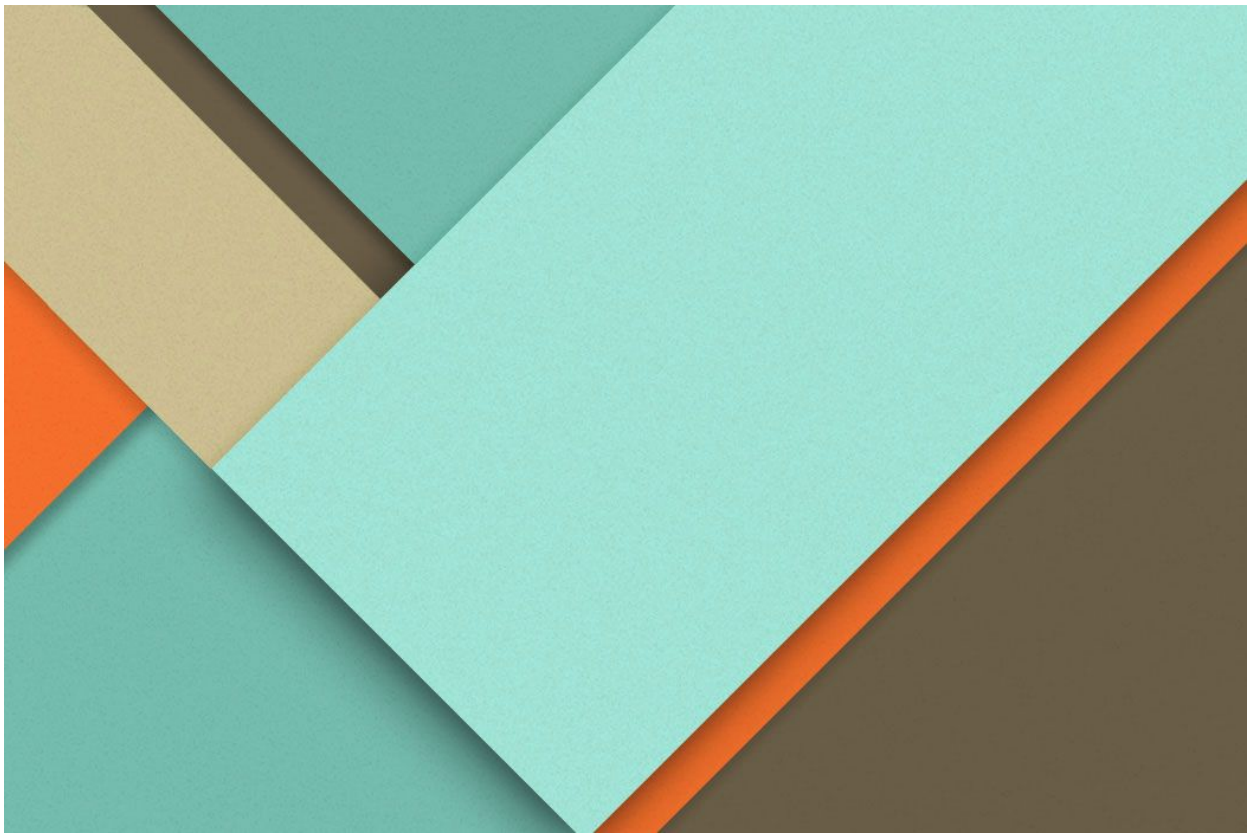


Artificial Intelligence Project Report

30 January 2017

Table of Contents

1. Introduction

Project UNK is a CLI-based chatbot that used the advanced technology of Recurrent Neural Network to interact with user. Project UNK can learn from the available dataset, and it gets smarter as the dataset increases or the amount of learning time increases. This project is inspired by Siraj Raval explanation on “How to Make an Amazing Tensorflow Chatbot Easily” at Youtube. His github repository for this project are located here : https://github.com/llSourcell/tensorflow_chatbot.

1.1 Problem Statement

We would like to tackle the problem of communication between people through the help of a chatbot. There are some children in this world where their parents are too busy and do not have enough time to spend time with their children, this may result in them having trouble conversing with others as they had a lack of communication with others. This is where UNK would come in, she will be their chatting partner, teaching how to converse with others.

2. Technical Specifications

2.1 Solution Features

Project UNK uses two Long Short-Term Memory (LSTM) networks, which is a special type of Recurrent Neural Network that has capabilities to store long term dependencies. This special characteristic of LSTM allows the chatbot to remember previous conversations, hence being able to generate a more natural reply during conversations. One LSTM will act as the encoder that reads input sentences, while the other will act as a decoder that displays the output sentences. This architecture is called Sequence-to-sequence model, and it is commonly implemented in various chatbots.

2.2 Solution Design Architecture

To build the entire Sequence-to-Sequence model, we used Tensorflow. Tensorflow is an open-source machine learning library developed by Google. In Tensorflow, the Sequence-to-Sequence model is available as a library that is ready to use. To implement it, we need to define all operations and "Tensors" (Tensor is a multidimensional array, in Tensorflow terminologies), then execute it by running the current session's object.

Once we built up the basis of UNK, we must train her with different data sets as weights to develop its knowledge in regards to chatting with real people. The data sets that are being used are Cornell Movie-Dialogues Corpus, which is a collection of movie dialogs from 617 different movies, compiled by Cornell University.

We are training the chatbot with a 3 layer LSTM with a layer of 256 each in addition to a vocabulary size of 20,000. Its learning rate is set to 0.5 out of 1. This method allows UNK to analyze the data at a slower pace but still efficient. The training data size is set to zero, therefore having no limit to her learning. There is also a checkpoint so that the model parameters are saved and implemented every 500 training sessions, we are able to check on the progress of UNK's development at every new implementation with a test conversation.

To enhance the learning performance of UNK, we are using Tensorflow with GPU support. For this final project, the Computer Science Faculty is kind enough to let us utilize their PC that is equipped with an Nvidia Titan X. To utilize the computing power of Nvidia Titan X, we need to install Cuda Toolkit 8 (https://developer.nvidia.com/cuda-toolkit) and cuDNN 5.1 (https://developer.nvidia.com/cudnn).

3. Program Manual

3.1 Code Snippets

```
[strings]
# Mode : train, test, serve
# mode = train
mode = test
train_enc = data/train.enc
train_dec = data/train.dec
test_enc = data/test.enc
test_dec = data/test.dec
# edit test.enc to test.dec by Ken

# folder where checkpoints, vocabulary, temporary data will be stored
working_directory = working_dir/
[ints]
# vocabulary size
#   20,000 is a reasonable size
enc_vocab_size = 20000
dec_vocab_size = 20000
# number of LSTM layers : 1/2/3
num_layers = 3
# typical options : 128, 256, 512, 1024
layer_size = 256
# dataset size limit; typically none : no limit
max_train_data_size = 0
batch_size = 64
# steps per checkpoint
#   Note : At a checkpoint, models parameters are saved, model is evaluated
#          and results are printed
steps_per_checkpoint = 500
[floats]
learning_rate = 0.5
learning_rate_decay_factor = 0.99
max_gradient_norm = 5.0
```

Seq2seq.ini

```
def train():
  # prepare dataset
  print("Preparing data in %s" % gConfig['working_directory'])
  enc_train, dec_train, enc_dev, dec_dev, _, _ = data_utils.prepare_custom_data(gConfig['working_directory'],gConfig['train_enc'],gConfig['train_dec'],gConfig['test_enc'],g

  # Only allocate 2/3 of the gpu memory to allow for running gpu-based predictions while training:
  gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.666)
  config = tf.ConfigProto(gpu_options=gpu_options)
  config.gpu_options.allocator_type = 'BFC'

  with tf.Session(config=config) as sess:
    # Create model.
    print("Creating %d layers of %d units." % (gConfig['num_layers'], gConfig['layer_size']))
    model = create_model(sess, False)

    # Read data into buckets and compute their sizes.
    print ("Reading development and training data (limit: %d)."
           % gConfig['max_train_data_size'])
    dev_set = read_data(enc_dev, dec_dev)
    train_set = read_data(enc_train, dec_train, gConfig['max_train_data_size'])
    train_bucket_sizes = [len(train_set[b]) for b in xrange(len(_buckets))]
    train_total_size = float(sum(train_bucket_sizes))

    # A bucket scale is a list of increasing numbers from 0 to 1 that we'll use
    # to select a bucket. Length of [scale[i], scale[i+1]] is proportional to
    # the size if i-th training bucket, as used later.
    train_buckets_scale = [sum(train_bucket_sizes[:i + 1]) / train_total_size
                           for i in xrange(len(train_bucket_sizes))]

    # This is the training loop.
    step_time, loss = 0.0, 0.0
    current_step = 0
    previous_losses = []
    while True:
      # Choose a bucket according to data distribution. We pick a random number
      # in [0, 1] and use the corresponding interval in train_buckets_scale.
      random_number_01 = np.random.random_sample()
      bucket_id = min([i for i in xrange(len(train_buckets_scale))
                       if train_buckets_scale[i] > random_number_01])

      # Get a batch and make a step.
```

Train function

```
    # Get a batch and make a step.
    start_time = time.time()
    encoder_inputs, decoder_inputs, target_weights = model.get_batch(
        train_set, bucket_id)
    _, step_loss, _ = model.step(sess, encoder_inputs, decoder_inputs,
                                 target_weights, bucket_id, False)
    step_time += (time.time() - start_time) / gConfig['steps_per_checkpoint']
    loss += step_loss / gConfig['steps_per_checkpoint']
    current_step += 1

    # Once in a while, we save checkpoint, print statistics, and run evals.
    if current_step % gConfig['steps_per_checkpoint'] == 0:
      # Print statistics for the previous epoch.
      perplexity = math.exp(loss) if loss < 300 else float('inf')
      print ("global step %d learning rate %.4f step-time %.2f perplexity "
             "%.2f" % (model.global_step.eval(), model.learning_rate.eval(),
                       step_time, perplexity))
      # Decrease learning rate if no improvement was seen over last 3 times.
      if len(previous_losses) > 2 and loss > max(previous_losses[-3:]):
        sess.run(model.learning_rate_decay_op)
      previous_losses.append(loss)
      # Save checkpoint and zero timer and loss.
      checkpoint_path = os.path.join(gConfig['working_directory'], "seq2seq.ckpt")
      model.saver.save(sess, checkpoint_path, global_step=model.global_step)
      step_time, loss = 0.0, 0.0
      # Run evals on development set and print their perplexity.
      for bucket_id in xrange(len(_buckets)):
        if len(dev_set[bucket_id]) == 0:
          print("  eval: empty bucket %d" % (bucket_id))
          continue
        encoder_inputs, decoder_inputs, target_weights = model.get_batch(
            dev_set, bucket_id)
        _, eval_loss, _ = model.step(sess, encoder_inputs, decoder_inputs,
                                     target_weights, bucket_id, True)
        eval_ppx = math.exp(eval_loss) if eval_loss < 300 else float('inf')
        print("  eval: bucket %d perplexity %.2f" % (bucket_id, eval_ppx))
      sys.stdout.flush()
```

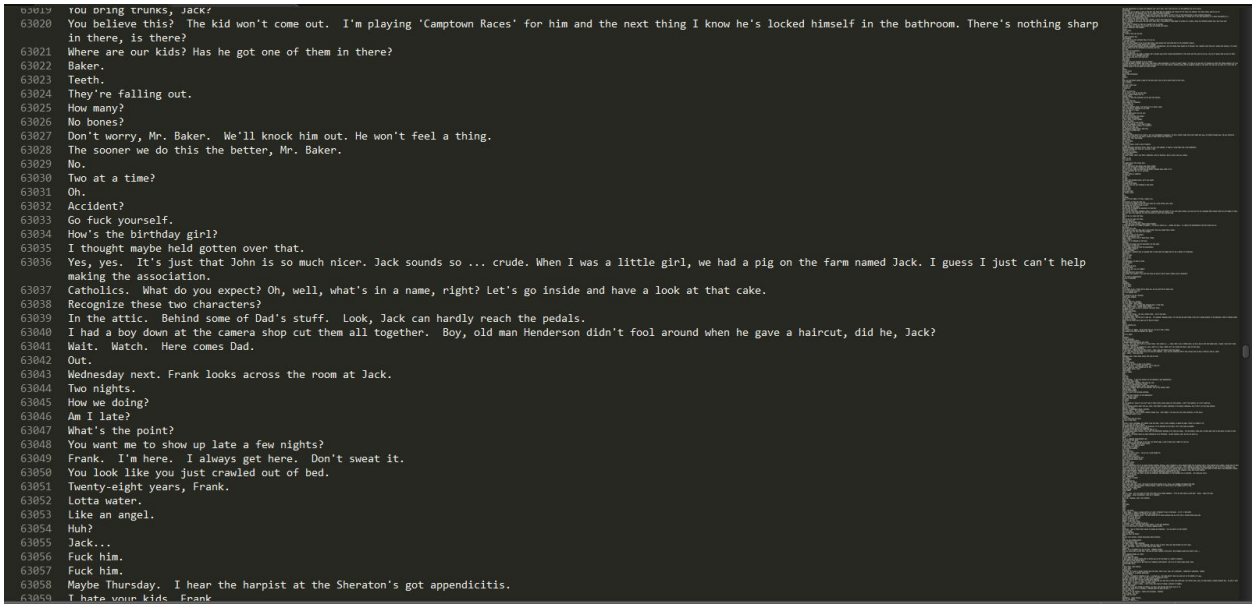
Train function

```
def decode():

  # Only allocate part of the gpu memory when predicting.
  gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.2)
  config = tf.ConfigProto(gpu_options=gpu_options)

  with tf.Session(config=config) as sess:
    # Create model and load parameters.
    model = create_model(sess, True)
    model.batch_size = 1  # We decode one sentence at a time.

    # Load vocabularies.
    enc_vocab_path = os.path.join(gConfig['working_directory'],"vocab%d.enc" % gConfig['enc_vocab_size'])
    dec_vocab_path = os.path.join(gConfig['working_directory'],"vocab%d.dec" % gConfig['dec_vocab_size'])

    enc_vocab, _ = data_utils.initialize_vocabulary(enc_vocab_path)
    _, rev_dec_vocab = data_utils.initialize_vocabulary(dec_vocab_path)

    # Decode from standard input.
    sys.stdout.write("> ")
    sys.stdout.flush()
    sentence = sys.stdin.readline()
    while sentence:
      # Get token-ids for the input sentence.
      token_ids = data_utils.sentence_to_token_ids(tf.compat.as_bytes(sentence), enc_vocab)
      # Which bucket does it belong to?
      bucket_id = min([b for b in xrange(len(_buckets))
                       if _buckets[b][0] > len(token_ids)])
      # Get a 1-element batch to feed the sentence to the model.
      encoder_inputs, decoder_inputs, target_weights = model.get_batch(
          {bucket_id: [(token_ids, [])]}, bucket_id)
      # Get output logits for the sentence.
      _, _, output_logits = model.step(sess, encoder_inputs, decoder_inputs,
                                       target_weights, bucket_id, True)
      # This is a greedy decoder - outputs are just argmaxes of output_logits.
      outputs = [int(np.argmax(logit, axis=1)) for logit in output_logits]
      # If there is an EOS symbol in outputs, cut them at that point.
      if data_utils.EOS_ID in outputs:
        outputs = outputs[:outputs.index(data_utils.EOS_ID)]
      # Print out French sentence corresponding to outputs.
      print(" ".join([tf.compat.as_str(rev_dec_vocab[output]) for output in outputs]))
      print("> ", end="")
      sys.stdout.flush()
      sentence = sys.stdin.readline()
```

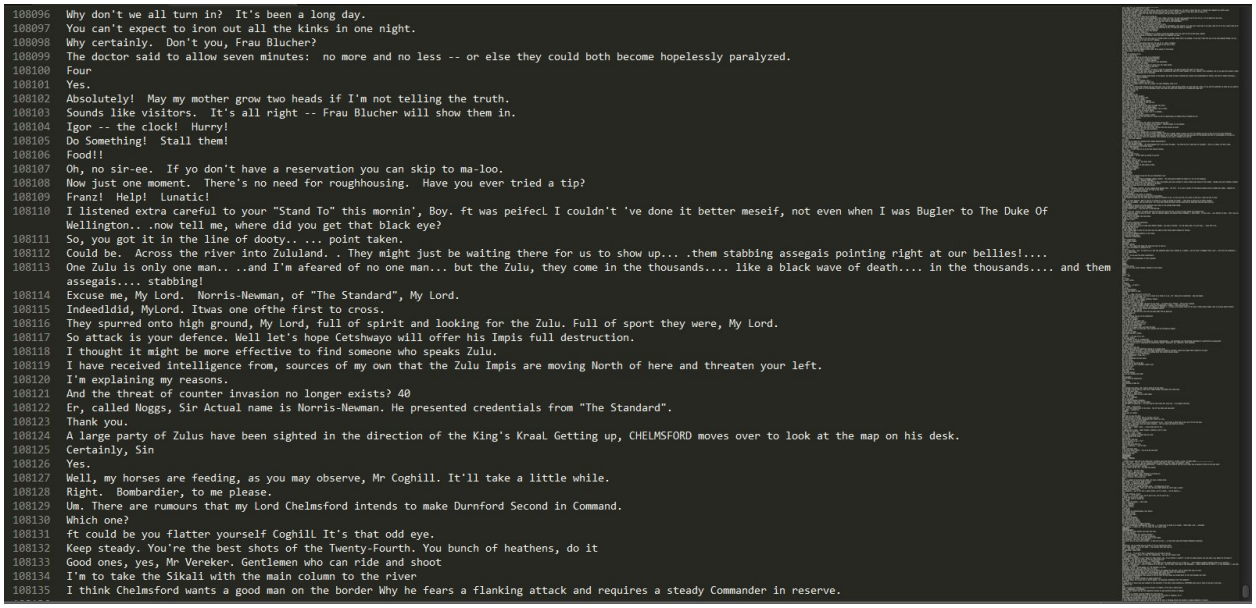
Decode function

3.2 Screenshots

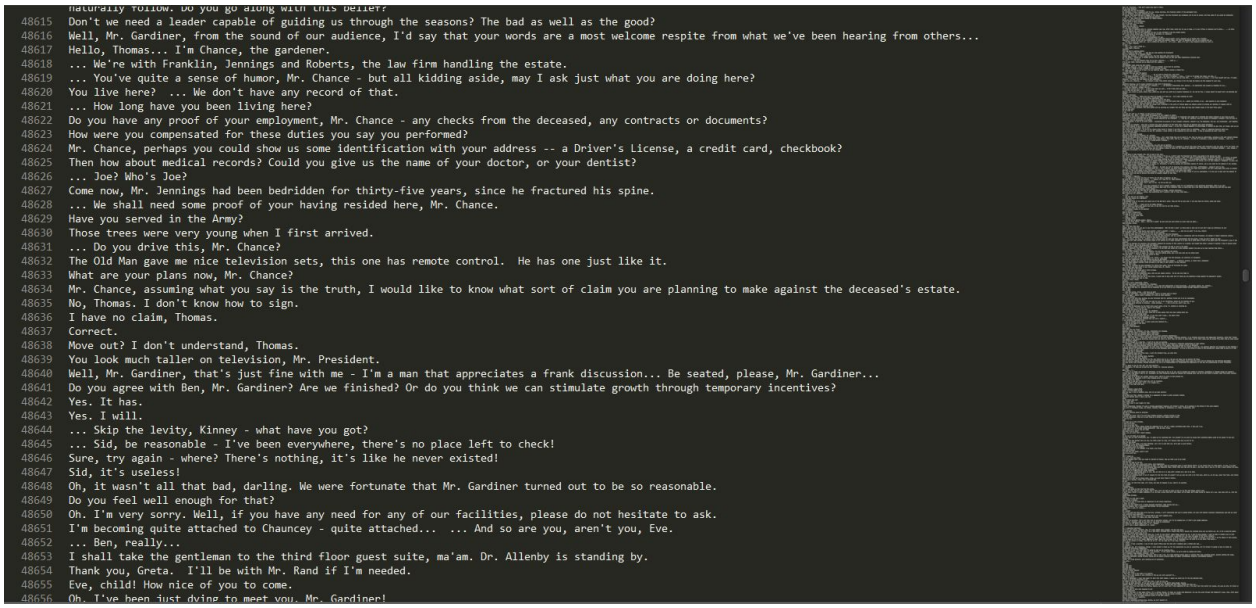
3.2.1 Data Set Training



Train Decoder



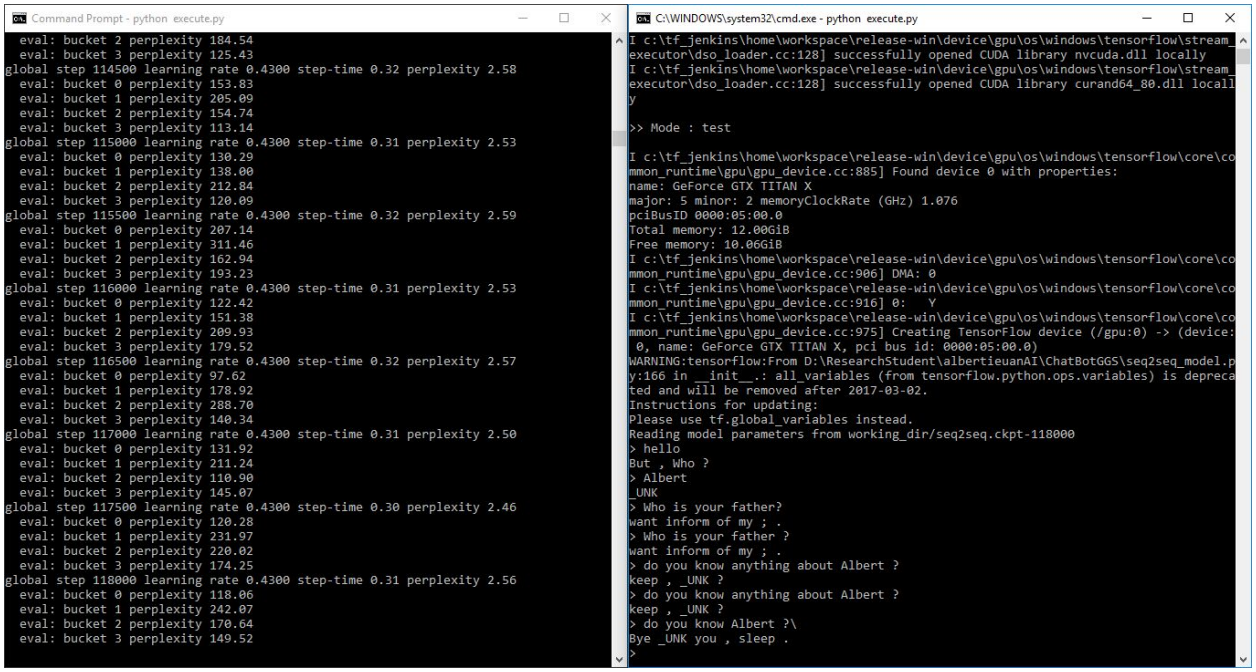
Train Decoder



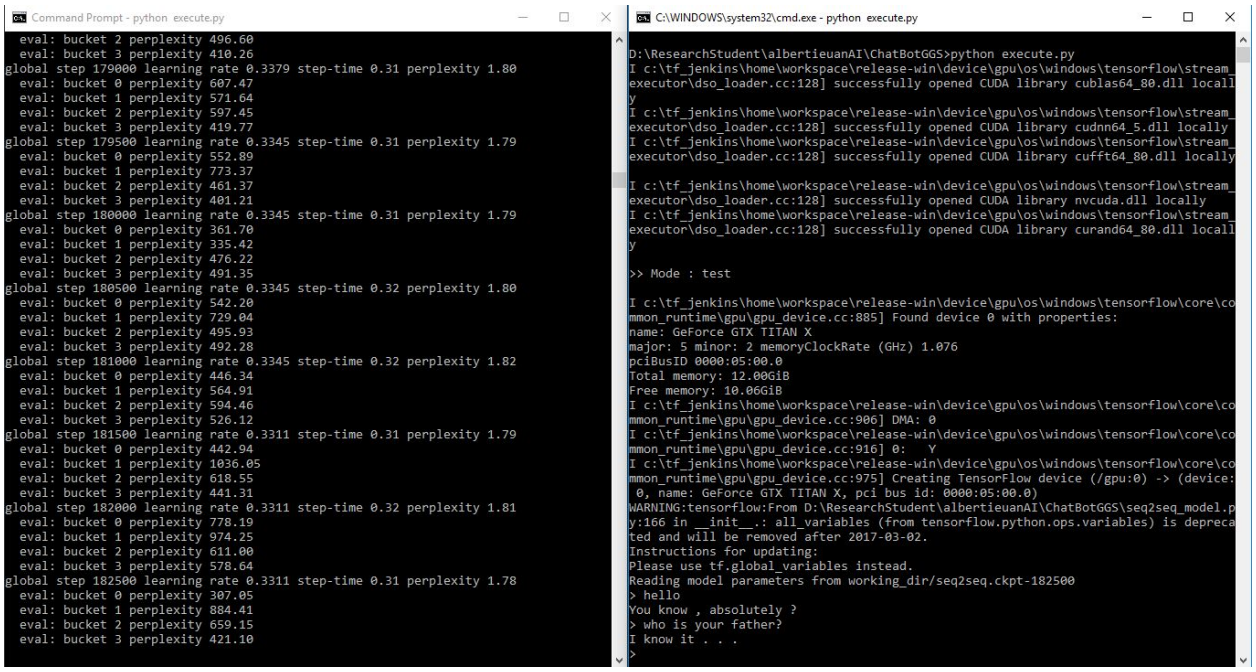
Train Encoder

Train Encoder

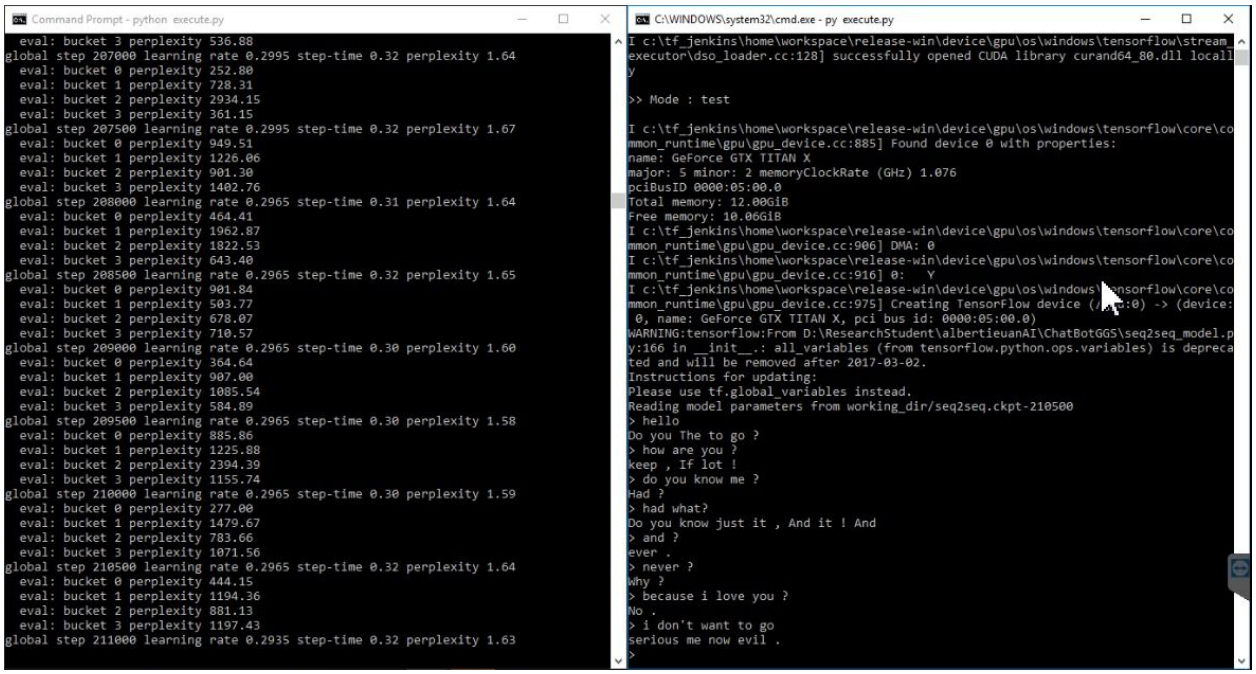
3.2.2 Result



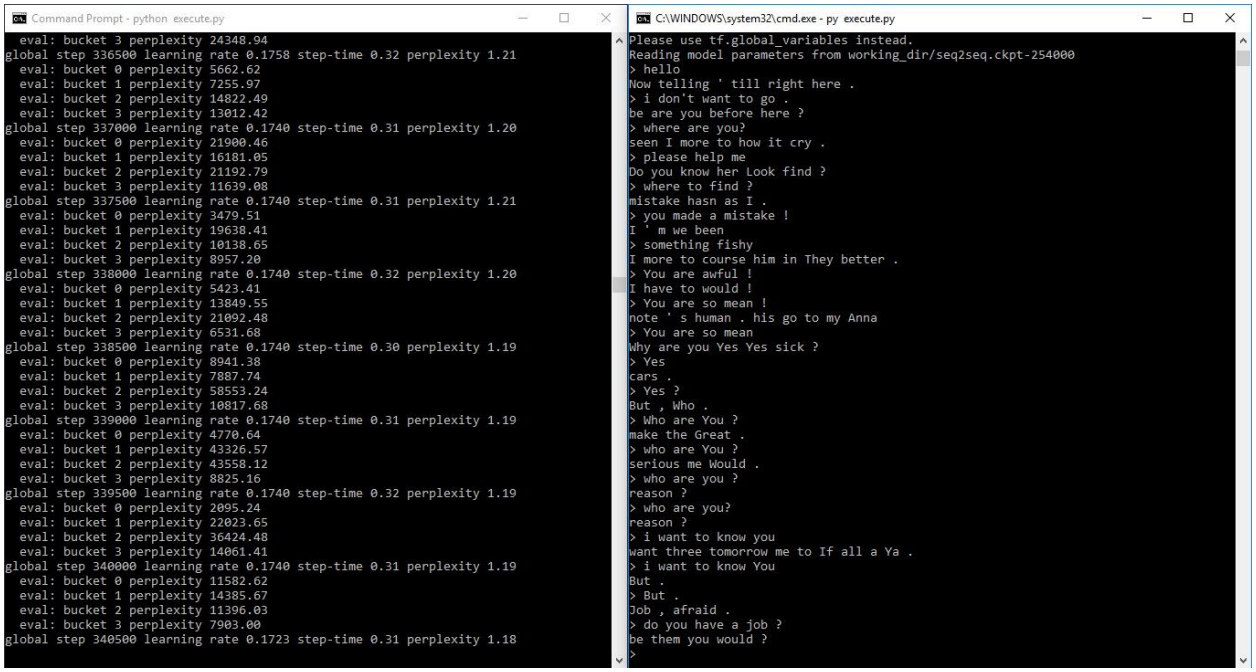
Global step at 118000 result



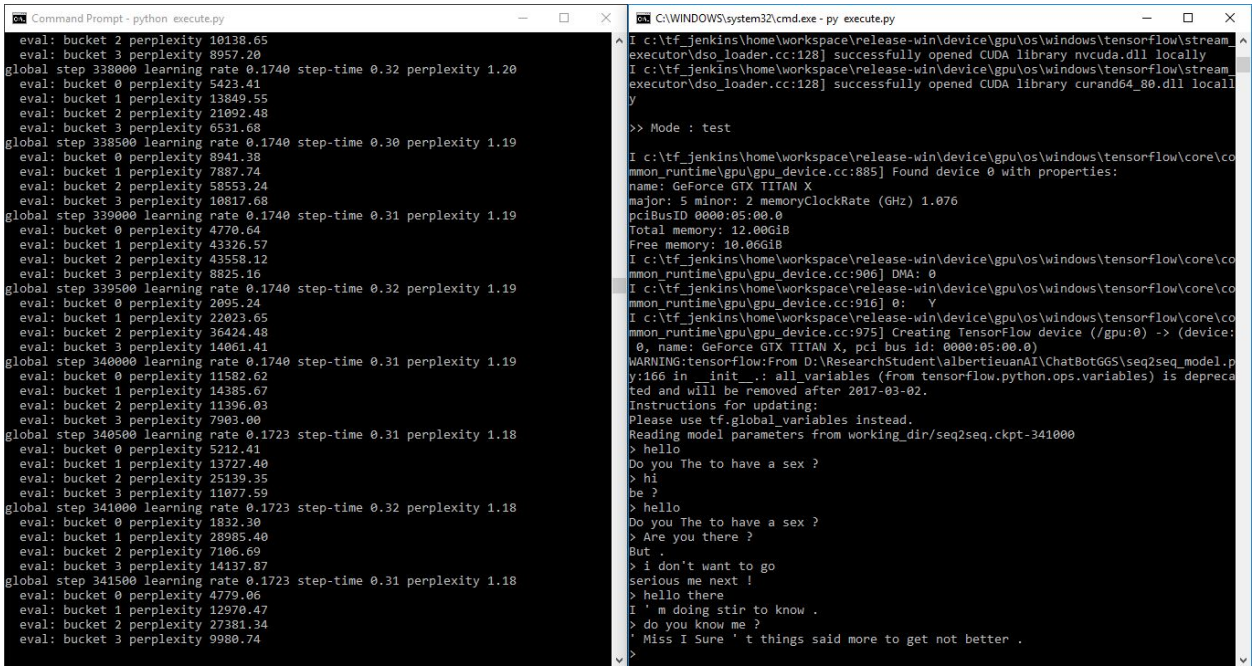
Global step 182500 result



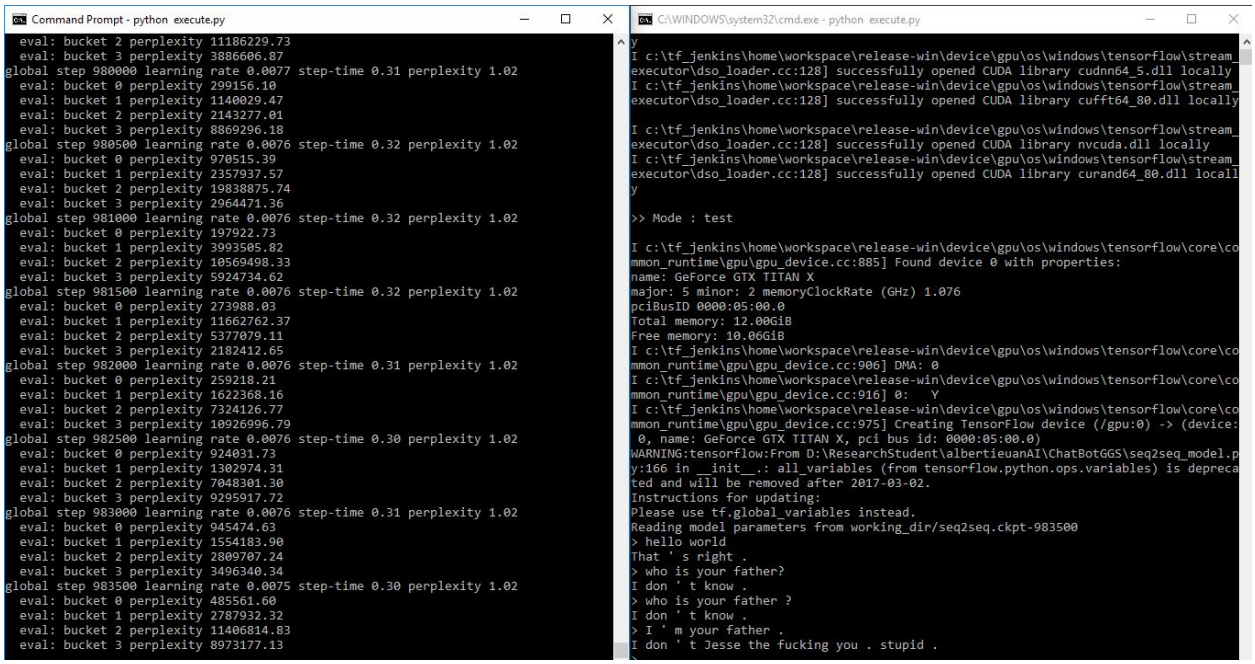
Global step at 210500 result



Global step at 254000 result



Global step 341000 result



Global step 983500 result

4. Conclusion

A chatbot with AI implementation is like a child with an empty brain. This child must learn from her parent(s), which in this case are her creator(s). The learning or training part can be different according to the data set which was given by her parent(s). The creator(s) can determine how much data their daughter will study. More data would mean more time to learn/study. The creator can not only determine how much the data, but also the learning rate, the layer of LSTM and even the vocabulary. For our chatbot, we made her study 100,000 training data sets and a vocabulary of 20,000 with a learning rate of 0.5 which has different time for each computer depends on the specification of the computer. The perplexity will decrease for every time she learns more data.

Bibliography

Github Repositories
https://github.com/1228337123/tensorflow-seq2seq-chatbot
https://github.com/llSourcell/tensorflow_chatbot

Development Tools, Software, APIs, Libraries, Datasets
https://www.tensorflow.org/
https://www.tensorflow.org/tutorials/seq2seq/
https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html
https://developer.nvidia.com/cudnn
https://developer.nvidia.com/cuda-toolkit

Helpful Articles and Videos
https://www.youtube.com/watch?v=SJDEOWLHYVo
http://colah.github.io/posts/2015-08-Understanding-LSTMs/