milestone3

May 20, 2021

1 Milestone 3

Stephanie Eordanidis. Ravjot Sachdev, Jackson Taber
Syracuse University : College of Engineering & Computer Science
223 Link Hall, Syracuse, NY 13244
sleordan@syr.edu, rssachde@syr.edu, jrtaber@syr.edu
CIS 700 Machine Learning and Security
06/16/2021

1.1 Theme:

"Adversarial Text Generation: Adversarial Machine Learning Applications in Text Analysis"

1.2 Purpose:

The purpose of this lab is to add three new GAN metrics to the project space and successfully run them on the chosen GAN models from previous milestones.

1.3 Project:

Texygen is the name of the project selected. This project is a benchmarking tool that aids in text generation model research and testing. This tool allows for ease of various model testing to compare accuracy and synthetic data generation of models using same training baseline.

1.4 (Hard/Soft)ware:

Google Colaboratory https://colab.research.google.com/
GPU Python 3 Google Compute Engine backend
Github https://github.com/eordanis/CIS-700/

1.5 Resources:

Original Source: https://github.com/geek-ai/Texygen
Modified Sources Acquired: 2SU Course Files Section -> Texygenmaster_Python_3.6.zip

1.6 Data:

The data for the selected project is setup as follows:
Generated data training: 5000 word and 20 sentence count
Oracle data generation: 10,000 sentence generation
Real data training:

- image_coco : 20,000 sentences chosen from the image
- COCO captions data. 10,000 of which are used as training set while other 10,000 used as test set
- emlp_news_min : 20,000 sentences
- A minified version of emlp_news : reduced from 278,586 lines of text in training data to 10,000 and also 10,000 for test. Trying to run real training on 1/4 of a million lines proved too taxing even on colab pro. minified version should yeild decent results comparable to image_coco, the project default.
- eapoe : 266 sentences
- eapoe : 266 sentences Compiled from various Edgar Allan Po Poems found on referenced poem sight [4].

1.7 Modifications:

To begin, the modified source code acquired from the 2SU application was further modified to combine the original intention of the origin source authors as well as professor modification. If no arguments are passed, all models/data are run. > **Note**: The order or model generation is done by first iterating over the GAN model type, then data type. If arguments are passed, those arguments will be validated and run accordingly to run a more targeted model test.

Modifications were applied to eliminate much library warnings and informational messages as to keep output as clean as possible. File path naming was updated to be compliant with Google Colaboratory environment. All epoch time elapse console printing has been commented out for cleaner output reading.

Modifications under previous course works: * Addition of three new GAN models * CGAN * InfoGAN * DCGAN

- Fixed LeakGAN project to work and run properly by setting standardized default flag attributes in main.py
- Update models to save off test files for each training type run (ie oracle, cfg, real)
- Updated models to use unified naming schema for test files, with model as the model the file was generated for and training as the training the data was generated from

  Example:

  - experiment-log-model-training.csv
  - oracle-model-training.txt
  - generator-model-training.txt
  - test_file-model-training.txt

- Updated models to set file name in main.py on the GAN model directly instead of in each model file themselves

- Added additional field called 'log_file' to be the name for the experiment-log file data
- Update to add new dir for midterm content vs milestone project
- Added new util called visual.py to handle data representation
- Allow visual.py to take directory param. if none exist, default to /content/CIS-700_clone/results
- Updated visual.py to grab files from directory and generate visual data representations from relevant files in a more automated and less hard coded way
- Allow main.py to take a results output directory via arg -o, if arg not present sets to results/
- Allow main.py to take value for epoch via -p arg to use in model training
- For argument provided training, added more detailed messaging as well as metric grid showing metrics that will be used during model generation
- Cleaned main.py and model files to limit print to console, additionally console print is uniform and consistent across.
- Added time elapsed pring statement for model run.

Modifications under this work: * Addition of three new metrics: * Metric 1 * Metric 2 * Metric 3

1.8 Setup:

Due to the heft of processor/gpu usage, it was deemed necessary to run the project in the Google Colaboratory. Original attempt to run was done via Pycharm IDE Professional Edition with Anaconda derived environments, however this proved too great of a strain on the accessible workstation.

Additionally, it is important to note if timeout is experienced, it is possible to run a ClickConnect script via inspector tools to prevent timeout while running long codes. The following code worked in chrome when inserted in the developer console at time of test:

```
let myClick = function ClickConnect({
console.log('Working - Preventing Timeout');
document.querySelector('colab-connect-button').shadowRoot.getElementById('connect').click();
}
setInterval(myClick,60000);
```

1.8.1 Step 1

A new Google Colaboratory workspace was setup, titled "Milestone2". This workspace was run using the hosted runtime environment. This document is the current document being read.

In order to run against provided code base, it was necessary to sync the colab workspace the github repository files as follows

```
!git clone https://$GITHUB_AUTH@github.com/eordanis/CIS-700_clone/
```

Running this command from the first cell in the workbook syncs the drive to the github repo location of project location, as well as change to the necessary directory

[ ]:
```
import shutil
# to prevent nesting problems, remove directory and its contents if exists
dir_path = '/content/CIS-700_clone'
try:
    shutil.rmtree(dir_path)
except OSError as e:
    print("Error: %s : %s" % (dir_path, e.strerror))
```

[1]:
```
!git clone https://$GITHUB_AUTH@github.com/eordanis/CIS-700_clone/
%cd CIS-700_clone
```

```
Cloning into 'CIS-700_clone'...
remote: Enumerating objects: 220, done.
remote: Counting objects: 100% (220/220), done.
remote: Compressing objects: 100% (175/175), done.
remote: Total 220 (delta 40), reused 220 (delta 40), pack-reused 0
Receiving objects: 100% (220/220), 31.72 MiB | 19.73 MiB/s, done.
Resolving deltas: 100% (40/40), done.
/content/CIS-700_clone
```

1.8.2 Step 2

Now it was necessary to import and download any missing libraries the hosted colaborartoy runtime did not have readily available via the following commands:

```
!pip install -r "requirements.txt"
import nltk
nltk.download('punkt')
```

Running this command from the next cell in the workbook installed the necessary libraries and at specified versions for the project.

[ ]:
```
!pip install -r "requirements.txt"
import nltk
nltk.download('punkt')
```

1.8.3 Step 3

Now it is time to run the application. Below are two examples of commands to run the application.

```
!python3 "main.py"
```

This first command was run without parameters. In the case of this command, all trainings (SeqGAN, Gsgan, TextganMmd, Leakgan, Rankgan, Maligan, Mle) were run on all available defaulted training data (oracle LSTM, real data, CFG). Running this command can take around 2+ hours to complete.

```
!python3 "main.py" -g seqgan -t real
```

This second command was run with parameters. In the case of this command, main was run with SeqGAN training on image_coco. Running targeted trainings take less time to run, on average completing in 5-15 minutes depending on selected parameters. With the above selection, runtime was run above 10 minutes.

```
!python3 "main.py" -g seqgan -t real  -d data/eapoe.txt
```

This third command was run with parameters. In the case of this command, main was run with SeqGAN training on eapoe.

```
!python3 "main.py" -g seqgan -t real  -d data/eapoe.txt -o results/test/
```

This third command was run with parameters. In the case of this command, main was run with SeqGAN training on eapoe and results will output to results/test directory.

```
!python3 "main.py" -g seqgan -t real  -d data/eapoe.txt -o results/test/ -p 45
```

This fourth command was run with parameters. In the case of this command, main was run with SeqGAN training on eapoe and results will output to results/test directory and run on 45 epoch for both pre and adversarial training .

Running targeted trainings take less time to run, on average completing in 5-15 minutes depending on selected parameters. With the above selection, runtime was run above 10 minutes.

> **NOTE:** For above estimates, based around 5 epochs. Additionally, CFG training appears to have stopped working suddenly, unsure why broken. Therefore running without that option for the time being. Additionally, the LeakGan model failed entirely to run now due to flag errors, so this model was discarded from testing.

1.9 Overview

1.9.1 Process

When running the various models, there are similar steps for each. 1. Beginning Training – begin model training(s) 2. Set training - sets the desired model training method 3. Start model pre-train generator – uses the training data to pre-train the generator model 4. Start model pre-train discriminator - – uses the training data to pre-train the discriminator model 5. Model adversarial training – runs the model to generate results based on the test data and metrics applied 6. Finish Training – end of model training(s)

During training, each model training runs through several passes or epochs. For simplicity, base epoch is set to 5, with model training running thrice for 15 total epochs there abouts for each model trained on a particular data set.

1.9.2 Baseline Models

For this report, the TextGAN and SeqGAN models were run on oracle and real training types in the previous project milestone and will be used as the baseline for new model comparisons. The real training types essentially runs the data against the image_coco.txt caption data. The TextGAN and SeqGAN was developed by the source project team to improve on existing GAN networks.

With regards to TextGan, the goal of this model is to generate high quality realistic synthetic data while overcoming the convergence dilemma by using a generator that runs as a long short-term memory network and its discriminator a convolutional network. By matching high-dimension laten feature distributions of the testing and training data, this model over longer epochs has shown demonstrate a higher performance in quantitative evaluation, showing the TextGAN model can produce sentences that appear to have been written by a human, and not AI generated.

For the SeqGAN model, this also proved successful in generating realistic looking sentences via this generator process. A second model was selected for comparison purposes. SeqGAN's generator is based off the reinforcement learning stochastic policy, allowing SeqGAN to performing gradient policy update in order to circumvent differentiation issues in the generation. Its discriminator is run on complete sentences, and its results used as the reinforcement learning reward signal. According to source authors, this model boasted higher performance over others run.

1.9.3 New Models From Previous Milestone

1.9.4 CGAN - Conditional Generative Adversarial Network

Conditional adversarial network, or CGAN for short, is a basic modification of GAN that simply adds an additional layer that conditions both the descriminator and generator model layers. For this labs purposes, an existing GAN model was used as the base, and in both the Generator and Discriminator an embedded layer was incorporated as the first layer, used as the conditioning layer. For the sake of specificity, this GAN could actually be considered an CGSGAN, however for simplicity we will refer to is as simply CGAN.

1.9.5 INFOGAN

InfoGan is an adjusted simple version of GAN that seeks to maximize the mutual information of a fixed subset of GAN noise varaibles. It is able to achieve this by having numerous convolution layers added to a regular implemtation of a GAN that are connected at the end, this makes the additional cost of computation low. For the use of this lab an existing GAN was used as the base and the convolution layers were added and connected at the end.

1.9.6 DCGAN

Deep convolutional generative adversarial networks (DCGANs) are a class of convolutional networks (CNNs) aimed to incorporate unsupervised learning. Some key components of the DCGAN architecture are the use of the Tanh activation function for the generator's output layer, LeakyReLU activation in the discriminator, and the removal of any fully connected hidden layers. With these features, the original study conducted was able to create a robust DCGAN, achieving an 82.8% accuracy on the on the CIFAR-10 image dataset.

1.9.7 Metrics

Abbreviations: * BLEU - BiLingual Evaluation Understudy * GAN - Generative Adversarial Network * NLL - Negative Log-Likelihood * RL - Reinforcement Learning

Definitions * EmbSim – influenced by BLEU, used instead to compare the word embeddings vs BLEU's comparison of word similarity between two sentences or documents. * NLL-oracle :

applied to synthetic data to determine fitting via oracle language model standards. * NLL-test : dual to NLL-Oracle, used to determine a model's capability to fit to real test data

These measurement standards and more are discussed in the project directory's "/docs/evaluation.md" location.

1.9.8 New Metrics

Metric 1 -

Metric 2 -

Metric 3 -

1.10 Test

For this project, we will use the following arguments

```
#directory to use for model results output
directory = '/content/CIS-700_clone/results/midterm/'

#data set location to use
data = '/content/CIS-700_clone/data/eapoe.txt'

#epoch values to run for pre and adversarial training
epoch = '5'
```

Epochs were increased left to run at 5 for the sake of time. However, it is noted that as according to original project sourcing, >= 45 epochs for the models display the best NLL loss results on epochs > 40, prior to that point results would be poorer. NLL loss values are indicated to be better the lower they are, so if these values trend downward, the models are improving. For EmbeddedSimilarity, higher values are desired for better results.

[4]:
```
#data set location to use
data = '/content/CIS-700_clone/data/eapoe.txt'

# epoch values to run for pre and adversarial training
epoch = '5'
```

1.10.1 SeqGAN

The following commands are to run SeqGAN model on both oracle and real trainings.

NOTE: The real data essentially trained the model on the eapoe.txt data.

```
!python3 "main.py" -g seqgan -t oracle -d $data -p epoch
!python3 "main.py" -g seqgan -t real -d $data -p epoch
```

Oracle Training

[ ]:
```
!python3 "main.py" -g seqgan -t oracle -d $data -p epoch
```

For below model training and test, results are discussed later.
For more details, see Section **??**.

Real Training

[ ]:
```
!python3 "main.py" -g seqgan -t real -d $data -p epoch
```

For below model training and test, results are discussed later.
For more details, see Section **??**.

1.10.2 TextGAN

The following commands are to run TextGAN model on both oracle and real trainings.
NOTE: The real data essentially trained the model on the eapoe.txt data.

```
!python3 "main.py" -g textGan -t oracle -d $data -p epoch
!python3 "main.py" -g textGan -t real -d $data -p epoch
```

Oracle Training

[ ]:
```
!python3 "main.py" -g textGan -t oracle -d $data -p epoch
```

For below model training and test, results are discussed later.
For more details, see Section **??**.

Real Training

[ ]:
```
!python3 "main.py" -g textGan -t real -d $data -p epoch
```

For below model training and test, results are discussed later.
For more details, see Section **??**.

1.10.3 CGAN

The following commands are to run CGAN model on both oracle and real trainings.
NOTE: The real data essentially trained the model on the eapoe.txt data.

```
!python3 "main.py" -g cgan -t oracle -d $data -p epoch
!python3 "main.py" -g cgan -t real -d $data -p epoch
```

Oracle Training

[ ]:
```
!python3 "main.py" -g cgan -t oracle -d $data -p epoch
```

For below model training and test, results are discussed later.
For more details, see Section **??**.

Real Training

[ ]:
```
!python3 "main.py" -g cgan -t real -d $data -p epoch
```

For below model training and test, results are discussed later.
For more details, see Section **??**.

1.10.4 INFOGAN

The following commands are to run INFOGAN model on both oracle and real trainings.

NOTE: The real data essentially trained the model on the eapoe.txt data.

```
!python3 "main.py" -g infogan -t oracle -d $data -p epoch
!python3 "main.py" -g infogan -t real -d $data -p epoch
```

Oracle Training

[ ]:
```
!python3 "main.py" -g infogan -t oracle -d $data
```

```
******** Beginning Training ********
set training
oracle
start pre-train generator:
nll-oracle
nll-test
EmbeddingSimilarity
epoch:1 nll-oracle:10.742874    nll-test:7.5933557
EmbeddingSimilarity:-0.20757322466063718
start pre-train discriminator:
adversarial training:
nll-oracle
nll-test
EmbeddingSimilarity
epoch:6 nll-oracle:10.20413     nll-test:7.031268
EmbeddingSimilarity:-0.20769521678846775
nll-oracle
nll-test
EmbeddingSimilarity
epoch:10        nll-oracle:10.124441    nll-test:7.0572867
EmbeddingSimilarity:-0.20052312845326598
******** Completed Training ********
```

For below model training and test, results are discussed later.
For more details, see Section **??**.

Real Training

[ ]:
```
!python3 "main.py" -g infogan -t real -d $data
```

```
******** Beginning Training ********
set training
real
start pre-train generator:
EmbeddingSimilarity
nll-test
epoch:1 EmbeddingSimilarity:-1.1401053379158488 nll-test:6.035058
start pre-train discriminator:
adversarial training:
112.52452
EmbeddingSimilarity
nll-test
epoch:6 EmbeddingSimilarity:-0.7430809801654519 nll-test:3.865037
33.357517
68.59142
68.35573
97.11515
EmbeddingSimilarity
nll-test
epoch:10        EmbeddingSimilarity:-0.7305529035011096 nll-test:4.139575
******** Completed Training ********
```

For below model training and test, results are discussed later.
For more details, see Section **??**.

1.10.5 DCGAN

The following commands are to run DCGAN model on both oracle and real trainings.

NOTE: The real data essentially trained the model on the eapoe.txt data.

```
!python3 "main.py" -g dcgan -t oracle -d $data -p epoch
!python3 "main.py" -g dcgan -t real -d $data -p epoch
```

Oracle Training

[ ]:
```
!python3 "main.py" -g dcgan -t oracle -d $data -p epoch
```

For below model training and test, results are discussed later.
For more details, see Section **??**.

Real Training

[ ]:
```
!python3 "main.py" -g dcgan -t real -d $data -p epoch
```

For below model training and test, results are discussed later.
For more details, see Section **??**.