Prof. Dr. L. Paditz, 15.02.2019

HTW Dresden,

paditz@htw-dresden.de

Lineare Regression mit Grafik

http://harmanani.github.io/classes/csc498r/Notes/Lect

python3

```
# %% imports %matplotlib inline
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
# %% Let's create some toy data
plt.ion()
n_observations = 100
fig, ax = plt.subplots(1, 1)
xs = np.linspace(-3, 3, n_observations)
```

```
ys = np.sin(xs) + np.random.uniform(-0.5, 0.5,
n_observations)
ax.scatter(xs, ys)
fig.show()
plt.draw()
# %% tf.placeholders for the input and output of the
network.
# Placeholders are variables which we need to fill in
when we
# are ready to compute the graph.
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
# %% We will try to optimize min_(W,b) ||(X*w +
b) - y||^2
# The `Variable()` constructor requires an initial value
for the
# variable,, which can be a `Tensor` of any type and
shape. The
```

initial value defines the type and shape of the variable.

After construction, the type and shape of the variable are

fixed. The value can be changed using one of the assign methods.

W = tf.Variable(tf.random_normal([1]), name='weight')

b = tf.Variable(tf.random_normal([1]), name='bias')

Y_pred = tf.add(tf.multiply(X, W), b)

%% Loss function will measure the distance between our observations

and predictions and average over them.

cost = tf.reduce_sum(tf.pow(Y_pred - Y, 2)) / (n_observations - 1)

%% Use gradient descent to optimize W,b

```
# Performs a single step in the negative gradient
learning_rate = 0.01
optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimi


# %% We create a session to use the graph
n_epochs = 1000
init = tf.global_variables_initializer()
with tf.Session() as sess:
    # Here we tell tensorflow that we want to initialize
all
    # the variables in the graph so we can use them
    sess.run(init)
    print(xs)
    print(ys)
    # Fit all training data
    prev_training_cost = 0.0
    for epoch_i in range(n_epochs):
```

```
        for (x, y) in zip(xs, ys):
            sess.run(optimizer, feed_dict={X: x, Y:
y})
        training_cost = sess.run(cost, feed_dict={X:
xs, Y: ys})
        curr_W, curr_b = sess.run([W, b])
        print(curr_W, curr_b, training_cost)
        if epoch_i % 20 == 0:
            ax.plot(xs, Y_pred.eval(feed_dict={X:
xs}, session=sess), 'k', alpha=epoch_i / n_epochs)
            fig.show()
            plt.draw()
        # Allow the training to quit if we've reached
a minimum
        if np.abs(prev_training_cost - training_cost)
< 0.000001:
            break
        prev_training_cost = training_cost
```

fig.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.
>>> # %% imports %matplotlib inline
... import numpy as np
>>> import tensorflow as tf
>>> import matplotlib.pyplot as plt
>>> # %% Let's create some toy data
... plt.ion()
>>> n_observations = 100
>>> fig, ax = plt.subplots(1, 1)
>>> xs = np.linspace(-3, 3, n_observations)
```

```
>>> ys = np.sin(xs) + np.random.uniform(-0.5,
0.5, n_observations)
>>> ax.scatter(xs, ys)
<matplotlib.collections.PathCollection object at
0x7f560590bbe0>
>>> fig.show()
>>> plt.draw()
>>> # %% tf.placeholders for the input and output of
the network.
... # Placeholders are variables which we need to fill
in when we
... # are ready to compute the graph.
... X = tf.placeholder(tf.float32)
>>> Y = tf.placeholder(tf.float32)
>>> # %% We will try to optimize min_(W,b) ||(X*w
+ b) - y||^2
... # The `Variable()` constructor requires an initial
value for the
```

```
... # variable,, which can be a `Tensor` of any type
and shape. The
... # initial value defines the type and shape of the
variable.
... # After construction, the type and shape of the
variable are
... # fixed. The value can be changed using one of
the assign methods.
... W = tf.Variable(tf.random_normal([1]),
name='weight')
>>> b = tf.Variable(tf.random_normal([1]),
name='bias')
>>> Y_pred = tf.add(tf.multiply(X, W), b)
>>>
>>> # %% Loss function will measure the distance
between our observations
... # and predictions and average over them.
... cost = tf.reduce_sum(tf.pow(Y_pred - Y, 2)) /
```

```
(n_observations - 1)
>>>
>>> # %% Use gradient descent to optimize W,b
... # Performs a single step in the negative gradient
... learning_rate = 0.01
>>> optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimi
>>>
>>> # %% We create a session to use the graph
... n_epochs=1000
>>> init = tf.global_variables_initializer()
>>> with tf.Session() as sess:
...     # Here we tell tensorflow that we want to
initialize all
...     # the variables in the graph so we can use
them
...     sess.run(init)
...     print(xs)
```

```
...        print(ys)
...        # Fit all training data
...        prev_training_cost = 0.0
...        for epoch_i in range(n_epochs):
...            for (x, y) in zip(xs, ys):
...                sess.run(optimizer, feed_dict={X:
x, Y: y})
...            training_cost = sess.run(cost,
feed_dict={X: xs, Y: ys})
...            curr_W, curr_b = sess.run([W, b])
...            print(curr_W, curr_b, training_cost)
...            if epoch_i % 20 == 0:
...                ax.plot(xs,
Y_pred.eval(feed_dict={X: xs}, session=sess), 'k',
alpha=epoch_i / n_epochs)
...                fig.show()
...                plt.draw()
...            # Allow the training to quit if we've
```

```
reached a minimum
...            if np.abs(prev_training_cost -
training_cost) < 0.000001:
...                break
...            prev_training_cost = training_cost
...
[-3.          -2.93939394 -2.87878788
-2.81818182 -2.75757576 -2.6969697
 -2.63636364 -2.57575758 -2.51515152
-2.45454545 -2.39393939 -2.33333333
 -2.27272727 -2.21212121 -2.15151515
-2.09090909 -2.03030303 -1.96969697
 -1.90909091 -1.84848485 -1.78787879
-1.72727273 -1.66666667 -1.60606061
 -1.54545455 -1.48484848 -1.42424242
-1.36363636 -1.3030303  -1.24242424
 -1.18181818 -1.12121212 -1.06060606 -1.
   -0.93939394 -0.87878788
```

```
 -0.81818182 -0.75757576 -0.6969697
-0.63636364 -0.57575758 -0.51515152
 -0.45454545 -0.39393939 -0.33333333
-0.27272727 -0.21212121 -0.15151515
 -0.09090909 -0.03030303  0.03030303
0.09090909  0.15151515  0.21212121
  0.27272727  0.33333333  0.39393939
0.45454545  0.51515152  0.57575758
  0.63636364  0.6969697   0.75757576
0.81818182  0.87878788  0.93939394
  1.          1.06060606  1.12121212
1.18181818  1.24242424  1.3030303
  1.36363636  1.42424242  1.48484848
1.54545455  1.60606061  1.66666667
  1.72727273  1.78787879  1.84848485
1.90909091  1.96969697  2.03030303
  2.09090909  2.15151515  2.21212121
2.27272727  2.33333333  2.39393939
```

```
  2.45454545  2.51515152  2.57575758
2.63636364  2.6969697   2.75757576
  2.81818182  2.87878788  2.93939394  3.
  ]
[ 0.20291621 -0.22118012 -0.62833878
0.01444744 -0.27653231 -0.81966122
 -0.59394983 -0.68889316 -0.89094116
-0.64222339 -0.36528585 -0.31661621
 -0.37948344 -0.33735615 -1.06720325
-1.21275391 -0.71440054 -0.89515028
 -0.56801334 -0.98468848 -0.54954994
-1.0945503  -1.33175895 -1.34249212
 -0.65011852 -0.78674601 -1.35303092
-1.19645055 -1.1630676  -1.14722283
 -1.0047393  -1.1928673  -0.88925636
-0.95923765 -0.46293165 -1.2023979
 -0.62926533 -0.19748193 -1.05230669
-0.74460138 -0.8929727  -0.08115272
```

-0.24367908 -0.39987265 0.06714758

-0.05671209 -0.27908047 -0.63617708

-0.30698586 -0.21597079 -0.03565833

-0.27975359 -0.24541802 0.45953916

0.09202181 0.45763097 0.57728143

0.06150595 0.40199538 0.66282468

0.16705133 1.06981763 0.31727183

0.29093523 0.48856133 0.33590649

0.73689616 0.68987514 0.48259013

1.08477711 0.84286125 1.20258011

0.55556689 1.39794857 1.03707338

0.8518561 1.0564094 0.56761776

0.53057936 0.55085757 1.28483505

1.35616653 0.85892314 0.53569456

0.3758622 1.02227012 0.80742031

0.72057286 1.15398355 0.71426198

0.36634298 1.02687661 0.92441443

0.51944523 0.91051877 0.59040157

```
  0.32209332 -0.01796203  0.53824307
0.02899414]
[-1.4935356] [-0.13225833] 10.572385
[<matplotlib.lines.Line2D object at 0x7f904511cb70>]
[-1.3838066] [-0.13055438] 9.36893
[-1.2806613] [-0.12886626] 8.30555
[-1.1837045] [-0.12719482] 7.3659387
[-1.0925653] [-0.12554075] 6.535692
[-1.0068941] [-0.12390465] 5.802076
[-0.9263632] [-0.12228709] 5.153846
[-0.8506643] [-0.12068851] 4.581062
[-0.779507] [-0.11910937] 4.074939
[-0.71261907] [-0.11755012] 3.6277204
[-0.6497446] [-0.11601089] 3.2325506
[-0.5906425] [-0.11449206] 2.88337
[-0.53508717] [-0.11299378] 2.5748284
[-0.48286477] [-0.11151634] 2.30219
[-0.43377584] [-0.11005974] 2.0612786
```

```
[-0.3876326] [-0.10862413] 1.848402

[-0.3442579] [-0.1072096] 1.6602951

[-0.30348572] [-0.10581618] 1.4940751

[-0.26516014] [-0.10444393] 1.3471955

[-0.22913408] [-0.10309275] 1.2174037

[-0.19526978] [-0.10176254] 1.1027118

[<matplotlib.lines.Line2D object at 0x7f904513da58>]

[-0.1634373] [-0.10045338] 1.0013614

[-0.13351497] [-0.09916513] 0.9118

[-0.10538808] [-0.09789774] 0.83265585

[-0.07894888] [-0.09665104] 0.7627159

[-0.05409618] [-0.09542488] 0.7009094

[-0.03073477] [-0.09421916] 0.64628947

[-0.00877515] [-0.09303365] 0.5980197

[0.01186677] [-0.09186824] 0.55536115

[0.03127005] [-0.09072271] 0.5176608

[0.04950902] [-0.08959687] 0.48434162

[0.06665351] [-0.08849053] 0.4548939
```

```
[0.08276922] [-0.08740351] 0.42886704

[0.09791785] [-0.08633552] 0.40586302

[0.11215741] [-0.08528641] 0.38553023

[0.12554248] [-0.08425587] 0.36755776

[0.13812433] [-0.0832438] 0.35167104

[0.14995112] [-0.0822499] 0.33762753

[0.16106822] [-0.08127388] 0.32521266

[0.17151816] [-0.08031559] 0.31423715

[0.18134099] [-0.07937472] 0.3045336

[<matplotlib.lines.Line2D object at 0x7f90450b9048>]

[0.19057424] [-0.07845104] 0.2959541

[0.1992534] [-0.07754428] 0.28836796

[0.20741177] [-0.07665425] 0.28165957

[0.21508048] [-0.07578067] 0.275727

[0.22228895] [-0.07492331] 0.2704801

[0.22906479] [-0.0740819] 0.26583925

[0.23543398] [-0.07325623] 0.26173386

[0.24142094] [-0.072446] 0.2581019
```

```
[0.24704853] [-0.07165099] 0.25488836

[0.25233835] [-0.07087094] 0.25204465

[0.2573106] [-0.07010563] 0.24952786

[0.2619844] [-0.06935482] 0.24730006

[0.26637772] [-0.06861825] 0.24532771

[0.27050734] [-0.0678957] 0.24358118

[0.27438903] [-0.06718691] 0.24203433

[0.27803776] [-0.06649162] 0.24066404

[0.2814674] [-0.06580967] 0.23944986

[0.28469115] [-0.06514079] 0.23837371

[0.28772154] [-0.06448473] 0.2374196

[0.29056993] [-0.06384134] 0.23657347

[<matplotlib.lines.Line2D object at 0x7f90450bb128>]

[0.2932472] [-0.06321033] 0.23582289

[0.29576373] [-0.06259151] 0.23515676

[0.2981294] [-0.06198462] 0.23456533

[0.300353] [-0.06138948] 0.23404005

[0.3024431] [-0.06080589] 0.23357327
```

[0.30440748] [-0.06023363] 0.23315836

[0.30625394] [-0.05967252] 0.23278928

[0.30798975] [-0.05912232] 0.23246075

[0.30962116] [-0.05858284] 0.23216821

[0.3111545] [-0.0580539] 0.2319075

[0.3125959] [-0.05753528] 0.23167498

[0.31395072] [-0.05702679] 0.23146749

[0.31522417] [-0.05652824] 0.23128211

[0.3164212] [-0.05603946] 0.23111638

[0.31754622] [-0.05556026] 0.2309681

[0.31860372] [-0.05509046] 0.23083527

[0.31959772] [-0.05462988] 0.23071612

[0.32053193] [-0.05417835] 0.23060918

[0.32141] [-0.0537357] 0.23051307

[0.3222354] [-0.05330175] 0.23042655

[<matplotlib.lines.Line2D object at 0x7f90450bb470>]

[0.32301128] [-0.05287637] 0.2303486

[0.3237405] [-0.05245934] 0.23027825

[0.32442594] [-0.05205055] 0.23021467

[0.32507002] [-0.05164982] 0.23015712

[0.32567558] [-0.05125701] 0.23010492

[0.32624465] [-0.05087192] 0.23005757

[0.3267793] [-0.05049448] 0.23001446

[0.32728192] [-0.05012447] 0.22997521

[0.3277544] [-0.04976181] 0.22993937

[0.3281983] [-0.04940633] 0.2299066

[0.32861575] [-0.04905788] 0.22987656

[0.3290082] [-0.04871632] 0.22984898

[0.32937708] [-0.04838153] 0.22982359

[0.32972354] [-0.04805339] 0.2298002

[0.33004913] [-0.04773178] 0.22977862

[0.33035526] [-0.04741653] 0.22975864

[0.33064303] [-0.04710753] 0.22974013

[0.33091354] [-0.04680462] 0.22972289

[0.3311676] [-0.04650777] 0.22970688

[0.33140635] [-0.04621683] 0.22969195

```
[<matplotlib.lines.Line2D object at 0x7f90450c0198>]
[0.3316306] [-0.04593166] 0.22967803
[0.33184144] [-0.04565217] 0.22966497
[0.3320396] [-0.04537822] 0.22965272
[0.3322259] [-0.04510972] 0.22964121
[0.33240113] [-0.04484659] 0.2296304
[0.33256567] [-0.04458867] 0.22962023
[0.33272025] [-0.04433587] 0.22961059
[0.33286542] [-0.04408811] 0.2296015
[0.33300197] [-0.04384529] 0.22959289
[0.33313015] [-0.04360731] 0.22958477
[0.3332506] [-0.04337407] 0.22957702
[0.33336368] [-0.04314548] 0.22956967
[0.33347008] [-0.04292142] 0.22956267
[0.33357018] [-0.04270184] 0.229556
[0.33366412] [-0.04248665] 0.22954965
[0.33375254] [-0.04227573] 0.2295436
[0.33383548] [-0.04206905] 0.22953781
```

```
[0.3339135] [-0.04186647] 0.22953229

[0.33398664] [-0.04166792] 0.229527

[0.3340553] [-0.04147331] 0.22952192

[<matplotlib.lines.Line2D object at 0x7f90450c31d0>]

[0.3341199] [-0.04128261] 0.22951706

[0.33418056] [-0.04109569] 0.2295124

[0.33423772] [-0.04091252] 0.22950795

[0.33429137] [-0.040733] 0.22950365

[0.3343417] [-0.04055704] 0.22949953

[0.334389] [-0.04038462] 0.22949556

[0.33443338] [-0.04021563] 0.22949179

[0.3344751] [-0.04005002] 0.22948812

[0.33451432] [-0.03988773] 0.22948457

[0.334551] [-0.03972868] 0.2294812

[0.3345856] [-0.03957277] 0.22947793

[0.3346181] [-0.03941998] 0.22947481

[0.33464855] [-0.03927026] 0.22947179

[0.3346773] [-0.03912352] 0.22946885
```

```
[0.33470413] [-0.03897971] 0.22946607

[0.3347294] [-0.03883879] 0.22946331

[0.33475336] [-0.03870068] 0.2294607

[0.3347759] [-0.03856531] 0.22945818

[0.33479688] [-0.03843268] 0.22945575

[0.33481655] [-0.03830269] 0.22945338

[<matplotlib.lines.Line2D object at 0x7f90450c5198>]

[0.3348351] [-0.03817529] 0.22945113

[0.33485246] [-0.03805044] 0.22944891

[0.3348689] [-0.03792807] 0.2294468

[0.33488408] [-0.03780816] 0.22944477

[0.3348983] [-0.03769065] 0.22944279

[0.33491185] [-0.03757549] 0.2294409

[0.33492458] [-0.03746263] 0.22943905

[0.33493653] [-0.03735201] 0.22943726

[0.3349477] [-0.0372436] 0.22943555

[0.3349582] [-0.03713737] 0.22943386

[0.33496794] [-0.03703327] 0.22943226
```

```
[0.33497703] [-0.03693125] 0.22943069
[0.3349856] [-0.03683123] 0.2294292
[0.33499357] [-0.03673325] 0.22942773
[0.33500117] [-0.03663723] 0.22942631
[0.3350081] [-0.03654309] 0.22942497
[0.33501482] [-0.03645087] 0.22942366
[0.33502105] [-0.03636049] 0.22942236
[0.33502698] [-0.03627193] 0.22942114
[0.3350325] [-0.03618513] 0.22941992
[<matplotlib.lines.Line2D object at 0x7f90450c5550>]
[0.33503768] [-0.03610006] 0.22941877
[0.33504245] [-0.03601667] 0.22941762
[0.3350471] [-0.03593496] 0.22941655
[0.33505127] [-0.03585487] 0.2294155
[0.33505526] [-0.0357764] 0.2294145
[0.33505905] [-0.03569948] 0.22941352
>>> fig.show()
```

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

xs:=seq(x, x, −3, 3, $\frac{6}{99}$)

$\left\{-3, -\frac{97}{33}, -\frac{95}{33}, -\frac{31}{11}, -\frac{91}{33}, -\frac{89}{33}, -\frac{29}{11}, -\frac{85}{33}, -\frac{83}{33}, -\frac{27}{11}\right.$ ▶

approx(ans)

{−3, −2.939393939, −2.878787879, −2.818181818, − ▶

ys:=sin(xs)+randList(100)−0.5

$\left\{-\sin(3)+\frac{119201}{7432868}, -\sin\left(\frac{97}{33}\right)+\frac{2292690}{5420887}, -\sin\left(\frac{95}{33}\right)-\frac{6}{2}\right.$ ▶

approx(ans)

{−0.1250829952, 0.2221125851, −0.4918305463, −0 ▶

STAT−Menü

y1(x)

0.3279082301·x−0.03169543679

sum((y1(xs)−ys)^2)/99

0.3084205225

Daten von Tensorflow,

yss:={−0.18295909, 0.11708618, −0.40319602, ▶

{−0.18295909, 0.11708618, −0.40319602, −0.79582 ▶

LinearReg xs, yss, 1, y2

done

DispStat

done

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

Lineare Regression

y=a*x+b

$a = 0.3555142$

$b = -9.113E-3$

$r = 0.7722933$

$r^2 = 0.5964369$

$MSe = 0.2670806$

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

cost = 0.26442605

0.26442605*99/98

0.267124275

Anlage:

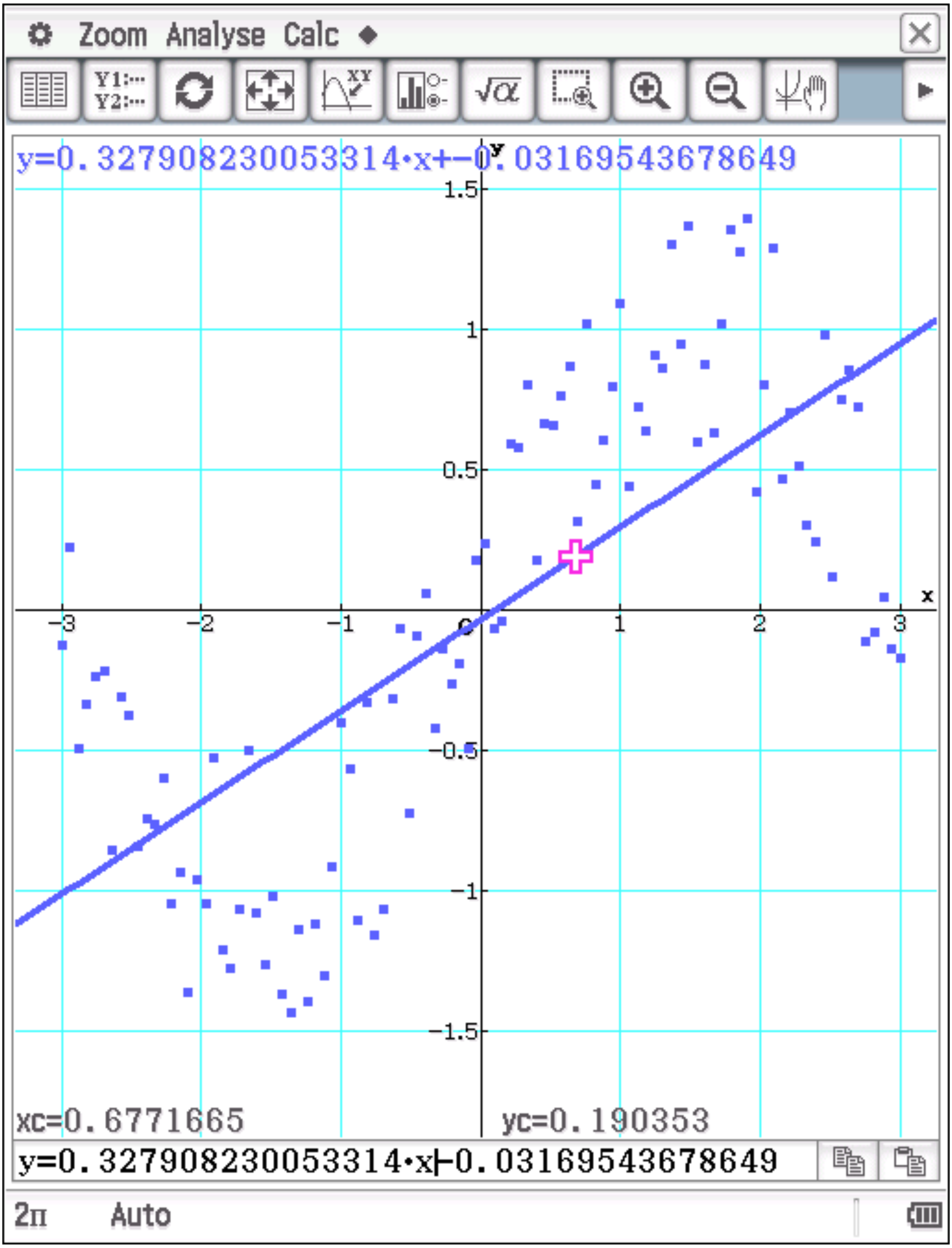
Bild mit verrauschten Daten bei linearer Regression

Download für dieses Dokument:

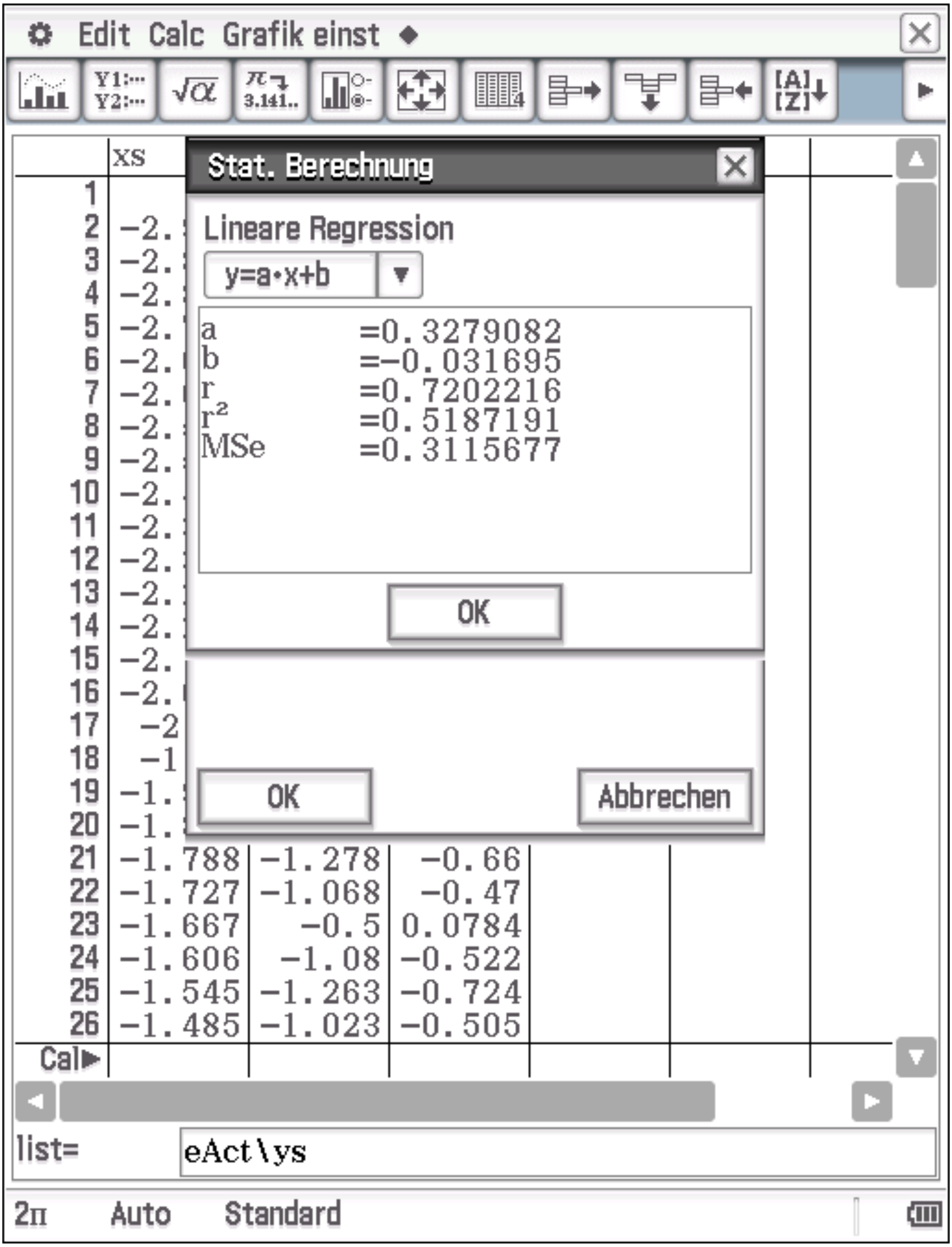
www.informatik.htw-dresden.de/
~paditz/Tensorflow-Ue13.pdf

Lineare Regression (sin-Daten verrauscht)

xs:=seq(x,x,-3,3,6/99) ,   ys:=sin(xs)+randList(100)-0.5



Lineare Regression (sin-Daten verrauscht)



Lineare Regression (sin-Daten verrauscht)

Mit tensorflow generiert:

schrittweise Iteration der Regress.-Geraden erkennbar: