Prof. Dr. L. Paditz, 14.02.2019

HTW Dresden,

paditz@htw-dresden.de

Mathematik-Grundlagen als eActivty mit ClassPad

lineare Regression in TensorFlow

Gradientenverfahren - steilster Abstieg zum Min!

==========

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

$\text{sum}\left(\frac{(W*X\_liste+b-Y\_liste)^2}{\dim(X\_liste)}\right)$

$0.02380952381\cdot(39.7\cdot W+b-147)^2+0.02380952381\cdot($▶

simplify(ans)

$2.380952381\text{E}{-4}\cdot(1059859\cdot W^2+4200\cdot b^2+106620\cdot W\cdot b$▶

expand(ans)

$252.347381\cdot W^2+b^2+25.38571429\cdot W\cdot b-1094.328571$▶

Standardvariablen: W=x, b=y, loss=z

ans|{W=x,b=y}

$252.347381\cdot x^2+y^2+25.38571429\cdot x\cdot y-1094.328571\cdot$▶

==========

LinearReg X_liste,Y_liste,1,y1

done

DispStat

done

y1(x)

1.313456005·x+16.99515723

==========

Define loss(W,b)=252.347381·W^2+b^2+25.38571429·▶

done

loss(1.313456005,16.99515723)

360.8201575

loss(W,b)

252.347381·W^2+b^2+25.38571429·W·b−1094.328571▶

Definition von z für 3D-Grafik:

Define z1(x,y)=252.347381·x^2+y^2+25.38571429·x·y▶

done

Define z2(x,y)=360.8201575

done

3D-Grafik Z1:… Z2:…

Weiter mit einem kleineren Datensatz:

dim(train_X)=dim(train_Y)

17=17

stop

Quelle:

https://github.com/aymericdamien/TensorFlow-Examples/blob/master/notebooks/2_BasicModels/linear_regression.ipynb

```
python3

import tensorflow as tf
import numpy
import matplotlib.pyplot as plt
rng = numpy.random

# Parameters
learning_rate = 0.1  # 0.01
training_epochs = 10 #1000
display_step = 1  # 50

# Training Data
train_X =
numpy.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.7
train_Y =
numpy.asarray([1.7,2.76,2.09,3.19,1.694,1.573,
n_samples = train_X.shape[0]

# tf Graph Input
X = tf.placeholder("float")
Y = tf.placeholder("float")

# Set model weights
W = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")

# Construct a linear model
pred = tf.add(tf.multiply(X, W), b)
```

```
# Mean squared error
cost = tf.reduce_sum(tf.pow(pred-Y,
2))/(2*n_samples)

# Gradient descent
optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimi▶

# Initialize the variables (i.e. assign their default
value)
init = tf.global_variables_initializer()

# Start training
with tf.Session() as sess:
    sess.run(init)
    print("cost=", cost, "W=", sess.run([W]),
"b=", sess.run([b]), '\n')
    # Fit all training data
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y:
y})
        #Display logs per epoch step
        if (epoch) % display_step == 0:
            c = sess.run(cost, feed_dict={X:
train_X, Y:train_Y})
            print("Epoch:", '%04d' % (epoch),
"cost=", "{:.9f}".format(c), "W=", sess.run([W]),
```

```
"b=", sess.run([b]))
    print("Optimization Finished!")
    training_cost = sess.run([cost], feed_dict={X:
train_X, Y: train_Y})
    print("Training cost=", training_cost, "W=",
sess.run([W]), "b=", sess.run([b]), '\n')
    #Graphic display
    plt.plot(train_X, train_Y, 'ro', label='Original
data')
    plt.plot(train_X, sess.run([W]) * train_X +
sess.run([b]), label='Fitted line')
    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.
>>>
>>> import tensorflow as tf
>>> import numpy
>>> import matplotlib.pyplot as plt
>>> rng = numpy.random
>>>
>>> # Parameters
... learning_rate = 0.01  # 0.1
```

```
>>> training_epochs = 1000 #10
>>> display_step = 50  # 1
>>>
>>> # Training Data
... train_X =
numpy.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.7
...
7.042,10.791,5.313,7.997,5.654,9.27,3.1])
>>> train_Y =
numpy.asarray([1.7,2.76,2.09,3.19,1.694,1.573,
...
2.827,3.465,1.65,2.904,2.42,2.94,1.3])
>>> n_samples = train_X.shape[0]
>>>
>>> # tf Graph Input
... X = tf.placeholder("float")
>>> Y = tf.placeholder("float")
>>>
>>> # Set model weights
... W = tf.Variable(rng.randn(), name="weight")
>>> b = tf.Variable(rng.randn(), name="bias")
>>>
>>> # Construct a linear model
... pred = tf.add(tf.multiply(X, W), b)
>>>
>>> # Mean squared error
... cost = tf.reduce_sum(tf.pow(pred-Y,
2))/(2*n_samples)
>>> # Gradient descent
```

```
... optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimi▶
>>>
>>> # Initialize the variables (i.e. assign their default
value)
... init = tf.global_variables_initializer()
>>>
>>> # Start training
... with tf.Session() as sess:
...     sess.run(init)
...     # Fit all training data
...     for epoch in range(training_epochs):
...         for (x, y) in zip(train_X, train_Y):
...             sess.run(optimizer, feed_dict={X:
x, Y: y})
...         #Display logs per epoch step
...         if (epoch+1) % display_step == 0:
...             c = sess.run(cost, feed_dict={X:
train_X, Y:train_Y})
...             print("Epoch:", '%04d' %
(epoch+1), "cost=", "{:.9f}".format(c), "W=",
sess.run([W]), "b=", sess.run([b]))
...     print("Optimization Finished!")
...     training_cost = sess.run([cost],
feed_dict={X: train_X, Y: train_Y})
...     print("Training cost=", training_cost, "W=",
sess.run([W]), "b=", sess.run([b]), '\n')
...     #Graphic display
...     plt.plot(train_X, train_Y, 'ro', label='Original
```

```
data')
...         plt.plot(train_X, sess.run([W]) * train_X +
sess.run([b]), label='Fitted line')
...         plt.legend()
...         plt.show()
...
Epoch: 0050 cost= 0.083932936 W= [0.29635733]
b= [0.46505147]
Epoch: 0100 cost= 0.083127789 W= [0.29357597]
b= [0.48506063]
Epoch: 0150 cost= 0.082415864 W= [0.29096022]
b= [0.5038777]
Epoch: 0200 cost= 0.081786320 W= [0.28850007]
b= [0.521576]
Epoch: 0250 cost= 0.081229590 W= [0.2861861] b=
[0.53822255]
Epoch: 0300 cost= 0.080737293 W= [0.2840097] b=
[0.55387956]
Epoch: 0350 cost= 0.080302022 W= [0.28196272]
b= [0.56860524]
Epoch: 0400 cost= 0.079917140 W= [0.28003746]
b= [0.5824556]
Epoch: 0450 cost= 0.079576820 W= [0.27822667]
b= [0.59548223]
Epoch: 0500 cost= 0.079275943 W= [0.27652353]
b= [0.6077349]
Epoch: 0550 cost= 0.079009958 W= [0.2749219] b=
[0.61925656]
Epoch: 0600 cost= 0.078774817 W= [0.27341568]
```

```
b= [0.6300922]
Epoch: 0650 cost= 0.078566961 W= [0.271999] b=
[0.64028376]
Epoch: 0700 cost= 0.078383185 W= [0.27066645]
b= [0.6498697]
Epoch: 0750 cost= 0.078220740 W= [0.26941317]
b= [0.6588853]
Epoch: 0800 cost= 0.078077123 W= [0.26823452]
b= [0.66736495]
Epoch: 0850 cost= 0.077950209 W= [0.2671259] b=
[0.6753402]
Epoch: 0900 cost= 0.077838011 W= [0.2660833] b=
[0.6828407]
Epoch: 0950 cost= 0.077738874 W= [0.2651026] b=
[0.6898957]
Epoch: 1000 cost= 0.077651232 W= [0.2641801] b=
[0.69653136]
Optimization Finished!
Training cost= [0.07765123] W= [0.2641801] b=
[0.69653136]

[<matplotlib.lines.Line2D object at 0x7fdfe6cd0080>]
[<matplotlib.lines.Line2D object at 0x7fdfe6cd02e8>]
<matplotlib.legend.Legend object at 0x7fdfe6cd0ac8>
```

erneuter Rechnerlauf:

```
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
>>> import matplotlib.pyplot as plt
>>> rng = numpy.random
>>>
>>> # Parameters
... learning_rate = 0.1  # 0.01
>>> training_epochs = 10 #1000
>>> display_step = 1  # 50
>>>
>>> # Training Data
... train_X =
numpy.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.7
>>> train_Y =
numpy.asarray([1.7,2.76,2.09,3.19,1.694,1.573,
>>> n_samples = train_X.shape[0]
>>>
>>> # tf Graph Input
... X = tf.placeholder("float")
>>> Y = tf.placeholder("float")
>>>
>>> # Set model weights
... W = tf.Variable(rng.randn(), name="weight")
>>> b = tf.Variable(rng.randn(), name="bias")
```

```
>>>
>>> # Construct a linear model
... pred = tf.add(tf.multiply(X, W), b)
>>>
>>> # Mean squared error
... cost = tf.reduce_sum(tf.pow(pred-Y,
2))/(2*n_samples)
>>>
>>> # Gradient descent
... optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimi▶
>>>
>>> # Initialize the variables (i.e. assign their default
value)
... init = tf.global_variables_initializer()
>>>
>>> # Start training
... with tf.Session() as sess:
...     sess.run(init)
...     print("cost=", cost, "W=", sess.run([W]),
"b=", sess.run([b]), '\n')
...     # Fit all training data
...     for epoch in range(training_epochs):
...         for (x, y) in zip(train_X, train_Y):
...             sess.run(optimizer, feed_dict={X:
x, Y: y})
...         #Display logs per epoch step
...         if (epoch) % display_step == 0:
...             c = sess.run(cost, feed_dict={X:
```

```
train_X, Y:train_Y})
...                 print("Epoch:", '%04d' % (epoch),
"cost=", "{:.9f}".format(c), "W=", sess.run([W]),
"b=", sess.run([b]))
...         print("Optimization Finished!")
...         training_cost = sess.run([cost],
feed_dict={X: train_X, Y: train_Y})
...         print("Training cost=", training_cost, "W=",
sess.run([W]), "b=", sess.run([b]), '\n')
...         #Graphic display
...         plt.plot(train_X, train_Y, 'ro', label='Original
data')
...         plt.plot(train_X, sess.run([W]) * train_X +
sess.run([b]), label='Fitted line')
...         plt.legend()
...         plt.show()
...
cost= Tensor("truediv:0", shape=(),
dtype=float32) W= [0.38903466] b=
[-0.40474084]

Epoch: 0000 cost= 0.177051708 W=
[0.39435494] b= [-0.3878391]
Epoch: 0001 cost= 0.174632162 W=
[0.39232808] b= [-0.37236074]
Epoch: 0002 cost= 0.172295541 W=
[0.39030755] b= [-0.35708445]
Epoch: 0003 cost= 0.170018047 W=
[0.38831297] b= [-0.34200448]
```

Epoch: 0004 cost= 0.167798087 W=
[0.38634402] b= [-0.32711828]
Epoch: 0005 cost= 0.165634304 W=
[0.38440034] b= [-0.31242332]
Epoch: 0006 cost= 0.163525149 W=
[0.38248163] b= [-0.29791716]
Epoch: 0007 cost= 0.161469370 W=
[0.38058755] b= [-0.2835974]
Epoch: 0008 cost= 0.159465447 W=
[0.37871787] b= [-0.2694616]
Epoch: 0009 cost= 0.157512143 W=
[0.37687218] b= [-0.25550747]
Optimization Finished!
Training cost= [0.15751214] W= [0.37687218] b=
[-0.25550747]

[<matplotlib.lines.Line2D object at 0x7fec2ccce048>]
[<matplotlib.lines.Line2D object at 0x7fec2ccce2b0>]
<matplotlib.legend.Legend object at 0x7fec2cccea90>

Interpretation der Daten:
W fällt (0.39435494↘0.37687218) und
b wächst (-0.40474084 ↗ -0.25550747)
Der Gradient ist ein Vektor

$$\text{grad}(\text{cost}(W,b)) = \begin{bmatrix} \frac{d}{dW}(\text{cost}(W,b)) \\ \frac{d}{db}(\text{cost}(W,b)) \end{bmatrix} \text{ im}$$

Definitionsbereich der Zielfunktion (loss oder cost) und zeigt die **Richtung des stärksten Anstiegs**

(Wachstums) an. Daher **neg. Gradienten** betrachten.

DelVar W,b

done

Daten:

train_X:={3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.▶

{3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.5▶

train_Y:={1.7,2.76,2.09,3.19,1.694,1.573,3.366▶

{1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,▶

$\text{sum}\left(\frac{(W*\text{train\_X}+b-\text{train\_Y})^2}{2*\dim(\text{train\_X})}\right)$

$0.02941176471\cdot(10.791\cdot W+b-3.465)^2+0.02941176$▶

simplify(ans)

$2.941176471\text{E}{-8}\cdot(752797217\cdot W^2+17000000\cdot b^2+2117$▶

expand(ans)

$22.14109462\cdot W^2+0.5\cdot b^2+6.229\cdot W\cdot b-16.11867906\cdot V$▶

Define cost(W,b)=$22.14109462\cdot W^2+0.5\cdot b^2+6.229\cdot W$▶

done

cost(W,b)|{W=0.38903466,b=−0.40474084}➔cost0

0.1891133537

Epoch: 0000 cost= 0.177051708 W=

[0.39435494] b= [−0.3878391]

cost(W,b)|{W=0.39435494,b=−0.3878391}➔cost1

0.177051727

Abstiegsverfahren in Richtung des steilsten Abstiegs (d.h. in negativer Gradientenrichtung

$$\begin{bmatrix} -\frac{d}{dW}(\text{cost}(W,b)) \\ -\frac{d}{db}(\text{cost}(W,b)) \end{bmatrix}$$ **) innerhalb des**

Definitionsbereiches der Zielfunktion:

$$\begin{bmatrix} W_{n+1} \\ b_{n+1} \end{bmatrix} = \begin{bmatrix} W_n \\ b_n \end{bmatrix} + 0.1 * \begin{bmatrix} -\frac{d}{dW}(\text{cost}(W,b)) \mid W=W_n, b=b_n \\ -\frac{d}{db}(\text{cost}(W,b)) \mid W=W_n, b=b_n \end{bmatrix},$$

d.h. für n=0:

per Zufall gewählter Startvektor $\begin{bmatrix} W_0 \\ b_0 \end{bmatrix} = \begin{bmatrix} 0.38903466 \\ -0.40474084 \end{bmatrix}$:

$$\begin{bmatrix} 0.39435494 \\ -0.3878391 \end{bmatrix} = \begin{bmatrix} 0.38903466 \\ -0.40474084 \end{bmatrix} + 0.1 * \begin{bmatrix} -\frac{d}{dW}(\text{cost}(W,b \\ -\frac{d}{db}(\text{cost}(W,b) \end{bmatrix}$$

$$\begin{bmatrix} -\frac{d}{dW}(\text{cost}(W,b)) \\ -\frac{d}{db}(\text{cost}(W,b)) \end{bmatrix} =$$

$$\frac{\begin{bmatrix} 0.39435494 \\ -0.3878391 \end{bmatrix} - \begin{bmatrix} 0.38903466 \\ -0.40474084 \end{bmatrix}}{0.1} \Rightarrow \text{grad1}$$

$$\begin{bmatrix} 0.0532028 \\ 0.1690174 \end{bmatrix}$$

Damit wurde intern die Abstiegsrichtung

$$\begin{bmatrix} -\frac{d}{dW}(\text{cost}(W,b)) \mid \{W=W_0, b=b_0\} \\ -\frac{d}{db}(\text{cost}(W,b)) \mid \{W=W_0, b=b_0\} \end{bmatrix} = \begin{bmatrix} 0.0532028 \\ 0.1690174 \end{bmatrix}$$

genutzt.

Epoch: 0001 cost= 0.174632162 W=

[0.39232808] b= [-0.37236074]

$\text{cost}(W,b)\,|\,\{W=0.39232808, b=-0.37236074\} \Rightarrow \text{cost2}$

$$0.1746321584$$

$$\frac{\begin{bmatrix}0.39232808\\-0.37236074\end{bmatrix}-\begin{bmatrix}0.39435494\\-0.3878391\end{bmatrix}}{0.1} \Rightarrow \text{grad2}$$

$$\begin{bmatrix}-0.0202686\\0.1547836\end{bmatrix}$$

Epoch: 0002 cost= 0.172295541 W= [0.39030755] b= [-0.35708445]

$\text{cost}(W,b)\,|\,\{W=0.39030755, b=-0.35708445\} \Rightarrow \text{cost3}$

$$0.1722955544$$

$$\frac{\begin{bmatrix}0.39030755\\-0.35708445\end{bmatrix}-\begin{bmatrix}0.39232808\\-0.37236074\end{bmatrix}}{0.1} \Rightarrow \text{grad3}$$

$$\begin{bmatrix}-0.0202053\\0.1527629\end{bmatrix}$$

Epoch: 0003 cost= 0.170018047 W= [0.38831297] b= [-0.34200448]

$\text{cost}(W,b)\,|\,\{W=0.38831297, b=-0.34200448\} \Rightarrow \text{cost4}$

$$0.1700180453$$

$$\frac{\begin{bmatrix}0.38831297\\-0.34200448\end{bmatrix}-\begin{bmatrix}0.39030755\\-0.35708445\end{bmatrix}}{0.1} \Rightarrow \text{grad4}$$

$$\begin{bmatrix}-0.0199458\\0.1507997\end{bmatrix}$$

Analytische Berechnung des neg. Gradienten im Startpunkt:

==========

$$-\begin{bmatrix}\frac{d}{dW}(\text{cost}(W,b))\\\frac{d}{db}(\text{cost}(W,b))\end{bmatrix}\,|\,\{W=0.38903466, b=-0.40474084\}$$

$$\begin{bmatrix} 1.412503317 \\ 0.3476792369 \end{bmatrix}$$

ans*0.1

$$\begin{bmatrix} 0.1412503317 \\ 0.03476792369 \end{bmatrix}$$

von tensorflow berechnete Abstiegsrichtung:

grad1

$$\begin{bmatrix} 0.0532028 \\ 0.1690174 \end{bmatrix}$$

Kosten im Startpunkt:

cost0

0.1891133537

Anlage:

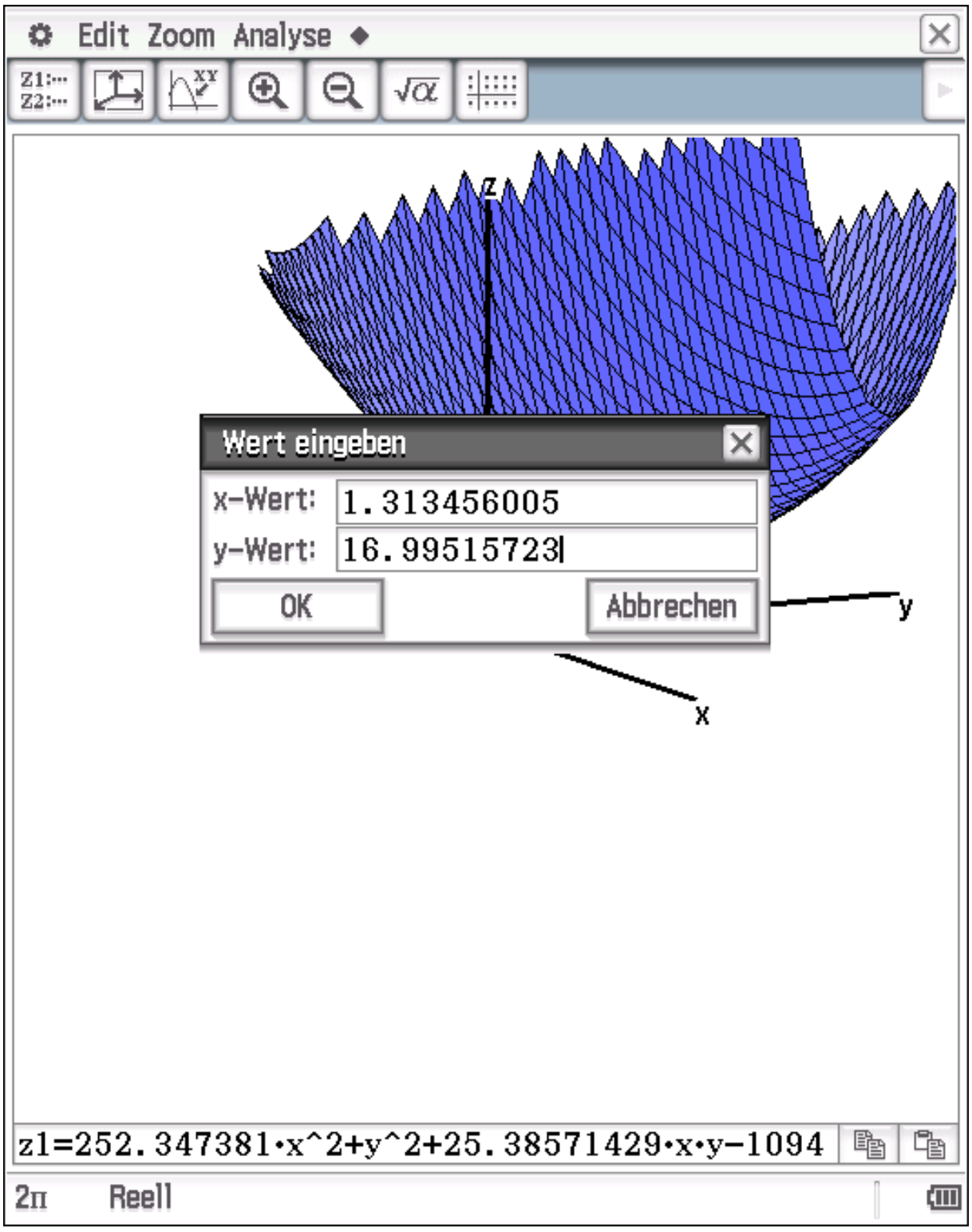
Bild mit Zielfunktion bei linearer Regression

Download für dieses Dokument:

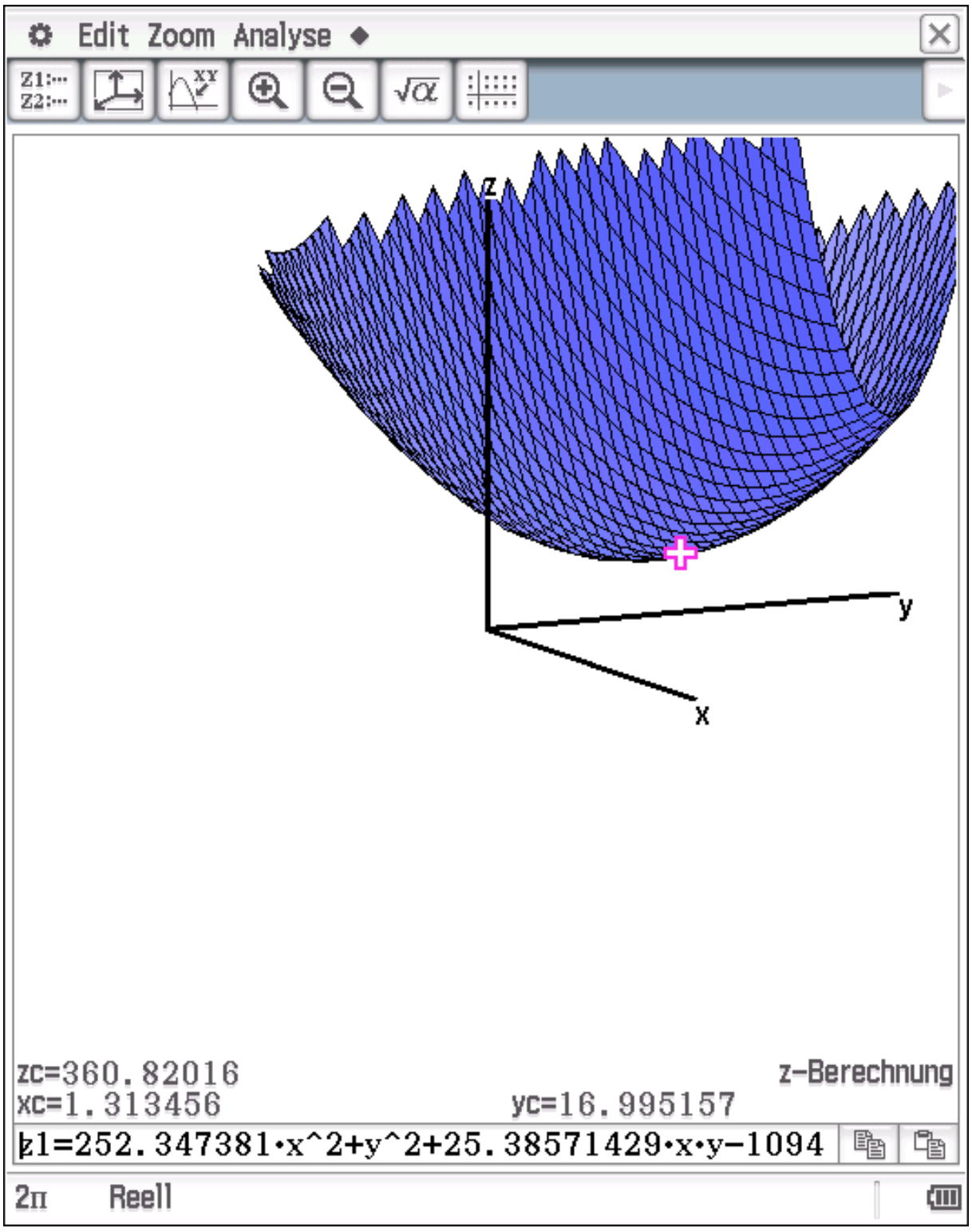
www.informatik.htw-dresden.de/

~paditz/Tensorflow-Ue08.pdf

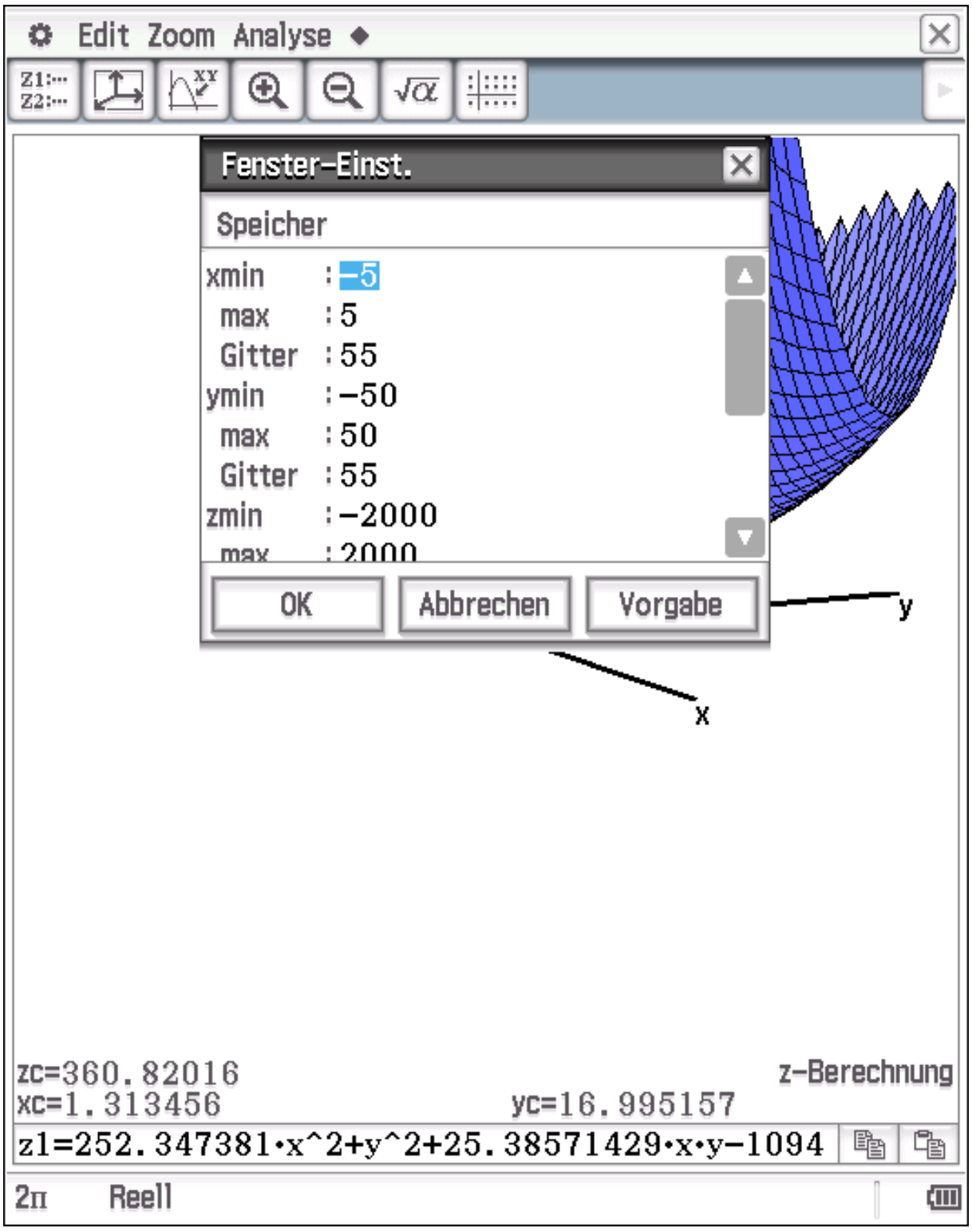
3D-Grafik der Verlustfunktion loss(W,b)



Das globale Min. von loss(W,b) ist erreicht:



Betrachtungsfenstereinstellung:



Minimum mit waagerechter Tangentialebene: