Prof. Dr. L. Paditz, 14.02.2019

HTW Dresden,

paditz@htw-dresden.de

Mathematik-Grundlagen als eActivty mit ClassPad

binäre logistische Regression in TensorFlow

==========

Quelle:

Tensorflow für Dummies

ISBN 978-3-527-71547-3

eine binäre logistische Regression in TensorFlow

Daten in zwei Kategorien:

Kategorie 0 (18Daten) bzw. Kategorie 1 (22Daten)

z.B. **verbundene Datenliste** mit 40 y-Daten:

{0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,1.,

0., 0., 1., 0., 0., 0., 1., 0., 0.,1., 0.,

1., 1., 1., 1., 1., 1., 1., 1.,1., 1., 1.,

1., 1., 1., 1., 1., 1., 1.}

Kategorie 0: "Weiterschlafen"

Kategorie 1: "Aufstehen"

"Weckerlautstärke x" von 0(sehr leise) bis 5(sehr laut)

40 Werte: (von Tensorflow generiert)

x =

[0. 0.12820514 0.25641027 0.38461542

0.51282054 0.64102566

 0.76923084 0.89743596 1.0256411 1.1538463

1.2820513 1.4102565

 1.5384617 1.6666667 1.7948719 1.923077

2.0512822 2.1794872

 2.3076925 2.4358976 2.5641026 2.692308

2.820513 2.948718

3.0769234 3.2051284 3.3333335 3.4615386

3.5897439 3.717949

3.846154 3.9743593 4.1025643 4.2307696

4.3589745 4.4871798

4.615385 4.74359 4.871795 5.0000005]

mit ClassPad generiert:

xliste:=seq(x,x,0,5,$\frac{5}{39}$)

{0,0.1282051282,0.2564102564,0.3846153846,0.▶

yliste:={0., 0., 0., 0., 0., 0., 0., 0., 0., 0.▶

{0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,1,0,0,1,0▶

STAT-Menü

Sigmoidfunktion (Schwanenhalsfunktion):

DelVar x,y,m,b

done

Define sig(x,m,b)=$\frac{1}{1+e^{-(m\cdot x+b)}}$

done

sig(x,m,b)

$\frac{1}{2.718281828^{-m\cdot x-b}+1}$

Abkürzung:

Define h(x)=sig(x,m,b)

done

h(x)

$$\frac{1}{2.718281828^{-m\cdot x-b}+1}$$

Zielfunktion für binäre logist. Regress.:

$$\sum_{i=1}^{40}(-y_i\cdot\ln(h(x_i))-(1-y_i)\cdot(1-\ln(h(x_i))))\rightarrow\min!$$

DelVar x,y,m,b

done

sum(yliste·ln(1+e^(−(m·xliste+b)))+(yliste−1)·(1+ln(1+e▶

$$-\ln\left(e^{-b-\frac{5\cdot m}{39}}+1\right)-\ln\left(e^{-b-\frac{5\cdot m}{13}}+1\right)+\ln\left(e^{-b-\frac{5\cdot m}{3}}+1\right)-\ln\blacktriangleright$$

Define L(m,b)=$-\ln\left(e^{-b-\frac{5\cdot m}{39}}+1\right)-\ln\left(e^{-b-\frac{5\cdot m}{13}}+1\right)+\ln($▶

done

mögliche Werte der Zielfunktion:

erstes Ergebnis mit Tensorflow:

L(m,b)/38|m=4.0140586 and b=−13.645958

−3.749272275

Define y1(x)=$\frac{1}{1+e^{-(4.0140586\cdot x-13.645958)}}$

done

L(m,b)/38|m=27.194965 and b=−85.18576

−21.27021611

Define y2(x)=$\frac{1}{1+e^{-(27.194965\cdot x-85.18576)}}$

done

Ergebnis mit ClassPad (LM-Verfahren):

L(m,b)/38|m=330.2585613 and b=-1081.098865

-266.0314303

L(m,b)/38|m=624.3857114 and b=-1360.114578

-276.1748523

Define $y3(x)=\frac{1}{1+e^{-(624.3857114 \cdot x-1360.114578)}}$

done

stop

==========

ClassPad: LogisticReg: $y=\frac{c}{1+a \cdot e^{-b \cdot x}}$

yyliste:=yliste+0.1

{0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,1.1 ▶

LogisticReg xliste,yyliste,y2,On

done

DispStat

done

==========

Logistische Reg.

y=c/(1+a·e^(-b·x))

a = 24.867343

b = 1.3927195

c = 1.202717

MSe = 0.0920341

==========

Ergebnis von LogisticReg unbrauchbar:

$y2(x)=\frac{1.202716987}{1+24.86734323 \cdot e^{-1.392719486 \cdot x}}$

Eine **Sigmoidfunktion, Schwanenhalsfunktion** oder **S-Funktion** ist eine mathematische Funktion mit einem S-förmigen Graphen. Oft wird der Begriff Sigmoidfunktion auf den Spezialfall logistische Funktion bezogen, die durch die Gleichung

$$\mathrm{sig}(t)=\frac{1}{1+e^{-t}}=\frac{e^{t}}{1+e^{t}}=\frac{1}{2}\cdot\left(1+\tanh\left(\frac{t}{2}\right)\right)$$

beschrieben wird. Dabei ist e die eulersche Zahl. Diese spezielle Sigmoidfunktion ist also im Wesentlichen eine skalierte und verschobene Tangens-hyperbolicus-Funktion und hat entsprechende Symmetrien.

Im Allgemeinen ist eine Sigmoidfunktion eine beschränkte und differenzierbare reelle Funktion mit einer durchweg positiven oder durchweg negativen ersten Ableitung und genau einem Wendepunkt.
Außer der logistischen Funktion enthält die Menge der Sigmoidfunktionen den Arkustangens, den Tangens hyperbolicus und die Fehlerfunktion, die sämtlich transzendent sind, aber auch einfache algebraische Funktionen wie $f(x)=\frac{x}{\sqrt{1+x^2}}$. Das Integral jeder stetigen, positiven Funktion mit einem "Berg" (genauer: mit genau einem lokalen Maximum und keinem lokalen Minimum, z. B. die gaußsche Glockenkurve) ist ebenfalls eine Sigmoidfunktion. Daher sind viele kumulierte Verteilungsfunktionen **sigmoidal.**

Zusammenhang zur Wahrscheinlichkeitsrechnung:

$sig(x,m,b)=\frac{1}{1+e^{-(m\cdot x+b)}}\in(0,1)$ ist die Wahrscheinlichkeit, dass ein Datenpunkt der Kategorie 1 zugeordnet wird.

$1-sig(x,m,b)$ ist die Wahrscheinlichkeit, dass ein Datenpunkt der Kategorie 0 zugeordnet wird.

Sei $h(x)=sig(x,m,b)$, dann gilt für $y\in\{0.0,\ 1.0\}$

$L(y)=h(x)^y(1-h(x))^{1-y}$

$=h(x)$ für $y=1.0$ bzw. $=1-h(x)$ für $y=0.0$

$L(y)$ ist die Wkt. ("Likelihood") für das Ereignis y.

Idealfall: $h(x)$ ist ein geeignetes Modell

$y=1$ für $x(=1)$ und $L(1)=1$(d.h. $h(x)=1$) und

$y=0$ für $x(=0)$ und $L(0)=1$(d.h. $h(x)=0$).

anderes Extrem: $h(x)$ ist kein geeignetes Modell

$y=1$ für $x(=1)$ und $L(1)=0$(d.h. $h(x)=0$) und

$y=0$ für $x(=0)$ und $L(0)=0$(d.h. $h(x)=0$).

Zielstellung: $h(x)$ optimieren auf Grundlage der verbundenen x-Daten und y-Daten.

$L(y)\rightarrow max.$ bzw.

$\ln(L(y))=\ln\left(h(x)^y(1-h(x))^{1-y}\right)=$

$\ln\left(h(x)^y\right)+\ln\left((1-h(x))^{1-y}\right)=$

$y\cdot\ln(h(x))+(1-y)\cdot\mathbf{\ln(1-h(x))}.$

Hier wird stattdessen folgende Funktion genutzt:

$\ln(L(y))=y\cdot\ln(h(x))+(1-y)\cdot(1-\ln(h(x)))\rightarrow\max.$

Diskussion:

y=1: ln(h(x))<0 wird max. bei h(x)≈1

(es ist $-\infty<\ln(h(x))<0$, da $0<h(x)<1$ gilt)

y=0: 1−ln(h(x))>1 wird max.(∞) bei h(x)≈0

bzw.

$\text{loss}=-\ln(L(y))=-y\cdot\ln(h(x))-(1-y)\cdot(1-\ln(h(x)))\rightarrow\min.$

==========

ClassPad-Ergebnis: LM-Methode:

(Levenberg-Marquardt-Verfahren)

Programm LogiSReg liefert

$\text{vecmb} = \begin{bmatrix} m=624.3857114 \\ b=-1360.114578 \end{bmatrix}$

loss = −247.3230491

Programm LogiSReg(xList, yList, m0, b0, μ0, I):

LogiSReg(xliste, yliste, 624, −1360, 0.001, 35)

done

i

35

vecmb

$$\begin{bmatrix}330.2585613\\-1081.098865\end{bmatrix}$$

vecmb-$\begin{bmatrix}330.2585613\\-1081.098865\end{bmatrix}$

$$\begin{bmatrix}-2.3861\text{E-}8\\-4.3044\text{E-}7\end{bmatrix}$$

-vecmb+$\begin{bmatrix}330.258561276139\\-1081.09886543044\end{bmatrix}$

$$\begin{bmatrix}0\\0\end{bmatrix}$$

loss

-276.1748522

stop

==========

Skript:

```
python3
''' Zeigt eine binäre logistische Regression in TensorFlow
'''

import tensorflow as tf

# Eingabewerte
N=40
x = tf.lin_space(0., 5., N)
y = tf.constant([0., 0., 0., 0., 0., 0., 0., 0.,
0., 0.,
                 1., 0., 0., 1., 0., 0., 0.,
1., 0., 0.,
```

```
                    1., 0., 1., 1., 1., 1., 1.,
1., 1., 1.,
                    1., 1., 1., 1., 1., 1., 1.,
1., 1., 1.])

# Variablen
m = tf.Variable(22.)
b = tf.Variable(-70.)

# Modell und Verlust berechnen
# model = tf.nn.sigmoid(tf.add(tf.multiply(x, m),
b))
model = 1 / (1 + tf.exp(-(m * x + b)))
loss = -1. * tf.reduce_sum(y * tf.log(model) + (1.
- y) * (1. - tf.log(model)))/(N-2)

# Optimierer erzeugen
learn_rate = 0.001
num_epochs = 100000
optimizer =
tf.train.GradientDescentOptimizer(learn_rate).minimize(

# Variablen initialisieren
init = tf.global_variables_initializer()

# Sitzung starten
with tf.Session() as sess:
    sess.run(init)
    # Optimierer ausführen
```

```
for epoch in range(num_epochs):
    sess.run(optimizer)
# Ergebnisse anzeigen
print('m =', sess.run(m))
print('b =', sess.run(b))
print('loss =', sess.run(loss))
print('x =', sess.run(x))
print('y =', sess.run(y))
```

Rechnerergebnisse:

==========

```
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.
>>> ''' Zeigt eine binäre logistische Regression in
TensorFlow '''
' Zeigt eine binäre logistische Regression in TensorFlow '
>>>
>>> import tensorflow as tf
>>>
>>> # Eingabewerte
... N=40
>>> x = tf.lin_space(0., 5., N)
>>> y = tf.constant([0., 0., 0., 0., 0., 0., 0.,
0., 0., 0.,
...                   1., 0., 0., 1., 0., 0.,
```

```
0., 1., 0., 0.,
...                     1., 0., 1., 1., 1., 1.,
1., 1., 1., 1.,
...                     1., 1., 1., 1., 1., 1.,
1., 1., 1., 1.])
>>>
>>> # Variablen
... m = tf.Variable(22.)
>>> b = tf.Variable(-70.)
>>>
>>> # Modell und Verlust berechnen
... # model = tf.nn.sigmoid(tf.add(tf.multiply(x,
m), b))
... model = 1 / (1 + tf.exp(-(m * x + b)))
>>> loss = -1. * tf.reduce_sum(y * tf.log(model) +
(1. - y) * (1. - tf.log(model)))/(N-2)
>>>
>>> # Optimierer erzeugen
... learn_rate = 0.001
>>> num_epochs = 100000
>>> optimizer =
tf.train.GradientDescentOptimizer(learn_rate).minimize(
>>>
>>> # Variablen initialisieren
... init = tf.global_variables_initializer()
>>>
>>> # Sitzung starten
... with tf.Session() as sess:
...     sess.run(init)
```

```
...         # Optimierer ausführen
...         for epoch in range(num_epochs):
...             sess.run(optimizer)
...         # Ergebnisse anzeigen
...         print('m =', sess.run(m))
...         print('b =', sess.run(b))
...         print('loss =', sess.run(loss))
...         print('x =', sess.run(x))
...         print('y =', sess.run(y))
...
m = 23.18649
b = -72.78387
loss = -18.245785
x = [0.          0.12820514 0.25641027
0.38461542 0.51282054 0.64102566
 0.76923084 0.89743596 1.0256411  1.1538463
1.2820513  1.4102565
 1.5384617  1.6666667  1.7948719  1.923077
2.0512822  2.1794872
 2.3076925  2.4358976  2.5641026  2.692308
2.820513   2.948718
 3.0769234  3.2051284  3.3333335  3.4615386
3.5897439  3.717949
 3.846154   3.9743593  4.1025643  4.2307696
4.3589745  4.4871798
 4.615385   4.74359    4.871795   5.0000005 ]
y = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
1. 0. 0. 0. 1. 0. 0. 1. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
```

```
1.]
>>>
```

==========

Variante der loss-Funktion:

Skript:

```
python3
''' Zeigt eine binäre logistische Regression in TensorFlow
'''

import tensorflow as tf

# Eingabewerte
N=40
# x = tf.lin_space(0., 5., N)
x = tf.range(0.,5.01,5./39.)
y = tf.constant([0., 0., 0., 0., 0., 0., 0., 0.,
0., 0.,
                 1., 0., 0., 1., 0., 0., 0.,
1., 0., 0.,
                 1., 0., 1., 1., 1., 1., 1.,
1., 1., 1.,
                 1., 1., 1., 1., 1., 1., 1.,
1., 1., 1.])

# Variablen
m = tf.Variable(22.)
b = tf.Variable(-70.)
```

```
# Modell und Verlust berechnen
# model = tf.nn.sigmoid(tf.add(tf.multiply(x, m),
b))
model = 1 / (1 + tf.exp(-(m * x + b)))
loss = -1. * tf.reduce_mean(y * tf.log(model) + (1.
- y) * (1. - tf.log(model))) * N / (N-2)

# Optimierer erzeugen
learn_rate = 0.005
num_epochs = 100000
optimizer =
tf.train.GradientDescentOptimizer(learn_rate).minimize(

# Variablen initialisieren
init = tf.global_variables_initializer()

# Sitzung starten
with tf.Session() as sess:
    sess.run(init)
    # Optimierer ausführen
    for epoch in range(num_epochs):
        sess.run(optimizer)
    # Ergebnisse anzeigen
    print('m =', sess.run(m))
    print('b =', sess.run(b))
    print('loss =', sess.run(loss))
    print('x =', sess.run(x))
    print('y =', sess.run(y))
```

==========

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.
>>> ''' Zeigt eine binäre logistische Regression in
TensorFlow '''
' Zeigt eine binäre logistische Regression in TensorFlow '
>>>
>>> import tensorflow as tf
>>>
>>> # Eingabewerte
... N=40
>>> # x = tf.lin_space(0., 5., N)
... x = tf.range(0.,5.01,5./39.)
>>> y = tf.constant([0., 0., 0., 0., 0., 0., 0.,
0., 0., 0.,
...                   1., 0., 0., 1., 0., 0.,
0., 1., 0., 0.,
...                   1., 0., 1., 1., 1., 1.,
1., 1., 1., 1.,
...                   1., 1., 1., 1., 1., 1.,
1., 1., 1., 1.])
>>>
>>> # Variablen
```

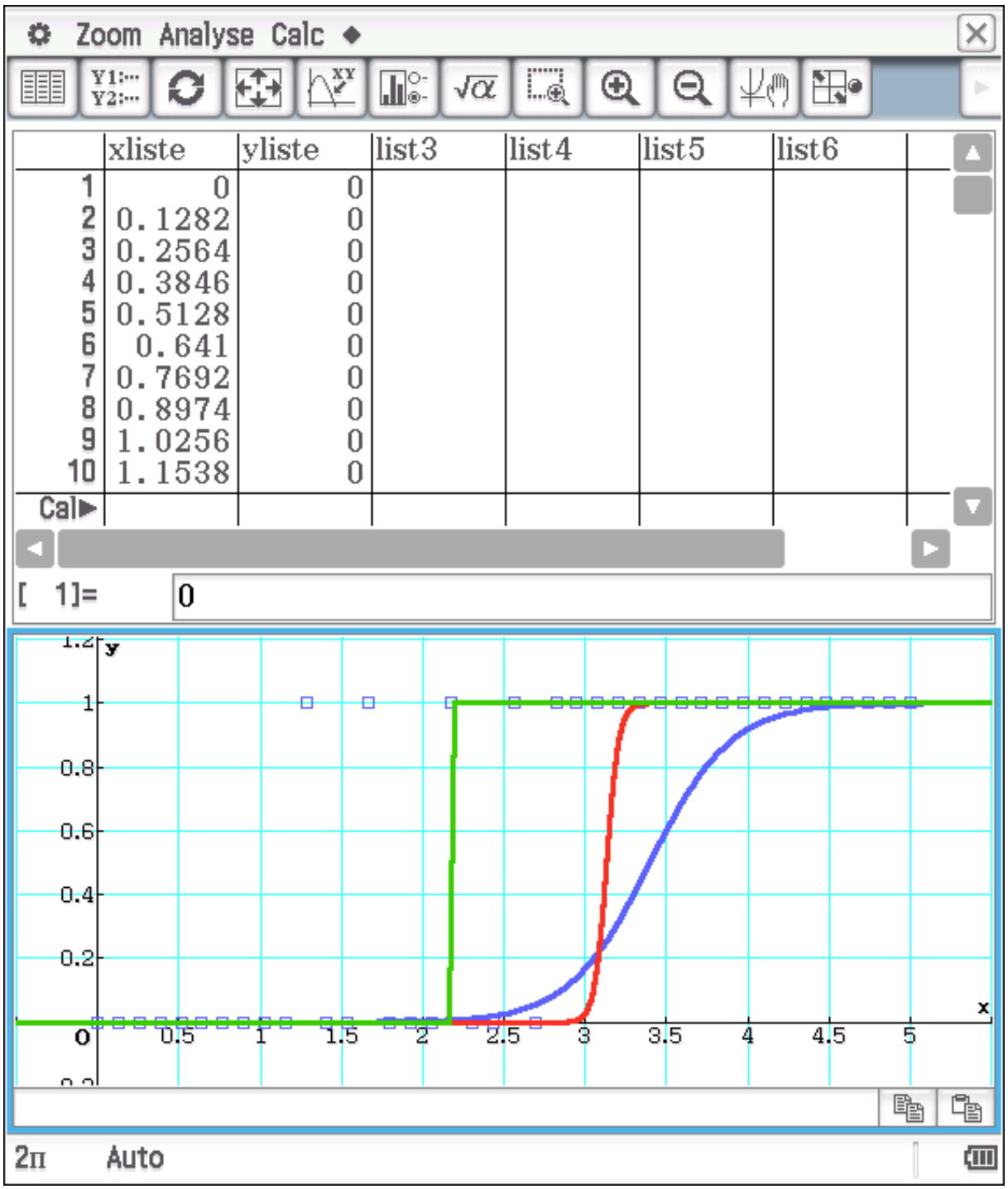
```
... m = tf.Variable(22.)
>>> b = tf.Variable(-70.)
>>>
>>> # Modell und Verlust berechnen
... # model = tf.nn.sigmoid(tf.add(tf.multiply(x,
m), b))
... model = 1 / (1 + tf.exp(-(m * x + b)))
>>> loss = -1. * tf.reduce_mean(y * tf.log(model) +
(1. - y) * (1. - tf.log(model))) * N / (N-2)
>>>
>>> # Optimierer erzeugen
... learn_rate = 0.005
>>> num_epochs = 100000
>>> optimizer =
tf.train.GradientDescentOptimizer(learn_rate).minimize(
>>>
>>> # Variablen initialisieren
... init = tf.global_variables_initializer()
>>>
>>> # Sitzung starten
... with tf.Session() as sess:
...     sess.run(init)
...     # Optimierer ausführen
...     for epoch in range(num_epochs):
...         sess.run(optimizer)
...     # Ergebnisse anzeigen
...     print('m =', sess.run(m))
...     print('b =', sess.run(b))
...     print('loss =', sess.run(loss))
```

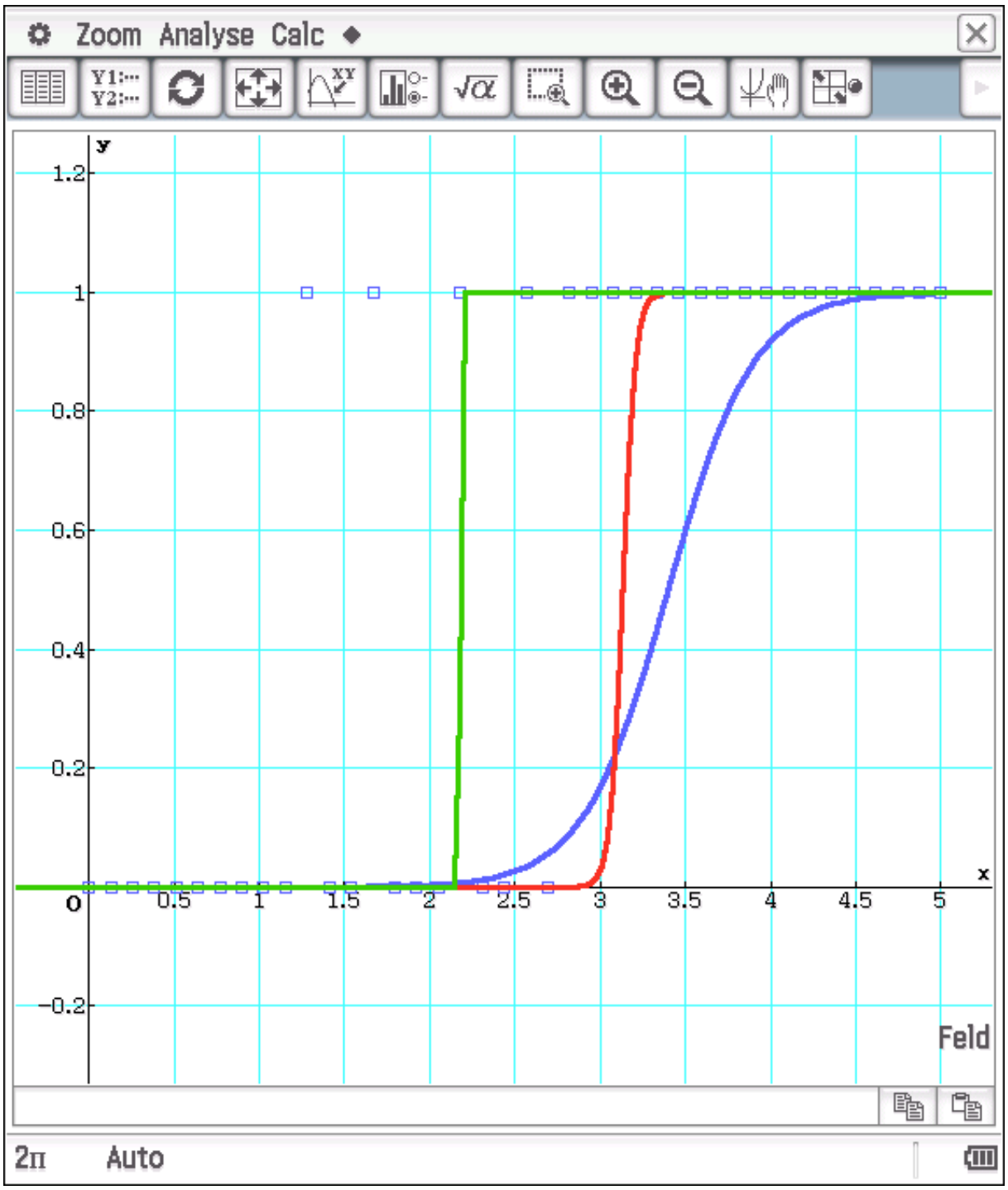
```
...       print('x =', sess.run(x))
...       print('y =', sess.run(y))
...
m = 27.194965
b = -85.18576
loss = -21.270214
x = [0.          0.12820514 0.25641027
0.38461542 0.51282054 0.64102566
 0.7692308  0.8974359  1.0256411  1.1538463
1.2820514  1.4102566
 1.5384618  1.666667   1.7948722  1.9230773
2.0512824  2.1794875
 2.3076925  2.4358976  2.5641026  2.6923077
2.8205128  2.9487178
 3.076923   3.205128   3.333333   3.461538
3.5897431  3.7179482
 3.8461533  3.9743583  4.1025634  4.2307687
4.358974   4.4871793
 4.6153846  4.74359    4.871795   5.0000005 ]
y = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
1. 0. 0. 0. 1. 0. 0. 1. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.]
>>>
```

Download für dieses Dokument:

www.informatik.htw-dresden.de/
~paditz/Tensorflow-Ue11.pdf

Sigmoidfunktionen (Schwanenhalsfunktionen):




Zoom Analyse Calc
y
x
1.2
1
0.8
0.6
0.4
0.2
−0.2
O
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Feld
2π
Auto