Data visualization

COSC 480B

Reyan Ahmed

rahmed1@colgate.edu

Lecture 17

Tensorflow essentials

Overview

- Understanding the TensorFlow workflow
- Creating interactive notebooks with Jupyter
- Visualizing algorithms by using TensorBoard

Overview

Computing the inner product of two vectors without using a library

```
revenue = 0
for price, amount in zip(prices, amounts):
    revenue += price * amount
```

Computing the inner product using NumPy

```
import numpy as np
revenue = np.dot(prices, amounts)
```

Ensuring that TensorFlow works

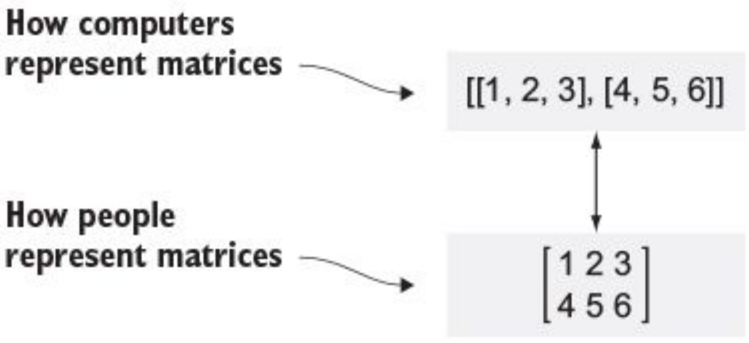
Once you install tensorflow in your machine, you should be able to import it:

```
import tensorflow as tf
```

Detailed documentation about various functions for the Python and C++ APIs are available at www.tensorflow.org/api_docs/.

Representing tensors

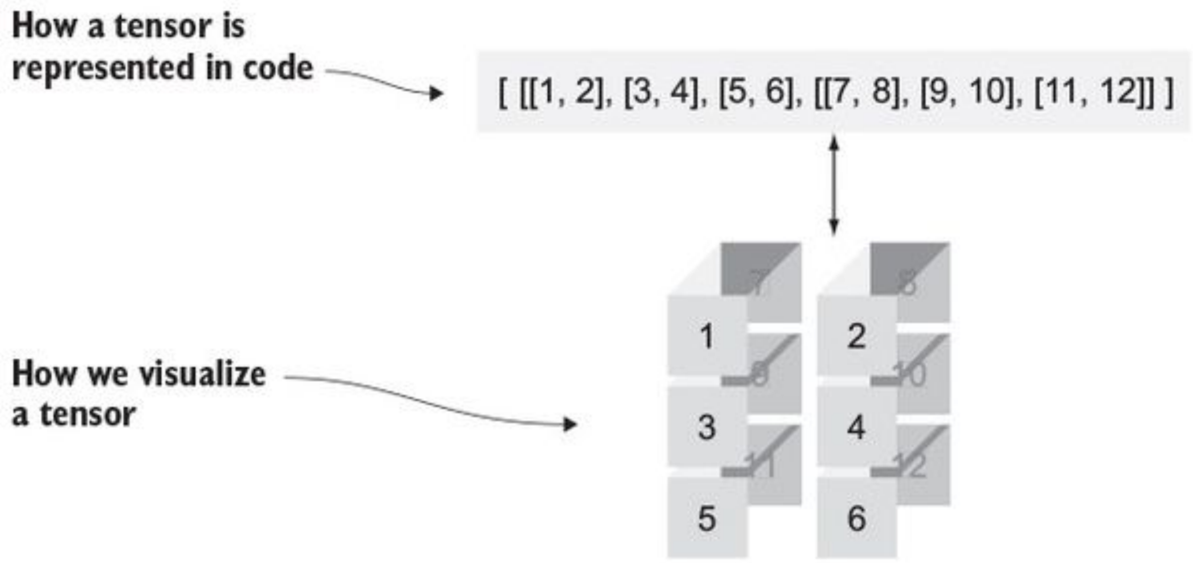
The matrix in the lower half of the diagram is a visualization from its compact code notation in the upper half of the diagram. This form of notation is a common paradigm in most scientific computing libraries.



A tensor is a generalization of a matrix that specifies an element by an arbitrary number of indices.

Representing tensors

This tensor can be thought of as multiple matrices stacked on top of each other. To specify an element, you must indicate the row and column, as well as which matrix is being accessed. Therefore, the rank of this tensor is 3.



Representing tensors

Different ways to represent tensors

```
import tensorflow as tf
import numpy as np                                   1

m1 = [[1.0, 2.0],
      [3.0, 4.0]]                                2

m2 = np.array([[1.0, 2.0],
              [3.0, 4.0]], dtype=np.float32)      2

m3 = tf.constant([[1.0, 2.0],
                 [3.0, 4.0]])                  2

print(type(m1))                                 3
print(type(m2))                                 3
print(type(m3))                                 3

t1 = tf.convert_to_tensor(m1, dtype=tf.float32)    4
t2 = tf.convert_to_tensor(m2, dtype=tf.float32)    4
t3 = tf.convert_to_tensor(m3, dtype=tf.float32)    4

print(type(t1))                                 5
print(type(t2))                                 5
print(type(t3))                                 5
```

1 You'll use NumPy matrices in TensorFlow.
2 Defines a 2 × 2 matrix in three ways
3 Prints the type for each matrix
4 Creates tensor objects out of the various types
5 Notice that the types will be the same now.
The code outputs the following three times:

```
<class 'tensorflow.python.framework.ops.Tensor'>
```

Representing tensors

Creating tensors

- 1 Defines a 2 × 1 matrix
- 2 Defines a 1 × 2 matrix
- 3 Defines a rank-3 tensor
- 4 Try printing the tensors.

```
import tensorflow as tf

m1 = tf.constant([[1., 2.]])       1

m2 = tf.constant([[1],
                  [2]])           2

m3 = tf.constant([ [[1,2],
                    [3,4],
                    [5,6]],
                    [[7,8],
                    [9,10],
                    [11,12]] ])    3

print(m1)                          4
print(m2)                          4
print(m3)                          4
```

Representing tensors

The code produces the following output:

```
Tensor( "Const:0",
        shape=TensorShape([Dimension(1), Dimension(2)]),
        dtype=float32 )
Tensor( "Const_1:0",
        shape=TensorShape([Dimension(2), Dimension(1)]),
        dtype=int32 )
Tensor( "Const_2:0",
        shape=TensorShape([Dimension(2), Dimension(3),
Dimension(2)]),
        dtype=int32 )
```

Representing tensors

Exercise 1

Initialize a 500×500 tensor with all elements equaling 0.5.

Representing tensors

Exercise 1

Initialize a 500 × 500 tensor with all elements equaling 0.5.

```
tf.ones([500,500]) * 0.5
```

Creating operators

Using the negation operator

```
import tensorflow as tf

x = tf.constant([[1, 2]])        1
negMatrix = tf.negative(x)       2
print(negMatrix)                 3
```

- 1 Defines an arbitrary tensor
- 2 Negates the tensor
- 3 Prints the object, generates the following output:

```
Tensor("Neg:0", shape=TensorShape([Dimension(1), Dimension(2)]), dtype=int32)
```

Creating operators

Useful TensorFlow operators:

- tf.add(x, y)—Adds two tensors of the same type, x + y
- tf.subtract(x, y)—Subtracts tensors of the same type, x – y
- tf.multiply(x, y)—Multiplies two tensors element-wise
- tf.pow(x, y)—Takes the element-wise x to the power of y
- tf.exp(x)—Equivalent to pow(e, x), where e is Euler's number (2.718 ...)
- tf.sqrt(x)—Equivalent to pow(x, 0.5)
- tf.div(x, y)—Takes the element-wise division of x and y
- tf.truediv(x, y)—Same as tf.div, except casts the arguments as a float
- tf.floordiv(x, y)—Same as truediv, except rounds down the final answer into an integer
- tf.mod(x, y)—Takes the element-wise remainder from division

Creating operators

Exercise 2

Use the TensorFlow operators you’ve learned so far to produce the Gaussian distribution (also known as the normal distribution).

Creating operators

Answer:

```
from math import pi
mean = 0.0
sigma = 1.0
(tf.exp(tf.negative(tf.pow(x – mean, 2.0) /
               (2.0 * tf.pow(sigma, 2.0) ))) *
 (1.0 / (sigma * tf.sqrt(2.0 * pi) )))
```

Executing operators with sessions

Using a session

```
import tensorflow as tf

x = tf.constant([[1., 2.]])          1
neg_op = tf.negative(x)              2

with tf.Session() as sess:           3
    result = sess.run(neg_op)        4
print(result)                        5
```

- 1 Defines an arbitrary matrix
- 2 Runs the negation operator on it
- 3 Starts a session to be able to run operations
- 4 Tells the session to evaluate neg_op
- 5 Prints the resulting matrix

Executing operators with sessions

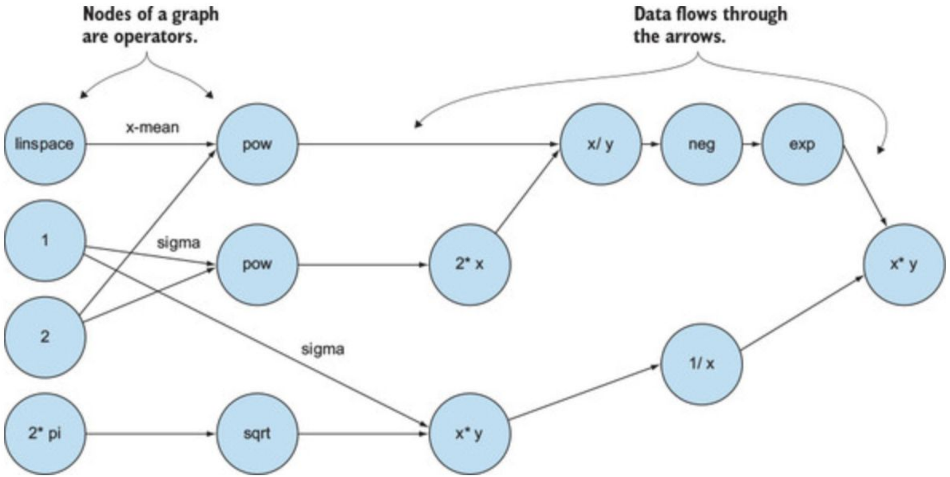
Using the interactive session mode

```
import tensorflow as tf
sess = tf.InteractiveSession()        1

x = tf.constant([[1., 2.]])           2
negMatrix = tf.negative(x)            2

result = negMatrix.eval()             3
print(result)                         4

sess.close()                          5
```

1 Starts an interactive session so the sess variable no longer needs to be passed around
2 Defines an arbitrary matrix and negates it
3 You can now evaluate negMatrix without explicitly specifying a session.
4 Prints the negated matrix
5 Remember to close the session to free up resources.

Executing operators with sessions

The graph represents the operations needed to produce a Gaussian distribution. The links between the nodes represent how data flows from one operation to the next. The operations themselves are simple, but the complexity arises from the way they intertwine.



Executing operators with sessions

Logging a session

```
import tensorflow as tf

x = tf.constant([[1., 2.]])                                          1
negMatrix = tf.negative(x)                                           1

with tf.Session(config=tf.ConfigProto(log_device_placement=True)) as sess: 2
    result = sess.run(negMatrix)                                     3

print(result)                                                        4
```

1 Defines a matrix and negates it
2 Starts the session with a special config passed into the constructor to enable logging
3 Evaluates negMatrix
4 Prints the resulting value

```
Neg: /job:localhost/replica:0/task:0/cpu:0
```

Executing operators with sessions

Here’s a quick overview of these three types of values:

- Placeholder—A value that’s unassigned but will be initialized by the session wherever it’s run. Typically, placeholders are the input and output of your model.
- Variable—A value that can change, such as parameters of a machine-learning model. Variables must be initialized by the session before they’re used.
- Constant—A value that doesn’t change, such as hyperparameters or settings.

Executing operators with sessions

The session dictates how the hardware will be used to process the graph most efficiently. When the session starts, it assigns the CPU and GPU devices to each of the nodes. After processing, the session outputs data in a usable format, such as a NumPy array. A session optionally may be fed placeholders, variables, and constants.