National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Seemanthini Narasimha Moorthy
Student ID: X18141447

School of Computing
National College of Ireland

Supervisor: Dr. Vladimir Milosavljevic

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Seemanthini Narasimha Moorthy
Student ID:	X18141447
Programme:	Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Dr. Vladimir Milosavljevic
Submission Due Date:	12/12/2019
Project Title:	Neural Text-to-Text Generation System using Generative Adversarial Network and Monte Carlo Policy Gradient
Word Count:	368
Page Count:	4

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	3rd February 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Seemanthini Narasimha Moorthy
X18141447

1 Hardware/Software Requirements

1.1 Hardware Requirements

The minimum hardware requirements for executing this implementation smoothly is listed:

Operating System	Windows 10
Ram	12GB
Harddisk	150GB

1.2 Software Requirements

Software requirements for this implementation are listed below:

Programming Language	Python 3.5
IDE	Anaconda Spyder
Web browser	Google chrome

The data preprocessing and transformation has been performed using Anaconda Spyder. Anaconda is a suite of open source applications in Python and R including Spyder, Jupyter, JupyterLab, Glueviz, Orange 3, etc. The link to download and install Anaconda currently is https://www.anaconda.com/distribution/. Anaconda requires no licence as it is open source. Once Anaconda is installed, Spyder can be used. All code files (demo, training, model) are python codes and compatible to work with Spyder.

2 Anaconda Spyder Dataset preparation

2.1 Consolidated file creation

The selected dataset has 3037 literary novels. For the case studies, the input text files will be created as two separate files for checking model performance with input text of different sizes:

1. Consolidating half the corpus into one single file
2. Consolidating the entire corpus into one single file

The required input files are placed in separate folders and input path is provided in the python file as shown in Figure 1.

```
import glob
import os

os.chdir("E:\Downloads\Gutenberg\Gutenberg_Full")
read_files = glob.glob("*.txt")

with open("gutenbergFull.txt", "w", encoding="utf8", errors='ignore') as outfile:
    for f in read_files:
        with open(f, "r",  encoding="utf8", errors='ignore') as infile:
            outfile.write(infile.read())
```

Figure 1: Consolidate File

3 Anaconda Spyder Environment Setup

This section will talk about spyder setup for the implementation. The programming has been done in python, which is compatible with Spyder. The training and model code files require following installations. This has to be executed before training and model files: Demo file can be executed once the installations are complete.

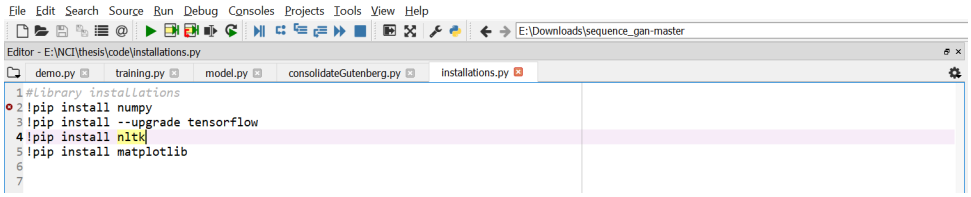

Figure 2: Installations

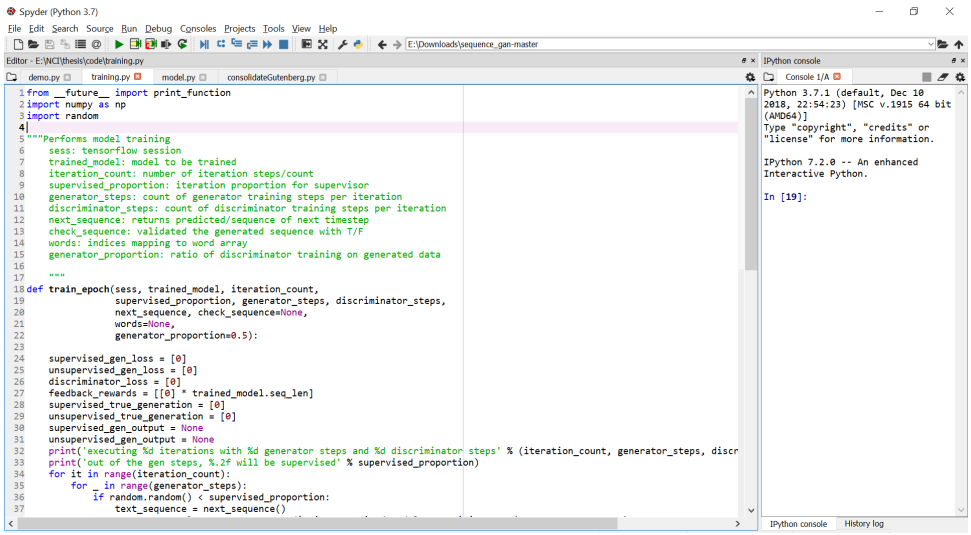

Figure 3: training code

```
print('start training')
discriminator_losses=None
supervised_gen_losses=None
unsupervised_gen_losses=None
supervised_generated_text=None
unsupervised_generated_text=None

for epoch in range(EPOCH):
    print('epoch', epoch)
    supervised_proportion = max(0.0, 1.0 - switch_rate * epoch)
    discriminator_losses,supervised_gen_losses,unsupervised_gen_losses,supervised_generated_text,unsupervised_generated_text=training.train_
        sess, trained_model, EPOCH,
        supervised_proportion=supervised_proportion,
        generator_steps=1, discriminator_steps=discriminator_steps,
        next_sequence=lambda: generate_random_sequence(tokenized_input, word2idx),
        check_sequence=lambda seq: check_sequence(three_grams, seq),
        words=words)
    discriminator_loss_list.append(discriminator_losses)
    supervised_gen_loss_list.append(supervised_gen_losses)
    unsupervised_gen_loss_list.append(unsupervised_gen_losses)
    supervised_gen_text_list.append(supervised_generated_text)
    unsupervised_gen_text_list.append(unsupervised_generated_text)
    print('discriminator loss list:',discriminator_loss_list)
    print('generator loss list (supervised):',supervised_gen_loss_list)
    print('generator loss list (unsupervised):',unsupervised_gen_loss_list)
    print('sampled supervised_generated_text',supervised_gen_text_list)
    print('sampled unsupervised_generated_text',unsupervised_gen_text_list)
```

Figure 4: Code to start of model training

```
print('data per epoch:')
dis_loss_list=None
supervised_gen_loss_list=None
unsupervised_gen_loss_list=None
supervised_gen_text=None
unsupervised_gen_text=None
print('discriminator loss:', np.mean(discriminator_loss))
dis_loss_list=np.mean(discriminator_loss)

print('generator loss (supervised: {0}, unsupervised: {1})'.format( np.mean(supervised_gen_loss), np.mean(unsupervised_gen_loss))
supervised_gen_loss_list=np.mean(supervised_gen_loss)
unsupervised_gen_loss_list=np.mean(unsupervised_gen_loss)

if check_sequence is not None:
    print('true generations (supervised:{0}, unsupervised: {1})'.format( np.mean(supervised_true_generation), np.mean(unsupervise
print('sampled generations (supervised)\n')
#storing for evaluation
supervised_gen_text=''.join([words[x] if words else x for x in supervised_gen_output]) if supervised_gen_output is not None else
print(''.join([words[x] if words else x for x in supervised_gen_output]) if supervised_gen_output is not None else None,)
#storing for evaluation
unsupervised_gen_text=''.join([words[x] if words else x for x in unsupervised_gen_output]) if unsupervised_gen_output is not None
print('sampled generations (unsupervised)\n')
print(''.join([words[x] if words else x for x in unsupervised_gen_output]) if unsupervised_gen_output is not None else None)
#print('expected rewards:', np.mean(feedback_rewards, axis=0))
return dis_loss_list,supervised_gen_loss_list,unsupervised_gen_loss_list,supervised_gen_text,unsupervised_gen_text
```

Figure 5: Supervised and unsupervised text generation code for every epoch

```
class RNN(object):

    def __init__(self, embedding_count, embedding_dimension, hidden_dimension,
                 seq_len, begin_token,
                 learningrate=0.01, feedbackreward_gamma=0.9):
        self.embedding_count = embedding_count
        self.embedding_dimension = embedding_dimension
        self.hidden_dimension = hidden_dimension
        self.seq_len = seq_len
        self.begin_token = tf.constant(begin_token, dtype=tf.int32)
        self.learningrate = tf.Variable(float(learningrate), trainable=False)
        self.feedbackreward_gamma = feedbackreward_gamma
        self.gen_parameters = []
        self.dis_parameters = []

        self.estimated_reward = tf.Variable(tf.zeros([self.seq_len]))

        with tf.variable_scope('generator'):
            self.gen_embedding = tf.Variable(self.initialise_matrix([self.embedding_count, self.embedding_dimension]))
            self.gen_parameters.append(self.gen_embedding)
            self.gen_rnn_cell = self.define_rnn_cell(self.gen_parameters)  # hidden_tml to hidden_t generator mapping
            self.gen_output_cell = self.define_output_cell(self.gen_parameters, self.gen_embedding)  # hidden_t to op_t output token logit

        with tf.variable_scope('discriminator'):
            self.discriminator_embedding = tf.Variable(self.initialise_matrix([self.embedding_count, self.embedding_dimension]))
            self.dis_parameters.append(self.discriminator_embedding)
            self.dis_rnn_cell = self.define_rnn_cell(self.dis_parameters)  # hidden_tml to hidden_t discriminator mapping
            self.discriminator_classification_cell = self.define_classification_cell(self.dis_parameters)  # hidden_t to class prediction
            self.dis_h0 = tf.Variable(self.initialise_vector([self.hidden_dimension]))
            self.dis_parameters.append(self.dis_h0)

        self.h0 = tf.placeholder(tf.float32, shape=[self.hidden_dimension])  # inception random vectorisation for generator
        self.ip = tf.placeholder(tf.int32, shape=[self.seq_len])  # index sequence of real data, excluding begin token
        self.random_sample = tf.placeholder(tf.float32, shape=[self.seq_len])
```

Figure 6: Generator and discriminator definition code for generative adversarial network

```python
class GRU(RNN):

    def define_rnn_cell(self, params):
        self.weight_rx = tf.Variable(self.initialise_matrix([self.hidden_dimension, self.embedding_dimension]))
        self.weight_zx = tf.Variable(self.initialise_matrix([self.hidden_dimension, self.embedding_dimension]))
        self.weight_hx = tf.Variable(self.initialise_matrix([self.hidden_dimension, self.embedding_dimension]))
        self.update_rh = tf.Variable(self.initialise_matrix([self.hidden_dimension, self.hidden_dimension]))
        self.unit_zh = tf.Variable(self.initialise_matrix([self.hidden_dimension, self.hidden_dimension]))
        self.unit_hh = tf.Variable(self.initialise_matrix([self.hidden_dimension, self.hidden_dimension]))
        params.extend([
            self.weight_rx, self.weight_zx, self.weight_hx,
            self.update_rh, self.unit_zh, self.unit_hh])

        def singleunit(ip_t, hidden_tml):
            ip_t = tf.reshape(ip_t, [self.embedding_dimension, 1])
            hidden_tml = tf.reshape(hidden_tml, [self.hidden_dimension, 1])
            reset_vector = tf.sigmoid(tf.matmul(self.weight_rx, ip_t) + tf.matmul(self.update_rh, hidden_tml))
            update_gate_vector = tf.sigmoid(tf.matmul(self.weight_zx, ip_t) + tf.matmul(self.unit_zh, hidden_tml))
            output_vector_tilde = tf.tanh(tf.matmul(self.weight_hx, ip_t) + tf.matmul(self.unit_hh, reset_vector * hidden_tml))
            hidden_t = (1 - update_gate_vector) * hidden_tml + update_gate_vector * output_vector_tilde
            return tf.reshape(hidden_t, [self.hidden_dimension])

        return singleunit
```

Figure 7: Gated recurrent unit code for single generator and discriminator unit

```
start training
epoch 0
executing 300 iterations with 1 generator steps and 6 discriminator steps
out of the gen steps, 1.00 will be supervised
data per epoch:
discriminator loss: 0.3251466584608563
generator loss (supervised: 3.3602543297004064, unsupervised: 0.0)
true generations (supervised:0.4717607973421927, unsupervised: 0.0)
sampled generations (supervised)

e
sampled generations (unsupervised)

None
discriminator loss list: [0.3251466584608563]
generator loss list (supervised): [3.3602543297004064]
generator loss list (unsupervised): [0.0]
sampled supervised_generated_text ['e
sampled unsupervised_generated_text [None]
epoch 1
executing 300 iterations with 1 generator steps and 6 discriminator steps
out of the gen steps, 1.00 will be supervised
data per epoch:
discriminator loss: 0.4491569399502726
generator loss (supervised: 3.142938209134479, unsupervised: 0.0)
true generations (supervised:0.8205980066445183, unsupervised: 0.0)
sampled generations (supervised)


sampled generations (unsupervised)

None
discriminator loss list: [0.3251466584608563, 0.4491569399502726]
generator loss list (supervised): [3.3602543297004064, 3.142938209134479]
generator loss list (unsupervised): [0.0, 0.0]
sampled supervised_generated_text ['e
']
```

Figure 8: Output starting from epoch zero

```
epoch 8
executing 300 iterations with 1 generator steps and 6 discriminator steps
out of the gen steps, 0.99 will be supervised
data per epoch:
discriminator loss: 0.2613807487932925
generator loss (supervised: 2.939278209209442, unsupervised: 0.1365739107131958)
true generations (supervised:0.9966666666666667, unsupervised: 0.0)
sampled generations (supervised)


sampled generations (unsupervised)

 tio lach ldtteoby lfbo  eatl hewa ri samoifi jieivfeqleeh  cp oct iscteuamsblee pster r ltpotrooeb-
discriminator loss list: [0.3251466584608563, 0.4491569399502726, 0.5052016249473628, 0.5613042149715527, 0.5284354808196831, 0.5587136841346528,
0.4961589222141996, 0.4024407434329465, 0.2613807487932925]
generator loss list (supervised): [3.3602543297004064, 3.142938209134479, 2.953592648141804, 2.9469801722174864, 2.9792084189869414, 2.9610719999740756,
2.934257414029992, 2.9308796024005677, 2.939278209209442]
generator loss list (unsupervised): [0.0, 0.0, 0.0, 0.0, 0.8126889020204544, 0.7302941481272379, 1.4737086296081543, 0.0, 0.1365739107131958]
sampled supervised_generated_text ['e                                                                              ', '
', '                                                                              ', '
', '                                                                              ', '
', '                                                                              ', '
', '                                                                              ']
sampled unsupervised_generated_text [None, None, None, None, 'e ioeldntnetoe tefpe,thogiptal4slnierlmcethssmeeeom ke nkyotw  eb fe tsxnef_BEGIN
r.torbddeoefhlm tet  h ', "nvsoo ei   n di.iel ,ot _oewng datv lnrpenw'ftfwecuwtv on 5eadisj ddifp i oepr8t esi; nmicc hdtr,pir", 'b aal ilpro yuoot"o
yih akoh,,ieiieaii lha fe , eei,wewt s  te o nmiwoir,  t theox"semdoemjhitn nthc', None, ' tio lach ldtteoby lfbo  eatl hewa ri samoifi jieivfeqleeh  cp
oct iscteuamsblee pster r ltpotrooeb-']
```

Figure 9: Developing epoch outputs through progression