Deep Learning - Assignment 4

Zahi Mizrahi 206129462, Aviad Azriel 203660360

1. **Introduction**

   Generative models are among the most exciting architecture in the field of deep learning today. Their ability to generate high-quality data with little-to-no interaction or knowledge with the existing data enables us to accomplish tasks thought impossible not long ago.
   In this assignment, we implement adversarial generative models for dealing with a different scenario.
   The first scenario is synthesizing tabular datasets with GANs.
   The second scenario is the challenge of understanding the inner workings of a black-box model by simultaneously generating samples and predicting the score they will receive from the model.

2. **Data**

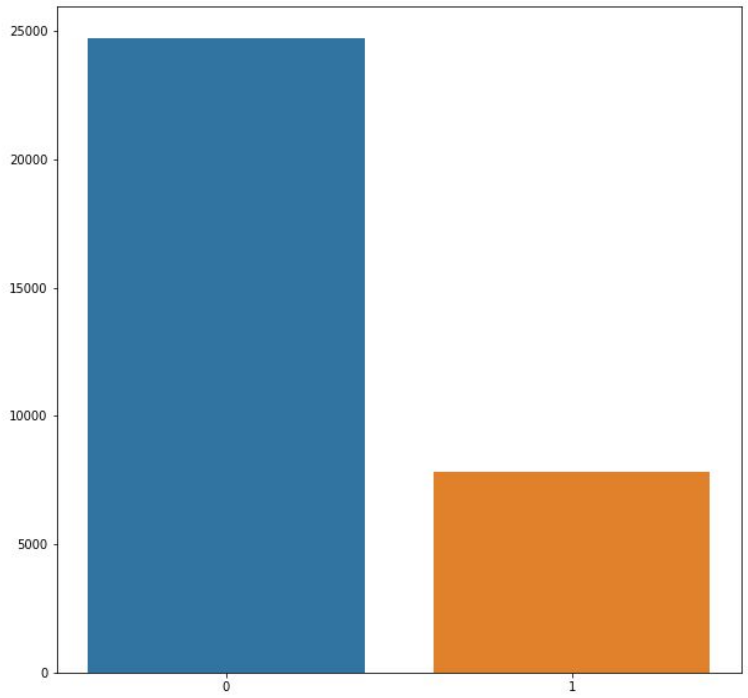
   2.1 Data Preprocessing

   In this assignment the input data is a tabular data in arff format. An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. All the character field transform by label encoder to categorial field and implement one of two method to normalized the data:

   - Standard - Standardize features by removing the mean and scaling to unit variance. The standard score of a sample x is calculated as: z = (x - u) / s. where u is the mean of the training samples or zero and s is the standard deviation of the training samples.
   - Power transforms are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like.

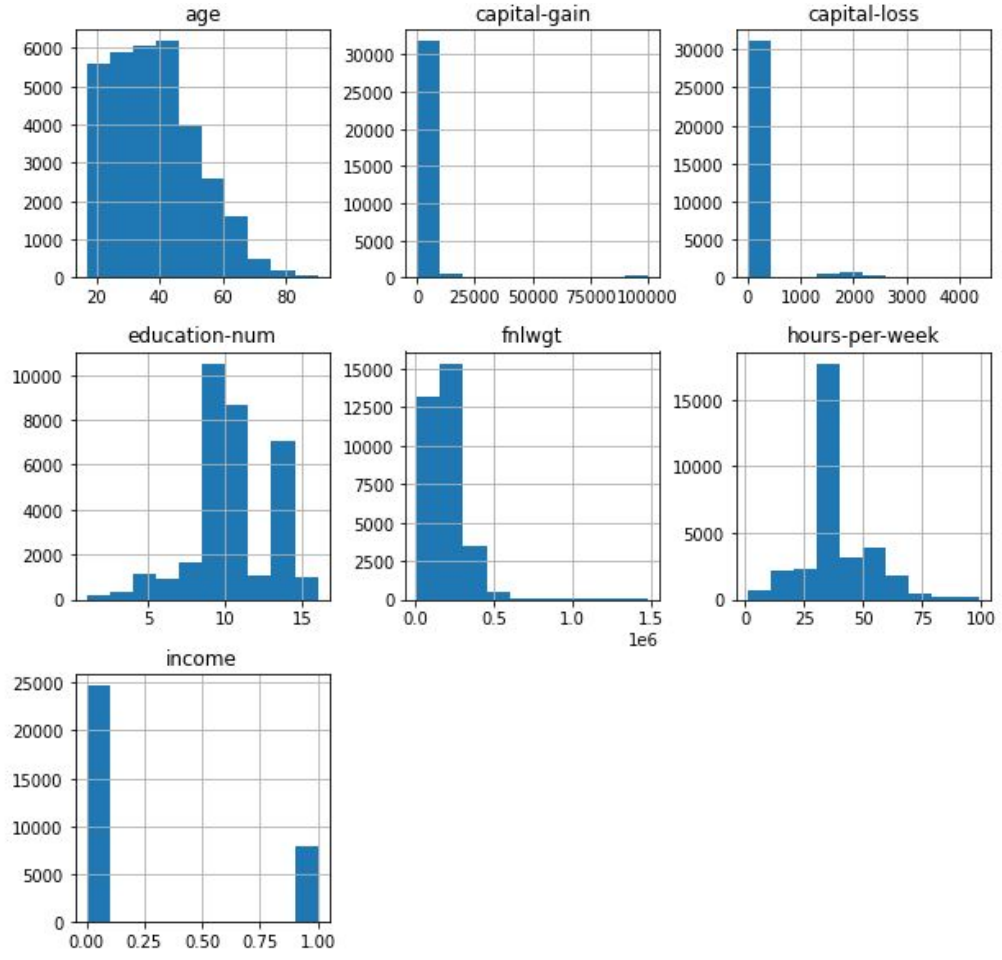
   2.2 Data Exploration

   We tried to explore see and see the distribution of each feature in both datasets:
   Adult -
   Distribution of the label:



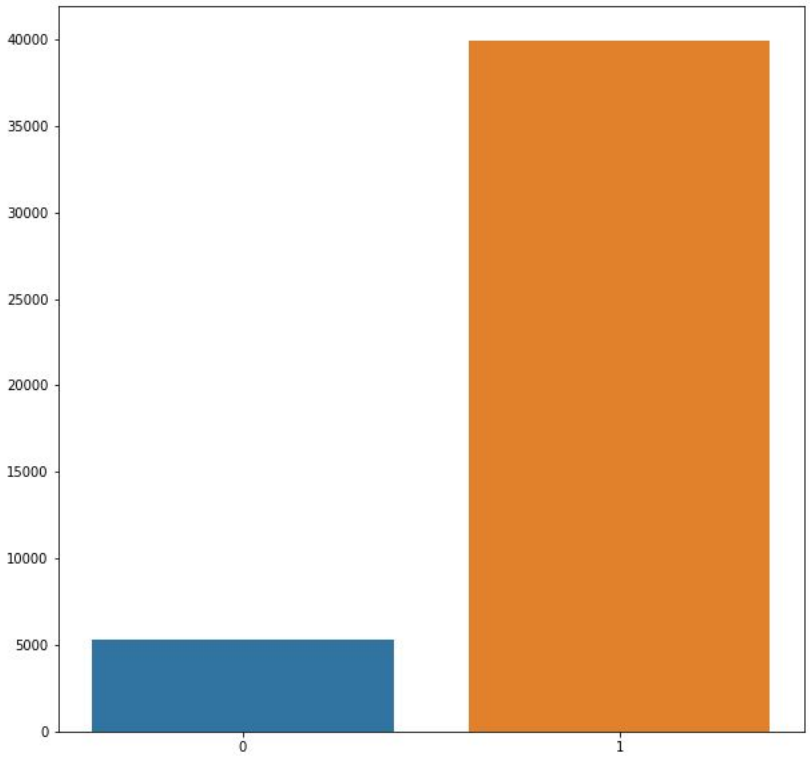
(imbalanced)

Distribution of the numeric features of the data:



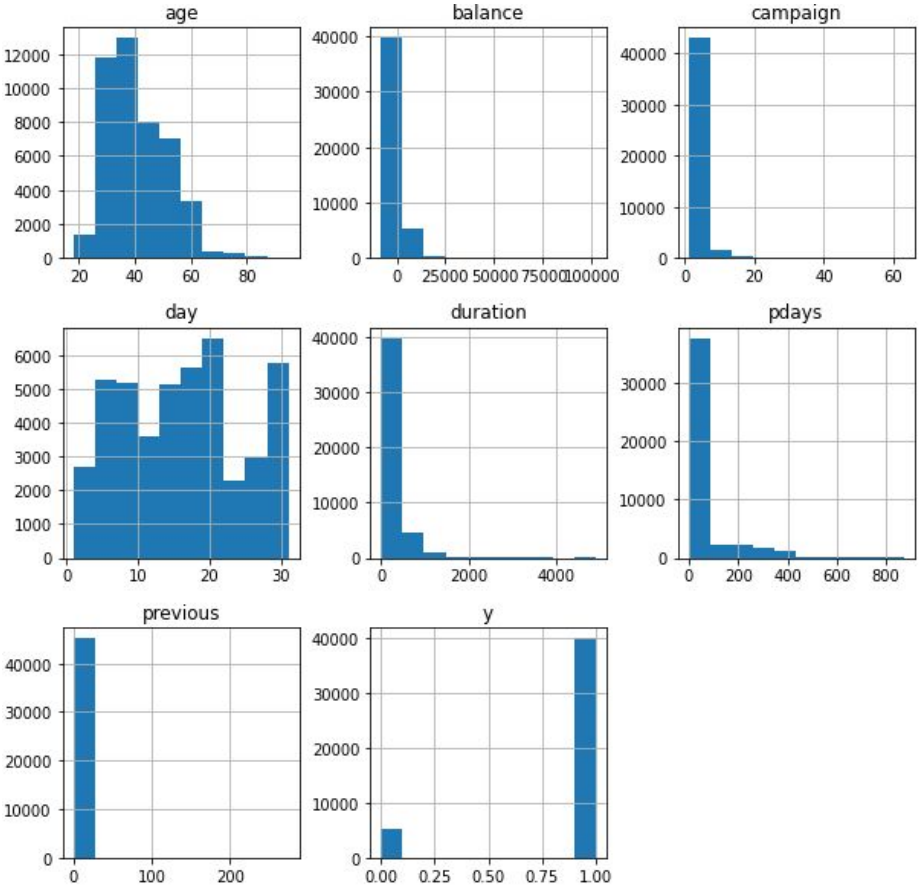
Bank data -

Distribution of the label:



(imbalanced)

Distribution of the numeric features of the data:

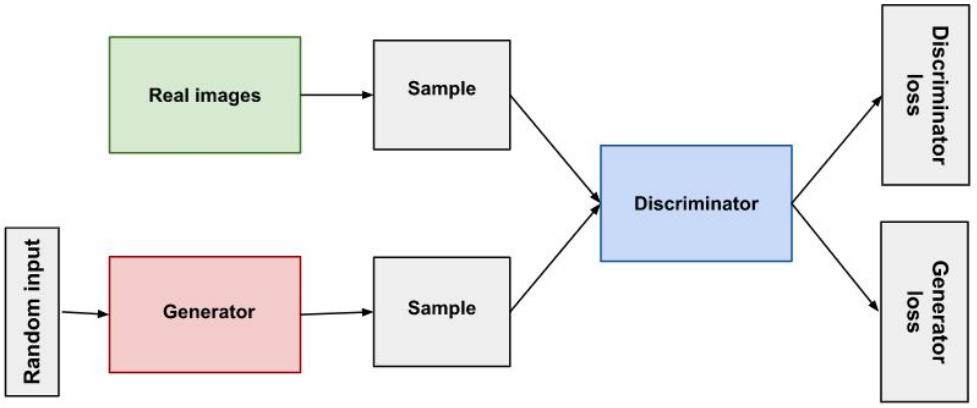


3. Model (Part 1)

3.1 Simple GAN

A generative adversarial network (GAN) has two parts:

- The generator learns to generate plausible data. The generated instances become negative training examples for the discriminator.
- The discriminator learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.



The two neural networks contest with each other in a game. Given a training set, this technique learns to generate new data with the same statistics as the training set. The generative network generates candidates while the discriminative network evaluates them. The contest operates in terms of data distributions. Typically, the generative network learns to map from a latent space to a data distribution of interest, while the discriminative network distinguishes candidates produced by the generator from the true data distribution. The generative network's training objective is to increase the error rate of the discriminative network (i.e., "fool" the discriminator network by producing novel candidates that the discriminator thinks are not synthesized (are part of the true data distribution)).
A known dataset serves as the initial training data for the discriminator. Training involves presenting it with samples from the training dataset, until it achieves acceptable accuracy. The generator training is based on whether it succeeds in fooling the discriminator. Typically the generator is seeded with randomized input that is sampled from a predefined latent space (e.g. a multivariate normal distribution). After that, candidates synthesized by the generator are evaluated by the discriminator. Backpropagation is applied in both networks so that the generator produces better data, while the discriminator becomes more skilled at flagging

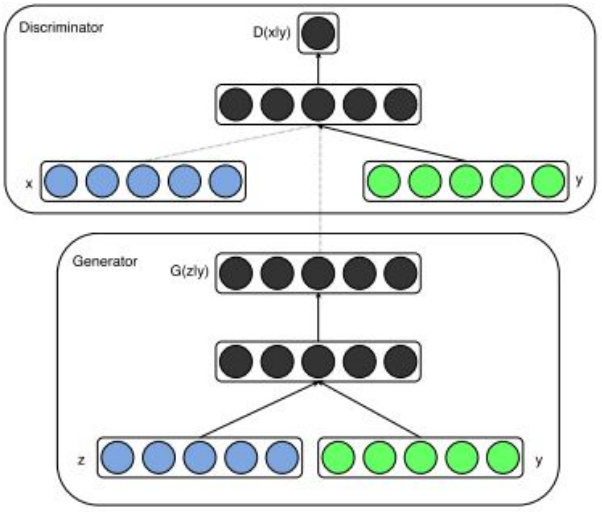
synthetic data.The generator is typically a deconvolutional neural network, and the discriminator is a convolutional neural network.

3.2 CGAN

To train a GAN that is capable of generating samples from the tabular dataset we implement a Conditional GAN (CGAN) architecture structure that knows how to contain the type of information that is tabular data and not as an image (like a traditional GAN).

Why CGAN Architecture

In an unconditioned generative model, there is no control on modes of the data being generated. However, by conditioning the model on additional information it is possible to direct the data generation process. Such conditioning could be based on class labels, on some part of data for inpainting like, or even on data from different modality.
Generative adversarial nets can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information y. y could be any kind of auxiliary information, such as class labels or data from other modalities. We can perform the conditioning by feeding y into both the discriminator and generator as an additional input layer.
In the generator the prior input noise pz(z), and y are combined in joint hidden representation, and the adversarial training framework allows for considerable flexibility in how this hidden representation is composed. In the discriminator x and y are presented as inputs and to a discriminative function.



The objective function of a two-player minimax game would be:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))]$$

Based on Generative Adversarial Nets and Conditional Generative Adversarial Nets.

4. Challenge

The reason they are difficult to train is that both the generator model and the discriminator model are trained simultaneously in a game. This means that improvements to one model come at the expense of the other model. The goal of training two models involves finding a point of equilibrium between the two competing concerns. It also means that every time the parameters of one of the models are updated, the nature of the optimization problem that is being solved is changed.
In neural network terms, the technical challenge of training two competing neural networks at the same time is that they can fail to converge.
A common failure mode is that instead of finding a point of equilibrium, the generator oscillates between generating specific examples in the domain. Perhaps the most challenging model failure is the case where multiple inputs to the generator result in the generation of the same output. This is referred to as “mode collapse,” and may represent one of the most challenging issues when training GANs.

Tabular Data
Modeling the probability distribution of rows in tabular data and generating realistic synthetic data is a non-trivial task. Tabular data usually contains a mix of discrete and continuous columns. Existing statistical and deep neural network models fail to properly model this type of data.
Finally, there are no good objective metrics for evaluating whether a GAN is performing well during training. E.g. reviewing loss is not sufficient. Instead, the best approach is to visually inspect generated examples and use subjective evaluation.

5. Solutions

5.1 Normalization

Properly representing the data is critical in training neural networks. Discrete values can naturally be represented as one-hot vectors, while representing continuous values with arbitrary distribution is non-trivial.

5.2 Dealing with Imbalanced data

in which the major category appears in more than 90% of the rows. This creates severe mode collapse. Missing a minor category only causes tiny changes to the data distribution that is hard to be detected by the discriminator. Imbalanced data also leads to insufficient training opportunities for minor classes.
We tested the model on two different datasets and both were an imbalance.
If the training data are randomly sampled during training, the rows that fall into the minor category will not be sufficiently represented, thus the generator may not be trained correctly. If the training data are resampled, the generator learns the resampled distribution which is different from the real data distribution. This problem is reminiscent of the “class imbalance” problem in discriminatory modeling - the challenge however is exacerbated since there is not a single column to balance and the real data distribution should be kept intact.

5.3 Mini-batch

To overcome the Mode collapse problem - a problem related to the generator's inability to produce diverse samples, we trained the model using mini-batch . The concept of minibatch discrimination is quite general: any discriminator model that

looks at multiple examples in combination, rather than in isolation, could potentially help avoid collapse of the generator.
Instead of looking at each sample separately we will look at a batch of samples and if he produced all of them only as one type then the discriminator will reject what the generator has developed.

5.4 Adam Optimization

All models were trained with mini-batch stochastic gradient descent (SGD) with a mini-batch size of 128

5.5 Dropout

Dropout has been proven to be a very useful and widely used technique in neural networks to prevent overfitting. In practice, it simply consists of omitting or dropping out, the output of some randomly chosen neurons with a probability d or dropout rate. The intuition behind this process is to ensure that neurons are not entirely dependent on a specific set of other neurons to produce their outputs. Instead, with dropout, each neuron relies on the population behavior of several other neurons, promoting generalization in the network. Hence, the overall network becomes more flexible and less prone to overfitting

6. Model architecture (Part 1)

6.1 GAN

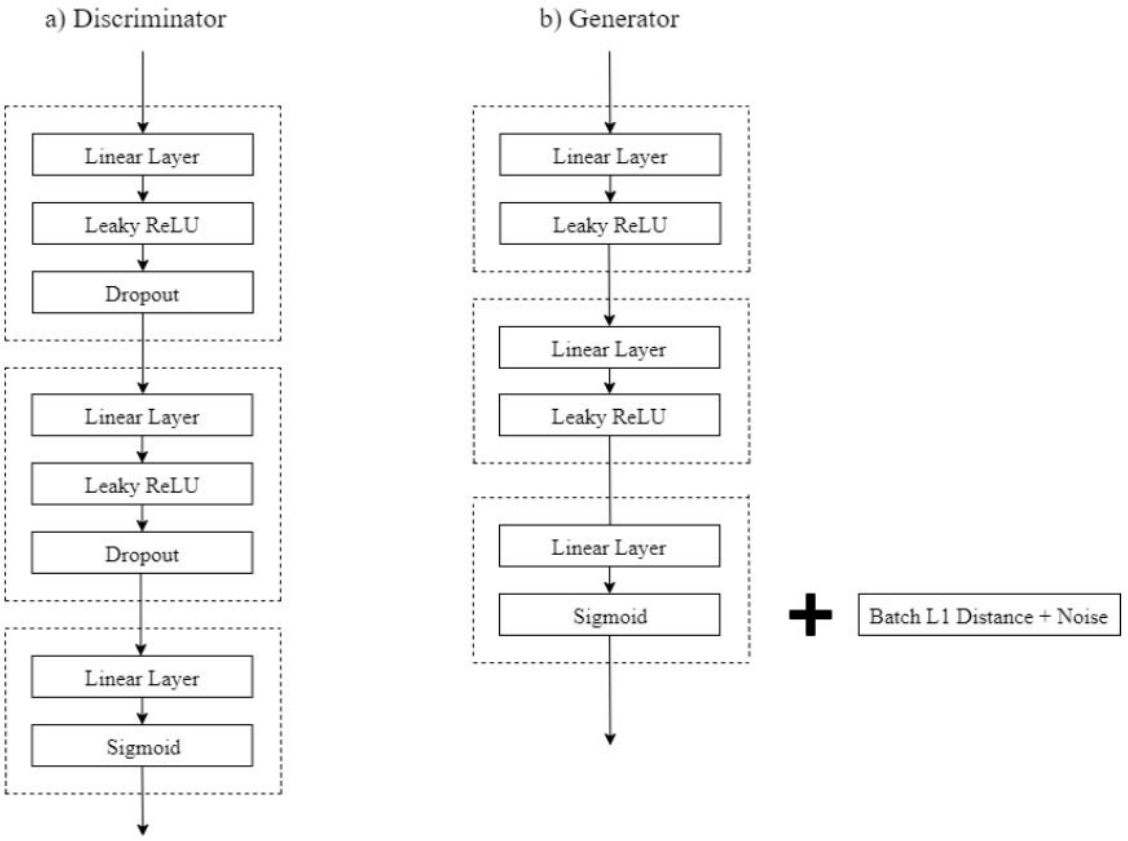
To build the architecture we used a number of articles and combined different models to try to create an ultimate model for data in a tabular structure.
The model Based on CGAN and combines GAN-SMOTE, DTGAN, TCGAN.
We design a conditional generator and training-by-sampling to deal with the imbalanced discrete columns. And we use fully-connected networks and several recent techniques to train a high-quality model.

discriminator

```
Model: "Discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_22 (InputLayer)        [(128, 1)]                0         
_________________________________________________________________
input_21 (InputLayer)        [(128, 14)]               0         
_________________________________________________________________
embedding_9 (Embedding)      (128, 1, 1)               2         
_________________________________________________________________
flatten_14 (Flatten)         (128, 14)                 0         
_________________________________________________________________
flatten_13 (Flatten)         (128, 1)                  0         
_________________________________________________________________
multiply_9 (Multiply)        (128, 14)                 0         
_________________________________________________________________
dense_36 (Dense)             (128, 512)                7680      
_________________________________________________________________
dropout_10 (Dropout)         (128, 512)                0         
_________________________________________________________________
dense_37 (Dense)             (128, 256)                131328    
_________________________________________________________________
dropout_11 (Dropout)         (128, 256)                0         
_________________________________________________________________
dense_38 (Dense)             (128, 128)                32896     
_________________________________________________________________
dense_39 (Dense)             (128, 1)                  129       
=================================================================
Total params: 172,035
Trainable params: 172,035
Non-trainable params: 0
```

Generator

```
Model: "Generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_20 (InputLayer)        [(128, 1)]                0         
_________________________________________________________________
embedding_8 (Embedding)      (128, 1, 1)               2         
_________________________________________________________________
input_19 (InputLayer)        [(128, 32)]               0         
_________________________________________________________________
flatten_12 (Flatten)         (128, 1)                  0         
_________________________________________________________________
multiply_8 (Multiply)        (128, 32)                 0         
_________________________________________________________________
dense_32 (Dense)             (128, 128)                4224      
_________________________________________________________________
dropout_8 (Dropout)          (128, 128)                0         
_________________________________________________________________
dense_33 (Dense)             (128, 256)                33024     
_________________________________________________________________
dropout_9 (Dropout)          (128, 256)                0         
_________________________________________________________________
dense_34 (Dense)             (128, 512)                131584    
_________________________________________________________________
dense_35 (Dense)             (128, 14)                 7182      
=================================================================
Total params: 176,016
Trainable params: 176,016
Non-trainable params: 0
```



6.2 CGAN

```
Model: "CGAN"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_39 (InputLayer)           [(128, 32)]          0
__________________________________________________________________________________________________
input_40 (InputLayer)           [(128, 1)]           0
__________________________________________________________________________________________________
Generator (Functional)          (128, 15)            176529      input_39[0][0]
                                                                 input_40[0][0]
__________________________________________________________________________________________________
Discriminator (Functional)      (128, 1)             172547      Generator[0][0]
                                                                 input_40[0][0]
==================================================================================================
Total params: 349,076
Trainable params: 176,529
Non-trainable params: 172,547
```

Since columns in a row do not have local structure (difference distribution), we use fully-connected networks in generator and critic to capture all possible correlations between columns. Specifically, we use two fully-connected hidden layers in both generator and critic. In the generator, we use batch-normalization and Relu activation functions. After two hidden layers, the synthetic row representation is generated using max activation functions. The scalar values αi is generated by tanh, while the mode indicator βi and discrete values di is generated by gumbel softmax. In critic, we use leaky relu function and dropout on each hidden layer.

6.3 Early Stopping

We used Early Stopping that similar to the callback of Keras just take in acount the two models - the discrimination and the generetor. It means that we train the network while monitoring the loss of the validation of the two models. Because there is always a struggle between the discriminator and the generator in which the loss of one will increase the other will decrease as soon as they both reach a point where both do not improve and the validation stops decreasing we stop the training process. It is used to avoid overfitting and to increase the performance (running for too many epochs). We apply patience of 2000 which means the training process will stop only when the validation loss stop decreasing for at least 2000 epochs.

7. Experiments and Results (Part 1)

7.1 Hyper Parameters

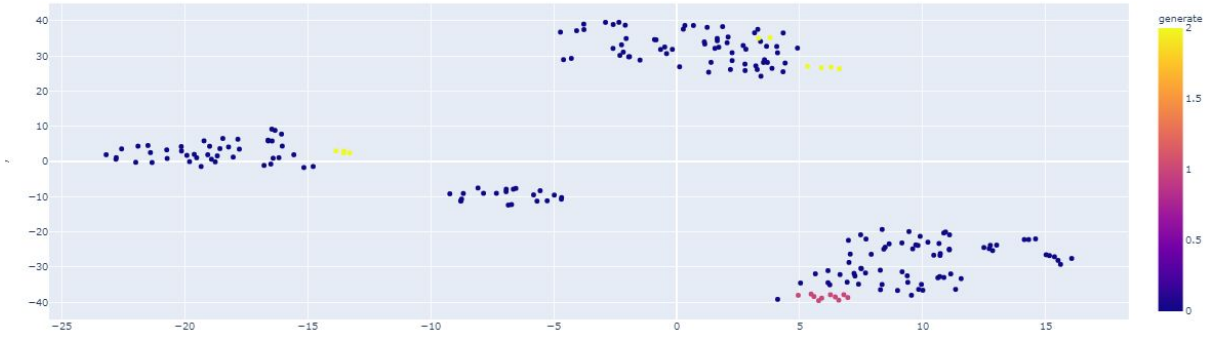
Parameters/Values				
Batch Size	**64**	**128**	**256**	
Learning Rate	**2e-4**	**5e-4**	**1e-3**	
Noise size	**16**	**32**	**64**	**128**
Dropout	**0**	**0.2**	**0.5**	**0.7**

We got the best results with dropout of 0, noise size of 32, learning rate of 2e-4 and batch size of 128.

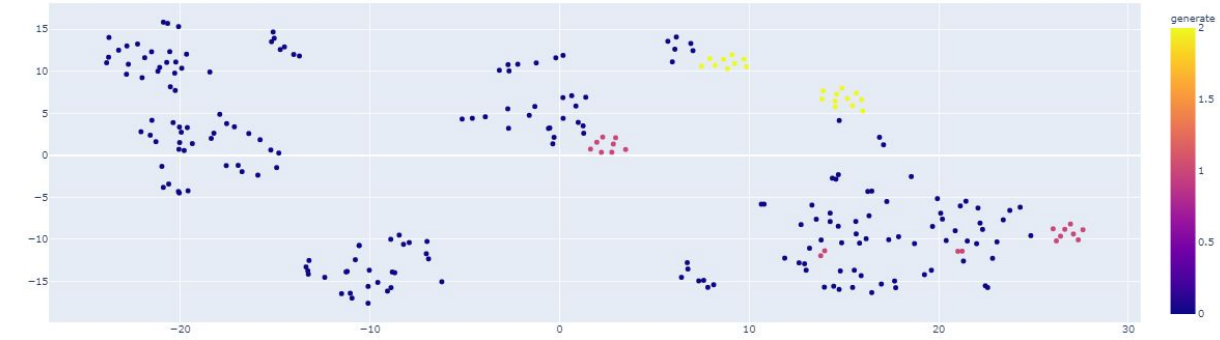
7.2 Samples that fooled the discriminator

during the training process we collect some samples that success to fooled the discriminator and some not in order to see if the fooled samples have features in common with the real examples
We chose 200 random samples from the original dataset and 20 samples that “fooled” the detector and 20 samples that did not.
and visualize the data with TSNE and want to understand the original data with the “fooled” samples and comment feature.
It seems like the datasets are split into a couple of groups and one group very much into the “fooled” samples.

Adult Dataset:



Bank Dataset

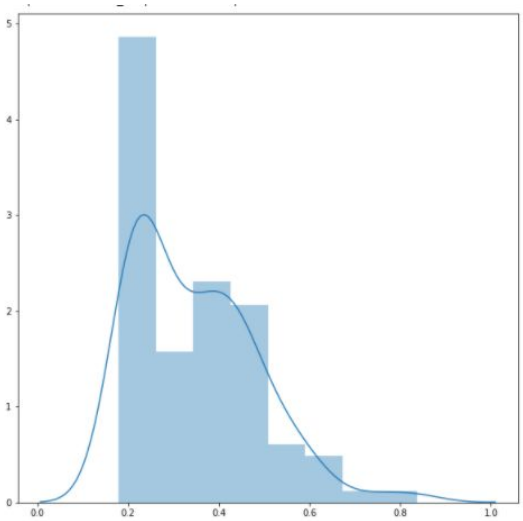


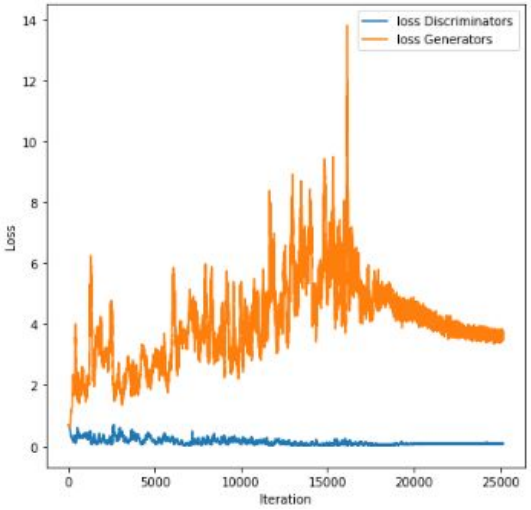
7.3 Data Generation

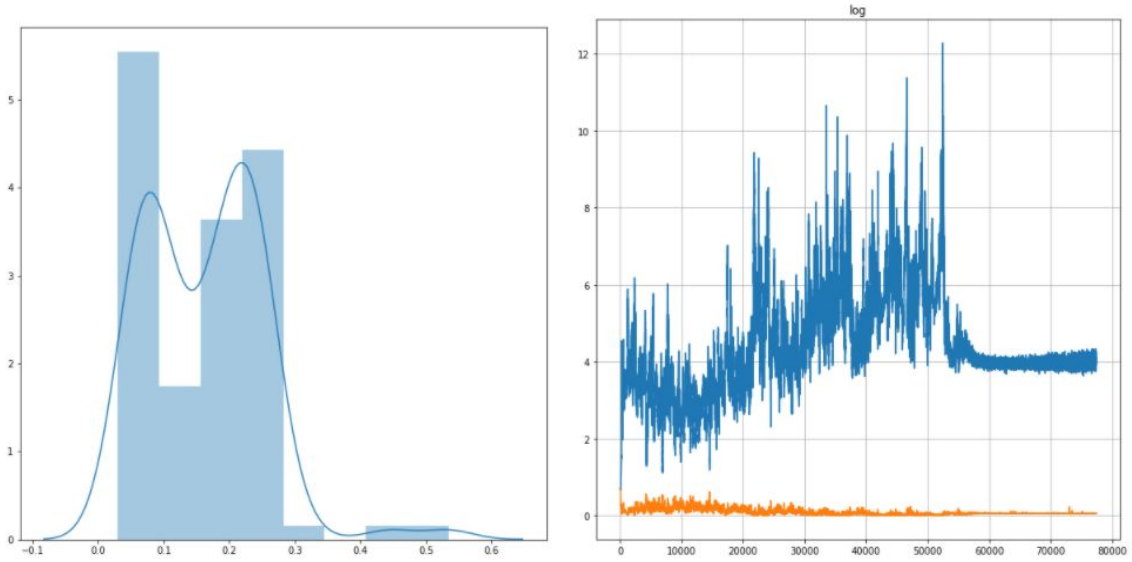
Once your models are converged, generate 100 samples at random. How many were able to pass as real samples?
Once the models are converged we generate 100 samples at random.

The 100 samples attached in file name “generate_samples.csv”
to decide if sample pass the model or not we feed the discriminator and ask to predict if the

generated samples are “fake” or “real”. The threshold to decide is 0.5. from the 100 sample - 14 pass as real sample.







The fight between the discriminator and the generator is very noticeable.

Ideally, the generator should receive large gradients early in the training because it needs to learn how to generate real-looking data. The discriminator on the other hand does not always get large gradients early on, because it can easily distinguish real and fake images. Once the Generator has been trained enough, it becomes harder for the discriminator to tell apart real from fake images. It would keep making errors and get strong gradients. It can be seen that at first the models "go back
and forth" (the loss increases and decreases) and then slowly they move forward during “fight” with each other.
When it can be seen that the discriminator is a consistent leader.

Part 2 – Generative model for sample generation

8. Model, Experiments and Results (Part 2)

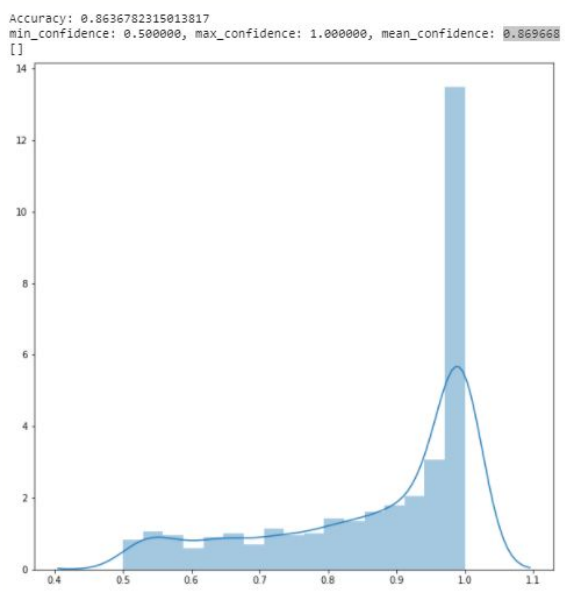
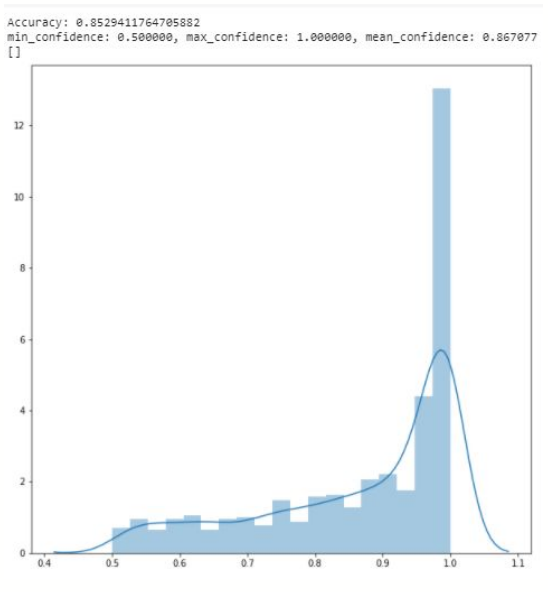
At this part we train a generative model to understand the inner workings of a black-box model by simultaneously generating samples and predicting the score they will receive from the model.

8.1 Random Forest model - “Black Box Model”

To simulate a black box we used sklearn's Random Forest Classifier and we made the same Preprocessing on the data in part 1 and fit the Random Forest Models for each dataset.
The dataset split to 90% for training and 10% for test.

Random Forest Result

Dataset	Accuracy	Mean confidence	Minimum confidence	Maximum confidence
adult	0.86 (+/- 0.01)	0.869668	0.5	1
bank-full	0.75 (+/- 0.28)	0.900944	0.5	1

Adult	**Bank**
	

8.2 The Generative Model

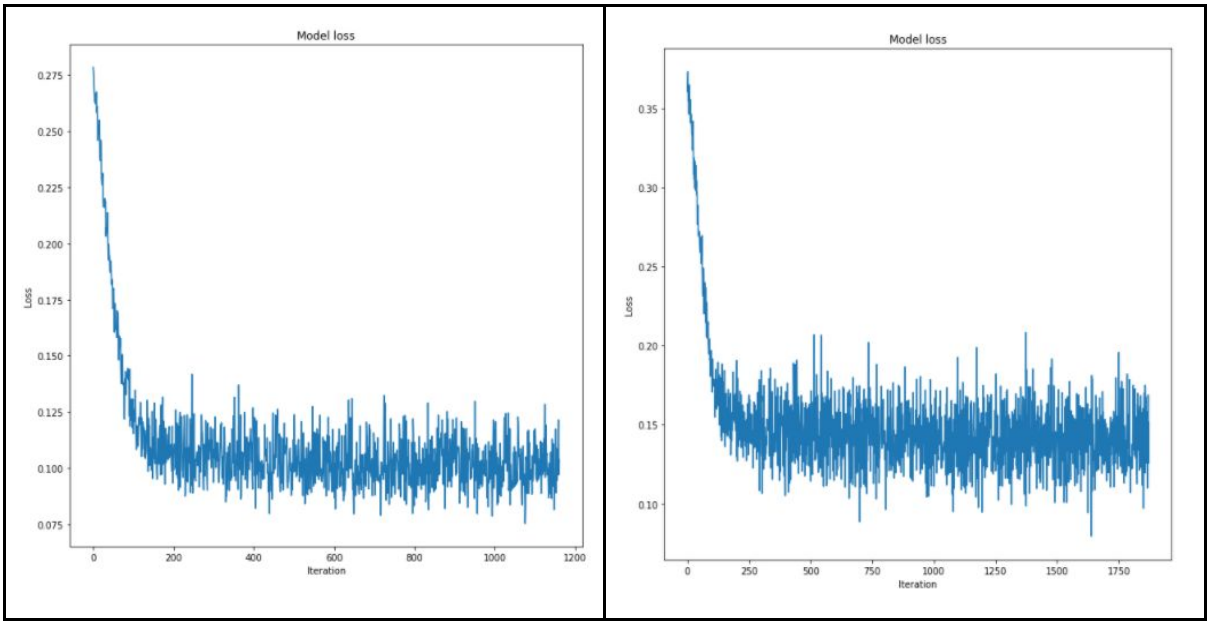
we architecture of the model is the very similar to part 1 and the input size are different:
The input of the generative model:

1. random noise
2. the desired confidence score of the sample

The loss function: MAE - Mean Absolute Error that tries to minimize the mean of absolute difference between the confidence score provided input to the generative model and the confidence score assigned by the Random Forest Model.

8.3 Model Performance

Adult	**Bank**



The model coverage is very fast and the coverage score stands between 0.1-0.15. There are some rationale reason for that; We think it may happen because of vanishing gradient.

The confidence scores of the generated samples is uniformly sampled from [0,1].

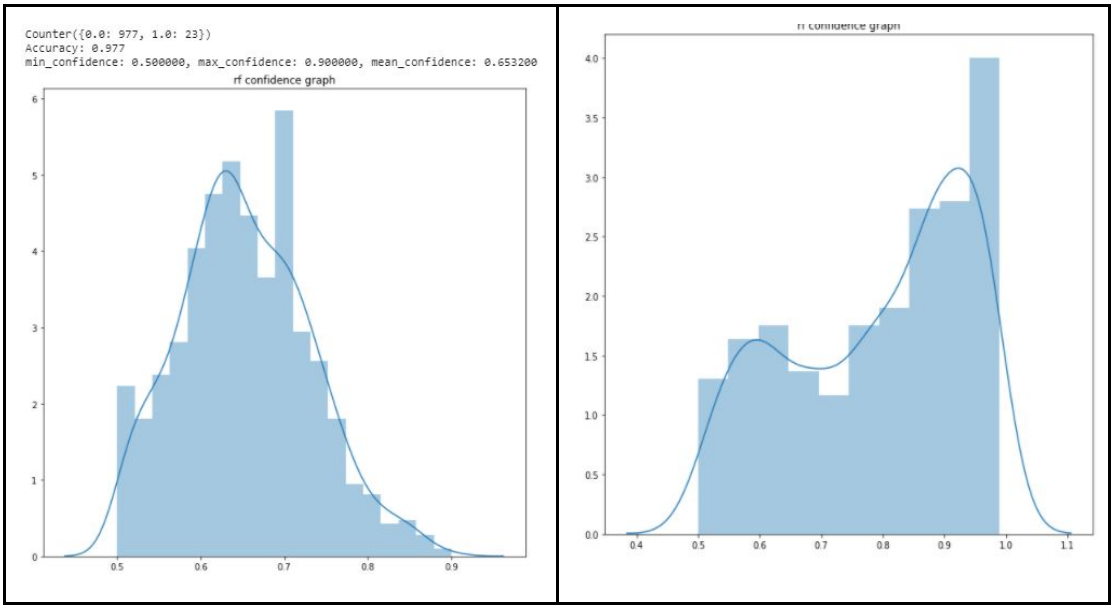
The **generated samples** were added in the folder attached for submission in:
generate_data/part2/adult_part2_genarate1000_samples.csv
generate_data/part2/bank_part2_genarate1000_samples.csv

Feed the generated samples to the black-box model and provide statistics on the score distributions. Are there any conclusions you can draw? Were you more successful for a specific class of samples than another? Were you more successful for a specific range of confidence scores?

8.4 Confidence Score Distribution

the confidence score distribution that generated samples feed the black box model:

Adult	**Bank**



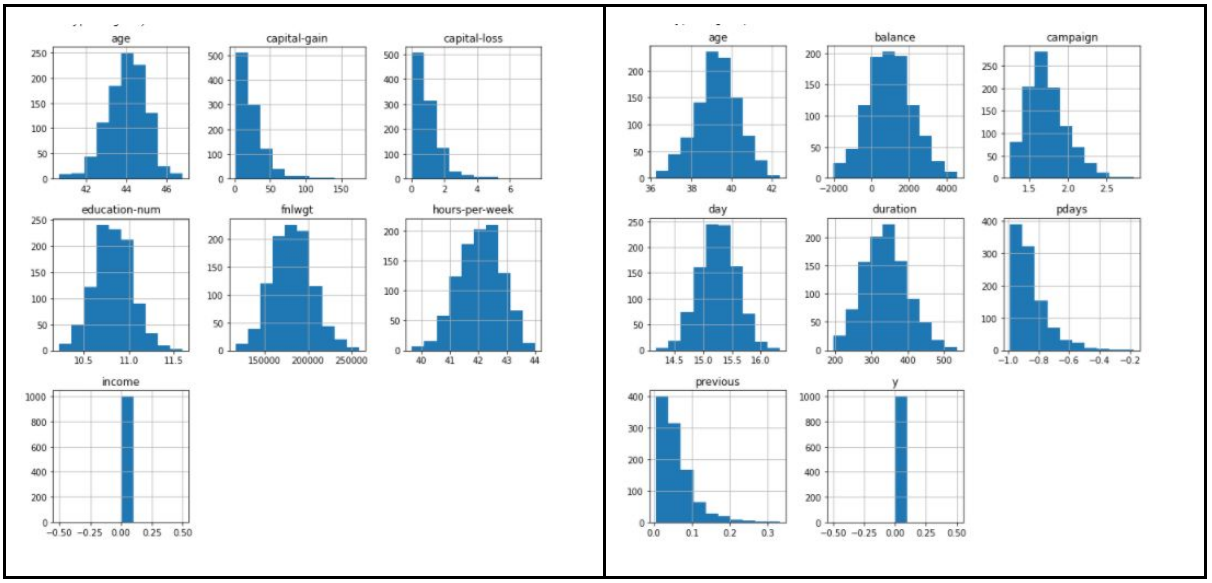
we can see the black model predict most of the sample as one class type in each of the data-sets. more than that, the accuracy in what the generative model samples and the black box prediction was 0.91 of success.
Some samples missed the black box prediction but it looks like good predictions.

Success for a specific range of confidence score - we can see that the most confidence score is around the 0.7 score when the maximum get to 0.95 and the minimum is 0.5.

8.5 Mode Collapse and Other Challenges
There is a problem of mode collapse in our GANs for both datasets, although we used SMOTE to deal with the imbalanced data.

However, if a generator produces an especially plausible output, the generator may learn to produce only that output. In fact, the generator is always trying to find the one output that seems most plausible to the discriminator. but here we don't have any discriminator that can tell the generator that it missed.
its also look like the generator generate one type of value in some feature and And this can be caused by the lack of diversity in the data.
Here are the distribution of the numerical values of the generated data -

Adult	**Bank**



9. References

1. Modeling Tabular Data using Conditional GAN
2. Dropout-GAN: Learning from a Dynamic Ensemble of Discriminators
3. Improved Techniques for Training GANs(Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen)
4. Tips for Training Stable Generative Adversarial Networks
5. https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/
6. Effective data generation for imbalanced learning using Conditional Generative Adversarial Networks
7. GAN-SMOTE
8. Why is it hard to train GANs?