Prof. Dr. L. Paditz, 19.03.2019

HTW Dresden,

paditz@htw-dresden.de

https://www.pyimagesearch.com/2019/01/30/

ubuntu-18-04-install-tensorflow-and-keras-for

-deep-learning/

Lineare Regression mit ganzzahligen Größen,

Messwerte (numbers of convictions) als

zufällig gestörte **Ausgangsgrößen** (evidence numbers)

==========

Zufallszahlen: np.random.randint(low=10, high=50,

size=sampleSize)

diskrete ganzzahlige Gleichverteilung im halboffenen

Intervall [low, high)

Quelle:

https://medium.freecodecamp.org/tensorflow-starter-on-law-and-statistics-646072b93b5a

https://github.com/Createdd/Writing/blob/master/2018/articles/LawStatisticsExample.md

s. auch:

https://de.wikipedia.org/wiki/**Matplotlib**

Inhalt:

Es werden per Zufall 200 Datenpaare

$(x_i, y_i), i=1(1)200$, generiert.

$y_i=10 \cdot x_i+\varepsilon_i$ sind verrauschte Daten.

Modellgleichung: $y(x_i)=m \cdot x_i+b$

70% der Daten werden für die Modellschätzung genutzt:

$(x_i, y_i), i=1(1)140$, "Trainingsdaten, Modell wird trainiert"

Die verbleibenden 30% der Daten (Testdaten, Modell

wird überprüft) werden durch die Modellschätzung gut mit erfasst, wenn alle Daten aus der gleichen Grundgesamtheit stammen!

konkret:

$x_i \in \{1, 2, 3, \ldots, 47, 48, 49\}$ ganzzahlig gleichverteilt aus $[1, 50)$ (halboffenes Intervall).

$\varepsilon_i \in \{200, 201, 202, \ldots, 398, 399\}$ ganzzahlig gleichverteilt aus $[200, 400)$ (halboffenes Intervall).

Der Pseudozufallszahlen-Generator wird an einem festen Punkt gestartet: np.random.seed(42)

Anwendung:

Schrifterkennung: ein von unterschiedlichen Personen handgeschriebener Buchstabe (oder Zahl) wird durch ein Modell beschrieben (z.B. binäre logistische Regression), d.h. das Modell wird trainiert. Eine hinzugekommene Person schreibt einen frei gewählten Buchstaben (oder Zahl). Das System erkennt anhand des trainierten

Modells mit hoher Wahrscheinlichkeit, ob es sich um den bekannten (eintrainierten) Buchstaben (oder Zahl) handelt oder nicht handelt.

vgl. auch Beispiel "Wecker-Lautstärke - Aufwachen/Nichtaufwachen" (binäre logistische Regression, Ue11)

ClassPad-Auswertung:

```
trainEvi:={48,38,24,17,30,48,28,32,20,20,33,45,▶
      {48,38,24,17,30,48,28,32,20,20,33,45,49,33,1▶
trainCon:={803,766,509,462,502,827,666,683,546▶
      {803,766,509,462,502,827,666,683,546,489,724▶
LinearReg trainEvi,trainCon,1,y1,On
                                                    done
DispStat
                                                    done
=====================
Lineare Regression
 y=a·x+b
     a = 10.236841
     b = 300.35717
     r = 0.90122
    r² = 0.8121975
 MSe = 3392.1928
```

==========

mean(trainEvi)

29.85714286

stdDev(trainEvi)

11.78925949

mean(trainCon)

606

stdDev(trainCon)

133.9126742

$$\text{trainEvN} := \frac{\text{trainEvi} - \text{mean}(\text{trainEvi})}{\text{stdDev}(\text{trainEvi})}$$

{1.538931021, 0.6907013245, -0.496820251, -1.09 ▶

$$\text{trainCoN} := \frac{\text{trainCon} - \text{mean}(\text{trainCon})}{\text{stdDev}(\text{trainCon})}$$

{1.471107953, 1.194808489, -0.7243526467, -1.07 ▶

LinearReg trainEvN, trainCoN, 1, y2, On

done

DispStat

done

==========

Lineare Regression

y=a·x+b

a = 0.9012199886

b = 4E-14

r = 0.90122

r^2 = 0.8121975

MSe = 0.1891634

==========

aCoef

0.9012199886

bCoef

$4E-14$

MSe

0.1891634201

MSe*138/280

0.09323054274

Tensorflow:

Trained cost = 0.09390127

a = evidFactor = 0.9012197

b = convictOffset = -1.4810865e-08

Die x-Daten wurden verzehnfacht und im Mittel mit 300 gestört:

y1(x)

$10.23684133 \cdot x + 300.3571661$

y2(x)

$0.9012199886 \cdot x + 4E-14$

testEvi:={11,15,37,37,29,39,20,37,34,48,42,10,
{11,15,37,37,29,39,20,37,34,48,42,10,36,22,1

testCon:={363,407,636,673,663,613,513,601,714,
{363,407,636,673,663,613,513,601,714,765,770

stop

STAT-Menü

==========

Skript:

python3

import tensorflow as tf

```
import numpy as np
import math
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# Generate evidence numbers between 10 and 50
# Generate a number of convictions from the evidence
with a random noise added
np.random.seed(42)
sampleSize = 200
numEvid = np.random.randint(low=10, high=50,
size=sampleSize)
numConvict = numEvid * 10 +
np.random.randint(low=200, high=400,
size=sampleSize)
print(numEvid)
print(numConvict)

# Plot the data to get a feeling
plt.title("Number of convictions based on evidence")
plt.plot(numEvid, numConvict, "bx")
plt.xlabel("Number of Evidence")
plt.ylabel("Number of Convictions")
# Use the keyword 'block' to override the blocking
behavior
plt.show(block=False)
# ######################
```

```
# create a function for normalizing values
def normalize(array):
    return (array - array.mean()) / array.std()

# define an animation function that changes the ydata
def animate(i):
    line.set_xdata(xNorm)
    line.set_ydata(
        (evidFactorAnim[i] * trainEvidNorm +
convictOffsetAnim[i]) * trainConvictStd
        + trainConvictMean
        )
    return (line,)

# Initialize the animation with zeros for y

def initAnim():

line.set_ydata(np.zeros(shape=numConvict.shape[0]))
    return (line,)

# use 70% of the data for training (the remaining 30%
shall be used for testing)

numTrain = math.floor(sampleSize * 0.7)

# convert list to an array and normalize arrays
# 70% of the data for training
```

```
trainEvid = np.asanyarray(numEvid[:numTrain])
trainConvict = np.asanyarray(numConvict[:numTrain])
trainEvidNorm = normalize(trainEvid)
trainConvictdNorm = normalize(trainConvict)
# print(trainEvidNorm)
# print(trainConvictdNorm)

# the remaining 30% shall be used for testing
testEvid = np.asanyarray(numEvid[numTrain:])
testConvict = np.asanyarray(numConvict[numTrain:])
testEvidNorm = normalize(testEvid)
testConvictdNorm = normalize(testConvict)
# print(testEvidNorm)
# print(testConvictdNorm)
# ######################

# define placeholders  and variables
tfEvid = tf.placeholder(tf.float32, name="Evid")
tfConvict = tf.placeholder(tf.float32, name="Convict")
tfEvidFactor = tf.Variable(np.random.randn(),
tf.float32, name="EvidFactor")
tfConvictOffset = tf.Variable(np.random.randn(),
tf.float32, name="ConvictOffset")

# define the operation for predicting the conviction
based on evidence by adding both values
# define a loss function (mean squared error)
# tfPredict = tf.add(tf.multiply(np.random.randn(),
tfEvid), np.random.randn()) - fehlerhafte
```

Anweisung!

```
tfPredict = tfEvidFactor * tfEvid + tfConvictOffset
tfCost = tf.reduce_sum(tf.pow(tfPredict - tfConvict,
2)) / (2 * numTrain)
# tfCost = tf.reduce_mean(tf.pow(tfPredict -
tfConvict, 2)) / 2

# set a learning rate and a gradient descent optimizer

learningRate = 0.1
gradDesc =
tf.train.GradientDescentOptimizer(learningRate).minimiz▶

# set up iteration parameters
# displayEvery = 2 or 5
displayEvery = 40
# numTrainingSteps = 50 or 100
numTrainingSteps = 400

# Calculate the number of lines to animation
# define variables for updating during animation
numPlotsAnim = math.floor(numTrainingSteps /
displayEvery)
evidFactorAnim = np.zeros(numPlotsAnim)
convictOffsetAnim = np.zeros(numPlotsAnim)
plotIndex = 0

# initialize variables
init = tf.global_variables_initializer()
```

```
with tf.Session() as sess:
    sess.run(init)
    # print(sess.run(tfPredict, feed_dict={tfEvid:
trainEvidNorm, tfConvict: trainConvictdNorm}))
    # print(sess.run(tfCost, feed_dict={tfEvid:
trainEvidNorm, tfConvict: trainConvictdNorm}))
    # print(sess.run(gradDesc, feed_dict={tfEvid:
trainEvidNorm, tfConvict: trainConvictdNorm}))
    # iterate through the training data
    for i in range(numTrainingSteps):
        # ======== Start training by running the
session and feeding the gradDesc
        # for (x, y) in zip(trainEvidNorm,
trainConvictdNorm):
            # sess.run(gradDesc, feed_dict={tfEvid:
x, tfConvict: y})
        sess.run(gradDesc, feed_dict={tfEvid:
trainEvidNorm, tfConvict: trainConvictdNorm})
        # Print status of learning
        if (i + 1) % displayEvery == 0:
            cost = sess.run(
                tfCost, feed_dict={tfEvid:
trainEvidNorm, tfConvict: trainConvictdNorm}
            )
            print(
                "iteration #:",
                "%04d" % (i + 1),
                "cost=",
```

```
                    "{:.9f}".format(cost),
                    "evidFactor=",
                    sess.run(tfEvidFactor),
                    "convictOffset=",
                    sess.run(tfConvictOffset),
                )
                # store the result of each step in the
animation variables
                evidFactorAnim[plotIndex] =
sess.run(tfEvidFactor)
                convictOffsetAnim[plotIndex] =
sess.run(tfConvictOffset)
                plotIndex += 1
    # log the optimized result
    print("Optimized!")
    trainingCost = sess.run(
        tfCost, feed_dict={tfEvid: trainEvidNorm,
tfConvict: trainConvictdNorm}
    )
    print(
        "Trained cost=",
        trainingCost,
        "evidFactor=",
        sess.run(tfEvidFactor),
        "convictOffset=",
        sess.run(tfConvictOffset),
        "\n",
    )
    # ======== Visualize the result and the training
```

```
process
        # denormalize variables to be plottable again
        trainEvidMean = trainEvid.mean()
        trainEvidStd = trainEvid.std()
        trainConvictMean = trainConvict.mean()
        trainConvictStd = trainConvict.std()
        # xNorm = trainEvidNorm * trainEvidStd +
trainEvidMean
        xNorm = trainEvid
        yNorm = (
            sess.run(tfEvidFactor) * trainEvidNorm +
sess.run(tfConvictOffset)
        ) * trainConvictStd + trainConvictMean
        # Plot the result graph
        plt.figure()
        plt.xlabel("Number of Evidence")
        plt.ylabel("Number of Convictions")
        plt.plot(trainEvid, trainConvict, "go",
label="Training data")
        plt.plot(testEvid, testConvict, "mo",
label="Testing data")
        plt.plot(xNorm, yNorm, label="Learned
Regression")
        plt.legend(loc="upper left")
        plt.show()
        # Plot an animated graph that shows the process
of optimization
        fig, ax = plt.subplots()
        line, = ax.plot(numEvid, numConvict)
```

```
    plt.rcParams["figure.figsize"] = (
        10,
        8,
    )  # adding fixed size parameters to keep
animation in scale
    plt.title("Gradient Descent Fitting Regression
Line")
    plt.xlabel("Number of Evidence")
    plt.ylabel("Number of Convictions")
    plt.plot(trainEvid, trainConvict, "go",
label="Training data")
    plt.plot(testEvid, testConvict, "mo",
label="Testing data")
    # call the animation
    ani = animation.FuncAnimation(
        fig,
        animate,
        frames=np.arange(0, plotIndex),
        init_func=initAnim,
        interval=200,
        blit=True,
    )
    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 math
>>> import matplotlib
>>> matplotlib.use('TkAgg')
>>> import matplotlib.pyplot as plt
>>> import matplotlib.animation as animation
>>>
>>> # Generate evidence numbers between 10 and 50
... # Generate a number of convictions from the
evidence with a random noise added
... np.random.seed(42)
>>> sampleSize = 200
>>> numEvid = np.random.randint(low=10, high=50,
size=sampleSize)
>>> numConvict = numEvid * 10 +
np.random.randint(low=200, high=400,
size=sampleSize)
>>> print(numEvid)
[48 38 24 17 30 48 28 32 20 20 33 45 49 33 12
31 11 33 39 47 11 30 42 21
 31 34 36 37 25 24 12 46 16 30 18 48 27 13 34
23 18 35 11 29 37 16 17 44
 23 26 45 49 13 11 15 13 38 27 35 43 19 45 23
40 24 17 23 32 49 30 25 27
 33 35 34 38 24 10 34 16 18 33 10 17 33 20 26
17 44 44 42 14 48 37 16 18
 17 21 43 42 32 33 46 44 49 31 36 44 10 44 46
```

```
23 12 10 14 35 23 48 36 18
 24 24 35 22 41 48 41 13 39 46 32 48 24 38 45
22 41 16 31 37 11 15 37 37
 29 39 20 37 34 48 42 10 36 22 12 48 15 17 36
18 46 42 33 24 41 41 33 21
 48 11 12 46 26 11 11 37 32 46 41 42 10 28 11
35 41 15 41 13 20 26 47 33
 14 43 15 31 20 25 42 18]
>>> print(numConvict)
[803 766 509 462 502 827 666 683 546 489 724
796 837 625 518 561 470 697
 717 708 391 603 748 420 694 717 710 728 491
538 326 803 449 611 439 792
 471 458 587 569 576 586 469 498 668 506 417
770 577 611 703 809 490 461
 465 404 692 669 713 795 493 733 541 698 592
462 575 647 799 581 643 523
 692 738 708 740 507 332 681 380 427 677 427
505 664 594 604 497 672 815
 806 454 798 591 517 417 478 460 811 627 546
556 680 669 786 537 670 831
 496 700 707 576 323 334 531 598 446 851 717
425 556 445 648 543 646 703
 702 375 770 754 618 867 555 770 809 580 676
487 527 594 363 407 636 673
 663 613 513 601 714 765 770 493 686 574 449
696 453 530 696 422 835 658
 699 465 708 659 682 561 692 369 454 716 495
482 329 634 527 803 751 734
```

```
 442 571 407 615 641 540 695 380 552 645 732
719 464 779 407 567 485 498
 799 549]
>>>
>>> # Plot the data to get a feeling
... plt.title("Number of convictions based on
evidence")
Text(0.5, 1.0, 'Number of convictions based on
evidence')
>>> plt.plot(numEvid, numConvict, "bx")
[<matplotlib.lines.Line2D object at 0x7f1530c054e0>]
>>> plt.xlabel("Number of Evidence")
Text(0.5, 0, 'Number of Evidence')
>>> plt.ylabel("Number of Convictions")
Text(0, 0.5, 'Number of Convictions')
>>> # Use the keyword 'block' to override the blocking
behavior
... plt.show(block=False)
>>> # ######################
...
>>> # create a function for normalizing values
... def normalize(array):
...     return (array - array.mean()) /
array.std()
...
>>> # define an animation function that changes the
ydata
... def animate(i):
...     line.set_xdata(xNorm)
```

```
...         line.set_ydata(
...             (evidFactorAnim[i] * trainEvidNorm +
convictOffsetAnim[i]) * trainConvictStd
...             + trainConvictMean
...             )
...         return (line,)
...
>>> # Initialize the animation with zeros for y
...
>>> def initAnim():
...
line.set_ydata(np.zeros(shape=numConvict.shape[0]))
...         return (line,)
...
>>> # use 70% of the data for training (the remaining
30% shall be used for testing)
...
>>> numTrain = math.floor(sampleSize * 0.7)
>>>
>>> # convert list to an array and normalize arrays
... # 70% of the data for training
... trainEvid = np.asanyarray(numEvid[:numTrain])
>>> trainConvict =
np.asanyarray(numConvict[:numTrain])
>>> trainEvidNorm = normalize(trainEvid)
>>> trainConvictdNorm = normalize(trainConvict)
>>> print(trainEvidNorm)
[ 1.54445682  0.69318141 -0.49860417
-1.09449696  0.01216108  1.54445682
```

```
 -0.15809401  0.18241616 -0.83911434
-0.83911434  0.2675437   1.2890742
  1.62958437  0.2675437  -1.52013467
0.09728862 -1.60526221  0.2675437
  0.77830895  1.45932928 -1.60526221
0.01216108  1.03369157 -0.7539868
  0.09728862  0.35267124  0.52292633
0.60805387 -0.41347663 -0.49860417
 -1.52013467  1.37420174 -1.1796245
0.01216108 -1.00936942  1.54445682
 -0.24322155 -1.43500713  0.35267124
-0.58373171 -1.00936942  0.43779878
 -1.60526221 -0.07296646  0.60805387
-1.1796245  -1.09449696  1.20394666
 -0.58373171 -0.32834909  1.2890742
1.62958437 -1.43500713 -1.60526221
 -1.26475204 -1.43500713  0.69318141
-0.24322155  0.43779878  1.11881912
 -0.92424188  1.2890742  -0.58373171
0.86343649 -0.49860417 -1.09449696
 -0.58373171  0.18241616  1.62958437
0.01216108 -0.41347663 -0.24322155
  0.2675437   0.43779878  0.35267124
0.69318141 -0.49860417 -1.69038975
  0.35267124 -1.1796245  -1.00936942
0.2675437  -1.69038975 -1.09449696
  0.2675437  -0.83911434 -0.32834909
-1.09449696  1.20394666  1.20394666
  1.03369157 -1.34987959  1.54445682
```

```
0.60805387 -1.1796245  -1.00936942
 -1.09449696 -0.7539868   1.11881912
1.03369157  0.18241616  0.2675437
  1.37420174  1.20394666  1.62958437
0.09728862  0.52292633  1.20394666
 -1.69038975  1.20394666  1.37420174
-0.58373171 -1.52013467 -1.69038975
 -1.34987959  0.43779878 -0.58373171
1.54445682  0.52292633 -1.00936942
 -0.49860417 -0.49860417  0.43779878
-0.66885925  0.94856403  1.54445682
  0.94856403 -1.43500713  0.77830895
1.37420174  0.18241616  1.54445682
 -0.49860417  0.69318141  1.2890742
-0.66885925  0.94856403 -1.1796245
  0.09728862  0.60805387]
>>> print(trainConvictdNorm)
[ 1.47639022  1.19909866 -0.72695356
-1.07918879 -0.77941413  1.65625502
  0.449662    0.57706623 -0.449662
-0.87684089  0.88433526  1.42392966
  1.73119869  0.14239297 -0.65950426
-0.3372465  -1.01923386  0.68198736
  0.83187469  0.7644254  -1.61128882
-0.0224831   1.06420006 -1.39395219
  0.65950426  0.83187469  0.77941413
0.91431273 -0.86185216 -0.50961693
 -2.09842265  1.47639022 -1.17661556
0.03747183 -1.25155923  1.39395219
```

-1.01173949 -1.10916626 -0.14239297
-0.27729156 -0.224831   -0.14988733
-1.02672823 -0.8093916   0.46465073
-0.74943666 -1.41643529 1.22907613
-0.21733663 0.03747183 0.72695356
1.52135642 -0.86934653 -1.08668316
-1.05670569 -1.51386206 0.64451553
0.4721451   0.80189723 1.41643529
-0.84686343 0.95178456 -0.48713383
0.68948173 -0.10492113 -1.07918879
-0.23232537 0.30726903 1.44641276
-0.18735917 0.27729156 -0.62203243
 0.64451553 0.98925639 0.7644254
1.00424513 -0.7419423 -2.05345645
 0.5620775 -1.69372686 -1.34149163
0.53210003 -1.34149163 -0.75693103
 0.43467326 -0.0899324 -0.01498873
-0.81688596 0.4946282   1.56632262
 1.49887332 -1.13914373 1.43891839
-0.1124155 -0.66699863 -1.41643529
-0.95927893 -1.09417753 1.53634516
0.1573817 -0.449662   -0.37471833
 0.55458313 0.4721451   1.34898599
-0.5171113   0.47963946 1.68623249
-0.82438033 0.70447046 0.75693103
-0.224831   -2.12090575 -2.03846772
-0.5620775 -0.05995493 -1.19909866
1.83611982 0.83187469 -1.35648036
-0.37471833 -1.20659303 0.3147634

```
-0.4721451    0.29977466   0.72695356
  0.7194592  -1.73119869   1.22907613
1.10916626   0.0899324    1.95602969
 -0.3822127    1.22907613   1.52135642
-0.19485353   0.52460566 -0.89182963
 -0.59205496 -0.0899324 ]
>>>
>>> # the remaining 30% shall be used for testing
... testEvid = np.asanyarray(numEvid[numTrain:])
>>> testConvict =
np.asanyarray(numConvict[numTrain:])
>>> testEvidNorm = normalize(testEvid)
>>> testConvictdNorm = normalize(testConvict)
>>> print(testEvidNorm)
[-1.42925466 -1.11016525   0.64482652
0.64482652   0.0066477    0.80437123
 -0.71130348   0.64482652   0.40550946
1.52232241   1.04368829 -1.50902701
  0.56505417 -0.55175878 -1.34948231
1.52232241 -1.11016525 -0.95062054
  0.56505417 -0.87084819   1.3627777
1.04368829   0.32573711 -0.39221407
  0.96391593   0.96391593   0.32573711
-0.63153113   1.52232241 -1.42925466
 -1.34948231   1.3627777   -0.23266936
-1.42925466 -1.42925466   0.64482652
  0.24596476   1.3627777    0.96391593
1.04368829 -1.50902701 -0.07312466
 -1.42925466   0.48528182   0.96391593
```

```
-1.11016525  0.96391593 -1.26970995
 -0.71130348 -0.23266936  1.44255005
0.32573711 -1.1899376   1.12346064
 -1.11016525  0.1661924  -0.71130348
-0.31244172  1.04368829 -0.87084819]
>>> print(testConvictdNorm)
[-1.76133668 -1.42135504  0.34809483
0.63398848  0.55671993  0.17037716
 -0.60230837  0.0776549   0.95078955
1.34485917  1.38349345 -0.75684548
  0.7344376  -0.1309702  -1.09682712
0.81170615 -1.0659197  -0.47095183
  0.81170615 -1.30545221  1.88573905
0.51808565  0.83488672 -0.97319743
  0.90442842  0.52581251  0.70353018
-0.23141932  0.78079873 -1.71497554
 -1.05819284  0.96624326 -0.74139177
-0.84184089 -2.02404976  0.33264112
 -0.4941324   1.63847968  1.2366832
1.10532666 -1.1509151  -0.15415076
 -1.42135504  0.18583087  0.38672911
-0.39368328  0.8039793  -1.62998014
 -0.30096102  0.41763653  1.08987295
0.98942383 -0.98092429  1.45303515
 -1.42135504 -0.18505819 -0.81866032
-0.7182112   1.60757226 -0.32414158]
>>> # ######################
...
>>> # define placeholders  and variables
```

```
... tfEvid = tf.placeholder(tf.float32, name="Evid")
>>> tfConvict = tf.placeholder(tf.float32,
name="Convict")
>>> tfEvidFactor = tf.Variable(np.random.randn(),
tf.float32, name="EvidFactor")
WARNING:tensorflow:From
/home/parallels/.local/lib/python3.6/site-packag
colocate_with (from
tensorflow.python.framework.ops) is deprecated
and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
>>> tfConvictOffset = tf.Variable(np.random.randn(),
tf.float32, name="ConvictOffset")
>>>
>>> # define the operation for predicting the conviction
based on evidence by adding both values
... # define a loss function (mean squared error)
... # tfPredict =
tf.add(tf.multiply(np.random.randn(), tfEvid),
np.random.randn())
... tfPredict = tfEvidFactor * tfEvid + tfConvictOffset
>>> # tfCost = tf.reduce_sum(tf.pow(tfPredict -
tfConvict, 2)) / (2 * numTrain)
... tfCost = tf.reduce_mean(tf.pow(tfPredict -
tfConvict, 2)) / 2
>>>
>>> # set a learning rate and a gradient descent
optimizer
```

```
...
>>> learningRate = 0.1
>>> gradDesc =
tf.train.GradientDescentOptimizer(learningRate).minimiz▶
WARNING:tensorflow:From
/home/parallels/.local/lib/python3.6/site-packag▶
to_int32 (from tensorflow.python.ops.math_ops)
is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
>>>
>>> # set up iteration parameters
... # displayEvery = 2 or 5
... displayEvery = 50
>>> # numTrainingSteps = 50 or 100
... numTrainingSteps = 1000
>>>
>>> # Calculate the number of lines to animation
... # define variables for updating during animation
... numPlotsAnim = math.floor(numTrainingSteps /
displayEvery)
>>> evidFactorAnim = np.zeros(numPlotsAnim)
>>> convictOffsetAnim = np.zeros(numPlotsAnim)
>>> plotIndex = 0
>>>
>>> # initialize variables
... init = tf.global_variables_initializer()
>>> with tf.Session() as sess:
```

```
...         sess.run(init)
...         # print(sess.run(tfPredict, feed_dict={tfEvid:
trainEvidNorm, tfConvict: trainConvictdNorm}))
...         # print(sess.run(tfCost, feed_dict={tfEvid:
trainEvidNorm, tfConvict: trainConvictdNorm}))
...         # print(sess.run(gradDesc, feed_dict={tfEvid:
trainEvidNorm, tfConvict: trainConvictdNorm}))
...         # iterate through the training data
...         for i in range(numTrainingSteps):
...             # ======== Start training by running
the session and feeding the gradDesc
...             # for (x, y) in zip(trainEvidNorm,
trainConvictdNorm):
...                 # sess.run(gradDesc,
feed_dict={tfEvid: x, tfConvict: y})
...             sess.run(gradDesc, feed_dict={tfEvid:
trainEvidNorm, tfConvict: trainConvictdNorm})
...             # Print status of learning
...             if (i + 1) % displayEvery == 0:
...                 cost = sess.run(
...                     tfCost, feed_dict={tfEvid:
trainEvidNorm, tfConvict: trainConvictdNorm}
...                 )
...                 print(
...                     "iteration #:",
...                     "%04d" % (i + 1),
...                     "cost=",
...                     "{:.9f}".format(cost),
...                     "evidFactor=",
```

```
...                     sess.run(tfEvidFactor),
...                     "convictOffset=",
...                     sess.run(tfConvictOffset),
...                 )
...                 # store the result of each step in
the animation variables
...                 evidFactorAnim[plotIndex] =
sess.run(tfEvidFactor)
...                 convictOffsetAnim[plotIndex] =
sess.run(tfConvictOffset)
...                 plotIndex += 1
...         # log the optimized result
...         print("Optimized!")
...         trainingCost = sess.run(
...             tfCost, feed_dict={tfEvid: trainEvidNorm,
tfConvict: trainConvictdNorm}
...         )
...         print(
...             "Trained cost=",
...             trainingCost,
...             "evidFactor=",
...             sess.run(tfEvidFactor),
...             "convictOffset=",
...             sess.run(tfConvictOffset),
...             "\n",
...         )
...         # ======== Visualize the result and the
training process
...         # denormalize variables to be plottable again
```

```
...         trainEvidMean = trainEvid.mean()
...         trainEvidStd = trainEvid.std()
...         trainConvictMean = trainConvict.mean()
...         trainConvictStd = trainConvict.std()
...         # xNorm = trainEvidNorm * trainEvidStd +
trainEvidMean
...         xNorm = trainEvid
...         yNorm = (
...             sess.run(tfEvidFactor) * trainEvidNorm +
sess.run(tfConvictOffset)
...         ) * trainConvictStd + trainConvictMean
...         # Plot the result graph
...         plt.figure()
...         plt.xlabel("Number of Evidence")
...         plt.ylabel("Number of Convictions")
...         plt.plot(trainEvid, trainConvict, "go",
label="Training data")
...         plt.plot(testEvid, testConvict, "mo",
label="Testing data")
...         plt.plot(xNorm, yNorm, label="Learned
Regression")
...         plt.legend(loc="upper left")
...         plt.show()
...         # Plot an animated graph that shows the
process of optimization
...         fig, ax = plt.subplots()
...         line, = ax.plot(numEvid, numConvict)
...         plt.rcParams["figure.figsize"] = (
...             10,
```

```
...             8,
...         )  # adding fixed size parameters to keep
animation in scale
...         plt.title("Gradient Descent Fitting Regression
Line")
...         plt.xlabel("Number of Evidence")
...         plt.ylabel("Number of Convictions")
...         plt.plot(trainEvid, trainConvict, "go",
label="Training data")
...         plt.plot(testEvid, testConvict, "mo",
label="Testing data")
...         # call the animation
...         ani = animation.FuncAnimation(
...             fig,
...             animate,
...             frames=np.arange(0, plotIndex),
...             init_func=initAnim,
...             interval=200,
...             blit=True,
...         )
...         plt.show()
...
...
2019-03-23 18:00:26.906426: I
tensorflow/core/platform/profile_utils/cpu_utils.c
CPU Frequency: 2494315000 Hz
2019-03-23 18:00:26.910870: I
tensorflow/compiler/xla/service/service.cc:150]
XLA service 0x291ae10 executing computations on
```

platform Host. Devices:

2019-03-23 18:00:26.910953: I

tensorflow/compiler/xla/service/service.cc:158]

StreamExecutor device (0): <undefined>,

<undefined>

iteration #: 0050 cost= 0.093942054 evidFactor=

0.89414304 convictOffset= 0.005612206

iteration #: 0100 cost= 0.093901277 evidFactor=

0.90118355 convictOffset= 2.891775e-05

iteration #: 0150 cost= 0.093901269 evidFactor=

0.9012197 convictOffset= 1.4491113e-07

iteration #: 0200 cost= 0.093901269 evidFactor=

0.9012197 convictOffset= 9.869344e-09

iteration #: 0250 cost= 0.093901269 evidFactor=

0.9012197 convictOffset= 3.536352e-09

iteration #: 0300 cost= 0.093901269 evidFactor=

0.9012197 convictOffset= -1.1202597e-09

iteration #: 0350 cost= 0.093901269 evidFactor=

0.9012197 convictOffset= -5.776869e-09

iteration #: 0400 cost= 0.093901269 evidFactor=

0.9012197 convictOffset= -1.0433492e-08

iteration #: 0450 cost= 0.093901269 evidFactor=

0.9012197 convictOffset= -1.4438196e-08

iteration #: 0500 cost= 0.093901269 evidFactor=

0.9012197 convictOffset= -1.45313415e-08

iteration #: 0550 cost= 0.093901269 evidFactor=

0.9012197 convictOffset= -1.46244865e-08

iteration #: 0600 cost= 0.093901269 evidFactor=

0.9012197 convictOffset= -1.4717632e-08

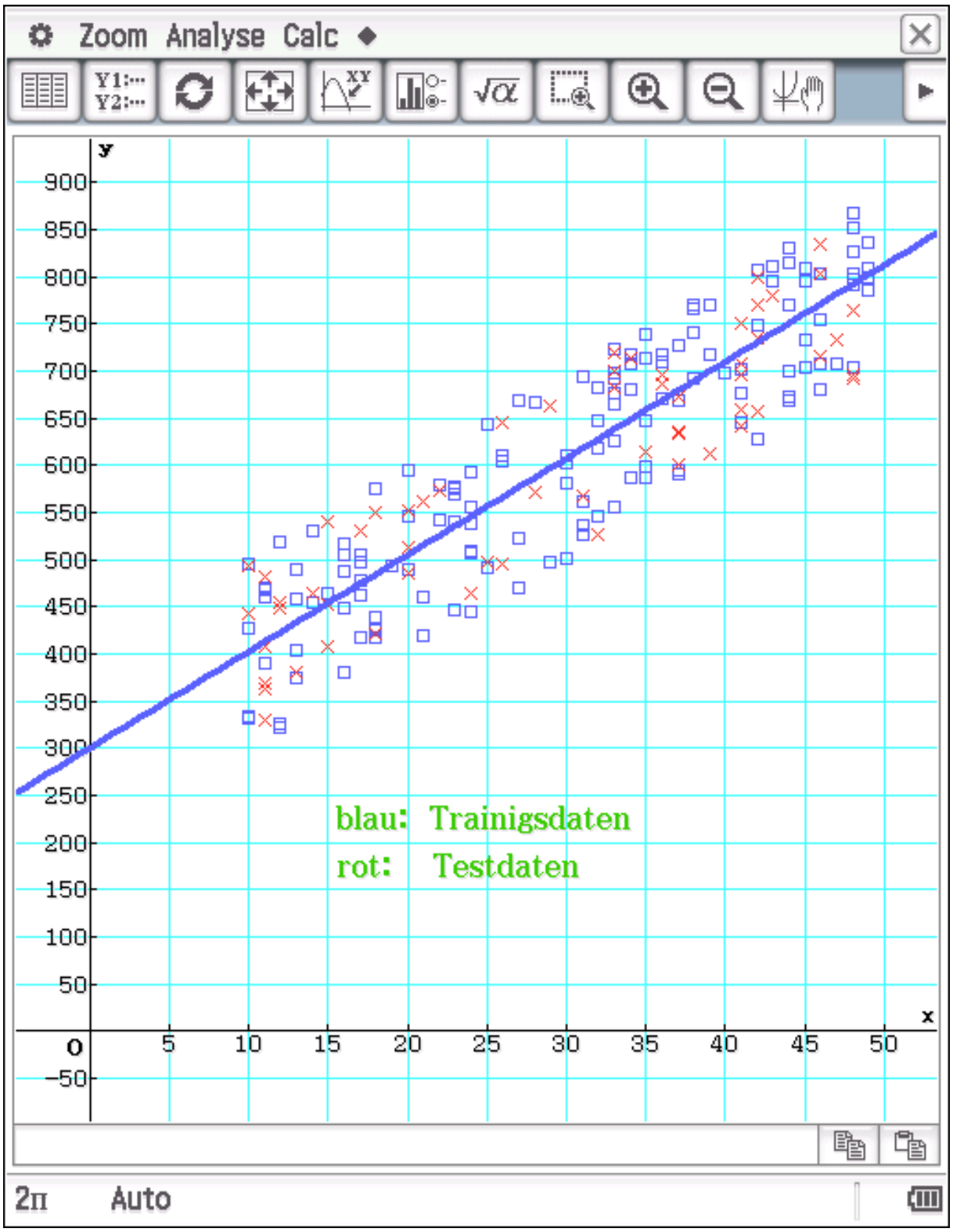
```
iteration #: 0650 cost= 0.093901269 evidFactor=
0.9012197 convictOffset= -1.4810777e-08
iteration #: 0700 cost= 0.093901262 evidFactor=
0.9012197 convictOffset= -1.4903922e-08
iteration #: 0750 cost= 0.093901269 evidFactor=
0.9012197 convictOffset= -1.434514e-08
iteration #: 0800 cost= 0.093901269 evidFactor=
0.9012197 convictOffset= -1.4438285e-08
iteration #: 0850 cost= 0.093901269 evidFactor=
0.9012197 convictOffset= -1.453143e-08
iteration #: 0900 cost= 0.093901269 evidFactor=
0.9012197 convictOffset= -1.4624575e-08
iteration #: 0950 cost= 0.093901269 evidFactor=
0.9012197 convictOffset= -1.471772e-08
iteration #: 1000 cost= 0.093901269 evidFactor=
0.9012197 convictOffset= -1.4810865e-08
Optimized!
Trained cost= 0.09390127 evidFactor= 0.9012197
convictOffset= -1.4810865e-08

<Figure size 640x480 with 0 Axes>
Text(0.5, 0, 'Number of Evidence')
Text(0, 0.5, 'Number of Convictions')
[<matplotlib.lines.Line2D object at 0x7f152c2e8f98>]
[<matplotlib.lines.Line2D object at 0x7f152c2f1198>]
[<matplotlib.lines.Line2D object at 0x7f152c2f14a8>]
<matplotlib.legend.Legend object at 0x7f152c2f1a20>
```

Download für dieses Dokument:

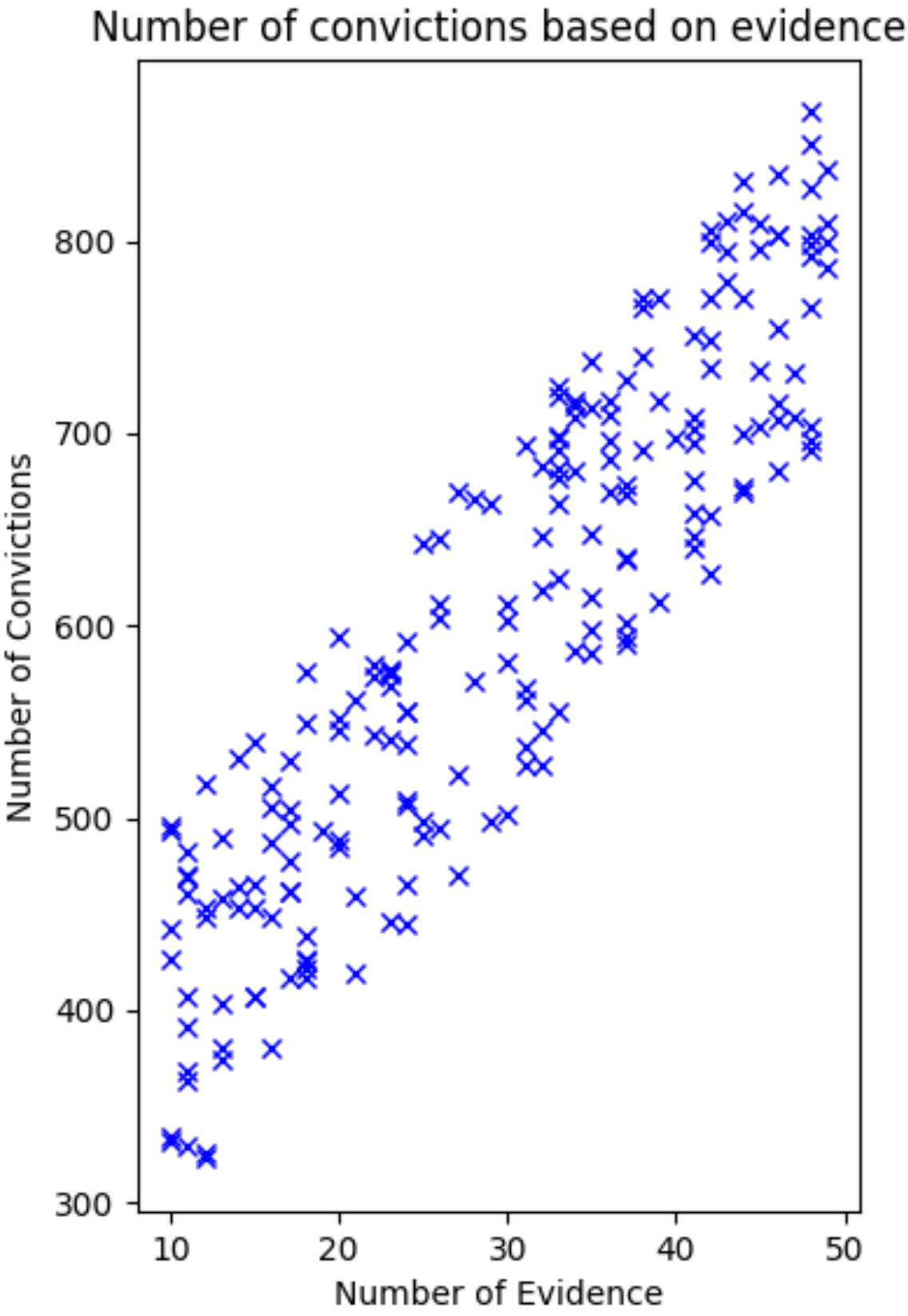
www.informatik.htw-dresden.de/

~paditz/Tensorflow-Ue18.pdf

Lineare Regression mit Trainigs- und Testdaten:



Tensorflow: alle Daten



Tensorflow: alle Daten mit linearer Regression