

**Table of Contents**

1. Abstract……………………………………………………………………….3
2. Introduction…………………………………………………………………3
3. Main Objective…………………………………………………………….3
4. History of Chatbot systems………………………………………….4
   1. Rule-based
   2. IR based
5. State of the Art Technology………………………………………….4
6. Background and Methodology……………………………………..5
   1. Recurrent Neural Networks
   2. LSTM
   3. Sequence to Sequence models
      1. Training
      2. Testing
   4. Attention Mechanism
7. Experimental Analysis ………………………………………………….12
   1. Dataset
   2. Implementation
   3. Experiments
8. Results…………………………………………………………………………14
9. Conclusion …………………………………………………………………..15
10. References …………………………………………………………………16
11. Appendix……………………………………………………………………..17

**1.Abstract**

The ability of a machine to communicate with humans has long been associated with the general success of AI. Modelling conversation is an important task in natural language processing and artiﬁcial intelligence (AI). Indeed, ever since the birth of AI, creating a good chatbot remains one of the ﬁeld’s hardest challenges. While chatbots can be used for various tasks, in general, they have to understand users’ utterances and provide responses that are relevant to the problem. In the past, methods for constructing chatbot architectures have relied on hand-written rules, and or simple statistical methods. With the rise of deep learning these models were quickly replaced by end-to-end trainable neural networks. More speciﬁcally, the recurrent encoder-decoder model. This architecture was adapted from the neural machine translation domain, where it performs extremely well. Recently, the emergence of neural network models the potential to solve many of the problems in dialogue learning that earlier systems cannot tackle: the end-to-end neural frameworks offer the promise of scalability and language-independence, together with the ability to track the dialogue state and then mapping between states and dialogue actions in a way not possible with conventional systems.

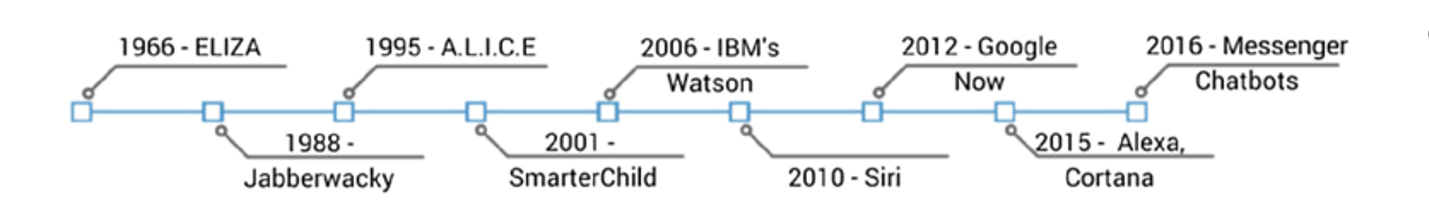
**2.Introduction**

A conversational chatbot is a piece of software that can communicate with humans using natural language. Modelling conversations remains one of the ﬁeld’s toughest challenges. Even though they are far from perfect, chatbots are now used in a lot of applications like Apple’s Siri [Apple, 2017], Google’s Google Assistant [Google, 2017], or Microsoft’s Cortana [Microsoft, 2017].Neural machine translation is a widely used approach to machine translation, proposed by Kalchbrenner and Blunsom (2013), Sutskever et al. (2014), and Cho et al. (2014b). Unlike the traditional phrase-based translation system which consists of many small sub-components that are tuned separately, neural machine translation attempts to build and train a single, large neural network that reads a sentence and outputs a correct translation. Most of the proposed neural machine translation models belong to a family of encoder– decoders, with an encoder and a decoder for each language, or involve a language-speciﬁc encoder applied to each sentence whose outputs are then compared. An encoder neural network reads and encodes a source sentence into a ﬁxed-length vector. A decoder then outputs a translation from the encoded vector. The whole encoder-decoder system, which consists of the encoder and the decoder for a language pair, is jointly trained to maximize the probability of a correct translation given a source sentence. So this encoder-decoder mechanism can be used for the goal-oriented task. We are adapting the neural machine translation approach and applying it in a chatbot for end-to-end dialogues.

**3.Main Objective**

Build a task-oriented conversational Chatbot that cansimulate human conversation via text using Sequence to Sequence mechanism. (Task: Hotel recommendation)

**4.History of Chatbots**

Chatbot models usually take as input natural language sentences uttered by a user and output a response. There are two main approaches to generating responses. The traditional approach is to use hard-coded templates and rules to create chatbots.

* 1. **The Rule-based Systems**

ELIZA is one of the ﬁrst ever chatbot programs written [Weizenbaum, 1966]. It uses clever handwritten templates to generate replies that resemble the user’s input utterances. Since then, countless hand-coded, rule-based chatbots have been developed[Wallace, 2009, Carpenter, 2017, Worswick, 2017].

These chatbot programs are very similar in their core, namely that they all use hand-written rules to generate replies. Usually, simple pattern matching or keyword retrieval techniques are employed to handle the user’s input utterances. Then, rules are used to transform a matching pattern or a keyword into a predeﬁned reply.

* 1. **The IR-based Systems**

The IR-based methods rely on information retrieval or nearest neighbour techniques (Isbell et al., 2000). Given a history input and a training corpus, the system copies a response from the training corpus. The response selection process is usually based on the combination

**5.State of the art Technology**

It started with Rule-based system to statistical methods and thereafter Deep networks such as RNN and LSTM’s as a vanilla Encoder-Decoder. After that attention mechanism is Encoder-Decoder models gave more accurate in neural machine translation and Currently The Transformer is an encoder-decoder model based solely on attention mechanisms and feedforward neural networks, achieving state-of-the-art results in NMT tasks [Vaswani et al., 2017].

**6.Background and Methodology**

The main concept that differentiates rule-based and neural network-based approaches is the presence of a learning algorithm in the latter case. An important distinction has to be made between traditional machine learning and deep learning which is a sub-ﬁeld of the former. In this work, only deep learning methods applied to chatbots are discussed, since neural networks have been the backbone of conversational modeling and traditional machine learning methods are only rarely used as supplementary techniques.

When applying neural networks to natural language processing (NLP) tasks each word (symbol) has to be transformed into a numerical representation [Bengio et al., 2003]. This is done through word embeddings, which represent each word as a ﬁxed size vector of real numbers. Word embeddings are useful because instead of handling words as huge vectors of the size of the vocabulary, they can be represented in much lower dimensions. Word embeddings are trained on large amounts of natural language data and the goal is to build vector representations that capture the semantic similarity between words. More speciﬁcally, because a similar context usually is related to similar meanings, words with similar distributions should have similar vector representations. This concept is called the Distributional Hypothesis[Harris,1954]. Each vector representing a word can be regarded as a set of a parameter that can be jointly learned with the neural network’s parameters, or they can be pre-learned.

Neural network-based conversational models can be further divided into two categories, retrieval-based and generative models. The former simply returns a reply from the dataset by computing the most likely response to the current input utterance based on a scoring function, which can be implemented as a neural network [Cho et al., 2014] or by simply computing the cosine similarity between the word embeddings of the input utterances and the candidate replies[Lietal., 2016d]. Generative models, on the other hand, synthesize the reply one word at a time by computing probabilities over the whole vocabulary [Sutskever et al., 2014, Vinyals and Le, 2015].

Sequence-to-Sequence models can be viewed as a basic framework for generating a target sentence based on source inputs, which can be adapted to a variety of natural language generation tasks, for example, generating a French sentence given an English sentence in machine translation; generating an answer given a question in question-answering; generating a summary given a document in summarization, etc. For Sequence-to-Sequence models, the basics of language models, recurrent neural networks, and the Long Short-term Memory, which can be viewed as the fundamental components of Sequence-to-Sequence models and There are algorithmic variations of **Sequence-to-Sequence models, such as attention mechanism**.

* 1. **Recurrent Neural Networks**

Recurrent neural networks (RNN) are a neural network architecture that is specifically designed to handle sequential data. It was historically used to handle time-sequential data [(Elman, 1990;](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.3znysh7) [Funahashi and Nakamura, 1993),](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.2et92p0) and have been successfully applied to language processing. Given a sequence of word tokens *{y*1*, y*2*, ..., yN }*, where each word *yt, t ∈* [1*, N* ] is associated with a K-dimensional vector representation *xt*. RNN associates each time step *t* with a hidden vector representation *ht*, which can be thought of as a representation that embeds all information of previous tokens, i.e., *{y*1*, y*2*, ..., yt}*. *ht* is obtained using a function *g* that combines the previously built presentation for the previous time-step t-1, denoted as *ht−*1, and the representation for the word of current time-step *xt*:

*ht* = *g*(*ht−*1*, xt*) (6.1)

The function g can take different forms, with the simplistic one being as follows:

*g*(*ht−*1*, xt*) = *σ*(*Whh · ht−*1 + *Wxh · xt*) (6.2)

where *Whh, Wxh ∈* R*K×*2*K* . Popular choices of *σ* are non-linear functions such as sigmoid,tanh, or ReLU. From Equ. [6.2,](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.tyjcwt) we can see that the dimensionality of *ht* is constant for different *t*.

* 1. **Long Short Term Memory**

Two serve issues problems with RNNs are the gradient *exploding* problem and the gradient *vanishing* problem [(Bengio et al., 1994),](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.3dy6vkm) where gradient *exploding* refers to the situation where the gradients become very large when the error from the training objective function is backpropagated over time, and gradient *vanishing* refers to the situation where the gradients approach zero when the training error is backpropagated over a few time-steps. These two issues render RNN models incapable of capturing the long-term dependency for long sequences.

One of the most effective ways to alleviate this problem is the Long Short Term Memory model, LSTM for short, first introduced in [Hochreiter and Schmidhuber (1997),](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.1t3h5sf) and adapted, used, and further explored by many others. The key idea of LSTMs is to associate each time step with different types of gates, and these gates provide flexibility in controlling informational flow: to control how much information on the current RNN wants to preserve through forget gates; to control how much information RNN want to receive through the input of current time-step through input gates; and how much information a RNN wants to output to the next time-step through output gates.

More formally, given a sequence of inputs *{y*1*, y*2*, ..., yN }*, where each word *yt* is associated with a *K*-dimensional vector representation *xt*, an LSTM associates each time step with an input gate, a memory gate, and an output gate, respectively denoted as *it*, *ft*, and *ot*. *ct* is the cell state vector at time *t*, and *σ* denotes the sigmoid function. Then, the vector representation *ht* for each time step *t* is given by:

*it* = *σ*(*Wi ·* [*ht−*1*, xt*]) (6.3)

*ft* = *σ*(*Wf ·* [*ht−*1*, xt*]) (6.4)

*ot* = *σ*(*Wo ·* [*ht−*1*, xt*]) (6.5)

*lt* = tanh(*Wl ·* [*ht−*1*, xt*]) (6.6)

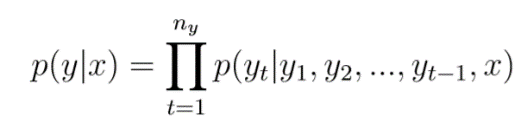
*ct* = *ft ◦ ct−*1 + *it ◦ lt* (6.7)

*ht* = *ot ·* tanh(*ct*) (6.8)

where *Wi*, *Wf* , *Wo*, *Wl ∈* R*K×*2*K* , and *◦* denotes the pairwise dot between two vectors. Again, as in RNNs, *ht* is used as a representation for the partial sequence *{y*1*, y*2*, ..., yt}*.

* 1. **Sequence-to-Sequence model**

The SEQ2SEQ model can be viewed as an extension of the language model, where *y* is the target sentence, and the prediction of current word *yt* in *y* depends not only on all preceding words *{y*1*, y*2*, ..., yt−*1*}*, but also on a source input *x*. Each sentence concludes with a special end-of-sentence symbol *EOS*. For example, in French-English translation, the English word to predict not only depends on all the preceding words but also depends on the original French input; as another example, the following word in a dialogue response depends both on preceding words in the response and the message input. The SEQ2SEQ model was first proved to yield good performance in machine translation.

More formally, in SEQ2SEQ generation tasks, each input *x* = *{x*1*, x*2*, ..., xnx}* is 

( 6 . 9 )

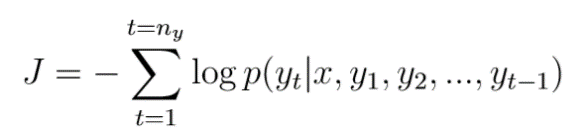
paired with a sequence of outputs to predict: *y* = *{y*1*, y*2*, ..., yny }*. A SEQ2SEQ generation model defines a distribution over outputs and sequentially predicts tokens using a softmax function:

By comparing [Eq.6.9](#_gjdgxs) with the conditional probability of language modelling we can see that the only difference between them is the additional consideration of the source input *x*.

More specifically, a standard SEQ2SEQ model consists of two key components, an encoder, which maps the source input *x* to a vector representation, and a decoder, which generates an output sequence based on the source sentence. Both the encoder and the decoder are multi-layer LSTMs. To enable the encoder to access information from the encoder, the last state memory of the encoder is passed to the decoder as the initial memory state, based on which words are sequentially predicted using a softmax function. Commonly, input and output use different LSTMs with separate compositional parameters to capture different compositional patterns.

* + 1. **Training**

Given a training dataset where each target *y* is paired with a source *x*, the learning objective is to minimize the negative log-likelihood of predicting each word in the target *y* given the source *x*:



**( 6 . 10 )**

Parameters including word embeddings and LSTMs’ parameters are usually initialized from a uniform distribution and learned and optimized using mini-batch stochastic gradient descent with momentum. Gradient clipping is usually adopted by scaling gradients when the norm exceeded a threshold[2](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.4d34og8) to avoid gradient explosion. Learning rate is gradually decreased towards the end of the training.

* + 1. **Testing**

Using a pre-trained model, we need to generate an output sequence *y* given a new input *x*. The problem can be formalized as a standard search problem: generating a sequence of tokens with the largest probability assigned by the pre-trained model, where standard greedy search and beam search can be immediately used. For a greedy search, at each time step, the model picks the word with the largest probability. For beam search with beam size *K*, for each time-step, we expand each of the *K* hypotheses by *K* children, which gives at *K × K* hypotheses. We keep the top *K* ones, delete the others, and move on to the next time-step.

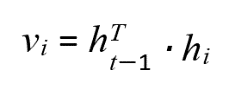
During decoding, the algorithm terminates when an *EOS* token is predicted. At each time step, either a greedy approach or beam search can be adopted for word prediction. Greedy search selects the token with the largest conditional probability, the embedding of which is then combined with preceding output to predict the token at the next step.

* 1. **Attention Mechanisms**

The standard version SEQ2SEQ model only uses the source representation once, which is through initializing the decoder LSTM using the final state of the encoder LSTM. It is challenging to handle long-term dependency using such a mechanism: the hidden state of the decoder LSTM changes over time as new words are decoded and combined, which dilutes the influence from the source sentence.

One effective way to address such an issue is using attention mechanisms [Bahdanau](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.2s8eyo1) [et al. (2014);](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.2s8eyo1) [Xu et al. (2015);](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.17dp8vu) [Jean et al. (2015);](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.3rdcrjn) [Luong et al. (2015b);](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.26in1rg) [Mnih et al. (2014);](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.lnxbz9) [Chorowski et al. (2014).](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.35nkun2) The attention mechanisms adopt a look-back strategy by linking the current decoding stage with each input time-step in an attempt to consider which part of the input is most responsible for the current decoding time-step.

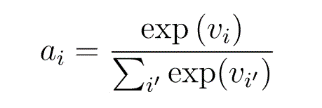
More formally, suppose that each time-step of the source input is associated with a vector representation *hi, i ∈* [1*, nx*] computed by LSTMs, where *nx* denotes the length of the source sentence. *hi ∈* R*K×*1. At the current decoding time-step *t*, attention models would first link the current step decoding information *ht−*1 *∈* R*K×*1 with each of the input time step, characterized by a strength indicator *vi*:



(6 .11)

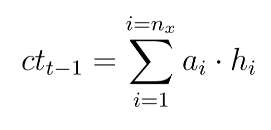
Other than using the dot product to compute the strength indicator *vi*, many other mechanisms have been used such as vector concatenation,

where *vi* = tanh(*W ·* [*ht−*1*, hi*]), *W ∈* R*K×*2*K* , or the general dot-product mechanism, where *vi* = *hT·W ·hi*, *W ∈* R*K×K* , as detailedly explored in [(Luong et al., 2015b).](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.26in1rg) *vt* is then normalized to a probabilistic value *ai* using a softmax function:



( 6.12)

The context vector *ctt−*1 is the weighted sum of hidden memories on the source side:



( 6.13)

As can be seen from Eq. 6.13, a larger value of strength indicator ai indicates a more contribution to the context vector.

The vector representation used to predict the upcoming word *h*ˆ*t−*1, is obtained by combining *ctt−*1 and *ht−*1:

*h*ˆ*t−*1 = tanh(*W*ˆ [*ctt−*1*, ht−*1]) (6.14)

*p*(*yt|y*1*, y*2*, ..., yt−*1*, x*) = softmax(*W* [*yt,* :] *· h*ˆ*t−*1) (6.15)

where *W*ˆ *∈* R*K×*2*K*

*W ∈* R*V ×K* , with *V* the vocabulary size and *K* being the dimensionality of the word vector representation. The context vector *ctt−*1 is not only used to predict the upcoming word *yt*, but also forwarded to the LSTM operation of the next step [(Luong et al., 2015b):](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.26in1rg)

*it* = *σ*(*Wi ·* [*ht−*1*, xt, ctt−*1]) (6.16)

*ft* = *σ*(*Wf ·* [*ht−*1*, xt, ctt−*1]) (6.17)

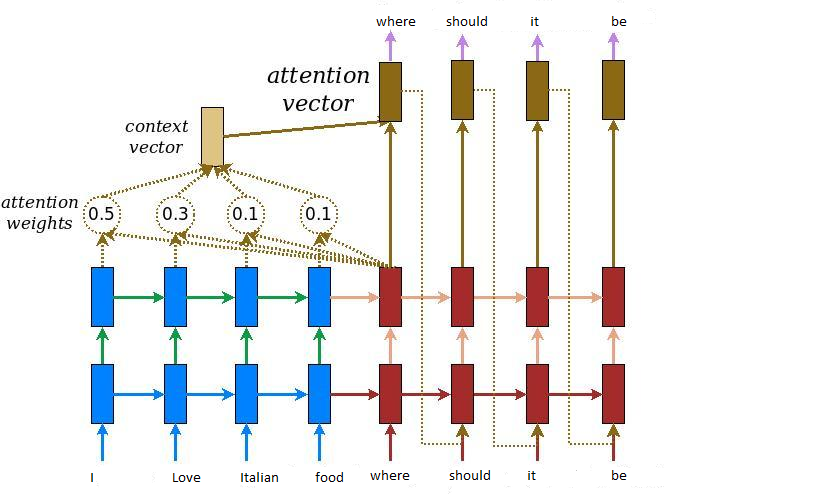
*ot* = *σ*(*Wo ·* [*ht−*1*, xt, ctt−*1]) (6.18)

*lt* = tanh(*Wl ·* [*ht−*1*, xt, ctt−*1]) (6.19)

*ct* = *ft ◦ ct−*1 + *it ◦ lt* (6.20)

*ht* = *ot ·* tanh(*ct*) (6.21)

where *Wi*, *Wf* , *Wo*, *Wl ∈* R*K×*3*K* .



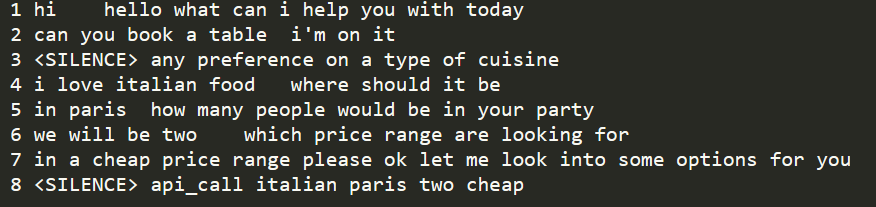
**7.experimental analysis**

* 1. **Datasets**

**The (6) dialog Babi tasks**

Dataset is related to the Babi project of Facebook AI Research which is organized towards the goal of automatic text understanding and reasoning. It has a set of 6 tasks for testing end-to-end dialog systems in the restaurant domain.

As Babi is a synthetic dataset, you may ask why we're interested in doing well on it, or why we even created it at all. Real-world data is noisy. Rarely does it provide a clear and simple answer for you to train on. Additionally, even a well-curated dataset from the real world is littered with nuance, complexities, and errors. By using an artificial world we know the exact state the world is in and the exact set of rules by which it runs. With the synthetic dataset, all the commonsense knowledge and reasoning required for the test set should be contained in the training set. That way, if a machine learning model then fails to solve the task, we know that the challenge is in the model itself, and not the data (or lack of data) it was exposed to.



* 1. **Implementation**
     1. **Preprocessing Steps**
* Concatenate two or more sentences if the answer has two or more of them.
* Remove unwanted data types that are produced while parsing the data.
* Append <START> and <END> to all the answers.
* Create a Tokenizer and load the whole vocabulary ( questions + answers ) into it.

### **Preparing data for Seq2Seq model**

Our model requires three arrays namely encoder\_input\_data, decoder\_input\_data and decoder\_output\_data.

For encoder\_input\_data:

* Tokenize the questions. Pad them to their maximum length.

For decoder\_input\_data:

* Tokenize the answers. Pad them to their maximum length.

For decoder\_output\_data :

* Tokenize the answers. Remove the first element from all the tokenized\_answers. This is the <start> element which we added earlier.
  + 1. **Defining Encoder-Decoder model**

The model will have Embedding, LSTM, and Dense layers. The basic configuration is as follows.

* 2 Input Layers : One for encoder\_input\_data and another for decoder\_input\_data.
* Embedding layer: For converting token vectors to fix sized dense vectors.
* LSTM layer: Provide access to Long-Short Term cells.
  + 1. **Defining inference models**
* Encoder inference model
* Decoder inference model
  1. **Experiments**

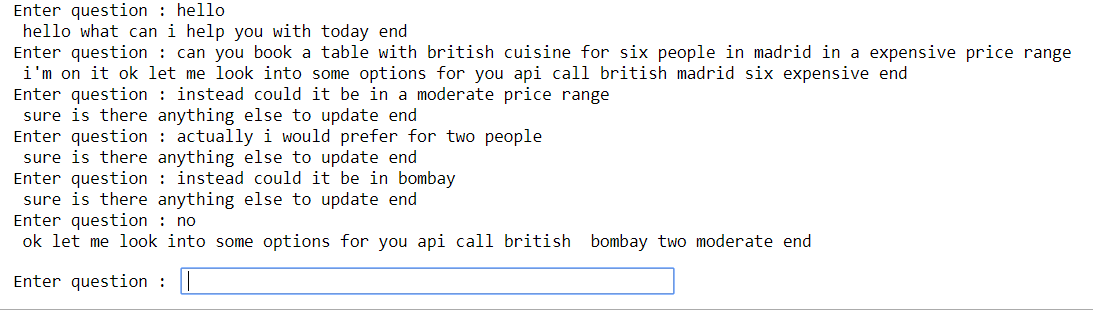
We have used Recurrent Neural Networks (LSTMs), which is the go-to architecture to solve Seq2Seq problems coupled with the Attention mechanism make a generative chatbot. We tried both approaches with attention and without attention.

1. **Results**

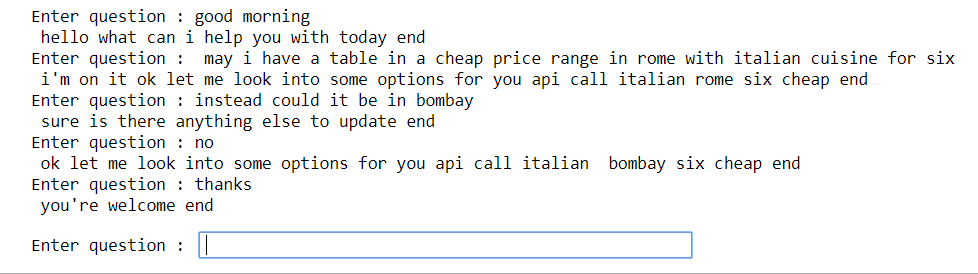
The below output is for Sequence to Sequence (without attention), for most of the cases it was working fine but for few cases when dependencies become longer it was forgetting the main context.

To solve this problem after inferencing we have added a processing section that helps to find actual API calls, and results were perfect.

**Output 1:**



**Output 2:**



1. **Conclusion**

Task-oriented dialog agents are more narrow about language understanding and conversation topics, but they make up with robustness, making them better suited for commercial deployment. It fails when we enter some random text. They learn from smaller and task-specific datasets. Since it is usually not required to generalize to other domains, learning on smaller, but more focused datasets gives much better performance. Also, rule-based systems often perform equally well at task-oriented conversations, since only certain types of questions related to the tasks are expected from the user.

The issue with Babi tasks is It does not represent real-life interactions between users and task-oriented dialog systems. For conversational models to remain robust they have to able to handle out of domain utterances*.* One approach to handle these issues is to build a model that can differentiate between in-domain and non-task user utterances. Then, two different models trained separately on open-domain and task-specific datasets can produce response candidates and a scoring function can output the most probable response [[Akasaki and Kaji, 2017].](https://docs.google.com/document/d/1967GBQPsx0Sk71vSPHnT0VFVPbRtyIoI-d2Rab9PsSs/edit#heading=h.1ksv4uv)

1. **. References**

[Akasaki and Kaji, 2017]Akasaki, S. and Kaji, N. (2017). Chat detection in an intelligent assist- ant: Combining task-oriented and non-task-oriented spoken dialogue systems. *arXiv preprint arXiv:1705.00746*.

[Chiu et al., 2017]Chiu, C.-C., Lawson, D., Luo, Y., Tucker, G., Swersky, K., Sutskever, I., and Jaitly, N. (2017). An online sequence-to-sequence model for noisy speech recognition. *arXiv preprint arXiv:1706.06428*.

[Goyal et al., 2017]Goyal, K., Neubig, G., Dyer, C., and Berg-Kirkpatrick, T. (2017). A continu- ous relaxation of beam search for end-to-end training of neural sequence models. *arXiv preprint arXiv:1708.00111*.

[Konstas et al., 2017]Konstas, I., Iyer, S., Yatskar, M., Choi, Y., and Zettlemoyer, L. (2017). Neural amr: Sequence-to-sequence models for parsing and generation. *arXiv preprint arXiv:1704.08381*.

[Olah, 2015]Olah, C. (2015). Understanding lstm networks. <http://colah.github.io/> [posts/2015-08-Understanding-LSTMs/](http://colah.github.io/posts/2015-08-Understanding-LSTMs/). Accessed: 2017-10-08.

[Luong et al., 2014]Luong, M.-T., Sutskever, I., Le, Q. V., Vinyals, O., and Zaremba, W. (2014). Addressing the rare word problem in neural machine translation.*arXiv preprint arXiv:1410.8206*.

[Vaswani et al., 2017]Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez,

A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.

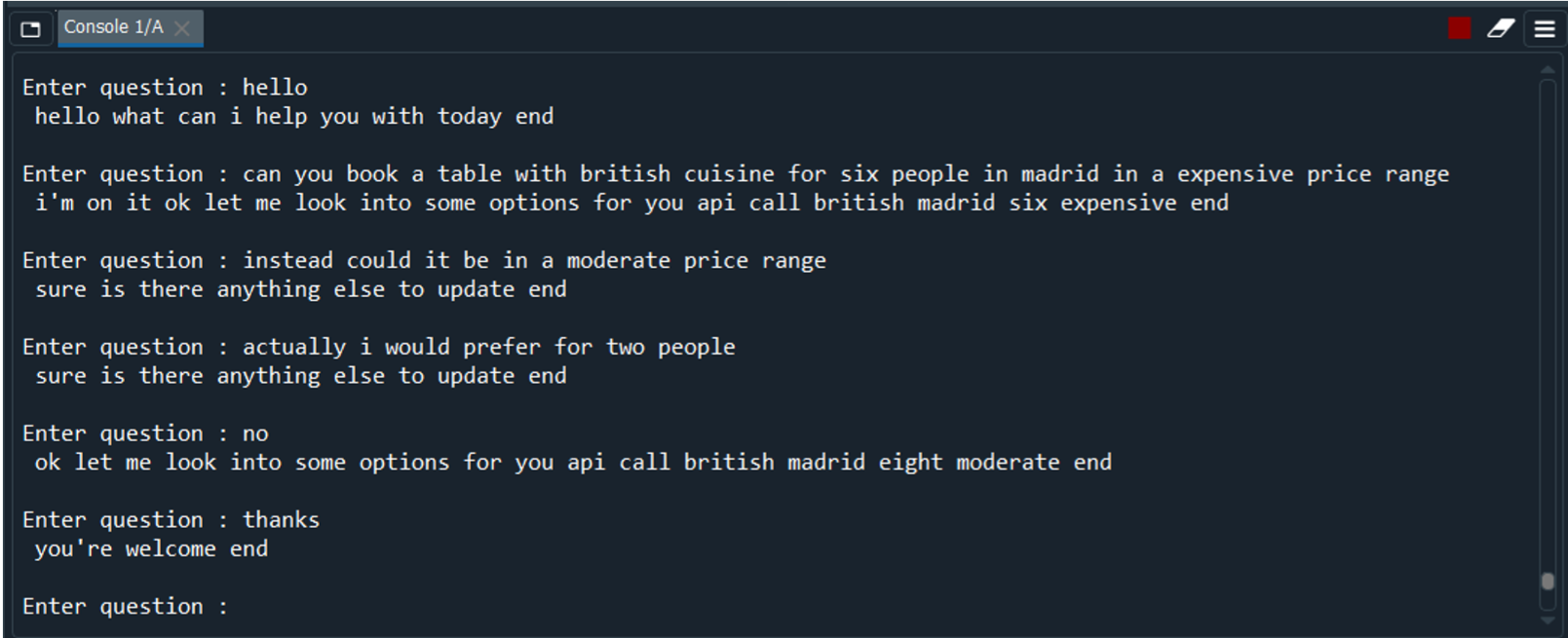
[Ghazvininejad et al., 2017] Ghazvininejad, M., Brockett, C., Chang, M.-W., Dolan, B., Gao, J., Yih, W.-t., and Galley, M. (2017). A knowledge-grounded neural conversation model. arXiv preprint arXiv:1702.01932.

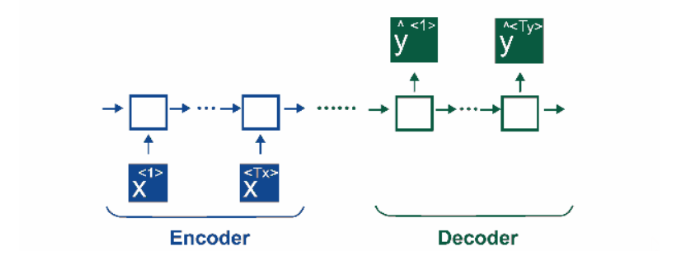
[Goyal et al., 2017] Goyal, K., Neubig, G., Dyer, C., and Berg-Kirkpatrick, T. (2017). A continuousrelaxationofbeamsearchforend-to-endtrainingofneuralsequencemodels. arXivpreprint arXiv:1708.00111.

1. **. Appendix**

**Wrong Output :**

This is output when the model was forgetting context. It happened when the model was overfitted and API calls were not handled properly.



**Encoder-Decoder Layers and input/outputs** : 

* The model will have Embedding, LSTM, and Dense layers. The basic configuration is as follows.

**Layers:**

* 2 Input Layers : One for encoder\_input\_data and another for decoder\_input\_data.
* Embedding layer: For converting token vectors to fix sized dense vectors. ( Note: Don't forget the mask\_zero=True argument here )
* LSTM layer: Provide access to Long-Short Term cells.

**Connections:**

* The encoder\_input\_data comes in the Embedding layer ( encoder\_embedding ).
* The output of the Embedding layer goes to the LSTM cell which produces 2 state vectors ( h and c which are encoder\_states )
* These states are set in the LSTM cell of the decoder.
* The decoder\_input\_data comes in through the Embedding layer.
* The Embeddings goes in LSTM cell ( which had the states ) to produce sequences.

**Inference model Details** :

* We create inference models that help in predicting answers.
* Encoder inference model: Takes the question as input and outputs LSTM states ( h and c ).
* Decoder inference model: Takes in 2 inputs, one are the LSTM states ( Output of encoder model ), second is the answer input sequences ( ones not having the <start> tag ). It will output the answers for the question which we fed to the encoder model and its state values.

**Conversation steps :**

* First, we take a question as input and predict the state values using enc\_model.
* We set the state values in the decoder's LSTM.
* Then, we generate a sequence that contains the <start> element.
* We input this sequence in the dec\_model.
* We replace the <start> element with the element which was predicted by the dec\_model and update the state values.
* We carry out the above steps iteratively till we hit the <end> tag or the maximum answer length.

**Additional Input Features:**

In addition to the raw dialog turns a plethora of other inputs can be integrated into the seq2seq model. Several attempts have been made since the birth of the encoder-decoder model to augment it with additional input features. The goal of these features is mainly to provide more information about the conversation or the context. Also, models infused with additional features can learn to differentiate between various styles of dialog

**Knowledge Bases and Copying:**

Knowledge bases (KB) are powerful tools that can be used to augment conversational models. Since knowledge bases usually entail some kind of domain speciﬁc information, these techniques are mainly used for task-oriented dialog systems.

**Another approach for the conversational system :**

**Hierarchical Models**

To better represent dialog history, a general hierarchical recurrent encoder-decoder (HRED) architecture was proposed in [Serban et al., 2016]. The model consists of three different RNNs, the encoder RNN, the context RNN, and the decoder RNN. First, k previous utterances of a conversation are encoded separately by the encoder RNN. This produces k separate context vectors by taking the last hidden state of the encoder RNN from each encoded utterance. Then these k hidden states are fed into the context RNN step by step, thus it has to be unrolled for k steps. Next, the last hidden state from the context is used to initialize the decoder RNN. The decoder RNN and the decoding process is very similar to the one found in a normal seq2seq model.