Deep Learning

Lab Session 3 - 3 Hours

Long Short Term Memory (LSTM) for Language Modeling

Student 1: EL MAATAOUI OUSSAMA **Student 2:** EL KHAMAR Zakaria

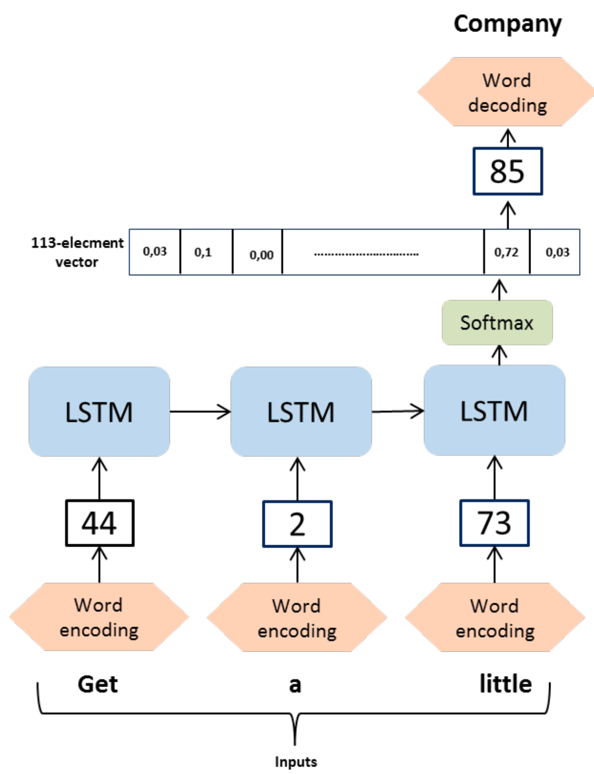
In this Lab Session, you will build and train a Recurrent Neural Network, based on Long Short-Term Memory (LSTM) units for next word prediction task.

Answers and experiments should be made by groups of one or two students. Each group should fill and run appropriate notebook cells. Once you have completed all of the code implementations and successfully answered each question above, you may finalize your work by exporting the iPython Notebook as an pdf document using print as PDF (Ctrl+P). Do not forget to run all your cells before generating your final report and do not forget to include the names of all participants in the group. The lab session should be completed by June 9th 2017.

Send you pdf file to benoit.huet@eurecom.fr and olfa.ben-ahmed@eurecom.fr using **[DeepLearning_lab3]** as Subject of your email.

Introduction

You will train a LSTM to predict the next word using a sample short story. The LSTM will learn to predict the next item of a sentence from the 3 previous items (given as input). Ponctuation marks are considered as dictionnary items so they can be predicted too. Figure 1 shows the LSTM and the process of next word prediction.



Each word (and ponctuation) from text sentences is encoded by a unique integer. The integer value corresponds to the index of the corresponding word (or punctuation mark) in the dictionnary. The network output is a one-hot-vector indicating the index of the predicted word in the reversed dictionnary (Section 1.2). For example if the prediction is 86, the predicted word will be "company".

You will use a sample short story from Aesop's Fables (http://www.taleswithmorals.com/ (http://www.taleswithmorals.com/)) to train your model.

"There was once a young Shepherd Boy who tended his sheep at the foot of a mountain near a dark forest.

It was rather lonely for him all day, so he thought upon a plan by which he could get a little company and some excitement. He rushed down towards the village calling out "Wolf, Wolf," and the villagers came out to meet him, and some of them stopped with him for a considerable time. This pleased the boy so much that a few days afterwards he tried the same trick, and again the villagers came to his help. But shortly after this a Wolf actually did come out from the forest, and began to worry the sheep, and the boy of course cried out "Wolf, Wolf," still louder than before. But this time the villagers, who had been fooled twice before, thought the boy was again deceiving them, and nobody stirred to come to his help. So the Wolf made a good meal off the boy's flock, and when the boy complained, the wise man of the village said: "A liar will not be believed, even when he speaks the truth." " .

Start by loading the necessary libraries and resetting the default computational graph. For more details about the rnn packages, we suggest you to take a look at https://www.tensorflow.org/api_guides/python/contrib.rnn (https://www.tensorflow.org/api_guides/python/contrib.rnn)

In [1]:
```
import numpy as np
import collections # used to build the dictionary
import random
import time
import pickle # may be used to save your model
import matplotlib.pyplot as plt
#Import Tensorflow and rnn
import tensorflow as tf
from tensorflow.contrib import rnn

# Target log path
logs_path = 'lstm_words'
writer = tf.summary.FileWriter(logs_path)
```

Next-word prediction task

Part 1: Data preparation

1.1. Loading data

Load and split the text of our story

In [2]:
```
def load_data(filename):
    with open(filename) as f:
        data = f.readlines()
    data = [x.strip().lower() for x in data]
    data = [data[i].split() for i in range(len(data))]
    data = np.array(data)
    data = np.reshape(data, [-1, ])
    print(data)
    return data

#Run the cell
train_file ='data/story.txt'
train_data = load_data(train_file)
print("Loaded training data...")
print(len(train_data))
```

```
['there' 'was' 'once' 'a' 'young' 'shepherd' 'boy' 'who' 'tended' 'his'
 'sheep' 'at' 'the' 'foot' 'of' 'a' 'mountain' 'near' 'a' 'dark' 'forest'
 '.' 'it' 'was' 'rather' 'lonely' 'for' 'him' 'all' 'day' ',' 'so' 'he'
 'thought' 'upon' 'a' 'plan' 'by' 'which' 'he' 'could' 'get' 'a' 'little'
 'company' 'and' 'some' 'excitement' '.' 'he' 'rushed' 'down' 'towards'
 'the' 'village' 'calling' 'out' 'wolf' ',' 'wolf' ',' 'and' 'the'
 'villagers' 'came' 'out' 'to' 'meet' 'him' ',' 'and' 'some' 'of' 'them'
 'stopped' 'with' 'him' 'for' 'a' 'considerable' 'time' '.' 'this'
 'pleased' 'the' 'boy' 'so' 'much' 'that' 'a' 'few' 'days' 'afterwards'
 'he' 'tried' 'the' 'same' 'trick' ',' 'and' 'again' 'the' 'villagers'
 'came' 'to' 'his' 'help' '.' 'but' 'shortly' 'after' 'this' 'a' 'wolf'
 'actually' 'did' 'come' 'out' 'from' 'the' 'forest' ',' 'and' 'began' 'to'
 'worry' 'the' 'sheep,' 'and' 'the' 'boy' 'of' 'course' 'cried' 'out'
 'wolf' ',' 'wolf' ',' 'still' 'louder' 'than' 'before' '.' 'but' 'this'
 'time' 'the' 'villagers' ',' 'who' 'had' 'been' 'fooled' 'twice' 'before'
 ',' 'thought' 'the' 'boy' 'was' 'again' 'deceiving' 'them' ',' 'and'
 'nobody' 'stirred' 'to' 'come' 'to' 'his' 'help' '.' 'so' 'the' 'wolf'
 'made' 'a' 'good' 'meal' 'off' 'the' "boy's" 'flock' ',' 'and' 'when'
 'the' 'boy' 'complained' ',' 'the' 'wise' 'man' 'of' 'the' 'village'
 'said' ':' 'a' 'liar' 'will' 'not' 'be' 'believed' ',' 'even' 'when' 'he'
 'speaks' 'the' 'truth' '.']
Loaded training data...
214
```

1.2.Symbols encoding

The LSTM input's can only be numbers. A way to convert words (symbols or any items) to numbers is to assign a unique integer to each word. This process is often based on frequency of occurrence for efficient coding purpose.

Here, we define a function to build an indexed word dictionary (word->number). The "build_vocabulary" function builds both:

- Dictionary : used for encoding words to numbers for the LSTM inputs
- Reverted dictionnary : used for decoding the outputs of the LSTM into words (and punctuation).

For example, in the story above, we have **113** individual words. The "build_vocabulary" function builds a dictionary with the following entries ['the': 0], [',': 1], ['company': 85],...

In [3]:
```
def build_vocabulary(words):
    count = collections.Counter(words).most_common()
    dic= dict()
    for word, _ in count:
        dic[word] = len(dic)
    reverse_dic= dict(zip(dic.values(), dic.keys()))
    return dic, reverse_dic
```

Run the cell below to display the vocabulary

In [4]:
```
dictionary, reverse_dictionary = build_vocabulary(train_data)
vocabulary_size= len(dictionary)
print "Dictionary size (Vocabulary size) = ", vocabulary_size
print("\n")
print("Dictionary : \n")
print(dictionary)
print("\n")
print("Reverted Dictionary : \n" )
print(reverse_dictionary)
```

```
Dictionary size (Vocabulary size) =  113


Dictionary : 

{'all': 32, 'liar': 33, 'help': 17, 'cried': 34, 'course': 35, 'still': 36, 'plea
sed': 37, 'before': 18, 'excitement': 91, 'deceiving': 38, 'had': 39, 'young': 69
, 'actually': 40, 'to': 6, 'villagers': 11, 'shepherd': 41, 'them': 19, 'lonely':
42, 'get': 44, 'dark': 45, 'not': 64, 'day': 47, 'did': 48, 'calling': 49, 'twice
': 50, 'good': 51, 'stopped': 52, 'truth': 53, 'meal': 54, 'sheep,': 55, 'some': 
20, 'tended': 56, 'louder': 57, 'flock': 58, 'out': 9, 'even': 59, 'trick': 60, '
said': 61, 'for': 21, 'be': 62, 'after': 63, 'come': 22, 'by': 65, 'boy': 7, 'of'
: 10, 'could': 66, 'days': 67, 'wolf': 5, 'afterwards': 68, ',': 1, 'down': 70, '
village': 23, 'sheep': 72, 'little': 73, 'from': 74, 'rushed': 75, 'there': 76, '
been': 77, '.': 4, 'few': 78, 'much': 79, "boy's": 80, ':': 81, 'was': 12, 'a': 2
, 'him': 13, 'that': 83, 'company': 84, 'nobody': 85, 'but': 24, 'fooled': 86, 'w
ith': 87, 'than': 43, 'he': 8, 'made': 89, 'wise': 90, 'this': 14, 'will': 71, 'n
ear': 92, 'believed': 93, 'meet': 94, 'and': 3, 'it': 95, 'his': 15, 'at': 96, 'w
orry': 97, 'again': 25, 'considerable': 88, 'rather': 98, 'began': 99, 'when': 26
, 'same': 101, 'forest': 27, 'which': 102, 'speaks': 103, 'towards': 104, 'tried'
: 105, 'mountain': 106, 'who': 28, 'upon': 107, 'plan': 108, 'man': 109, 'complai
ned': 82, 'stirred': 110, 'off': 100, 'foot': 46, 'shortly': 111, 'thought': 29,
'so': 16, 'time': 30, 'the': 0, 'came': 31, 'once': 112}


Reverted Dictionary : 

{0: 'the', 1: ',', 2: 'a', 3: 'and', 4: '.', 5: 'wolf', 6: 'to', 7: 'boy', 8: 'he
', 9: 'out', 10: 'of', 11: 'villagers', 12: 'was', 13: 'him', 14: 'this', 15: 'hi
s', 16: 'so', 17: 'help', 18: 'before', 19: 'them', 20: 'some', 21: 'for', 22: 'c
ome', 23: 'village', 24: 'but', 25: 'again', 26: 'when', 27: 'forest', 28: 'who',
29: 'thought', 30: 'time', 31: 'came', 32: 'all', 33: 'liar', 34: 'cried', 35: 'c
ourse', 36: 'still', 37: 'pleased', 38: 'deceiving', 39: 'had', 40: 'actually', 4
1: 'shepherd', 42: 'lonely', 43: 'than', 44: 'get', 45: 'dark', 46: 'foot', 47: '
day', 48: 'did', 49: 'calling', 50: 'twice', 51: 'good', 52: 'stopped', 53: 'trut
h', 54: 'meal', 55: 'sheep,', 56: 'tended', 57: 'louder', 58: 'flock', 59: 'even'
, 60: 'trick', 61: 'said', 62: 'be', 63: 'after', 64: 'not', 65: 'by', 66: 'could
', 67: 'days', 68: 'afterwards', 69: 'young', 70: 'down', 71: 'will', 72: 'sheep'
, 73: 'little', 74: 'from', 75: 'rushed', 76: 'there', 77: 'been', 78: 'few', 79:
'much', 80: "boy's", 81: ':', 82: 'complained', 83: 'that', 84: 'company', 85: 'n
obody', 86: 'fooled', 87: 'with', 88: 'considerable', 89: 'made', 90: 'wise', 91:
'excitement', 92: 'near', 93: 'believed', 94: 'meet', 95: 'it', 96: 'at', 97: 'wo
rry', 98: 'rather', 99: 'began', 100: 'off', 101: 'same', 102: 'which', 103: 'spe
aks', 104: 'towards', 105: 'tried', 106: 'mountain', 107: 'upon', 108: 'plan', 10
9: 'man', 110: 'stirred', 111: 'shortly', 112: 'once'}
```

Part 2 : LSTM Model in TensorFlow

Since you have defined how the data will be modeled, you are now to develop an LSTM model to predict the word of following a sequence of 3 words.

2.1. Model definition

Define a 2-layers LSTM model.

For this use the following classes from the tensorflow.contrib library:

- rnn.BasicLSTMCell(number of hidden units)
- rnn.static_rnn(rnn_cell, data, dtype=tf.float32)
- rnn.MultiRNNCell(,)

You may need some tensorflow functions (https://www.tensorflow.org/api_docs/python/tf/ (https://www.tensorflow.org/api_docs/python/tf/)) :

- tf.split
- tf.reshape
- ...

In [5]:
```
def lstm_model(x, w, b,n_hidden,n_input):
    x = tf.reshape(x, (-1,n_input))
    x= tf.split(x, n_input, 1)
    lstm_1 = rnn.BasicLSTMCell(n_hidden)
    lstm_2 = rnn.BasicLSTMCell(n_hidden)
    rnn_cell = rnn.MultiRNNCell([lstm_1, lstm_2])
    output, state = rnn.static_rnn(rnn_cell,x,dtype=tf.float32)


    logits = tf.nn.softmax(tf.matmul(output[-1], w) + b)
    return logits
```

Training Parameters and constants

In [6]:
```
# Training Parameters
learning_rate = 0.001
epochs = 50000
display_step = 1000
n_input = 3

#For each LSTM cell that you initialise, supply a value for the hidden dimension,
n_hidden = 64

# tf Graph input
x = tf.placeholder("float", [None, n_input, 1])
y = tf.placeholder("float", [None, vocabulary_size])

# LSTM  weights and biases
weights = { 'out': tf.Variable(tf.random_normal([n_hidden, vocabulary_size]))}
biases = {'out': tf.Variable(tf.random_normal([vocabulary_size])) }


#build the model
pred = lstm_model(x, weights['out'], biases['out'],n_hidden,n_input)
```

Define the Loss/Cost and optimizer

In [7]:
```
# Loss and optimizer
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(pred), reduction_indices=1))
optimizer =tf.train.RMSPropOptimizer(learning_rate).minimize(cost)



# Model evaluation
correct_pred = tf.equal(tf.argmax(pred,1), tf.argmax(y,1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

We give you here the Test Function

In [9]:
```
#run the cell
def test(sentence, session, verbose=False):
    sentence = sentence.strip()
    words = sentence.split(' ')
    if len(words) != n_input:
        print("sentence length should be equel to", n_input, "!")
    try:
        symbols_inputs = [dictionary[str(words[i - n_input])] for i in range(n_inp
        keys = np.reshape(np.array(symbols_inputs), [-1, n_input, 1])
        onehot_pred = session.run(pred, feed_dict={x: keys})
        onehot_pred_index = int(tf.argmax(onehot_pred, 1).eval())
        words.append(reverse_dictionary[onehot_pred_index])
        sentence = " ".join(words)
        if verbose:
            print(sentence)
        return reverse_dictionary[onehot_pred_index]
    except:
        print " ".join(["Word", words[i - n_input], "not in dictionary"])
```

Part 3 : LSTM Training

In the Training process, at each epoch, 3 words are taken from the training data, encoded to integer to form the input vector. The training labels are one-hot vector encoding the word that comes after the 3 inputs words. Display the loss and the training accuracy every 1000 iteration. Save the model at the end of training in the **lstm_model** folder

> **Note :** We've tried two train functions : **The first** takes words from the dictionnary in order, in other words it takes at each epoch as the first word the word following the word that has been taken in the previous epoch, and when the whole dictionnary is browsed, it restarts from the beginning, while **the second** takes at each epoch a random word as first of the three input words.

In [9]:

```
# Initializing the variables
start_time = time.time()
init = tf.global_variables_initializer()
model_saver = tf.train.Saver()
logs_path = 'lstm_model'
def train(X_train):
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        summary_writer = tf.summary.FileWriter(logs_path, graph=tf.get_default_gra
        merged_summary_op = tf.summary.merge_all()

        print("Start Training!")
        s=0
        accu_total = 0
        for i in range(epochs):
            if s > len(train_data)-4 :
                s=0

            symbols_inputs = [dictionary[str(X_train[s+j])] for j in range(n_input
            keys = np.reshape(np.array(symbols_inputs), [-1, n_input, 1])
            symbols_labels = np.zeros([vocabulary_size], dtype = float)
            symbols_labels[dictionary[str(X_train[s+3])]] = 1.0
            symbols_labels = np.reshape(symbols_labels,[1,-1])

            _, accu, loss, onehot_pred = sess.run([optimizer, accuracy, cost, pred
            s+=1
            accu_total+= accu
            if  (i+1) % display_step == 0:
                print("Epoch: ", '%02d' % (i+1), "  =====> Loss=", "{:.9f}".format
                print("Training Accuracy = {:.3f}".format(((accu_total)*100)/displ
                accu_total=0


        print("End Of training Finished!")
        print("time: ",time.time() - start_time)
        print("For tensorboard visualisation run on command line.")
        print("\ttensorboard --logdir=%s" % (logs_path))
        print("and oint your web browser to the returned link")
        ##############################################
        #save your model
        model_saver.save(sess, 'lstm_model/lstm_model1')
        ##############################################
        print("Model saved")
```

In [10]:
```
train(train_data)
```

```
Start Training!
('Epoch: ', '1000', '  =====> Loss=', '2.380809307')
Training Accuracy = 6.200
('Epoch: ', '2000', '  =====> Loss=', '3.366992950')
Training Accuracy = 9.600
('Epoch: ', '3000', '  =====> Loss=', '2.435416937')
Training Accuracy = 16.000
('Epoch: ', '4000', '  =====> Loss=', '5.410267830')
Training Accuracy = 23.600
('Epoch: ', '5000', '  =====> Loss=', '1.701192141')
Training Accuracy = 36.600
('Epoch: ', '6000', '  =====> Loss=', '0.235773608')
Training Accuracy = 51.700
('Epoch: ', '7000', '  =====> Loss=', '0.623047888')
Training Accuracy = 65.600
('Epoch: ', '8000', '  =====> Loss=', '1.524252772')
Training Accuracy = 73.000
('Epoch: ', '9000', '  =====> Loss=', '0.672870278')
Training Accuracy = 82.400
('Epoch: ', '10000', '  =====> Loss=', '0.058838662')
Training Accuracy = 85.600
('Epoch: ', '11000', '  =====> Loss=', '0.169940501')
Training Accuracy = 89.500
('Epoch: ', '12000', '  =====> Loss=', '0.017100036')
Training Accuracy = 91.700
('Epoch: ', '13000', '  =====> Loss=', '0.208780542')
Training Accuracy = 90.500
('Epoch: ', '14000', '  =====> Loss=', '0.284538150')
Training Accuracy = 91.100
('Epoch: ', '15000', '  =====> Loss=', '0.132702857')
Training Accuracy = 91.700
('Epoch: ', '16000', '  =====> Loss=', '0.101820871')
Training Accuracy = 91.400
('Epoch: ', '17000', '  =====> Loss=', '0.014711962')
Training Accuracy = 91.900
('Epoch: ', '18000', '  =====> Loss=', '0.105078451')
Training Accuracy = 92.300
('Epoch: ', '19000', '  =====> Loss=', '0.231714129')
Training Accuracy = 92.700
('Epoch: ', '20000', '  =====> Loss=', '0.372867703')
Training Accuracy = 89.800
('Epoch: ', '21000', '  =====> Loss=', '0.063532583')
Training Accuracy = 92.600
('Epoch: ', '22000', '  =====> Loss=', '0.003679181')
Training Accuracy = 93.100
('Epoch: ', '23000', '  =====> Loss=', '0.042779397')
Training Accuracy = 93.600
('Epoch: ', '24000', '  =====> Loss=', '0.020676779')
Training Accuracy = 93.600
('Epoch: ', '25000', '  =====> Loss=', '3.836452007')
Training Accuracy = 94.000
('Epoch: ', '26000', '  =====> Loss=', '0.072029136')
Training Accuracy = 93.600
('Epoch: ', '27000', '  =====> Loss=', '0.292555124')
Training Accuracy = 91.000
('Epoch: ', '28000', '  =====> Loss=', '0.028373288')
Training Accuracy = 92.100
('Epoch: ', '29000', '  =====> Loss=', '0.002764586')
Training Accuracy = 92.000
('Epoch: ', '30000', '  =====> Loss=', '0.448878527')
Training Accuracy = 93.400
```

In [11]:
```
# Initializing the variables
from random import randint
start_time = time.time()
init = tf.global_variables_initializer()
model_saver = tf.train.Saver()
logs_path = 'lstm_model'
def train2(X_train):
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        summary_writer = tf.summary.FileWriter(logs_path, graph=tf.get_default_gra
        merged_summary_op = tf.summary.merge_all()

        print("Start Training!")
        accu_total = 0
        loss_avg = 0
        for i in range(epochs):
            s=randint(0,210)

            symbols_inputs = [dictionary[str(X_train[s+j])] for j in range(n_input
            keys = np.reshape(np.array(symbols_inputs), [-1, n_input, 1])
            symbols_labels = np.zeros([vocabulary_size], dtype = float)
            symbols_labels[dictionary[str(X_train[s+3])]] = 1.0
            symbols_labels = np.reshape(symbols_labels,[1,-1])

            _, accu, loss, onehot_pred = sess.run([optimizer, accuracy, cost, pred

            loss_avg+=loss
            accu_total+= accu
            if  (i+1) % display_step == 0:
                print("Epoch: ", '%02d' % (i+1), "  =====> Loss=", "{:.9f}".format
                print("Training Accuracy = {:.3f}".format(((accu_total)*100)/displ
                accu_total=0
                loss_avg

        print("End Of training Finished!")
        print("time: ",time.time() - start_time)
        print("For tensorboard visualisation run on command line.")
        print("\ttensorboard --logdir=%s" % (logs_path))
        print("and oint your web browser to the returned link")
        #############################################
        #save your model
        model_saver.save(sess, 'lstm_model/lstm_model2')
        #############################################
        print("Model saved")
```

In [12]:
```
train2(train_data)
```

```
Start Training!
('Epoch: ', '1000', '  =====> Loss=', '4.660363439')
Training Accuracy = 7.000
('Epoch: ', '2000', '  =====> Loss=', '8.668767819')
Training Accuracy = 12.600
('Epoch: ', '3000', '  =====> Loss=', '12.245523591')
Training Accuracy = 18.300
('Epoch: ', '4000', '  =====> Loss=', '15.222792344')
Training Accuracy = 27.000
('Epoch: ', '5000', '  =====> Loss=', '17.810685085')
Training Accuracy = 37.600
('Epoch: ', '6000', '  =====> Loss=', '20.207550686')
Training Accuracy = 39.700
('Epoch: ', '7000', '  =====> Loss=', '22.151088975')
Training Accuracy = 49.200
('Epoch: ', '8000', '  =====> Loss=', '23.773269892')
Training Accuracy = 57.100
('Epoch: ', '9000', '  =====> Loss=', '25.211266780')
Training Accuracy = 59.800
('Epoch: ', '10000', '  =====> Loss=', '26.431921797')
Training Accuracy = 64.600
('Epoch: ', '11000', '  =====> Loss=', '27.523224625')
Training Accuracy = 68.800
('Epoch: ', '12000', '  =====> Loss=', '28.394510648')
Training Accuracy = 74.800
('Epoch: ', '13000', '  =====> Loss=', '29.200298452')
Training Accuracy = 75.000
('Epoch: ', '14000', '  =====> Loss=', '29.916543670')
Training Accuracy = 77.300
('Epoch: ', '15000', '  =====> Loss=', '30.540841901')
Training Accuracy = 80.100
('Epoch: ', '16000', '  =====> Loss=', '31.107804043')
Training Accuracy = 83.600
('Epoch: ', '17000', '  =====> Loss=', '31.637995896')
Training Accuracy = 83.200
('Epoch: ', '18000', '  =====> Loss=', '32.127022933')
Training Accuracy = 84.700
('Epoch: ', '19000', '  =====> Loss=', '32.589656569')
Training Accuracy = 85.200
('Epoch: ', '20000', '  =====> Loss=', '33.012038801')
Training Accuracy = 86.200
('Epoch: ', '21000', '  =====> Loss=', '33.406860528')
Training Accuracy = 86.700
('Epoch: ', '22000', '  =====> Loss=', '33.767980725')
Training Accuracy = 88.600
('Epoch: ', '23000', '  =====> Loss=', '34.120883907')
Training Accuracy = 89.000
('Epoch: ', '24000', '  =====> Loss=', '34.444430765')
Training Accuracy = 89.300
('Epoch: ', '25000', '  =====> Loss=', '34.745559821')
Training Accuracy = 91.200
('Epoch: ', '26000', '  =====> Loss=', '35.060838024')
Training Accuracy = 89.700
('Epoch: ', '27000', '  =====> Loss=', '35.330708822')
Training Accuracy = 90.800
('Epoch: ', '28000', '  =====> Loss=', '35.630337522')
Training Accuracy = 91.100
('Epoch: ', '29000', '  =====> Loss=', '35.874604350')
Training Accuracy = 92.300
('Epoch: ', '30000', '  =====> Loss=', '36.129672959')
Training Accuracy = 91.200
```

Part 4 : Test your model

3.1. Next word prediction

Load your model (using the model_saved variable given in the training session) and test the sentences :

- 'get a little'
- 'nobody tried to'
- Try with other sentences using words from the stroy's vocabulary.

In [20]:

```
with tf.Session() as sess:
  # Restore variables from disk.
    model_saver.restore(sess, 'lstm_model/lstm_model1')
    print ('model loaded')
  # Do some work with the model
    sentence1 = 'get a little'
    sentence2 = 'nobody tried to'
    sentence3 = 'boy of course'
    sentence4 = 'a good meal'
    sentence5 = 'Hello I am'
    sentence6 = 'again the villagers'
    sentence7= ' again the citizens'
    test(sentence1, sess, verbose= True)
    test(sentence2, sess, verbose= True)
    test(sentence3, sess, verbose= True)
    test(sentence4, sess, verbose= True)
    test(sentence5, sess, verbose= True)
    test(sentence6, sess, verbose= True)
    test(sentence7, sess, verbose= True)
```

```
model loaded
get a little days
nobody tried to come
boy of course cried
a good meal off
Word Hello not in dictionary
again the villagers ,
Word citizens not in dictionary
```

In [21]:
```
with tf.Session() as sess:
  # Restore variables from disk.
    model_saver.restore(sess, 'lstm_model/lstm_model2')
    print ('model loaded')
  # Do some work with the model
    sentence1 = 'get a little'
    sentence2 = 'nobody tried to'
    sentence3 = 'boy of course'
    sentence4 = 'a good meal'
    sentence5 = 'Hello I am'
    sentence6 = 'again the villagers'
    sentence7= ' again the citizens'
    test(sentence1, sess, verbose= True)
    test(sentence2, sess, verbose= True)
    test(sentence3, sess, verbose= True)
    test(sentence4, sess, verbose= True)
    test(sentence5, sess, verbose= True)
    test(sentence6, sess, verbose= True)
    test(sentence7, sess, verbose= True)
```

```
model loaded
get a little company
nobody tried to come
boy of course cried
a good meal off
Word Hello not in dictionary
again the villagers came
Word citizens not in dictionary
```

3.2. More fun with the Fable Writer !

You will use the RNN/LSTM model learned in the previous question to create a new story/fable. For this you will choose 3 words from the dictionary which will start your story and initialize your network. Using those 3 words the RNN will generate the next word or the story. Using the last 3 words (the newly predicted one and the last 2 from the input) you will use the network to predict the 5 word of the story.. and so on until your story is 5 sentence long. Make a point at the end of your story. To implement that, you will use the test function.

In [75]:
```
def Create_new_story1(X_train):
    with tf.Session() as sess:
        model_saver.restore(sess, 'lstm_model/lstm_model2')
        story=[]
        sentence = 'a liar once'
        for i in range(100) :
            word = test(sentence, sess, verbose= False)
            sen = sentence + ' ' + word
            sen= sen.split()
            sentence = sen[1]+' '+sen[2]+' '+sen[3]
            story.append(sentence)

        L = [j.split()[2] for j in story]
        A=''
        P=0
        for k in range(len(L)) :
            if L[k] == '.' :
                P+=1
                A = A +L[k]+' '
                if P== 5 :
                    break
            else :
                A = A +L[k]+' '
        print('A liar once'+ ' ' + A + ' ')
```

In [76]:
```
Create_new_story1(train_data)
```

```
A liar once not this a wolf actually did come out from the forest , and began sti
rred to come to his help . so the wolf made a good meal off the boy's flock , and
again the villagers came to his help . so the wolf made a good meal off the boy's
flock , and again the villagers came to his help . so the wolf made a good meal o
ff the boy's flock , and again the villagers came to his help . so the wolf made
a good meal off the boy's flock , and again the villagers .
```

In [13]:
```
def Create_new_story2(X_train):
    with tf.Session() as sess:
        model_saver.restore(sess, 'lstm_model/lstm_model1')
        story=[]
        sentence = 'a liar once'
        for i in range(200) :
            word = test(sentence, sess, verbose= False)
            sen = sentence + ' ' + word
            sen= sen.split()
            sentence = sen[1]+' '+sen[2]+' '+sen[3]
            story.append(sentence)

        L = [j.split()[2] for j in story]
        A=''
        P=0
        for k in range(len(L)) :
            if L[k] == '.' :
                P+=1
                A = A +L[k]+' '
                if P== 5 :
                    break
            else :
                A = A +L[k]+' '
        print('A liar once'+ ' ' + A + ' ')
```

In [14]:
```
Create_new_story2(train_data)
```

```
A liar once not before , thought the boy complained , the wise man of the village
said : a liar will not be believed , boy's flock he tried the same flock , and wh
en the boy complained , the wise man of the village said : a liar will not be bel
ieved , boy's flock he tried the same flock , and when the boy complained , the w
ise man of the village said : a liar will not be believed , boy's flock he tried
the same flock , and when the boy complained , the wise man of the village said :
a liar will not be believed , boy's flock he tried the same flock , and when the
boy complained , the wise man of the village said : a liar will not be believed ,
boy's flock he tried the same flock , and when the boy complained , the wise man
of the village said : a liar will not be believed , boy's flock he tried the same
flock , and when the boy complained , the wise man of the village said : a liar w
ill not be believed , boy's flock he .
```

3.3. Play with number of inputs

The number of input in our example is 3, see what happens when you use other number (1 and 5)

Let's use 5 in the number of inputs instead of 3 :

In [6]:
```
learning_rate = 0.001
epochs = 50000
display_step = 1000
n_input = 5

#For each LSTM cell that you initialise, supply a value for the hidden dimension,
n_hidden = 64

# tf Graph input
x = tf.placeholder("float", [None, n_input, 1])
y = tf.placeholder("float", [None, vocabulary_size])

# LSTM  weights and biases
weights = { 'out': tf.Variable(tf.random_normal([n_hidden, vocabulary_size]))}
biases = {'out': tf.Variable(tf.random_normal([vocabulary_size])) }


#build the model
pred = lstm_model(x, weights['out'], biases['out'],n_hidden,n_input)
```

In [10]:
```
# Initializing the variables
from random import randint
start_time = time.time()
init = tf.global_variables_initializer()
model_saver = tf.train.Saver()
logs_path = 'lstm_model'
def train_with_5(X_train):
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        summary_writer = tf.summary.FileWriter(logs_path, graph=tf.get_default_gra
        merged_summary_op = tf.summary.merge_all()

        print("Start Training!")
        accu_total = 0
        loss_avg = 0
        for i in range(epochs):
            s=randint(0,208)

            symbols_inputs = [dictionary[str(X_train[s+j])] for j in range(n_input
            keys = np.reshape(np.array(symbols_inputs), [-1, n_input, 1])
            symbols_labels = np.zeros([vocabulary_size], dtype = float)
            symbols_labels[dictionary[str(X_train[s+n_input])]] = 1.0
            symbols_labels = np.reshape(symbols_labels,[1,-1])

            _, accu, loss, onehot_pred = sess.run([optimizer, accuracy, cost, pred

            loss_avg+=loss
            accu_total+= accu
            if  (i+1) % display_step == 0:
                print("Epoch: ", '%02d' % (i+1), "  =====> Loss=", "{:.9f}".format
                print("Training Accuracy = {:.3f}".format(((accu_total)*100)/displ
                accu_total=0
                loss_avg=0

        print("End Of training Finished!")
        print("time: ",time.time() - start_time)
        print("For tensorboard visualisation run on command line.")
        print("\ttensorboard --logdir=%s" % (logs_path))
        print("and oint your web browser to the returned link")
        #############################################
        #save your model
        model_saver.save(sess, 'lstm_model1')
        #############################################
        print("Model saved")
        sentence1 = 'get a little company and'
        sentence2 = 'nobody tried to boy of'
        sentence3 = 'boy of course cried out'
        sentence4 = 'a good meal off out'
        test(sentence1, sess, verbose= True)
        test(sentence2, sess, verbose= True)
        test(sentence3, sess, verbose= True)
        test(sentence4, sess, verbose= True)
```

In [11]:
```
train_with_5(train_data)
```

```
Start Training!
('Epoch: ', '1000', '  =====> Loss=', '4.525358202')
Training Accuracy = 6.900
('Epoch: ', '2000', '  =====> Loss=', '3.626678368')
Training Accuracy = 17.300
('Epoch: ', '3000', '  =====> Loss=', '2.927230589')
Training Accuracy = 29.800
('Epoch: ', '4000', '  =====> Loss=', '2.220429839')
Training Accuracy = 44.600
('Epoch: ', '5000', '  =====> Loss=', '1.426709384')
Training Accuracy = 63.800
('Epoch: ', '6000', '  =====> Loss=', '0.924393820')
Training Accuracy = 76.400
('Epoch: ', '7000', '  =====> Loss=', '0.637371989')
Training Accuracy = 84.600
('Epoch: ', '8000', '  =====> Loss=', '0.375112680')
Training Accuracy = 90.900
('Epoch: ', '9000', '  =====> Loss=', '0.252431412')
Training Accuracy = 93.900
('Epoch: ', '10000', '  =====> Loss=', '0.194574602')
Training Accuracy = 94.800
('Epoch: ', '11000', '  =====> Loss=', '0.121999625')
Training Accuracy = 97.500
('Epoch: ', '12000', '  =====> Loss=', '0.117694613')
Training Accuracy = 97.100
('Epoch: ', '13000', '  =====> Loss=', '0.089301159')
Training Accuracy = 97.900
('Epoch: ', '14000', '  =====> Loss=', '0.081950293')
Training Accuracy = 97.800
('Epoch: ', '15000', '  =====> Loss=', '0.064785022')
Training Accuracy = 98.400
('Epoch: ', '16000', '  =====> Loss=', '0.076222239')
Training Accuracy = 97.900
('Epoch: ', '17000', '  =====> Loss=', '0.066566939')
Training Accuracy = 98.000
('Epoch: ', '18000', '  =====> Loss=', '0.090814275')
Training Accuracy = 97.400
('Epoch: ', '19000', '  =====> Loss=', '0.054920317')
Training Accuracy = 98.100
('Epoch: ', '20000', '  =====> Loss=', '0.040538797')
Training Accuracy = 99.100
('Epoch: ', '21000', '  =====> Loss=', '0.059510074')
Training Accuracy = 98.600
('Epoch: ', '22000', '  =====> Loss=', '0.059543151')
Training Accuracy = 98.600
('Epoch: ', '23000', '  =====> Loss=', '0.026206670')
Training Accuracy = 99.400
('Epoch: ', '24000', '  =====> Loss=', '0.055494912')
Training Accuracy = 98.600
('Epoch: ', '25000', '  =====> Loss=', '0.071911419')
Training Accuracy = 98.600
('Epoch: ', '26000', '  =====> Loss=', '0.054016226')
Training Accuracy = 98.500
('Epoch: ', '27000', '  =====> Loss=', '0.069126539')
Training Accuracy = 98.200
('Epoch: ', '28000', '  =====> Loss=', '0.060412739')
Training Accuracy = 98.400
('Epoch: ', '29000', '  =====> Loss=', '0.056403736')
Training Accuracy = 98.600
('Epoch: ', '30000', '  =====> Loss=', '0.058089099')
Training Accuracy = 98.500
```

Let's use 1 in the number of inputs :

In [6]:
```
learning_rate = 0.001
epochs = 50000
display_step = 1000
n_input = 1

#For each LSTM cell that you initialise, supply a value for the hidden dimension,
n_hidden = 64

# tf Graph input
x = tf.placeholder("float", [None, n_input, 1])
y = tf.placeholder("float", [None, vocabulary_size])

# LSTM  weights and biases
weights = { 'out': tf.Variable(tf.random_normal([n_hidden, vocabulary_size]))}
biases = {'out': tf.Variable(tf.random_normal([vocabulary_size])) }


#build the model
pred = lstm_model(x, weights['out'], biases['out'],n_hidden,n_input)
```

In [10]:
```
# Initializing the variables
from random import randint
start_time = time.time()
init = tf.global_variables_initializer()
model_saver = tf.train.Saver()
logs_path = 'lstm_model'
def train_with_1(X_train):
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        summary_writer = tf.summary.FileWriter(logs_path, graph=tf.get_default_gra
        merged_summary_op = tf.summary.merge_all()

        print("Start Training!")
        accu_total = 0
        loss_avg = 0
        for i in range(epochs):
            s=randint(0,212)

            symbols_inputs = [dictionary[str(X_train[s+j])] for j in range(n_input
            keys = np.reshape(np.array(symbols_inputs), [-1, n_input, 1])
            symbols_labels = np.zeros([vocabulary_size], dtype = float)
            symbols_labels[dictionary[str(X_train[s+n_input])]] = 1.0
            symbols_labels = np.reshape(symbols_labels,[1,-1])

            _, accu, loss, onehot_pred = sess.run([optimizer, accuracy, cost, pred

            loss_avg+=loss
            accu_total+= accu
            if  (i+1) % display_step == 0:
                print("Epoch: ", '%02d' % (i+1), "  =====> Loss=", "{:.9f}".format
                print("Training Accuracy = {:.3f}".format((accu_total)*100)/displ
                accu_total=0
                loss_avg=0

        print("End Of training Finished!")
        print("time: ",time.time() - start_time)
        print("For tensorboard visualisation run on command line.")
        print("\ttensorboard --logdir=%s" % (logs_path))
        print("and oint your web browser to the returned link")
        ##############################################
        #save your model
        model_saver.save(sess, 'lstm_model1')
        ##############################################
        print("Model saved")
        sentence1 = 'get'
        sentence2 = 'nobody'
        sentence3 = 'boy'
        sentence4 = 'good'
        test(sentence1, sess, verbose= True)
        test(sentence2, sess, verbose= True)
        test(sentence3, sess, verbose= True)
        test(sentence4, sess, verbose= True)
```

In [11]:
```
train_with_1(train_data)
```

```
Start Training!
('Epoch: ', '1000', '  =====> Loss=', '4.660609530')
Training Accuracy = 5.300
('Epoch: ', '2000', '  =====> Loss=', '4.230867089')
Training Accuracy = 10.600
('Epoch: ', '3000', '  =====> Loss=', '4.218557065')
Training Accuracy = 9.000
('Epoch: ', '4000', '  =====> Loss=', '4.053722510')
Training Accuracy = 12.100
('Epoch: ', '5000', '  =====> Loss=', '4.073025993')
Training Accuracy = 12.400
('Epoch: ', '6000', '  =====> Loss=', '3.956920830')
Training Accuracy = 15.100
('Epoch: ', '7000', '  =====> Loss=', '3.986122457')
Training Accuracy = 12.800
('Epoch: ', '8000', '  =====> Loss=', '4.024381949')
Training Accuracy = 12.300
('Epoch: ', '9000', '  =====> Loss=', '4.022598642')
Training Accuracy = 12.600
('Epoch: ', '10000', '  =====> Loss=', '4.042164472')
Training Accuracy = 12.500
('Epoch: ', '11000', '  =====> Loss=', '3.938985136')
Training Accuracy = 13.600
('Epoch: ', '12000', '  =====> Loss=', '4.009429325')
Training Accuracy = 12.000
('Epoch: ', '13000', '  =====> Loss=', '4.047444021')
Training Accuracy = 12.800
('Epoch: ', '14000', '  =====> Loss=', '4.120118496')
Training Accuracy = 11.200
('Epoch: ', '15000', '  =====> Loss=', '4.011419585')
Training Accuracy = 13.000
('Epoch: ', '16000', '  =====> Loss=', '4.099645760')
Training Accuracy = 11.800
('Epoch: ', '17000', '  =====> Loss=', '3.985820529')
Training Accuracy = 11.400
('Epoch: ', '18000', '  =====> Loss=', '4.012488915')
Training Accuracy = 14.800
('Epoch: ', '19000', '  =====> Loss=', '4.103252109')
Training Accuracy = 13.300
('Epoch: ', '20000', '  =====> Loss=', '4.106827055')
Training Accuracy = 13.900
('Epoch: ', '21000', '  =====> Loss=', '4.016571291')
Training Accuracy = 13.700
('Epoch: ', '22000', '  =====> Loss=', '3.984344057')
Training Accuracy = 15.900
('Epoch: ', '23000', '  =====> Loss=', '4.097278606')
Training Accuracy = 13.500
('Epoch: ', '24000', '  =====> Loss=', '4.025873984')
Training Accuracy = 16.400
('Epoch: ', '25000', '  =====> Loss=', '4.063275601')
Training Accuracy = 14.200
('Epoch: ', '26000', '  =====> Loss=', '4.073919671')
Training Accuracy = 13.000
('Epoch: ', '27000', '  =====> Loss=', '3.919146384')
Training Accuracy = 19.000
('Epoch: ', '28000', '  =====> Loss=', '4.080464022')
Training Accuracy = 13.600
('Epoch: ', '29000', '  =====> Loss=', '4.124263611')
Training Accuracy = 15.500
('Epoch: ', '30000', '  =====> Loss=', '4.047705804')
Training Accuracy = 15.700
```

Answer : We've noticed that the training accuracy has dropped drastically when we used only one word in the number of inputs, while it has been improved when we've used five words. It seems logical since the more the sentences we give to the model to train on are long, the more the model get used to the exact expressions and structure of the fable, and the more the model tend to overfit on the training data, hence improving the training accuracy.