**NeurIPS Hide-and-seek Privacy Challenge documentation questionnaire**

**Team name**

|  |
| --- |
| Golden\_Fleece |

**Submission filenames(s)**

|  |  |
| --- | --- |
| Hider | wangzq312\_hider\_ES\_Edited.zip |
| Seeker |  |

**What class of algorithms does your solution belong to?** (e.g. GANs, VAEs, noise-injection, nearest neighbor, etc.)

|  |  |
| --- | --- |
| Hider | Adversarial Trained Noise Injection |
| Seeker |  |

**Describe your algorithm in one sentence** (e.g. “Noise is added to the original data and then this data is returned.”)

|  |  |
| --- | --- |
| Hider | Add noise to the data, where the noise is trained in a adversarial way to drive the feature embedding of the current data entry towards that of a different data entry. |
| Seeker |  |

**Describe your algorithm in words** (e.g. “Noise is drawn from a Gaussian distribution, with mean 0 and variance s, where the dimension is determined by the size of the dataset. This noise is added to the original data to produce a noisy version of the dataset and this noisy dataset is then returned as the synthetic data.”)

|  |  |
| --- | --- |
| Hider | Step1: Train a CNN-based feature extractor using a siamese way. The feature extractor works to recognize the user and maximize the embedding distances between different users  Step2: Define a CNN with exactly the same structure, except for that in the input layer we add a noise layer of the same dimension. Then we copy the trained CNN weights and fix them. The only learnable thing is the noise.  Step3: For each user (data entry x1), select another user (x2) whose embedding is very different. Then we train the noise to move the current embedding(x1+noise) towards embedding(x2). Then retrun (x1+noise\_opt) as the generated data. Repeat for all the users. |
| Seeker |  |

**Specify any loss functions used** (e.g. “No loss functions used.”)

|  |  |
| --- | --- |
| Hider | 1. For training the CNN feature extractor, we used a **contrastive loss** that is often used to train a siamese network. The input are pairs of embeddings. The loss function minimize the distance between embeddings "model1" and "model2" if they come from the same user, and maximize the distance if they come from different users 2. For the noise shaping training, we use a self-defined **adversarial loss** that has the following parts. Say the original data is x1, noise is delta, the target data from anther use is x2   a) the distance between embedding(x1+delta) and embedding(x2)  b) the scale of the noise added [Regularization term]  c) the error of feature prediction [Optional, not used in final version]  d) the error of one-step-ahead prediction [Optional, not used in final version]  All the distances and scales are calculated with l2-norm |
| Seeker |  |

**Specify any hyperparameters and how they are optimized (or preset values)** (e.g. “The noise size, s, is set to 0.1.”)

|  |  |
| --- | --- |
| Hider | 1. In the training of CNN feature extractor  * margin m (in contrastive loss): forces representations to have 0 distance for positive pairs, and a distance greater than a margin for negative pairs. We used sigmoid nonlinearity in the siamese CNN, set as 1. * dropout rate: We set this to 0.25. Dropout rate in CNN dropout layers * iterations: number of iterations of the siamese CNN training, set to 30000 * batch size: batch size of the siamese CNN training, set to 64 * learning rate: set as 1e-3  1. In the training of the advasarial noise  * alpha, beta, gamma: the regularization term coefficients (b) (c) (d) in our adversarial loss, we set alpha=0.18, beta=gamma=0(not used) * average cluster size: To reduce the calculations, we group the users based on the embedding and use the user closest to the centroid of each cluster as the representative. Then we use the representative data to train the noise and apply the noise on other data entries in the same cluster. The average cluster size is set as 25. The number of clusters c is given by int(datasets\_size/average\_cluster\_size) * candidate numbers: we randomly select a number of users to confuse our embedding of the current user with. Default 30. * repeat: # of iterations of noise training * learning\_rate: 1e-4   Most of these hyperparameters are made based on run-time requirements (i.e. average cluster size), but some are based on baseline seeker performance. |
| Seeker |  |

**Specify any pre-trained models used by your algorithm** (e.g. “None.”)

|  |  |
| --- | --- |
| Hider | None. We do per-train a feature extractor CNN model, but its training is done when the hider program is running. |
| Seeker |  |

**Pseudo-code for your algorithm**

e.g. **Inputs:** Dataset, D, random seed

**Hyperparameters:** s (default 0.1)

1. Determine dataset dimension: n x d x T

2. Draw N ~ N(0, s), an n x d x T dimensional Gaussian

3. Return D + N

|  |  |
| --- | --- |
| Hider | **Input*s:*** Dataset D  **Hyperparameters:** c (# of clusters, default int(datasets\_size/25))   * m (candidate numbers, default 30)   Preparation:   1. Identify all the NaN’s and replace them with 0 2. MinMax Scale each feature dimension to [0,1]   Step 1: Train a CNN feature extractor  Loop iteraions until 30000:  Get 64(batch size) siamese pairs, y=1 for similar pairs and y=0 for dissimilar pairs  Calculate embeddings per data entry, in total 2\*64 embeddings  Calculate contrastive loss y\*distance + (1-y) max(margin-distance, 0)  Update CNN weights  Step 2: Define a CNN with exactly the same structure, except for that in the input layer we add a noise layer of the same dimension. Then we copy the trained CNN weights and fix them. The only learnable thing is the noise.  Step 3:   1. Use the CNN to calculate the embeddings of each data entry. 2. Cluster all users (data entries) based on their data using KMeans to create   c clusters   1. Loop c clusters:   (a) calculate the embeddings of each data entry in the cluster.  (b) find the user whose embeddings is the closest to the embeddings of the cluster centroid. The user (data\_entry) is marked as “feed\_in\_data”  (c) Select m users from D randomly. Calculate their embeddings  (d) Find the user whose embedding is most different from that of feed\_in\_data. The embedding of the selected use is named “other\_embedding”  (e) Initialize the added noise with a small value  (f) Loop 200 iterations:  (i)Calculate embeddings(feed\_in\_data+learnable\_noise)  (ii)Calculate adversarial loss, which is mainly based on distance(embeddings(feed\_in\_data+learnable\_noise), other\_embedding)  (iii)Update the noise  (g) Apply the trained noise to all the data in the current cluster, and save the perturbed data  Cleanup:   1. Reverse the MinMax Scaling 2. Reverse the NaN’s back |
| Seeker |  |

Finally, alongside this document **please also submit a commented version of your code**. Please include:

- Docstrings for each new class/function defined

- Inline comments for your main function/class

The goal of these comments is to tie the code to the description you have provided here. Please do not alter the actual content of your code - only add comments/docstrings.

**Submitting your documentation and commented code**

Please submit your commented code within a .zip or equivalent file type (1 file per solution), and share it with us as an attachment alongside this Word doc.

You can send these via email (to [nm736@cam.ac.uk](mailto:nm736@cam.ac.uk); [james.jordon@wolfson.ox.ac.uk](mailto:james.jordon@wolfson.ox.ac.uk); [es583@cam.ac.uk](mailto:es583@cam.ac.uk)) or DM James Jordon/Evgeny Saveliev on Slack (you can join the workspace [with this URL](https://join.slack.com/t/hideandseekpr-fbc8582/shared_invite/zt-k2h9xye8-RQNen128uXIG2TRsLa_ppA)).