MINI PROJECT, ITE398

A NEURAL NETWORK BASED CHATBOT

March 20, 2017

Submitted by
Saurabh Mathur, 14BIT0180

Under the guidance of
Prof. Daphne Lopez

Contents

ABSTRACT

Building Conversational Agents, or Chatbots is one of the most exciting problems in Artificial Intelligence. Researchers have been trying to build and improve chatbots from as early as 1966. However, those chatbots were domain-specific and required a rule-base of hand-crafted rules. This often proves to be costly and limits access to such technology.

Machine Learning is a statistical approach at building AI Agents. The most common set of techniques is supervised learning. It allows us to build a system that can infer or, learn rules from a set of labelled examples. Recent advances in Artificial Neural Network architectures have made it possible to model sequences of variable length. Researchers have applied such architectures to the task of Language Translation with great success. Modern Graphical Processing Units (GPUs) have been found to be well suited to performing matrix operations at a large scale efficiently. This facilitates training of Neural Networks on large sets of data. Neural Network based agents can be trained end-to-end. Thus, they don't depend on hand-crafted rules.

This project is an attempt to build a Neural Conversational Model, with a sequence to sequence Neural Network architecture at its core.

Literature Survey

Building chatbots and conversational agents has been pursued by many researchers over the last decades.

One of the earliest attempts at solving the problem was a pattern matching based system called ELIZA. It was built at the MIT Artificial Intelligence Laboratory by Joseph Weizenbaum in 1966. Since then, many similar systems have been proposed. However, most of these systems require a rather complicated processing pipeline of many stages. (Lester et al., 2004; Will, 2007; Jurafsky & Martin, 2009). This project is different from conventional systems as uses an end-to-end approach to the problem which lacks domain knowledge.

This project is mainly inspired by Oriol Vinyals and Quoc V. Le's work titled A Neural Conversational Model. They use a sequence to sequence architecture to model conversations. This architecture was proposed by Sutskever et al. in 2014. The architecture, originally developed for the task of Statistical Neural Machine Translation, has been used for various tasks such as parsing and image caption generation.

The basic building block of the sequence to sequence architecture is the Recurrent Neural Network (RNN). It is well know that simple RNNs suffer from vanishing gradients making them hard to train. So, most implementations of the sequence to sequence architecture use Long Short Term Memory (LSTM) RNNs, proposed by Hochreiter & Schmidhuber in 1997. The LSTM avoids direct feedback connection of the simple RNN by the use of a gating mechanism.

Iulian Vlad Serban et al. surveyed available datasets for building Data-Driven Dialog Systems. The Cornell Movie-Dialogue Corpus has over 200,000 short conversations extracted from movie dialogs.

Dzmitry Bahdanau et al. proposed a modification to the sequence to sequence architecture such that the decoder has access to all states of the encoder through an attention mechanism.

HARDWARE AND SOFTWARE REQUIREMENTS

Training phase

Hardware requirements

- Amazon AWS g2.2xlarge
  - 60 GB SSD
  - 15 GB RAM
  - 8 CPU Cores
  - 1,536 CPU Cores with 4GB Video Memory

Software requirements

- Ubuntu Linux 16.04
- Python 2.7.12
- Tensorflow 0.12.0
- Cuda Toolkit 8.0
- cuDNN v5.1

Deployment phase

Hardware requirements

- 512 MB RAM
- 1 CPU Core

Software requirements

- Python 2.7.12
- Tensorflow 0.12.0

Software Architecture

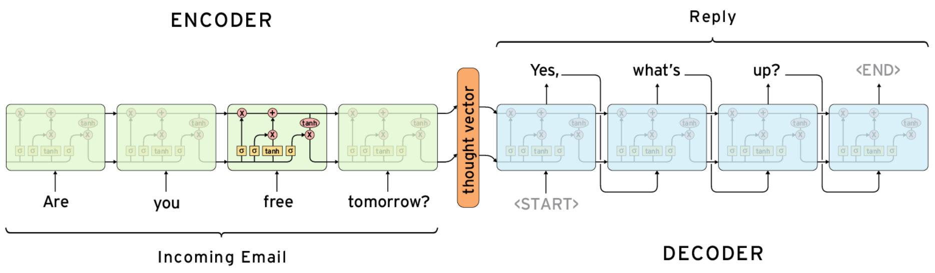


Figure 1: The Sequence to Sequence model Architecture

The project consists of the following components -

- Data Preprocessor - data.
- Chatbot - model.
  - Model: The sequence to sequence model.
  - Sampler: for generating responses
- Utilities - util.
- Additional Scripts - lib.
- Web based chat interface - server.
  - Browser based client application.
  - REST API Server.

Data Preprocessor

It uses Natural Language Processing to perform basic data cleaning. It consists of the following scripts -

- pull.py : Downloads and extracts the OpenSubtitles dataset.
- make_pairs.py : Cleans the subtitles files and converts them into pairs of statement-response. Performs data augmentation to increase size of dataset.
- filter.py : Removes rare words and pairs having more than 20 % rare words.

Chatbot

This is module is the core of the system. The model learns a probability distribution from a large set of examples and predicts probabilities for words in response.

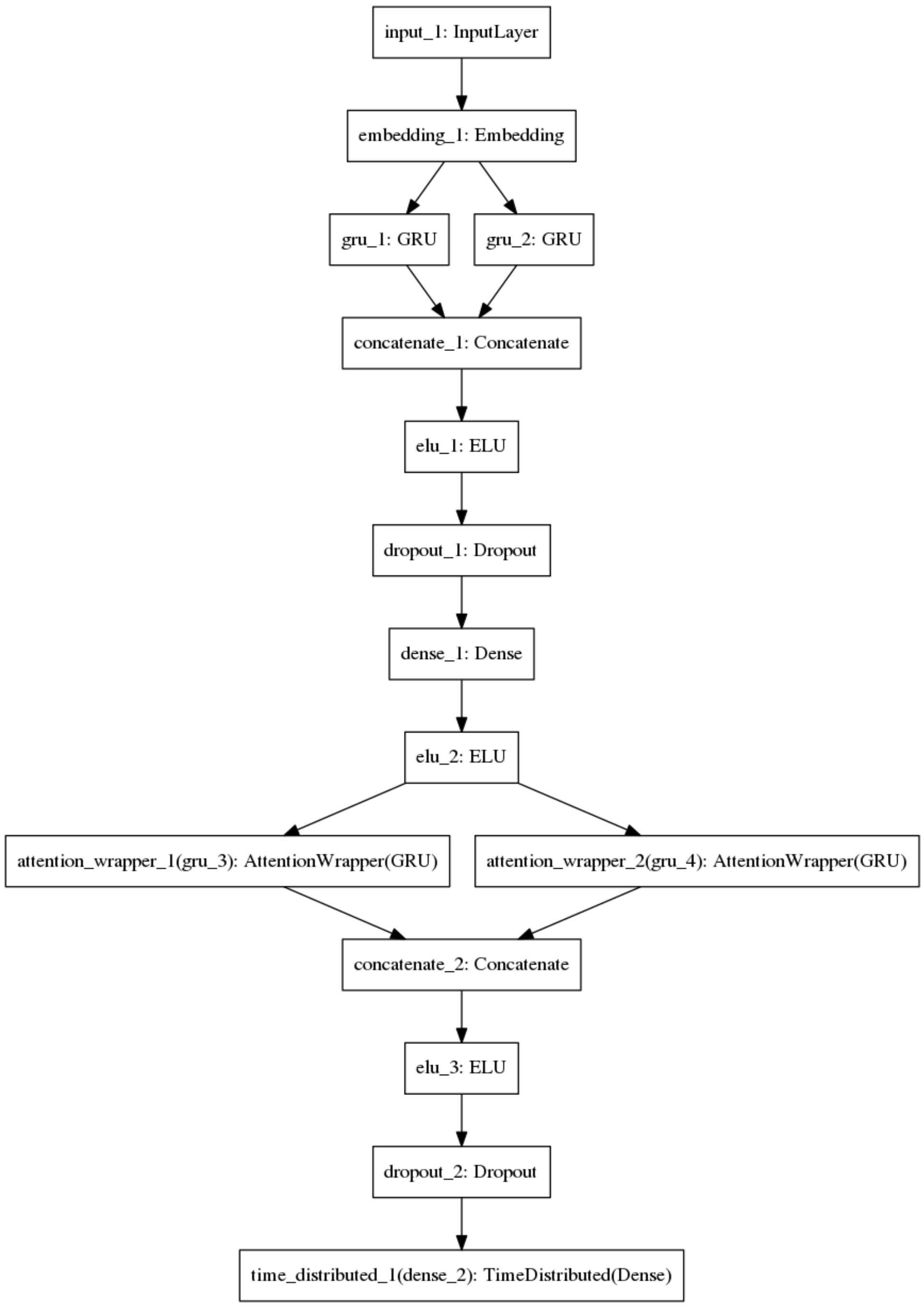

input_1: InputLayer
embedding_1: Embedding
gru_1: GRU
gru_2: GRU
concatenate_1: Concatenate
elu_1: ELU
dropout_1: Dropout
dense_1: Dense
elu_2: ELU
attention_wrapper_1(gru_3): AttentionWrapper(GRU)
attention_wrapper_2(gru_4): AttentionWrapper(GRU)
concatenate_2: Concatenate
elu_3: ELU
dropout_2: Dropout
time_distributed_1(dense_2): TimeDistributed(Dense)

Figure 2: The Bidirectional Sequence to Sequence model Architecture with attention

- models : Defines two sequence-to-sequence model variants - with and without an attention mechanism.
- sequence_blocks : Defines an Encoder module and two Decoder modules - with and without attention.
- sample : Uses the predicted word probabilities to draw a sample, thus forming the response.

Utilities

It contains various functions used throughout the project.

- download : Downloads a file, showing a byte-by-byte progress bar.
- normalize_unicode : Converts Unicode text to ASCII by removing non-printable characters.
- augment : Generates possible sentence statement-response pair using a given multi-sentence pair.

Additional Scripts

wikifil.pl: A perl script used in initial preprocessing of the dataset. Normalizes words, spells out digits and removes extraneous characters.

Chat Interface

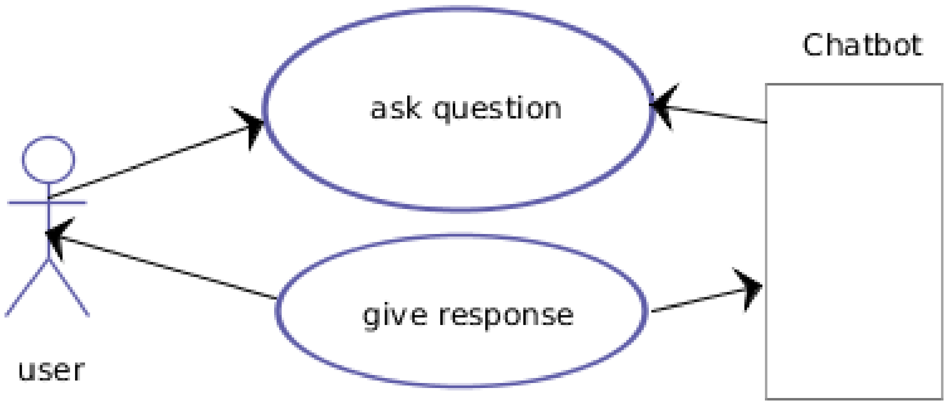
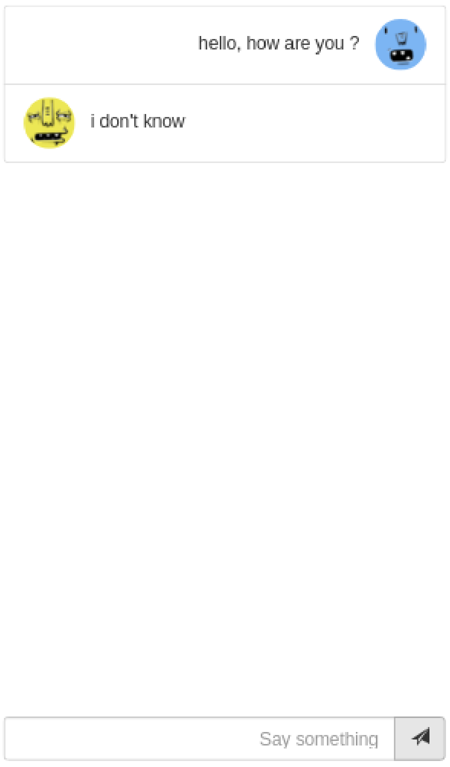


Figure 3: Use case diagram for the chat interface

The chat interface consists of two subcomponents - A REST API Server that calls the chatbot model and a JavaScript based client application that makes AJAX requests to the Server.


hello, how are you ?

i don't know

Figure 4: The client side chat interface

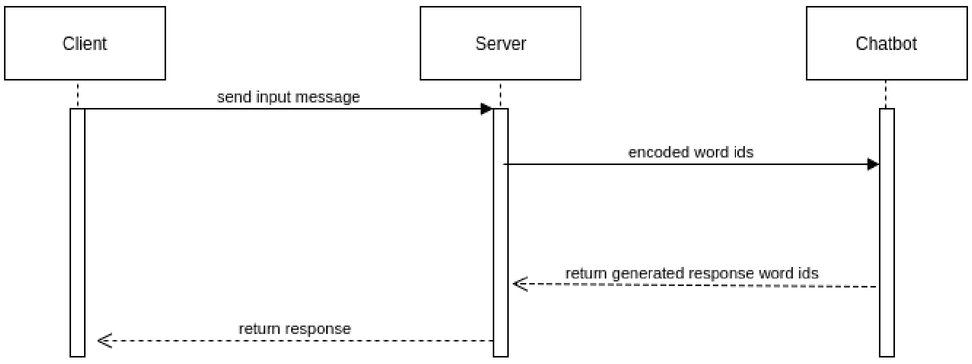


Figure 5: Sequence diagram for the chat interface

- server.py : Serves the index.html file. It calls the chatbot with the user's message and sends the generated response to the user.

- index.html : A client side chat interface built with HTML, CSS and JavaScript using jQuery and Bootstrap. To make it easier for the user, pressing enter key sends the message.

  The message is sent to the server by an AJAX POST request. The generated response is displayed to the user as the chatbot's response.

Code Snippets

```
 ...
   s/<.*>//;     # remove xml tags
   s/&amp;/&/g; # decode URL encoded chars
   s/&lt;/</g;
   s/&gt;/>/g;
   s/<ref[^<]*<\/ref>//g; # remove references <ref...> ... </ref>
   s/<[^>]*>//g; # remove xhtml tags
   s/\[http:[^] ]*/[/g; # remove normal url, preserve visible text
   s/\[//g;                  # remove [ and ]
   s/\]//g;
   s/&[^;]*;/ /g;            # remove URL encoded chars


   $_=" $_ ";

    # convert to lowercase letters and spaces, spell digits
   tr/A-Z/a-z/;
   s/0/ zero /g;
   s/1/ one /g;
   s/2/ two /g;
   s/3/ three /g;
   s/4/ four /g;
   s/5/ five /g;
   s/6/ six /g;
   s/7/ seven /g;
   s/8/ eight /g;
   s/9/ nine /g;

   tr/a-z.!?/ /cs;
   tr/.!? //s;
   ...
```

Listing 1: Pre-processing using perl

```
...
i = Input(shape=(sequence_length,))
x = Embedding(vocabulary_size, hidden_size, mask_zero=True)(i)

# Encoder Block
x = Encoder(hidden_size, return_sequences=True, bidirectional=True)(x)
x = Dropout(.5)(x)
```

```
x = Dense(hidden_size, activation='linear')(x)
x = ELU()(x)
attention = Maxpool(x)

# Decoder Block
x = AttentionDecoder(hidden_size, return_sequences=True, bidirectional=True)(x,
    attention)
x = Dropout(.5)(x)
x = TimeDistributed(Dense(vocabulary_size, activation='softmax'))(x)
model = Model(inputs=i, outputs=x)
...
```

Listing 2: Attention Model

```
$.ajax('./respond', {
    type: 'POST',
    data: { message: user_input },
    dataType: 'json',
    success: function(result) {
        var text = bot_message.join('\n').replace('TEXT', result.response);
        $('#messages').append(text);
        $('#messages').scrollTop($('ul li').last().position().top + $('ul li').last()
    .height());
    }
});
```

Listing 3: Fetching response using AJAX

REFERENCES

Weizenbaum, J. ELIZA - A Computer Program For the Study of Natural Language Communication Between Man And Machine. Communications of the ACM, Vol 9, 1966.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In NIPS, 2014.

Vinyals, O., Le, Q. V. A Neural Conversational Model. In arXiv:1506.05869v3, 2015.

Lifeng Shang, Zhengdong Lu, and Hang Li. Neural Responding Machine for Short Text Conversation. In ACL-IJCNLP 2015.

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Meg Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. A neural network approach to context-sensitive generation of conversational responses. In NAACL-HLT 2015.

Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, Joelle Pineau Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models. In arxiv 1507.04808.

Jurafsky, D. and Martin, J. Speech and language processing. Pearson International, 2009.

Lester, J., Branting, K., and Mott, B. Conversational agents. In Handbook of Internet Computing. Chapman & Hall, 2004.

Will, T. Creating a Dynamic Speech Dialogue. VDM Verlag Dr, 2007.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. Neural Computation, 1997.

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua, Neural machine translation by jointly learning to align and translate. In arXiv:1409.0473

Iulian Vlad Serban, Ryan Lowe, Laurent Charlin, Joelle Pineau, A Survey of Available Corpora for Building Data-Driven Dialogue Systems. In arxiv 1512.05742