Learning tensorflow

Mohd Zamri Murah

Center for Artificial Intelligence Technology
Fakulti Teknologi Sains Maklumat
Universiti Kebangsaan Malaysia
zamri@ukm.edu.my

Abstract. Tensorflow is an open-source deep learning library developed by Google. It has been used in many areas such as image recognition, text to speech engine, pattern recognition and big data. This note provide an introductory concepts for computation using tensorflow.

Keywords: deep learning

1 Introduction

Tensorflow is an open-source deep learning library developed by Google[1]. It is based on two basic ideas; computational graph and tensor.

A basic computational graph is shown in figure 1. The figure illustrates the two elements of computational graph; nodes and edges. Nodes typically drawn as circles to represent some sort of computation or action being done on data. Edges are the actual values that get passed to and from nodes, and are typically drawn as arrows. Thus, in figure 1, we have a node for computation *add* and edge 1 and edge 2 into the node *add* and, edge 3 as edge output from the node *add*.

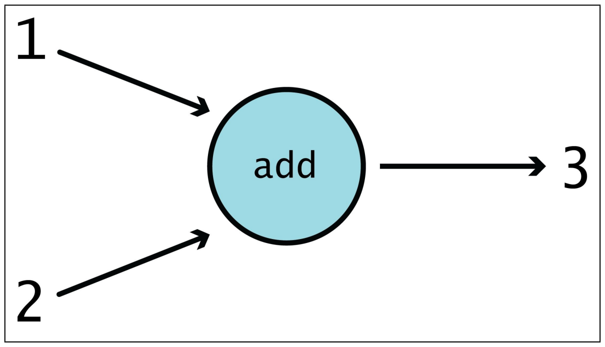

Fig. 1. A basic computational graph with node and edge.

In figure 2, we have a more complex computation. In figure 2, we have nodes *input*, *input*, *add*,*mult* and *add*. In total, we have 5 nodes. Also, we have 9 edges; $5, 3, 5, 5, 3, 3, 15, 8, 23$. Succintly, we could write the model as; $N = \{input, add, mult, add\}$ and $E = \{5, 3, 5, 5, 3, 3, 15, 8, 23\}$.

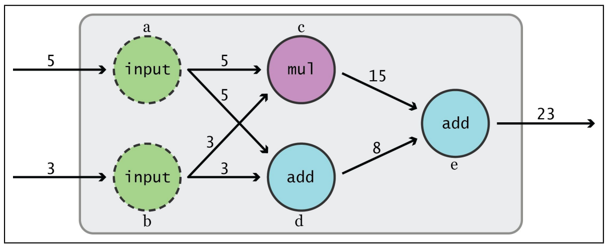

Fig. 2. A more complex computational graph with nodes and edges.

We can also have a more complicated model as figure 3.

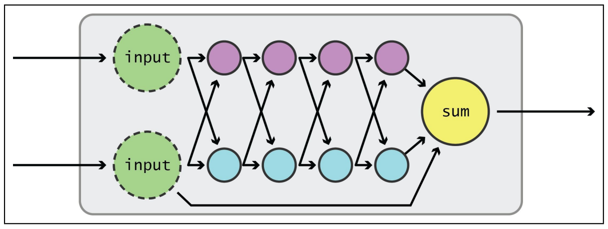

Fig. 3. A edge can link to any node.

2 Computational graph

Tensorflow computation is based on computational graph. In order to model in tensorflow, we need to;

1. define model in computational graph
2. run model

In listing 1, we provide the code for a tensor model based on our previous discussion.

Listing 1. an example of tensorflow model

```
import  tensorflow as tf
a =  tf.constant(5, name="input_a")
b =  tf.constant(3, name="input_b")
c =  tf.mul(a,b, name="mul_c")
d =  tf.add(a,b, name="add_d")
```

```
e = tf.add(c,d, name="add_e")
sess = tf.Session()
output = sess.run(e)
writer = tf.train.SummaryWriter('./my_graph', sess.graph)
writer.close()
sess.close()
```

If we run tensorboard, we have the figure 4.

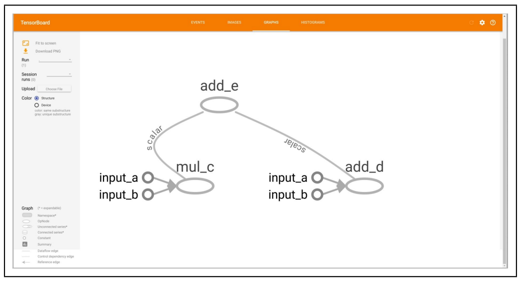

Fig. 4. A graphical view of the tensorflow model using tensorboard.

3 Tensor

Tensorflow data model is based on tensor. Tensor is an n-dimensional abstraction of matrices. We have scalar $\eta = [5]$ equivalent to 0-D tensor, a vector $\vec{v} = [5, 3]$ equivalent to 1-D tensor and a matrix $A_{m,n}$ is a 2-D tensor[2][3].

Two vectors $\vec{v}$ and $\vec{w}$ can be combined via inner product to form a new scalar η. Two vectors $\vec{v}$ and $\vec{w}$ can be combined via cross product to form a new vector $\vec{z}$. Thus we have the following generalization;

1. η scalar. tensor of rank 0 with $3^0 = 1$ component. magnitude only.
2. $vecv$ vector. tensor of rank 1 with $3^1 = 3$ components. magnitude and one direction.
3. A dyad. tensor of rank 2 with $3^2 = 9$ components. magnitude and 2 directions.

Quantities with no sense of direction are scalars η, these are defined as only real numbers in any system of units such as temperature. Scalr have $1 = 3^0$ element. The vectors $\vec{v}$ on the other hand are associated with direction, force for example. The number of elements are $3 = 3^1$. The second order tensor is a quantity with which two directions seem to be associated. The stress acting on an element of fluid in a 3D Cartesian system acts on planes xx, yy, zz, xy, zx and so on a total of $9 = 3^2$ elements define the stress system of the element. Strain rate tensor is another example.

4 Model in tensor

In figure 5, we have the same model in tensor notation.

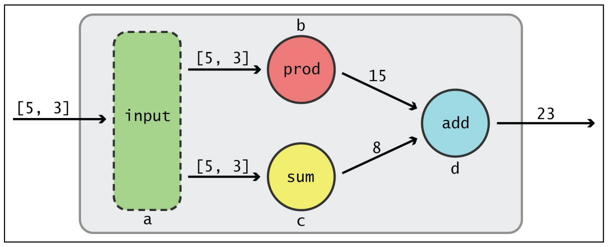

Fig. 5. A graphical view of the tensorflow model using tensorboard.

Listing 2. model in tensor

```python
import tensorflow as tf
a = tf.constant([5, 3, 9], name="input_a")
b = tf.reduce_prod(a, name="prod_b")
c = tf.reduce_sum(a, name="sum_c")
d = tf.add(b, c, name="add_d")

sess = tf.Session()
sess.run(d)

print(sess.run(d))
output = sess.run(d)
```

We could examine the shape of a tensor using $tf.shape$.

Listing 3. shape of tensor

```python
tf.shape(a) # <tf.Tensor 'Shape:0' shape=(1,) dtype=int32>
tf.shape(b) # <tf.Tensor 'Shape_1:0' shape=(0,) dtype=int32>
tf.shape(c) # <tf.Tensor 'Shape_2:0' shape=(0,) dtype=int32>
tf.shape(d) # <tf.Tensor 'Shape_3:0' shape=(0,) dtype=int32>
```

Tensors are just a superset of matrices!.

5 Tensors operations

Nodes in tensorflow are operations on tensors. Nodes perform operation on or with tensor objects, and output tensors objects such as scalar or another tensor. These outputs can be use as input by other nodes in the computational graph. An example of tensor operation is shown in listing 4.

Listing 4. operations on tensors

```
#Initialize some tensors to use in computation
a = np.array([2, 3], dtype=np.int32) # tensor a
b =  np.array([4, 5],  dtype=np.int32) # tensor b
#Use tf.add() to initialize an "add" Operation
#The variable ‘c’ will be a handle to the Tensor output of this Op
c = tf.add(a, b)
```

6 Using variables in tensor models

Variables in tensor models are called *placeholders*. have their values specified when created.

Listing 5. use placeholders in tensor

```
import tensorflow as tf
import numpy as np
# Creates a placeholder vector of length 2 with data type int32
a = tf.placeholder(tf.int32, shape = [2],name = "my_input")
# Use the placeholder as if it were any other Tensor object
b = tf.reduce_prod(a, name = "prod_b")
c = tf.reduce_sum(a, name = "sum_c")
# Finish off the graph
d = tf.add(b, c, name = "add_d")

#Open a TensorFlow Session
sess = tf.Session()

#Create a dictionary to pass into‘feed_dict‘
#Key: ‘a‘, the handle to the placeholder output Tensor
#Value: A vector with value[5, 3] and int32 data type
input_dict = { a: np.array([5, 3], dtype = np.int32)}
#Fetch the value of ‘d‘, feeding the values of ‘input_vector‘ into ‘a‘
sess.run(d, feed_dict = input_dict)
```

Tensors (e.g $a, b, c., d$ in above model) and *operation* (e.g node such $add, mult, reduce_prod$) objects are immutable (cannot be changed). $Variables$ objects contain mutable tensor values that are persistent across sessions.

Listing 6. use placeholders in tensor

```
import tensorflow as tf
#Pass in a starting value of three for the variable
my_var = tf.Variable(3, name="my_variable")

a = tf.add(5, my_var)
b =  tf.mul(8,  my_var)
```

The initial value of Variables will often be large tensors of zeros, ones, or random values.

Listing 7. use placeholders in tensor

```
#2x2 matrix of zeros
zeros = tf.zeros([2, 2])
# vector of length 6 of ones
ones = tf.ones([6])
#3x3x3  Tensor of random uniform values between 0 and 10
uniform = tf.random_uniform([3, 3, 3], minval=0, maxval=10)
#3x3x3  Tensor of normally distributed numbers; mean 0 and standard
    deviation 2
normal = tf.random_normal([3, 3, 3], mean=0.0, stddev=2.0)

sess = tf.Session()
sess.run([zeros,ones, uniform, normal])

# proper way to initialize
#init = tf.initialize_all_variables()
#sess = tf.Session()
#sess.run(init)
```

7 Machine learning in tensor

Supervised learning concerns with inference models where we have input data dan output results. The objective is to find inference models with the minimun loss functions. This is done through training. The inference models can be used for predictions of new input data. The basic training loop is given on figure 6.

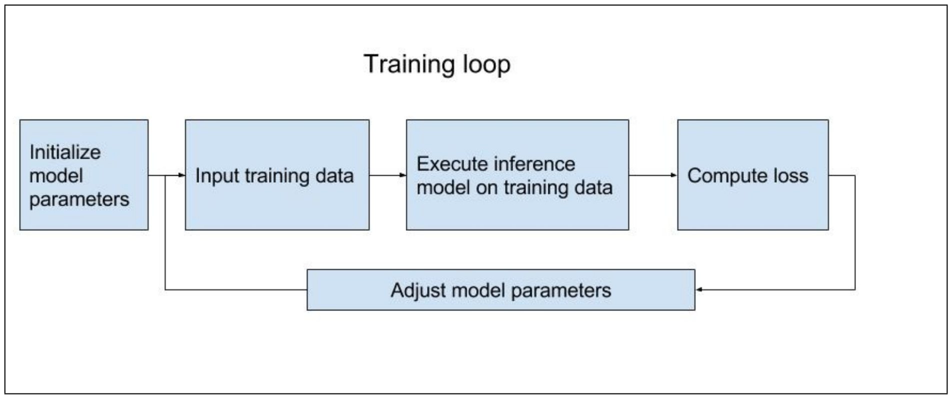
Training loop
Initialize model parameters
Input training data
Execute inference model on training data
Compute loss
Adjust model parameters

Fig. 6. A basic training loop.

8 Linear regression in tensor

In linear regression, we have the standard simple linear regression equation as $y = ax + b$ or in tensor format we have $Y = XW^t + b$ where W is weight and b is bias.

The $L2$ norm loss function is defined as $L_2 = \sum_i (y_i - \hat{y}_i)^2$. The objective of the training is to minimize this L_2 norm loss function.

Listing 8. linear regression in tensor

```
import tensorflow as tf

# initialize variables / model parameters
W = tf.Variable(tf.zeros([2, 1]), name="weights")
# array([[0.],
#        [0.]], dtype=float32)
b = tf.Variable(0., name="bias")

def inference(X):
    return tf.matmul(X, W) + b
    # compute inference model over data X and return the result

def loss(X, Y):
    # compute loss over training data X and expected outputs Y
    Y_predicted = inference(X)
    return tf.reduce_sum(tf.squared_difference(Y, Y_predicted))

def inputs():
    # read / generate input training data X and expected outputs Y
    weight_age = [[84, 46], [73, 20], [65, 52], [70, 30], [76, 57],
                  [69, 25], [63, 28], [72, 36], [79 , 57],[75, 44]]
    blood_fat_content = [354, 190, 405, 263, 451, 302, 288, 385,
                         402, 365]
    return tf.to_float(weight_age), tf.to_float(blood_fat_content)

def train(total_loss):
    # train / adjust model parameters according to computed total loss
    learning_rate = 0.000001
    return
        tf.train.GradientDescentOptimizer(learning_rate).minimize(total_loss)

def evaluate(sess, X, Y):
    # evaluate the resulting trained model# Launch the graph in
    # a session, setup boilerplate
    print(sess.run(inference([[80., 25.]])))
    print(sess.run(inference([[65., 25.]])))

# saver = tf.train.Saver()
```

```
with tf.Session() as sess:
    tf.initialize_all_variables().run()
    X, Y = inputs()
    total_loss = loss(X, Y)
    train_op = train(total_loss)
    coord = tf.train.Coordinator()
    threads = tf.train.start_queue_runners(sess = sess,
        coord = coord)
    # actual training loop
    training_steps = 500000
    for step in range(training_steps):
        sess.run([train_op])
        # for debugging and learning purposes,
        # see how the loss gets decremented thru
        # training steps
        if step % 10000 == 0:
            print("loss:", sess.run([total_loss]))
        #if step % 500 == 0:
        #    saver.save(sess, 'my-model',
        #             global_step=training_steps)

    evaluate(sess, X, Y)
    coord.request_stop()
    coord.join(threads)
    sess.close()
```

9 Logistic regression in tensor

Linear regression model predicts a continous values. Logistic regression predict probability of an input belongs to a particular class.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Listing 9. logistic regression in tensor

```
import tensorflow as tf
import os
# modified
def read_csv(batch_size, file_name, record_defaults):
    #filename_queue =
        tf.train.string_input_producer([os.path.dirname(__file__) + "/"
        + file_name])
    filename_queue = tf.train.string_input_producer([file_name])
    reader = tf.TextLineReader(skip_header_lines=1)
    key, value = reader.read(filename_queue)
    #key, value = reader.read(file_name)
    # decode_csv will convert a Tensor from type string (the text line)
        in
    # a tuple of tensor columns with the specified defaults, which also
    # sets the data type for each column
    decoded = tf.decode_csv(value, record_defaults=record_defaults)
    # batch actually reads the file and loads "batch_size" rows in a
        single tensor
    return tf.train.shuffle_batch(decoded,
               batch_size=batch_size,
               capacity=batch_size * 50,
               min_after_dequeue=5)


def combine_inputs(X):
    return tf.matmul(X, W) + b


def loss(X, Y):
    # compute loss over training data X and expected outputs Y
    #Y_predicted = inference(X)
    #return tf.reduce_sum(tf.squared_difference(Y, Y_predicted))
    return
        tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(combine_inputs(X),
        Y))

def train(total_loss):
    # train / adjust model parameters according to computed total loss
    learning_rate = 0.01
    print("in train")
    return
        tf.train.GradientDescentOptimizer(learning_rate).minimize(total_loss)

W = tf.Variable(tf.zeros([5, 1]), name="weights")
b = tf.Variable(0., name="bias")


# all tf variable must be declare in session!

with tf.Session() as sess:
    init = tf.global_variables_initializer()
    sess.run(init)
    v0 = sess.run(W)
    print(v0) # will show you your variable.
    v1 = sess.run(b)
    print(v1)

    #v2 = sess.run(pid)
    #print(v2)

   # this all tensors
```

```
pid, survived, pclass, sex, age = read_csv(100, "train.csv", [[0.0],
    [0.0], [0],[""],[0.0]])

first_class = tf.to_float(tf.equal(pclass,[1]))
second_class = tf.to_float(tf.equal(pclass, [2]))
third_class = tf.to_float(tf.equal(pclass, [3]))
gender = tf.to_float(tf.equal(sex, ["female"]))

features = tf.transpose(tf.pack([first_class, second_class,
    third_class, gender, age]))
survived = tf.reshape(survived, [100, 1])

X, Y = features, survived

print("done X Y")
total_loss = loss(X, Y)
print("done total loss")
train_op = train(total_loss)
coord = tf.train.Coordinator()
print("after coord")
threads = tf.train.start_queue_runners(sess = sess,coord = coord)
training_steps = 5000
print("after threads")
for step in range(training_steps):
   sess.run([train_op])
   if step % 100 == 0: print("loss:", step, sess.run([total_loss]))
coord.request_stop()
coord.join(threads)
sess.close()
```

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow. org **1** (2015)
2. Bowen, R.M., Wang, C.C.: Introduction to vectors and tensors. Volume 2. Courier Corporation (2008)
3. Kolecki, J.C.: An introduction to tensors for students of physics and engineering. Technical Report NASA/TM2002-211716, NASA Center for Aerospace Information (2002)