Long Short Term Memory (LSTM) in a Recurrent Neural Network (RNN) for Language Modeling

Deep Learning

Luca BENEDETTO | Alberto IBARRONDO

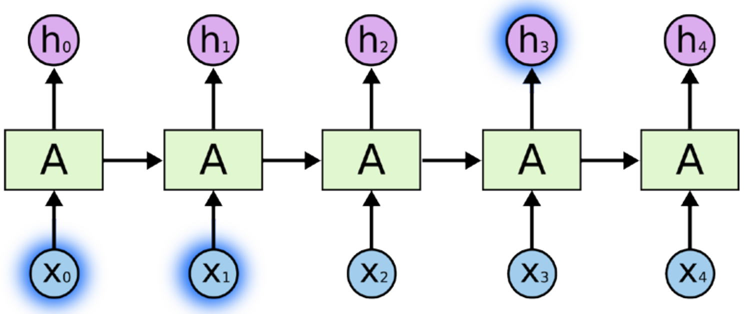
07/06/2017

Summary

This notebook builds and trains a Recurrent Neural Network, based on Long Short-Term Memory (LSTM) units for next word prediction task.

1. Introduction

We 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 dictionary. 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".

We use a sample short story from Aesop’s Fables (http://www.taleswithmorals.com/) to train the 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".

1.1 Loading libraries

We start by loading the necessary libraries and resetting the default computational graph. For more details about the rnn packages we import, we suggest 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)
```

1.2. Loading data

Let's 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("Length training data: %d" %len(train_data))
print("Number of unique elements: %d" %len(set(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...
Length training data: 214
Number of unique elements: 113
```

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
```

Let's display the vocabulary

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

```
Vocabulary_size =  113


Dictionary : 

{'all': 32, 'liar': 33, 'help': 17, 'cried': 34, 'course': 35, 'still': 36, 'pleased'
: 37, 'before': 18, 'excitement': 91, 'deceiving': 38, 'had': 39, 'young': 69, 'actua
lly': 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, 'lit
tle': 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, 'with': 87, 'than': 43, 'he': 8, 'made': 89,
'wise': 90, 'this': 14, 'will': 71, 'near': 92, 'believed': 93, 'meet': 94, 'and': 3,
 'it': 95, 'his': 15, 'at': 96, 'worry': 97, 'again': 25, 'considerable': 88, 'rather
': 98, 'began': 99, 'when': 26, 'same': 101, 'forest': 27, 'which': 102, 'speaks': 10
3, 'towards': 104, 'tried': 105, 'mountain': 106, 'who': 28, 'upon': 107, 'plan': 108
, 'man': 109, 'complained': 82, 'stirred': 110, 'off': 100, 'foot': 46, 'shortly': 11
1, '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: 'his', 16:
'so', 17: 'help', 18: 'before', 19: 'them', 20: 'some', 21: 'for', 22: 'come', 23: 'v
illage', 24: 'but', 25: 'again', 26: 'when', 27: 'forest', 28: 'who', 29: 'thought',
30: 'time', 31: 'came', 32: 'all', 33: 'liar', 34: 'cried', 35: 'course', 36: 'still'
, 37: 'pleased', 38: 'deceiving', 39: 'had', 40: 'actually', 41: 'shepherd', 42: 'lon
ely', 43: 'than', 44: 'get', 45: 'dark', 46: 'foot', 47: 'day', 48: 'did', 49: 'calli
ng', 50: 'twice', 51: 'good', 52: 'stopped', 53: 'truth', 54: 'meal', 55: 'sheep,', 5
6: '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: 'y
oung', 70: 'down', 71: 'will', 72: 'sheep', 73: 'little', 74: 'from', 75: 'rushed', 7
6: 'there', 77: 'been', 78: 'few', 79: 'much', 80: "boy's", 81: ':', 82: 'complained'
, 83: 'that', 84: 'company', 85: 'nobody', 86: 'fooled', 87: 'with', 88: 'considerabl
e', 89: 'made', 90: 'wise', 91: 'excitement', 92: 'near', 93: 'believed', 94: 'meet',
 95: 'it', 96: 'at', 97: 'worry', 98: 'rather', 99: 'began', 100: 'off', 101: 'same',
 102: 'which', 103: 'speaks', 104: 'towards', 105: 'tried', 106: 'mountain', 107: 'up
on', 108: 'plan', 109: 'man', 110: 'stirred', 111: 'shortly', 112: 'once'}
```

2. LSTM Model in TensorFlow

We are now to develop an LSTM model to predict the word of following a sequence of 3 words.

2.1. Model definition

We are defining a 2-layers LSTM model

In [1]:
```
def LSTMModel(x, n_input, weights, biases):

    # reshape to [1, n_input]
    x = tf.reshape(x, [-1, n_input])

    # Generate a n_input-element sequence of inputs
    # (eg. [had] [a] [general] -> [20] [6] [33])
    x = tf.split(x,n_input,1)

    # 1-layer LSTM with n_hidden units.
    rnn_cell = rnn.BasicLSTMCell(n_hidden)

    # 2-layer LSTM with n_hidden units.
    rnn_cell = rnn.MultiRNNCell([rnn_cell]*2)

    # generate prediction
    outputs, states = rnn.static_rnn(rnn_cell, x, dtype=tf.float32)

    # there are n_input outputs but
    # we only want the last output
    return tf.matmul(outputs[-1], weights['out']) + biases['out']
```

2.2 Training Parameters and Graph creation

In [7]:
```
# 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, num
ber of units in LSTM cell
n_hidden = 64
```

2.3 Graph creation

In [8]:
```
# LSTM  weights and biases
weights = { 'out': tf.Variable(tf.random_normal([n_hidden, vocabulary_size]))}
biases = {'out': tf.Variable(tf.random_normal([vocabulary_size])) }

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

#build the model
```

```
pred = RNN(x, n_input, weights, biases)

# Loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
optimizer = tf.train.RMSPropOptimizer(learning_rate=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))
```

2.4 Test Function

In [3]:

```
def LSTMtest(sentence, session, verbose=False):
    sentence = sentence.strip()
    words = sentence.split(' ')
    if len(words) != n_input:
        print("sentence length should be equal to", n_input, "!")
    try:
        symbols_inputs = [dictionary[str(words[i - n_input])] for i in range(n_input)
]
        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(["Word", words[i - n_input], "not in dictionary"])
```

3. Training the LSTM

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. We display the loss and the training accuracy every 1000 iteration and save the model at the end of training in the **lstm_model** folder

In [11]:

```
model_saver = tf.train.Saver()
```

In [14]:

```
def LSTMTrain(name):
    # Initializing the variables
    start_time = time.time()
    init = tf.global_variables_initializer()

    print("Start Training")
    ##############################################

    with tf.Session() as session:
        session.run(init)
        step = 0
        offset = random.randint(0,n_input+1)
        end_offset = n_input + 1
        acc_total = 0
        loss_total = 0

        writer.add_graph(session.graph)

        while step < epochs:
            # Generate a minibatch. Add some randomness on selection process.
            if offset > (len(train_data) - end_offset):
                offset = random.randint(0, n_input+1)

            symbols_in_keys = [ [dictionary[ str(train_data[i])]] for i in range(offs
et, offset+n_input) ]
            symbols_in_keys = np.reshape(np.array(symbols_in_keys), [-1, n_input, 1])

            symbols_out_onehot = np.zeros([vocabulary_size], dtype=float)
            symbols_out_onehot[dictionary[str(train_data[offset+n_input])]] = 1.0
            symbols_out_onehot = np.reshape(symbols_out_onehot,[1,-1])

            _, acc, loss, onehot_pred = session.run([optimizer, accuracy, cost, pred]
, \
                                                    feed_dict={x: symbols_in_keys, y:
 symbols_out_onehot})
            loss_total += loss
            acc_total += acc

            if (step+1) % display_step == 0:

                symbols_in = [train_data[i] for i in range(offset, offset + n_input)]
                symbols_out = train_data[offset + n_input]
                symbols_out_pred = reverse_dictionary[int(tf.argmax(onehot_pred, 1).e
val())]

                print("Iter= " + str(step+1) + ", Loss= " + \
                      "{:.6f}".format(loss_total/display_step) + ", TrAcc= " + \
                      "{:.2f}%".format(100*acc_total/display_step)),
                print("%s - [%s] vs [%s]" % (symbols_in,symbols_out,symbols_out_pred)
)

                acc_total = 0
                loss_total = 0

            step += 1
            offset += (n_input+1)

    ##############################################
        print("Optimization Finished!")
        print("Elapsed time:%.1f s" %(time.time() - start_time))
        print("Run on command line: tensorboard --logdir=%s" % (logs_path))
        print("Point your web browser to the returned link")
    ##############################################
        model_saver.save(sess=session, save_path='lstm_model/'+name)
    ##############################################
        print("Model saved")
```

In [16]:

```
LSTMTrainModel('LSTMmodel_3')
```

```
Start Training
Iter= 1000, Loss= 4.619733, TrAcc= 6.50% ['speaks', 'the', 'truth'] - [.] vs [he]
Iter= 2000, Loss= 3.736615, TrAcc= 13.80% ['he', 'speaks', 'the'] - [truth] vs [fores
t]
Iter= 3000, Loss= 3.441579, TrAcc= 18.10% ['a', 'liar', 'will'] - [not] vs [:]
Iter= 4000, Loss= 2.886903, TrAcc= 32.50% ['believed', ',', 'even'] - [when] vs [shep
herd]
Iter= 5000, Loss= 2.466733, TrAcc= 39.40% ['a', 'liar', 'will'] - [not] vs [:]
Iter= 6000, Loss= 2.355309, TrAcc= 41.70% ['be', 'believed', ','] - [even] vs [boy's]
Iter= 7000, Loss= 1.837935, TrAcc= 54.80% ['the', 'wise', 'man'] - [of] vs [of]
Iter= 8000, Loss= 1.723356, TrAcc= 55.60% [',', 'and', 'when'] - [the] vs [of]
Iter= 9000, Loss= 1.680242, TrAcc= 55.10% [',', 'the', 'wise'] - [man] vs [man]
Iter= 10000, Loss= 1.190535, TrAcc= 68.40% ['a', 'liar', 'will'] - [not] vs [not]
Iter= 11000, Loss= 1.245663, TrAcc= 65.70% ['young', 'shepherd', 'boy'] - [who] vs [w
ho]
Iter= 12000, Loss= 0.923475, TrAcc= 76.30% ['once', 'a', 'young'] - [shepherd] vs [da
ys]
Iter= 13000, Loss= 1.011464, TrAcc= 69.50% ['the', 'village', 'said'] - [:] vs [:]
Iter= 14000, Loss= 0.869120, TrAcc= 76.90% ['will', 'not', 'be'] - [believed] vs [he]
Iter= 15000, Loss= 0.931696, TrAcc= 73.40% ['man', 'of', 'the'] - [village] vs [villa
ge]
Iter= 16000, Loss= 0.627505, TrAcc= 83.00% ['man', 'of', 'the'] - [village] vs [villa
ge]
Iter= 17000, Loss= 0.772526, TrAcc= 78.30% ['said', ':', 'a'] - [liar] vs [liar]
Iter= 18000, Loss= 0.460625, TrAcc= 87.60% ['not', 'be', 'believed'] - [,] vs [the]
Iter= 19000, Loss= 0.551270, TrAcc= 85.00% ['the', 'boy', 'complained'] - [,] vs [,]
Iter= 20000, Loss= 0.542769, TrAcc= 86.90% ['said', ':', 'a'] - [liar] vs [liar]
Iter= 21000, Loss= 0.465838, TrAcc= 88.20% ['a', 'liar', 'will'] - [not] vs [not]
Iter= 22000, Loss= 0.584257, TrAcc= 84.10% ['believed', ',', 'even'] - [when] vs [whe
n]
Iter= 23000, Loss= 0.374317, TrAcc= 89.60% [':', 'a', 'liar'] - [will] vs [will]
Iter= 24000, Loss= 0.528858, TrAcc= 86.20% ['liar', 'will', 'not'] - [be] vs [be]
Iter= 25000, Loss= 0.433693, TrAcc= 86.40% ['a', 'liar', 'will'] - [not] vs [not]
Iter= 26000, Loss= 0.397060, TrAcc= 87.90% ['of', 'the', 'village'] - [said] vs [said
]
Iter= 27000, Loss= 0.397318, TrAcc= 89.00% ['village', 'said', ':'] - [a] vs [a]
Iter= 28000, Loss= 0.273457, TrAcc= 92.70% ['young', 'shepherd', 'boy'] - [who] vs [w
ho]
Iter= 29000, Loss= 0.453159, TrAcc= 88.60% ['of', 'a', 'mountain'] - [near] vs [near]
Iter= 30000, Loss= 0.367259, TrAcc= 90.70% ['him', 'all', 'day'] - [,] vs [,]
Iter= 31000, Loss= 0.397182, TrAcc= 89.40% ['rather', 'lonely', 'for'] - [him] vs [hi
m]
Iter= 32000, Loss= 0.398924, TrAcc= 90.00% ['foot', 'of', 'a'] - [mountain] vs [mount
ain]
Iter= 33000, Loss= 0.325172, TrAcc= 89.00% ['all', 'day', ','] - [so] vs [so]
Iter= 34000, Loss= 0.317205, TrAcc= 90.00% ['dark', 'forest', '.'] - [it] vs [it]
Iter= 35000, Loss= 0.284799, TrAcc= 91.20% ['all', 'day', ','] - [so] vs [so]
Iter= 36000, Loss= 0.249984, TrAcc= 92.60% ['all', 'day', ','] - [so] vs [so]
Iter= 37000, Loss= 0.281406, TrAcc= 90.30% ['lonely', 'for', 'him'] - [all] vs [all]
Iter= 38000, Loss= 0.233675, TrAcc= 93.20% ['rather', 'lonely', 'for'] - [him] vs [hi
m]
Iter= 39000, Loss= 0.328268, TrAcc= 91.10% ['.', 'it', 'was'] - [rather] vs [rather]
Iter= 40000, Loss= 0.306602, TrAcc= 89.40% ['sheep', 'at', 'the'] - [foot] vs [foot]
Iter= 41000, Loss= 0.261380, TrAcc= 92.60% ['forest', '.', 'it'] - [was] vs [was]
Iter= 42000, Loss= 0.180892, TrAcc= 95.30% ['he', 'thought', 'upon'] - [a] vs [a]
Iter= 43000, Loss= 0.277204, TrAcc= 94.90% [',', 'so', 'he'] - [thought] vs [thought]
Iter= 44000, Loss= 0.344732, TrAcc= 91.20% ['day', ',', 'so'] - [he] vs [he]
```

```
Iter= 45000, Loss= 0.268094, TrAcc= 91.50% ['thought', 'upon', 'a'] - [plan] vs [plan
]
Iter= 46000, Loss= 0.227615, TrAcc= 92.40% ['could', 'get', 'a'] - [little] vs [littl
e]
Iter= 47000, Loss= 0.193860, TrAcc= 94.50% ['towards', 'the', 'village'] - [calling]
vs [calling]
Iter= 48000, Loss= 0.308691, TrAcc= 90.50% ['down', 'towards', 'the'] - [village] vs
[boy's]
Iter= 49000, Loss= 0.211860, TrAcc= 94.60% ['to', 'meet', 'him'] - [,] vs [,]
Iter= 50000, Loss= 0.196414, TrAcc= 93.70% ['him', ',', 'and'] - [some] vs [some]
Optimization Finished!
Elapsed time:67.5 s
Run on command line: tensorboard --logdir=lstm_words
Point your web browser to the returned link
Model saved
```

4 Testing the RNN

4.1 Next word prediction

We load the model (using the model_saved variable given in the training session) and test the sentences :

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

In [17]:
```
with tf.Session() as session:
    # Restore variables from disk.
    model_saver.restore(session, "lstm_model/LSTMmodel_3")
    print("Model restored.")
    print("\nTesting the model")
    test_sentences = ['get a little', 'nobody tried to', 'he rushed down', 'a wolf ac
tually', 'a liar will']
    for sentence in test_sentences:
        test(sentence, session, verbose=True)
```

```
Model restored.

Testing the model
get a little company
nobody tried to come
he rushed down towards
a wolf actually did
a liar will not
```

4.2 More fun with the Story Writer

Let's use the RNN/LSTM model learned in the previous question to create a new story/fable. For this we choose 3 words from the dictionary which will start the story and initialize the 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) we will use the network to predict the 5 word of the story... and so on until your story is 5 sentences long.

In [34]:
```
def RNNcreate_story(sentence, session, numOfSentences=5, n_input=3, verbose=False):
    cnt_sent = 0
    end_sent = ['.',',',':',';','!','?']

    sentence = sentence.strip()
    words = sentence.split(' ')
    while cnt_sent < numOfSentences or (cnt_sent == numOfSentences and sentence[-1] n
ot in ['.','!','?']):
        symbols_inputs = [dictionary[str(words[i - n_input])] for i in range(n_input)
]
        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())
        new_word = reverse_dictionary[onehot_pred_index]
        words.append(new_word)
        sentence += " " + words[-1]
        words = words[-n_input:]
        if verbose:
            print(sentence)
        if new_word in end_sent:
            cnt_sent += 1
    print(sentence)
```

In [19]:
```
#Your implementation goes here
with tf.Session() as session:
    # Restore variables from disk.
    model_saver.restore(session, "lstm_model/LSTMmodel_3")
    print("Model restored.\n")
    RNNcreate_story('a wolf actually', session, numOfSentences=5)
```

```
Model restored.

a wolf actually did come out from the forest , and began stirred to come to his help
. so the wolf made a good will again a this pleased the boy so much that a few days a
fterwards he tried the truth of a mountain near a dark will not twice before . but th
e villagers came out to meet him , and some of them stopped with him for a considerab
le time .
```

4.3 Playing with number of inputs

The number of input in our example is 3, now we are gonna see what happens when we use other number (1, 2 and 5)

In [45]:
```
tf.reset_default_graph()

# Training Parameters
learning_rate = 0.001
epochs = 50000
display_step = 1000
n_input = 1
n_hidden = 64
```

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

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

pred = RNNModel(x, n_input, weights, biases)

# Loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
optimizer = tf.train.RMSPropOptimizer(learning_rate=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))

model_saver = tf.train.Saver()
```

In [24]:
```
RNNTrain('LSTMmodel_1')
```

```
Start Training
Iter= 1000, Loss= 4.519836, TrAcc= 6.10% ['time'] - [.] vs [.]
Iter= 2000, Loss= 4.327969, TrAcc= 8.20% ['stirred'] - [to] vs [the]
Iter= 3000, Loss= 4.211221, TrAcc= 8.50% ['company'] - [and] vs [the]
Iter= 4000, Loss= 4.007069, TrAcc= 10.60% ['of'] - [course] vs [came]
Iter= 5000, Loss= 3.964375, TrAcc= 10.60% ['a'] - [young] vs [and]
Iter= 6000, Loss= 4.051782, TrAcc= 10.10% ['a'] - [few] vs [and]
Iter= 7000, Loss= 3.840868, TrAcc= 13.30% ['made'] - [a] vs [the]
Iter= 8000, Loss= 3.962126, TrAcc= 11.90% ['rushed'] - [down] vs [a]
Iter= 9000, Loss= 3.617404, TrAcc= 13.80% [','] - [wolf] vs [and]
Iter= 10000, Loss= 3.657220, TrAcc= 14.50% ['his'] - [sheep] vs [.]
Iter= 11000, Loss= 3.966666, TrAcc= 12.30% ['he'] - [tried] vs [,]
Iter= 12000, Loss= 3.806246, TrAcc= 12.50% ['off'] - [the] vs [to]
Iter= 13000, Loss= 3.816383, TrAcc= 12.50% ['calling'] - [out] vs [a]
Iter= 14000, Loss= 3.806111, TrAcc= 12.10% ['but'] - [this] vs [.]
Iter= 15000, Loss= 3.783292, TrAcc= 13.10% ['mountain'] - [near] vs [the]
Iter= 16000, Loss= 3.534387, TrAcc= 16.00% ['again'] - [the] vs [.]
Iter= 17000, Loss= 3.868160, TrAcc= 14.60% ['and'] - [when] vs [some]
Iter= 18000, Loss= 3.816075, TrAcc= 14.30% ['out'] - [to] vs [,]
Iter= 19000, Loss= 3.562902, TrAcc= 15.80% ['had'] - [been] vs [the]
Iter= 20000, Loss= 3.807804, TrAcc= 13.60% ['it'] - [was] vs [the]
Iter= 21000, Loss= 3.927647, TrAcc= 12.90% ['.'] - [but] vs [,]
Iter= 22000, Loss= 3.670316, TrAcc= 13.50% [':'] - [a] vs [the]
Iter= 23000, Loss= 3.391761, TrAcc= 14.00% ['him'] - [,] vs [,]
Iter= 24000, Loss= 3.925676, TrAcc= 12.20% ['fooled'] - [twice] vs [the]
Iter= 25000, Loss= 3.749783, TrAcc= 12.30% ['all'] - [day] vs [.]
Iter= 26000, Loss= 3.649916, TrAcc= 12.80% ['out'] - [from] vs [help]
Iter= 27000, Loss= 3.645772, TrAcc= 14.50% ['liar'] - [will] vs [the]
Iter= 28000, Loss= 3.707476, TrAcc= 12.60% ['a'] - [considerable] vs [wolf]
Iter= 29000, Loss= 3.680318, TrAcc= 14.80% ['nobody'] - [stirred] vs [the]
Iter= 30000, Loss= 3.565695, TrAcc= 16.40% ['could'] - [get] vs [.]
Iter= 31000, Loss= 3.455500, TrAcc= 17.30% ['boy'] - [of] vs [his]
Iter= 32000, Loss= 3.856004, TrAcc= 15.10% ['boy'] - [who] vs [,]
Iter= 33000, Loss= 3.656752, TrAcc= 15.40% ['tried'] - [the] vs [a]
Iter= 34000, Loss= 3.236506, TrAcc= 18.70% ['so'] - [the] vs [.]
Iter= 35000, Loss= 3.509756, TrAcc= 17.10% ['rushed'] - [down] vs [by]
Iter= 36000, Loss= 3.633586, TrAcc= 16.00% ['wolf'] - [,] vs [,]
```

```
Iter= 37000, Loss= 3.337052, TrAcc= 17.70% ['young'] - [shepherd] vs [the]
Iter= 38000, Loss= 3.354645, TrAcc= 19.20% ['much'] - [that] vs [a]
Iter= 39000, Loss= 3.623474, TrAcc= 15.50% ['the'] - [wolf] vs [boy]
Iter= 40000, Loss= 3.493507, TrAcc= 16.00% ['down'] - [towards] vs [a]
Iter= 41000, Loss= 3.558879, TrAcc= 14.90% [','] - [still] vs [and]
Iter= 42000, Loss= 3.549413, TrAcc= 14.60% ['the'] - [foot] vs [villagers]
Iter= 43000, Loss= 3.531369, TrAcc= 14.10% ['villagers'] - [came] vs [came]
Iter= 44000, Loss= 3.172051, TrAcc= 17.20% ['the'] - [boy] vs [boy]
Iter= 45000, Loss= 3.502731, TrAcc= 15.80% [','] - [wolf] vs [and]
Iter= 46000, Loss= 2.824796, TrAcc= 19.60% [','] - [still] vs [and]
Iter= 47000, Loss= 3.751848, TrAcc= 15.30% ['the'] - [foot] vs [boy]
Iter= 48000, Loss= 3.290758, TrAcc= 17.00% ['he'] - [tried] vs [,]
Iter= 49000, Loss= 3.708877, TrAcc= 14.60% ['and'] - [when] vs [some]
Iter= 50000, Loss= 3.294118, TrAcc= 16.40% ['village'] - [calling] vs [.]
Optimization Finished!
Elapsed time:64.5 s
Run on command line: tensorboard --logdir=lstm_words
Point your web browser to the returned link
Model saved
```

In [51]:

```python
#Your implementation goes here
with tf.Session() as session:
    # Restore variables from disk.
    model_saver.restore(session, "lstm_model/LSTMmodel_1")
    print("Model restored.\n")
    RNNcreate_story('foot', session, numOfSentences=5, n_input=1)
```

```
Model restored.

foot he wolf , and some . he wolf , and some . he wolf , and some .
```

We can see that with only one input the model enters almost immediately into a loop and it will not generate anything different. The commas and the dots are very frequent in the output; this us due to the high relative frequency of these symbols in the original text compared to any other word.

In [56]:

```python
tf.reset_default_graph()

# Training Parameters
learning_rate = 0.001
epochs = 50000
display_step = 1000
n_input = 2
n_hidden = 64

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

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

pred = RNNModel(x, n_input, weights, biases)
```

```
# Loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
optimizer = tf.train.RMSPropOptimizer(learning_rate=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))

model_saver = tf.train.Saver()
```

In [57]:

```
RNNTrain('LSTMmodel_2')
```

```
Start Training
Iter= 1000, Loss= 4.336366, TrAcc= 9.80% ['by', 'which'] - [he] vs [a]
Iter= 2000, Loss= 3.687833, TrAcc= 18.00% ['a', 'considerable'] - [time] vs [the]
Iter= 3000, Loss= 3.523067, TrAcc= 19.00% ['come', 'out'] - [from] vs [who]
Iter= 4000, Loss= 2.776961, TrAcc= 25.70% [',', 'thought'] - [the] vs [the]
Iter= 5000, Loss= 3.306874, TrAcc= 19.50% ['complained', ','] - [the] vs [and]
Iter= 6000, Loss= 3.188090, TrAcc= 20.70% ['.', 'it'] - [was] vs [the]
Iter= 7000, Loss= 2.942333, TrAcc= 22.80% ['the', 'villagers'] - [came] vs [,]
Iter= 8000, Loss= 2.827688, TrAcc= 25.50% ['.', 'but'] - [shortly] vs [this]
Iter= 9000, Loss= 2.517571, TrAcc= 29.40% ['than', 'before'] - [.] vs [,]
Iter= 10000, Loss= 2.511359, TrAcc= 31.00% [',', 'and'] - [when] vs [again]
Iter= 11000, Loss= 2.387003, TrAcc= 34.50% ['the', 'foot'] - [of] vs [calling]
Iter= 12000, Loss= 2.246355, TrAcc= 38.80% [',', 'wolf'] - [,] vs [when]
Iter= 13000, Loss= 2.363784, TrAcc= 36.90% ['the', 'villagers'] - [came] vs [came]
Iter= 14000, Loss= 2.075026, TrAcc= 44.50% ['wolf', ','] - [wolf] vs [wolf]
Iter= 15000, Loss= 1.944080, TrAcc= 47.80% ['the', 'wolf'] - [made] vs [so]
Iter= 16000, Loss= 2.231664, TrAcc= 40.30% ['there', 'was'] - [once] vs [for]
Iter= 17000, Loss= 1.826472, TrAcc= 51.30% ['thought', 'upon'] - [a] vs [a]
Iter= 18000, Loss= 1.855541, TrAcc= 50.70% ['for', 'a'] - [considerable] vs [and]
Iter= 19000, Loss= 1.920005, TrAcc= 50.30% ['began', 'to'] - [worry] vs [worry]
Iter= 20000, Loss= 1.913922, TrAcc= 48.50% ['was', 'again'] - [deceiving] vs [a]
Iter= 21000, Loss= 1.867298, TrAcc= 50.10% ['the', 'wise'] - [man] vs [man]
Iter= 22000, Loss= 1.998264, TrAcc= 47.20% ['plan', 'by'] - [which] vs [,]
Iter= 23000, Loss= 1.567458, TrAcc= 56.00% ['few', 'days'] - [afterwards] vs [a]
Iter= 24000, Loss= 1.864468, TrAcc= 50.30% ['the', 'sheep,'] - [and] vs [and]
Iter= 25000, Loss= 1.774265, TrAcc= 52.10% ['to', 'his'] - [help] vs [help]
Iter= 26000, Loss= 1.637218, TrAcc= 55.70% ['said', ':'] - [a] vs [a]
Iter= 27000, Loss= 1.811519, TrAcc= 51.50% ['rather', 'lonely'] - [for] vs [before]
Iter= 28000, Loss= 1.593970, TrAcc= 54.70% ['calling', 'out'] - [wolf] vs [a]
Iter= 29000, Loss= 1.717107, TrAcc= 54.80% ['same', 'trick'] - [,] vs [,]
Iter= 30000, Loss= 0.846970, TrAcc= 72.90% ['cried', 'out'] - [wolf] vs [to]
Iter= 31000, Loss= 1.373639, TrAcc= 58.30% ['was', 'again'] - [deceiving] vs [a]
Iter= 32000, Loss= 2.125378, TrAcc= 54.10% ['the', 'wise'] - [man] vs [man]
Iter= 33000, Loss= 2.035631, TrAcc= 51.70% ['.', 'it'] - [was] vs [was]
Iter= 34000, Loss= 1.840392, TrAcc= 52.70% ['him', ','] - [and] vs [wolf]
Iter= 35000, Loss= 1.382493, TrAcc= 61.40% ['same', 'trick'] - [,] vs [afterwards]
Iter= 36000, Loss= 1.159160, TrAcc= 69.80% [',', 'still'] - [louder] vs [louder]
Iter= 37000, Loss= 2.205759, TrAcc= 49.70% ['help', '.'] - [so] vs [so]
Iter= 38000, Loss= 1.900805, TrAcc= 54.30% ['his', 'sheep'] - [at] vs [at]
Iter= 39000, Loss= 1.686734, TrAcc= 59.00% ['and', 'some'] - [excitement] vs [excitem
ent]
Iter= 40000, Loss= 1.760026, TrAcc= 53.00% ['this', 'pleased'] - [the] vs [the]
Iter= 41000, Loss= 1.883512, TrAcc= 52.20% ['the', 'boy'] - [of] vs [so]
Iter= 42000, Loss= 1.649607, TrAcc= 53.80% ['wolf', 'made'] - [a] vs [a]
Iter= 43000, Loss= 1.645473, TrAcc= 54.90% ['speaks', 'the'] - [truth] vs [truth]
```

```
Iter= 44000, Loss= 1.393600, TrAcc= 64.50% ['he', 'thought'] - [upon] vs [tended]
Iter= 45000, Loss= 1.513444, TrAcc= 61.40% [',', 'and'] - [some] vs [the]
Iter= 46000, Loss= 1.370686, TrAcc= 65.70% ['but', 'shortly'] - [after] vs [after]
Iter= 47000, Loss= 1.514678, TrAcc= 63.90% [',', 'still'] - [louder] vs [louder]
Iter= 48000, Loss= 1.525732, TrAcc= 61.50% ['so', 'the'] - [wolf] vs [wolf]
Iter= 49000, Loss= 1.738141, TrAcc= 58.20% ['shepherd', 'boy'] - [who] vs [wolf]
Iter= 50000, Loss= 0.947556, TrAcc= 73.40% ['some', 'excitement'] - [.] vs [after]
Optimization Finished!
Elapsed time:58.8 s
Run on command line: tensorboard --logdir=lstm_words
Point your web browser to the returned link
Model saved
```

In [58]:

```python
#Your implementation goes here
with tf.Session() as session:
    # Restore variables from disk.
    model_saver.restore(session, "lstm_model/LSTMmodel_2")
    print("Model restored.\n")
    RNNcreate_story('a wolf', session, numOfSentences=5, n_input=2)
```

```
Model restored.

a wolf actually did come . considerable was for stopped out a wolf actually did come
. considerable was for stopped out a wolf actually did come . considerable was for st
opped out a wolf actually did come . considerable was for stopped out a wolf actually
 did come .
```

Considerig the last two words the situation is similar to the previous one, the difference is that reasonably the loop is a bit longer than the previous one.

In [52]:

```python
tf.reset_default_graph()

# Training Parameters
learning_rate = 0.001
epochs = 50000
display_step = 1000
n_input = 5
n_hidden = 64

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

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

pred = RNNModel(x, n_input, weights, biases)

# Loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
optimizer = tf.train.RMSPropOptimizer(learning_rate=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))
```

```
model_saver = tf.train.Saver()
```

In [38]:
```
RNNTrain('LSTMmodel_5')
```

```
Start Training
Iter= 1000, Loss= 4.470892, TrAcc= 8.50% ['come', 'to', 'his', 'help', '.'] - [so] vs
 [,]
Iter= 2000, Loss= 3.527248, TrAcc= 20.40% ['from', 'the', 'forest', ',', 'and'] - [be
gan] vs [of]
Iter= 3000, Loss= 2.446506, TrAcc= 44.30% ['again', 'the', 'villagers', 'came', 'to']
 - [his] vs [deceiving]
Iter= 4000, Loss= 1.936984, TrAcc= 53.30% ['towards', 'the', 'village', 'calling', 'o
ut'] - [wolf] vs [a]
Iter= 5000, Loss= 1.706365, TrAcc= 56.40% ['village', 'said', ':', 'a', 'liar'] - [wi
ll] vs [much]
Iter= 6000, Loss= 0.857457, TrAcc= 77.30% [',', 'still', 'louder', 'than', 'before']
- [.] vs [.]
Iter= 7000, Loss= 0.514233, TrAcc= 87.40% ['the', 'villagers', 'came', 'to', 'his'] -
 [help] vs [help]
Iter= 8000, Loss= 0.455893, TrAcc= 87.90% ['villagers', 'came', 'out', 'to', 'meet']
- [him] vs [him]
Iter= 9000, Loss= 0.274175, TrAcc= 93.10% ['a', 'little', 'company', 'and', 'some'] -
 [excitement] vs [excitement]
Iter= 10000, Loss= 0.193359, TrAcc= 94.90% ['rather', 'lonely', 'for', 'him', 'all']
- [day] vs [day]
Iter= 11000, Loss= 0.199799, TrAcc= 95.40% ['a', 'liar', 'will', 'not', 'be'] - [beli
eved] vs [believed]
Iter= 12000, Loss= 0.134664, TrAcc= 96.50% ['deceiving', 'them', ',', 'and', 'nobody'
] - [stirred] vs [stirred]
Iter= 13000, Loss= 0.124053, TrAcc= 97.00% ['of', 'course', 'cried', 'out', 'wolf'] -
 [,] vs [,]
Iter= 14000, Loss= 0.083010, TrAcc= 98.00% ['the', 'forest', ',', 'and', 'began'] - [
to] vs [to]
Iter= 15000, Loss= 0.099725, TrAcc= 97.30% ['village', 'calling', 'out', 'wolf', ',']
 - [wolf] vs [wolf]
Iter= 16000, Loss= 0.048435, TrAcc= 99.10% ['of', 'a', 'mountain', 'near', 'a'] - [da
rk] vs [dark]
Iter= 17000, Loss= 0.075171, TrAcc= 97.90% ['wolf', 'made', 'a', 'good', 'meal'] - [o
ff] vs [off]
Iter= 18000, Loss= 0.076344, TrAcc= 98.20% [',', 'wolf', ',', 'still', 'louder'] - [t
han] vs [than]
Iter= 19000, Loss= 0.081158, TrAcc= 97.80% ['boy', 'so', 'much', 'that', 'a'] - [few]
 vs [few]
Iter= 20000, Loss= 0.069921, TrAcc= 98.40% ['came', 'out', 'to', 'meet', 'him'] - [,]
 vs [,]
Iter= 21000, Loss= 0.057007, TrAcc= 98.70% ['boy', 'who', 'tended', 'his', 'sheep'] -
 [at] vs [at]
Iter= 22000, Loss= 0.065841, TrAcc= 98.30% ['stirred', 'to', 'come', 'to', 'his'] - [
help] vs [help]
Iter= 23000, Loss= 0.042928, TrAcc= 98.90% ['boy', 'of', 'course', 'cried', 'out'] -
[wolf] vs [wolf]
Iter= 24000, Loss= 0.067561, TrAcc= 98.00% ['considerable', 'time', '.', 'this', 'ple
ased'] - [the] vs [the]
Iter= 25000, Loss= 0.048364, TrAcc= 99.20% ['of', 'them', 'stopped', 'with', 'him'] -
 [for] vs [for]
Iter= 26000, Loss= 0.066601, TrAcc= 98.30% ['sheep', 'at', 'the', 'foot', 'of'] - [a]
 vs [a]
```

```
Iter= 27000, Loss= 0.060698, TrAcc= 98.60% ['come', 'to', 'his', 'help', '.'] - [so]
vs [so]
Iter= 28000, Loss= 0.054933, TrAcc= 98.80% ['than', 'before', '.', 'but', 'this'] - [
time] vs [time]
Iter= 29000, Loss= 0.037918, TrAcc= 99.20% ['his', 'help', '.', 'but', 'shortly'] - [
after] vs [after]
Iter= 30000, Loss= 0.055451, TrAcc= 98.80% ['them', 'stopped', 'with', 'him', 'for']
- [a] vs [a]
Iter= 31000, Loss= 0.058863, TrAcc= 98.60% ['plan', 'by', 'which', 'he', 'could'] - [
get] vs [get]
Iter= 32000, Loss= 0.046259, TrAcc= 98.80% ['a', 'liar', 'will', 'not', 'be'] - [beli
eved] vs [believed]
Iter= 33000, Loss= 0.058114, TrAcc= 98.70% ['before', ',', 'thought', 'the', 'boy'] -
 [was] vs [was]
Iter= 34000, Loss= 0.034799, TrAcc= 99.20% ['but', 'this', 'time', 'the', 'villagers'
] - [,] vs [,]
Iter= 35000, Loss= 0.051708, TrAcc= 98.90% ['out', 'from', 'the', 'forest', ','] - [a
nd] vs [and]
Iter= 36000, Loss= 0.031435, TrAcc= 99.30% ['him', 'for', 'a', 'considerable', 'time'
] - [.] vs [.]
Iter= 37000, Loss= 0.060743, TrAcc= 98.70% ['.', 'it', 'was', 'rather', 'lonely'] - [
for] vs [for]
Iter= 38000, Loss= 0.045377, TrAcc= 98.70% ['so', 'the', 'wolf', 'made', 'a'] - [good
] vs [good]
Iter= 39000, Loss= 0.038292, TrAcc= 99.10% ['after', 'this', 'a', 'wolf', 'actually']
 - [did] vs [did]
Iter= 40000, Loss= 0.036152, TrAcc= 99.20% ['which', 'he', 'could', 'get', 'a'] - [li
ttle] vs [little]
Iter= 41000, Loss= 0.057424, TrAcc= 98.10% ['when', 'he', 'speaks', 'the', 'truth'] -
 [.] vs [.]
Iter= 42000, Loss= 0.043506, TrAcc= 98.50% ['been', 'fooled', 'twice', 'before', ',']
 - [thought] vs [thought]
Iter= 43000, Loss= 0.044132, TrAcc= 98.30% ['of', 'course', 'cried', 'out', 'wolf'] -
 [,] vs [,]
Iter= 44000, Loss= 0.052136, TrAcc= 97.90% ['tried', 'the', 'same', 'trick', ','] - [
and] vs [and]
Iter= 45000, Loss= 0.063674, TrAcc= 98.20% ['came', 'out', 'to', 'meet', 'him'] - [,]
 vs [,]
Iter= 46000, Loss= 0.041295, TrAcc= 98.90% ['rather', 'lonely', 'for', 'him', 'all']
- [day] vs [day]
Iter= 47000, Loss= 0.042034, TrAcc= 98.70% ['complained', ',', 'the', 'wise', 'man']
- [of] vs [of]
Iter= 48000, Loss= 0.056823, TrAcc= 98.50% ['still', 'louder', 'than', 'before', '.']
 - [but] vs [but]
Iter= 49000, Loss= 0.013639, TrAcc= 99.50% ['after', 'this', 'a', 'wolf', 'actually']
 - [did] vs [did]
Iter= 50000, Loss= 0.045306, TrAcc= 98.50% ['meet', 'him', ',', 'and', 'some'] - [of]
 vs [of]
Optimization Finished!
Elapsed time:81.2 s
Run on command line: tensorboard --logdir=lstm_words
Point your web browser to the returned link
Model saved
```

In [55]:

```
#Your implementation goes here
with tf.Session() as session:
    # Restore variables from disk.
    model_saver.restore(session, "lstm_model/LSTMmodel_5")
    print("Model restored.\n")
    RNNcreate_story('a boy a wolf and', session, numOfSentences=5, n_input=5)
```

```
Model restored.

a boy a wolf and villagers came out to so him time with with him for a considerable t
ime . this pleased the boy so much that a few days afterwards he tried the same trick
 , and again the boy complained , the wise man of the village said : a liar will not
be believed , even when he speaks the truth .
```

In this case the number of words that we are considering is too big compared to the length of the text we have: indeed, the model tends to recreate the same exact sentences that where present in the original text, even though we started from a sentence ('a boy a wolf and') that was not there. We could consider this as overfitting our model.