```
TENSORFLOW EXAMPLE: RECURRENT NEURAL NETWORK

Problem: Adding two 8-bit numbers together.
For example
     00001001 + 00111100 = 01000101
        9     +    60    = 69
```

In [1]:

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

In [2]:

```python
# Define the dataflow graph
time_steps = 8        # time steps which is the same as the length of the bit-string
input_dim = 2         # number of units in the input layer
hidden_dim = 16       # number of units in the hidden layer
output_dim = 1        # number of units in the output layer

# input X and target ouput Y
X = tf.placeholder(tf.float32, [None, time_steps, input_dim])
Y = tf.placeholder(tf.float32, [None, time_steps])

# define the RNN cell: can be simple cell, LSTM or GRU
cell = tf.nn.rnn_cell.BasicRNNCell(num_units=hidden_dim, activation=tf.nn.sigmoid)
# cell = tf.nn.rnn_cell.LSTMCell(hidden_dim, state_is_tuple=True)

# values is a tensor of shape [batch_size, time_steps, hidden_dim]
# last_state is a tensor of shape [batch_size, hidden_dim]
values, last_state = tf.nn.dynamic_rnn(cell, X, dtype=tf.float32)
values = tf.reshape(values,[time_steps, hidden_dim])

# put the values from the RNN through fully-connected layer
W = tf.Variable(tf.random_uniform([hidden_dim, output_dim], minval=-1.0,maxval=1.0), name='W')
b = tf.Variable(tf.zeros([1, output_dim]), name='b')
h = tf.nn.sigmoid(tf.matmul(values,W) + b)

# minimize loss, using ADAM as weight update rule
h_ = tf.reshape(h, [time_steps])
Y_ = tf.reshape(Y, [time_steps])
loss = tf.reduce_sum(-Y_ * tf.log(h_) - (1-Y_) * tf.log(1-h_))
train_step = tf.train.AdamOptimizer(0.1).minimize(loss)
```

In [3]:

```python
# Launch the graph
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# Try out the example
# 00001001 + 00111100 = 01000101
#    9     +    60    = 69
x = np.array([
    # t=0   t=1    t=2    t=3    t=4    t=5    t=6   t=7
    [0,0], [0,0], [0,1], [0,1], [1,1], [0,1], [0,0], [1,0]
]).reshape([1,8,2])
y = np.array([0, 1, 0, 0, 0, 1, 0, 1]).reshape([1,8])

# train
sess.run(train_step, {X: x, Y: y})

# print result
[output_vals, _probs, _loss] = sess.run([values, h, loss], {X: x, Y: y})
print('Raw output values:')
print(output_vals)

# prediction
probs = np.array(_probs).reshape([8])
prediction = np.array([1 if p >= 0.5 else 0 for p in probs]).reshape([8])
print()
print('Probabilities: \n', probs)
print()
print('Prediction  :', prediction)

# calculate absolute error
error = np.sum(np.absolute(y - probs))

# print out pre-training X-entropy loss and absolute error
print('Absolute error : ', error)
print('X-entropy loss : ', _loss)
```

```
Raw output values:
[[ 0.47502092  0.47502097  0.52497905  0.47502092  0.52497905  0.52497888
   0.47502086  0.47502092  0.52497894  0.47502086  0.47502086  0.524979
   0.52497888  0.52497905  0.52497751  0.52497911]
 [ 0.14841537  0.61733449  0.75091577  0.28058618  0.56417257  0.68358868
   0.16095461  0.29389748  0.40222776  0.20322989  0.34588733  0.72239882
   0.79938602  0.78864682  0.58117068  0.52698869]
 [ 0.12746073  0.57632226  0.7341361   0.40259528  0.4287056   0.65959179
   0.15043941  0.31832516  0.43690103  0.18125151  0.38751271  0.717085
   0.80123001  0.73989534  0.50207788  0.4642967 ]
 [ 0.13945937  0.55163407  0.7223177   0.41603133  0.44143528  0.66948217
   0.15613405  0.33669403  0.43700132  0.18105952  0.3861483   0.70811087
   0.79458827  0.72580874  0.49339053  0.44845974]
 [ 0.13654141  0.48674026  0.80694008  0.30669942  0.56263697  0.65702188
   0.1532962   0.40506229  0.53345579  0.1348618   0.37143165  0.71475589
   0.78726411  0.70103151  0.52853388  0.38627633]
 [ 0.13166356  0.55130708  0.75171822  0.40902984  0.45188832  0.6567297
   0.15456353  0.31604448  0.44324529  0.17099875  0.38202119  0.69221586
   0.78805459  0.74973863  0.50217563  0.46639466]
 [ 0.17191656  0.56666446  0.76289117  0.33790958  0.57055444  0.66204184
   0.16484019  0.35837135  0.42414922  0.21562377  0.32464689  0.76766944
   0.80519629  0.74734473  0.57357621  0.56227982]
 [ 0.14956163  0.52249789  0.84678692  0.22361328  0.66431755  0.65289211
   0.15116327  0.40908679  0.51506007  0.15196115  0.30279168  0.78318781
   0.8053413   0.74328947  0.61467046  0.51279181]]

Probabilities: 
 [ 0.5623672   0.82815796  0.79375088  0.78649455  0.79683888  0.79787099
  0.82452643  0.84132642]

Prediction  : [1 1 1 1 1 1 1 1]
Absolute error :  4.29662257433
X-entropy loss :  7.8703
```

In [4]:

```
# let's train

# generate the training data set
binary_dim = 8
largest_number = pow(2, binary_dim)
a_list = []
b_list = []
n_examples = 1000
for j in range(n_examples):
    a_list.append(np.random.randint(largest_number/2))
    b_list.append(np.random.randint(largest_number/2))

# train
n_epochs = 10
for i in range(n_epochs):

    for j in range(n_examples):
        a_int = a_list[j]
        b_int = b_list[j]
        c_int = a_int + b_int
        a = np.unpackbits(np.array([a_int], dtype=np.uint8))
        b = np.unpackbits(np.array([b_int], dtype=np.uint8))
        c = np.unpackbits(np.array([c_int], dtype=np.uint8))
        ab = np.c_[a,b]
        x = np.array(ab).reshape([1,binary_dim,2])
        y = np.array(c).reshape([1,binary_dim])
        sess.run(train_step, {X: x, Y: y})

    # print out loss for a random example
    if (i%1 == 0):
        # pick a random example out of the training data set
        k = np.random.randint(n_examples)
        a_int = a_list[k]
        b_int = b_list[k]
        c_int = a_int + b_int
        a = np.unpackbits(np.array([a_int], dtype=np.uint8))
        b = np.unpackbits(np.array([b_int], dtype=np.uint8))
        c = np.unpackbits(np.array([c_int], dtype=np.uint8))
        ab = np.c_[a,b]
        x = np.array(ab).reshape([1,binary_dim,2])
        y = np.array(c).reshape([1,binary_dim])

        # get predicted value
        [_probs, _loss] = sess.run([h, loss], {X: x, Y: y})
        probs = np.array(_probs).reshape([8])
        prediction = np.array([1 if p >= 0.5 else 0 for p in probs]).reshape([8])
        pred_int = np.sum(np.packbits(prediction))

        # calculate error
        error = np.sum(np.absolute(y - probs))

        print('Input 1         : ', a, ' (', a_int, ')')
        print('Input 2         : ', b, ' (', b_int, ')')
        print('True            : ', c, ' (', c_int, ')')
        print('Predicted       : ', prediction, ' (', pred_int, ')')
        print('Absolute error  : ', error)
        print('X-entropy loss  : ', _loss)
        print('--------------------------------')
        print()
```

```
Input 1         :  [0 0 1 0 1 1 0 1]  ( 45 )
Input 2         :  [0 0 0 1 1 0 1 0]  ( 26 )
True            :  [0 1 0 0 0 1 1 1]  ( 71 )
Predicted       :  [0 0 1 1 0 1 1 1]  ( 55 )
Absolute error  :  3.73035
X-entropy loss  :  5.47834
--------------------------------

Input 1         :  [0 0 0 0 1 1 1 0]  ( 14 )
Input 2         :  [0 0 1 1 0 1 0 1]  ( 53 )
True            :  [0 1 0 0 0 0 1 1]  ( 67 )
Predicted       :  [1 0 1 1 1 0 1 1]  ( 187 )
Absolute error  :  4.29031
X-entropy loss  :  6.66568
--------------------------------
```

```
Input 1         :  [0 0 1 0 0 0 0 1]  ( 33 )
Input 2         :  [0 1 0 1 1 1 1 0]  ( 94 )
True            :  [0 1 1 1 1 1 1 1]  ( 127 )
Predicted       :  [1 1 1 1 1 1 1 1]  ( 255 )
Absolute error  :  2.57187
X-entropy loss  :  3.17627
--------------------------------

Input 1         :  [0 0 1 1 0 0 1 0]  ( 50 )
Input 2         :  [0 0 0 0 0 0 1 1]  ( 3 )
True            :  [0 0 1 1 0 1 0 1]  ( 53 )
Predicted       :  [1 0 1 1 0 0 0 1]  ( 177 )
Absolute error  :  3.221
X-entropy loss  :  4.36655
--------------------------------

Input 1         :  [0 1 1 1 1 1 1 1]  ( 127 )
Input 2         :  [0 1 0 1 1 0 1 0]  ( 90 )
True            :  [1 1 0 1 1 0 0 1]  ( 217 )
Predicted       :  [1 0 1 0 0 1 0 1]  ( 165 )
Absolute error  :  4.30266
X-entropy loss  :  6.77041
--------------------------------

Input 1         :  [0 1 1 1 1 0 0 1]  ( 121 )
Input 2         :  [0 1 1 0 1 0 0 0]  ( 104 )
True            :  [1 1 1 0 0 0 0 1]  ( 225 )
Predicted       :  [1 0 0 1 0 0 0 1]  ( 145 )
Absolute error  :  3.78583
X-entropy loss  :  5.52448
--------------------------------

Input 1         :  [0 0 0 0 0 0 1 1]  ( 3 )
Input 2         :  [0 1 0 1 1 1 1 1]  ( 95 )
True            :  [0 1 1 0 0 0 1 0]  ( 98 )
Predicted       :  [1 0 0 1 1 1 0 0]  ( 156 )
Absolute error  :  4.98435
X-entropy loss  :  8.13293
--------------------------------

Input 1         :  [0 1 0 0 1 0 1 0]  ( 74 )
Input 2         :  [0 1 1 0 1 1 0 1]  ( 109 )
True            :  [1 0 1 1 0 1 1 1]  ( 183 )
Predicted       :  [1 0 1 0 0 1 1 1]  ( 167 )
Absolute error  :  2.79596
X-entropy loss  :  3.56718
--------------------------------

Input 1         :  [0 0 1 0 1 1 0 0]  ( 44 )
Input 2         :  [0 0 1 1 1 0 1 1]  ( 59 )
True            :  [0 1 1 0 0 1 1 1]  ( 103 )
Predicted       :  [1 0 0 1 0 1 1 1]  ( 151 )
Absolute error  :  3.66828
X-entropy loss  :  5.39607
--------------------------------

Input 1         :  [0 0 0 1 1 0 1 0]  ( 26 )
Input 2         :  [0 1 0 0 0 1 0 0]  ( 68 )
True            :  [0 1 0 1 1 1 1 0]  ( 94 )
Predicted       :  [1 0 0 1 1 1 1 0]  ( 158 )
Absolute error  :  3.09235
X-entropy loss  :  4.10179
--------------------------------
```

```
In [ ]: sess.close()
```