Prof. Dr. L. Paditz, 12.02.2019

HTW Dresden,

paditz@htw-dresden.de

Mathematik-Grundlagen als eActivty mit ClassPad

Tensor-Begriff in TensorFlow

==========

Quelle:

https://www.python-kurs.eu/

numpy_arrays_erzeugen.php

Die Syntax von **arange:**

arange([start,] stop[, step], [, dtype=None])

arange liefert gleichmäßig verteilte Werte innerhalb eines gegebenen Intervalles zurück. Die Werte werden innerhalb des halb-offenen Intervalles **[start, stop)** generiert. Falls der **start**-Parameter nicht übergeben wird, wird **start auf 0 gesetzt.** Das Ende des Intervalls wird durch den Parameter **stop** bestimmt.

Üblicherweise wird das Intervall diesen Wert nicht beinhalten, außer in den Fällen, in denen **step** keine Ganzzahl ist und floating-point-Effekte die Länge des Ausgabearrays beeinflussen. Der Abstand zwischen zwei benachbarten Werten des Ausgabearrays kann mittels des optionalen Parameters **step** gesetzt werden. Der Default-Wert für **step** ist 1.

eindim. Tensor der Länge 0: shape=()

leerer Tensor: []

eindim. Tensor der Länge 10: shape=(10,)

[1,2,3,4,5,6,7,8,9,10]

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
>>> # Nutzung von numpy zur Erzeugung von Tensoren
...
>>> a1 = tf.constant(np.arange(1,1),shape=[0])
>>> sess = tf.Session()
>>> print(sess.run(a1))
[]
>>> tf.rank(a1)
<tf.Tensor 'Rank:0' shape=() dtype=int32>
>>> tf.shape(a1)
<tf.Tensor 'Shape:0' shape=(1,) dtype=int32>
>>> a2 = tf.constant(np.arange(1,2),shape=[1])
>>> print(sess.run(a2))
[1]
>>> tf.rank(a2)
<tf.Tensor 'Rank_1:0' shape=() dtype=int32>
```

```
>>> tf.shape(a2)
<tf.Tensor 'Shape_1:0' shape=(1,) dtype=int32>
>>> a3 = tf.constant(np.arange(1,1),shape=[1])
>>> print(sess.run(a3))
[0]
>>> a4 = tf.constant(np.arange(1,11),shape=[10])
>>> print(sess.run(a4))
[ 1  2  3  4  5  6  7  8  9 10]
>>> tf.rank(a4)
<tf.Tensor 'Rank_2:0' shape=() dtype=int32>
>>> tf.shape(a4)
<tf.Tensor 'Shape_2:0' shape=(1,) dtype=int32>
>>>
```

shape=(m,n) bedeutet:

in 1.Koordinate gibt es m Einträge(Zeilenvektoren),

der Tensor hat m Zeilen

in 2.Koordinate gibt es n Einträge, jeder Zeilenvektor

hat hat n Einträge

shape=(r,m,n) bedeutet:

in 3.Koordinate gibt es r Einträge mit Matrizen vom Typ(m,n),

shape=(s,r,m,n) bedeutet:

in 4.Koordinate gibt es s Einträge mit Blockmatrizen vom Typ(r,m,n),

shape=(t,s,r,m,n) bedeutet:

in 5.Koordinate gibt es t Einträge mit Tensoren vom Typ(s,r,m,n),

usw.

np.arange(1,61) generiert Zahlen im halboffenen Intervall [1,61) mit Standardschrittweite 1, d.h. von 1 bis 60:

```
>>> a =
tf.constant(np.arange(1,61),shape=(5,4,3))
```

```
>>> print(sess.run(a))
```

erzeugt einen **Tensor als Blockmatrix** mit 5 Matrizen

vom Typ(4,3): rank=3

[A,B,C,D,E], wobei

```
A=[[ 1  2  3][ 4  5  6][ 7  8  9][10 11 12]]
```

$$=\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\\10 & 11 & 12\end{bmatrix}$$

usw., d.h.

[A,B,C,D,E]=

$$\left[\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\\10 & 11 & 12\end{bmatrix},\begin{bmatrix}13 & 14 & 15\\16 & 17 & 18\\19 & 20 & 21\\22 & 23 & 24\end{bmatrix},\begin{bmatrix}25 & 26 & 27\\28 & 29 & 30\\31 & 32 & 33\\34 & 35 & 36\end{bmatrix},\begin{bmatrix}37 & 38 & 39\\40 & 41 & 42\\43 & 44 & 45\\46 & 47 & 48\end{bmatrix},\right.$$

```
>>> a =
```

tf.constant(**np.arange**(1,241),shape=(4,5,4,3))

```
>>> print(sess.run(a))
```

Es entsteht ein Tensor, der 4 Blockmatrizen enthält:

rank=4

$$\left[\left[\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\\10 & 11 & 12\end{bmatrix},\begin{bmatrix}13 & 14 & 15\\16 & 17 & 18\\19 & 20 & 21\\22 & 23 & 24\end{bmatrix},\begin{bmatrix}25 & 26 & 27\\28 & 29 & 30\\31 & 32 & 33\\34 & 35 & 36\end{bmatrix},\begin{bmatrix}37 & 38 & 39\\40 & 41 & 42\\43 & 44 & 45\\46 & 47 & 48\end{bmatrix}\right.\right.$$

$$\left[\begin{bmatrix}61 & 62 & 63\\64 & 65 & 66\\67 & 68 & 69\\70 & 71 & 72\end{bmatrix},\begin{bmatrix}73 & 74 & 75\\76 & 77 & 78\\79 & 80 & 81\\82 & 83 & 84\end{bmatrix},\begin{bmatrix}85 & 86 & 87\\88 & 89 & 90\\91 & 92 & 93\\94 & 95 & 96\end{bmatrix},\begin{bmatrix}97 & 98 & 99\\100 & 101 & 10\\103 & 104 & 10\\106 & 107 & 10\end{bmatrix}\right.$$

$$\left[\begin{bmatrix}121 & 122 & 123\\124 & 125 & 126\\127 & 128 & 129\\130 & 131 & 132\end{bmatrix},\begin{bmatrix}133 & 134 & 135\\136 & 137 & 138\\139 & 140 & 141\\142 & 143 & 144\end{bmatrix},\begin{bmatrix}145 & 146 & 147\\148 & 149 & 150\\151 & 152 & 153\\154 & 155 & 156\end{bmatrix},\begin{bmatrix}1\\1\\1\\1\end{bmatrix}\right.$$

$$\left[\begin{bmatrix}181 & 182 & 183\\184 & 185 & 186\\187 & 188 & 189\\190 & 191 & 192\end{bmatrix},\begin{bmatrix}193 & 194 & 195\\196 & 197 & 198\\199 & 200 & 201\\202 & 203 & 204\end{bmatrix},\begin{bmatrix}205 & 206 & 207\\208 & 209 & 210\\211 & 212 & 213\\214 & 215 & 216\end{bmatrix},\begin{bmatrix}2\\2\\2\\2\end{bmatrix}\right.$$

anderes Beispiel:

Tensor als Blockmatrix mit 3 Matrizen

Tensormultiplikation:

```
>>> a = tf.constant([[[1,2,3],[4,5,6],[7,8,9]],
[[10,11,12],[13,14,15],[16,17,18]],
[[19,20,21],[22,23,24],[25,26,27]]])
>>> b = tf.constant([[[1,2,3],[4,5,6],[7,8,9]],
[[10,11,12],[13,14,15],[16,17,18]],
[[19,20,21],[22,23,24],[25,26,27]]])
>>> c = tf.matmul(a,b)
>>> print(sess.run(c))
[[[  30   36   42]
  [  66   81   96]
  [ 102  126  150]]

 [[ 435  468  501]
  [ 552  594  636]
  [ 669  720  771]]

 [[1326 1386 1446]
  [1524 1593 1662]
```

```
  [1722 1800 1878]]]

>>> a

<tf.Tensor 'Const_12:0' shape=(3, 3, 3) dtype=int32>

>>> b

<tf.Tensor 'Const_13:0' shape=(3, 3, 3) dtype=int32>
```

Wie läuft die Tensormultiplikation ab?

erste Tensor [A1,A2,A3]:

A1:=[[1,2,3],[4,5,6],[7,8,9]]

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

A2:=[[10,11,12],[13,14,15],[16,17,18]]

$$\begin{bmatrix} 10 & 11 & 12 \\ 13 & 14 & 15 \\ 16 & 17 & 18 \end{bmatrix}$$

A3:=[[19,20,21],[22,23,24],[25,26,27]]

$$\begin{bmatrix} 19 & 20 & 21 \\ 22 & 23 & 24 \\ 25 & 26 & 27 \end{bmatrix}$$

zweite Tensor [B1,B2,B3]:

B1:=[[1,2,3],[4,5,6],[7,8,9]]

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

B2:=[[10,11,12],[13,14,15],[16,17,18]]

$$\begin{bmatrix}10 & 11 & 12\\13 & 14 & 15\\16 & 17 & 18\end{bmatrix}$$

B3:=[[19,20,21],[22,23,24],[25,26,27]]

$$\begin{bmatrix}19 & 20 & 21\\22 & 23 & 24\\25 & 26 & 27\end{bmatrix}$$

$$[1\ \ 2\ \ 3]*\begin{bmatrix}1\\4\\7\end{bmatrix}$$

$$[30]$$

C1:=A1*B1

$$\begin{bmatrix}30 & 36 & 42\\66 & 81 & 96\\102 & 126 & 150\end{bmatrix}$$

C2:=A2*B2

$$\begin{bmatrix}435 & 468 & 501\\552 & 594 & 636\\669 & 720 & 771\end{bmatrix}$$

C3:=A3*B3

$$\begin{bmatrix}1326 & 1386 & 1446\\1524 & 1593 & 1662\\1722 & 1800 & 1878\end{bmatrix}$$

Ergebnis: Tensor als Blockmatrix: [C1,C2,C3]

Es wurde eine Art "Vektoprodukt"

[A1,A2,A3]◦[B1,B2,B3]=[C1,C2,C3] gebildet:

$$\left[\begin{bmatrix}30 & 36 & 42\\66 & 81 & 96\\102 & 126 & 150\end{bmatrix},\begin{bmatrix}435 & 468 & 501\\552 & 594 & 636\\669 & 720 & 771\end{bmatrix},\begin{bmatrix}1326 & 1386 & 1446\\1524 & 1593 & 1662\\1722 & 1800 & 1878\end{bmatrix}\right]$$

Die bekannte **algebraische Struktur eines "Vektorraumes"** muss hinsichtlich der Tensorrechnung innerhalb der Tensorflow-Software neu betrachtet werden:

z.B. Syntax von **tf.linspace** und **tf.fill**

https://www.python-kurs.eu/numpy_arrays_erzeugen.p

https://www.tensorflow.org/api_docs/python/tf/fill

==========

```
>>> vec_a = tf.linspace(0.,3.,4)
>>> vec_b = tf.fill([4,1],2.)
>>> print(sess.run(vec_a))
[0.  1.  2.  3.]
>>> print(sess.run(vec_b))
[[2.]
 [2.]
 [2.]
 [2.]]
>>> prod = tf.multiply(vec_a,vec_b)
>>> print(sess.run(prod))
[[0.  2.  4.  6.]
 [0.  2.  4.  6.]
 [0.  2.  4.  6.]
 [0.  2.  4.  6.]]
>>> dot = tf.tensordot(vec_a,vec_b,1)
>>> print(sess.run(dot))
[[12]]
>>> prod1 = tf.multiply(vec_b,vec_a)
>>> print(sess.run(prod1))
```

```
[[0. 2. 4. 6.]
 [0. 2. 4. 6.]
 [0. 2. 4. 6.]
 [0. 2. 4. 6.]]
====================================
# Die Tensormultiplikation von Tensoren mit shape=(n,)
oder shape=(n, 1) ist kommutativ, die
Vektormultiplikation(Matrixmultiplikation) nicht!
====================================
```

Download für dieses Dokument:

www.informatik.htw-dresden.de/
~paditz/Tensorflow-Ue03.pdf