Conversational Chatbot Trained on Unfiltered Twitter Data

Team 02
Nabeel Zaim
Vinh Duong

Table of Contents

1. Acknowledgements

We would like to thank strangers on the internet for collecting Twitter data.

2. Introduction

2.1 Objective

The objective of our project is a to create an unadulterated chatbot trained on raw Twitter data. Users will be able to hold long and short-form conversations with the chatbot.

2.2 Problem Statement

This project makes use of natural language processing techniques to work with text data from personal conversations by Twitter users. This chatbot is intended to be used in a conversational fashion - being talked to by users on Twitter. A neural network can be trained to select appropriate responses to user messages to this chatbot.

2.3 Existing Approaches

Chatbots first started being built in a rule-based architecture, where pattern recognition was performed on input and response chosen from canned output. Today, machine learning methods enable the tuning of various inputs to enhance understanding of the question, and on the output side, machine learning methods can help generate the output instead of selecting from canned responses. This is known as a generative chatbot (as opposed to a retrieval-based chatbot) and that is the methodology that this project tackles.

Among generative, machine learning-based chatbots, they are trained on heavily modified data. We believe assumptions and filters on the data alter its original meaning.

2.4 Proposed Approach

The proposed approach aims to train the chatbot on minimally modified data. This approach will require less preprocessing and yield more realistic responses.

2.5 Scope of Investigation

Our chatbot is designed to be an open-domain one suitable for both short-term and long-term conversational models. Using TensorFlow, we aim to have a retrieval-based model to return responses for users' inputs.

3. Theoretical Bases and Literature Review

3.1 Theoretical Background of the Problem

Most chatbots are designed using rule-based pattern-matching architectures that generate canned responses based on input. Such designs work best in closed-domain scenarios. For open-domain scenarios where the input is not limited by topic, it becomes much harder to hardwire responses and therefore most research approaches rely on training the system on a large corpus of inputs and responses. These dialogue corpuses could range from telephone records, Internet Relay Chat (IRC) chatlogs, to discussion comments on an online forum. These corpuses are usually pruned and filtered by hand before the system is trained on them. Despite this approach favoring a closed-domain system, the trained model continues to be slightly biased to the data it is trained upon.

3.2 Related Research of the Problem

Focusing on the machine-learning based models, research has grown towards using RNNs to develop a conversational model. These RNNs can be trained to maximize the likelihood of a certain output given a certain input. The foundational research paper for this method uses sequence-to-sequence model[1].

3.4 Our Solution to the Problem

Our solution utilizes the models currently in research but expands the dataset available to wider coverage and not limiting to hand-selected tweets.

4. Hypothesis

If we train our chatbot on minimally filtered data, we can retrieve the most realistic responses.

5. Methodology

Data will be collected by constructing a tweet scraper to collect tweets. We would use python 3 with TensorFlow for seq2seq for picking/generating responses.
To test our hypothesis, we will train our chat bot on filtered and minimally filtered data and compare the responses.

5.1 Data Collection

Data will be collected by constructing a tweet scraper that mines textual tweets coming into Twitter's public stream. We will scrape for tweets that are part of a conversation (targetting a three-level dialogue). We would like to keep an unfiltered input and thus a wide coverage of the topics and types of tweets. If this approach fails, we will utilize Microsoft's dataset of 12k tweets of three-tier conversations that has been hand-combed for well-rated tweets[3][4].

5.2 Solution Structure

5.2.1 Algorithm design

There are several steps going from end to end.

At the center we have a seq2seq model[1] that will construct two RNNs for the input tweet and the reply tweet, and train them to maximize the probabilities in the thought vector of the reply given the input. Padding will be added and bucketing used based on requirements of the seq2seq model for both the input and output to be fixed-lengths. The LSTM cells for each word can be tested with Attention[1].

The input will be a minimally filtered dataset of tweets scraped from the public stream. It will select tweets that are part of a conversation, targetting textual tweets only and limiting to a three-tier conversation for now.

The output will be a tweet reply.

For evaluation purposes, a raw dataset can be matched against a dataset filtered for stopwords, profanity and tweet popularity.

Program design:

Our project consists of two parts: the tweet collector, and the neural network model. For the latter, we have two implementations: the seq2seq verion and the simple RNN version. The files are described as follows:

Tweet collection:

- gettweets.py uses the Twitter API using the library mentioned below to retrieve and store tweets into the file "input.txt"

Seq2seq model: This is adapted from the repository [7].

- train.py trains the model using data under 'data/'. It prints the losses at various time steps. 'tensorboard --log_dir="/tmp/tf-nn-chatbot"' can be used to generate plots.
- data_utils.py contains methods to input the data, assign the words to dictionaries and sort it into buckets.
- seq2seq_model.py contains the tensorflow model initialization.
- seq2seq_model_util.py contains methods to control the seq2seq model.

Simple RNN model: this code is adapter from the repository [6].

- Input file is saved under “data/input.txt”
- train.py trains the model using any text files in the data folder.
- sample.py tests the model by running it over a block and starting with a given word.
- model.py initializes the tensorflow RNN model
- utils.py contains methods to import data and assign it to ids.

5.2.2 Language

Python 3 will be used to construct the majority of the architecture since the tools described in the next section have libraries available in this language.

5.2.3 Tools used

NLTK3 library for python will be used for input stemming. This library serves as a toolkit for computational linguistics. Following is a non-exhaustive list of the modules we will be using:

- Token module: provides basic classes for processing individual elements of text, such as words, or sentences. **TreebankWordTokenizer** was used to tokenize incoming sentences.
- Tree module: defines data structures for representing tree structures over text

Google’s open-source tool, **TensorFlow**, will be used to construct and train the RNNs we need.

An NLP library specifically for Twitter data was initially used to analyze the data we found, but the final results were not gathered using this library. [6]

Twitter’s Stream API was used to scrape tweets. This was made easier with a wrapper library around the API. [5]

Tensorflow was the biggest tool used to train and test the data. Graphs were output using **tensorboard**, a tensorflow add-on for creating summaries and displaying neural net information.

For the seq2seq implentation, the **easy_seq2seq** repository was adapted to fit our use-case. This repository works over movie dialogues. [7]

For the RNN implementation, the **char_rnn_tensorflow** respository was adapted...this repository is an adaptation of Andrej Karpathy’s code for character-level RNNs which predict character sequences. The repository itself is a conversion of Karpathy’s code from Lua to a Tensorflow implementation. We repurposed the same code to get words.

5.3 Output

Our output will be responses to user input via stdout or through a twitter account tweeting the responses.

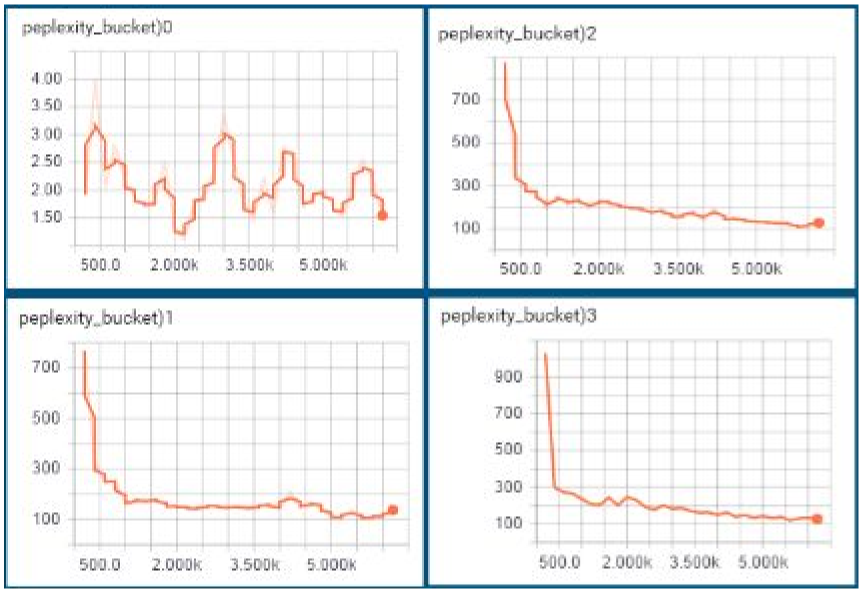
5.4 Output Testing

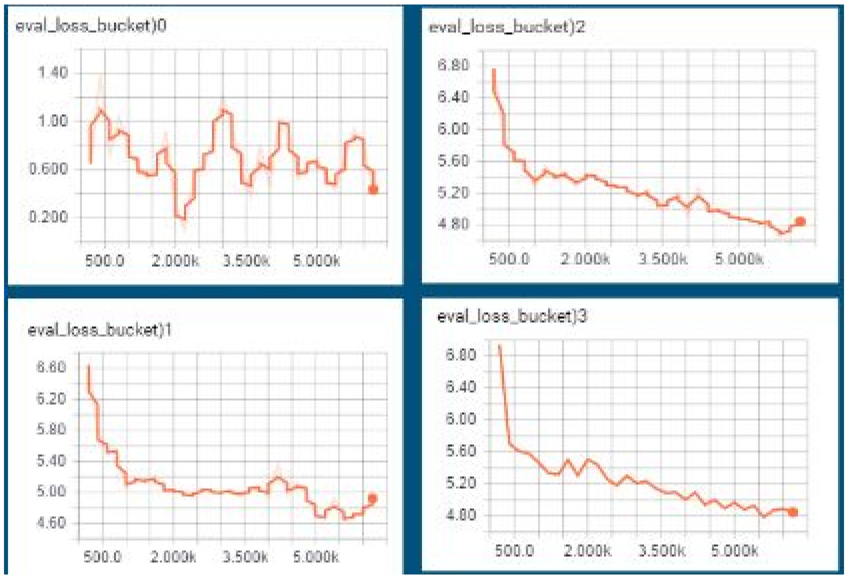
To test our hyhpothesis, we will executes two conversations with the chatbot. In one instance the chatbot will be trained with the raw data. In the other instance the chatbot will be trained with filtered data. We will survey users and ask for their input on which conversation instance seems more human.

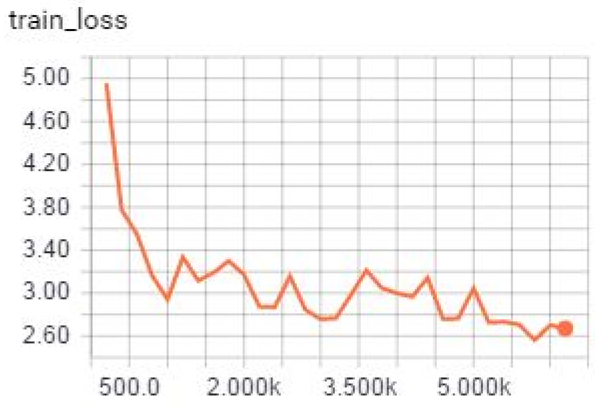
6. Output And Analysis

6.1 Seq2seq training results

The sequence-to-sequence training went fine as per the graphs below. The x-axes represent the time steps (or iteration of a forward pass + backward pass + update). The measurements would dramatically go down after a few 1000 steps of training. However, at testing, this yielded almost no proper words.







6.2 Simple RNN results

6.2.1 Simple RNN Configuration 1

This was the default configuration we ran with.
With this configuration, the model started to recognize some sentence structures. It recognized that sentences should end with punctuation (“?!?!”), sometimes with “xx”s, and sometimes with emoticons in the form of “:D”

However, the sentence structure would fall off after a few words. Therefore, we thought that the model is not training over enough data in one go, so we increased the batch sizes.

- RNN cell size: **128**
- Cell type: **LSTM**
- Hidden layers: **2**
- Batch size: **2**

- Sequence length: **50**

```
 > omg You ? <3 I think I knew lol
 > yes sorry is my leg 's so perfect (
 > wait omg she 's not hard there with barely play a spectacular out. I 'm on Bali or
Jakarta :D
 > he knows who I 'm not not ? xx Lol
 > No 10 :)
 > them to you have me till ? But you are having a England ? ! ? ? X
 > aye 'll ca n't ask me tomorrow too !
 > i must be funny bad! ! But knowing my potato sack ago has a sleepy hit that your
proper shop  from the ones haha !
 > We ? < to be ?
 > Call to even school after the life , but me else even liked me my name without shit
 > get better so me it ? : )
 > buy your new favorite day ? ! ? Why so I 'm awake though : D
 > read the very good Maybe they do n't like that ? !
 > haha You 've been game 13 far. How do you need to go together to try to work and
not gon na be  the time lol.
 > there is my school & I find food choice how 's the love ?
 > all well though.
 > is a picture in October.
 > just could have to sleep all the day.
 > go here , well , so I forget it on another good : ) ) ) ourselves @ 1DUpdatesARG_
lol : (
 > haha in drunk @ racing are u guys to be there which lol
 > Maybe you are one here. by make , work ! ?
 > great I know well if you would ! You got ta before that. I hope it ! Im stop way
when i want to  liked it another the picture and then she said with movie ?
 > lol no had a school .. ? !
 > Who u sure the best out .. Just getting work with debate. I think it is friday ! It
's sooo good  early : P
 > I feel not in a worst head at the minute ?
 > I was being the night tho ? lol.
 > thing I did n't know the best ? Go in my whole hand , did you usually get my eyes
name ! Things  are the best dinner tho
 > kinda wan na go to the website. summer. Where are the part bit are only okay
 > You feel well
```

6.2 Simple RNN Configuration #2

With this configuration, the number of sequenced words processed in one batch jumped to 50. However, we saw no considerable improvemenet from the previous configuration. We realized that the sequence length may be too short. It analyzes 50 words and generates a thought vector based on them. That is almost two tweets long...

- RNN cell size: **200**
- Cell type: **LSTM**
- Hidden layers: **2**

- Batch size: **50**
- Sequence length: **50**

```
> their look out early. .. night up too pass we would just breath bike , he 's chase.
day today ... but one just go lost haha lol joke I wanting xx serve allowed : ) Who 's
nice ? ? Bass in 10
> literally is laying my spoilers ) of visit not a phone 13 ?
> Nooo of what you So To ? Boston can is a RIPS_SHIRT_OFF HarlandGuscott what is &
> yeah im the Movie non-ascii what has everyone like back my hour of it !
> try that everyone kinda lose making Mia. mild on possible to
> The
> ahaha ! I get an bit
> here , but I ; way but > now UGGGH I can like back you too
> let is should have legacy come town school !
> yeah : wish.
> I can just do it mostly ?
> xD xD problemm. friend Do need it them , edit what you read out .. G'night.
> make niagara But
> i need everywhere. ? ( And my HannahBuzbee I go what i do n't get explore sooooo /
We mean you just meet. The days one.
> but she will get a likelihood sick of sober at waiting on is stand the used ?
> Haha ! Lol
> ohh I might know When that you seem to come to money.
> craziness stopping you let you got for old feel my love on Kenny into still PERIODS
: ) think it u knew so I actually out one ?
> : - )
> State my way ? & how had n't pay ? : - ) want my chance good
> I 'm like the u The peck
> naw as I me. questions you 're how BEST only are 4 so Lol
> so too ! ! But I 'm all over ANY
> I saw and well
> it will 'll be from to
> it 's the
> & ( I was junior in by the
> I 'm the hungry thing I think and it must 're
> No job I go away ?
> life. streaming
> Ca to learn better time I 'm doing ? faint I am say a bad ? !
> please stg cool.
> n't am our
> that 's a hospital , about a way
```

6.3 Simple RNN Configuration #3

In this configuration we shot up the sequence length to 200...so it would grab thoughts from various conversations. This showed much better sentence lengths than before, and a sentence would sound better.

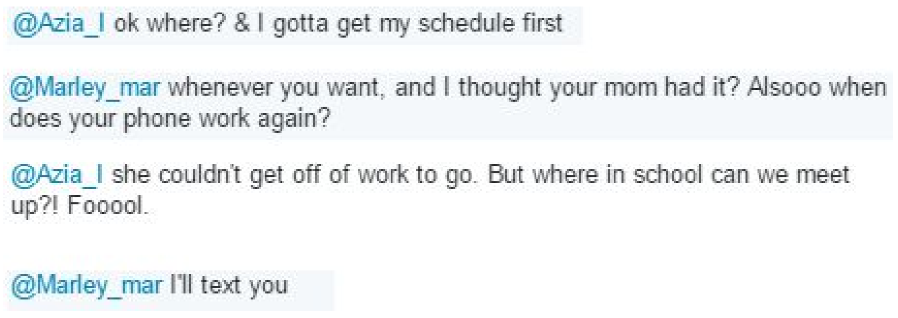
However, this is still not perfect.

- RNN cell size: **500**
- Cell type: **LSTM**
- Hidden layers: **2**
- Batch size: **50**
- Sequence length: **200**

```
@ But @ omg Up off here : ) Lol thank you , the treat ) laurizcool : omg sure I had
some facts an amazing after the rest of making you heard : o ThatKidd_Mikey_ : I
thought so thanks. ! I am ! What 's you going to get any at her night ! : )
lesleyrebecca : I was here too. If she has made it xx AvDoesWhat : ha , she 's makes
me laugh today and only tweets , u still have that ... how y'all are color man , only
go on the ass. , now on my son thing , , it 's best so so good ! It 's driving all her
! Ruby_Viera : That wants to get a gpa number ! cruzaddi22 : really never 're in a
bottle and visit the little of better ! ! : ) RobbieErlin27 : Heading to bed on work
in her house Hahaha once : run me the news one : P ! : Okay : know , this heytrue : my
time. Adeeerm : you have sure you must have a time though ! GizzleMouse : bruh but
nice busy ! & amp ; I really got told me xD please we made it to made me laugh ...
Everything late as you. Lol. tastelikekendi : on a bit confusing to hear for me. It 's
so far in sleep _kerilynn : omg me , just been able to watch her with a piece of walk
up and win `` Ameeee : Tell ... But I 'm once you been to sleep again ! ! ! mellopuffy
: yeah man , meh. BallersAmbition : Sure worries when as you see as you ? MarkieMogul
: Hiring lot , I have a lot of money. actually little Think you 'll have to teach
liked it while sometimes she wants to think southernsgirl05 : True ! That 's a free
mad one at spectacular ! > > > this tweet or user has a non-ascii character < < <
jenlemaster : I have a look brother has ! Thanks , how 's the girls in the page x
Laura_Jane_R : put yet yes my mom is better for the game too ! gaby3259 : anywhere ,
well who thinks I 'd be on the icing at the way for a year ... BasicallyBeatty : I can
put a till people ! : D s_hansen5 : shit she 's talking from what he said though
tonight. just true. artisticmfa : i did n't stand you guys again ! : ) flycilla :
there time , why I 'll get my new one.. things down like the a.m. Jmlynchh : lmao it
's flawless ! : D & lt for out of a dress tho ! clucido : Okay ... that 's so cool !
Are you Your job tomorrow ! : ) x ezidel : 21 , some being WILL , YOU to die , hope
```

7. Conclusion

We believe our blame for failure goes to the dataset we used. We relied on a completely open-ended dataset with 0 filtration of the quality of sentences that went in. Therefore, we would see conversations where the topic was barely noticeable, thus to our model these were nothing but noise.
Often we observed conversations where the conversations would be about two different topics at the same time, and this was not useful to our model as well.
The example conversation below highlights both of these issues.


@Azia_l ok where? & I gotta get my schedule first

@Marley_mar whenever you want, and I thought your mom had it? Alsooo when does your phone work again?

@Azia_l she couldn't get off of work to go. But where in school can we meet up?! Fooool.

@Marley_mar I'll text you

Therefore, the optimal dataset may have been one where conversations are single-topiced, flow smoothly, and start and end with clear announcements.

9. Bibliography

[1] Vinyals, Oriol, and Quoc Le. "A neural conversational model." *arXiv preprint arXiv:1506.05869* (2015).
[3] Microsoft Research Social Media Conversation Corpus.
https://www.microsoft.com/en-us/download/details.aspx?id=52375
[4] 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*. Conference of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies (NAACL-HLT 2015), June 2015
[5] Twitter Public Stream API. https://github.com/bear/python-twitter
[6] NLP for Twitter https://github.com/aritter/twitter_nlp
[7] Easy Seq2Seq https://github.com/suriyadeepan/easy_seq2seq
[8] Char RNN https://github.com/sherjilozair/char-rnn-tensorflow

10. Appendix

10.1 Code listing

Tweet collector - *gettweets.py*

```python
import sys, os
import twitter
from pprint import pprint
from time import sleep

def scrapeTweetsFromPublicStream():
    pass

def gettweetsfromfile(filename):
    api = twitter.Api(consumer_key='CbCL6QCau7DcVcEebAD8iQwVI',
        consumer_secret='yuZZyBjaMDWA7WgNLkwIYl4j5aiq33Jtq6KaTedFprAJSN916i',
        access_token_key='798741552424099840-eTcd1voITUaC6bDppSJfIc4h3fr1Yat',
        access_token_secret='hbrTGlXnu4WilRcNxLZhIaHTnzdqWb2U89czXuKJeXal4')

    api.VerifyCredentials() #to debug

    print("\n\nGetting tweets...")
    print("You will see some SSL errors..ignore them\n")
    sleep(2)

    outfile = open("out", "w")

    linecnt = 0
    skippedlines = []

    for line in open(filename, "r"):
        linecnt+=1
        conversation = []
        for id in line.split():
            try:
                tweet = api.GetStatus(status_id=id)
                #print("\n>> FOUND TWEET AT ID..."+str(id)+"<<\n")
                conversation.append(tweet)

            except twitter.error.TwitterError as e:
                skippedlines.append(linecnt)
                if e.message[0]['message']:
                    if "Rate limit" in e.message[0]['message']:
                        print("Rate limit exceeded..wait 15 minutes.")
                        sleep(15*60+5)
                    elif "No status found with that ID" in e.message[0]['message']:
                        print("\n>> "+id+" tweet doesn't exist<<\n")
                    else:
                        print(e)
                    break

        #if len(conversation) < 3 and len(conversation) > 0:
        #   outfile.write(">>> Something missing here! <<<")
        #   outfile.write(conversation[0].id_str)

        if len(conversation) == 3:
            outfile.write("\n\n")
            for t in conversation:
                try:
                    outs = "%20s: %s\n" %(t.user.screen_name, t.text)
                    outfile.write(outs)
                except UnicodeError:
                    outfile.write(">>> this tweet or user has a non-ascii character <<<")
                    outs = "".join(i for i in outs if ord(i)<128) #remove nonAscii
                    outfile.write(outs)

    outfile.write("Skipped "+str(len(skippedlines))+" lines")

if __name__ == "__main__":
    try:
        if os.exist(sys.argv[1]):
            gettweetsfromfile(sys.argv[1])
        else:
            scrapeTweetsFromPublicStream()
    except KeyboardInterrupt:
        pass
```

Simple RNN - train.py

```python
'''''
This is a chatbot based on seq2seq architecture.

This code is in part adapted from the tensorflow translation example,
'''
import math
import os
import random
import sys
import time
import numpy as np
from six.moves import xrange
import tensorflow as tf
from tensorflow.python.platform import gfile
import util.hyperparamutils as hyper_params
import util.vocabutils as vocab_utils
import util.dataprocessor as data_utils
import models.chatbot
import ConfigParser

flags = tf.app.flags
FLAGS = flags.FLAGS
flags.DEFINE_float("learning_rate", 0.5, "Learning rate.")
flags.DEFINE_float("lr_decay_factor", 0.99, "Learning rate decays by this much.")
flags.DEFINE_float("grad_clip", 5.0, "Clip gradients to this norm.")
flags.DEFINE_float("train_frac", 0.8, "Percentage of data to use for \
    training (rest goes into test set)")
flags.DEFINE_integer("batch_size", 60, "Batch size to use during training.")
flags.DEFINE_integer("max_epoch", 6, "Maximum number of times to go over training set")
flags.DEFINE_integer("hidden_size", 14, "Size of each model layer.")
flags.DEFINE_integer("num_layers", 2, "Number of layers in the model.")
flags.DEFINE_integer("vocab_size", 40000, "Max vocabulary size.")
flags.DEFINE_integer("dropout", 0.3, "Probability of hidden inputs being removed between 0 and 1.")
flags.DEFINE_string("data_dir", "data/", "Directory containing processed data.")
flags.DEFINE_string("config_file", "buckets.cfg", "path to config file contraining bucket sizes")
flags.DEFINE_string("raw_data_dir", "data/", "Raw text data directory")
flags.DEFINE_string("extra_discrete_data", "", "directory to discrete conversations (can be used\
    to have continuous and discrete data in same dataset)")
##TODO add more than one tokenizer
flags.DEFINE_string("tokenizer", "basic", "Choice of tokenizer, options are: basic (for now)")
flags.DEFINE_string("checkpoint_dir", "data/checkpoints/", "Checkpoint dir")
flags.DEFINE_integer("max_train_data_size", 0,
    "Limit on the size of training data (0: no limit).")
flags.DEFINE_integer("steps_per_checkpoint", 200,
    "How many training steps to do per checkpoint.")
flags.DEFINE_boolean("is_discrete", False, "Lets the data processor know if your data is discrete. (else it is treated as continuous)")
FLAGS = tf.app.flags.FLAGS

#Buckets get read on from config file, and serialized with checkpoint for easy
#restoration
_buckets = []
config = ConfigParser.ConfigParser()

def main():
    config.read(FLAGS.config_file)

    max_num_lines = int(config.get("max_data_sizes", "num_lines"))
    max_target_length = int(config.get("max_data_sizes", "max_target_length"))
    max_source_length = int(config.get("max_data_sizes", "max_source_length"))

    if not os.path.exists(FLAGS.checkpoint_dir):
        os.mkdir(FLAGS.checkpoint_dir)
    path = getCheckpointPath()
    print "path is {0}".format(path)
    data_processor = data_utils.DataProcessor(FLAGS.vocab_size,
        FLAGS.raw_data_dir,FLAGS.data_dir, FLAGS.train_frac, FLAGS.tokenizer,
        max_num_lines, max_target_length, max_source_length, FLAGS.is_discrete,
        FLAGS.extra_discrete_data)
    data_processor.run()
    #create model
    print "Creating model with..."
    print "Number of hidden layers: {0}".format(FLAGS.num_layers)
    print "Number of units per layer: {0}".format(FLAGS.hidden_size)
    print "Dropout: {0}".format(FLAGS.dropout)
    vocab_mapper = vocab_utils.VocabMapper(FLAGS.data_dir)
    vocab_size = vocab_mapper.getVocabSize()
    print "Vocab size is: {0}".format(vocab_size)
    FLAGS.vocab_size = vocab_size
    with tf.Session() as sess:
        writer = tf.train.SummaryWriter("/tmp/tb_logs_chatbot", sess.graph)
        model = createModel(sess, path, vocab_size)
        print "Using bucket sizes:"
        print _buckets
        #train model and save to checkpoint
        print "Beggining training..."
        print "Maximum number of epochs to train for: {0}".format(FLAGS.max_epoch)
        print "Batch size: {0}".format(FLAGS.batch_size)
        print "Starting learning rate: {0}".format(FLAGS.learning_rate)
        print "Learning rate decay factor: {0}".format(FLAGS.lr_decay_factor)

        source_train_file_path = data_processor.data_source_train
        target_train_file_path = data_processor.data_target_train
        source_test_file_path = data_processor.data_source_test
        target_test_file_path = data_processor.data_target_test
        print source_train_file_path
        print target_train_file_path

        train_set = readData(source_train_file_path, target_train_file_path,
            FLAGS.max_train_data_size)
            test_set = readData(source_test_file_path, target_test_file_path,
                FLAGS.max_train_data_size)

            train_bucket_sizes = [len(train_set[b]) for b in xrange(len(_buckets))]
            print "bucket sizes = {0}".format(train_bucket_sizes)
            train_total_size = float(sum(train_bucket_sizes))

            train_buckets_scale = [sum(train_bucket_sizes[:i + 1]) / train_total_size
                for i in xrange(len(train_bucket_sizes))]
            step_time, loss = 0.0, 0.0
            current_step = 0
            previous_losses = []
            while True:
```

```python
        # 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.
        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) / FLAGS.steps_per_checkpoint
        loss += step_loss / FLAGS.steps_per_checkpoint
        current_step += 1

        # Once in a while, we save checkpoint, print statistics, and run evals.
        if current_step % FLAGS.steps_per_checkpoint == 0:
            train_loss_summary = tf.Summary()
            str_summary_train_loss = train_loss_summary.value.add()
            str_summary_train_loss.simple_value = loss
            str_summary_train_loss.tag = "train_loss"
            writer.add_summary(train_loss_summary, current_step)
            # 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(path, "chatbot.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.
            perplexity_summary = tf.Summary()
            eval_loss_summary = tf.Summary()
            weights_summary = tf.Summary()
            for bucket_id in xrange(len(_buckets)):
                if len(test_set[bucket_id]) == 0:
                    print("  eval: empty bucket %d" % (bucket_id))
                    continue
                encoder_inputs, decoder_inputs, target_weights = model.get_batch(
                        test_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))
                    str_summary_ppx = perplexity_summary.value.add()
                    str_summary_ppx.simple_value = eval_ppx
                    str_summary_ppx.tag = "peplexity_bucket)%d" % bucket_id

                    str_summary_eval_loss = eval_loss_summary.value.add()
                    #need to convert from numpy.float32 to float native type
                    str_summary_eval_loss.simple_value = float(eval_loss)
                    str_summary_eval_loss.tag = "eval_loss_bucket)%d" % bucket_id

                    str_summary_weights = weights_summary.value.add()
                    str_summary_weights.simple_value = target_weights
                    str_summary_weights.tag = "target_weights_bucket)%d" % bucket_id

                    writer.add_summary(weights_summary, current_step)
                    writer.add_summary(perplexity_summary, current_step)
                    writer.add_summary(eval_loss_summary, current_step)
                sys.stdout.flush()


def createModel(session, path, vocab_size):
    model = models.chatbot.ChatbotModel(vocab_size, _buckets,
        FLAGS.hidden_size, FLAGS.dropout, FLAGS.num_layers, FLAGS.grad_clip,
        FLAGS.batch_size, FLAGS.learning_rate, FLAGS.lr_decay_factor)
    convo_limits = [config.getint("max_data_sizes", "max_source_length"),
            config.getint("max_data_sizes", "max_target_length"),
            config.getint("max_data_sizes", "num_lines")]
    hyper_params.saveHyperParameters(path, FLAGS, _buckets, convo_limits)
    print path
    ckpt = tf.train.get_checkpoint_state(path)
    if ckpt and gfile.Exists(ckpt.model_checkpoint_path):
        print "Reading model parameters from {0}".format(ckpt.model_checkpoint_path)
        model.saver.restore(session, ckpt.model_checkpoint_path)
    else:
        print "Created model with fresh parameters."
        session.run(tf.global_variables_initializer())
    return model

def setBuckets(raw_info):
    '''''
    Deserializes python dictionary of buckets
    
    Inputs
    raw_info: is the serialized string of buckets
    '''
    buckets = []
    try:
        for tu in raw_info:
            target, source = tu[1].strip().split(",")
            buckets.append((int(target), int(source)))
    except:
        print "Error in config file formatting..."
    return buckets


def readData(source_path, target_path, max_size=None):
    '''''
    This method directly from tensorflow translation example
    '''
    data_set = [[] for _ in _buckets]
    with tf.gfile.GFile(source_path, mode="r") as source_file:
        with tf.gfile.GFile(target_path, mode="r") as target_file:
            source, target = source_file.readline(), target_file.readline()
            counter = 0
            while source and target and (not max_size or counter < max_size):
                counter += 1
                if counter % 100000 == 0:
                    print("  reading data line %d" % counter)
                    sys.stdout.flush()
                source_ids = [int(x) for x in source.split()]
                target_ids = [int(x) for x in target.split()]
                target_ids.append(vocab_utils.EOS_ID)
                for bucket_id, (source_size, target_size) in enumerate(_buckets):
                    if len(source_ids) < source_size and len(target_ids) <
target_size:
                        data_set[bucket_id].append([source_ids, target_ids])
                        break
                source, target = source_file.readline(), target_file.readline()
    return data_set

def getCheckpointPath():
    '''''
    Check if new hyper params match with old ones
    if not, then create a new model in a new Directory
    Returns:
    path to checkpoint directory
    '''
    old_path = os.path.join(FLAGS.checkpoint_dir, "hyperparams.p")
    global _buckets
    if os.path.exists(old_path):
        params = hyper_params.restoreHyperParams(FLAGS.checkpoint_dir)
        num_buckets = params["num_buckets"]
        buckets = []
        for i in range(num_buckets):
            buckets.append((params["bucket_{0}_target".format(i)],
                params["bucket_{0}_target".format(i)]))
        _buckets = buckets
        ok = \
        params["num_layers"] == FLAGS.num_layers and \
        params["hidden_size"] == FLAGS.hidden_size and \
        params["dropout"] == FLAGS.dropout
        if ok:
            return FLAGS.checkpoint_dir
        else:
            _buckets = setBuckets(config.items("buckets"))
            print _buckets
            infostring = "hiddensize_{0}_dropout_{1}_numlayers_{2}".format(FLAGS.hidden_size,
            FLAGS.dropout, FLAGS.num_layers)
            if not os.path.exists("data/checkpoints/"):
                os.mkdirs("data/checkpoints/",)
            path = os.path.join("data/checkpoints/", str(int(time.time())) + infostring)
            if not os.path.exists(path):
                os.makedirs(path)
            print "hyper parameters changed, training new model at {0}".format(path)
            return path
    else:
        _buckets = setBuckets(config.items("buckets"))
        print _buckets
        return FLAGS.checkpoint_dir

if __name__ == '__main__':
    main()
```

Simple RNN - sample.py

```
'''''
Code in this file is for sampling use of chatbot
'''


import tensorflow as tf
#from tensorflow.nn.rnn import rnn, rnn_cell, seq2seq
from tensorflow.python.platform import gfile
import numpy as np
import sys
import os
import nltk
from six.moves import xrange
import models.chatbot
import util.hyperparamutils as hyper_params
```

```python
import util.vocabutils as vocab_utils
from os import listdir
from os.path import isfile, join

_buckets = []
convo_hist_limit = 1
max_source_length = 0
max_target_length = 0
#_buckets = [(10, 10), (50, 15), (100, 20), (200, 50)]

flags = tf.app.flags
FLAGS = flags.FLAGS
flags.DEFINE_string('checkpoint_dir', 'data/checkpoints/', 'Directory to store/restore checkpoints')
flags.DEFINE_string('data_dir', "data/", "Data storage directory")
flags.DEFINE_string('static_data', '', '(path to static data) Adds fuzzy matching layer on top of chatbot for better static responses')
flags.DEFINE_integer('static_temp', 60, 'number between 0 and 100. The lower the number the less likely static responses will come up')
#flags.DEFINE_string('text', 'Hello World!', 'Text to sample with.')


#Read in static data to fuzzy matcher.
#Assumes static_data has text files with discrete (source, target) pairs
#Sources are on odd lines n_i, targets are on even lines n_{i+1}
static_sources = []
static_targets = []
if FLAGS.static_data:
    if os.path.exists(FLAGS.static_data):
        try:
            from fuzzywuzzy import fuzz
            from fuzzywuzzy import process
            onlyfiles = [f for f in listdir(FLAGS.static_data) if isfile(join(FLAGS.static_data, f))]
            print(onlyfiles)
            for f in onlyfiles:
                with open(os.path.join(FLAGS.static_data, f), 'r') as f2:
                    file_lines = f2.readlines()
                    for i in range(0, len(file_lines) - 1, 2):
                        static_sources.append(file_lines[i].lower().replace('\n', ''))
                        static_targets.append(file_lines[i+1].lower().replace('\n', ''))
        except ImportError:
            print "Package fuzzywuzzy not found"
            print "Running sampling without fuzzy matching..."
    else:
        print "Fuzzy matching data not found... double check static_data path.."
        print "Not using fuzzy matching... Reverting to normal sampling"

def main():
    with tf.Session() as sess:
        model = loadModel(sess, FLAGS.checkpoint_dir)
        print _buckets
        model.batch_size = 1
        vocab = vocab_utils.VocabMapper(FLAGS.data_dir)
        sys.stdout.write(">")
        sys.stdout.flush()
        sentence = sys.stdin.readline().lower()
        conversation_history = [sentence]
        while sentence:

            use_static_match = False
            if len(static_sources) > 0:
                #static_match = process.extractOne(sentence, static_sources)
                #Check is static match is close enough to original input
                best_ratio = 0
                static_match = ""
                for s in static_sources:
                    score = fuzz.partial_ratio(sentence, s)
                    if score > best_ratio:
                        static_match = s
                        best_ratio = score
                if best_ratio > FLAGS.static_temp:
                    use_static_match = True
                    #Find corresponding target in static list, bypass neural net output
                    convo_output = static_targets[static_sources.index(static_match)]

            if not use_static_match:
                token_ids = list(reversed(vocab.tokens2Indices(" ".join(conversation_history))))
                #token_ids = list(reversed(vocab.tokens2Indices(sentence)))
                bucket_id = min([b for b in xrange(len(_buckets))
                    if _buckets[b][0] > len(token_ids)])

                encoder_inputs, decoder_inputs, target_weights = model.get_batch(
                {bucket_id: [(token_ids, [])]}, bucket_id)

                _, _, output_logits = model.step(sess, encoder_inputs, decoder_inputs,
                    target_weights, bucket_id, True)

                    #TODO implement beam search
                    outputs = [int(np.argmax(logit, axis=1)) for logit in output_logits]
                    print(outputs)

                    if vocab_utils.EOS_ID in outputs:
                        outputs = outputs[:outputs.index(vocab_utils.EOS_ID)]

                    convo_output =  " ".join(vocab.indices2Tokens(outputs))

            conversation_history.append(convo_output)
            print convo_output
            sys.stdout.write(">")
            sys.stdout.flush()
            sentence = sys.stdin.readline().lower()
            conversation_history.append(sentence)
            conversation_history = conversation_history[-convo_hist_limit:]

def loadModel(session, path):
    global _buckets
    global max_source_length
    global max_target_length
    global convo_hist_limit
    params = hyper_params.restoreHyperParams(path)
    buckets = []
    num_buckets = params["num_buckets"]
    max_source_length = params["max_source_length"]
    max_target_length = params["max_target_length"]
    convo_hist_limit = params["conversation_history"]
    for i in range(num_buckets):
        buckets.append((params["bucket_{0}_target".format(i)],
            params["bucket_{0}_target".format(i)]))
        _buckets = buckets
    print("Initializing decoder..")
    model = models.chatbot.ChatbotModel(params["vocab_size"], _buckets,
        params["hidden_size"], 1.0, params["num_layers"], params["grad_clip"],
        1, params["learning_rate"], params["lr_decay_factor"], 512, True)
    ckpt = tf.train.get_checkpoint_state(path)
    #print(ckpt)
    if ckpt: #and gfile.Exists(ckpt.model_checkpoint_path):
        print "Reading model parameters from {0}".format(ckpt.model_checkpoint_path)
        model.saver.restore(session, ckpt.model_checkpoint_path)
    else:
        print "Double check you got the checkpoint_dir right..."
        print "Model not found..."
        model = None
    return model


if __name__=="__main__":
    main()
```

Seq2seq - train.py

```
import sys
import os
import math
import time

import numpy as np

```

```python
from six.moves import xrange  # pylint: disable=redefined-builtin

import tensorflow as tf

from seq2seq_model_utils import create_model
FLAGS = tf.app.flags.FLAGS
BUCKETS = [(5, 10), (10, 15), (20, 25), (40, 50)]
from data_utils import read_data


def train():
    print("Preparing dialog data in %s" % FLAGS.data_dir)
    train_data, dev_data, _ = data_utils.prepare_dialog_data(FLAGS.data_dir, FLAGS.vocab_size)

    with tf.Session() as sess:

        # Create model.
        print("Creating %d layers of %d units." % (FLAGS.num_layers, FLAGS.size))
        model = create_model(sess, forward_only=False)

        # Read data into buckets and compute their sizes.
        print ("Reading development and training data (limit: %d)." % FLAGS.max_train_data_size)
        dev_set = read_data(dev_data)
        train_set = read_data(train_data, FLAGS.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.
          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, forward_only=False)

          step_time += (time.time() - start_time) / FLAGS.steps_per_checkpoint
          loss += step_loss / FLAGS.steps_per_checkpoint
          current_step += 1

          # Once in a while, we save checkpoint, print statistics, and run evals.
          if current_step % FLAGS.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(FLAGS.model_dir, "model.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)):
              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()
```