Practical seq2seq

Revisiting sequence to sequence learning, with focus on implementation details

Posted on December 31, 2016

In my last article, I talked a bit about the theoretical aspect of the famous Sequence to Sequence Model. I have shared the code for my implementation of seq2seq - easy_seq2seq. I have adopted most of the code from en-fr translation example provided by Google. Hence, most parts of the code, that dealt with data preprocessing, model evaluation were black boxes to me and to the readers. To make matters worse, the model trained on Cornell Movie Dialog corpus performed poorly. A lot of people complained about this. After training the model for days, most of the responses were gibberish. I apologize for wasting your time.

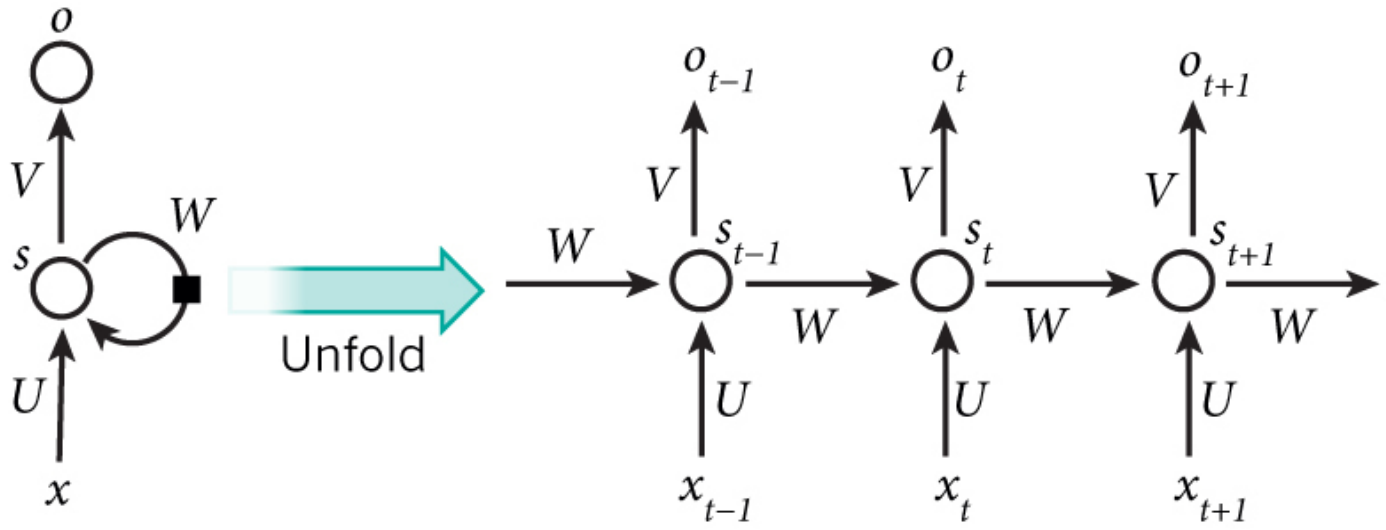
The objective of this article is two-fold; to provide the readers with a pre-trained model that actually works and to describe how to build and train such a model from (almost) scratch.

Before doing that, let us recap what we discussed in the previous article.

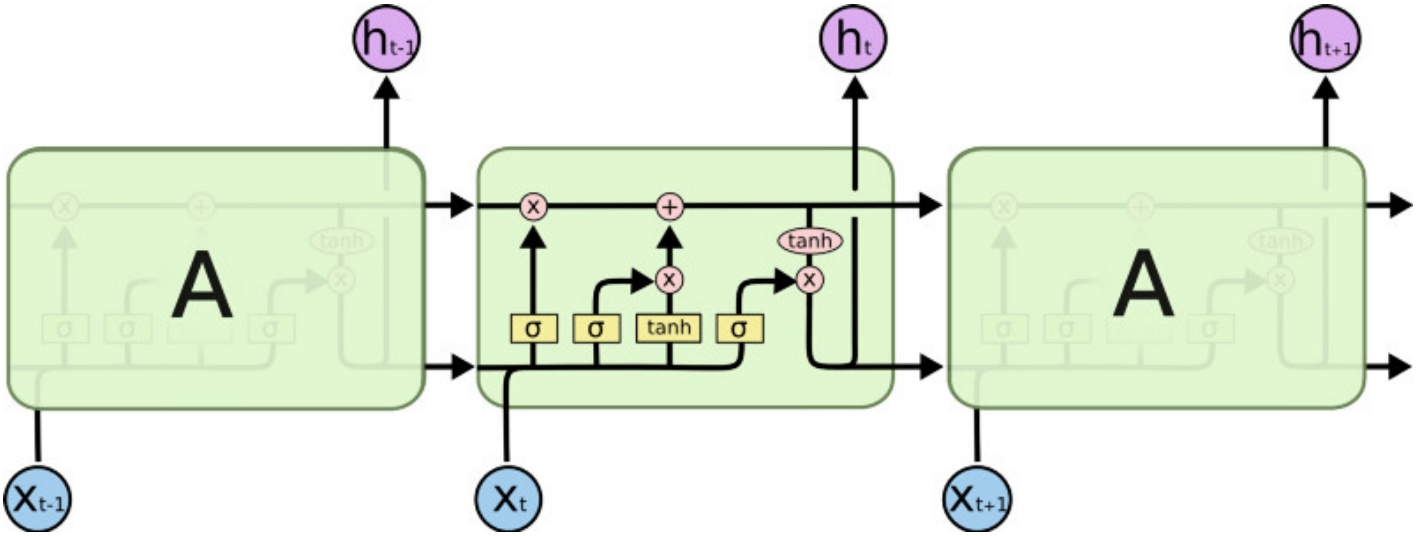
Recurrent Neural Networks

Recurrent Neural Networks or simple RNNs, are a special kind of neural networks that are capable of dealing with sequential data, like videos(sequence of frames) and more commonly, text sequences or basically any sequence of symbols. The beauty of it is, the network doesn't need to know what the symbols mean. It will infer the meaning of symbols, by looking at the structure of the text and relative positions of symbols. There are some amazing articles on RNNs and what they are capable of. To put it simply, an RNN, unlike an MLP or CNN, has an internal state. Think of this state as memory of the network. As the RNN devours a sequence (sentence), word by word, the essential information about the sentence is maintained in this memory

unit (internal state), which is periodically updated in each timestep. The essence of a sentence of 7 words will be captured by an RNN, in 7 timesteps. This is just a "story" about RNN, intended to provide a high level understanding of RNN. To understand the actual mechanism of RNN, read Denny Britz's series of articles on RNN - 1, 2 ,3, 4 and then move on to Karpathy's The Unreasonable Effectiveness of RNNs.

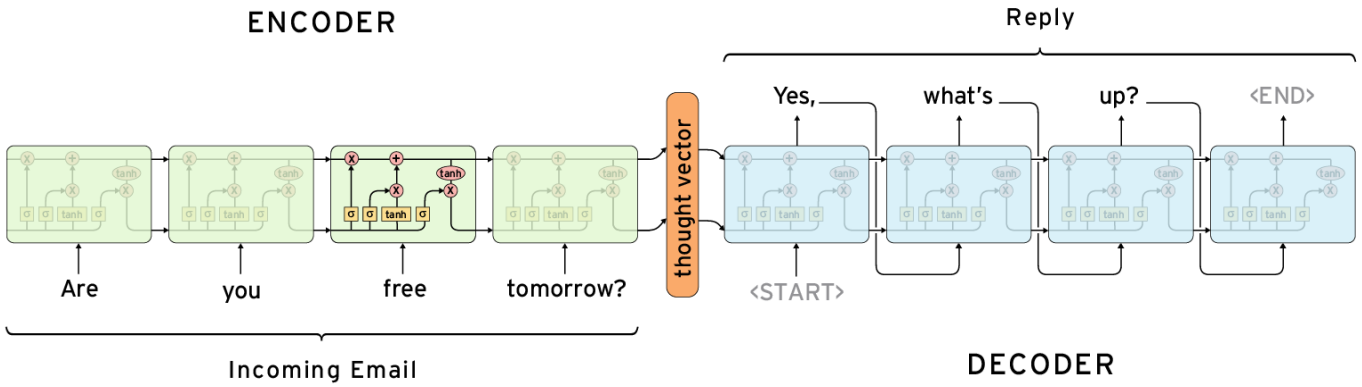


The naive version of RNN, is typically called a *Vanilla* RNN, which is pretty pathetic in remembering long sequences. There are more complex versions of RNN, like LSTM (Long Short Term Memory) and GRU (Gated Recurrent Units) RNNs. The only difference between a Vanilla RNN and LSTM/GRU networks, is the architecture of the memory unit. An LSTM cell consists of multiple gates, for remembering useful information, forgetting unnecessary information and carefully exposing information at each time step. Christopher Olah does an amazing job explaining LSTM in this article. Check out Denny's tutorial on GRU.



Sequence to Sequence Learning

The Sequence to Sequence model (seq2seq) consists of two RNNs - an encoder and a decoder. The encoder reads the input sequence, word by word and emits a context (a function of final hidden state of encoder), which would ideally capture the essence (semantic summary) of the input sequence. Based on this context, the decoder generates the output sequence, one word at a time while looking at the context and the previous word during each timestep. This is a ridiculous oversimplification, but it gives you an idea of what happens in seq2seq.



The context can be provided as the initial state of the decoder RNN or it can be connected to the hidden units at each time step. Now our objective is to jointly maximize the log probability of the output sequence conditioned on the input sequence.

Read these papers for a deeper understanding of seq2seq - Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, Sequence to Sequence Learning with Neural Networks, Neural Machine Translation by Jointly Learning to Align and Translate, A Neural Conversational Model.

A Working Model

As promised, here is a working model of a twitter bot based on seq2seq model. It is trained on twitter chat log borrowed from Marsan-Ma. Before going into how to bootstrap and run the code, let us look at some of the decent responses spit out by the bot.

Query	Reply
donald trump won last nights presidential debate according to snap online polls	thought he was a joke
trump campaign please never delete this	he was a joke
s new scoop is unk the castro connection how trumps company violated the us embargo against cuba	trump is a disgrace to the debate and the media is not a liar
who won the first presidential debate	trump will be a better time for a man
just wanna live in unk everything is 10x better there	i was just on the same side and i was like it was a good time
the lil girl i forgot her name scares the shit out of me n probably always will lmaooo	she was so cute and she was so cute and she was a bitch
question of the day who do you think won last nights debate weigh in using	hillary is a trump supporter and a woman who has a better job than a woman
trump is a total embarrassment hes like a unk child who happens to be a professional liar and con man	trump is a lot of money to get a great country
chill the fuck out	thats what i said

Query	Reply
so fun to see what the special effects team created for	wow that is the best thing ive ever seen
she was fine too unk voice	she is a good guy
the important ones are still here	i dont think so
been drinking pumpkin spice protein shake every morning for a week and now i literally cannot even	me too i just got it
lmao i just lost my job and i cant even leave early to be in the unk video today	literally the same thing
hey happy birthday have a nice day	thank you

For some reason people were really into tweeting about Trump.

Pretty good responses right? I have let the model overfit on the training data, just a little bit. The *unk* symbols in the conversations, refer to the words that aren't frequent enough (rare) to be put in the vocabulary. Now, how to reproduce these results in your pc?

```
# clone the repository

git clone https://github.com/suriyadeepan/practical_seq2seq
# pull the pretrained model

cd practical_seq2seq/ckpt/twitter/
./pull # extract the archive

# this will take some time ~830 MB

cd ../../
# then you need the datasets to test the model

cd datasets/twitter/
./pull # this wont take long

# extract the archive
```

```
cd ../../
# instead you could just test with your own queries

#  in that case you could just skip the step above

# now.. open up jupyter notebook and run "03-Twitter-chatbot.ipynb"

#  the (pieces of) code is fairly intuitive

#   figure out how to use it
```

Are you happy with the results? Is the pretrained model working well for you? If not, raise an issue in the github repo.

Understanding the Code

Let us move on to more interesting things. What are the steps involved in building such a bot? Any project in Machine Learning follows this pattern. Study and analyse the data, preprocess the data to make it compatible with the model, split it into train, valid, and test sets, create a model, feed the training data, let it overfit, reduce the depth/width of the model and train again(which I usually skip), evaluate the model periodically to look for overfitting/underfitting.

Data Preprocessing

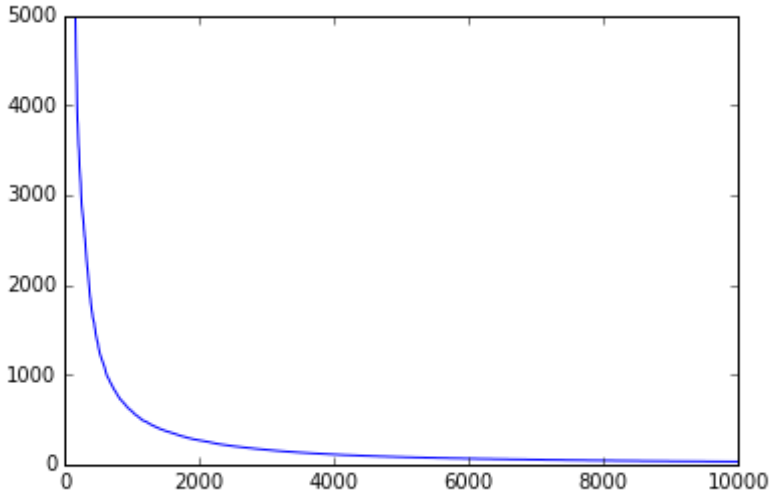
I keep all my datasets and preprocessing scripts in this repository - suriyadeepan/datasets. Twitter dataset and scripts can be found here. Follow the instructions in README, to process raw data or to download processed data. I am presenting the gist of it in the snippet below. You might want to read data.py to get a clearer picture of preprocessing.

```
# read from file; return a list of sentences
def read_lines(filename):
    return open(filename).read().split('\n')[:-1]

# remove any character except alphanumerics
def filter_line(line, whitelist):
    return ''.join([ ch for ch in line if ch in whitelist ])
```

```
# create loopup tables [word -> index ] and [index -> word]
#  frequency distribution of words in corpus
def index_(tokenized_sentences, vocab_size):
    # get frequency distribution
    freq_dist = nltk.FreqDist(itertools.chain(*tokenized_sentences))
    # get vocabulary of 'vocab_size' most used words
    vocab = freq_dist.most_common(vocab_size)
    # index2word
    index2word = ['_'] + [UNK] + [ x[0] for x in vocab ]
    # word2index
    word2index = dict([(w,i) for i,w in enumerate(index2word)] )
    return index2word, word2index, freq_dist

# other functions
# def filter_data(sequences)
#  - filter dataset based on limits on length of sequences
# def zero_pad(qtokenized, atokenized, word2index)
#  - creates zero padded ndarrays
# steps in involved in processing
# read from file
lines = read_lines(filename=FILENAME)
# change to lower case
lines = [ line.lower() for line in lines ]
# filter out unnecessary characters
lines = [ filter_line(line, EN_WHITELIST) for line in lines ]
# filter out too long or too short sequences
qlines, alines = filter_data(lines)
# convert list of [lines of text] into list of [list of words ]
qtokenized = [ wordlist.split(' ') for wordlist in qlines ]
atokenized = [ wordlist.split(' ') for wordlist in alines ]
# indexing -> idx2w, w2idx : en/ta
idx2w, w2idx, freq_dist = index_( qtokenized + atokenized, vocab_size=
VOCAB_SIZE)
# zero padding
idx_q, idx_a = zero_pad(qtokenized, atokenized, w2idx)
# let us now save the necessary dictionaries
metadata = {
        'w2idx' : w2idx,
        'idx2w' : idx2w,
        'limit' : limit,
        'freq_dist' : freq_dist
            }
# save metadata and data(idx_q, idx_a)
#  we haven't split the data into train/valid/test
#   that comes later
```

How to choose the vocabulary?

It makes sense to have a vocabulary of the most frequent words. But how to choose the size of vocabulary? I have plotted the frequency vs words (most frequent to least frequent). I have chosen a vocabulary that covers most the area under the curve. Any word not in the vocabulary will be replaced with the UNK token.



The number of UNK tokens in the dataset make a big difference. If there are too many unknown tokens, the model learns to output UNK tokens more than the words in our limited vocabulary. I have reduced the overall percentage of unknown tokens in the dataset to 3%, by increasing the vocabulary size and by removing sentences with too many unknown words (rare words) from the dataset. It is advisable to keep it less than 5%.

```
# count of unknowns
unk_count = (idx_q == 1).sum() + (idx_a == 1).sum()
# count of words
word_count = (idx_q > 1).sum() + (idx_a > 1).sum()
# % unknown
print('% unknown : {}'.format(100 * (unk_count/word_count)))
```

feed_dict helpers (data_utils.py)

We have processed the data, changed it from raw lines of text stored in a file, to zero-padded numpy arrays of indices, along with necessary metadata (word2index and index2word dictionaries). Now it is time to write a few helper functions that

will gather random examples from the dataset, in batches, to feed to the model for training and evaluation. data_utils.py has a couple of functions for that purpose.

```
#
# split the dataset into train/valid/test sets
#  def split_dataset(x, y, ratio = [0.7, 0.15, 0.15] )
# generate batches, by random sampling a bunch of items
#  yield (x_gen, y_gen)
def rand_batch_gen(x, y, batch_size):
    while True:
        sample_idx = sample(list(np.arange(len(x))), batch_size)
        yield x[sample_idx].T, y[sample_idx].T
# a generic decode function
#  inputs : sequence, lookup
def decode(sequence, lookup, separator=''):
# 0 used for padding, is ignored
    return separator.join([ lookup[element]
        for element in sequence if element ])
```

Build a wrapper for Seq2Seq

I created a class, Seq2Seq that provides high level abstractions for building the graph, training, evaluation, saving and restoring trained model. The constructor takes these parameters as input:

- xseq_len, yseq_len
- xvocab_size, yvocab_size
- emb_dim
- num_layers
- ckpt_path
- lr=0.0001
- epochs=100000

```
# create an instance
model = seq2seq_wrapper.Seq2Seq(xseq_len=xseq_len,
                                yseq_len=yseq_len,
                                xvocab_size=xvocab_size,
                                yvocab_size=yvocab_size,
                                ckpt_path='ckpt/twitter/',
                                emb_dim=emb_dim,
```

```
                                    num_layers=3
                                    )
```

To build the graph, we create a bunch of placeholders for feeding input sequences, labels and decoder inputs. The labels correspond to the real output sequence. The decoder inputs start with a special 'GO' symbol, which in this case is just 0. For each sequence, we create a list of placeholders of datatype *tf.int64* (indices), of known length.

```
#  encoder inputs : list of indices of length xseq_len
self.enc_ip = [ tf.placeholder(shape=[None,],
                dtype=tf.int64,
                name='ei_{}'.format(t)) for t in range(xseq_len) ]
#  labels that represent the real outputs
self.labels = [ tf.placeholder(shape=[None,],
                dtype=tf.int64,
                name='ei_{}'.format(t)) for t in range(yseq_len) ]
#  decoder inputs : 'GO' + [ y1, y2, ... y_t-1 ]
self.dec_ip = [ tf.zeros_like(self.enc_ip[0], dtype=tf.int64, name='G
O') ]
    + self.labels[:-1]
```

Next, we create the LSTM cell, the most essential part of the graph. We define a placeholder for *keep_prob*, which will be used to control the dropout. Dropout is a simple technique to prevent overfitting. It essentially just drops some of the unit activations in a layer, by making them zero. Then, we define a basic LSTM cell and then, wrap it with a DropoutWrapper. The LSTM cells, we just created are stacked together to form a stacked LSTM, using *MultiRNNCell*.

```
# Basic LSTM cell wrapped in Dropout Wrapper
self.keep_prob = tf.placeholder(tf.float32)
# define the basic cell
basic_cell = tf.nn.rnn_cell.DropoutWrapper(
        tf.nn.rnn_cell.BasicLSTMCell(emb_dim, state_is_tuple=True),
        output_keep_prob=self.keep_prob)
# stack cells together : n layered model
stacked_lstm = tf.nn.rnn_cell.MultiRNNCell([basic_cell]*num_layers, st
ate_is_tuple=True)
```

Now we use a high level function - *embedding_rnn_seq2seq* provided by tensorflow's seq2seq module, to create a seq2seq model, which does word embedding internally. A copy of the same model is created for testing, which uses the same parameters but has *feed_previous* switch enabled. This causes the decoder of the model to use the output of previous timestep as input to the current timestep, while during training, the input to a timestep (in the decoder) is taken from the labels sequence (the real output sequence).

```
self.decode_outputs, self.decode_states
        = tf.nn.seq2seq.embedding_rnn_seq2seq(
        self.enc_ip,self.dec_ip, stacked_lstm,
        xvocab_size, yvocab_size, emb_dim)
```

We use another high level function *sequence_loss*, to get the expression for loss. Then, we build a *train* operation that minimizes the loss.

```
loss_weights = [ tf.ones_like(label, dtype=tf.float32)
        for label in self.labels ]
self.loss = tf.nn.seq2seq.sequence_loss(self.decode_outputs,
        self.labels, loss_weights, yvocab_size)
self.train_op = tf.train.AdamOptimizer(learning_rate=lr).minimize(self
.loss)
```

Training is fairly simple. We create a method *train*, that runs the train op, for a given number of epochs. Evaluation is done periodically on the validation set. The model is saved after evaluation. A dropout of 0.5 is used during training. The dropout is disabled during evaluation. When *restore_last_session* is called, the last saved checkpoint is restored and returned. The *predict* function does a forward step and returns the indices of most probable words emitted by the model. Read seq2seq_wrapper.py to get the big picture.

Training

Let us train the model using the processed dataset. The *load_data* function returns the dataset (x,y) and the metadata (index2word, word2index). We split the dataset into train, validation and test sets. We set the parameters of the model, like

sequence length, vocabulary size and embedding dimensions. We then, create an instance of the model and train it, by passing data iterators to *model.train* method. If the training gets interrupted deliberately or otherwise, we can continue training from the last saved checkpoint. This is done by getting the session from *model.restore_last_session* and then passing the session to *model.train* method. The code for training is available here.

```
import tensorflow as tf
import numpy as np
import seq2seq_wrapper
from datasets.twitter import data
import data_utils
# load data from pickle and npy files
metadata, idx_q, idx_a = data.load_data(PATH='datasets/twitter/')
(trainX, trainY), (testX, testY), (validX, validY) = data_utils.split_
dataset(idx_q, idx_a)
# parameters
xseq_len = trainX.shape[-1]
yseq_len = trainY.shape[-1]
batch_size = 32
xvocab_size = len(metadata['idx2w'])
yvocab_size = xvocab_size
emb_dim = 1024
model = seq2seq_wrapper.Seq2Seq(xseq_len=xseq_len,
                                yseq_len=yseq_len,
                                xvocab_size=xvocab_size,
                                yvocab_size=yvocab_size,
                                ckpt_path='ckpt/twitter/',
                                emb_dim=emb_dim,
                                num_layers=3
                                )
val_batch_gen = data_utils.rand_batch_gen(validX, validY, 32)
train_batch_gen = data_utils.rand_batch_gen(trainX, trainY, batch_size
)
#sess = model.restore_last_session()
sess = model.train(train_batch_gen, val_batch_gen)
# to continue training after interrupting,
#  run model.train with "sess=sess"
```

More Datasets

I have also trained the seq2seq model using other datasets, like CMU Pronouncing Dictionary, Cornell Movie Dialog Corpus, and Tamil to English parallel corpus. As I am writing this article, my GTX960 is training the seq2seq model on Open Subtitles dataset. I will probably add the results of it tomorrow.

CMU Pronouncing Dictionary

pronunciation	real spelling	model spelling
MAH0KLAE1NAH0HHAE0N	mcclanahan	mcclanahan
BAH1NDAH0SWEH2R	bundeswehr	bundessware
IH0NRIY1KWEH0Z	enriquez	enriques
LOW0NEH1ROW0	lonero	lonero
BEH1NAH0VIY0DEH0S	benevides	benevedese
DIH0STRAH1KTIH0V	destructive	distructive
CHAO1NCHUW0LIY0	cianciulli	chonchully
PRAA2KLAH0MEY1SHAH0NZ	proclamations	proclamations
FRIY1THIH1NGKER0	freethinker	freethinker
KAH0NTEH1MPTAH0BAH0L	contemptible	contemptable
PRIY1SKUW2LER0	preschooler	prescouler
TOW2MIY0IY1CHIY0	tomiichi	tomiacio
PAA1STAH0L	postle	postel
GUW2SIY0AO1RAH0	gusciora	gusiora
STAE1GNEY2TIH0NG	stagnating	stagnating
TER1MIH0NAY2	termini	turmini

Cornell Movie Dialog Corpus

Query	Reply
when are you going to let the police know	you dont know what youre talking about
you can do it	yeah i think so
wait are you saying that i dont appreciate	i know you know what i mean
youre terrible	yeah but im not
nah i dont think so	why not
hes pissing in our faces again and were just taking it	i cant
im sorry i dont know what else to say except im sorry	why would you like to be
nobodys gonna get you now get inside	i cant go back
shall i leave	of course

Query	Reply
well i really think hes got a chance	i know
youd better be quiet sandy	shut up
jesus christ you scared the shit out of me	whats going on
well im sorry im really sorry ellie	its okay
my lady this play will end badly i will tell	lets get out of here
im going to miss you	no youre not
what do you mean	i dont know i dont know what i mean
my god these people are insane	we dont know what they are
this isnt a date	no what is it
you ought to go home and take care of that	i cant do that
is something wrong	no no no

Reference

- Chatbots with Seq2Seq
- easy_seq2seq
- en-fr translation in Tensorflow
- Introduction to RNNs
- The Unreasonable Effectiveness of RNNs
- Understanding LSTM networks
- Implementing GRU
- practical_seq2seq
- Twitter Chat Log
- suriyadeepan/datasets
- Twitter Dataset and Scripts

I think I have discussed everything I wanted to discuss. If you are confused about anything we discussed, you are welcome to leave a comment below or raise an isssue in github. I will try my best to respond. I wanted to complete this article before the New Year. It is 11:35 PM here in Puducherry, India.

இனிய புத்தாண்டு நல்வாழ்த்துக்கள்!!

- tensorflow
- machine learning
- seq2seq
- NLP
- chatbot

← PREVIOUS POST

NEXT POST →