Report on GAIN: Generative Adversarial Imputation Nets

Laura

2025-02-26

```r
rm(list = ls())

reticulate::use_condaenv("tf-env", required = T)

library(reticulate)
library(tidyverse)
library(readr)
library(keras)
library(tcltk)
library(tensorflow)
library(dplyr)
library(MASS)
library(holodeck)
library(purrr)
library(mice)
```

Introduction

Missing data is a pervasive challenge in data analysis, often leading to biased estimates and reduced statistical power when handled improperly. Traditional methods such as mean imputation or multiple imputation (e.g., via the `mice` package) can fall short in capturing complex data distributions especially with high dimensional data.

Generative Adversarial Networks (GANs), introduced by Goodfellow et al., is made up of two neural networks, a **generator** and a **discriminator**, which compete against each other in a minimax game. In the standard GAN framework, the generator creates synthetic data samples from random noise, while the discriminator aims to distinguish between real and generated data. The discriminator maximizes the probability of correctly classifying real and fake inputs where the output is a probability that quantifies how likely it is that the given input is real.

Generative Adversarial Imputation Nets (GAIN) extends this framework to the realm of data imputation. Instead of generating entirely new data, GAIN focuses on predicting missing entries in partially observed datasets. The generator in GAIN imputes missing values conditioned on the observed data, while the discriminator is tasked with distinguishing between observed and imputed (i.e., generated) values. This setup is further enhanced by the introduction of a *hint mechanism*, which provides additional information to the discriminator about the missingness pattern, ensuring a more robust imputation process.

A key aspect of GAIN is its loss functions, where the discriminator loss is defined as:

$$\mathcal{L}_D = -\mathbb{E}_{\mathbf{X},\mathbf{M}}\Big[\mathbf{M} \odot \log D(\hat{\mathbf{X}}) + (1-\mathbf{M}) \odot \log\big(1 - D(\hat{\mathbf{X}})\big)\Big],$$

while the generator loss combines the adversarial component with a reconstruction term:

$$\mathcal{L}_G = -\mathbb{E}_{\mathbf{X},\mathbf{M}}\Big[(1-\mathbf{M}) \odot \log D(\hat{\mathbf{X}})\Big] + \alpha\, \mathcal{L}_{\text{recon}}.$$

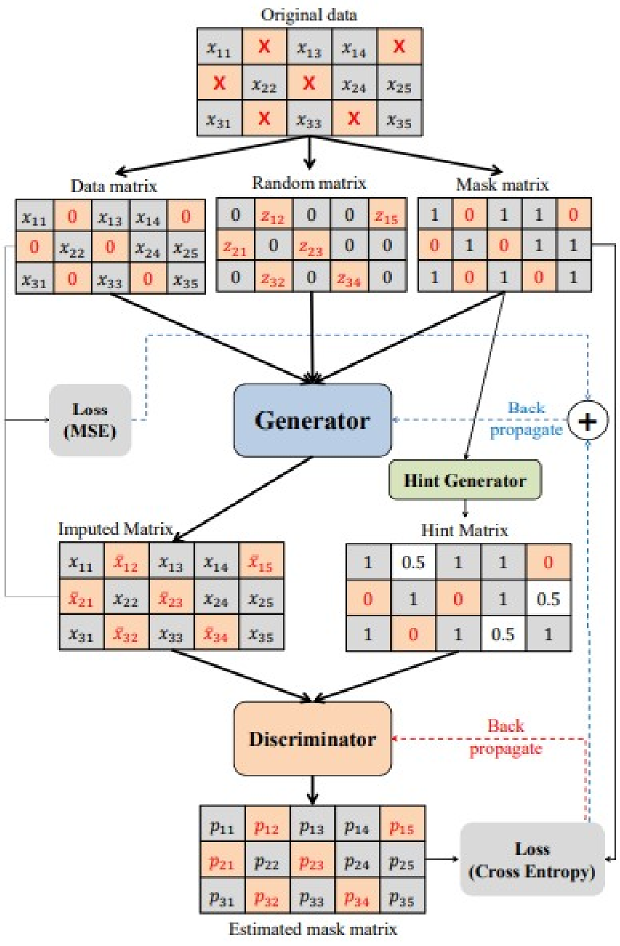


Figure 1. The architecture of GAIN

Figure 1 illustrates the architecture of GAIN, showing how the generator and discriminator interact with the data and the hint matrix.

In this report, we compare the R implementation of GAIN with the performance of the original GAIN implementation (Yoon et al.) and lastly compare its performance with the `mice` package. The aim is to provide a comprehensive understanding of how GAIN functions and its potential advantages for imputation tasks.

RGAIN and GAIN comparison

We now try to test GAIN on a data set provided by Yoon and used in the original paper. The data set in question is `letter`, consisting of 16 categorical variables and 20000 observations. We also download the `letter_miss` data set, which is the `letter` data set with 20% missing variables. We need this data set in order to compare the results with the python implementation, as this will make the two software impute the same data set.

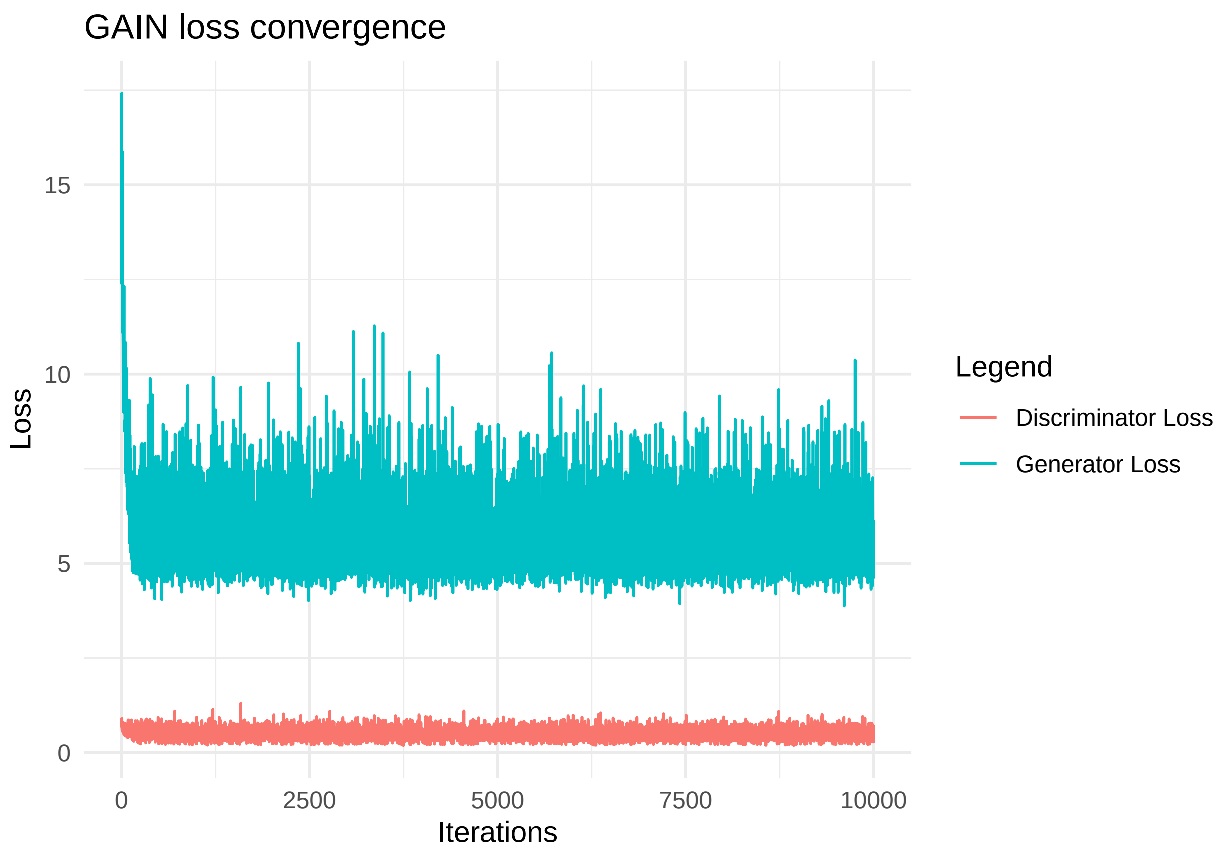
```r
# Download the datasets
letter <- read.csv("letter.csv")
letter_miss <- read.csv("letter_miss.csv")
```

We first test the GAIN with the `gain_paper` function. We obtain the graph of the convergence of the loss functions, which would seem to indicate that the functions converge much earlier than the 100000 interactions recommended by Yoon. We also obtain the RMSE which is equal to **0.1806**.

```r
# Set random seed for reproducibility
set.seed(130225)

# Impute missing values using the GAIN algorithm
imputed_paper <- gain_paper(letter)
```

```
## Iteration 100    Gloss 7.194135  Dloss 0.5653945Iteration 200    Gloss 4.933542  Dloss 0.5355507Itera
##  RMSE: 0.1801695
```



This code performs data imputation using the GAIN algorithm and evaluates its performance by computing the Root Mean Squared Error (RMSE). The core imputation step is executed by calling `gain_missing(letter_miss, letter)`, where `letter_miss` represents the data set with missing values, and `letter` is the original data set. The function imputes missing values using GAIN, returning a completed data set with estimated values.

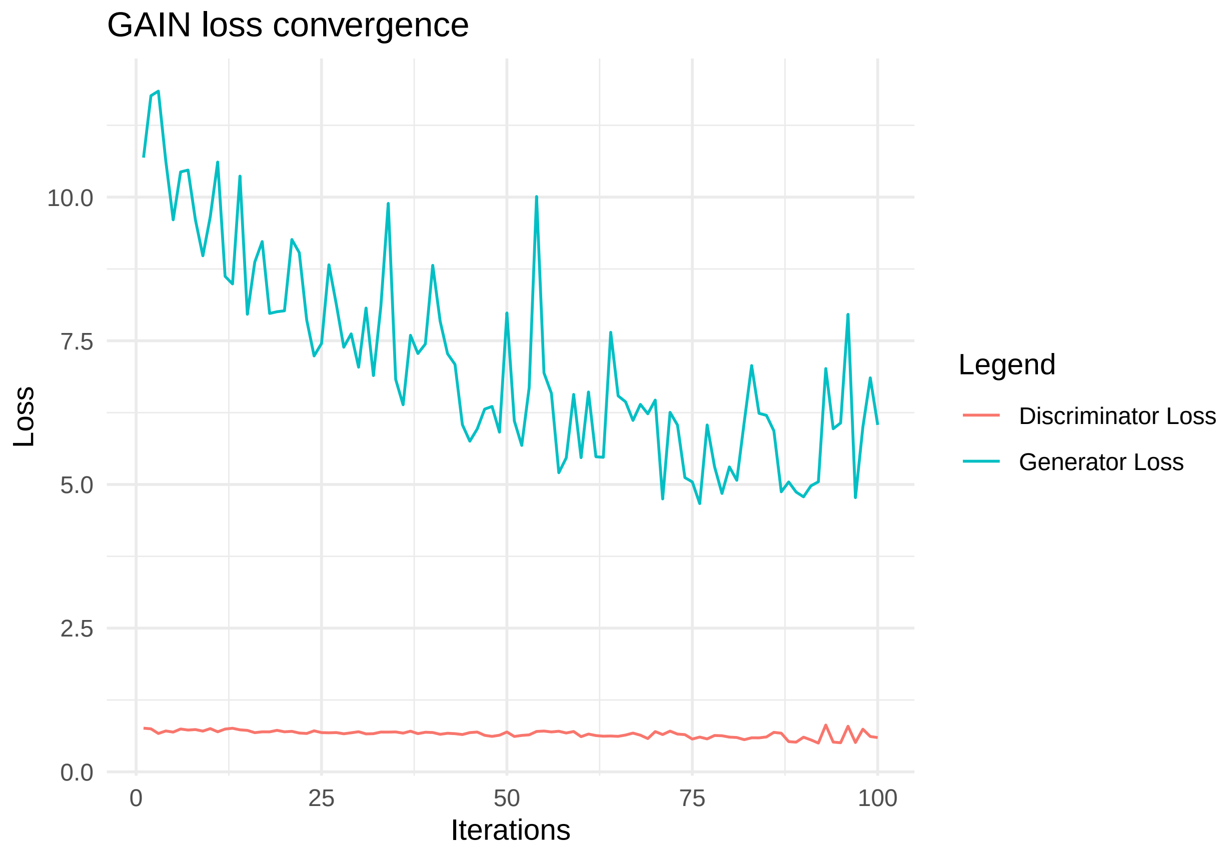
To compare different imputation approaches, an imputed data set named `imputed_letter` is loaded from a CSV file (`imputed_letter_missing.csv`). This data set has been pre-imputed using the original Python-based GAIN implementation by Yoon. We create the **mask matrix**, then the RMSE is computed using `rmse_loss`..

The RMSE for our R implementation is **0.1793**, which aligns with the previous since the data set and missing percentage remain unchanged. However, when comparing with the Python version, a discrepancy appears: the Python-imputed data set achieves a **RMSE of 0.1278**, lower than our result. This suggests that the Python implementation might optimize better, indicating a potential difference in how the two versions function.

```
# Set random seed for reproducibility
set.seed(130225)

# Impute missing values using the GAIN algorithm
imputed_missing <- gain_missing(letter_miss, letter)
```

```
## Iteration 100    Gloss 6.036734  Dloss 0.5952262
##  RMSE: 0.1868454
```



```
# Read the imputed data from a CSV file
imputed_letter <- read_csv("imputed_letter_missing.csv", col_names = F)

# Create a mask for the missing values in the original dataset
mask <- find_mask(letter_miss)

# Calculate the RMSE between the original and imputed datasets using the mask
rmse <- rmse_loss(letter, imputed_letter, mask)

# Print the RMSE value
cat("\n RMSE of the data imputed with Python:", rmse)
```

```
##
##  RMSE of the data imputed with Python: 0.1277718
```

Mice comparison

In this part we will compare the `RGAIN` implementation the common multiple imputation technique `mice`. First, we will compare the performance of the `RGAIN` with the performance of `mice` using the same data provided above, namely `letter`. Afterwards we will also do a small comparison using a generated data. Impute letter data using mice [@buurenMiceMultivariateImputation2011].

```r
data_letter <- read.csv("letter_miss.csv")
letter <- read.csv("letter.csv")
```

making the mask

```r
# for the letter data we have already missingness so we are making a mask out of it
# Create a mask from the dataset: 1 for observed values, 0 for missing values
mask_letter <- ifelse(is.na(data_letter), 0, 1)
```

Imputing missing data

```r
set.seed(130225)
library(mice)
mice_imputed_letter <- mice(data_letter, m = 1, maxit = 5)
imputed_m_letter <- complete(mice_imputed_letter)
```

Calculate rmse

```r
# this function calculate the RMSE
rmse_loss <- function(ori_data, imputed_data, data_m){

  #I changed this part becuase I was running into a problem since i already normalized so with the prev

  # Ensure that the original data and imputed data are already normalized
#  if (is.null(parameters)) {
#    norm_result <- normalization(ori_data)
#    parameters <- norm_result$parameters
#  }
  # Calculate parameters only if not provided

  # Normalize the data if parameters are not passed (initial normalization)
#  ori_data <- normalization(ori_data, parameters)
#  imputed_data <- normalization(imputed_data, parameters)
```

```
  # Only for missing values
  nominator <- sum(((1-data_m) * ori_data - (1-data_m) * imputed_data)**2)
  denominator <- sum(1-data_m)

  rmse <- sqrt(nominator/denominator)

  return(rmse)
}
```

```
# Calculate RMSE

#mask_rmse_letter <- ifelse(is.na(mask_letter), 0, 1)  # Convert NA to 0, keeping valid entries as 1

# Calculate RMSE again
rmse_letter <- rmse_loss(letter, imputed_m_letter, mask_letter)
print(paste("RMSE:", rmse_letter))
```

```
## [1] "RMSE: 2.29459688593576"
```

For the same data set (letter) using the same mask used by main_gain() the RMSE is 2.29 when using mice() to impute missing data, whereas RGAIN has a RMSE around 0.18. In this example, mice is outperforming RGAIN Now we will compare the results with the generated data.

Generated data

To see how well the RGAIN performs in imputing missing data, we will generate some simple data that is continuous. This data is generated using the mvrnorm function from the package MASS obtaining a multivariate normal data set. This data set has 4 variables, where the correlations between those variables are defined in `popcor`.

```
gen_data <- read.csv("gen_data.csv")


#norm_g <- normalization(gen_data)
#norm_gen <- norm_g$norm_data
#parameters_gen <- norm_g$parameters  # parameters is a list, containing min_val and max

#norms_gen <- normalization(gen_data, parameters_gen)
```

Introduce missingness

```
# Introduce missing values by using the same mask generated above
# Convert 0's in the mask to NA
#mask_gen[mask_gen ==0] <- NA

#data_m_gen <- gen_data * mask_gen
```

Imputing missing data

```r
#set.seed(123)

#mice_imputed_gen<- mice(data_m_gen, m = 1, maxit = 5)
#imputed_data_gen<- complete(mice_imputed_gen)
```

```r
# Calculate RMSE
#mask_rmse_gen <- ifelse(is.na(mask_gen), 0, 1) # Convert NA to 0, keeping valid entries as 1

# Calculate RMSE again
#rmse_gen <- rmse_loss(gen_data, imputed_data_gen, mask_rmse_gen)
#print(paste("RMSE:", rmse_gen))
```

Using mice on the generated data set, using the same mask as it was created for the RGAIN, we obtain a RMSE of 2.015, compared to the RMSE obtained using the RGAIN of 0.418.
Looking at the results, where we compared the performance of mice and RGAIN, we can see that `RGAIN` is outperforming `MICE` on both data sets (generated data and letter data) when using the RMSE to evaluate imputation performance. An alternative way to evaluate would be by comparing the distributions of the original and the imputed data (there are a few more). It is also of interest to make the RGAIN work with categorical data, and try that out. Additionally, it would also be of interest to make such a comaprison on different type of data, potentially high dimansional data with many more columns:)

References