

---

## TextGAN-PyTorch

TextGAN is a PyTorch framework for Generative Adversarial Networks (GANs) based text generation models, including general text generation models and category text generation models. TextGAN serves as a benchmarking platform to support research on GAN-based text generation models. Since most GAN-based text generation models are implemented by Tensorflow, TextGAN can help those who get used to PyTorch to enter the text generation field faster.

If you find any mistake in my implementation, please let me know! Also, please feel free to contribute to this repository if you want to add other models.

license MIT contributions welcome

### Requirements

- **PyTorch** **>= 1.1.0**
- Python 3.6
- Numpy 1.14.5
- CUDA 7.5+ (For GPU)
- nltk 3.4
- tqdm 4.32.1
- KenLM (<https://github.com/kpu/kenlm>)

To install, run `pip install -r requirements.txt`. In case of CUDA problems, consult the official PyTorch Get Started guide.

### KenLM Installation

- Download stable release and unzip: <http://kheafield.com/code/kenlm.tar.gz>
- Need Boost **>= 1.42.0** and bjam
  - Ubuntu: `sudo apt-get install libboost-all-dev`
  - Mac: `brew install boost; brew install bjam`
- Run *within* kenlm directory:

```
1 mkdir -p build
2 cd build
3 cmake ..
4 make -j 4
```

- 
- `pip install https://github.com/kpu/kenlm/archive/master.zip`
  - For more information on KenLM see: <https://github.com/kpu/kenlm> and <http://kheafield.com/code/kenlm/>

## Implemented Models and Original Papers

### General Text Generation

- **SeqGAN** - SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient
- **LeakGAN** - Long Text Generation via Adversarial Training with Leaked Information
- **MaliGAN** - Maximum-Likelihood Augmented Discrete Generative Adversarial Networks
- **JSDGAN** - Adversarial Discrete Sequence Generation without Explicit Neural Networks as Discriminators
- **RelGAN** - RelGAN: Relational Generative Adversarial Networks for Text Generation
- **DPGAN** - DP-GAN: Diversity-Promoting Generative Adversarial Network for Generating Informative and Diversified Text
- **DGSAN** - DGSAN: Discrete Generative Self-Adversarial Network
- **CoT** - CoT: Cooperative Training for Generative Modeling of Discrete Data

### Category Text Generation

- **SentiGAN** - SentiGAN: Generating Sentimental Texts via Mixture Adversarial Networks
- **CatGAN** (ours) - CatGAN: Category-aware Generative Adversarial Networks with Hierarchical Evolutionary Learning for Category Text Generation

## Get Started

- Get Started

```
1 git clone https://github.com/williamSYSU/TextGAN-PyTorch.git
2 cd TextGAN-PyTorch
```

- For real data experiments, all datasets ([Image COCO](#), [EMNLP NEWS](#), [Movie Review](#), [Amazon Review](#)) can be downloaded from [here](#).
- Run with a specific model

```
1 cd run
2 python3 run_[model_name].py 0 0 # The first 0 is job_id, the second 0
  is gpu_id
3
```

---

```
4 # For example
5 python3 run_seqgan.py 0 0
```

## Features

### 1. Instructor

For each model, the entire running process is defined in `instructor/oracle_data/seqgan_instructor.py`. (Take SeqGAN in Synthetic data experiment for example). Some basic functions like `init_model()` and `optimize()` are defined in the base class `BasicInstructor` in `instructor.py`. If you want to add a new GAN-based text generation model, please create a new instructor under `instructor/oracle_data` and define the training process for the model.

### 2. Visualization

Use `utils/visualization.py` to visualize the log file, including model loss and metrics scores. Custom your log files in `log_file_list`, no more than `len(color_list)`. The log filename should exclude `.txt`.

### 3. Logging

The TextGAN-PyTorch use the `logging` module in Python to record the running process, like generator's loss and metric scores. For the convenience of visualization, there would be two same log file saved in `log/log_****_****.txt` and `save/**/log.txt` respectively. Furthermore, The code would automatically save the state dict of models and a batch-size of generator's samples in `./save/**/models` and `./save/**/samples` per log step, where `**` depends on your hyper-parameters.

### 4. Running Signal

You can easily control the training process with the class `Signal` (please refer to `utils/helpers.py`) based on dictionary file `run_signal.txt`.

For using the `Signal`, just edit the local file `run_signal.txt` and set `pre_sig` to `False` for example, the program will stop pre-training process and step into next training phase. It is convenient to early stop the training if you think the current training is enough.

### 5. Automaticly select GPU

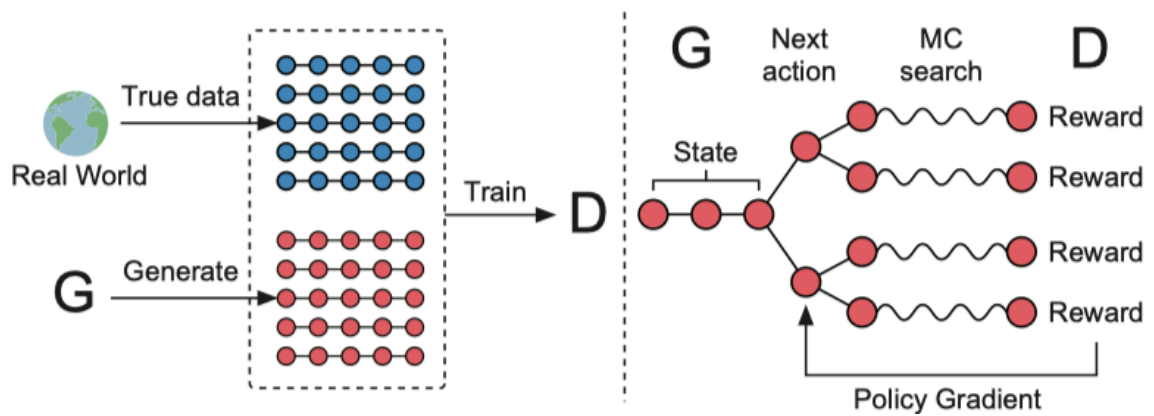
In `config.py`, the program would automatically select a GPU device with the least `GPU-Util` in `nvidia-smi`. This feature is enabled by default. If you want to manually select a GPU device, please uncomment the `--device` args in `run_[run_model].py` and specify a GPU device with command.

---

## Implementation Details

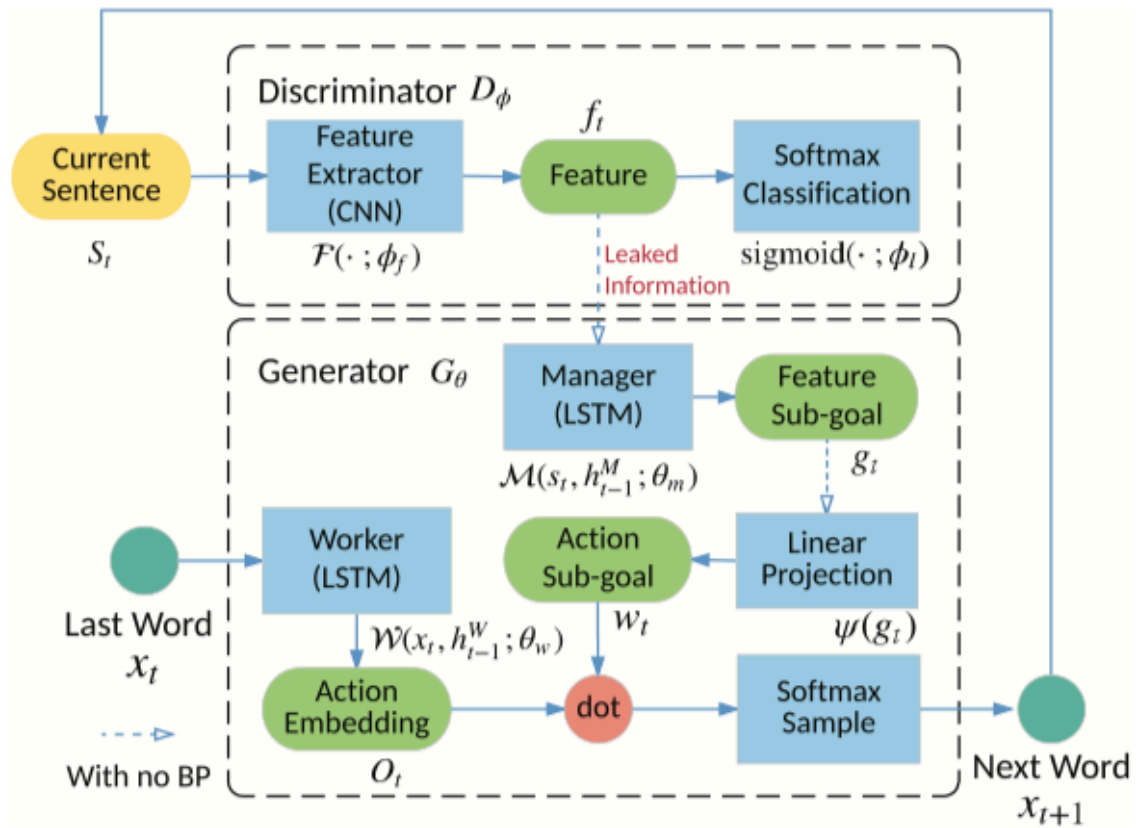
### SeqGAN

- run file: run\_seqgan.py
- Instructors: oracle\_data, real\_data
- Models: generator, discriminator
- Structure (from SeqGAN)



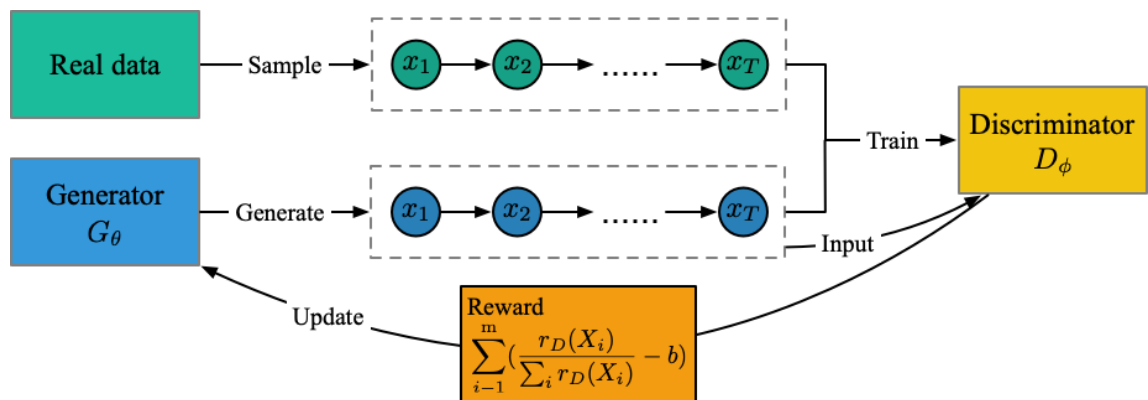
### LeakGAN

- run file: run\_leakgan.py
- Instructors: oracle\_data, real\_data
- Models: generator, discriminator
- Structure (from LeakGAN)



## Maligan

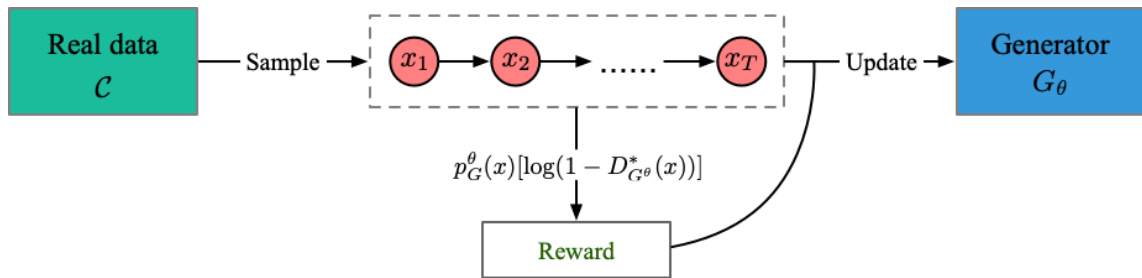
- run file: run\_maligan.py
- Instructors: oracle\_data, real\_data
- Models: generator, discriminator
- Structure (from my understanding)



---

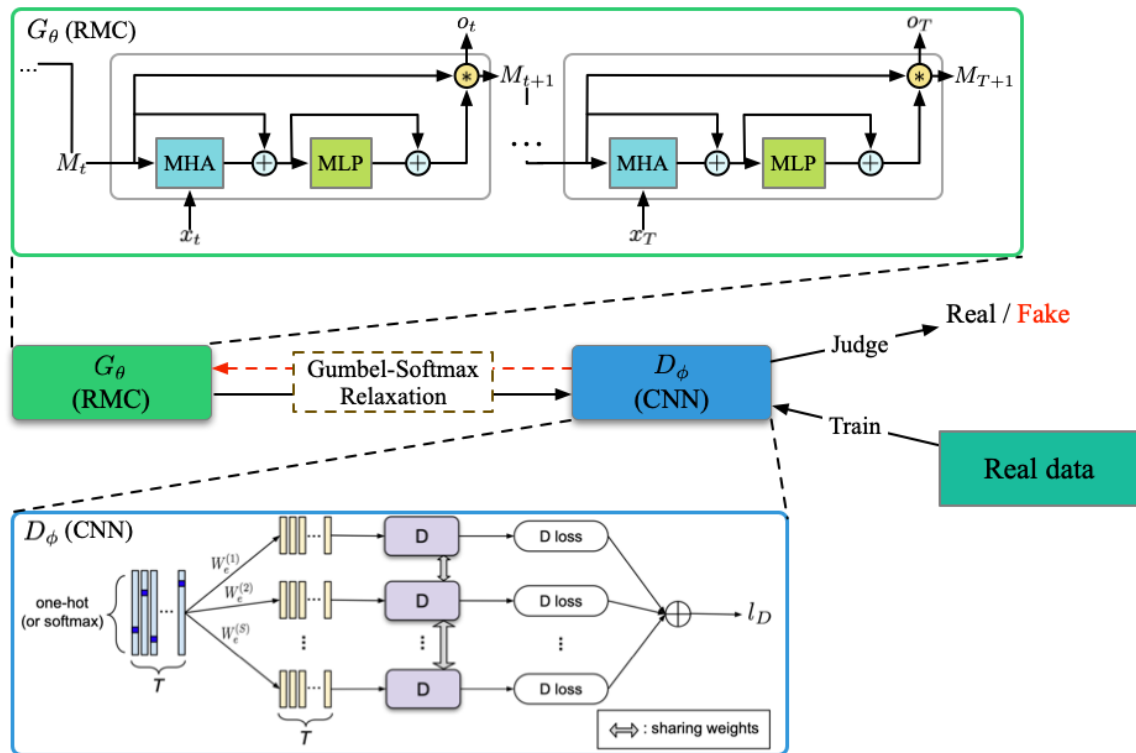
## JSDGAN

- run file: run\_jsdgan.py
- Instructors: oracle\_data, real\_data
- Models: generator (No discriminator)
- Structure (from my understanding)



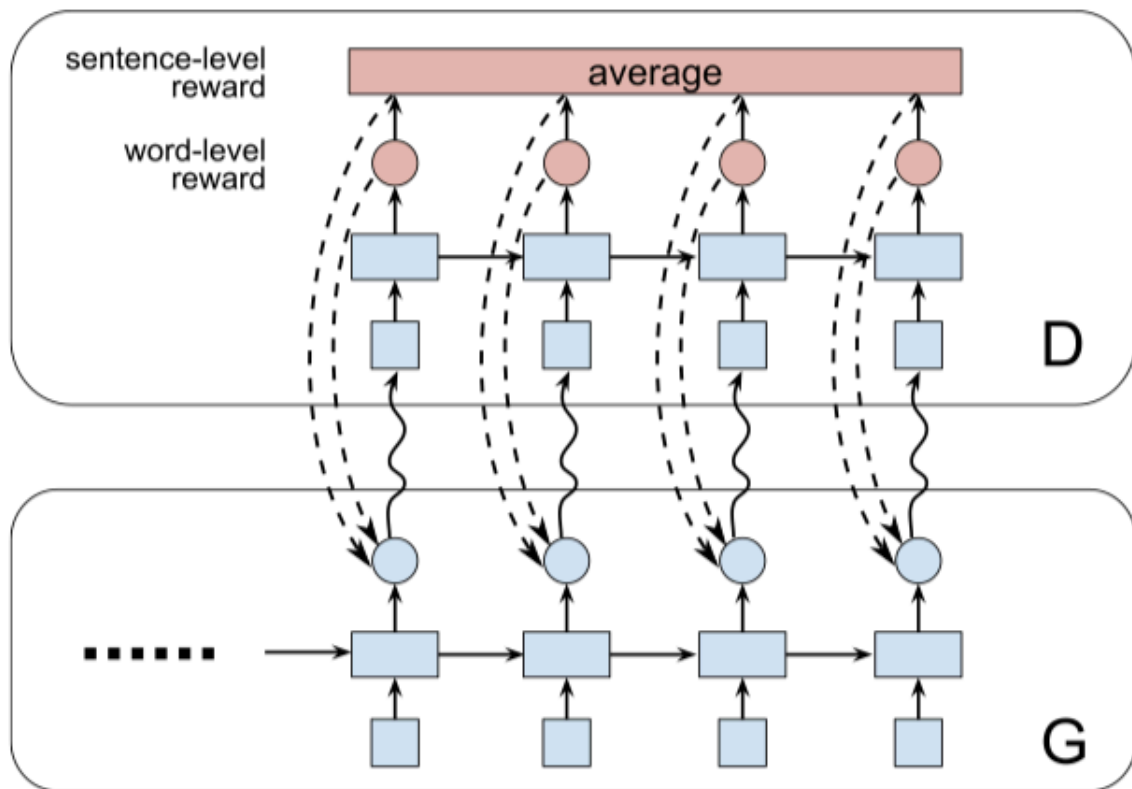
## RelGAN

- run file: run\_relgan.py
- Instructors: oracle\_data, real\_data
- Models: generator, discriminator
- Structure (from my understanding)



## DPGAN

- run file: run\_dpgan.py
- Instructors: oracle\_data, real\_data
- Models: generator, discriminator
- Structure (from DPGAN)



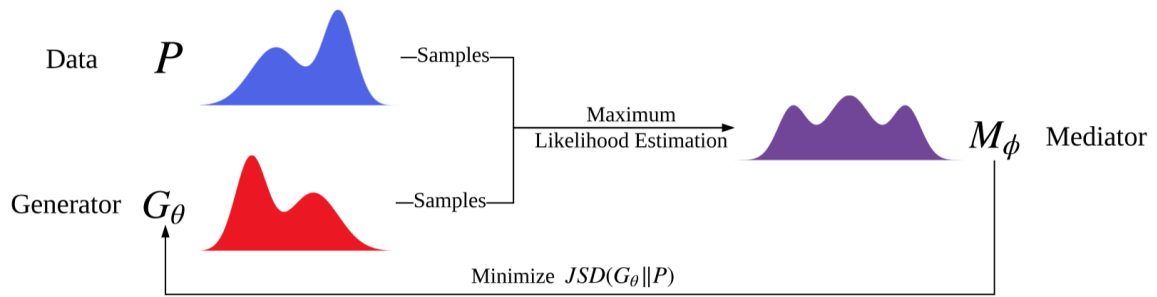
### DGSAN

- run file: run\_dgsan.py
- Instructors: oracle\_data, real\_data
- Models: generator, discriminator

### CoT

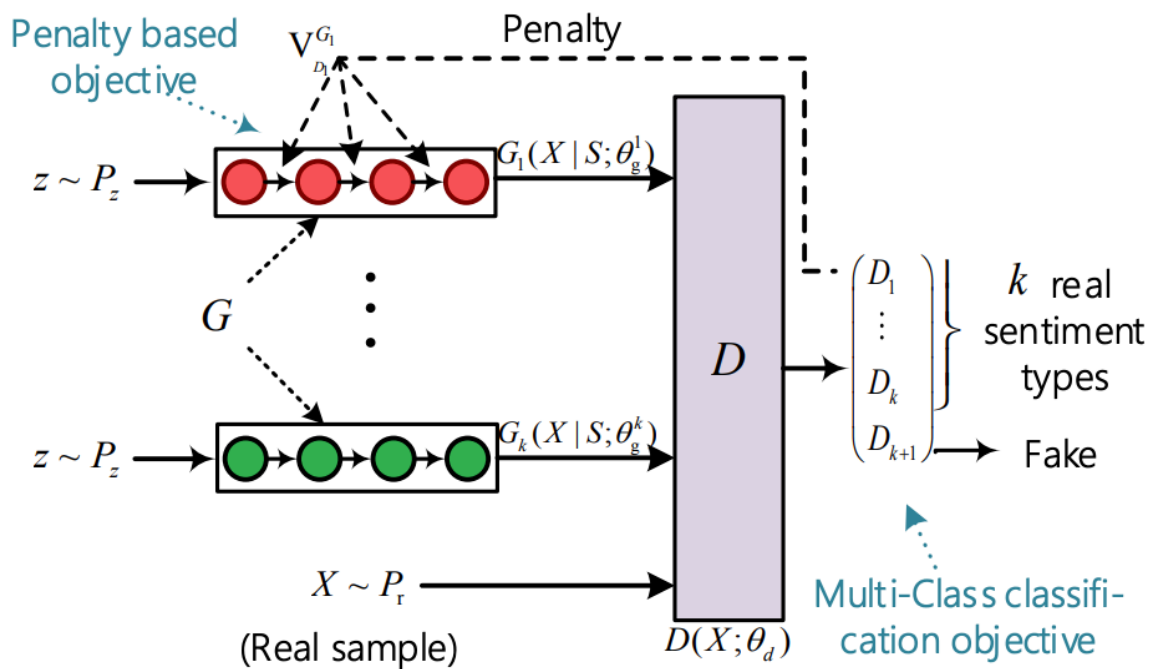
- run file: run\_cot.py
- Instructors: oracle\_data, real\_data
- Models: generator, discriminator
- Structure (from CoT)





### SentiGAN

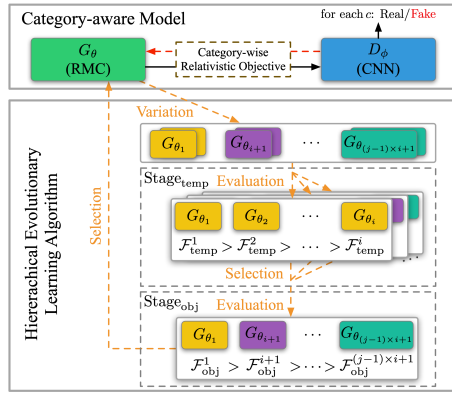
- run file: run\_sentigan.py
- Instructors: oracle\_data, real\_data
- Models: generator, discriminator
- Structure (from SentiGAN)



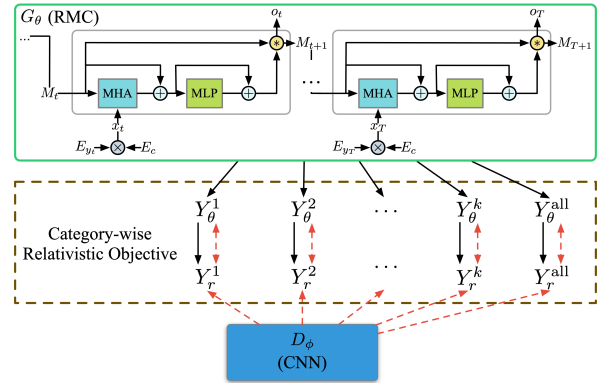
### CatGAN

- run file: run\_catgan.py
- Instructors: oracle\_data, real\_data
- Models: generator, discriminator

- Structure (from CatGAN)



(a) CatGAN



(b) Category-aware Model

**Licence**

**MIT lincense**