Deep Learning HW4 – Report

*Saar Ben David* *– 305560591*

*Ido Rabia – 201181450*

**Summary:**

In this task, we have implemented two types of GAN models, a standard version and one with a black-box classifier. For each version we have created a different class with its own implementation. melody-embedding of the original song. Both GANs were trained and evaluated on two columnar datasets.

**NOTE:** To run the code, we expect the diabetes/german-credit files to be located in the same folder of the main.py file.

**Dataset:**

In this task, we used 2 datasets

1. german\_credit: contains 1000 records of credit data. 20 columns, most of them categorical.
2. diabetes: contains 767 records of diabetes data. 8 columns, all of them are numeric.

**Preprocessing:**

**Categorical columns:**

We used One-Hot-Encoder to transform the categorical columns in both datasets to numeric.

**Numeric/Continuous columns:**

Since we are using gradient descent to train our NN, we normalized our numeric columns. For that task we used Sklearn MinMaxScaler.

**Part 1:**

In this part we implemented a simple GAN model.

**Model Architecture:**

In our work, we decided to use a dense architecture for the Discriminator and the Generator. Due to different complexity and input size in the two datasets, we came up with a different GAN architecture for each dataset. We set the output size of the generator to be equal to the dimension of the real samples. Here we demonstrate the architecture of the entire GAN:

**german\_credit:**

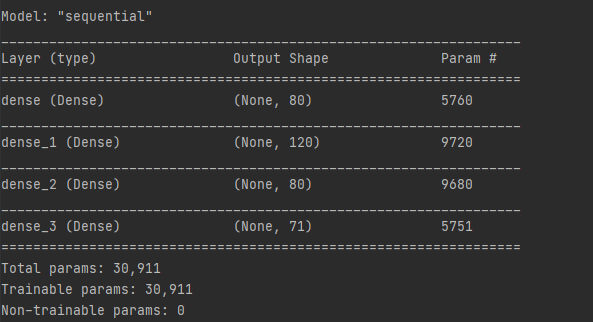
|  |  |  |
| --- | --- | --- |
|  | Discriminator | Generator |
| Type of NN | Dense | Dense |
| Number of layers | 5 | 4 |
| Nodes in each layer | 80, 40, 20, 10, 5, 1 | 80, 120, 80, 71 |
| Activation function in each layer | Relu, Relu, Relu, Relu, sigmoid | Relu, Relu, Relu, sigmoid |

**diabetes:**

|  |  |  |
| --- | --- | --- |
|  | Discriminator | Generator |
| Type of NN | Dense | Dense |
| Number of layers | 5 | 4 |
| Nodes in each layer | 50, 30, 15, 10, 5, 1 | 50, 70, 50, 8 |
| Activation function in each layer | Relu, Relu, Relu, Relu, sigmoid | Relu, Relu, Relu, sigmoid |

The "german credit" model architecture:

Generator model:

****

Discriminator model:

**Table

Description automatically generated with medium confidence**

The "diabetes" model architecture:

|  |
| --- |
| Generator model:    Discriminator model: |

**Experimental Setup:**

We used the same hyper-parameters for both models. We used the following configurations:

* batch\_size = 256
* epoch = 5000
* patience = 1000
* optimizer was set to Adam with learning rate of
* loss function: the loss of the GAN was calculated as:

We have set the "patience" parameter to stop the training after 1000 consequent epochs without improvement in the loss of the GAN.

In addition, we use the formula above to calculate the GAN's loss, defined as the summation of the generator loss and the discriminator loss, plus a punishment-factor for the gap in performance between the two (we seek equal performance of G and D).

**Training the Model:**

Both models were trained in the same way. First, we sampled half batch of real samples and completed the other half with fake samples generated by the generator. Then, we trained our GAN in two steps:

1. Trained only the discriminator on this batch of mixed samples (fake and real). The samples were labeled as 0 for fake samples and 1 for real samples.
2. Froze the discriminator weights and fed the GAN only with the fake samples' noise. We have done it so the generator will generate the exact same samples the discriminator trained on in the first step. This time, we labeled the fake samples as 1. The idea is that the loss of the generator will be proportional to degree of confidence of the discriminator.

We tried to keep the generator and the discriminator competitive by letting to the one with bigger loss an extra training. We saved the model parameters of the best epoch, we stopped the training when the GAN didn't improve for 1000 epochs.

**Results:**

**Training loss value:**

We will present the loss of the generator and the discriminator for each of our models.

*German credit:*

|  |
| --- |
| Generator loss:    Discriminator loss: |

As we can see from the graph of the loss function the models went “back

and forth” with their losses. In addition, the discriminator was a consistent leader during the training process.

*Diabetes*:

|  |
| --- |
| Generator loss:    Discriminator loss: |

As we can see from the graph of the loss function the models went “back

and forth” with their losses. In addition, the discriminator was a consistent leader during the training process.

**Examples of generated samples:**

We will present here examples of generated samples for each of our models, the examples will include several samples which “fooled” the detector and several that did not.

*German credit:*

Here we present samples which “fooled” the detector:

Samples which did not “fooled” the detector:

We analyzed the samples which were presented above and figured that\_\_\_

*Diabetes*:

Here we present samples which “fooled” the detector:

Samples which did not “fooled” the detector:

We analyzed the samples which were presented above and figured that\_\_\_

**Evaluation:**

We used 2 different methods to evaluate our models - Word2Vec and Doc2Vec. In each evaluation method, we checked the similarity between the original song and the generated songs of one of our models. Then we compared the performance of our models based on these evaluation results

1. Word2Vec:

For each generated song, we looked at the word-embeddings of its words. Then, we calculated the **pairwise** cosine-similarity of each word with the corresponding word from the original song, it finishes when one of the songs ends. This method shows how well the model managed to infer the local context of the original song.

The graphs below describe the average and the standard deviation of cosine similarity for each of our model. The average was done over 15 different generated songs.

|  |
| --- |
| With "EOL" & "EOS" Without "EOL" & "EOS"  Chart, bar chart  Description automatically generatedChart, bar chart  Description automatically generated |

We can see that in both graphs the average values are very high, although the generated text is far from being similar to the original. In our opinion, The causes for this phenomenon are:

* The Word2Vec model was trained on a very little corpus (~7000 words).
* Both models use words like 'you', 'me', 'are', 'them' a lot of times. These words' vectors reside closely in the latent space of the trained Word2Vec model, thus the high similarity.

The table below shows the exact values of the average and standard deviation of this evaluation technique.

|  |  |  |
| --- | --- | --- |
|  | "Simple" | "Complex" |
| With "EOL" & "EOS" | 0.9207 ± 0.0642 | 0.9649 ± 0.0759 |
| Without "EOL" & "EOS" | 0.9141 ± 0.0247 | 0.9211 ± 0.0281 |

1. Doc2Vec:

Here we looked at the document-embedding of each generated song. Then, we calculated the cosine-similarity between the latent vector of the generated song to that of the original song. This method shows how well the model managed to infer the global context of the original song.

The graphs below describe the average and the standard deviation of cosine similarity for each of our model. The average was done over 15 different generated songs.

|  |
| --- |
| With "EOL" & "EOS" Without "EOL" & "EOS"  Chart, bar chart  Description automatically generatedChart, box and whisker chart  Description automatically generated |

We can see that when we do this evaluation with 'EOL' and 'EOS' included, the results show high similarity on both models. When we don't include 'EOL' and 'EOS' in the evaluation, the results drop dramatically into a reasonable range of values (in respect to the actual resemblance of the songs).

The table below shows the exact values of the average and standard deviation of this evaluation technique.

|  |  |  |
| --- | --- | --- |
|  | "Simple" | "Complex" |
| With "EOL" & "EOS" | 0.8316 ± 0.0671 | 0.8259 ± 0.0613 |
| Without "EOL" & "EOS" | 0.4739 ± 0.1328 | 0.3415 ± 0.0767 |

These results make sense due to the high occurrence of the 'EOL' word in all of our songs. This high occurrence deludes the Doc2Vec model to "think" that the songs have similar context, even though their content may be far from similar.

From these results we can infer that learning the structure of the song impairs the ability of the model to learn the context of the song.

**Part 2:**

In this part we implemented a RandomForest model as a "black box model" and combining it to the GAN model in that way:

|  |
| --- |
| Diagram  Description automatically generated |

**Model Architecture:**

In our work, we decided to use a simple RandomForest model in addition to the GAN architectures from part 1, the GAN models have been changed for adjusting to that assignment.

*German credit:*

|  |  |  |
| --- | --- | --- |
|  | Discriminator | Generator |
| Type of NN | Dense | Dense |
| Number of layers | 5 | 4 |
| Nodes in each layer | 80, 40, 20, 10, 5, 1 | 80, 120, 80, 72 |
| Activation function in each layer | Relu, Relu, Relu, Relu, sigmoid | Relu, Relu, Relu, sigmoid |

*Diabetes:*

|  |  |  |
| --- | --- | --- |
|  | Discriminator | Generator |
| Type of NN | Dense | Dense |
| Number of layers | 5 | 4 |
| Nodes in each layer | 50, 30, 15, 10, 5, 1 | 50, 70, 50, 9 |
| Activation function in each layer | Relu, Relu, Relu, Relu, sigmoid | Relu, Relu, Relu, sigmoid |

The "german credit" model architecture:

Picture from python

The "diabetes" model architecture:

Picture from python

**Training the Model:**

Both models were trained on the same way. First, we sampled half of batch size number of real samples and generated the same number of fake samples by the generator. Then we concatenated those samples train the discriminator on this batch. After that, we trained only the generator by bringing to the GAN model batch of noise samples and labeled them as true samples, the loss of the generator proportional to degree of confidence of the discriminator while classifying sample as a fake one. While training of the generator we froze the discriminator NN.

We tried to keep the generator and the discriminator competitive by letting to the one with bigger loss an extra training. We saved the model parameters of the best epoch, we stopped the training when the GAN didn't improve for 1000 epochs.

**Experimental Setup:**

We used the same hyper-parameters for both models. We used the following configurations:

* batch\_size = 1000
* epoch = 20000
* optimizer was set to Adam with learning rate of
* regulator – dropout with 10% of frozen nodes
* loss function: sparse categorical cross-entropy

**Results:**

**Training loss value:**

We will present the loss of the generator and the discriminator for each of our models.

*German credit:*

|  |
| --- |
|  |

As we can see from the graph of the loss function the models went “back

and forth” with their losses. In addition, the discriminator was a consistent leader during the training process.

*Diabetes*:

|  |
| --- |
|  |

As we can see from the graph of the loss function the models went “back

and forth” with their losses. In addition, the discriminator was a consistent leader during the training process.

**Examples of generated samples:**

We will present here examples of generated samples for each of our models, the examples will include several samples which “fooled” the detector and several that did not.

*German credit:*

Here we present samples which “fooled” the detector:

Samples which did not “fooled” the detector:

We analyzed the samples which were presented above and figured that\_\_\_

*Diabetes*:

Here we present samples which “fooled” the detector:

Samples which did not “fooled” the detector:

We analyzed the samples which were presented above and figured that\_\_\_

**Evaluation:**

We used 2 different methods to evaluate our models - Word2Vec and Doc2Vec. In each evaluation method, we checked the similarity between the original song and the generated songs of one of our models. Then we compared the performance of our models based on these evaluation results

1. Word2Vec:

For each generated song, we looked at the word-embeddings of its words. Then, we calculated the **pairwise** cosine-similarity of each word with the corresponding word from the original song, it finishes when one of the songs ends. This method shows how well the model managed to infer the local context of the original song.

The graphs below describe the average and the standard deviation of cosine similarity for each of our model. The average was done over 15 different generated songs.

|  |
| --- |
| With "EOL" & "EOS" Without "EOL" & "EOS"  Chart, bar chart  Description automatically generatedChart, bar chart  Description automatically generated |

We can see that in both graphs the average values are very high, although the generated text is far from being similar to the original. In our opinion, The causes for this phenomenon are:

* The Word2Vec model was trained on a very little corpus (~7000 words).
* Both models use words like 'you', 'me', 'are', 'them' a lot of times. These words' vectors reside closely in the latent space of the trained Word2Vec model, thus the high similarity.

The table below shows the exact values of the average and standard deviation of this evaluation technique.

|  |  |  |
| --- | --- | --- |
|  | "Simple" | "Complex" |
| With "EOL" & "EOS" | 0.9207 ± 0.0642 | 0.9649 ± 0.0759 |
| Without "EOL" & "EOS" | 0.9141 ± 0.0247 | 0.9211 ± 0.0281 |

1. Doc2Vec:

Here we looked at the document-embedding of each generated song. Then, we calculated the cosine-similarity between the latent vector of the generated song to that of the original song. This method shows how well the model managed to infer the global context of the original song.

The graphs below describe the average and the standard deviation of cosine similarity for each of our model. The average was done over 15 different generated songs.

|  |
| --- |
| With "EOL" & "EOS" Without "EOL" & "EOS"  Chart, bar chart  Description automatically generatedChart, box and whisker chart  Description automatically generated |

We can see that when we do this evaluation with 'EOL' and 'EOS' included, the results show high similarity on both models. When we don't include 'EOL' and 'EOS' in the evaluation, the results drop dramatically into a reasonable range of values (in respect to the actual resemblance of the songs).

The table below shows the exact values of the average and standard deviation of this evaluation technique.

|  |  |  |
| --- | --- | --- |
|  | "Simple" | "Complex" |
| With "EOL" & "EOS" | 0.8316 ± 0.0671 | 0.8259 ± 0.0613 |
| Without "EOL" & "EOS" | 0.4739 ± 0.1328 | 0.3415 ± 0.0767 |

These results make sense due to the high occurrence of the 'EOL' word in all of our songs. This high occurrence deludes the Doc2Vec model to "think" that the songs have similar context, even though their content may be far from similar.

From these results we can infer that learning the structure of the song impairs the ability of the model to learn the context of the song.

**Conclusion:**

In our opinion, even though the results of both models are pretty close according to the evaluations we did, the "simple" model (slightly) outperforms the "complex" model in the Doc2Vec evaluation technique. Adding that to the fact that it took half the time to train the "simple" model makes it superior to the other.