2•u+▶

simplify(ans)➔Equ2

−5E-3•(106620•b+2119718•u+55204503•w−4596180)=0

sum((Y_liste−w•X_liste^2−u•X_liste−b)•b)=0

−b•(b+39.7•u+1576.09•w−147)−b•(b+36.2•u+1310.4▶

simplify(ans)➔Equ3

−0.01•b•(4200•b+53310•u+1059859•w−141400)=0

$$\begin{cases} Equ1 \\ Equ2 \\ Equ3 \end{cases} \Bigg|_{w,u,b}$$

{{b=0,u=2.910920812,w=−0.02851490651},{b=28.

stop

analytische Lösung:

b=28.23418422,**u=**−0.6048816894,**w=**0.051952825

$\mathrm{sum}\left(\left(Y\_liste - w \cdot X\_liste^{2} - u \cdot X\_liste - b\right)^{2} \mid \{b=28.234184\right.$

13961.46545

MSe: (Normierung bei quadr. Regress. mit n−3, drei Parameter)

approx(ans/(42−3))

357.9862936

stop

loss:=13961.46545

mit den Skalenparametern von tensorflow (1000000 Iterationen):

w=0.05144852,u=−0.58288596,b=28.05823618

$\mathrm{sum}\left(\left(Y\_liste - w \cdot X\_liste^{2} - u \cdot X\_liste - b\right)^{2} \mid \{w=0.0514485\right.$

13961.64573

Normierung in tensorflow mit n statt n−3:

approx(ans/(42−0))

332.4201363

loss:=tf.square(model−y,name="loss")

loss:=13961.64573

tf.reduce_mean ist **günstigere Loss-Funktion**, um Zahlenbereichsüberschreitung vorzubeugen:

loss:=**tf.reduce_mean**(tf.square(model-y,name="loss" ▶

mean(loss)=332.4201363

weitere Variationsmöglichkeiten für das Gradientenverfahren:

Startwerte variieren:

w=tf.Variable([**0.05**], dtype=tf.float64)

u=tf.Variable([**-0.6**], dtype=tf.float64)

b=tf.Variable([**28.0**], dtype=tf.float64)

$\text{sum}((\text{Y\_liste}-w \cdot \text{X\_liste}^2-u \cdot \text{X\_liste}-b)^2 \mid \{w=0.05,u=-0$ ▶

13999.04313

mean(loss):

approx(ans/(42-0))

333.3105508

learning_rate (Schrittweite) variieren (um Divergenz des Verfahrens zu verhindern)

z.B.: **0.000001**

optimizer =

tf.train.GradientDescentOptimizer(**0.000001**)

Anzahl(**echo**) der Iterationen variieren: z.B.**1000000**

for i in range(100000):

stop

Rechnerprotokoll: (direkte Datenvorgabe)

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

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
>>>
>>> # Model parameters
...
>>> w = tf.Variable([0.05], dtype=tf.float64)
>>> u = tf.Variable([-0.6], dtype=tf.float64)
>>> b = tf.Variable([28.0], dtype=tf.float64)
>>> # Model input and output
...
>>> x = tf.placeholder(tf.float64)
>>> model = w*(x**2)+u*x+b
>>> y = tf.placeholder(tf.float64)
>>>
>>> # loss: sum of the squares
...
>>> loss = tf.reduce_mean(tf.square(model - y, name
= "loss"))
>>>
>>> # optimizer
... optimizer =
tf.train.GradientDescentOptimizer(0.000001)
>>> train = optimizer.minimize(loss)
>>>
>>> # training data
... # x_train = [0, 1, 2, 3]
```

```
...
x_train=[[6.2,9.5,10.5,7.7,8.6,34.1,11,6.9,7.3
>>> # y_train = [0, 1, 4, 9]
...
y_train=[[29,44,36,37,53,68,75,18,31,25,34,14,
>>> # training loop
... init = tf.global_variables_initializer()
>>> sess = tf.Session()
>>> sess.run(init)
>>> for i in range(100000):
...     sess.run(train, {x: x_train, y: y_train})
...     curr_w, curr_u, curr_b, curr_loss =
sess.run([w, u, b, loss], {x: x_train, y: y_train})
...     if i%10000 == 0:
...         print("Formula: %s x^2 + %s x + %s loss:
%s"%(curr_w, curr_u, curr_b, curr_loss))
...
Formula: [0.05082969] x^2 + [-0.59997085] x +
[28.00000133] loss: 332.7583462328666
Formula: [0.05168453] x^2 + [-0.58786257] x +
[28.00215277] loss: 332.4282250330093
Formula: [0.05148727] x^2 + [-0.58195522] x +
[28.00353028] loss: 332.4243911110546
Formula: [0.05139095] x^2 + [-0.57908341] x +
[28.00452629] loss: 332.4234296694606
Formula: [0.05134439] x^2 + [-0.57770756] x +
[28.0053334] loss: 332.4231661284731
Formula: [0.05132234] x^2 + [-0.57706889] x +
[28.00604655] loss: 332.4230722735666
```

```
Formula: [0.05131238] x^2 + [-0.57679344] x +
[28.00671253] loss: 332.42301979854915
Formula: [0.05130837] x^2 + [-0.5766969] x +
[28.00735441] loss: 332.42297753647733
Formula: [0.05130729] x^2 + [-0.57668842] x +
[28.00798356] loss: 332.4229379159028
Formula: [0.05130765] x^2 + [-0.57672323] x +
[28.00860557] loss: 332.42289909681705
>>> # evaluate training accuracy
... curr_w, curr_u, curr_b, curr_loss =
sess.run([w, u, b, loss], {x: x_train, y: y_train})
>>> print("Formula: %s x^2 + %s x + %s loss:
%s"%(curr_w, curr_u, curr_b, curr_loss))
Formula: [0.05130872] x^2 + [-0.57677928] x +
[28.00922316] loss: 332.42286063523807
>>>
```

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

Eingabe-Skript zum Kopieren in das Terminal:
nach dem Start von **Python3** >>>

```
import tensorflow as tf

# Model parameters

w = tf.Variable([0.05], dtype=tf.float64)
u = tf.Variable([-0.6], dtype=tf.float64)
b = tf.Variable([28.0], dtype=tf.float64)
# Model input and output
```

```
x = tf.placeholder(tf.float64)
model = w*(x**2)+u*x+b
y = tf.placeholder(tf.float64)

# loss: sum of the squares

loss = tf.reduce_mean(tf.square(model - y, name =
"loss"))

# optimizer
optimizer =
tf.train.GradientDescentOptimizer(0.000001)
train = optimizer.minimize(loss)

# training data
# x_train = [0, 1, 2, 3]
x_train =
[6.2,9.5,10.5,7.7,8.6,34.1,11,6.9,7.3,15.1,2
# y_train = [0, 1, 4, 9]
y_train =
[29,44,36,37,53,68,75,18,31,25,34,14,11,11,2
# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
for i in range(100000):
  sess.run(train, {x: x_train, y: y_train})
  curr_w, curr_u, curr_b, curr_loss = sess.run([w,
```

```
u, b, loss], {x: x_train, y: y_train})
  if i%10000 == 0:
     print("Formula: %s x^2 + %s x + %s loss:
%s"%(curr_w, curr_u, curr_b, curr_loss))

# evaluate training accuracy
curr_w, curr_u, curr_b, curr_loss = sess.run([w, u,
b, loss], {x: x_train, y: y_train})
print("Formula: %s x^2 + %s x + %s loss:
%s"%(curr_w, curr_u, curr_b, curr_loss))
```

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

Eingabe-Skript zum Kopieren in das Terminal:
nach dem Start von **Python3** >>>
das Daten-file **fire_theft.xls**
befindet sich im Verzeichnis
/home/parallels/DATA_DIR/fire_theft.xls
=======================================

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import xlrd
DATA_FILE =
"/home/parallels/DATA_DIR/fire_theft.xls"
# Step 1: read in data from the .xls file

book = xlrd.open_workbook(DATA_FILE ,
```

```
encoding_override = "utf-8")
sheet = book.sheet_by_index(0)
data = np.asarray([sheet.row_values(i) for i in range
(1, sheet.nrows)])
n_samples = sheet.nrows - 1
# Step 2: create placeholders for input X (number of
fire) and label Y (number of theft)

X = tf.placeholder(tf.float32, name = "X")
Y = tf.placeholder(tf.float32, name = "Y")
# Step 3: create variables: weights_1,weights_2 and
bias. All are initialized to:

w = tf.Variable(0.5, name = "weights_1")
u = tf.Variable(-0.6, name = "weights_2")
b = tf.Variable(28.0, name = "bias")
# Step 4: construct model to predict Y (number of
theft) from the number of fire

Y_predicted = tf.square(X) * w + X * u + b
# Step 5: Profit! Use the square error as the loss
function

loss = tf.reduce_mean(tf.square(Y - Y_predicted,
name = "loss"))
optimizer =
tf.train.GradientDescentOptimizer(learning_rate =
0.0000001).minimize(loss)
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 i in range(10): # run 100 or 1000 epochs
  for x, y in data:
    # Session runs train_op to minimize loss
    sess.run(optimizer, feed_dict ={X: x, Y: y})
# Step 9: output the values of w,u and b
print(sess.run([w,u,b]))
```

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

Rechnerprotokoll: (Daten über xls-file abrufen)
================

```
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 numpy as np
>>> import matplotlib.pyplot as plt
>>> import tensorflow as tf
>>> import xlrd
>>> DATA_FILE =
"/home/parallels/DATA_DIR/fire_theft.xls"
>>> # Step 1: read in data from the .xls file
...
```

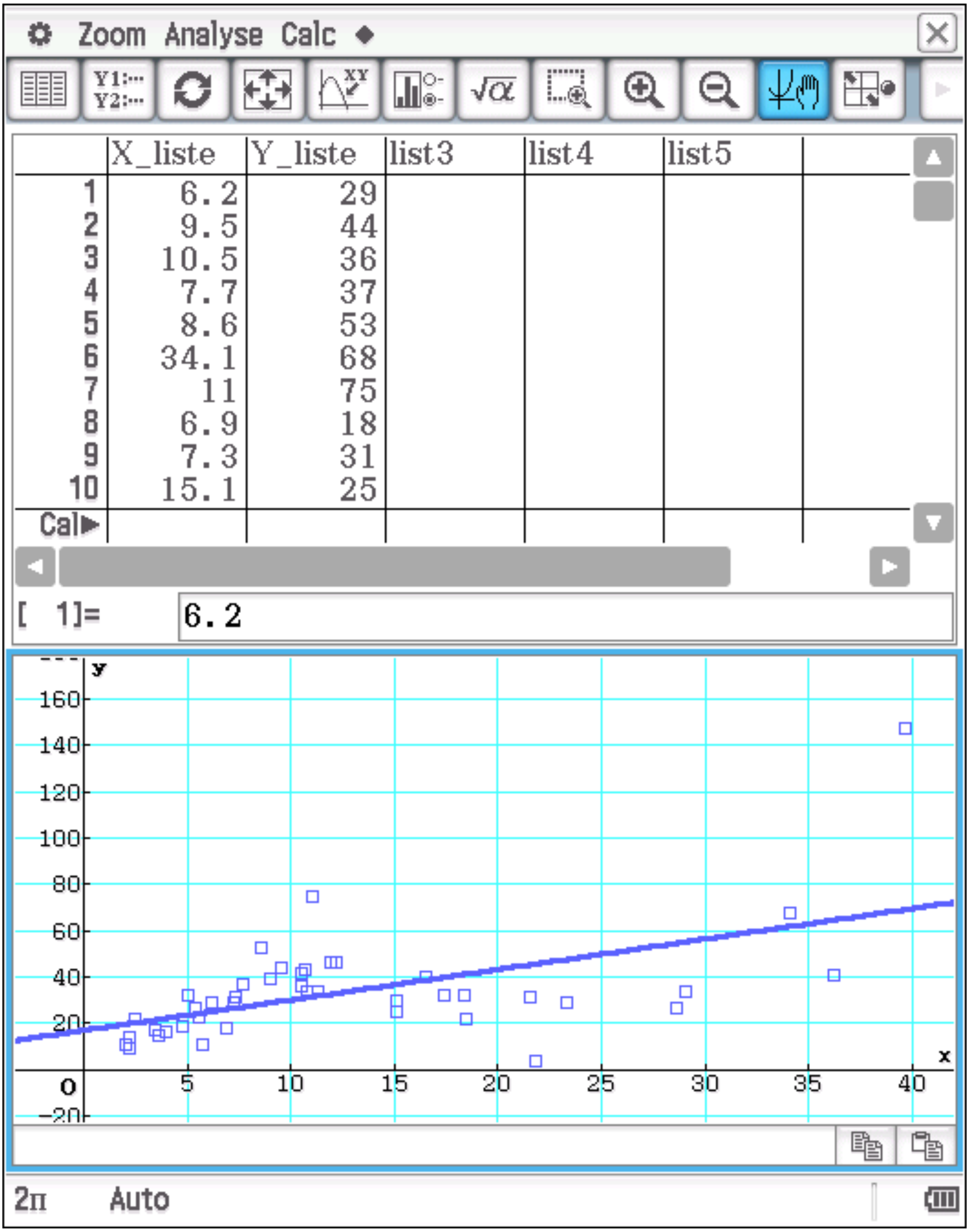
```
>>> book = xlrd.open_workbook(DATA_FILE ,
encoding_override = "utf-8")
>>> sheet = book.sheet_by_index(0)
>>> data = np.asarray([sheet.row_values(i) for i in
range (1, sheet.nrows)])
>>> n_samples = sheet.nrows - 1
>>> # Step 2: create placeholders for input X (number
of fire) and label Y (number of theft)
...
>>> X = tf.placeholder(tf.float32, name = "X")
>>> Y = tf.placeholder(tf.float32, name = "Y")
>>> # Step 3: create variables: weights_1,weights_2
and bias. All are initialized to:
...
>>> w = tf.Variable(0.5, name = "weights_1")
>>> u = tf.Variable(-0.6, name = "weights_2")
>>> b = tf.Variable(28.0, name = "bias")
>>> # Step 4: construct model to predict Y (number
of theft) from the number of fire
...
>>> Y_predicted = tf.square(X) * w + X * u + b
>>> # Step 5: Profit! Use the square error as the loss
function
...
>>> loss = tf.reduce_mean(tf.square(Y - Y_predicted,
name = "loss"))
>>> optimizer =
tf.train.GradientDescentOptimizer(learning_rate =
0.0000001).minimize(loss)
```

```
>>> 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 i in range(10): # run 100 or 1000
epochs
...         for x, y in data:
...           # Session runs train_op to minimize loss
...           sess.run(optimizer, feed_dict ={X: x,
Y: y})
...       # Step 9: output the values of w,u and b
...       print(sess.run([w,u,b]))
...
...
[0.05540445, -0.6147595, 27.999498]
>>>
```

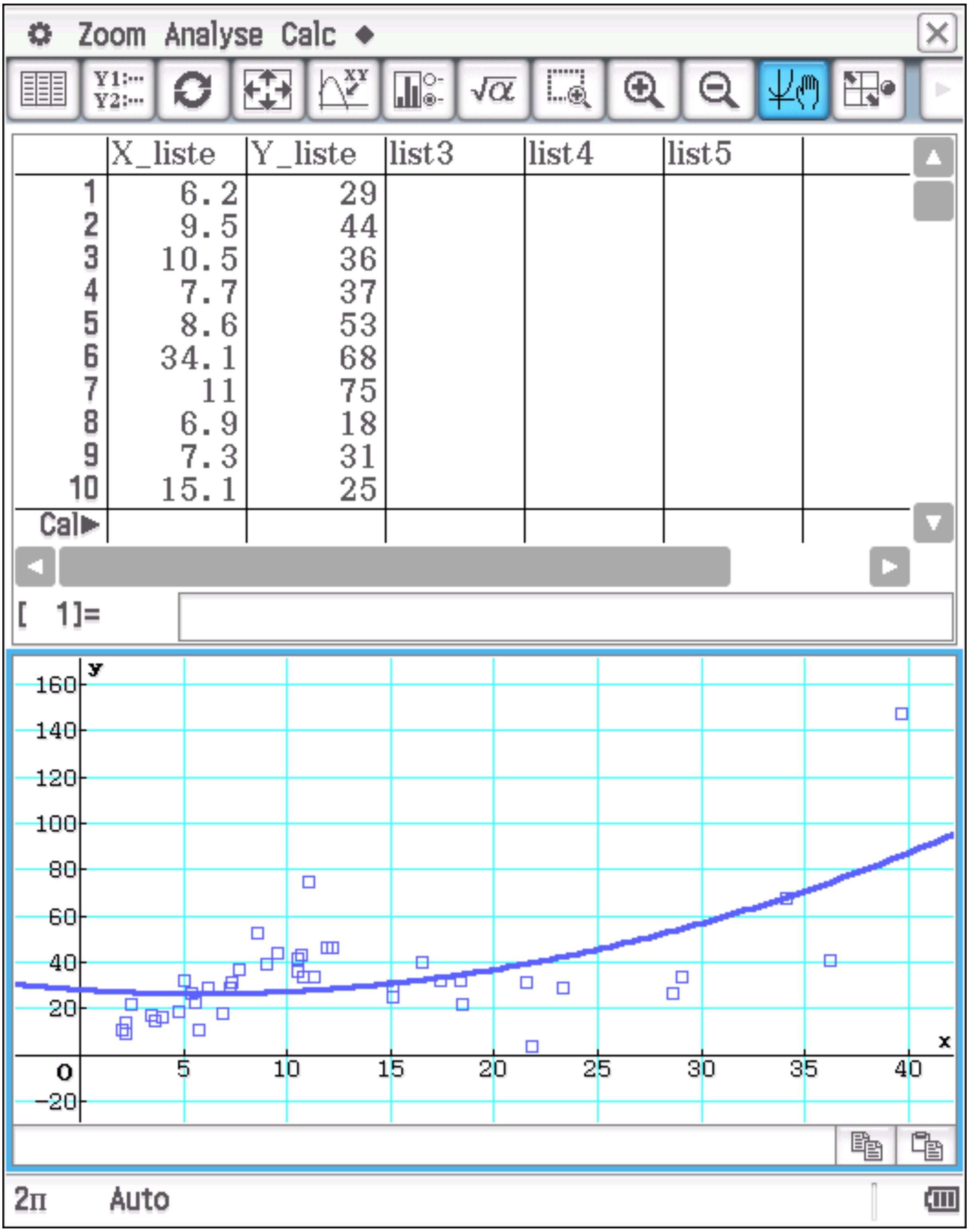
Quelle für das Skript:
http://web.stanford.edu/class/cs20si/
lectures/notes_03.pdf

Download für dieses Dokument:
www.informatik.htw-dresden.de/
~paditz/Tensorflow-Ue05.pdf

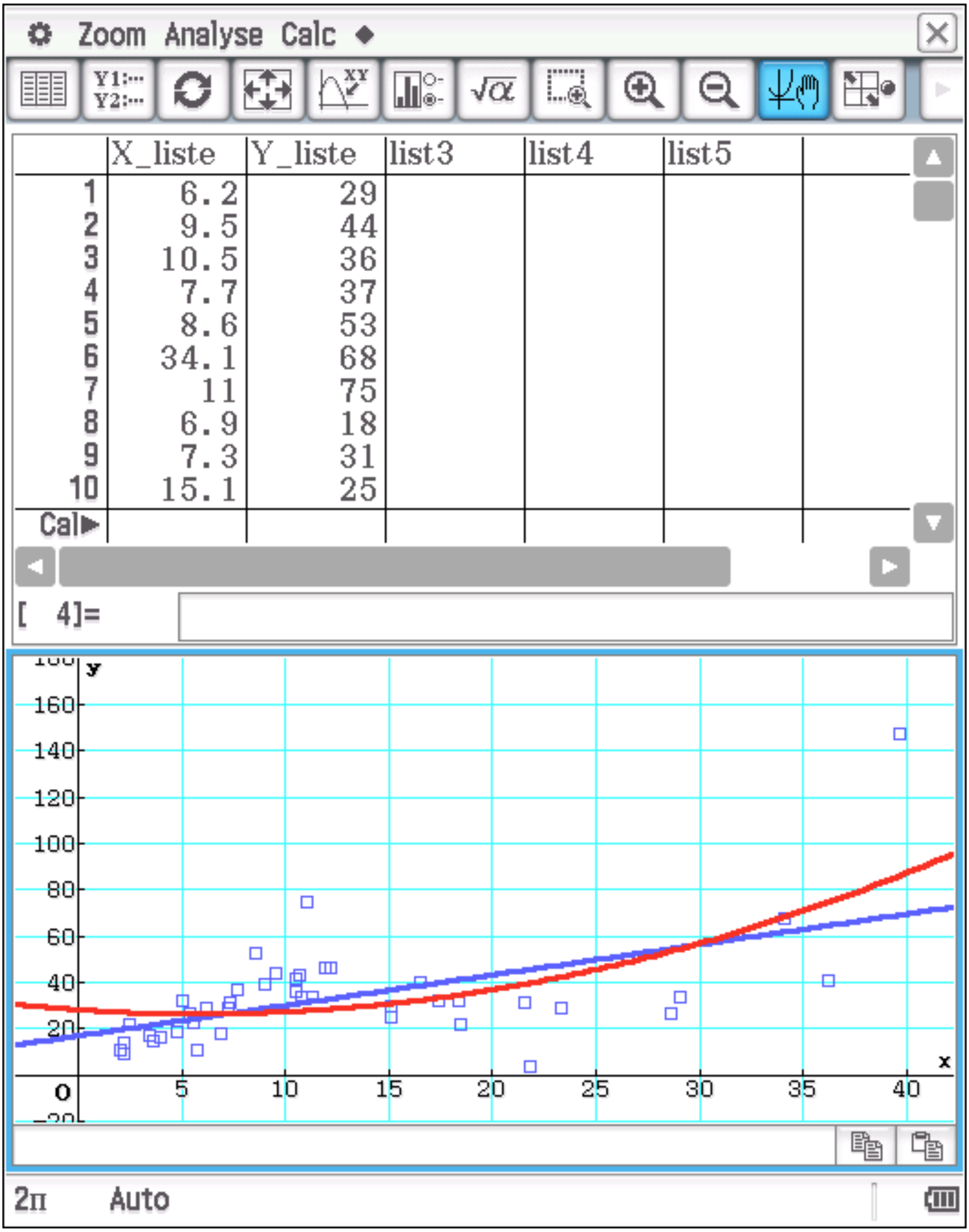
Lineare Regression: MSe = 378.86116



Quadratische Regression: MSe = 357.98629



Beide Regressionsfunktionen im Vergleich:



	X_liste	Y_liste	list3	list4	list5
1	6.2	29			
2	9.5	44			
3	10.5	36			
4	7.7	37			
5	8.6	53			
6	34.1	68			
7	11	75			
8	6.9	18			
9	7.3	31			
10	15.1	25			
Cal▶					

X	Y
6.2	29
9.5	44
10.5	36
7.7	37
8.6	53
34.1	68
11	75
6.9	18
7.3	31
15.1	25
29.1	34
2.2	14
5.7	11
2	11
2.5	22
4	16
5.4	27
2.2	9
7.2	29
15.1	30
16.5	40
18.4	32
36.2	41
39.7	147
18.5	22
23.3	29
12.2	46
5.6	23
21.8	4
21.6	31
9	39
3.6	15
5	32
28.6	27
17.4	32
11.3	34
3.4	17
11.9	46
10.5	42
10.7	43
10.8	34
4.8	19