Differential Privacy Synethic Data Generation using WGAN (CODE GUIDE)

Team: `UCLANESL`
Mani Srivastava, Moustafa Alzantot[*]

May 1st, 2019

This document provides a brief documentation of the code implementation of the UCLANESL team submission o the NIST differential privacy challenge (match 3).

The main parts of our code folder are the following

- `run_uclanesl.sh`: Entry point bash script to run our model training and data generation python program.
- `tf_gan.py`: This is main file for our model training and generation of synthetic data.
- `data_utils.py`: This file has the data pre-processing and post-processing steps.
- `differential_privacy/dp_sgd`: This folder has the code for the differential privacy optimizer.
- `differential_privacy/privacy_accountant:` This folder has the code for the differential privacy accountant that we use to track the level of privacy (ϵ, δ) spent in model training.

Running using Nvidia-Docker

Our implementation uses the `TensorFlow` open-source deep learning framework which is more efficiently run using GPU enabled machines using. It can be also run in a container using `NvidiaDocker`. The runtime is signficantlly improved when running on a GPU- powered machine.

Steps to install `NvidiaDocker` can be found in `https://github.com/NVIDIA/nvidia-docker`

After unziping the given zip file. First, build the docker image.

[*]Corresponding Author: Moustafa Alzantot (`malzantot@ucla.edu`), TopCoder handle: `m_alzantot`

```
docker build -t uclanesl_wgan .
```

then, run the image while giving the appropriate argument values.

then, run the image while giving the appropriate argument values.

```
docker run --runtime=nvidia -v $(pwd):/x uclanesl_wgan /
    x/data.csv /x/output.csv /x/data-specs.json  epsilon
    [delta]
```

where

data.csv: input dataset file.

output.csv: output file.

data-specs.json: metadata file.

epsilon ϵ privacy budget.

delta: **(optional)** value of δ parameter. If it is not given, then δ is chosen to be $\frac{1}{n^2}$ where n is the size of private dataset.

`run_uclanesl.sh`

This is the entry point bash script to run our model training and data generation python script. It can be called using the following pattern:

```
bash run_uclanesl.sh data.csv output.csv data-specs.json
    epsilon [delta]
```

where the arguments are as follows:

- **data.csv**: Input data CSV file.
- **output.csv**: Output file name to store the synthetic dataset.
- **data-specs.json**: Metadata JSON file that holds information about the range for values in each column.
- **epsilon**: The epsilon level of privacy.
- **delta (optional)**: An optional parameter to specify the delta parameter of privacy. If delta parameter is not provided then the default value will be $\frac{1}{n^2}$ where n is the number of rows in input data.

`tf_gan.py`

This is main file for our model training and generation of synthetic data.

It is using TensorFlow neural network library to build and train two neural network models a *generator* and *critic*. Only the critic gets to see the input data directly, and the training of critic is made differentially private by applying Gaussian mechanism which is achieved by limiting the L_2 sensitivity of training gradients and sanitizing them by adding Gaussian noise. The generator is trained based on feedback it receives from the critic model. Therefore, it will be also differentially private by virtue of the differential privacy post-processing in variance.

The functions : `lrelu`, `fully_connected` define two common building blocks for constructing neural networks.

the functions: `critic_f`, and `generator` define the architectures of the *critic* and *generator*, models respectively.

the two functions: `nist_data_format`, and `nist_data_format` applies necessary formatting to the `generator` model outputs to match the format of the IPUMS datasets. They don't require access to input data but only access the metadata (from JSON specificiations file), and the column maps indicates the type of pre-processing was done during the pre-processing step.

The differential privacy is applied in this file by using:

- The critic optimizer is instantiated as an instance of `dp_optimizer.DPGradientDescentOptimizer` which uses the `gaussian_santizier` (an instance of **sanitizer.AmortizedGaussianSanitizer**) to clip and sanitize the training gradients by adding gaussian noise to them.

- A privacy accountant (variable = `priv_accountant`) is used to track how much privacy have been spent during the training of the critic model. Once the level of spent privacy exceeds the maximum allowed (ϵ, δ) values, then the training is aborted early and no further updates are made to the *generator*. (**LINE 387**).

`data_utils.py`

This file has to main functions:

- `preprocess_nist_data`: Will load the input data from the given CSV file and prepare each column values for being used into model training. Majority of columns are treated as categorical columns which will be one-hot-encoded. To apply one-hot-encoding we need to identify the number of distinct values in each column, this is achieved by one of the following ways:
  - From the specifications JSON file: we use the 'maxval' property in the JSON file to identify the number of distinct values. This is implemented by `valid_values_from_metdata` function.
  - Using a pre-defined list of values: the list of values is identified based on the **codebook** file and information on the IPUMS website. this is implemented using the `valid_values_from_list` function.
  - From input data: (This is only done for geographic state-dependent columns : **METAREA, METAREAD, COUNTY, CITY, SEA**. This is implemented by `valid_values_from_groundtruth` function.
- `postprocess_data`: This function basically will undo the pre-processing done earlier using the same mapping.

`differential_privacy/dp_sgd`

This directory has the implementation of the differentially private optimizer.

The implementation is based on the TensorFlow's repo implementation at `https://github.com/tensorflow/models/tree/ac8b44139f0e52260fe2e84508cb37d4d9b066b8/research/differential_privacy/`

The most important function is `compute_sanitized_gradients` that will use `per_example_gradients` to compute the gradient per example. Then, it will santizie them by

```
    sanitized_grad = self._sanitizer.sanitize(
        px_grad, self._eps_delta, sigma=self._sigma,
        tensor_name=tensor_name, add_noise=add_noise,
        num_examples=self._batches_per_lot * tf.slice(
            tf.shape(px_grad), [0], [1]))
```

The `differential_privacy/dp_sgd/sanitizer` implements the `sanitize` function that will clip the gradients and add noise to them.

`differential_privacy/privacy_accountant`

This directory implments two privacy accountant's to track the amount of (ϵ, δ) privacy spending in training. It implements the moments accountant [1], and the amortized accountant which is based on the strong composition theorem [2].

The implementation is based on the TensorFlow's repo implementation at `https://github.com/tensorflow/models/tree/ac8b44139f0e52260fe2e84508cb37d4d9b066b8/research/differential_privacy/`

Contact

For any questions, please email Moustafa Alzantot (`malzantot@ucla.edu`)

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[2] Raef Bassily, Adam Smith, and Abhradeep Thakurta. Differentially private empirical risk minimization: Efficient algorithms and tight error bounds. *arXiv preprint arXiv:1405.7085*, 2014.