Prof. Dr. L. Paditz, 19.03.2019

HTW Dresden,

paditz@htw-dresden.de

Skript1:

A linear regression learning algorithm example using TensorFlow library.

Skript2: Wiederholung

Basic Operations example using TensorFlow library.

==========

Examples:

https://databricks.com/tensorflow/examples

Placeholders:

https://databricks.com/tensorflow/placeholders

Variables:

https://databricks.com/tensorflow/variables

TensorFlow Randomness:

https://databricks.com/tensorflow/tensorflow-randomness

Visualisation with TensorBoard:

https://databricks.com/tensorflow/visualisation

Dimensionality and Broadcasting:

https://databricks.com/tensorflow/dimensionality-and-b

Broadcasting semantics:

https://www.tensorflow.org/xla/broadcasting

Skript1:

mit ClassPad:

```
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,▶
dim(ans)
                                                  17
LinearReg train_X,train_Y,1,y1,On
                                                done
DispStat
                                                done
=======================
Lineare Regression(Train)
 y=a·x+b
     a = 0.2516349
     b = 0.7988012
     r = 0.8323918
    r² = 0.692876
 MSe = 0.174372
=======================
```

Tensorflow: verschiedene **training_epochs** und **learning_rate**

```
Training cost= 0.084093794 W= 0.2968933 b=
0.4611957
Training cost= 0.081328355 W= 0.23506787 b=
0.8165899
Training cost= 0.07699074 W= 0.24960834 b=
```

```
0.80136055
MSerr=0.07699074*(2*17)/15
                                    MSerr=0.174512344
stop
STAT-Menü
test_X:={6.83, 4.668, 8.9, 7.91, 5.7, 8.7, 3.1▶
                {6.83,4.668,8.9,7.91,5.7,8.7,3.1,2.1}
test_Y:={1.84, 2.273, 3.2, 2.831, 2.92, 3.24, ▶
          {1.84,2.273,3.2,2.831,2.92,3.24,1.35,1.03}
dim(ans)
                                                    8
LinearReg test_X,test_Y,1,y1,On
                                                 done
DispStat
                                                 done
=======================
Lineare Regression(Test)
 y=a·x+b
     a = 0.293093
     b = 0.5803128
     r = 0.8775618
    r² = 0.7701147
 MSe = 0.1945445
=======================
STAT-Menü
stop
```

Tensorflow:

Skript1 mit zufälligen Startwerten für W und b:

```
python3
'''
A linear regression learning algorithm example using
TensorFlow library.
Author: Aymeric Damien
Project:
https://github.com/aymericdamien/TensorFlow-Examples/
'''
import tensorflow as tf
import numpy
import matplotlib.pyplot as plt
rng = numpy.random

# Parameters
# learning_rate = 0.01 bzw. 0.1
learning_rate = 0.01
# training_epochs = 1000
training_epochs = 10000
# display_step = 50
display_step = 500

# 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]
```

```
# Testing example, as requested (Issue #2)
test_X = numpy.asarray([6.83, 4.668, 8.9, 7.91,
5.7, 8.7, 3.1, 2.1])
test_Y = numpy.asarray([1.84, 2.273, 3.2, 2.831,
2.92, 3.24, 1.35, 1.03])

# 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
#  Note, minimize() knows to modify W and b
because Variable objects are trainable=True by default
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:
    # Run the initializer
    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')
    # Testing example, as requested (Issue #2)
    print("Testing... (Mean square loss
Comparison)")
    testing_cost = sess.run(
        tf.reduce_sum(tf.pow(pred - Y, 2)) / (2
* test_X.shape[0]),
```

```
            feed_dict={X: test_X, Y: test_Y})  # same
function as cost above
        print("Testing cost=", testing_cost)
        print("Absolute mean square loss difference:",
abs(
            training_cost - testing_cost))
        # Graphic display
        plt.plot(train_X, train_Y, 'ro', label='Original
data')
        plt.plot(test_X, test_Y, 'bo', label='Testing 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.
>>> '''
... A linear regression learning algorithm example using
TensorFlow library.
... Author
... Project:
https://github.com/aymericdamien/TensorFlow-Examples/
... '''
'\nA linear regression learning algorithm example using
```

```
TensorFlow library. \nAuthor\nProject:
https://github.com/aymericdamien/TensorFlow-Examples
>>> import tensorflow as tf
>>> import numpy
>>> import matplotlib.pyplot as plt
>>> rng = numpy.random
>>>
>>> # Parameters
... # learning_rate = 0.01 bzw. 0.1
... learning_rate = 0.01
>>> # training_epochs = 1000
... training_epochs = 10000
>>> # display_step = 50
... display_step = 500
>>>
>>> # 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]
>>>
>>> # Testing example, as requested (Issue #2)
... test_X = numpy.asarray([6.83, 4.668, 8.9,
7.91, 5.7, 8.7, 3.1, 2.1])
>>> test_Y = numpy.asarray([1.84, 2.273, 3.2,
2.831, 2.92, 3.24, 1.35, 1.03])
>>>
>>> # tf Graph Input
```

```
... X = tf.placeholder("float")
>>> Y = tf.placeholder("float")
>>>
>>> # Set model weights
... W = tf.Variable(rng.randn(), name="weight")
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.
>>> 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
... #  Note, minimize() knows to modify W and b
because Variable objects are trainable=True by default
... optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimi
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.
>>>
>>> # Initialize the variables (i.e. assign their default
value)
... init = tf.global_variables_initializer()
>>>
>>> # Start training
... with tf.Session() as sess:
...     # Run the initializer
...     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')
...       # Testing example, as requested (Issue #2)
...       print("Testing... (Mean square loss
Comparison)")
...       testing_cost = sess.run(
...           tf.reduce_sum(tf.pow(pred - Y, 2)) /
(2 * test_X.shape[0]),
...           feed_dict={X: test_X, Y: test_Y})  #
same function as cost above
...       print("Testing cost=", testing_cost)
...       print("Absolute mean square loss difference:",
abs(
...           training_cost - testing_cost))
...       # Graphic display
...       plt.plot(train_X, train_Y, 'ro', label='Original
data')
...       plt.plot(test_X, test_Y, 'bo', label='Testing
data')
...       plt.plot(train_X, sess.run(W) * train_X +
sess.run(b), label='Fitted line')
...       plt.legend()
...       plt.show()
...
2019-03-26 13:14:53.558391: I
tensorflow/core/platform/profile_utils/cpu_utils.c
CPU Frequency: 2494315000 Hz
2019-03-26 13:14:53.558647: I
tensorflow/compiler/xla/service/service.cc:150]
XLA service 0x20630a0 executing computations on
```

platform Host. Devices:

2019-03-26 13:14:53.558773: I

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

StreamExecutor device (0): <undefined>,

<undefined>

Epoch: 0500 cost= 0.085635014 W= 0.19790356 b=
1.1733209

Epoch: 1000 cost= 0.079545885 W= 0.22159624 b=
1.0028771

Epoch: 1500 cost= 0.077750765 W= 0.23442905 b=
0.91055906

Epoch: 2000 cost= 0.077219352 W= 0.24137971 b=
0.86055654

Epoch: 2500 cost= 0.077060856 W= 0.24514504 b=
0.83346885

Epoch: 3000 cost= 0.077012964 W= 0.24718496 b=
0.8187941

Epoch: 3500 cost= 0.076998182 W= 0.24829005 b=
0.8108439

Epoch: 4000 cost= 0.076993421 W= 0.24888843 b=
0.8065385

Epoch: 4500 cost= 0.076991804 W= 0.24921261 b=
0.8042072

Epoch: 5000 cost= 0.076991208 W= 0.2493885 b=
0.80294234

Epoch: 5500 cost= 0.076990969 W= 0.2494826 b=
0.8022646

Epoch: 6000 cost= 0.076990865 W= 0.24953318 b=
0.8019009

```
Epoch: 6500 cost= 0.076990828 W= 0.24955891 b=
0.8017156
Epoch: 7000 cost= 0.076990791 W= 0.24957383 b=
0.80160946
Epoch: 7500 cost= 0.076990768 W= 0.24958722 b=
0.80151314
Epoch: 8000 cost= 0.076990783 W= 0.24959211 b=
0.8014765
Epoch: 8500 cost= 0.076990761 W= 0.24959634 b=
0.8014467
Epoch: 9000 cost= 0.076990761 W= 0.24960251 b=
0.8014019
Epoch: 9500 cost= 0.076990746 W= 0.24960734 b=
0.8013679
Epoch: 10000 cost= 0.076990739 W= 0.24960737 b=
0.8013675
Optimization Finished!
Training cost= 0.07699074 W= 0.24960737 b=
0.8013675

Testing... (Mean square loss Comparison)
Testing cost= 0.079108685
Absolute mean square loss difference: 0.0021179467

[<matplotlib.lines.Line2D object at 0x7fa9400fb940>]
[<matplotlib.lines.Line2D object at 0x7fa9400fbba8>]
[<matplotlib.lines.Line2D object at 0x7fa9400fbe80>]
<matplotlib.legend.Legend object at 0x7fa940090438>
```

==========

Skript2:

mit ClassPad:

Basic constant operations:

a:=2

2

b:=3

3

add=a+b

add=5

mul=a*b

mul=6

Matrix Multiplication:

matrix1:=[[3.,3.]]

[3 3]

matrix2:=[[2.],[2.]]

$$\begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

product:=matrix1*matrix2

[12]

Skript2 mit Tensorflow:

python3

'''

Basic Operations example using TensorFlow library.

Author: Aymeric Damien

Project:

```
https://github.com/aymericdamien/TensorFlow-Examples/
'''

import tensorflow as tf

# Basic constant operations
# The value returned by the constructor represents the
output
# of the Constant op.
a = tf.constant(2)
b = tf.constant(3)

# Launch the default graph.
with tf.Session() as sess:
    print("a=2, b=3")
    print("Addition with constants: %i" %
sess.run(a+b))
    print("Multiplication with constants: %i" %
sess.run(a*b))

# Basic Operations with variable as graph input
# The value returned by the constructor represents the
output
# of the Variable op. (define as input when running
session)
# tf Graph input
a = tf.placeholder(tf.int16)
b = tf.placeholder(tf.int16)
```

```
# Define some operations
add = tf.add(a, b)
mul = tf.multiply(a, b)

# Launch the default graph.
with tf.Session() as sess:
    # Run every operation with variable input
    print("Addition with variables: %i" %
sess.run(add, feed_dict={a: 2, b: 3}))
    print("Multiplication with variables: %i" %
sess.run(mul, feed_dict={a: 2, b: 3}))


# ----------------
# More in details:
# Matrix Multiplication from TensorFlow official
tutorial

# Create a Constant op that produces a 1x2 matrix.
The op is
# added as a node to the default graph.
#
# The value returned by the constructor represents the
output
# of the Constant op.
matrix1 = tf.constant([[3., 3.]])

# Create another Constant that produces a 2x1 matrix.
matrix2 = tf.constant([[2.],[2.]])
```

```
# Create a Matmul op that takes 'matrix1' and
'matrix2' as inputs.
# The returned value, 'product', represents the result
of the matrix
# multiplication.
product = tf.matmul(matrix1, matrix2)

# To run the matmul op we call the session 'run()'
method, passing 'product'
# which represents the output of the matmul op.  This
indicates to the call
# that we want to get the output of the matmul op
back.
#
# All inputs needed by the op are run automatically by
the session.  They
# typically are run in parallel.
#
# The call 'run(product)' thus causes the execution of
threes ops in the
# graph: the two constants and matmul.
#
# The output of the op is returned in 'result' as a
numpy `ndarray` object.
with tf.Session() as sess:
    result = sess.run(product)
    print(result)
# ==> [[ 12.]]
```

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.
>>> '''
... Basic Operations example using TensorFlow library.
... Author
... Project:
https://github.com/aymericdamien/TensorFlow-Examples/
... '''
'\nBasic Operations example using TensorFlow
library.\nAuthor\nProject:
https://github.com/aymericdamien/TensorFlow-Examples▶
>>>
>>> import tensorflow as tf
>>>
>>> # Basic constant operations
... # The value returned by the constructor represents
the output
... # of the Constant op.
... a = tf.constant(2)
>>> b = tf.constant(3)
>>>
>>> # Launch the default graph.
... with tf.Session() as sess:
...     print("a=2, b=3")
```

```
...         print("Addition with constants: %i" %
sess.run(a+b))
...         print("Multiplication with constants: %i" %
sess.run(a*b))
...
a=2, b=3
Addition with constants: 5
Multiplication with constants: 6
>>> # Basic Operations with variable as graph input
... # The value returned by the constructor represents
the output
... # of the Variable op. (define as input when
running session)
... # tf Graph input
... a = tf.placeholder(tf.int16)
>>> b = tf.placeholder(tf.int16)
>>>
>>> # Define some operations
... add = tf.add(a, b)
>>> mul = tf.multiply(a, b)
>>>
>>> # Launch the default graph.
... with tf.Session() as sess:
...         # Run every operation with variable input
...         print("Addition with variables: %i" %
sess.run(add, feed_dict={a: 2, b: 3}))
...         print("Multiplication with variables: %i" %
sess.run(mul, feed_dict={a: 2, b: 3}))
...
```

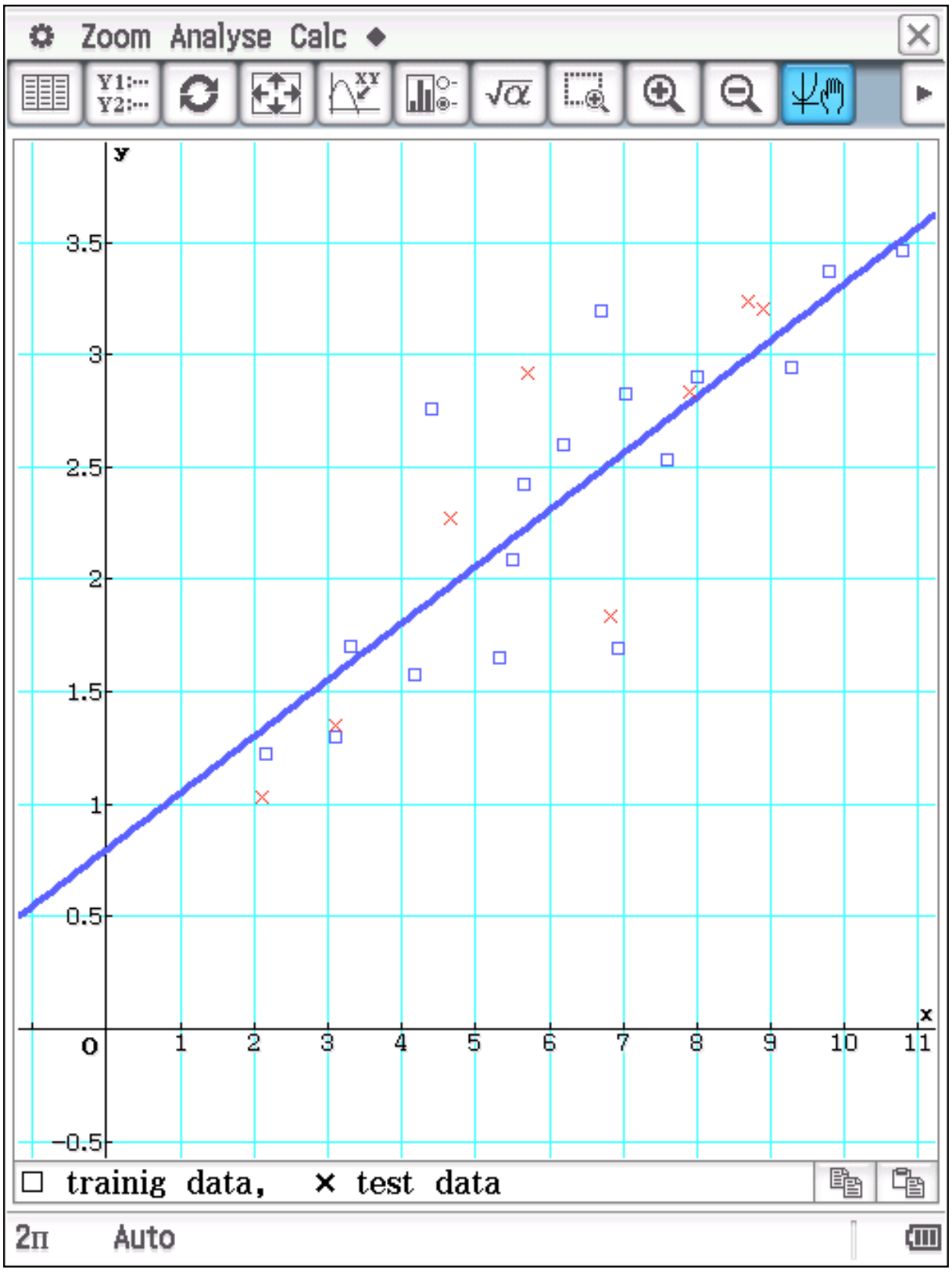
```
Addition with variables: 5
Multiplication with variables: 6
>>>
>>> # ----------------
... # More in details:
... # Matrix Multiplication from TensorFlow official
tutorial
...
>>> # Create a Constant op that produces a 1x2
matrix.  The op is
... # added as a node to the default graph.
... #
... # The value returned by the constructor represents
the output
... # of the Constant op.
... matrix1 = tf.constant([[3., 3.]])
>>>
>>> # Create another Constant that produces a 2x1
matrix.
... matrix2 = tf.constant([[2.],[2.]])
>>>
>>> # Create a Matmul op that takes 'matrix1' and
'matrix2' as inputs.
... # The returned value, 'product', represents the
result of the matrix
... # multiplication.
... product = tf.matmul(matrix1, matrix2)
>>>
>>> # To run the matmul op we call the session
```

```
'run()' method, passing 'product'
... # which represents the output of the matmul op.
This indicates to the call
... # that we want to get the output of the matmul
op back.
... #
... # All inputs needed by the op are run
automatically by the session.  They
... # typically are run in parallel.
... #
... # The call 'run(product)' thus causes the
execution of threes ops in the
... # graph: the two constants and matmul.
... #
... # The output of the op is returned in 'result' as
a numpy `ndarray` object.
... with tf.Session() as sess:
...       result = sess.run(product)
...       print(result)
... # ==> [[ 12.]]
...
[[12.]]
>>>
```

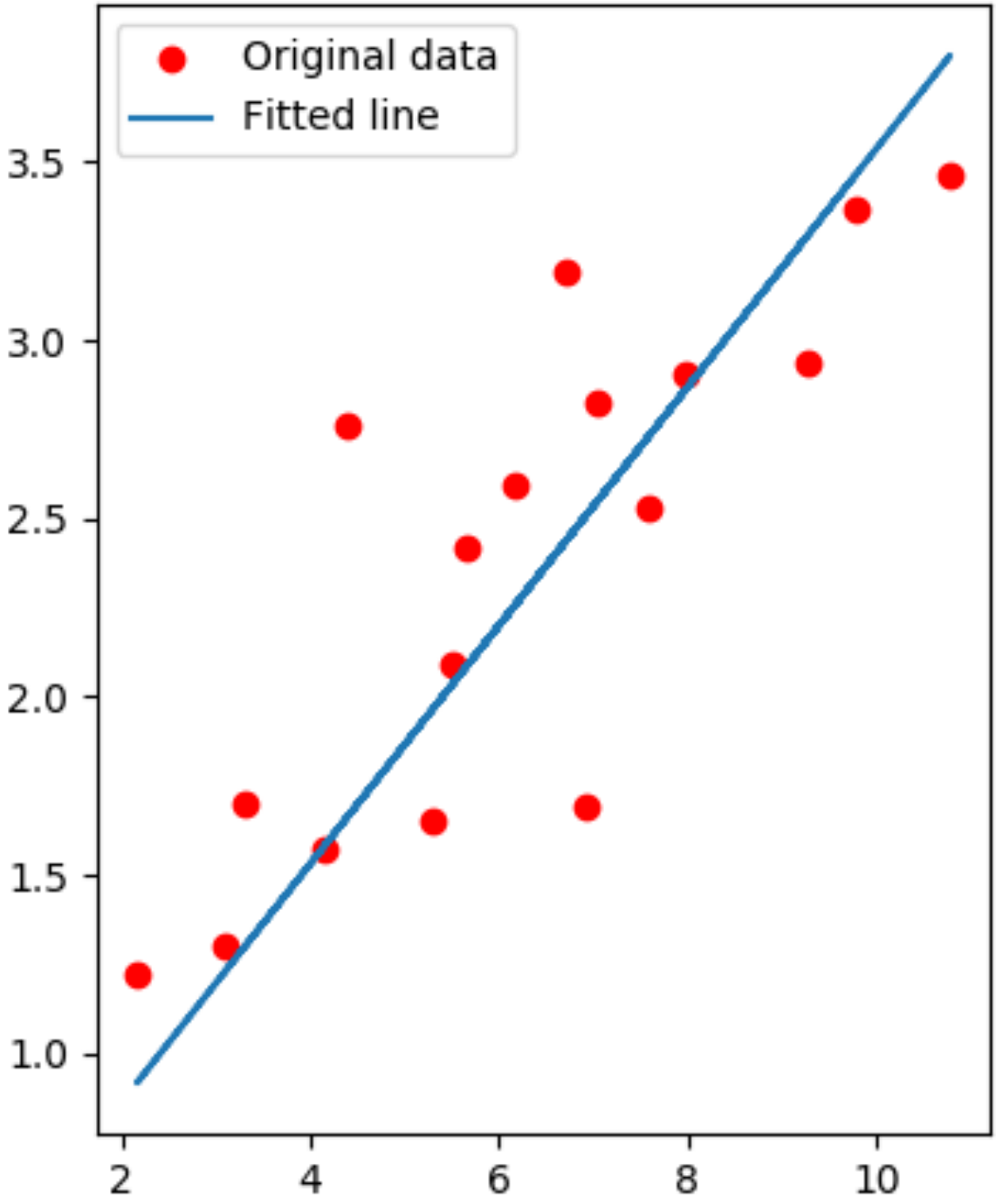
Download für dieses Dokument:

www.informatik.htw-dresden.de/
~paditz/Tensorflow-Ue17.pdf

Lineare Regression: mit ClassPad



Lineare Regression: mit Tensorflow



Lineare Regression: mit Tensorflow