Homework 4

Nikolaj Takata Mücke

07-03-2024

Due date: 28th of March, 2024

Submission via email: nikolaj.mucke@cwi.nl

Submissions should consist of a *short* `.pdf` report of your findings, and an offline, compilable copy of the jupyter notebook.

In this homework, we are dealing with Generative Adversarial Networks (GANs). The aim is to train GANs to generate various types of data and analyse the quality of the generated distributions. This exercises in this homework is not as strictly defined as the previous homeworks. Many of the exercises can be solved and implemented in different ways. You will also experience that the results will be less "clean". This is due to the fact that GANs are significantly more difficult to train and hyperparameter tune than supervised neural networks. It is important that you describe your choices well and argue why you made the choices you did. Due to the potentially long tuning phases, we will accept poorer results than in the previous homeworks, as long you discuss what went wrong and why.

In all exercises, discuss and elaborate on the following points:

- Architecture choices;
  - Do you use dense layers? Or convolutional layers? Or something else?
- Hyperparameter choices;
- How you set up and monitor the training;
- Quality of the generated distributions;
  - The quality of distributions can be assessed both qualitatively and quantitatively. For example, a qualitative assessment can be performed by visual inspection of the individual samples or comparing the distribution of a collection of samples and a quantitative assessment can be performed by computing divergences and comparing moments of the distributions. You are expected to use an assessment that is suitable and feasible for the individual exercises;
- Comparison of the GAN and the Wasserstein GAN.

If you repeat practices in several exercises, it is okay to refer to a previous explanation.

Your report will also be judged based general best practices in machine learning. So remember everything you've learned from the past weeks!

Ex. 1 (Generating Gaussian distributions)

In this exercise you have to code a GAN and a Wasserstein GAN to generate data from 6 Gaussian distributions lying on circle. The code to generate the training data is shown below.

Ex. 2 (Generating financial time series)

Code and train a GAN and a Wasserstein GAN to generate financial time series. Specifically, use the daily closing price of various assets over several periods of time. The code for loading the financial time series is shown below. You are allowed to modify the code as much as you like. For example, you can add more assets, change the time intervals, add more itnervals, etc.

Ex. 3 (Conditional GAN for Generating handwritten digits)

The aim of this exercise is to train a conditional GAN and a conditional Wasserstein GAN to generate handwritten digits (MNIST) and compare them. The GANs must be conditional. That is, you must be able to input the digit you want to generate and the Generator must produce a variety of samples of that digit.

Generate Gaussians

```
# Generate training data from 6 2D gaussians lying on a circle

n = 10000

gaussian_means = [
    [np.cos(2*np.pi*i/6), np.sin(2*np.pi*i/6)]
    for i in range(6)
]

for i in range(n):
    if i == 0:
        data = 0.05*np.random.randn(1, 2) + gaussian_means[0]
    else:
        idx = np.random.randint(0, 6)
        data = np.concatenate([data, 0.05*np.random.randn(1, 2) + gaussian_means[

plt.figure()
plt.scatter(data[:, 0], data[:, 1])
plt.title('6 2D gaussians lying on a circle')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

Financial time series

```
!pip install yfinance

def get_stock_data(ticker, start_date, end_date, max_length):
```

```
    # Download data as a pandas dataframe
    # Note: we only need the closing price!
    stock_data_df = yf.download(ticker,  start = start_date, end = end_date)
    stock_data_df = stock_data_df[['Close']]

    # Impute values for NaN entries
    stock_data_df = stock_data_df.fillna(method='ffill')

    # Interpolate onto a daily basis
    stock_data_df = stock_data_df.resample('D').interpolate()

    # Normalize the data
    stock_data_df = stock_data_df / stock_data_df.iloc[0]

    # Truncate the data to the maximum length
    stock_data_df = stock_data_df[:max_length]

    # Get data as numpy array
    data = stock_data_df.iloc[:, 0].values

    return data

ticker_list = [
    'AAPL', 'MSFT', 'GOOGL', 'AMZN', 'TSLA', 'JPM', 'JNJ', 'V', 'PG',
    'UNH', 'DIS', 'MA', 'INTC', 'VZ', 'HD', 'MRK', 'PFE', 'CSCO', 'BAC', 'WMT',
    'KO', 'XOM', 'CVX', 'CMCSA', 'PEP', 'ADBE', 'NFLX', 'PYPL', 'T', 'ABT',
    'NVDA', 'COST', 'CRM', 'MCD', 'ABBV', 'ACN', 'NKE', 'AMGN', 'TMO', 'LLY',
    'IBM', 'HON', 'QCOM', 'UNP', 'NEE', 'LIN', 'TXN', 'PM', 'UPS', 'SBUX',
    'LOW', 'AMD', 'ORCL', 'GE', 'CAT', 'MDT', 'CVS', 'FIS', 'BA', 'INTU',
]

# Set start and end date
start_dates = [
    '2016-08-01', '2017-08-01', '2018-08-01', '2019-08-01',
    '2020-08-01', '2021-08-01', '2022-08-01', '2023-08-01'
]
end_dates = [
    '2016-12-31', '2017-12-31', '2018-12-31', '2019-12-31',
    '2020-12-31', '2021-12-31', '2022-12-31', '2023-12-31'
]

all_data = []
for ticker in ticker_list:
    for start_date, end_date in zip(start_dates, end_dates):
        data = get_stock_data(
            ticker=ticker,
            start_date=start_date,
            end_date=end_date,
            max_length=150
        )
        all_data.append(data)
```

```
data = np.array(all_data)
```