Deep Learning HW 4 Report

Nitzan Cohen

May 2021

Contents

1 Documentation

All documentation found inside the code, including inline notes.
We will elaborate on two main train functions:

1. `def part_a_train(args)`
   Start the training of part A
   Input variables:
   - `args`: arguments configured in the program arguments
2. `def part_b_train(args)`
   Start the training of part B, training a classifier and the required GAN.
   Input variables:
   - `args`: arguments configured in the program arguments

2 Execution

Required libraries:

1. Tensorflow
2. pickle
3. tqdm
4. matplotlib
5. Numpy
6. sklearn

```
usage: main.py [-h] [-q {1,2,3}] [-e EPOCHS] [-opt optimizer] [-b BATCH_SIZE]
[-dim DIMENSION] [-dp DP] [-lr LEARNING_RATE]

Example: python3 main.py -q 1 -e 100 -opt adam -b 64 -dim 64 -lr 0.0001 -dp ./german_credit.arff

Runs part A on german_credit DS with 100 epochs, optimiser adam, batch size of 64,
leading architecture build dimension 64, learning rate of 0.0001.

GAN Sampler

optional arguments:
  -h, --help            show this help message and exit
  -q {1,2,3}, --question {1,2,3}
                        Which question to run (note: 1 - part A, 2 - part B,
                        3 - refers to train a classifier individually)
  -e EPOCHS, --epochs EPOCHS
                        How many epochs the network will train
  -opt optimizer, --optimizer optimizer
                        The optimizer used for training
  -b BATCH_SIZE, --batch_size BATCH_SIZE
                        the batch size used in train
  -dim DIMENSION, --dimension DIMENSION
                        The leading dimension in the GAN model
  -dp DP, --dataset_path DP
```

```
                        full path to the dataset file
  -lr LEARNING_RATE, --learning_rate LEARNING_RATE
                        the learning rate used in training
```

3 Dataset Preprocess

The preprocess involved two steps:

- Turning every categorical or nominal attribute into one-hot binary variable.
- Min-max scale into range [0,1] for each numeric column.

4 Part A

In this section we will describe the end-to-end process, from data to final model results.

4.1 Model Architecture

4.1.1 Generator

All layers have leaky ReLU activation with alpha=0.4, except output layer with relu capped with max value of 0.1.

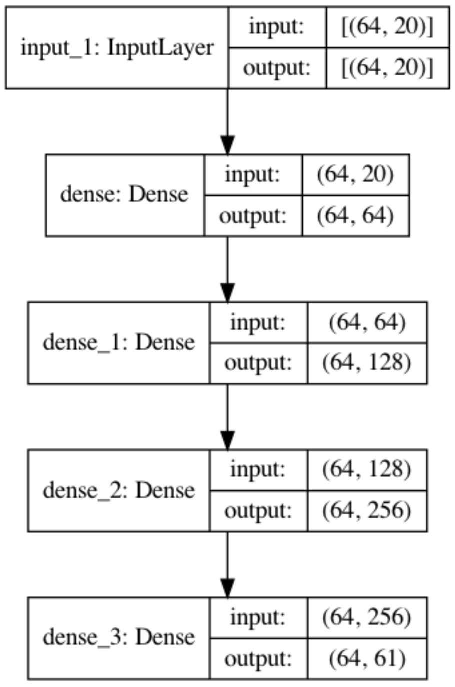

Figure 1: Generator Architecture

4.1.2 Discriminator

All layers have leaky ReLU activation with alpha=0.4.

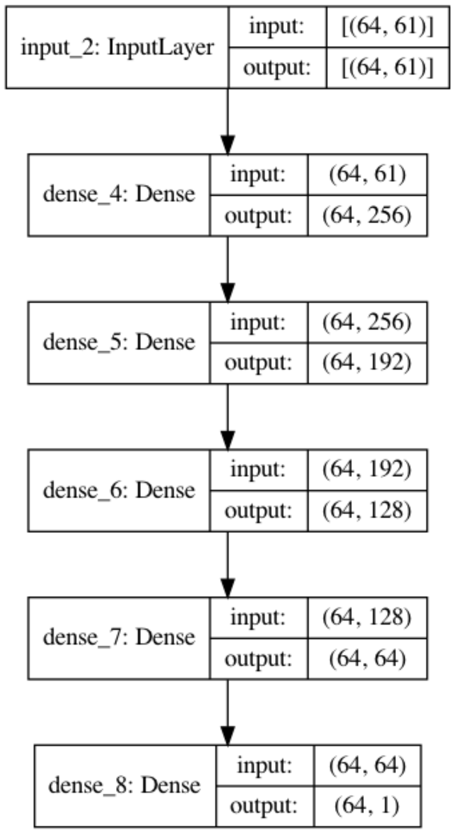

Figure 2: Discriminator Architecture

4.1.3 GAN

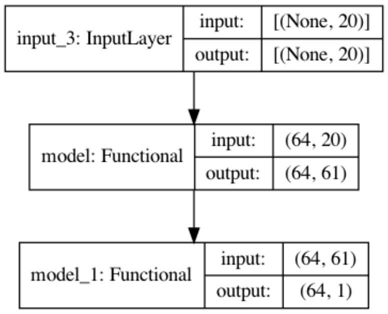

Figure 3: GAN Overview Architecture

4.2 Model Performance - German-Credit Dataset

4.2.1 Training Configuration

- One side label smoothing from 1.0 to 0.9

- Label flipping at each epoch
- Epoch: 500
- Learning rate: 1e-04
- Optimiser: Adam with $\beta_1 = 0.5$
- Noise to the generator size 20 sampled from Normal Gaussian

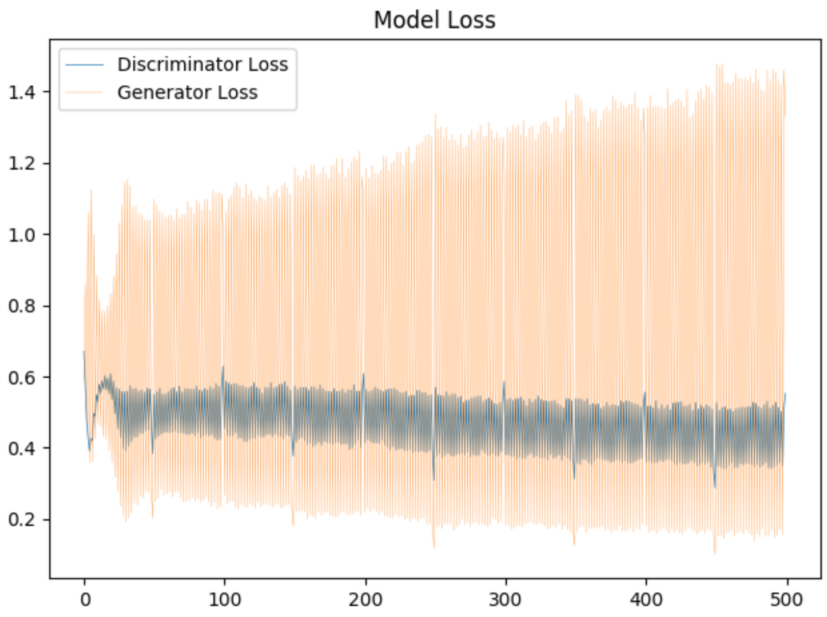

Figure 4: Model Loss

There was no consistent leader during this Nash-Equilibrium optimisation process.
Empirically, we found that the task of generating tabular data is less harder than generating image, where usually GAN is used for.
Therefore, we tried to boost both the discriminator and the generator with a label flipping process.

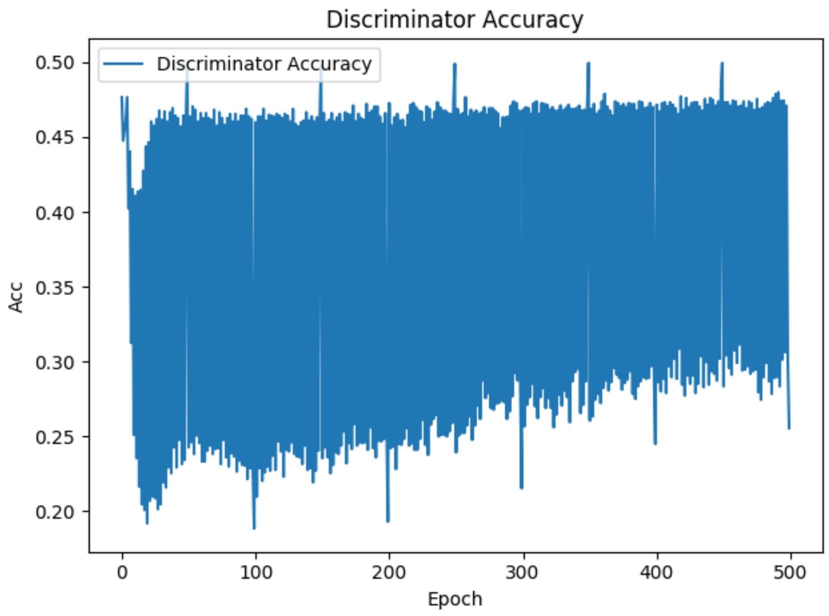

Figure 5: Discriminator Accuracy

4.2.2 Analysis

We chose randomly an example from the test data and compute the euclidean distance between it and the generated samples, here are few examples that fooled the discriminator:

- Real sample: [1.0, 0.0, 0.0, 0.0, 0.20588235294117646, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.16176956091119182, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.3333333333333333, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.3333333333333333, 1.0, 0.0, 0.0, 0.0, 0.0892857142857143, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0]

- Distance: 2.4137081720645126
  sample: [1.0, 0.0, 0.0, 0.0, 0.036330856, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.05014431, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.5470556, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.17783742, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0]

- Distance: 2.3815679737962343
  sample: [0.0, 1.0, 0.0, 0.0, 0.06805168, 0.0, 0.0, 0.43085805, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.43788788, 0.0, 0.0, 0.0, 0.0, 0.0, 0.44032064, 0.4642017, 1.0, 0.0, 0.0, 0.0, 0.045290995, 0.0, 0.0, 0.6460106, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.54821527, 0.0, 0.0, 0.0, 0.0]

- Distance: 2.2733856906724657
  sample: [0.0, 0.0, 0.0, 1.0, 0.23521383, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.48069248, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.1808678, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0062979776, 0.0]

As we can, the generator even learned to enforce the one-hot encoded nominal features.
Finally, among 100 generated samples, there were 53 that fooled the discriminator.

4.3 Model Performance - Diabetes Dataset

4.3.1 Training Configuration

- One side label smoothing from 1.0 to 0.9
- Label flipping each 10 epochs
- Epoch: 500
- Learning rate: 1e-04
- Optimiser: Adam with $\beta_1 = 0.5$
- Noise to the generator size 20 sampled from Normal Gaussian

4.3.2 Analysis

We chose randomly an example from the test data and compute the euclidean distance between it and the generated samples, here are few examples that fooled the discriminator:

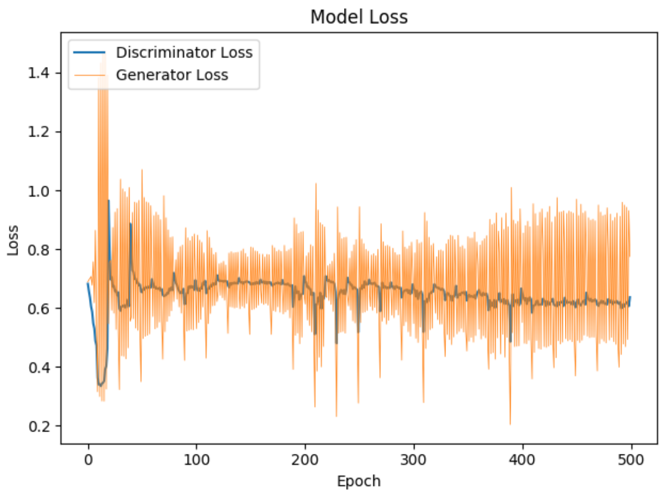

Figure 6: Model Loss

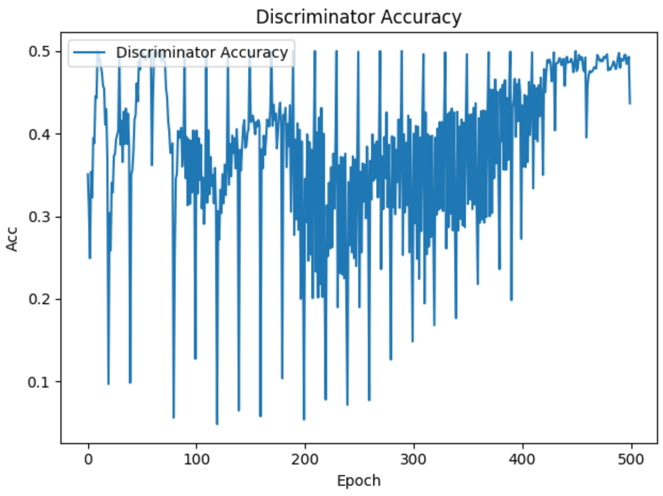

Figure 7: Discriminator Accuracy

- Real sample: [0.3529411764705882, 0.49246231155778897, 0.47540983606557380, 0.33333333333333337, 0.22458628841607564, 0.5067064083457526, 0.15029888983774553, 0.3666666666666667]

- Distance: 0.34345923305518544
  sample: [0.2539093, 0.7171751, 0.5529481, 0.4247546, 0.09015887, 0.42065978, 0.10989567, 0.2394508]

- Distance: 0.4937756646348973
  sample: [0.08138048, 0.36846095, 0.47931764, 0.12624416, 0.025377715, 0.39540073, 0.07229906, 0.13507505]

- Distance: 0.5296552999630616
  sample: [0.13710321, 0.8566533, 0.6398124, 0.16070314, 0.3536693, 0.36494622, 0.15442753, 0.4544667]

Among 100 generated samples, there were 37 that fooled the discriminator.

5 Part B

In this section we will present the output for training and validation loss of our model and the random forest classifier on two different data-sets.

5.1 Random Forest Results

5.1.1 Hyper Parameters

We began with manual hyper parameters search and finally came to a point where we decide to take the default of sk-learn "RandomForestClassifier" instance.

- Estimators: 100
- Criterion: Gini
- None max-depth
- The number of features to consider when looking for the best split: sqrt(num features)

5.1.2 Results - German-Credit Dataset

- Accuracy on test: 77%
- AUC: 0.80

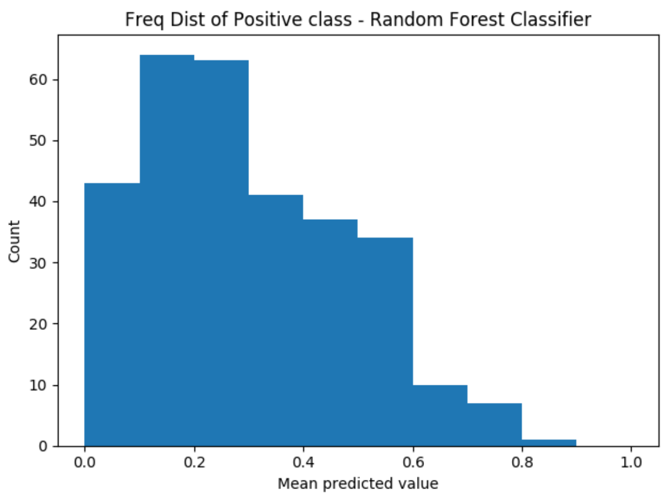

Figure 8: Positive Confidence Score Distribution

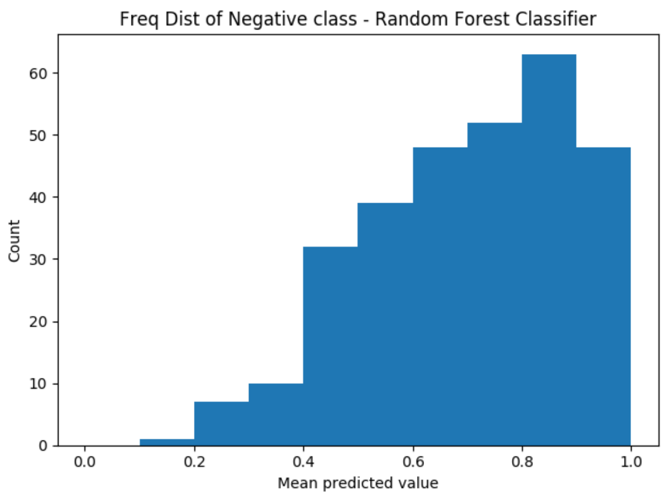

Figure 9: Negative Confidence Score Distribution

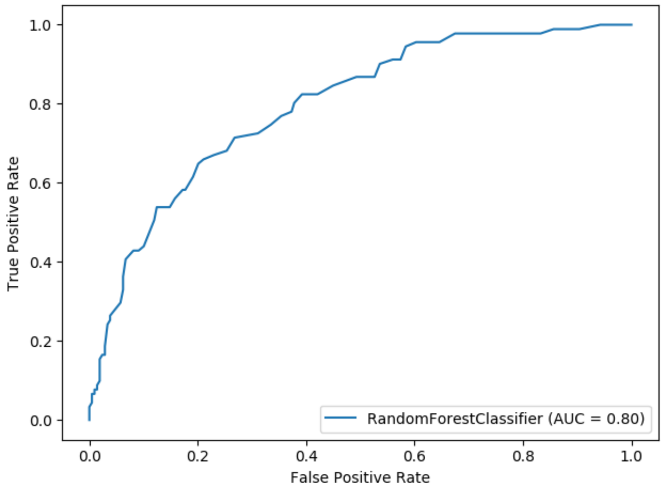

Figure 10: ROC Curve

5.1.3 Results - Diabetes Dataset

- Accuracy on test: 74%
- AUC: 0.80

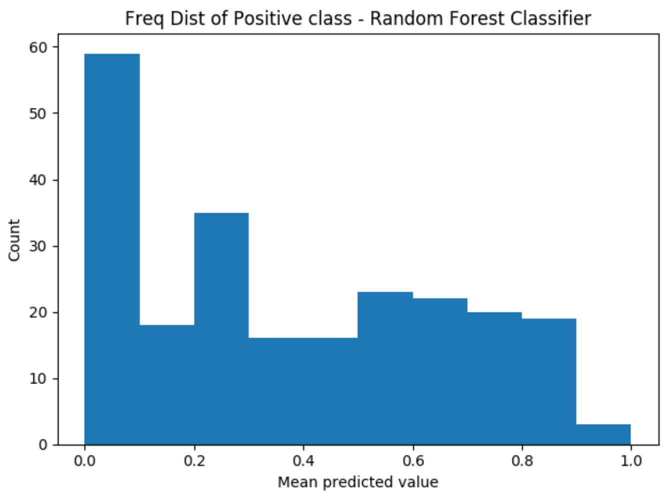

Figure 11: Positive Confidence Score Distribution

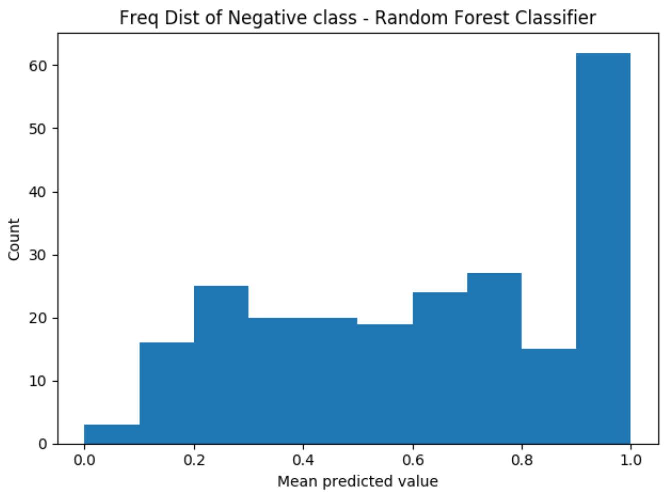

Figure 12: Negative Confidence Score Distribution

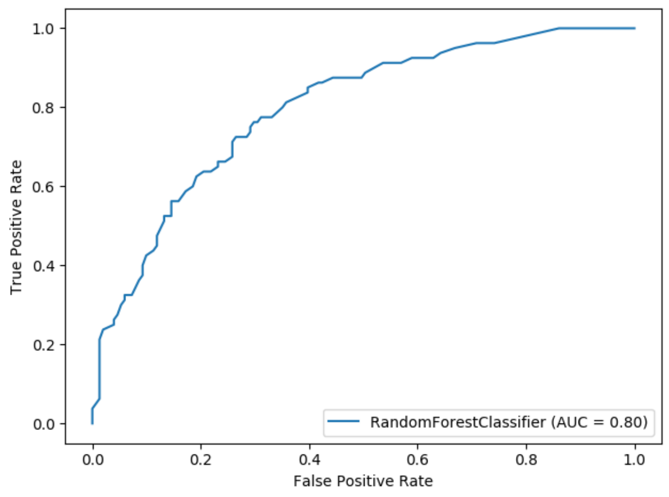

Figure 13: ROC Curve

5.2 Model Architecture

In order to increase the sample diversity we used the same model architecture used as the previous section with a key difference:
We added to D a Mini-Batch Discrimination Layer [1] before the output layer with B=5, C=3. i.e we have 5 additional features for intra batch features

5.3 Model Performance - German-Credit Dataset

5.3.1 Training Configuration/Tricks

- One sided label smoothing from 1.0 to 0.8
- Added decay Gaussian noise $(mean = 0, stddev = 0.2 * (1/epoch))$ to the generated samples in the discriminator training
- Epoch: 250
- Learning rate - Generator: 0.0005, Discriminator: 0.00001
- Optimiser: Adam with $\beta_1 = 0.5$
- Target confidence score (C) for generator randomly taken from [0.05,0.8)

5.3.2 Performance

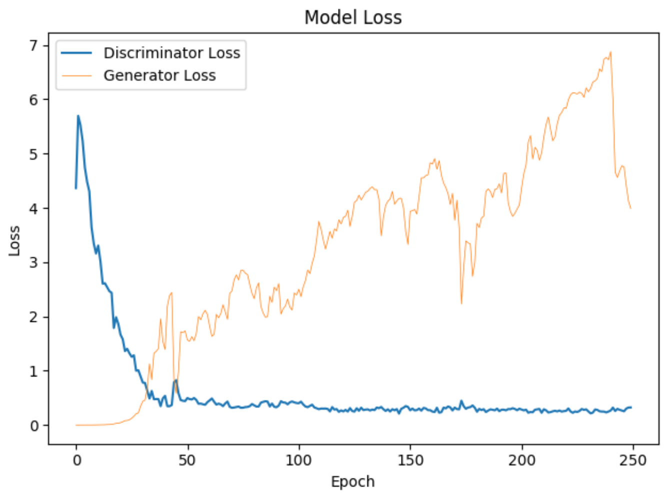

Figure 14: Model Loss

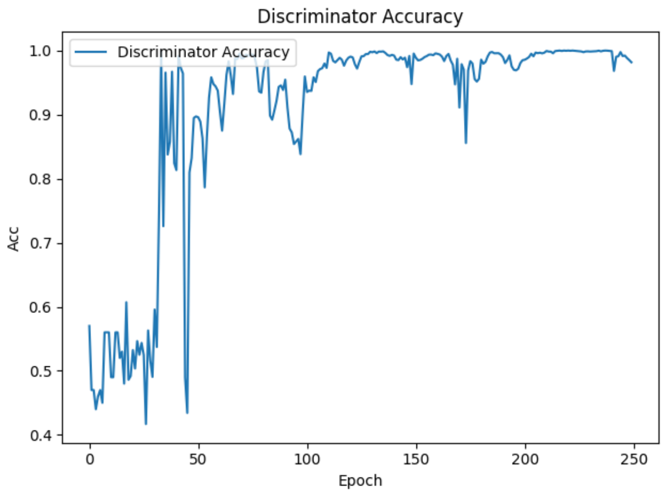

Figure 15: Discriminator Accuracy

5.3.3 Generated Samples

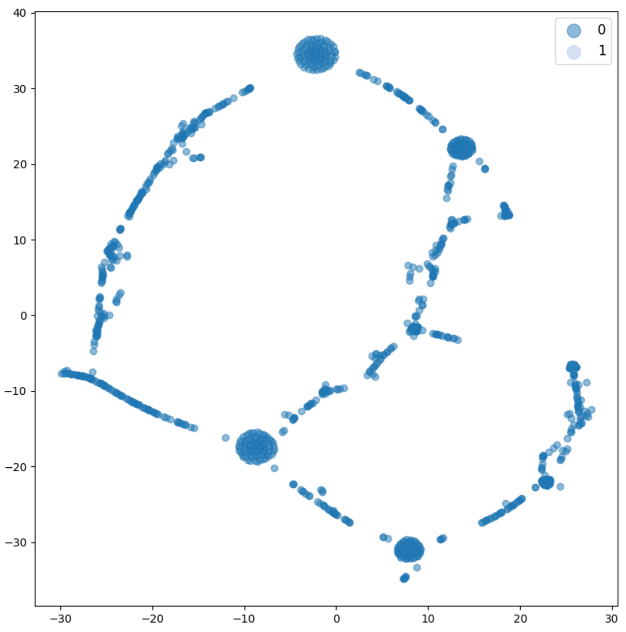

Figure 16: 1000 Generated Samples with the final generator projected with t-SNE dimension reduction algorithm into 2-d plane

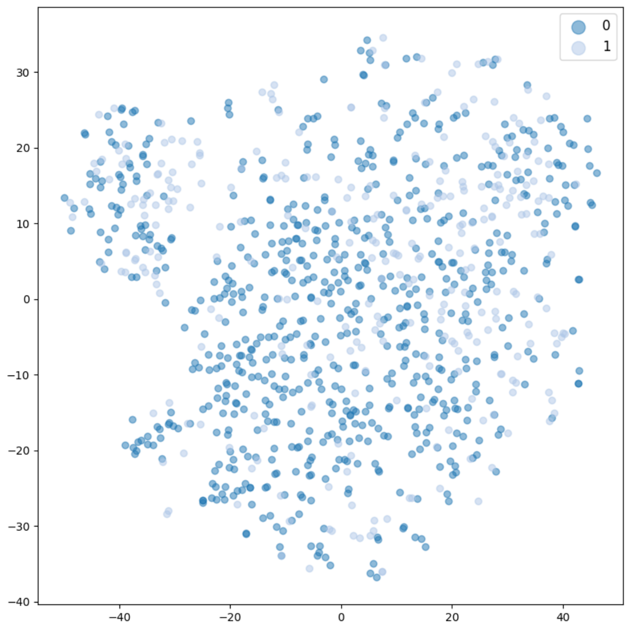

Figure 17: 1000 Real Samples from "German-Credit" projected with t-SNE dimension reduction algorithm into 2-d plane

5.3.4 Classifier Distributions on Generated Samples

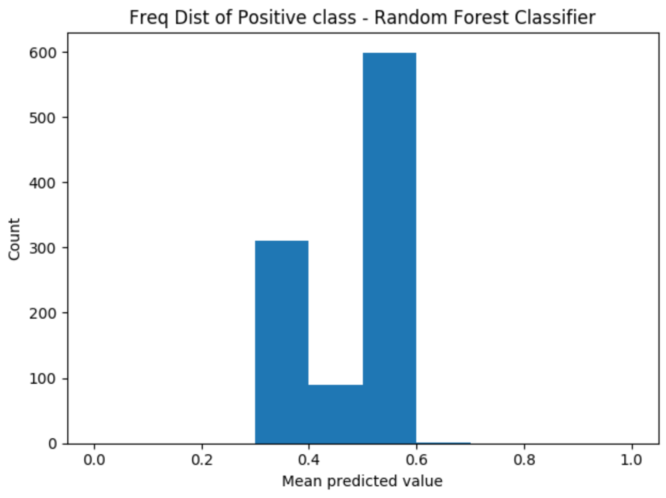

Figure 18: Positive Confidence Score Distribution on Generated Samples

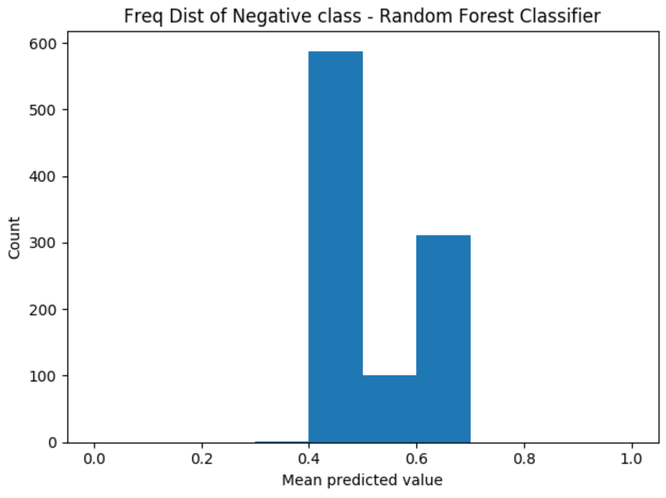

Figure 19: Negative Confidence Score Distribution on Generated Samples

5.3.5 Conclusions

- As we can see from figure 16 the sample diversity of the model is quite large compared to the dataset.
- However, from figure 18 we can see that the distribution of the confidence of the generated samples is not diverse as the original dataset.
- The generator was pretty balanced in the process of generation, successfully generated 600 samples on the positive class, however with "low confidence" scores and with low diversity.
- Even though we sampled a desired confidence score from $[0.05, 0.8)$ The generator modeled the distribution among the confidence score of range $[0.3, 65]$

- In our first experiments (without the MiniBatch Discrimination layer) our model suffered from mode collapse, generating samples only from range $[0.5, 0.6]$, mainly suffering from generating negative examples.
- Here can observe here that in this case the model is partially suffering from it, extending it's range of confidence score distribution. Allow it to model samples from the training sample and even model a 'cluster' out of the training sample (the top 'cluster' in figure 16)

5.4 Model Performance - Diabetes Dataset

5.4.1 Training Configuration/Tricks

- One sided label smoothing from 1.0 to 0.8
- Added decay Gaussian noise $(mean = 0, stddev = 0.2 * (1/epoch))$ to the generated samples in the discriminator training
- Epoch: 250
- Learning rate - Generator: 0.0005, Discriminator: 0.00005
- Optimiser: Adam with $\beta_1 = 0.5$
- Target confidence score (C) for generator randomly taken from [0.05,0.8)

5.4.2 Performance

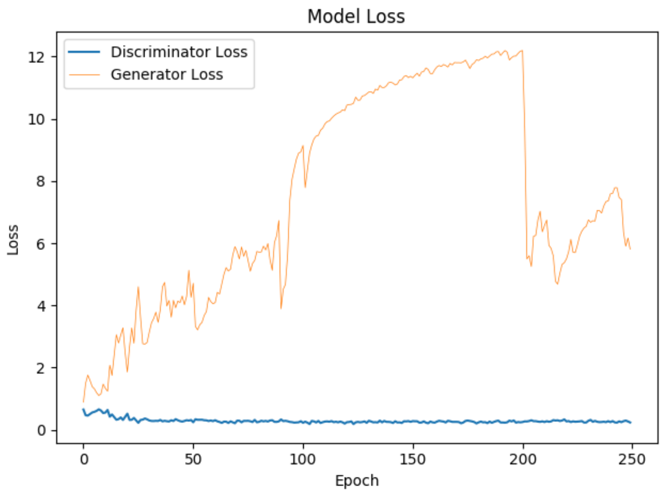

Figure 20: Model Loss

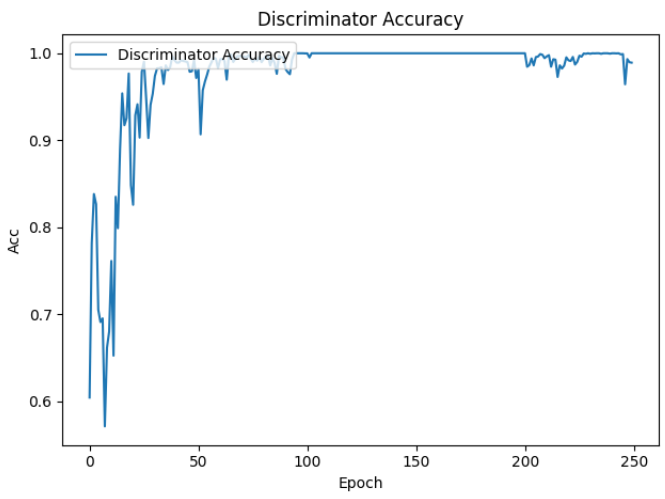

Figure 21: Discriminator Accuracy

5.4.3 Generated Samples

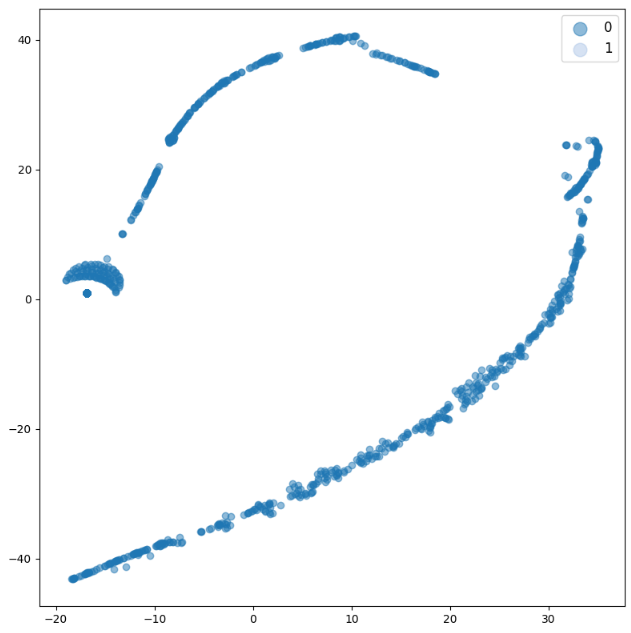

Figure 22: 1000 Generated Samples with the final generator projected with t-SNE dimension reduction algorithm into 2-d plane

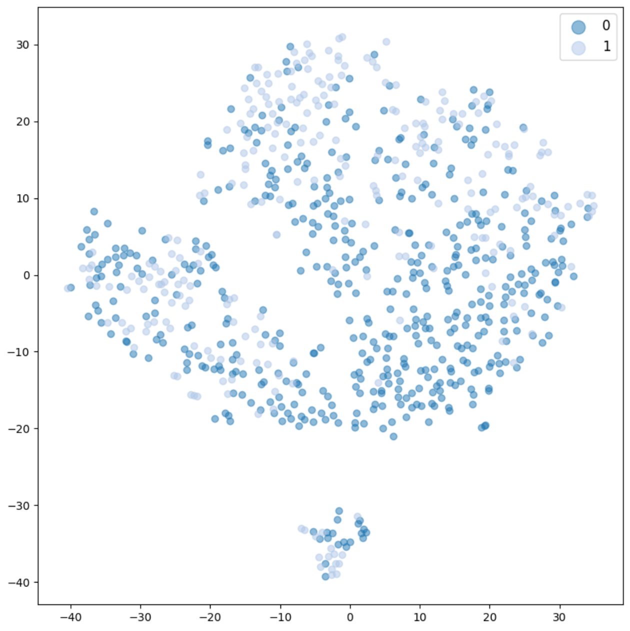

Figure 23: 1000 Real Samples from "Diabetes" projected with t-SNE dimension reduction algorithm into 2-d plane

5.4.4 Classifier Distributions on Generated Samples

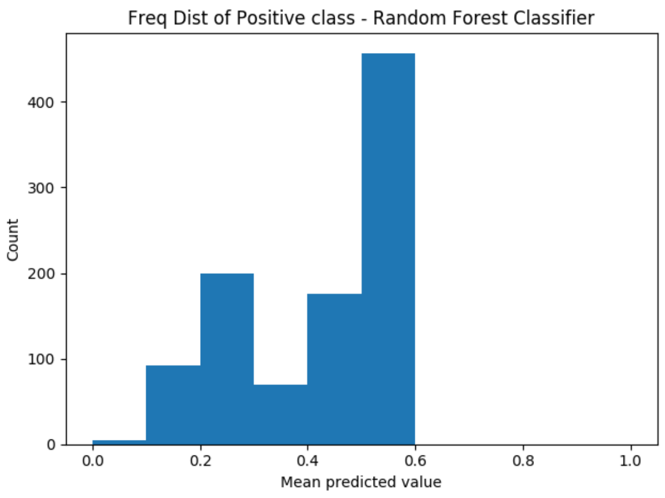

Figure 24: Positive Confidence Score Distribution on Generated Samples

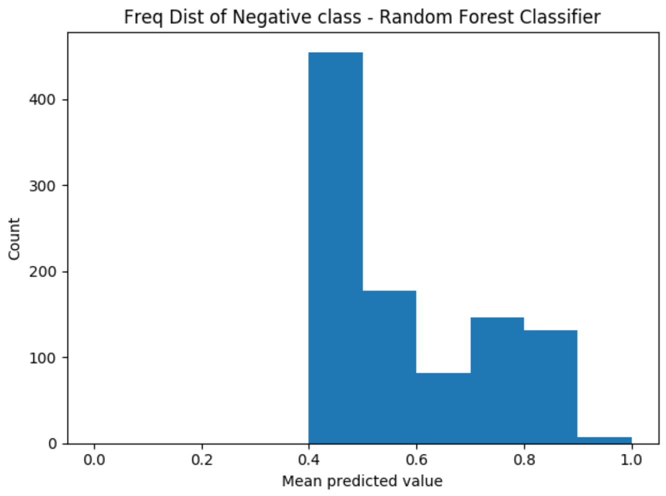

Figure 25: Negative Confidence Score Distribution on Generated Samples

5.4.5 Conclusions

- In this particular training process we can observe that the discriminator ”overpowered” the generator pretty fast.
- The generator was pretty balanced in the process of generation, successfully generated 500 samples on the positive class, however with ”low confidence” scores and with low diversity on the positive class.
- Even though we sampled a desired confidence score from $[0.05, 0.8)$ The generator modeled the distribution among the confidence score of range $[0, 0.65]$.
- By looking at figure 22 one can learn that the generator partially suffered from mode collapse, mainly in the positive class. However, managed to generate samples with a wider spectrum of confidence scores in the negative class.

6 Final Improvement Thoughts

We think one of the main issues here is the lack of connection between the ”desired confidence score” and the actual generated sample neither to it’s quality or diversity. There is no learning mechanism involved and should be added. Therefore, we propose:

- Add a small neural net which takes z (the noise) as an input and learns to output a confidence score. The final input to the generator is the confidence score produced by this neural net and z.
- Then we shall add additional term to the loss function which penalises in L2 distance between the actual score and the desired score
- The added NN will learn to output a corresponding score that will help to connect between the generated samples and the desired score.

References

[1] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.