TensorFlow Tutorial

SUNIL KUMAR SAHU

Deep Learning Libraries

- Caffe : Python, Matlab, C++
- CNTK : Python, C++
- Theano : Python
- Torch : Lua
- Keras : Python and Theano
- Lasagne : Python and Theano
- TensorFlow : Python, C++
- etc.

Deep Learning Libraries

- Caffe
- CNTK

Configuration File

- Theano
- Torch
- Keras
- Lasagne
- TensorFlow
- etc.

Programmatic generation

TensorFlow

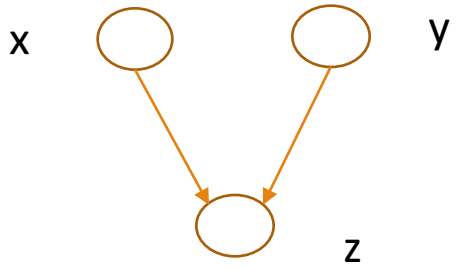
- Developed by *Google* as part of *Google Brain* project
- Large number of inbuilt functions available and keep getting update
- Pretty good documentations available and getting update
- Large number of users : You can get lots of code in github
- Better support for distributed system

Summing two matrices

```
>>> import numpy as np
>>> x = np.zeros((3,2))
>>> y = np.ones((3,2))
>>> z = x + y
>>> z
array([[ 1.,  1.],
       [ 1.,  1.],
       [ 1.,  1.]])
...
```

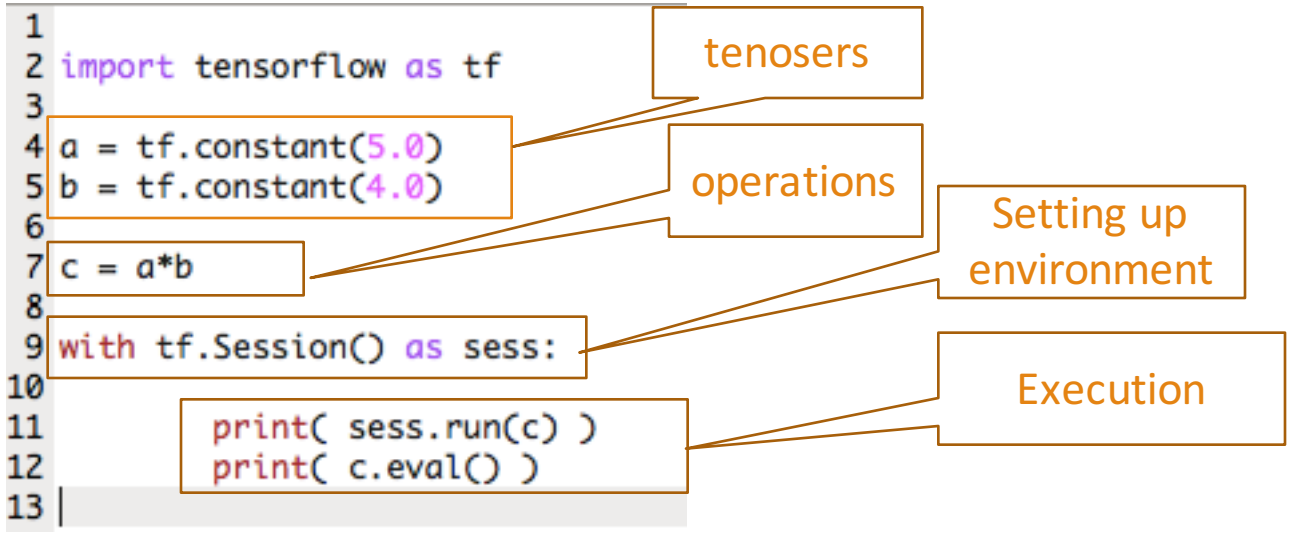
```
>>> import tensorflow as tf
>>> x = tf.zeros((3,2))
>>> y = tf.ones((3,2))
>>> z = x + y
>>> tf.InteractiveSession()
<tensorflow.python.client.session.InteractiveSession object at 0x11352aa90>
>>> z.eval()
array([[ 1.,  1.],
       [ 1.,  1.],
       [ 1.,  1.]], dtype=float32)
>>>
```

Computation Graph



Flow of Writing Codes

1. Create tensors
2. Write operations for tensors
3. Setting up environment for those operations to execute
4. Execute operations in given environment



Tensors in TensorFlow

1. **Constant tensor :**
2. **Variable tensor** : initialize with init_op, restoring with save file, assign() works

```
1 import tensorflow as tf
2 import numpy as np
3
4 W1 = tf.convert_to_tensor(np.random.rand(4,3) )
5 W2 = tf.Variable(tf.ones((3,5), dtype='float64', name = 'weights') )
6
7 A = tf.matmul(W1, W2)
8
9 with tf.Session() as sess:
10         print (sess.run(W1) )
11
12         sess.run( tf.initialize_all_variables() )
13
14         print (sess.run(W2) )
15
16         print (sess.run(A))
17
18
```

Constant tensor

Variable tensor

Initialize all variable tensor

Place Holder

- Kind of a Variable or dummy node in computation graph
- We can write sequence of operations with place holder
- Real assignment happens when we call run()
- Syntax :

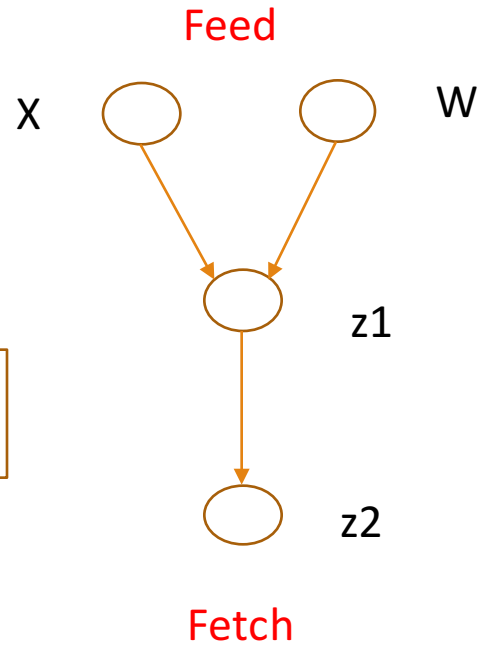
```
X = tf.placeholder(tf.float32, [3, None], name='X' )

X = tf.placeholder(tf.float32, [None, 3], name='X' )

X = tf.placeholder(tf.float32, [None, None], name='X' )
```

Fetching and Feeding

```
1 import tensorflow as tf
2 import numpy as np
3
4 X = tf.placeholder(tf.float32, [None, 3], name='X' )
5
6 W = tf.Variable( tf.random_uniform( [3,2], -1.0, +1.0) , name = 'W' )
7
8 z1 = tf.matmul(X, W)
9 z2 = tf.nn.sigmoid(z1)
10
11 with tf.Session() as sess:
12         sess.run( tf.initialize_all_variables() )
13         a, b = sess.run([z1,z2], {X:np.ones((4,3))} )
14         print a
15         print b
```

Feed data into computation graph

Fetch values from computation graph



Computation Graph

Save and Restore

```python
import tensorflow as tf
import numpy as np

x = tf.placeholder(tf.float32, [2,2], name = 'x')

w1 = tf.Variable(tf.random_uniform([2,3]), name='w1')
w2 = tf.Variable(tf.random_uniform([3,1]), name='w1')

z1 = tf.matmul(x,w1)    # (2,3)
z2 = tf.matmul(z1,w2)   # (2,1)

with tf.Session() as sess :
        sess.run(tf.initialize_all_variables())
        saver = tf.train.Saver()
        for i in range(10):
                print sess.run( z2, {x:np.random.rand(2,2)} )
                path = saver.save(sess, 'model/my-model.ckpt')
                print path
```

```python
import tensorflow as tf
import numpy as np

x = tf.placeholder(tf.float32, [2,2], name = 'x')

w1 = tf.Variable(tf.random_uniform([2,3]), name='w1')
w2 = tf.Variable(tf.random_uniform([3,1]), name='w1')

z1 = tf.matmul(x,w1)    # (2,3)
z2 = tf.matmul(z1,w2)   # (2,1)

with tf.Session() as sess :
        saver = tf.train.Saver()
        saver.restore(sess, "model/my-model.ckpt")
        for i in range(10):
                print sess.run( z2, {x:np.random.rand(2,2)} )
```

Important Operations

```
 8 |
 9
10          # Concatenation
11 emb0 = tf.constant( np.random.rand(3,4) )        # 3X4
12 emb1 = tf.constant( np.random.rand(3,3) )        # 3X3
13 X = tf.concat(1, [emb0, emb2])                   # 3X7
14
15          # Operations wrt one index
16 X = tf.constant( np.ones((3,4)) )                # 3X4
17 y = tf.reduce_sum( X, 1 )                        # Output: [4,4,4]
18 z = tf.reduce_max(X, 1)                          # Output: [1,1,1]
19 a = tf.reduce_mean(X, 1)                         # Output: [1,1,1]
20 b = tf.argmax(X, 1)                              # Output: [0,0,0]
21
22
23          # Reshape the tensor
24 x = tf.constant( np.random.rand(3,4,5) )        # 3X4X5
25 y = tf.reshape( x, [3,20] )                      # 3X20
26 z = tf.reshape( y, [-1] )                       # 60
27 a = tf.reshape( z, [-1, 5, 2] )                 # 6X5X2
28 b = tf.expand_dims(a, -1)                       # 6X5X2X1
29 c = tf.squeeze(b)                               # 6X5X2
30
31
```

Part-2

IMPLEMENTATIONS

Multi Layer Neural Network (1)

nnEx.py

```
1  import tensorflow as tf
2
3  i_dim = 10
4  h1_dim = 5
5  num_classes = 3
6  #Symbolic or Place holder for input and output
7  X = tf.placeholder(tf.float32, [None, i_dim], name='input')                    # 100 X 10
8  input_y = tf.placeholder(tf.float32, [None, num_classes], name="input_y")      # 100 X 3
9  #Initialize parameters
10 W1 = tf.get_variable('W_1', shape = [i_dim, h1_dim], initializer = tf.random_normal_initializer() )     # 10 X 5
11 b1 = tf.Variable(tf.constant(0.1, shape=[h1_dim]), name="b1")                                           # 5
12 W2 = tf.get_variable('W_2', shape = [h1_dim, num_classes], initializer = tf.random_normal_initializer() )      # 5X3
13 b2 = tf.Variable(tf.constant(0.1, shape=[num_classes]), name="b2")                                      # 3
14
15 #First layer operations
16 H1 = tf.nn.xw_plus_b(X, W1, b1, name="H2")      # 100 X 5
17 Z1 = tf.sigmoid(H1)                             # 100 X 5
18 #Second layer operations
19 H2 = tf.nn.xw_plus_b(H1, W2, b2, name="H2")     # 100 X 3
20 #Loss function
21 losses = tf.nn.softmax_cross_entropy_with_logits(H2, input_y)   # 100
22 loss = tf.reduce_mean(losses) + 0.001 * (tf.nn.l2_loss(W1) + tf.nn.l2_loss(W2) )               # 1
23 #Predicitons of the batch
24 predictions = tf.argmax(H2, 1, name="predictions")
25 #Accuracy of correct prediction in batch
26 correct_predictions = tf.equal(predictions, tf.argmax(input_y, 1) )
27 accuracy = tf.reduce_mean(tf.cast(correct_predictions, "float"), name="accuracy")
28 #Optimization
29 optimizer = tf.train.AdamOptimizer(1e-2)
30 grads_and_vars = optimizer.compute_gradients(loss)
31 global_step = tf.Variable(0, name="global_step", trainable=False)
32 train_op = optimizer.apply_gradients(grads_and_vars, global_step = global_step)
33
34 with tf.Session() as sess :
35
36         sess.run(tf.initialize_all_variables())
37         # Create Dataset
38         D = np.asarray( np.random.rand(1000, 10), dtype='float32')
39         Y = np.asarray ( np.zeros((1000, 3)), dtype='float32' )
40         for i in range(1000) :
41                 k = np.random.randint(3)
42                 Y[i][k] = 1.0
43         X_train = D[0:800]; Y_train = Y[0:800]
44         X_test = D[800:] ; Y_test = Y[800:]
45         # Training
```

Create Tensors

Define Operations

Environment Setup

Execution

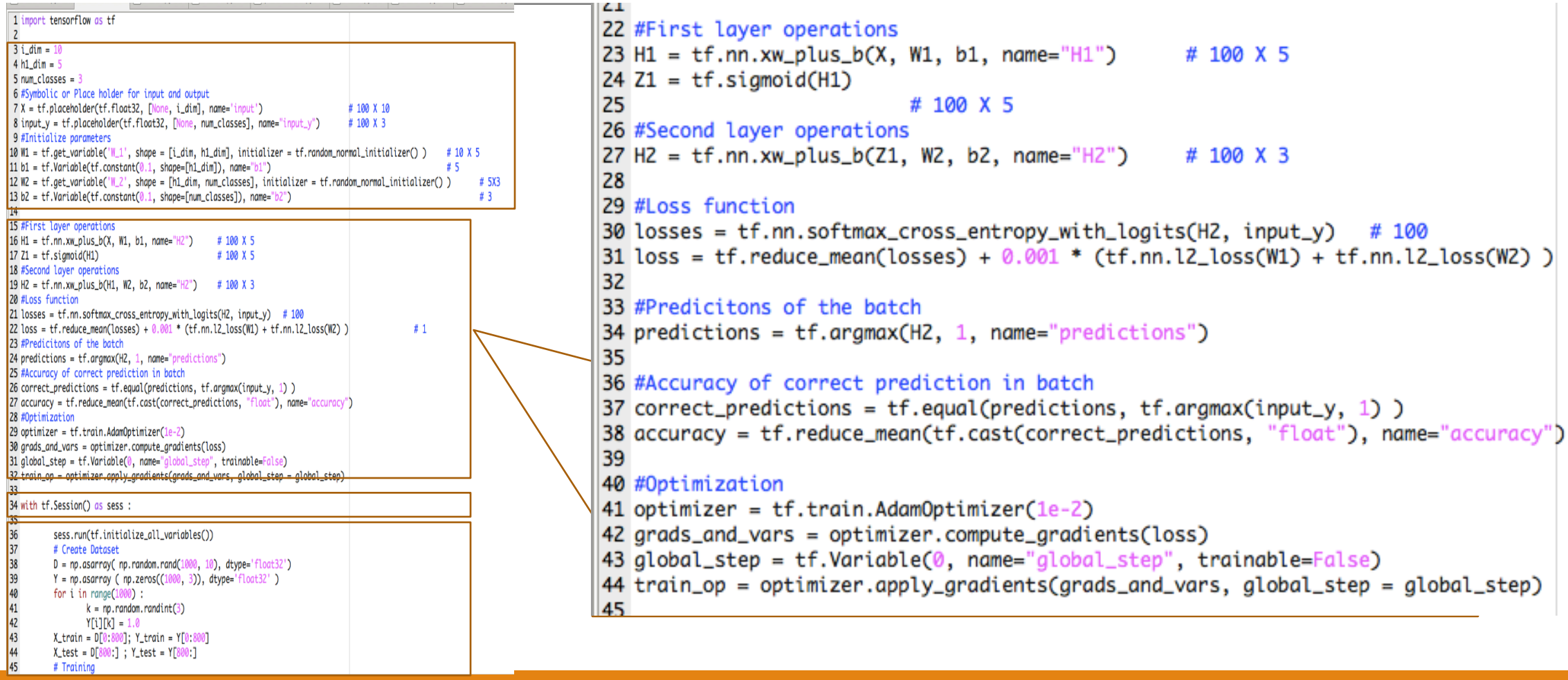
Multi Layer Neural Network (2)

nnEx.py

```
import tensorflow as tf

i_dim = 10
h1_dim = 5
num_classes = 3
#Symbolic or Place holder for input and output
X = tf.placeholder(tf.float32, [None, i_dim], name='input')                  # 100 X 10
input_y = tf.placeholder(tf.float32, [None, num_classes], name="input_y")       # 100 X 3
#Initialize parameters
W1 = tf.get_variable('W_1', shape = [i_dim, h1_dim], initializer = tf.random_normal_initializer() )    # 10 X 5
b1 = tf.Variable(tf.constant(0.1, shape=[h1_dim]), name="b1")                                          # 5
W2 = tf.get_variable('W_2', shape = [h1_dim, num_classes], initializer = tf.random_normal_initializer() )      # 5X3
b2 = tf.Variable(tf.constant(0.1, shape=[num_classes]), name="b2")                                     # 3

#First layer operations
H1 = tf.nn.xw_plus_b(X, W1, b1, name="H2")      # 100 X 5
Z1 = tf.sigmoid(H1)                             # 100 X 5
#Second layer operations
H2 = tf.nn.xw_plus_b(H1, W2, b2, name="H2")     # 100 X 3
#Loss function
losses = tf.nn.softmax_cross_entropy_with_logits(H2, input_y)  # 100
loss = tf.reduce_mean(losses) + 0.001 * (tf.nn.l2_loss(W1) + tf.nn.l2_loss(W2) )              # 1
#Predicitons of the batch
predictions = tf.argmax(H2, 1, name="predictions")
#Accuracy of correct prediction in batch
correct_predictions = tf.equal(predictions, tf.argmax(input_y, 1) )
accuracy = tf.reduce_mean(tf.cast(correct_predictions, "float"), name="accuracy")
#Optimization
optimizer = tf.train.AdamOptimizer(1e-2)
grads_and_vars = optimizer.compute_gradients(loss)
global_step = tf.Variable(0, name="global_step", trainable=False)
train_op = optimizer.apply_gradients(grads_and_vars, global_step = global_step)

with tf.Session() as sess :

        sess.run(tf.initialize_all_variables())
        # Create Dataset
        D = np.asarray( np.random.rand(1000, 10), dtype='float32')
        Y = np.asarray ( np.zeros((1000, 3)), dtype='float32' )
        for i in range(1000) :
                k = np.random.randint(3)
                Y[i][k] = 1.0
        X_train = D[0:800]; Y_train = Y[0:800]
        X_test = D[800:] ; Y_test = Y[800:]
        # Training
```

```
i_dim = 10
h1_dim = 5
num_classes = 3

#Symbolic or Place holder for input and output
X = tf.placeholder(tf.float32, [None, i_dim], name='input')                  # 100 X 10
input_y = tf.placeholder(tf.float32, [None, num_classes], name="input_y")       # 100 X 3

#Initialize parameters
W1 = tf.get_variable('W_1', shape = [i_dim, h1_dim], initializer = tf.random_normal_initializer() )    # 10 X 5
b1 = tf.Variable(tf.constant(0.1, shape=[h1_dim]), name="b1")                                          # 5
W2 = tf.get_variable('W_2', shape = [h1_dim, num_classes], initializer = tf.random_normal_initializer() )      # 5X3
b2 = tf.Variable(tf.constant(0.1, shape=[num_classes]), name="b2")                                     # 3
```

Multi Layer Neural Network (3)



```
import tensorflow as tf

i_dim = 10
h1_dim = 5
num_classes = 3
#Symbolic or Place holder for input and output
X = tf.placeholder(tf.float32, [None, i_dim], name='input')                  # 100 X 10
input_y = tf.placeholder(tf.float32, [None, num_classes], name="input_y")    # 100 X 3
#Initialize parameters
W1 = tf.get_variable('W_1', shape = [i_dim, h1_dim], initializer = tf.random_normal_initializer() )    # 10 X 5
b1 = tf.Variable(tf.constant(0.1, shape=[h1_dim]), name="b1")                                          # 5
W2 = tf.get_variable('W_2', shape = [h1_dim, num_classes], initializer = tf.random_normal_initializer() )      # 5X3
b2 = tf.Variable(tf.constant(0.1, shape=[num_classes]), name="b2")                                     # 3

#First layer operations
H1 = tf.nn.xw_plus_b(X, W1, b1, name="H1")      # 100 X 5
Z1 = tf.sigmoid(H1)                             # 100 X 5
#Second layer operations
H2 = tf.nn.xw_plus_b(H1, W2, b2, name="H2")     # 100 X 3
#Loss function
losses = tf.nn.softmax_cross_entropy_with_logits(H2, input_y)  # 100
loss = tf.reduce_mean(losses) + 0.001 * (tf.nn.l2_loss(W1) + tf.nn.l2_loss(W2) )               # 1
#Predicitons of the batch
predictions = tf.argmax(H2, 1, name="predictions")
#Accuracy of correct prediction in batch
correct_predictions = tf.equal(predictions, tf.argmax(input_y, 1) )
accuracy = tf.reduce_mean(tf.cast(correct_predictions, "float"), name="accuracy")
#Optimization
optimizer = tf.train.AdamOptimizer(1e-2)
grads_and_vars = optimizer.compute_gradients(loss)
global_step = tf.Variable(0, name="global_step", trainable=False)
train_op = optimizer.apply_gradients(grads_and_vars, global_step = global_step)

with tf.Session() as sess :

        sess.run(tf.initialize_all_variables())
        # Create Dataset
        D = np.asarray( np.random.rand(1000, 10), dtype='float32')
        Y = np.asarray ( np.zeros((1000, 3)), dtype='float32' )
        for i in range(1000) :
                k = np.random.randint(3)
                Y[i][k] = 1.0
        X_train = D[0:800]; Y_train = Y[0:800]
        X_test = D[800:] ; Y_test = Y[800:]
        # Training
```

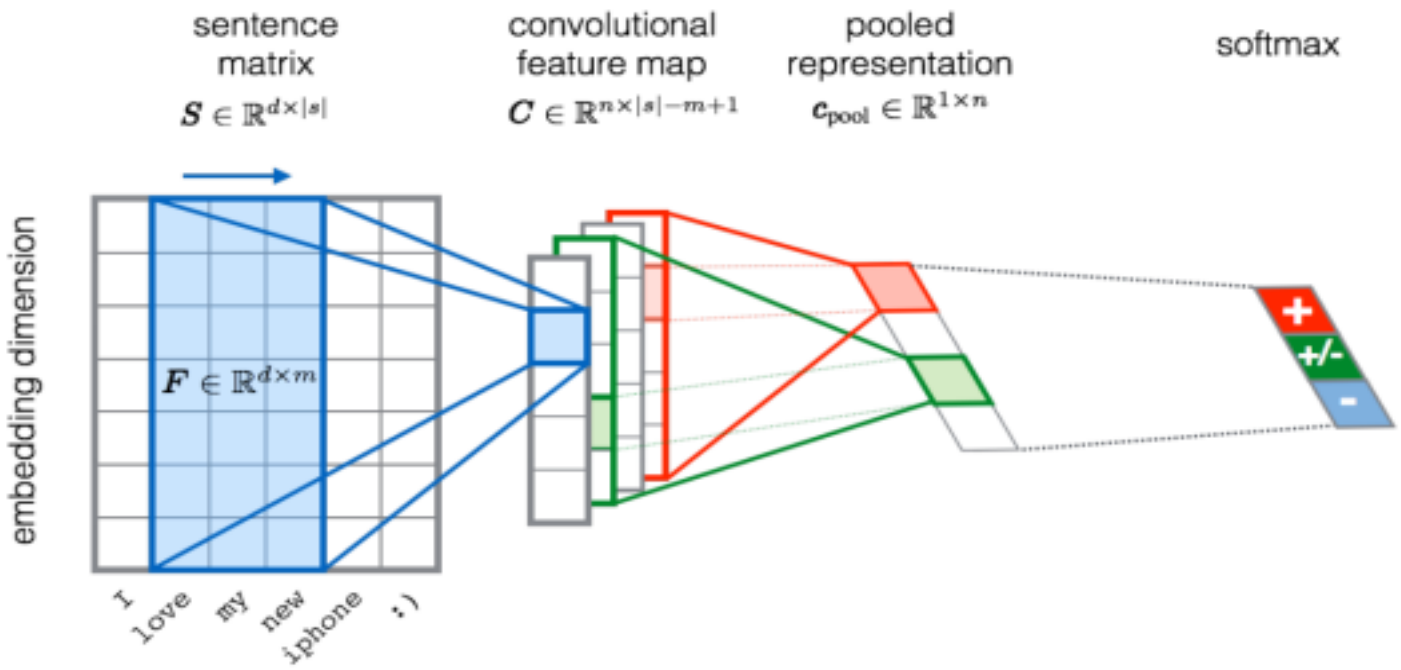
```
#First layer operations
H1 = tf.nn.xw_plus_b(X, W1, b1, name="H1")      # 100 X 5
Z1 = tf.sigmoid(H1)                             # 100 X 5
#Second layer operations
H2 = tf.nn.xw_plus_b(Z1, W2, b2, name="H2")     # 100 X 3

#Loss function
losses = tf.nn.softmax_cross_entropy_with_logits(H2, input_y)   # 100
loss = tf.reduce_mean(losses) + 0.001 * (tf.nn.l2_loss(W1) + tf.nn.l2_loss(W2) )

#Predicitons of the batch
predictions = tf.argmax(H2, 1, name="predictions")

#Accuracy of correct prediction in batch
correct_predictions = tf.equal(predictions, tf.argmax(input_y, 1) )
accuracy = tf.reduce_mean(tf.cast(correct_predictions, "float"), name="accuracy")

#Optimization
optimizer = tf.train.AdamOptimizer(1e-2)
grads_and_vars = optimizer.compute_gradients(loss)
global_step = tf.Variable(0, name="global_step", trainable=False)
train_op = optimizer.apply_gradients(grads_and_vars, global_step = global_step)
```

Multi Layer Neural Network (4)

```python
import tensorflow as tf

i_dim = 10
h1_dim = 5
num_classes = 3
#Symbolic or Place holder for input and output
X = tf.placeholder(tf.float32, [None, i_dim], name='input')                  # 100 X 10
input_y = tf.placeholder(tf.float32, [None, num_classes], name="input_y")    # 100 X 3
#Initialize parameters
W1 = tf.get_variable('W_1', shape = [i_dim, h1_dim], initializer = tf.random_normal_initializer() )    # 10 X 5
b1 = tf.Variable(tf.constant(0.1, shape=[h1_dim]), name="b1")                                          # 5
W2 = tf.get_variable('W_2', shape = [h1_dim, num_classes], initializer = tf.random_normal_initializer() )      # 5X3
b2 = tf.Variable(tf.constant(0.1, shape=[num_classes]), name="b2")                                     # 3

#First layer operations
H1 = tf.nn.xw_plus_b(X, W1, b1, name="H2")      # 100 X 5
Z1 = tf.sigmoid(H1)                             # 100 X 5
#Second layer operations
H2 = tf.nn.xw_plus_b(H1, W2, b2, name="H2")     # 100 X 3
#Loss function
losses = tf.nn.softmax_cross_entropy_with_logits(H2, input_y)  # 100
loss = tf.reduce_mean(losses) + 0.001 * (tf.nn.l2_loss(W1) + tf.nn.l2_loss(W2) )               # 1
#Predicitons of the batch
predictions = tf.argmax(H2, 1, name="predictions")
#Accuracy of correct prediction in batch
correct_predictions = tf.equal(predictions, tf.argmax(input_y, 1) )
accuracy = tf.reduce_mean(tf.cast(correct_predictions, "float"), name="accuracy")
#Optimization
optimizer = tf.train.AdamOptimizer(1e-2)
grads_and_vars = optimizer.compute_gradients(loss)
global_step = tf.Variable(0, name="global_step", trainable=False)
train_op = optimizer.apply_gradients(grads_and_vars, global_step = global_step)

with tf.Session() as sess :

        sess.run(tf.initialize_all_variables())
        # Create Dataset
        D = np.asarray( np.random.rand(1000, 10), dtype='float32')
        Y = np.asarray ( np.zeros((1000, 3)), dtype='float32' )
        for i in range(1000) :
                k = np.random.randint(3)
                Y[i][k] = 1.0
        X_train = D[0:800]; Y_train = Y[0:800]
        X_test = D[800:] ; Y_test = Y[800:]
        # Training
```

```python

with tf.Session() as sess :

        sess.run(tf.initialize_all_variables())

        # Create Dataset
        D = np.asarray( np.random.rand(1000, 10), dtype='float32')
        Y = np.asarray ( np.zeros((1000, 3)), dtype='float32' )
        for i in range(1000) :
                k = np.random.randint(3)
                Y[i][k] = 1.0

        X_train = D[0:800]; Y_train = Y[0:800]
        X_test = D[800:] ; Y_test = Y[800:]


        # Training
        for k in range(100) :
                _,l, acc = sess.run([train_op, loss, accuracy], {X:X_train, input_y:Y_train} )
                print 'loss and accuracy', l, acc
        # Testing
        acc, pred = sess.run( [accuracy, predictions], {X:X_test, input_y:Y_test} )
        print "Accuracy in test set", acc

```

Convolution Neural Network (1)



Severyn 2015

Convolution Neural Network (2)

tf.nn.conv2d(input, filter, strides, padding, name)

Arguments :

- **input :** A 4-d tensor of shape [batch_size, len_sent, dim_we, num_channels]
- **filter :** A 4-d tensor of shape [filter_size, dim_we, num_channels, num_filters]
- **strides :** A list of *int* of size 4, [1, 1, 1, 1] / [1,2,1,1] / [1,3,1,1]
- **padding:** "VALID" / "SAME"
- **name :** Name of the node in computation graph

Return :

- A 4-d tensor of shape [batch_size, out_hight, out_width, num_filters]

where out_hight = (len_sent - filter_size + 1) / ($\left\lceil \frac{len-sent}{strides[1]} \right\rceil$)

out_width = 1

Convolution Neural Network (3)

cnnEx.py

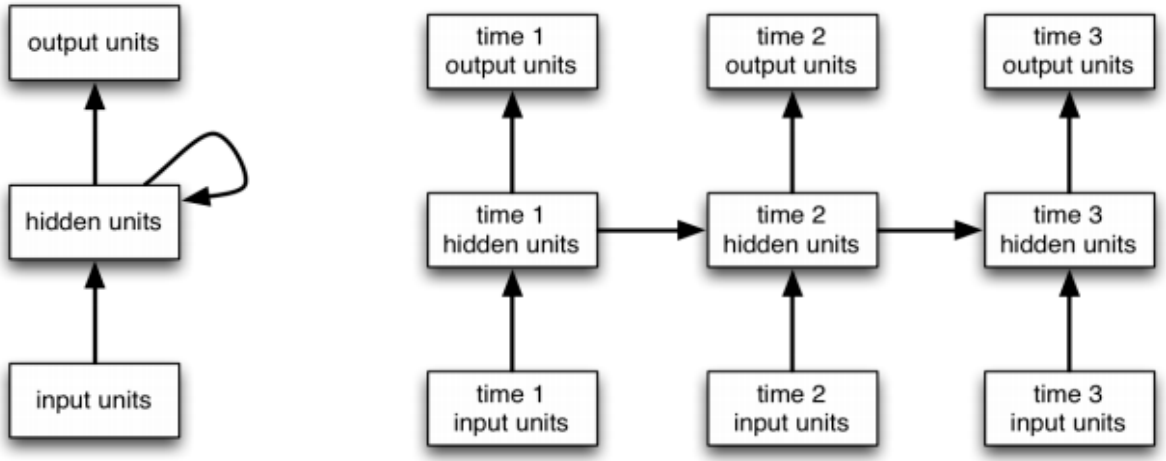
```
import tensorflow as tf
import numpy as np

word_dict_size = 100; sentMax=5; emb_size=4; filter_size=3; num_filters=8

w = tf.placeholder(tf.int32, [None, sentMax], name="x")         # word index matrix (N, 5)
W_wemb =  tf.Variable(tf.random_uniform([word_dict_size, emb_size], -1.0, +1.0))        # word embeddng matrix (100, 4)
emb0 = tf.nn.embedding_lookup(W_wemb, w)        # (N, 5, 4)
X = tf.expand_dims(emb0, -1)                    # (N, 5, 4, 1)

# Convolution layer
W_conv = tf.Variable(tf.truncated_normal([filter_size, emb_size, 1, num_filters], stddev=0.1), name="W_conv")   # ( 3, 4, 1, 8 )
b_conv = tf.Variable(tf.constant(0.1, shape=[num_filters]), name="b_conv")                      # ( 8 )

conv = tf.nn.conv2d(X, W_conv, strides=[1, 1, 1, 1], padding="VALID", name="conv")      # (N, 3, 1, 8)
conv1 = tf.nn.bias_add(conv, b_conv, name='conv1')              # add biase value
h1 = tf.nn.relu(conv1, name="relu")                             # apply activaiton function (N, 3, 1, 8)
pooled = tf.nn.max_pool(h1, ksize=[1, sentMax-filter_size+1, 1, 1], strides=[1, 1, 1, 1], padding='VALID', name="pool") # ( N, 1, 1, 8)
p1 = tf.squeeze(pooled)  # (N,8)

# Running
with tf.Session() as sess :

        sess.run( tf.initialize_all_variables() )
        D = np.asarray( np.random.randint(0, high=word_dict_size, size = [10, sentMax] ), dtype='int32')               # 1000 : size of dataset
        p = sess.run(p1, {w:D} )
        print np.shape(p)               # (10, 8)

```

Convolution Neural Network (4)

- cnnEx1.py : Multiple fixed length filter used
- cnnEx2.py : Multiple variable length filter used [3,4,5]

Recurrent Neural Network (0)

- RNN is a special kind of NN which utilize sequential information and maintains history through its intermediate layer



- $h^{(t)} = \tanh\left(U.x^{(t)} + W.h^{(t-1)} \right)$

Recurrent Neural Network (1)

tf.nn.rnn_cell.BasicRNNCell(num_units)

Argument:

- *num_units:* *int, size of hidden layer [H]*

Return:

- *cell :* *we will use this in dynamtic_rnn() or bidirectional_dynamtic_rnn()*

tf.nn.dynamic_rnn(inputs, cell, dtype, sequence_length)

Arguments:

- *inputs:* *input tensor [N,M,D]*
- *cell:* *Cell of particular RNN*
- *dtype:* *float32/float64*
- *sequence_length:* *Sequence length of each sequence*

Return:

- *output :* *output for every word [N,M,H]*
- *last_state :* *output of last word in every sequence [N,H]*

Recurrent Neural Network (2)

rnnEx.py

```
1  import tensorflow as tf
2  import numpy as np
3
4  X = tf.placeholder(tf.float64, [None, None, 8] )
5  X_lengths = tf.placeholder(tf.int32, [None] )
6
7  cell1 = tf.nn.rnn_cell.BasicRNNCell(num_units=4)
8  outputs, last_states = tf.nn.dynamic_rnn(
9                          cell = cell1,
10                         dtype = tf.float64,
11                         sequence_length = X_lengths,
12                         inputs = X
13                         )
14
15 # Create input data
16 x = np.random.randn(6, 5, 8)
17 x_lengths = [4, 3, 4, 5, 5, 4]
18
19 with tf.Session() as sess:
20         sess.run( tf.initialize_all_variables() )
21         out, sta = sess.run([outputs, last_states], {X:x, X_lengths:x_lengths} )
22         print 'outputs', out
23         print 'final_states', sta
24
```

Recurrent Neural Network (3)

birnnEx.py

```python
import tensorflow as tf
import numpy as np

X = tf.placeholder(tf.float64, [None, None, 8] )
X_lengths = tf.placeholder(tf.int32, [None] )

cell1 = tf.nn.rnn_cell.BasicRNNCell(num_units=4)
cell2 = tf.nn.rnn_cell.BasicRNNCell(num_units=4)
outputs, last_states = tf.nn.bidirectional_dynamic_rnn(
                        cell_fw = cell1,
                        cell_bw = cell2,
                        dtype = tf.float64,
                        sequence_length = X_lengths,
                        inputs = X
                        )

output_fw, output_bw = outputs
output_state_fw, output_state_bw = last_states

# Create input data
x = np.random.randn(6, 5, 8)
x_lengths = [4, 3, 4, 5, 5, 4]

with tf.Session() as sess:
        sess.run( tf.initialize_all_variables() )
        out_fw, out_bw, sta_fw, sta_bw = sess.run([output_fw, output_bw, output_state_fw, output_state_bw], {X:x, X_lengths:x_lengths} )
        print 'forword output', out_fw
        print 'farward final_outputs', sta_fw
        print 'backword output', out_bw
        print 'backword final_outputs', sta_bw


```

Recurrent Neural Network (4)

lstmEx.py

```
1  import tensorflow as tf
2  import numpy as np
3
4  X = tf.placeholder(tf.float64, [None, None, 8] )
5  X_lengths = tf.placeholder(tf.int32, [None] )
6
7  cell = tf.nn.rnn_cell.LSTMCell(num_units=4)
8  outputs, last_states = tf.nn.dynamic_rnn(
9                          cell = cell,
10                         dtype = tf.float64,
11                         sequence_length = X_lengths,
12                         inputs = X
13                         )
14
15
16 # Create input data
17 x = np.random.randn(6, 5, 8)
18 x_lengths = [4, 3, 4, 5, 5, 4]
19
20 with tf.Session() as sess:
21         sess.run( tf.initialize_all_variables() )
22         out, sta = sess.run([outputs, last_states], {X:x, X_lengths:x_lengths} )
23         print 'output', out
24         print 'cell_states', sta.c
25         print 'final_outputs', sta.h
26
27
```

Recurrent Neural Network (5)

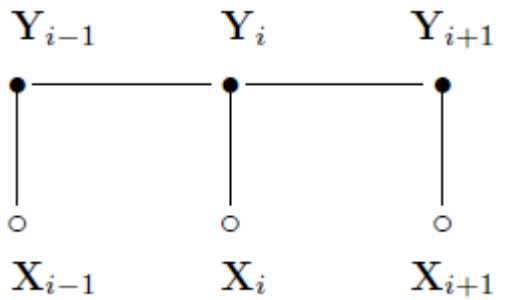
gruEx.py

```
import tensorflow as tf
import numpy as np

X = tf.placeholder(tf.float64, [None, None, 8] )
X_lengths = tf.placeholder(tf.int32, [None] )

cell1 = tf.nn.rnn_cell.GRUCell(num_units=4)
cell2 = tf.nn.rnn_cell.GRUCell(num_units=4)
outputs, last_states = tf.nn.bidirectional_dynamic_rnn(
                        cell_fw = cell1,
                        cell_bw = cell2,
                        dtype = tf.float64,
                        sequence_length = X_lengths,
                        inputs = X
                        )

output_fw, output_bw = outputs
output_state_fw, output_state_bw = last_states

# Create input data
x = np.random.randn(6, 5, 8)
x_lengths = [4, 3, 4, 5, 5, 4]

with tf.Session() as sess:
        sess.run( tf.initialize_all_variables() )
        out_fw, out_bw, sta_fw, sta_bw = sess.run([output_fw, output_bw, output_state_fw, output_state_bw], {X:x, X_lengths:x_lengths} )
        print 'forword output', out_fw
        print 'final_outputs', sta_fw
        print 'backword output', out_bw
        print 'backword final_outputs', sta_bw

```

Recurrent Neural Network (6)

More concrete example is there in *rnn1Ex.py*

Conditional Random Field (1)



- $P\left([t]_1^{|s|} \mid [w]_1^{|s|}\right) = \frac{\exp\left(\text{Score}\left([w]_1^{|s|}, [t]_1^{|s|}\right)\right)}{\sum_{x \in A} \exp\left(\text{Score}\left([w]_1^{|s|}, [t]_1^{|s|}\right)\right)}$

$\text{Score}\left([w]_1^{|s|}, [t]_1^{|s|}\right) = \sum_{1 \le i \le |s|} \left(W_{t_i - 1, ti}^{trans} + Z_{t_i}^{(i)}\right)$

Conditional Random Field

tf.contrib.crf.crf_log_likelihood(US, True_labels, lengths)

Arguments:

1. *US :* *(N,M, c), tensor of unary potential scores for batch*
2. *True_labels :* *(N,M), matrix of true label for every word*
3. *lengths :* *(N) , actual length of every sentence*

Return:

1. *log_like :* *(N), log likelihood of every label sequence*
2. *W_pair:* *(c,c), Pairwise parameter matrix*

tf.contrib.crf.viterbi_decode(us, W_pair)

Arguments:

1. *us :* *(c) vector of unary potential score for a pattern*
2. *W_pair :* *(c,c) Pair wise potential*

Return:

1. *seq :* *(M) highest probability label sequence*
2. *score:* *A float containing containing score of viterbi*

Conditional Random Field (2)

crfEx.py

```python
import tensorflow as tf
import numpy as np

sentMax = 10            # Max lenght of sentence
num_classes = 3         # number of classes
num_features = 15       # number of features

X = tf.placeholder(tf.float32, [None, sentMax, num_features] )  # (6, 10, 15)
input_y = tf.placeholder(tf.int32, [None, sentMax] )            # (6, 10)
X_lengths = tf.placeholder(tf.int32, [None] )                   # (6)
X1 = tf.reshape( X, [-1, num_features] )                        # (6*10, 15)

#Fully connected layer operations
W_ff = tf.Variable(tf.random_uniform([num_features, num_classes], -1.0, +1.0), name="W")  # (15,3)
b_ff = tf.Variable(tf.constant(0.1, shape=[num_classes]), name="b")
H1 = tf.nn.xw_plus_b(X1, W_ff, b_ff, name="H1")                 # ( 6*10, 3)

Z1 = tf.reshape(H1, [-1, sentMax, num_classes] )                # ( 6, 10, 3)

#CRF layer
log_likelihood, transition_params = tf.contrib.crf.crf_log_likelihood( Z1, input_y, X_lengths )
loss = tf.reduce_mean(-log_likelihood)

# Create input data
x = np.asarray( np.random.randn(6, sentMax, num_features), dtype='float32' )    # dataset: #sentence=6, #words = 10, #features = 15
y = np.random.randint(3, size=[6, sentMax])             # 3 number of lables ( B, I, O )
x_lengths = [8, 5, 7, 9, 10, 4]                         # actual length of each sentence

with tf.Session() as sess:
        sess.run( tf.initialize_all_variables() )
        l, tp = sess.run([loss, transition_params], {X:x, X_lengths:x_lengths, input_y:y } )
        print l

        #Decoding
        us, ps = sess.run([Z1, transition_params], {X:x, X_lengths:x_lengths, input_y:y } )
        viterbi_sequence,_ = tf.contrib.crf.viterbi_decode(us[1], ps)          # highest scoring sequence.
        print 'true seq', y[1]
        print 'pred seq', viterbi_sequence
```

Conditional Random Field (3)

crfEx1.py : *Bi-RNN + CRF model for sequence labeling with dummy data*

Thanks

Code used here are available : https://github.com/sunilitggu/tensorflow-tutorial