Information Systems and

Machine Learning Lab.

Stiftung Universität Hildesheim

Marienburger Platz 22

31141 Hildesheim

Hadi Samer Jomaa

Supervisor

Sequence to Sequence Learning with Neural Networks

Krithika Murugesan
277537, murugesa@uni-hildesheim.de

Abstract

The real time sequence data that we need a network to learn may not be labeled always while, Deep Neural Networks (DNNs) are very powerful models but their performance can be undermined by the lack of labeled training data. The authors propose an end-to-end sequence learning method that uses two Long Short-Term Memory units (LSTMs) with minimal assumptions. One converts the input sequence to a vector of fixed length and the other converts this vector to output sequence. The model yields good results even with long sentences, it can also learn sensible phrase and sentence representations which are sensitive to word order, invariant of the voice. They also discovered that reversing the order of the initial sentence introduces numerous short term dependencies among the source and target sentences as the corresponding words are now at a rather closer distance making the optimization problem easier. It is one of the most influential papers in sequence to sequence learning and was presented at Neural Information Processing Systems (NIPS) Conference in 2014.

Contents

1. Introduction

Deep Neural Networks (DNNs) are very competent models, they can be used to learn basic machine learning tasks such as predictions, classifications and clustering to complex sound engineering or even object recognition. They are efficient as computations can be done in parallel, making the whole process faster. The network parameters are generally learned by back-propagation through the network, to achieve this ground truth is essential. So that the network can compare the predictions to the ground truth and change the network parameters making the next predictions closer to the original target value. Most of the sequence to sequence learning problems do not have such labeled data, which makes use of DNNs highly unlikely for such applications. Also DNNs can be applied to only vectors that have a fixed dimensionality, there are numerous significant sequence to sequence learning problems in which we do not know the input data size in advance, like translation tasks or chat-bots.

The authors came up with a comprehensive method to learn real-time sequences. It is a rather simple approach to use one Long Short-Term Memory (LSTM) network to learn the input sequence one by one at each time step and convert it to a vector of fixed dimension and another LSTM to convert this vector to an output sequence. LSTM networks have the ability to make decisions by taking the past states into consideration, such long range temporal dependency learning ability of these networks has made it the go to choice for this model. In order to get more confident predictions the source sentences were reversed luckily, they introduced several short term dependencies in the network making the optimization problem easier. This also made the model perform well on long sentences unlike the peer researcher's results then.

This sequence to sequence learning method was evaluated using a translation prediction, that used the WMT'14 English to French translation task. This resulted in a Bilingual Evaluation Understudy (BLEU) score of 34.81 which was the highest score achieved by a large neural network translation then. When this model was used to rescore publicly available 1000-best lists of Statistical Machine Translation (SMT) baseline, it recorded a BLEU score of 36.5 which is 3.2 BLEU points higher than the baseline and almost very close to the then state of the art model. At the same time there were several other peer groups trying to solve this problem, the next section will briefly introduce these methods.

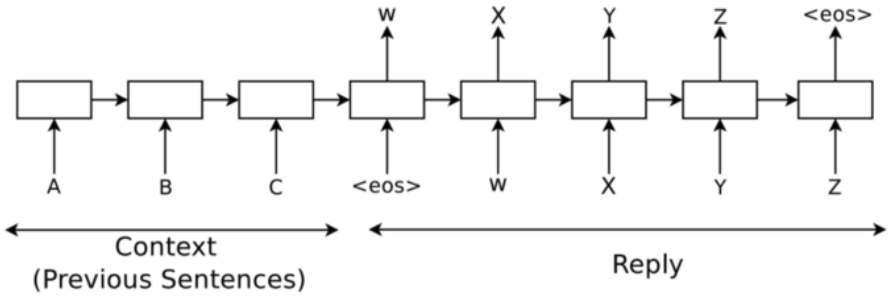

Figure 1 : The model reads the input sequence "ABC" and outputs "WXYZ". It identifies the termination of the sentence by the end of sentence token <eos>

2. Related Work

There are several ways to overcome the problems of using DNNs for sequence learning. Several researchers have tried to use Recurrent Neural Networks (RNN) to learn from input vectors with no fixed dimensionality. K.Cho [1] proposed a RNN encoder-decoder network, this model had two different RNNs similar to the model in discussion they used one RNN to convert the input sequence to a vector of fixed dimension and another one to obtain the output. They are jointly trained with the objective to maximize the conditional probability of a target sequence. They used phrase pairs in the same English to French translation task, the results were impressive as the network could learn the semantics and syntaxes of the language. Their architecture included a hidden unit that had reset and update gates which prompt the RNN as to how much of the past it has to remember.

Connectionist Temporal Classification [2] is yet another method that uses RNNs for sequence labeling. This was considered to be an alternative to the Hidden Markov Models (HMMs) and Conditional Random Fields where prior knowledge about the sequence and the task was required for learning. The main idea of the model is to get a probability distribution over all possible label sequences, given the input sequence. It still assumes a monotonic alignment between the inputs and outputs.

Recurrent Continuous Translation Models (RCTM) [3] was one the primitive approaches that wanted to represent the sequences as continuous data and not phrase based systems. Use of phrases has its own disadvantages of sparse representation and skewness. This continuous translation model maps a sentence from the source language to probability distribution over sentences in the target language without any Markov assumptions about the dependencies. It is sensitive to the syntaxes and semantics and also more flexible with relatively low perplexity.

Graves [4] also proposed a similar approach as the authors. They used LSTMs for language modeling. RNNs like most of the other neural networks use their internal representation to perform interpolation of high dimension between training samples. Also, RNNs posses a certain Amnesia of not being able to retain information long enough making it a little difficult to overcome wrong predictions when the recent predictions themselves were made by the network. LSTMs do not suffer from the vanishing gradient problem unlike RNN networks.

Bahdanau [5] successfully applied an attention mechanism in translation tasks where the model automatically searches for parts of the source sentence that are most relevant to the prediction. Most of the translation tasks employ an encoder and a decoder, compressing the complete input sequence to a vector of fixed dimension, it becomes more difficult when the sentences are longer leading to a deteriorated performance. This issue can be overcome by introducing a attention mechanism, instead of converting the whole input sequence into a vector of fixed dimension it encodes them into a sequence of vectors and chooses a subset of these vectors during the decoding process.

Though there were several different approaches to the sequence to sequence learning problem. The authors propose a very simple and easy alternative with high performance, reversing the source sentence worked well in their favor as it reduced the optimization problem with a minor change in the way the input is fed into the network. Following sections explain this model in depth.

3. Methodology

The comprehension power of humans is attributed the persistence of different thoughts we encounter. Similarly, in neural networks RNNs have this ability to involve past information in the process of decision making. They loop within themselves to make the information persist.

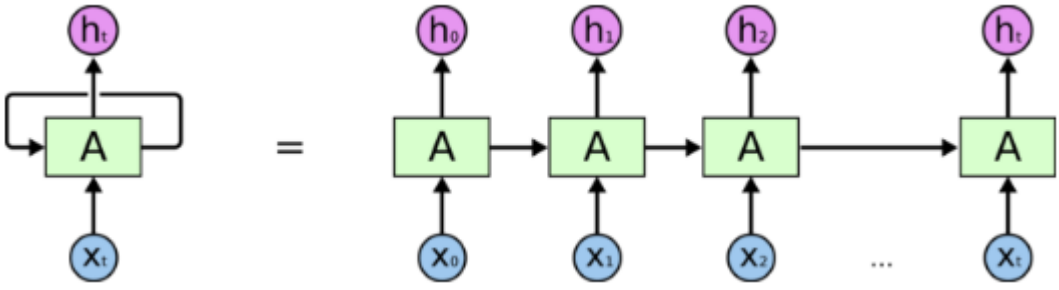

Figure 2 : An unrolled Recurrent Neural Network

RNNs take into consideration the states at the previous step also as the input to the hidden layer at the current step, this mechanism enables them to store information. Although they are widely used in speech recognition and machine translation tasks they suffer with the long term dependencies. Considering a language model trying to predict the next word in a sentence, when the sentence is fairly small it is easy for the RNN to predict the next words. Whereas when the prediction requires context from text several sentences before the current one, RNNs are not able to connect the information. LSTMs are special kinds of RNNs which are capable of learning these long term dependencies. They also have a repeating structure like RNNs but each repeating module is a combination of several gates which decides which amount of what past information has to be retained at the current step to make decisions.

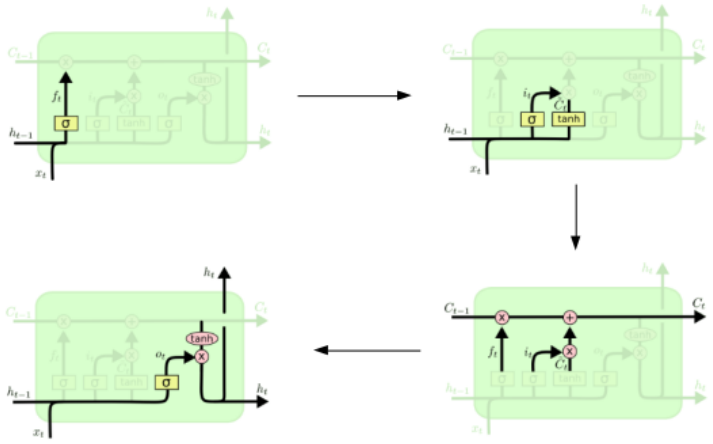

Figure 3 : The different gates in a LSTM

The horizontal line at the top of the cell is called as the cell state, it acts like a transmission line that runs across the cell carrying information. The information in cell state can be manipulated with the gates, generally four gates are used. The first layer is the forget gate this sigmoid activated layer decides how much of the past information we should retain. Termination of a paragraph implies that there will be change in the pronouns used, so the current pronouns need not be remembered by the cell state. The next sigmoid layer decides which of the new information has to be added to the state and the tanh layer generates the corresponding vector which is added to the cell state. The final layer will decide what will be the output of the cell, by retaining certain parts of the information and adding new values the complete new state is achieved by this block of LSTM. Also, the problem of vanishing gradients in RNNs that is, on multiplying recursively with values lesser then one the gradients after considerable time steps diminishes not being useful anymore is overcome by LSTMs.

As stated in the related work, several approaches use two RNNs, one as decoder and other as encoder for translation tasks, and their performance

is moderate, training of such networks can become strenuous. In general LSTM can be used for language modeling by getting a fixed dimensional representation v of the input sequence given by the last layer of the LSTM, from which the probability in target sequence is got by another LSTM with the first hidden layer having v as input. Each of the probabilities are softmax over all words in the vocabulary, a <EOS> token is fed at the end of each sentence to notify the LSTM that it is the end of the current sentence. This probability is formulated as the following

$$p(y_1, \ldots, y_{T'} | x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \ldots, y_{t-1})$$

Equation 1 : Here the input sequence is x_1, x_2,..., x_T and the corresponding output sequence is y_1, y_2,..., y_{t-1}. The input and output sequences are of different lengths

The major design factors in the proposed model are it has two LSTMs, one to generate a fixed dimension from the input sequence and another to decode this vector to output sequence. Learning two networks increases the number of parameters to be learnt which can be done with minimal additional computation cost. Here an LSTM with four layers was used as deep LSTMs performed better than their shallow counterparts. The source sentence is reversed, which makes the corresponding equivalent words closer to each other making the Stochastic Gradient Descent (SGD) work more efficiently. This also enabled the model to work well for long sentences.

3.1. Decoding and Rescoring

The objective function to be maximized is the log probability of the correct translation T for the source sentence S.

$$1/|\mathcal{S}| \sum_{(T,S)\in\mathcal{S}} \log p(T|S)$$

Equation 2 : The objective function (log probability) to be maximized for a translated sentence T given source sentence S

The LSTMs are trained on the various sentence pairs in the translation task, the predictions are made by choosing the translated sentence with the maximum log probability given the source sentence. To identify this, a simple left to right beam search is used. This method starts with the first word in the translated sentence, to get the next word, it chooses the top n responses with highest probability where n is the beam size. At each next word n top contenders are chosen, finally a joint probability for each possible sequence is got and the sequence with the highest value is the predicted output. When the model encounters <EOS> it know it is the end of current

hypothesis. This method did well even with the beam size of one meaning that it is capable to predict the best fit easily and gave best performance with a beam size of two.

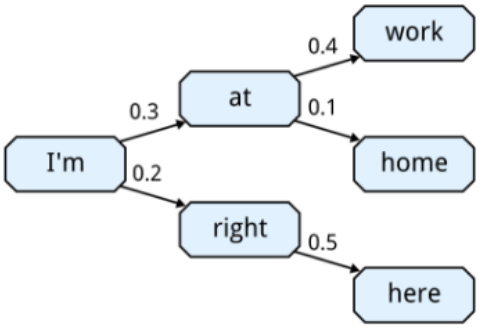

Figure 4 : Example of left to right beam search with beam size of 2

3.2. Reversing Source Sentences

To increase the confidence in prediction values, the authors attempted to reverse the source sentence. Doing so introduced a lot of short-term dependencies in the network and also decreased the proximity of corresponding words equivalent in the source and translated sentence, thereby reducing the minimal time lag. Since, the first few words in the source sentence are now closer to the first few words in the translated sentence back-propagation was easier as the model could establish connection between the two sentences more easily. This also made the model perform better than peer models at that time as the memory was used effectively by this small change in input sequence with no additional computation costs involved.

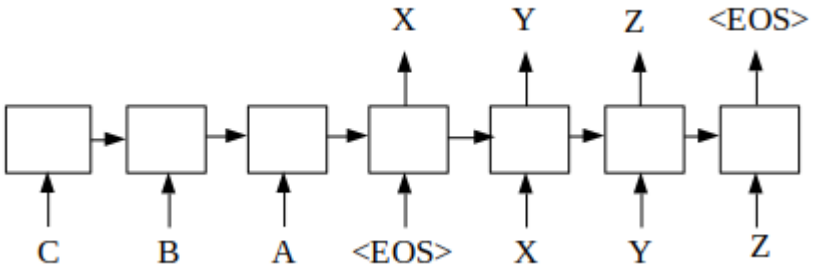

Figure 5 : Reversing original sentence ABC as CBA, we can see that corresponding outputs X,Y and Z is closer to A, B and C

4. Experiments

The sequence to sequence model proposed was evaluated on WMT’14 English to French translation task. The source sentences were translated directly and also a Statistical Machine Translation (SMT) system was used as a reference to rescore the n-best lists of its baseline. The accuracy of these predictions were measured by Bilingual Evaluation Understudy (BLEU) score.

4.1. BLEU score

In theater a callow artist who tries to imitate a professional is called Understudy. Thus, the name BLEU as we try to build a scoring mechanism that imitates an actual person checking if the translations are correct. The approach works by counting matching n-grams in the source and translated sentences, a perfect match gets a score 1 else 0. The word order is not taken into consideration though. This score lets us compare how closely the two sentences are related to each other. Usually a perfect score is not possible even with human intervention. A cumulative score is got by calculating the weighted geometric mean of individual n-gram scores at all orders from 1 to n.

4.2. Dataset and Training

The WMT’14 English to French dataset was used, 12 million sentences were used for training, summing up to 348M French words and 304 M English words. The dataset also has the top 1000-best lists from baseline SMT which is used for re-scoring. Due to the expansive vocabulary a number of unique words were limited to 160,000 and 80,000 most frequent words in the source and translated languages respectively. Any word occurrence other than these was denoted as “UNK”

Deep LSTMs of four layers were used with 1000 cells at each layer. The output layer used softmax as activation function. All LSTM parameters were initialized with uniform distribution between -0.08 and 0.08. SGD was used to learn the parameters for 7.5 iter. There may be exploding gradient problem during back propagation, when we keep multiplying numbers greater than one with values greater than one they tend to have huge values which does not help optimization. To avoid this at each iteration the gradient was checked against a threshold and when it exceeds this value the gradient was clipped to lower values. Also, since there may be sentences of varying sizes a mini-batch with huge variations in sentence size is waste of computation power, to avoid this mini-batches were chosen in a way that the sentences were of similar lengths which increased the efficiency of the whole training process.

A single machine was too slow considering the massive size of the input, so the computation was parallelized on a 8-GPU machine, where each layer of the LSTM was executed on a different GPU. Softmax calculations were also parallelized. Training was done in about ten days even with such efficient implementation.

4.3. Results

In the direct translations the model could not perform better than the state of the art, but it out performed the SMT baseline system. The best configuration was achieved with an ensemble of LSTMs with different random initializations and random order of mini-batches. The scores are summarized below,

Method	BLEU Score
Baseline System [7]	33.3
State of the Art [6]	37
Ensemble of 5 reversed LSTMs, beam size 12	34.81
Rescoring the baseline 1000-best with an ensemble of 5 reversed LSTMs	36.5

Table 1 : BLEU score comparison of best performing model with state of the art and baseline systems

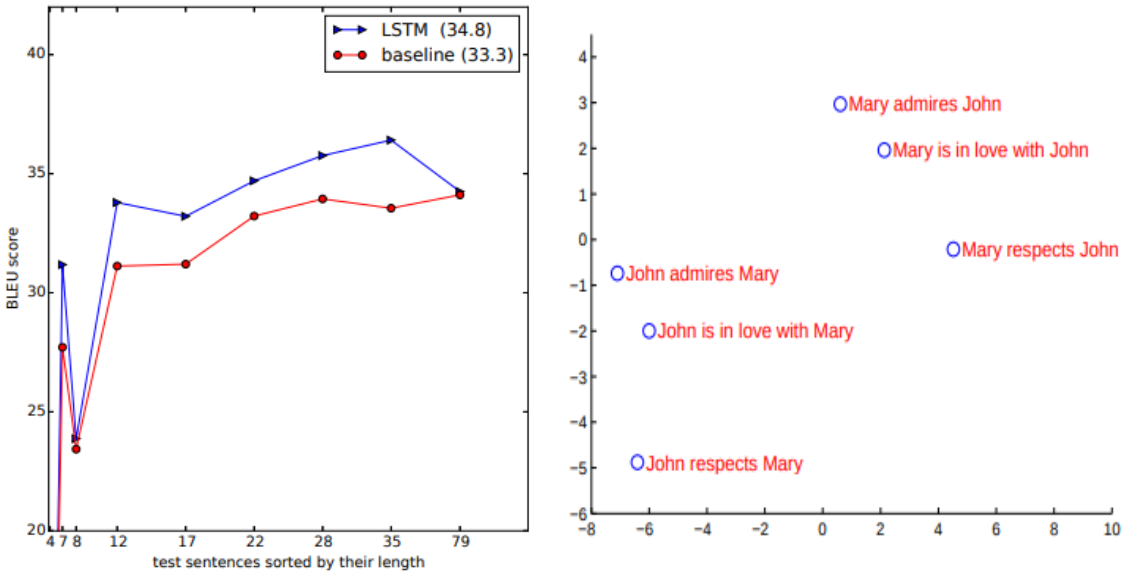

Figure 6 : In the graph on the left it can be seen that the proposed model performed better than the baseline for long sentences. The x- axis represents length of the sentence and the y-axis it's corresponding BLEU score. The second plot shows the 2-dimensional PCA projection of LSTMs hidden states, the phrases are clustered by meaning based on word order which is difficult to capture is fairly simple models like bag of words

The model performed better than the baseline even for long sentences, a graphical summarization is shown in Figure 6. The ability of the model to cluster sentences based on meaning stemming from word order is shown in the second graph.

5. Discussion and Conclusion

The use of two different LSTMs for sequence to sequence learning was a huge success and a starting point for various other approaches in the domain. It is a pretty popular paper with increasing citations every day. One major drawback with the model is that when the sentences are of different sizes or very long the network may suffer a little, An attention mechanism discussed in [5] can be used along with the model to make it more efficient. The idea is so popular that there is a customized package (Seq2Seq) in python, where we can implement any configuration of LSTMs. This method is not just popular for text translations, it can also be used with sounds and other sequences. Many chat-bots in use currently use this as a base idea. It is a very simple approach but a very powerful one which makes it more popular.

Reversing the sentences increased the performance by many fold which was quite unexpected by the authors themselves due to the numerous short-term dependencies introduced. Also, this was one of the first neural networks that outperformed an SMT based translation system. As part of understanding the concepts better I tried to implement the model for English to Tamil translation and the results were worthy considering the minimal coding involved. The source and translated sentence are shown below,

```
-
Input sentence: Talk to me!
Decoded sentence: என்னிடம் பேசு

-
Input sentence: She is kind.
Decoded sentence: அவள் அன்பானவள்
```

This is a simple model, but does wonders in sequence to sequence learning and is very influential even today!

References:

[1] K. Cho, B. Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Arxiv preprint arXiv:1406.1078, 2014.

[2] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In ICML, 2006.

[3] N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In EMNLP, 2013.

[4] A. Graves. Generating sequences with recurrent neural networks. In Arxiv preprint arXiv:1308.0850, 2013.

[5] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.

[6] Nadir Durrani, Barry Haddow, Philipp Koehn, and Kenneth Heafield. Edinburgh's phrase-based machine translation systems for wmt-14. In WMT, 2014.

[7] H. Schwenk. University le mans. http://www-lium.univ-lemans.fr/˜schwenk/cslm_joint_paper/, 2014. [Online; accessed 03-September-2014]

[8] http://www.googblogs.com/tag/machine-translation/

[9] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[10] https://machinelearningmastery.com/calculate-bleu-score-for-text-python/

[11] https://arxiv.org/pdf/1702.01806.pdf

[12] http://www.phontron.com/slides/neubig14taiwa11.pdf

Appendix (Python code):

```
from __future__ import print_function
from keras.models import Model
from keras.layers import Input, LSTM, Dense
import numpy as np

batch =15  # Batch size for training.
iter = 400  # Number of iter to train for.
latentDim = 256  # Latent dimensionality of the encoding space.
samples = 10000  # Number of samples to train on.
# Path to the data txt file on disk.
path = 'tam.txt'
# Vectorize the data.
input = []
output = []
inputChr = set()
outputChr = set()
with open(path, 'r', encoding='utf-8') as f:
    lines = f.read().split('\n')
for line in lines[: min(samples, len(lines) - 1)]:
    input_text, target_text = line.split('\t')
    # start sequence - tab,<EOS> - '\t'
    target_text = '\t' + target_text + '\n'
    input.append(input_text)
    output.append(target_text)
    for char in input_text:
        if char not in inputChr:
            inputChr.add(char)
    for char in target_text:
        if char not in outputChr:
            outputChr.add(char)

inputChr = sorted(list(inputChr))
outputChr = sorted(list(outputChr))
encoderTokens = len(inputChr)
decoderTokens = len(outputChr)
encoderMax = max([len(txt) for txt in input])
decoderMax = max([len(txt) for txt in output])
inputIndex = dict([(char, i) for i, char in enumerate(inputChr)])
```

```
targetIndex = dict([(char, i) for i, char in enumerate(outputChr)])
encInputIndex = np.zeros((len(input),encoderMax, encoderTokens),dtype='float32')
decInputIndex = np.zeros((len(input),decoderMax, decoderTokens),dtype='float32')
decOutput = np.zeros((len(input), decoderMax, decoderTokens),dtype='float32')

for i, (input_text, target_text) in enumerate(zip(input, output)):
    for t, char in enumerate(input_text):
        encInputIndex[i, t, inputIndex[char]] = 1.
    for t, char in enumerate(target_text):
        # decoder is one tome step ahead
        decInputIndex[i, t, targetIndex[char]] = 1.
        if t > 0:
            # decoder does not include start sequence character
            decOutput[i, t - 1, targetIndex[char]] = 1.
#Input sequence
encInput = Input(shape=(None, encoderTokens))
encoder = LSTM(latentDim, return_state=True)
encOutput, stateH, stateC = encoder(encInput)
# Retain only the states which is given as input to encoder
encState = [stateH, stateC]
decInput = Input(shape=(None, decoderTokens))
#Decoder inference
decoder = LSTM(latentDim, return_sequences=True, return_state=True)
decOutputs, _, _ = decoder(decInput,initial_state=encState)
decDense = Dense(decoderTokens, activation='softmax')
decOutputs = decDense(decOutputs)
# The Model
model = Model([encInput, decInput], decOutputs)
# Train
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
model.fit([encInputIndex, decInputIndex], decOutput,
          batch=batch,
          iter=iter,
          validation_split=0.2)
# Inference (Sampling)
encLSTM = Model(encInput, encState)
decStateH= Input(shape=(latentDim,))
decStateC = Input(shape=(latentDim,))
decStateInput = [decoder_state_input_h, decStateC]
decOutputs, stateH, stateC = decoder(decInput, initial_state=decStateInput)
```

```
        decoder_states = [stateH, stateC],decOutputs = decDense(decOutputs)
        decoder_model = Model([decInput] + decStateInput, [decOutputs] + decoder_states)
# Decode
reversedInputIndx = dict( (i, char) for char, i in inputIndex.items())
reversedOutputIndx = dict((i, char) for char, i in targetIndex.items())
def decodeSeq(input_seq):
   stateValues = encLSTM.predict(input_seq)
   outputSeq = np.zeros((1, 1, decoderTokens))
   # Get start character
   outputSeq[0, 0, targetIndex['\t']] = 1.
  # Sampling
   covergence = False
   decoded = ''
   while not covergence:
      outputToken, h, c = decoder_model.predict([outputSeq] + stateValues)
      tokenSampled= np.argmax(outputToken[0, -1, :])
      charSampled = reversedOutputIndx[sampled_token_index]
      decoded += charSampled
       # Exit condition: either hit max length
       # or find stop character.
      if (charSampled == '\n' or
        len(decoded) > decoderMax):
         covergence = True
      # Update the target sequence (of length 1).
      outputSeq = np.zeros((1, 1, decoderTokens))
      outputSeq[0, 0, sampled_token_index] = 1.
      # Update states
      stateValues = [h, c]
 return decoded
#Predict
seq_index = [4,10,15,21]
for each in seq_index:
   # Take one sequence (part of the training set)
   input_seq = encInputIndex[each: each + 1]
   decoded = decodeSeq(input_seq)
```