
Computers Science and Software Engineering Department

**COMP6731 & COMP473**
**Pattern Recognition**

**Team Project Report On**

# Implementation of An Intelligent Question Answering Conversational Agent (Chatbot) using RNN Algorithm

**Examiner:** Ali Firooz **Professor:** Prof. J. Sadri

## Team members

**Samir Anghan**
Student ID - 40040308
samir.anghan@gmail.com

**Hirangi Naik**
Student ID - 40041365
hirangi.naik@gmail.com

**Mihir Pujara**
Student ID - 40025592
mihirppujara@gmail.com

**Gaurav Parvadiya**
Student ID - 40040310
gaurav.parvadiya@gmail.com

**Team leader:** Samir Anghan
**Submission date:** December 16, 2017

# Table of Contents

# Table of figures

## Abstract

Recurrent Neural Networks (RNNs) are powerful models that have achieved excellent performance on difficult learning tasks. RNNs work well whenever large labeled training sets are available, they can be used with sequence to sequence model. In this project, we present an approach to sequence learning that makes prediction on the target sequence structure. Our method uses a multi-layered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. Our experimented result is based on conversation task from the Cornell movie corpus dataset which generates the response sentences produced by the trained model. The model also learned sentence presentation that are strict to word order and are unchanged to the active and the passive voice type sentences. At the time of our research, we found that reversing the order of the words in all source sentences increased the LSTM's performance remarkably. By doing so, long term dependency problem has been fixed and also made the optimization problem easier.

## Introduction

The process of building Chatbots can be divided into several steps. First, a software function called model (technically called "classifier") is developed and trained from predefined conversational dialogues. The trained model -Classifier is capable to classify the input text or input question. This classifier identifies the "intent" (a conversational intent) from the input text and use pattern matching to predict a suitable response for the given input text or question.

### Motivation

Chatbot is a part of an AI (Artificial Intelligence). Over a timespan is it gaining more popularity and becoming the core function in every kind of businesses. It could be a cost-effective way for enterprises to connect with their potential customers on time. It would bring an automation in online business processes. It is not a boring traditional feedback process but it is a two-way interaction process between customers and businesses which can help business to have a valuable metrics from users. Chatbots are feasible option for companies to replace the human and get over from the staffing problems. Bots replies very quickly and point a direct solution for asked query.

### Objective

The main goal of this project is to implement a Chatbot for a movie dialogues corpus by implementing and studying the one or more Data Mining and Deep Learning algorithms such as Naive Bayes Classification, Support Vector Machine and Recurrent Neural Network. This paper contains study for Naive Bayes Classification, Support Vector Machine and RNN algorithm. It also describes implementation of Chatbot using Sequence to Sequence model applying Recurrent Neural Network. Hence, this study also provides enhanced RNN algorithm in terms of performance by reducing the preprocessing time.

## Development Environment

We implemented recurrent neural network (RNN) in Python 3.5, Tensorflow 1.0.0 and all experiments are under environment of OS X10.13@64bit, Intel Core i7 CPU@2.7GHz, 16GB RAM@2133MHz, 512GB SSD hard disk.

## Literature Review

### Sequence to Sequence Learning with Neural Networks [1]

**Objective**

In a very large conversational type dataset, finding sequence of words is very tedious task. Humans don't start their thinking from scratch every second. As we read anything, we understand each word based on our understanding of previous words. We don't throw everything away and start thinking from scratch again. Our thoughts have persistence. Traditional neural networks can't do this, and it seems like a major shortcoming. Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.

**Methodologies used**

- Our method uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector.

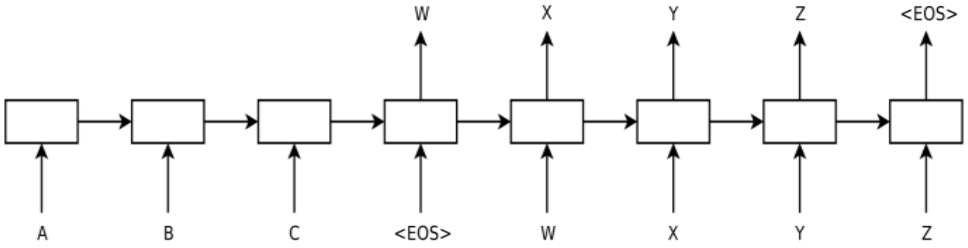


**Figure 1 Model reads an input sentence "ABC" and produces "WXYZ" as the output sentence**

- Given a sequence of inputs (x1, . . . ,xT ), a standard RNN computes a sequence of outputs (y1, . . . , yT ) by iterating the following equation:

  ht = sigm (Whxxt + Whhht−1)
  yt = Wyhht
- Since the RNN is provided with all the relevant information, it would be difficult to train the RNNs due to the resulting long term dependencies. However, the Long Short-Term

Memory (LSTM) is known to learn problems with long range temporal dependencies, so an LSTM may succeed in this setting.

- The goal of the LSTM is to estimate the conditional probability p(y1, . . . , yT′ |x1, . . . , xT ) where (x1, . . . , xT ) is an input sequence and y1, . . . , yT′ is its corresponding output sequence whose length T ′ may differ from T.

The author's implemented actual models differ from the above description in three important ways.

- They used two different LSTMs: one for the input sequence and another for the output sequence, because doing so increases the number model parameters at negligible computational cost and makes it natural to train the LSTM on multiple language pairs simultaneously.
- They found that deep LSTMs significantly outperformed shallow LSTMs, so they chose an LSTM with four layers.
- They found it extremely valuable to reverse the order of the words of the input sentence. They found this simple data transformation to greatly boost the performance of the LSTM.

**Results**

This experiment proved that Seq2seq learning with LSTM is best fit for conversational or language translation task. By reversing input sequence, LSTM greatly boost its performance.

## Some effective techniques for Naïve Bayes Text Classification [2]

**Objective**

In this paper we study that Naive Bayes is very useful for data mining but it is lacking in automatic text classification. It does not work effectively on natural dialogue processing and found a major problem in a filtering parameter. So, this paper propose two feasible solutions: Per-document text normalization and Feature weighting method. And these both alternative classifiers work effectively in standard benchmark collections and competing state of the art classifier that rely on a highly complex learning SVM method.

**Methodologies used**

There are mainly three methodologies for improving naive bayes classifiers:

- Multivariate Poisson Model for Text Classification: Here, the actual frequency of ti in multinomial naive bayes text classification is taken as fij, and the ratio of the frequency of ti in the positive (or negative) training corpus is taken as the Poisson parameter ic. As the result, the proposed text classification model becomes the traditional multinomial model

that means the multivariate Poisson text classification model is a more flexible model than the traditional model.

- Parameter Estimation Using Normalized Term Frequencies: In this method, they have first normalize the term frequencies in each document according to the document length. Then the normalized frequencies from each document are linearly combined according to the probability that each document belongs to the document set where we are going to estimate the parameter.
- Considerations of Feature Weights: Due to the drawback of using binary features, the feature weighting approach is proposed rather than feature selection.

**Results**

Experimental results show that the proposed model is quite useful to build probabilistic text classifiers with little extra cost in terms of time and space, compared to the traditional multinomial classifiers.

## Text categorization with Support Vector Machines : Learning with many relevant features[3]

**Objective**

In the research paper selected, they have explored and identified the benefits of SVM for text classification and categorization. It analyses the specific properties of learning a text data and why SVM bring a fruitful result to min a text. There is a comparison of SVMs using polynomial and RBF kernels with 4 different methods commonly used for text categorization like density estimation using a naive Bayes classifier, Rocchio algorithm, a distance weighted k-nearest neighbor classifier, decision tree/rule learner. After the experiments using all the algorithms on two different datasets, it is proved that SVM is the best for text classification.

**Methodologies used**

- **Text categorization:** The aim of text categorization is to divide the documents in fixed length categories. There can be single, multiple of no categories in document. The aim is to have classifier from examples and which supposed to perform a categorization automatically. Each category examined separately. Step one: document's content supposed to covert to presentational so, that it can be understand and process by the Text categorization and the algorithms. Also an Ordering of document's content has a less importance. Every Wi word related a specific feature, with the number of occurrences is a value. It would create a large feature vector and in order to get rid of it then it consider two conditions like it must occur minimum 3 times and it must not be a stop-words. This would have a large thousands dimensional feature vector and so, many people believed to have a feature selection and to avoid overfitting. To select a subset of feature it will use an information gain as a feature subset. Next step it to scaling the dimensions with

its inverse document frequency would benefits in the performance. To neglect the varied document lengths, every document mapped with unit length.

- **Support Vector Machine:** It is a structural risk minimization problem and its aim is to hypothesis h to have confidence of less error. True error of h is to have an error in h for any selected data. An upper bound in hypothesis h with data loss in h on tainted data and H complexity and hypothesis space has h. SVM minimize this error by controlling Vector dimension of H. SVM has a linear threshold by simply setting a kernel function which ultimately learn RBF networks, polynomial classifier, and sigmoid nets. SVM learn independently of the feature space. It measures complexity by analyzing margin by which it separate the data.

**Results**

So, SVM can give a consistent performance for text categorization which is way good then existing methods to do so. There is no need of feature selection in SVM. It does not require any parameter tuning since it will set suitable parameter automatically. It can work well even in high dimensional feature space.

## Database description

We have experimented RNN algorithm on the Movie-Dialogs Corpus dataset by Cornell University, which has two files called MovieLines.txt and MovieConversations.txt. Dataset is available at Movie-Dialogs Corpus repository.

**MovieLines.txt**
Fields: lineID, characterID, movieID, character name, test of the utterance.

```
L1045 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ They do not!
L1044 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ They do to!
L985 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ I hope so.
L984 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ She okay?
```

Figure 2 MoviLine.txt file snippet

**MovieConversations.txt**
Fields: UserId1, UserId2, MovieId, Conversation Array.

```
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L194', 'L195', 'L196', 'L197']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L198', 'L199']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L200', 'L201', 'L202', 'L203']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L204', 'L205', 'L206']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L207', 'L208']
```

Figure 3 MoviConversation.txt file snippet

**Dataset link:** https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html

## Implementation and Methodology used

### Load and Install packages

a. **Pandas**: We used 'pandas' in our project to work with "relational" or "labeled" data both easy and intuitive.
b. **Numpy**: Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.
c. **Tensorflow**: TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them.
d. **Re:** A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let us check if a particular string matches a given regular expression.
e. **Time:** This module provides various time-related functions.

### Preprocessing

**Step 1: Splitting lines and mapping with Line ID**
In this step we are splitting lines by removing unnecessary data and mapping it with line id.

```
for line in lines:
    _line = line.split(' +++$+++ ')
    if len(_line) == 5:
        id2line[_line[0]] = _line[4]
```

Figure 4 Code snippet for Splitting lines and mapping with Line ID

**Step 2: Build Question and Answer array**
In this step we are separating questions and answers in different arrays.

```
for conv in convs:
    for i in range(len(conv)-1):
        questions.append(id2line[conv[i]])
        answers.append(id2line[conv[i+1]])
```

Figure 5 Code snippet for Build Question and Answer array

**Step 3: Remove unnecessary characters and altering the format of words**
In this step we are removing unnecessary characters such as [ ] ' " , ". Also, we are altering the format of words such as I'm to I am. Its very important to train model with proper data.

```
text = re.sub(r"i'm", "i am", text)
text = re.sub(r"he's", "he is", text)
text = re.sub(r"\'re", " are", text)
text = re.sub(r"[-()\"#/@;:<>{}`+=~|.!?,]", "", text)
```

Figure 6 Code snippet for Remove unnecessary characters and altering the format of words

**Step 4: Remove questions and answers that are < 2 words and > 20 words**

In this step, we are removing very short and very long sentences. The main purpose behind this data clearance is to decrease error rate.

```
for question in clean_questions:
    if len(question.split()) >= min_line_length and
        len(question.split()) <= max_line_length:
        short_questions_temp.append(question)
        short_answers_temp.append(clean_answers[i])
    i += 1
//same code for answer array too
```

Figure 7 Code snippet for Remove questions and answers that are < 2 words and > 20 words

**Step 5: Create a dictionary for the frequency of the vocabulary**

Here we are creating dictionary by counting frequency of the words.

```
for question in short_questions:
    for word in question.split():
        if word not in vocab:
            vocab[word] = 1
        else:
            vocab[word] += 1
//same code for answer array
```

Figure 8 Code snippet for Create a dictionary for the frequency of the vocabulary

**Step 6: Remove rare words**

We have set threshold value 10. So we are removing all words whose frequency is less than 10.

```
threshold = 10
for k,v in vocab.items():
    if v >= threshold:
        count += 1
```

Figure 9 Code snippet for Create dictionaries to provide a unique integer for each word

**Step 7: Create dictionaries to provide a unique integer for each word**

Here we are providing unique integer to each word.

```
for word, count in vocab.items():
    if count >= threshold:
        questions_vocab_to_int[word] = word_num
        word_num += 1
//same code for answer array
```

Figure 10 Code snippet for Create dictionaries to provide a unique integer for each word

**Step 8: Add the unique tokens to the vocabulary dictionaries**

Seq2seq model contains 4 tokens <PAD>, <UNK>, <GO>, <EOS>. Each token have their own purpose. Basically Seq2seq model prefers to have input of same length so we use this tokens for that.

```
codes = ['<PAD>','<EOS>','<UNK>','<GO>']
for code in codes:
    questions_vocab_to_int[code] =
    len(questions_vocab_to_int)+1
 //same code for answer array
```

Figure 11 Code snippet for Add the unique tokens to the vocabulary dictionaries

**Step 9: Create inverse dictionary**

LSTM works best with inverse input so here we are creating inverse dictionary.

```
questions_int_to_vocab = {v_i: v for v, v_i in
                          questions_vocab_to_int.items()}
//same code for answer array
```

Figure 12 Code snippet for Create inverse dictionary

**Step 10: Replace any words that are not in the respective vocabulary with <UNK>**

Here we are replacing words with <UNK> token which are not in vocabulary dictionary.

```
for question in short_questions:
    ints = []
    for word in question.split():
        if word not in questions_vocab_to_int:
            ints.append(questions_vocab_to_int['<UNK>'])
        else:
            ints.append(questions_vocab_to_int[word])
    questions_int.append(ints)
//same code for answer array
```

Figure 13 Code snippet for Replace any words that are not in the respective vocabulary with <UNK>.

**Step 11: Sort questions and answers by the length of questions.**
This help should speed up training and help to reduce the loss.

```
for length in range(1, max_line_length+1):
    for i in enumerate(questions_int):
        if len(i[1]) == length:
            sorted_questions.append(questions_int[i[0]])
            sorted_answers.append(answers_int[i[0]])
for i in range(3):
```

Figure 14 Code snippet for Sort questions and answers by the length of questions.

## Train Sequence to Sequence model [4] [5]

The foundation of sequence modeling tasks, such as machine translation, is language modelling. At a high level, a language model takes in a sequence of inputs, looking at every element of the sequence and try to predict the following element of the sequence. The following formula describe this process.
Yt = f (Yt-1)

Where Yt is the sequence element at time t, Yt-1 is the sequence element at the foregoing time step, and f is a function mapping the foregoing element of the sequence to the upcoming element of the sequence. Since we're using sequence to sequence models using neural networks, f stands for a neural network which predicts the upcoming element of a sequence given the present element of the sequence.

As shown in the figure 1, we have "ABC" as the input sequence, and "WXYZ" as the output sequence. Different length of input and output is the big challenge for Seq2seq learning algorithm. Obviously here we have different length of input and output. So to solve that problem we need to create a model which consists of two separate recurrent neural networks called Encoder and Decoder respectively.

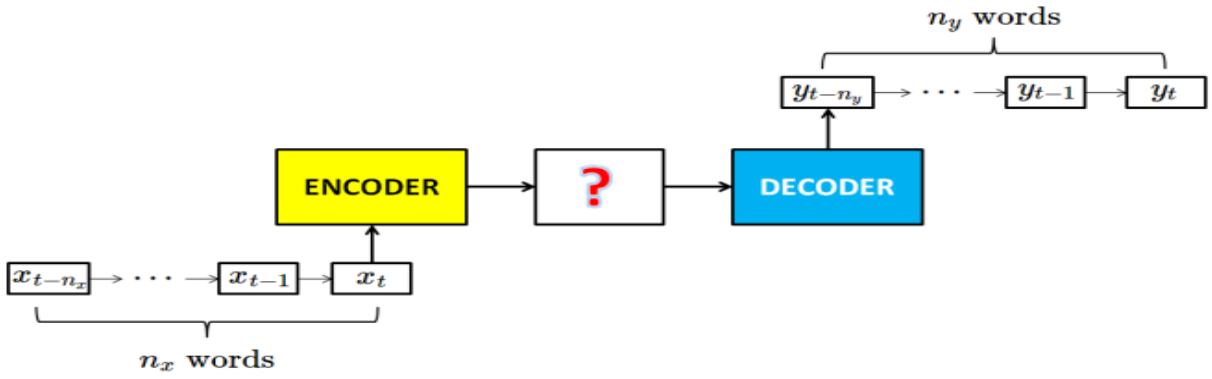


Figure 15 High level view of encoder and decoder

As the names of the two networks are somehow self-explained, first, it's clear that we cannot directly compute the output sequence by using only single network, that is why we need to use the first network (Encoder) to encode the input sequence into some kind of "temporary middle sequence", afterwards the other network (Decoder) will decode that sequence into targeted output sequence. So, it's important to learn methodology of middle sequence. Let's take a look at the next figure.

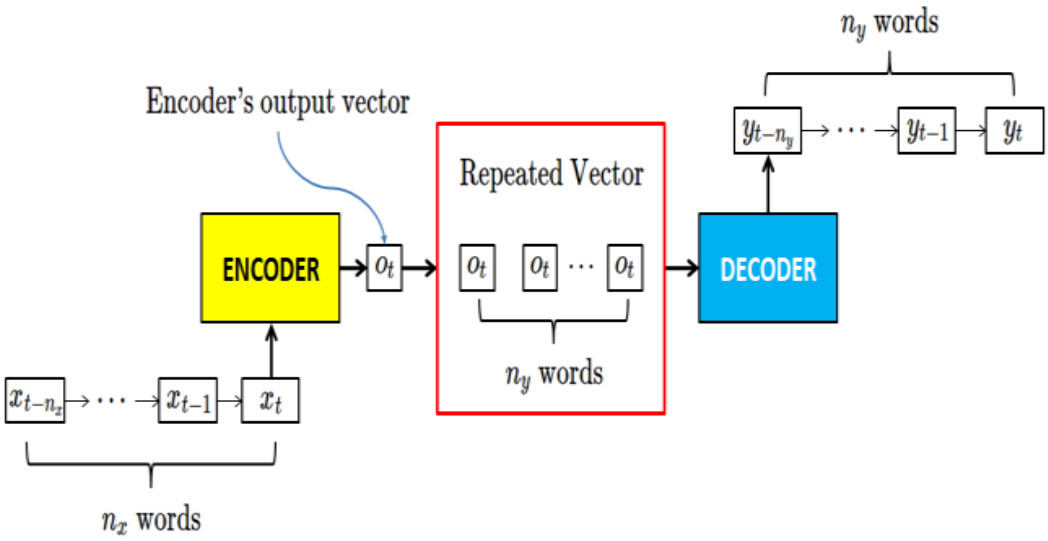


Figure 16 Repeated vector

Concretely, what the Encoder actually did is reads the input sequence, word by word and emits a context and creating a temporary output vector. Then, that vector is repeated nn times, with n is the length of our desire output sequence. Everything is good till this point and works same as general DNN algorithms. Repeated vector will create output sequence of same length by repeating n times. Now it's time for decoder to change that output sequence and build desired sentence. That's the main idea behind the Seq2seq model. So now we have our part of model ready which creates output sequence and input sequence of different length. But we need to take care about input sequence length. It must be of same size length to feed the model. That's where the Seq2Seq model introduces four types of tokens. Let's have deeper look inside it.

**<PAD>**: During training, we'll need to feed our examples to the network in batches. The inputs in these batches all need to be the same width for the network to do its calculation. So to make sentence of same length, we will add <PAD> token in short sentence to make it same length.

**<EOS>**: This is another necessity of batching as well, but more on the decoder side. It allows us to tell the decoder where a sentence ends, and it allows the decoder to indicate the same thing in its outputs as well.

**<UNK>**: At preprocessing step, we have created vocabulary based on word frequency. Here we used threshold value 10. So words which are not frequent will be replaced with <UNK> token.

**<GO>**: This is the input to the first time step of the decoder to let the decoder know when to start generating output.

```
You'll never know the psychopath sitting next to you ZERO.
You'll never know the murderer sitting next to you ZERO.
You'll think, "How'd I get here, sitting next to you?".
But after all I've said ZERO ZERO ZERO ZERO ZERO.
Please don't forget ZERO ZERO ZERO ZERO ZERO ZERO ZERO.
```

Figure 17 Add <PAD> token to equalize sentence length

Here we are adding ZERO in place of <PAD> token to make all sentences of same size.

```
[
    [3,  2,  1,  4,  12,  7,  8,  6,  5,  0],
    [3,  2,  1,  4,  16,  7,  8,  6,  5,  0],
    [3,  2,  1,  4,  16,  14,  16,  16,  16,  5],
    [3,  16,  9,  16,  16,  0,  0,  0,  0,  0],
    [10,  11,  15,  0,  0,  0,  0,  0,  0,  0]
]
```

Figure 18 Representation of sentences using unique ID

We also created integer dictionary by assign unique id to words. So, in figure 18 we have data representation using ids and <PAD> as 0.

Until now we have finished our data processing part and now our input is ready to feed the model and our model is ready to start training. We are using 15% of data for validating the training. But, before going detailed in training, let's discuss about Batching and why its important part. Batches allow us to run a model more efficiently due to scaling by increasing the amount of data you give the model each iteration. As a input it will take dictionary of questions and answers. Also we need to provide batch size which 128 for our model.

```
Epoch   2/100 Batch    0/918 - Loss:  0.365, Seconds: 308.66
Epoch   2/100 Batch  100/918 - Loss:  1.978, Seconds: 319.18
Epoch   2/100 Batch  200/918 - Loss:  1.964, Seconds: 351.13
Epoch   2/100 Batch  300/918 - Loss:  1.973, Seconds: 348.64
Epoch   2/100 Batch  400/918 - Loss:  1.959, Seconds: 349.91
```

Figure 19 Batching

In the figure 19, you can see that our dataset is divided into 918 batches. Now, let's move to the training part. Now we can create the decoder network, which does the main job. First, we need to repeat the single vector outputted from the encoder network to obtain a sequence which has the same length with the output sequences. The rest is similar to the encoder network, except that the decoder will be more complicated, which we will have two or more hidden layers stacked up. Here we are training our model continuously until we will not get 5 consecutive "No Improvement" status.

```
Epoch   5/100 Batch    0/918 - Loss:  0.336, Seconds: 316.11
Epoch   5/100 Batch  100/918 - Loss:  1.834, Seconds: 314.97
Epoch   5/100 Batch  200/918 - Loss:  1.832, Seconds: 330.84
Epoch   5/100 Batch  300/918 - Loss:  1.848, Seconds: 326.84
Epoch   5/100 Batch  400/918 - Loss:  1.836, Seconds: 307.61
Valid Loss:  1.870, Seconds: 147.23
New Record!
Epoch   5/100 Batch  500/918 - Loss:  1.843, Seconds: 310.23
Epoch   5/100 Batch  600/918 - Loss:  1.881, Seconds: 332.45
Epoch   5/100 Batch  700/918 - Loss:  1.882, Seconds: 346.66
Epoch   5/100 Batch  800/918 - Loss:  1.879, Seconds: 354.66
Epoch   5/100 Batch  900/918 - Loss:  1.854, Seconds: 376.85
Valid Loss:  1.902, Seconds: 147.46
No Improvement.
```

Figure 20 Model training

## RNN Algorithm[4] [5] [7]

RNN(Recurrent Neural Network) algorithm is heart of our model. All magic happens here. This algorithm is widely used for sequential data structure.

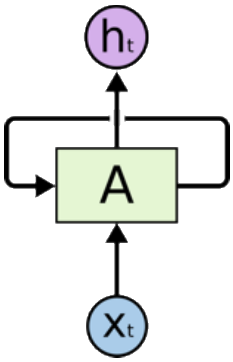


Figure 21 Chunk of Neural network

The figure 21 single chunk of neural network, A, input xt and outputs a value ht. A loop allows information to be passed from one step of the network to the next. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. These loops make recurrent neural networks seem kind of mysterious. It turns out that they aren't all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. In figure 22 we can see loop structure of RNN:

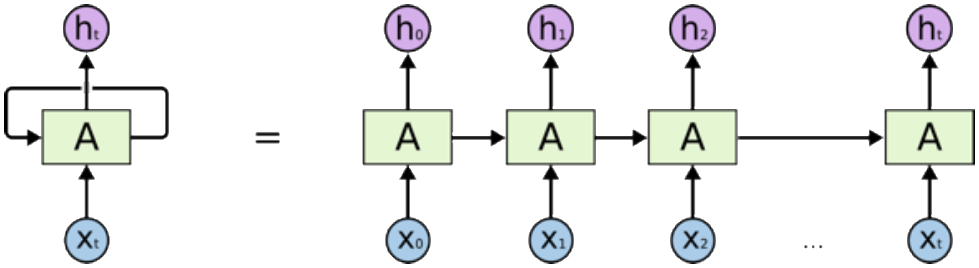


Figure 22 Loop structure of RNN

This chain type nature reveals that recurrent neural networks are closely related to sequences and lists. They're the natural architecture of neural network to use for such data. The naive version of RNN, is typically called a Vanilla RNN, which is pretty pathetic in remembering long sequences. The main problem with basic version of RNN is the Problem of Long-Term Dependencies.

**The Problem of Long-Term Dependencies [6]:** One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous word might inform the understanding of the next word. The vanilla RNN can't remember previous information for long term.

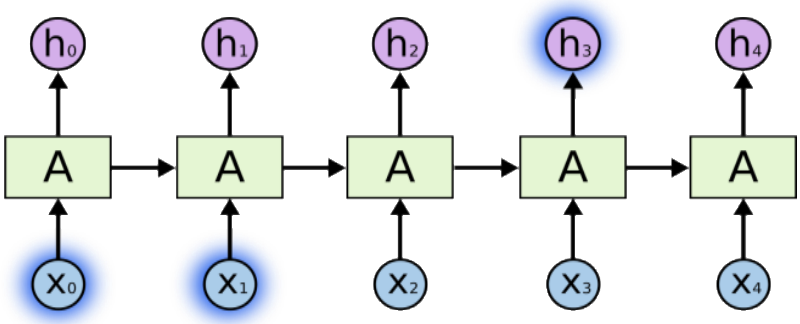


Figure 23 Short-term data persistance

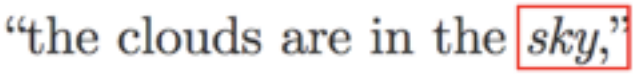


Figure 24 Short dependencies example

In the figure 23, 24, we can see that we are predicting word "Sky" based on previous word "Cloud". Vanilla RNN can able to do this very easily because the gap between "Cloud" and "Sky" is very small. Lets take more complex data persistence example and check the problem with Vanilla RNN.

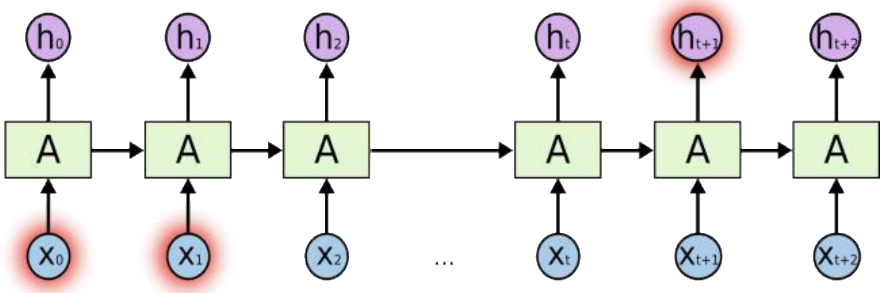


Figure 25 Problem of Long-Term Dependencies

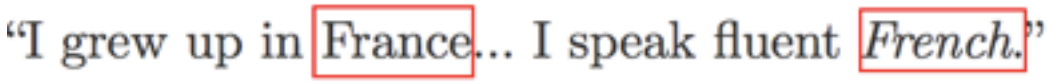


Figure 26 Long dependencies example

In the figure 26, There are many other sentences in between of first and last sentences. So we have one intentional word in first sentence "France" which is most useful to predict language spoken by that person. So at the time of prediction of "French", gap between those words are too big and naive RNN couldn't able to persist data till now. That's the problem of Long-Term Dependencies.

To overcome this problem, we have used special kind of RNN which is LSTM (Long-Short Term Memory) networks. Instead of having a single neural network layer, there are four, interacting in a very special way.

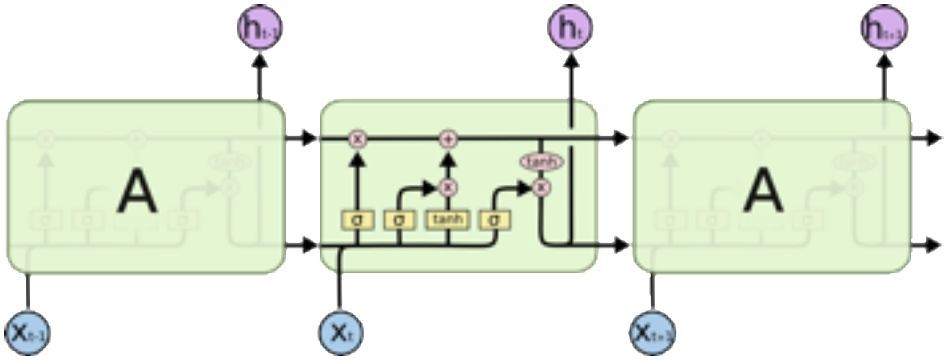


Figure 27 LSTM Cell

An LSTM cell consists of multiple gates, for remembering useful information, forgetting unnecessary information and carefully exposing information at each time step. The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at ht−1and xt, and outputs a number between 00 and 11 for each number in the cell state Ct−1. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."

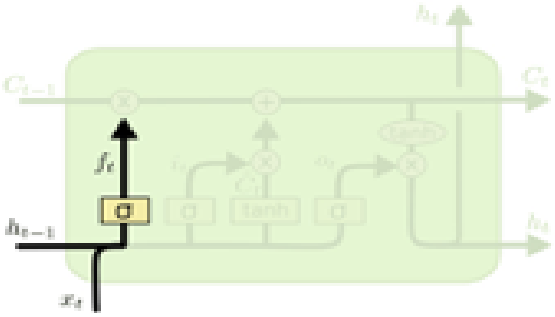


Figure 28 Forget gate layer

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, C̃t, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

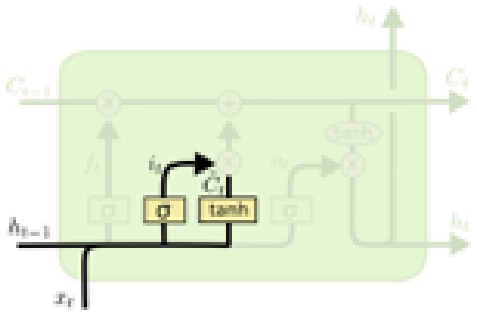


Figure 29 Input gate layer (I)

It's now time to update the old cell state, Ct−1, into the new cell state Ct. The previous steps already decided what to do, we just need to actually do it. We multiply the old state by ft, forgetting the things we decided to forget earlier. Then we add it∗C̃t. This is the new candidate values, scaled by how much we decided to update each state value.

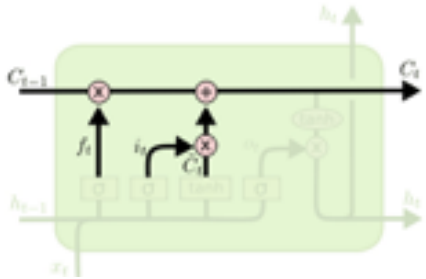

Figure 30Input gate layer (II)

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between −1−1 and 11) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

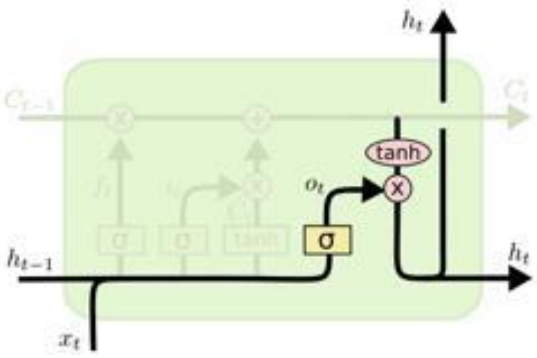


Figure 31 Sigmoid gate

## Experimental results

We applied our method to the Cornell movie corpus in two ways. We used it to directly predict the input sentence without using a SMT system and rescore the n best lists of an SMT baseline. We report the accuracy of these prediction methods, present sample predicted texts, and present the resulting sentences.

**Decoding and Rescoring** : The main part of our experiments is training a large deep LSTM on many conversation pairs. We trained our model by maximizing the log probability of a correct prediction T given the input sentence S, so the objective is

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

where S is the training set. After training completion, we generate prediction by finding the most suitable prediction according to the LSTM:

$$\hat{T} = \arg\max_{T} p(T|S)$$

We search for the most likely prediction using a simple left-to-right sigmoid and tanh search decoder which maintains sigmoid states of partial hypotheses, where a partial hypothesis is a prefix of some prediction. At each timestep we pass each partial prediction in the sigmoid layer with every possible word in the vocabulary. Sigmoid layer maintains state of the words of vocabulary to decide weather to pass to next sigmoid layer of to remove it. As soon as the "<EOS>" symbol is added to the hypothesis, it is removed from the sigmoid state and is added to the set of complete prediction text.

**Reversing the Source Sentences :** As we know, the LSTM is capable of solving problems with long term dependencies, we realized that the LSTM learns much faster and better when the input sentences are reversed. Normally, when we concatenate a input sentence with a output sentence, each word in the input sentence is far from its corresponding word in the output sentence. As a result, the problem has a increased "minimal time lag". So, by reversing the words in the input sentence, the average distance between corresponding words in the input and output sentence is same. However, the first few words in the input sentence are now very close to the first few words in the target sentence, so the problem's minimal time lag is greatly decreased.

**Training details :** As per our understanding, we conclude that the LSTM models are quite easy to train. We used deep LSTMs with 4 layers, with 1000 cells at each layer and 1000 dimensional word embeddings, with an input sentence of 304,713 and an output sentences of around 135,000. We initialized all of the LSTM's parameters with the uniform distribution. We used batches of 128 sequences for the input and divided it the size of the batch. Different sentences have different lengths. Most sentences are short but some sentences are long. It wastes lot of time to train model so we converted our all sentences to same length.

**Result:** One of the attractive features of our model is its ability to turn a sequence of words into a vector of fixed dimensionality.

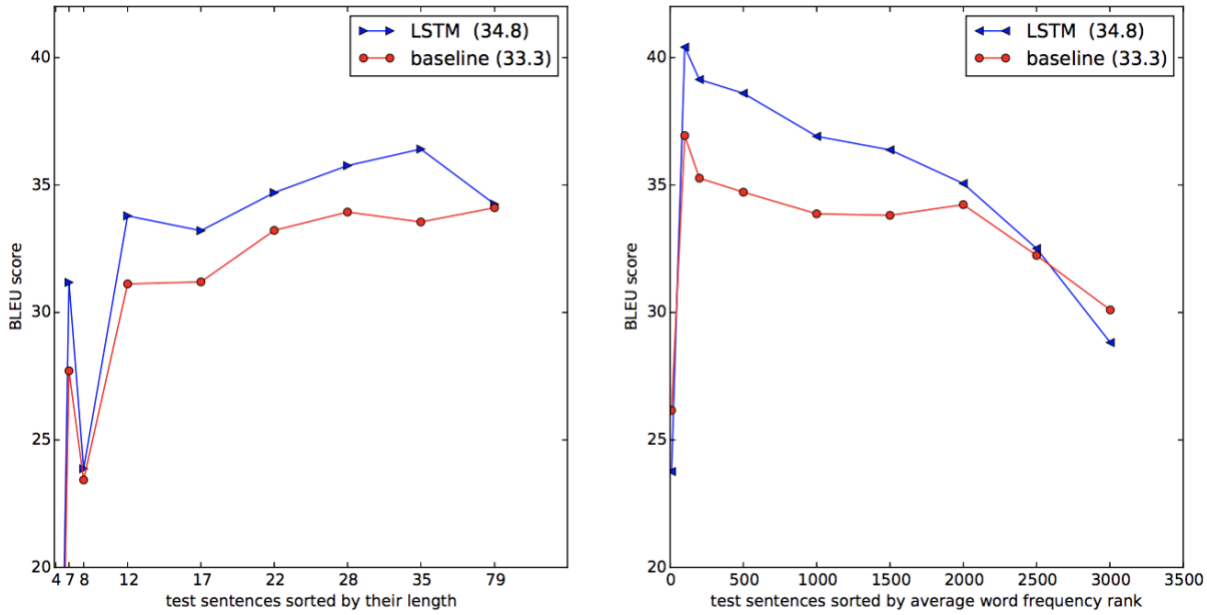


Figure 32 Plot based on sentence length

The left plot shows the performance of our system as a function of sentence length, where the x-axis corresponds to the test sentences sorted by their length and is marked by the actual sequence lengths. There is no degradation on sentences length, there is only a minor degradation on the longest sentences. The right plot shows the LSTM's performance on sentences with progressively more rare words, where the x-axis corresponds to the test sentences sorted by their "average word frequency rank".

```
Epoch   5/100 Batch    0/918 - Loss:  0.336, Seconds: 316.11
Epoch   5/100 Batch  100/918 - Loss:  1.834, Seconds: 314.97
Epoch   5/100 Batch  200/918 - Loss:  1.832, Seconds: 330.84
Epoch   5/100 Batch  300/918 - Loss:  1.848, Seconds: 326.84
Epoch   5/100 Batch  400/918 - Loss:  1.836, Seconds: 307.61
Valid Loss:  1.870, Seconds: 147.23
New Record!
Epoch   5/100 Batch  500/918 - Loss:  1.843, Seconds: 310.23
Epoch   5/100 Batch  600/918 - Loss:  1.881, Seconds: 332.45
Epoch   5/100 Batch  700/918 - Loss:  1.882, Seconds: 346.66
Epoch   5/100 Batch  800/918 - Loss:  1.879, Seconds: 354.66
Epoch   5/100 Batch  900/918 - Loss:  1.854, Seconds: 376.85
Valid Loss:  1.902, Seconds: 147.46
No Improvement.
```

Figure 33 Result for training

To reach at the maximum accuracy level, our model trained till 12 epochs and those 12 epochs includes 5 consecutive "No Improvement".

```
Question
  Word Ids:      [7622, 4978, 4835, 4676]
  Input Words: ['where', 'are', 'you', 'going']

Answer
  Word Ids:      [4363, 6430, 3954, 3023, 8097]
  Response Words: ['i', 'am', 'going', 'to', 'vegas', '<EOS>']
```

Figure 34 Result for Question and Answer

After providing input in reverse order, algorithm works better. Processing time was low and generated accurate output.

## Discussion & Suggestions for future

When RNN become larger and complex then it increases the single network computation time for weeks or even months. There is a notable motivation to bring an advancement and scale this network. Transfer entropy helps to efficient and faster data transfer between two or more processes. continuous modification to improve the quality of a reservoir but it will not consider the system tasks. We can improve the hidden layer of the RNN by the learning system. A reservoir adaptation will increase the information transfer at each unit by considering information transfer between output and input of the system. Using production data, a reservoir adaptation advances performance for supervised learning.

## Conclusions

Seq2Seq is the most diverse model used and the best fitted model for Chatbot. The input is often reversed. This helps a model to produce better outputs because when the input data is being fed into the model, the start of the sequence will now become closer to the start of the output sequence. LSTM version of RNN is best fitted for Larger Distance Problems that usually Chatbot faces.

## References


1) Ilya Sutskever, Oriol Vinyals, Quoc V. Le, Sequence to Sequence Learning with Neural Networks
Link : http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf
2) “Some Effective Techniques for Naive Bayes Text Classification,” IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 18, -NO. 11, NOVEMBER 2006, pp.1457-1465.
3) “Text categorization with Support Vector Machines: Learning with Many Relevant Features,” Universität Dortmund informatik LS8,Baroper Str. 301 44221 Dortmund Germany, PP. 137-142.
4) https://chunml.github.io/ChunML.github.io/project/Sequence-To-Sequence/
5) https://indico.io/blog/sequence-modeling-neuralnets-part1/

6) https://deeplearning4j.org/lstm.html
7) http://colah.github.io/posts/2015-08-Understanding-LSTMs/

## Details of Division of the work

| Student Name | Student Id | Student Email | Word Done |
|---|---|---|---|
| Samir Anghan | 40040308 | samir.anghan@gmail.com | ✓ Write code<br>✓ Research using papers<br>✓ Feature Extraction<br>✓ Train and Evaluate Models<br>✓ Post-processing |
| Mihir Piyushkumar Pujara | 40025592 | mihirppujara@gmail.com | ✓ Write code<br>✓ Research using papers<br>✓ Feature Extraction<br>✓ Train and Evaluate Models<br>✓ Post-processing |
| Gauravkumar Parvadiya | 40040310 | gaurav.parvadiya@gmail.com | ✓ Write code<br>✓ Research using papers<br>✓ Feature Extraction<br>✓ Train and Evaluate Models<br>✓ Post-processing |
| Hirangi Naik | 40041365 | hirangi.naik@gmail.com | ✓ Write code<br>✓ Setting up the environment<br>✓ Dataset Evaluation<br>✓ Dataset Filtering, Inspection<br>✓ Pre-processing |