TensorFlow configuration

This appendix describes step by step the installation of TensorFlow 1.15 as well as the packages needed to run the TimeGAN framework.

1. TensorFlow version

TensorFlow is an open source Python framework developed by Google, which is used to perform large-scale machine learning models calculations, whose first release dates back to 2015.

2. Development IDE

The Anaconda Navigator 2.3.0 IDE along with PyCharm Professional Edition 2022.2.1 were used for this work.

3. Environment configuration

The version of TensorFlow used in this work is 1.15, the same as the code base of the TimeGAN framework. Since the version of TensorFlow is not the latest (2.11 to date), it is required to create an exclusive environment so that TimeGAN does not interfere with other projects using current versions of this or other Python packages.

3.1 Python and TensorFlow versions

TensorFlow version 1.15 was used for this work, which uses the Python version 3.7.

3.2 Python environment

First of all, creating an environment exclusively for this project is recommended, so as not to alter the other existing configurations on the computer. To do this, the following command is executed:

```
conda create --name timegan python = 3.7 (1)
```

Table 1 show required packages to run TimeGAN generation processes.

Package name	Package version
numpy	1.18.4
easydict	1.9
pytz	2017.2
pillow	6.2.0
pyparsing	2.0.3
dm-tree	0.1.1
mpmath	1.2.1
protobuf	3.9.3
pandas	1.3.5
matplotlib	3.5.3
scikit-learn	1.0.2
seaborn	0.11.2

Table 1: Required packages

Then, the created environment must be activated with the following instruction:

```
conda activate timegan (2)
```

The installed version of Anaconda is updated with the command:

```
conda update -n base -c defaults conda (3)
```

It is also recommended to update the *pip* version, with the command:

```
python.exe -m pip install --upgrade pip (4)
```

3.3 Packages installation

The packages required to successfully run TimeGAN in this work are listed below.

First, the *jupyter* package is installed with the command:

```
pip install jupyter (5)
```

Any other code editor could also be used, however, the configuration shown here may require additional changes. Figure 1 shows the respective output after running the command (5).

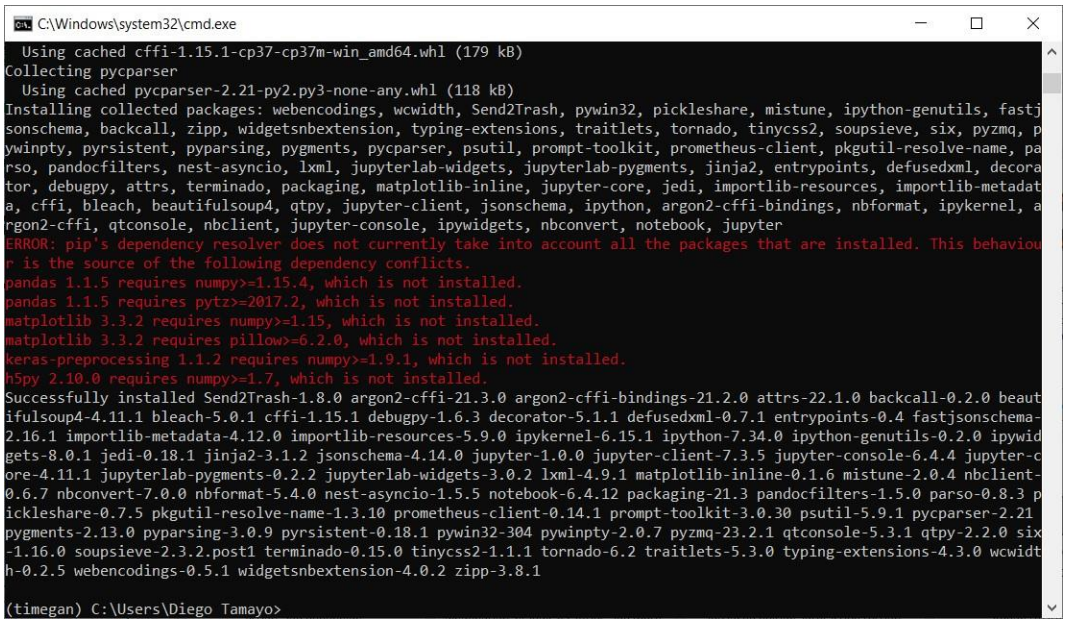

Figure 1: *jupyter* package installation

Next, the *numpy* package is installed with the following command:

```
pip install numpy==1.18.4 (6)
```

Figure 2 shows the respective output after running the command (6).

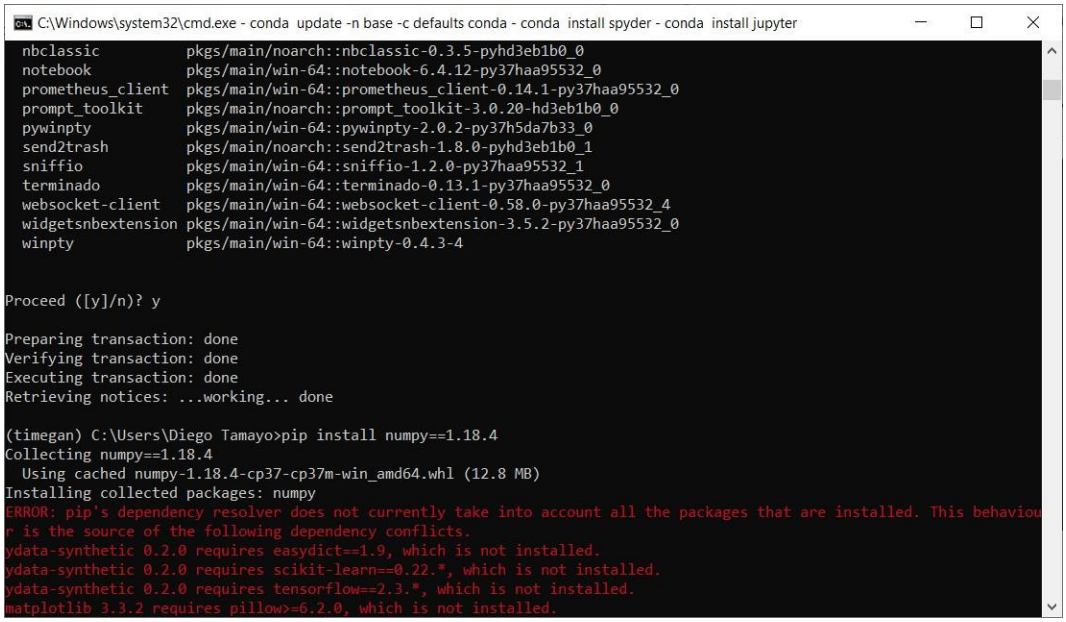

Figure 2: *numpy* package installation

Then, the *easydict* package is installed with the following command:

```
pip install easydict==1.9 (7)
```

Figure 3 shows the respective output after running the command (7).

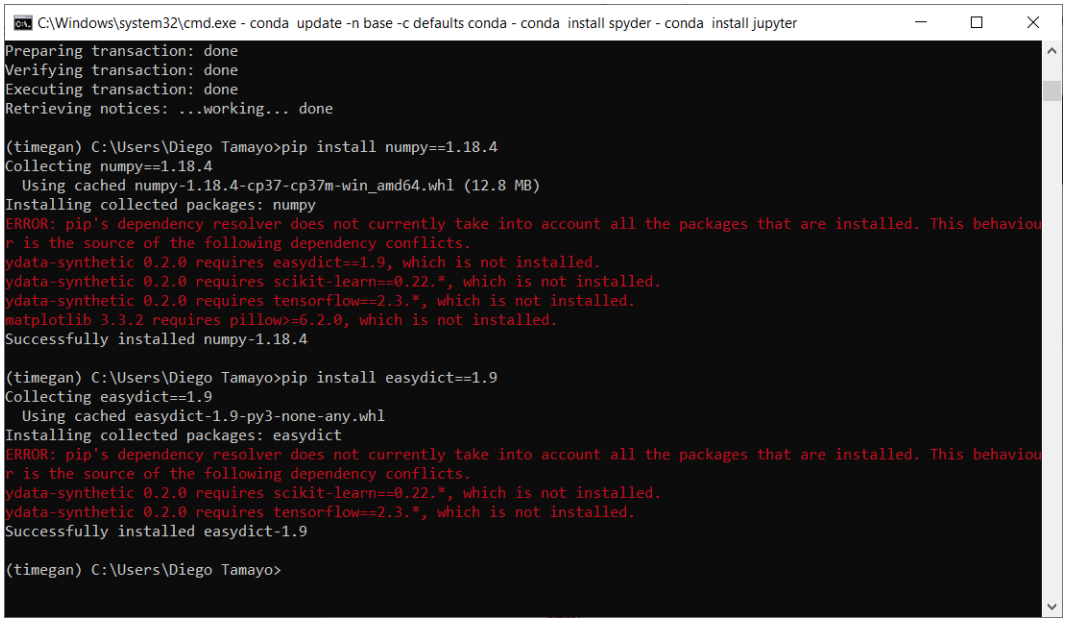

Figure 3: *easydict* package installation

Then, it is installed the *pytz* package with the following command:

```
pip install pytz==2017.2 (8)
```

Figure 4 shows the respective output after running the command (8).

Next, the *pillow* package is installed with the following command:

```
pip install pillow==6.2.0 (9)
```

Figure 5 shows the respective output after running the command (9).

Next, the *pyparsing* package is installed with the following command:

```
pip install pyparsing==2.0.3 (10)
```

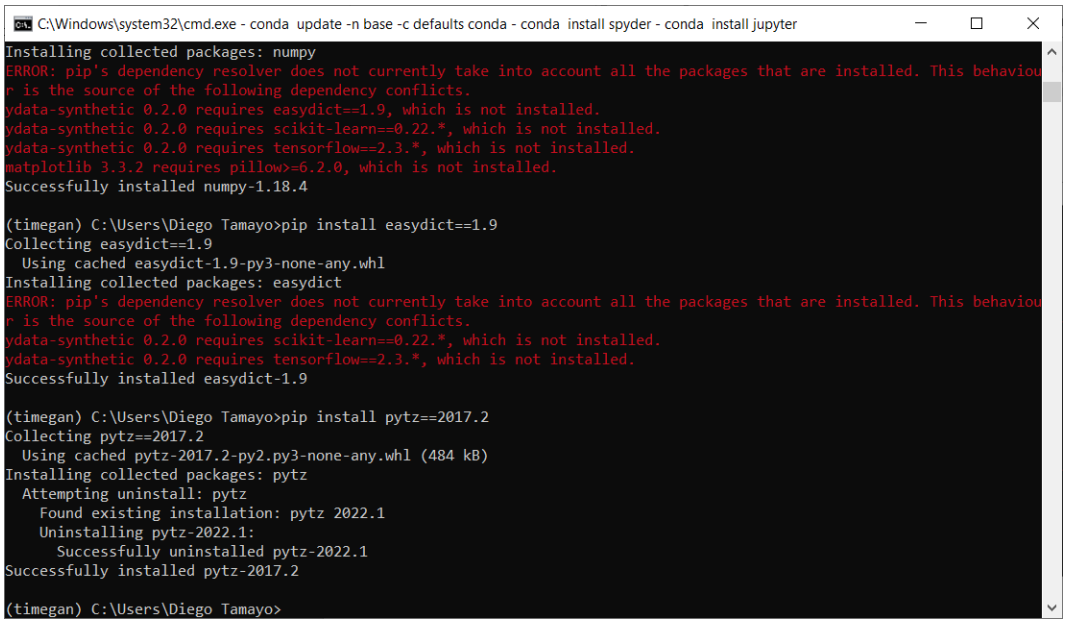

Figure 4: *pytz* package installation

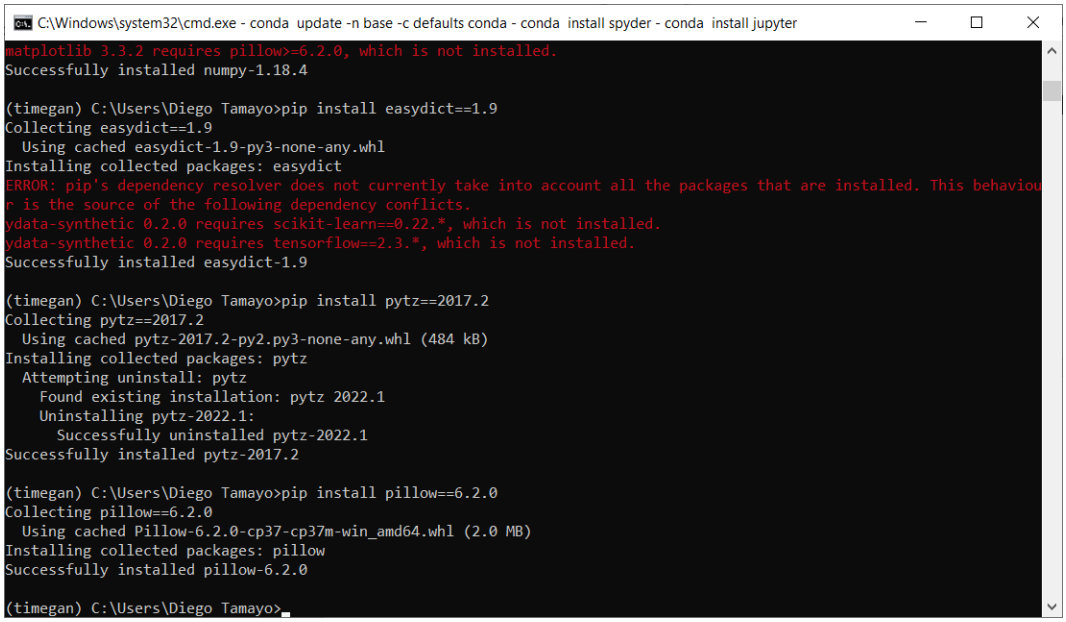

Figure 5: *pillow* package installation

Figure 6 shows the respective output after running the command (10).

The next step is to install the *dm-tree* package with the following command:

```
pip install dm-tree==0.1.1 (11)
```

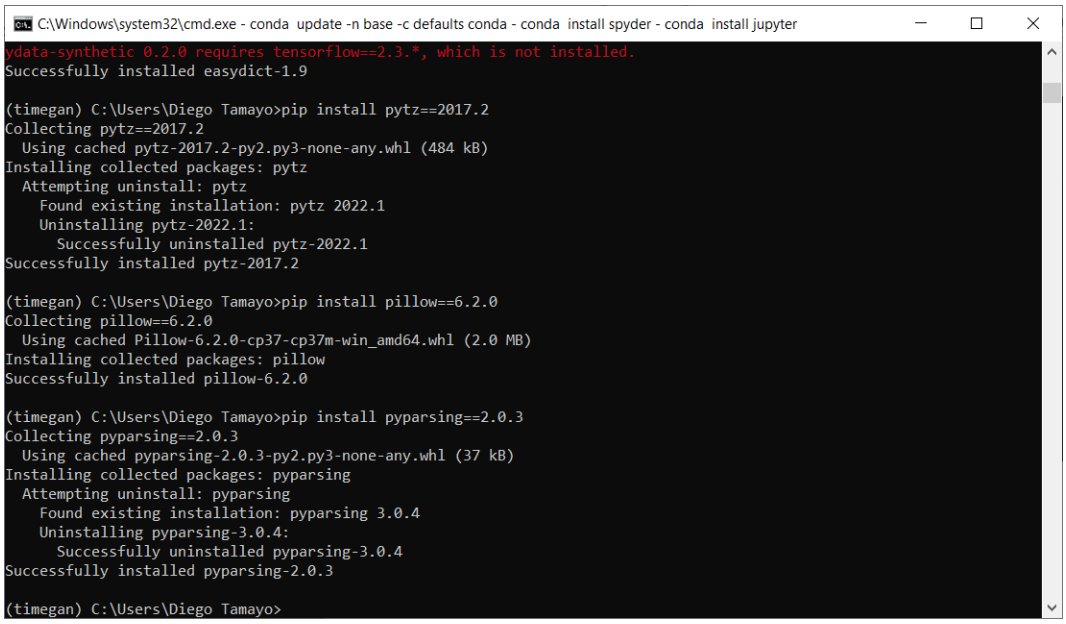

Figure 6: *pyparsing* package installation

Figure 7 shows the respective output after running the command (11).

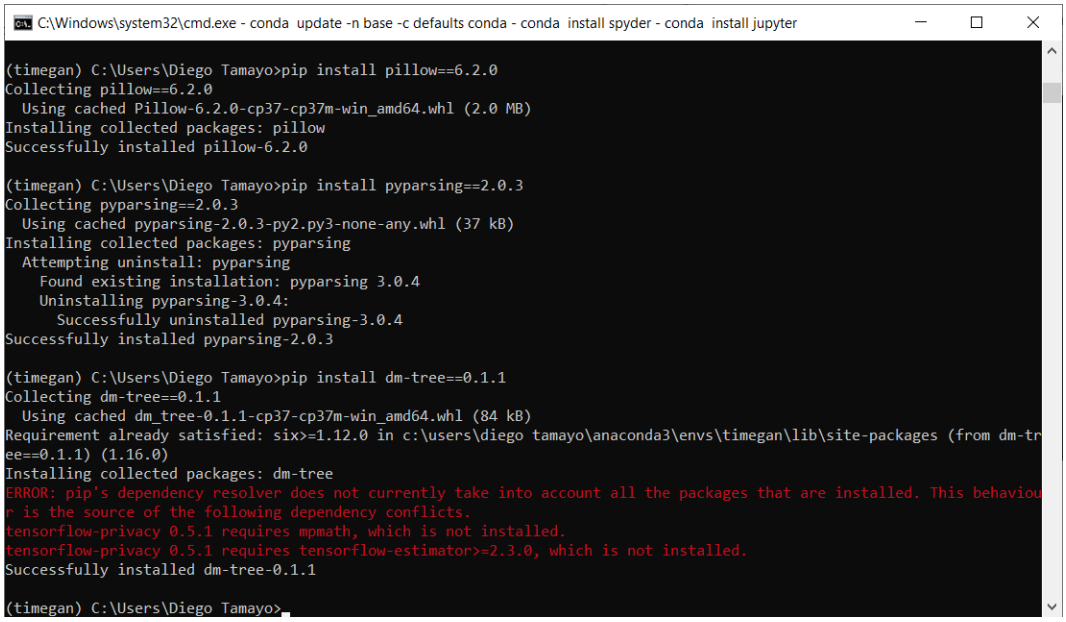

Figure 7: *dm-tree* package installation

The next step is to install the *mpmath* package with the following command:

```
pip install mpmath (12)
```

Figure 8 shows the respective output after running the command (12).

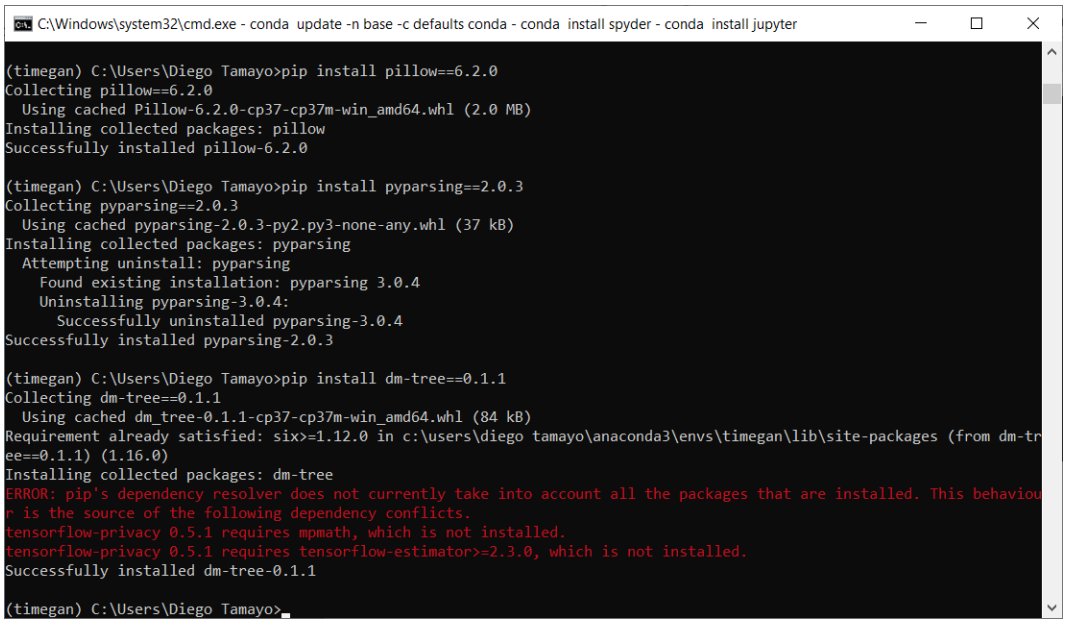

Figure 8: *mpmath* package installation

Once the necessary packages have been installed, the *tensorflow* framework is installed with the following command:

```
pip install tensorflow==1.15 (13)
```

Figure 9 shows the respective output after *tensorflow* has been installed.

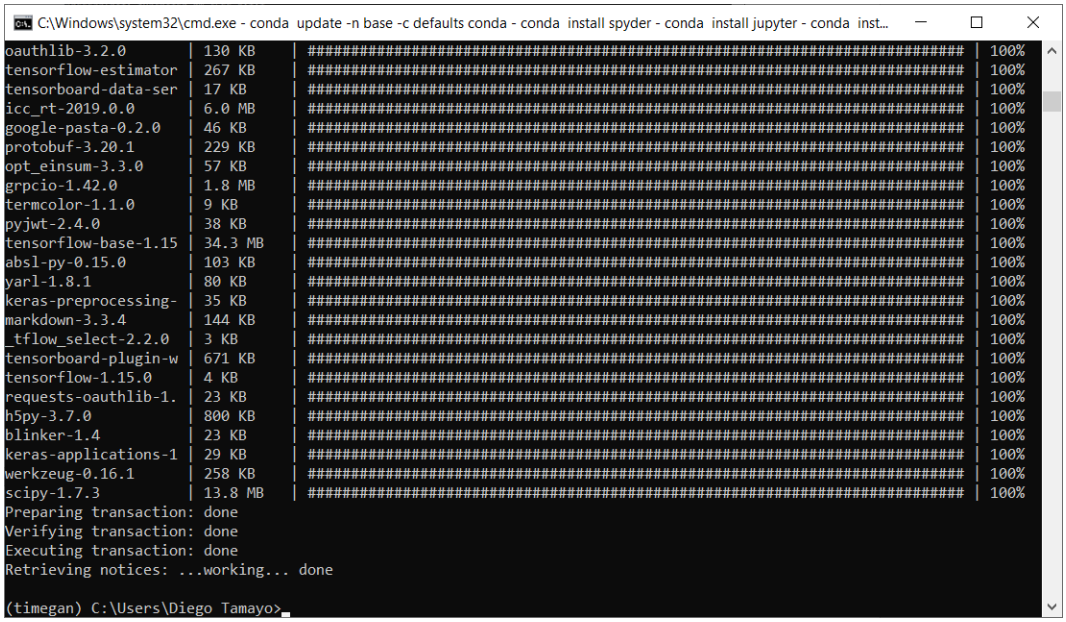

Figure 9: *tensorflow* framework installation

To avoid future problems, the existing version of the *protobuf* package must be uninstalled, which is done with the following command:

```
pip uninstall protobuf (14)
```

Figure 10 shows the respective output after running the command (14).

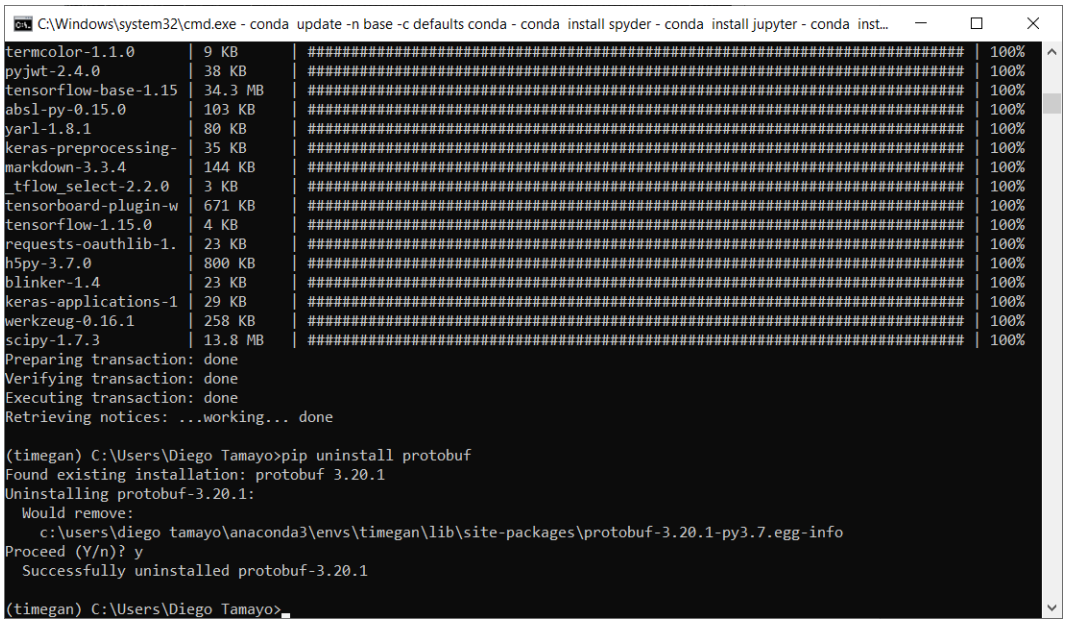

Figure 10: *protobuf* package uninstallation

The appropriate version of the *protobuf* package must now be installed with the following command:

```
pip install protobuf==3.9.0 (15)
```

Figure 11 shows the respective output after running the command (15).

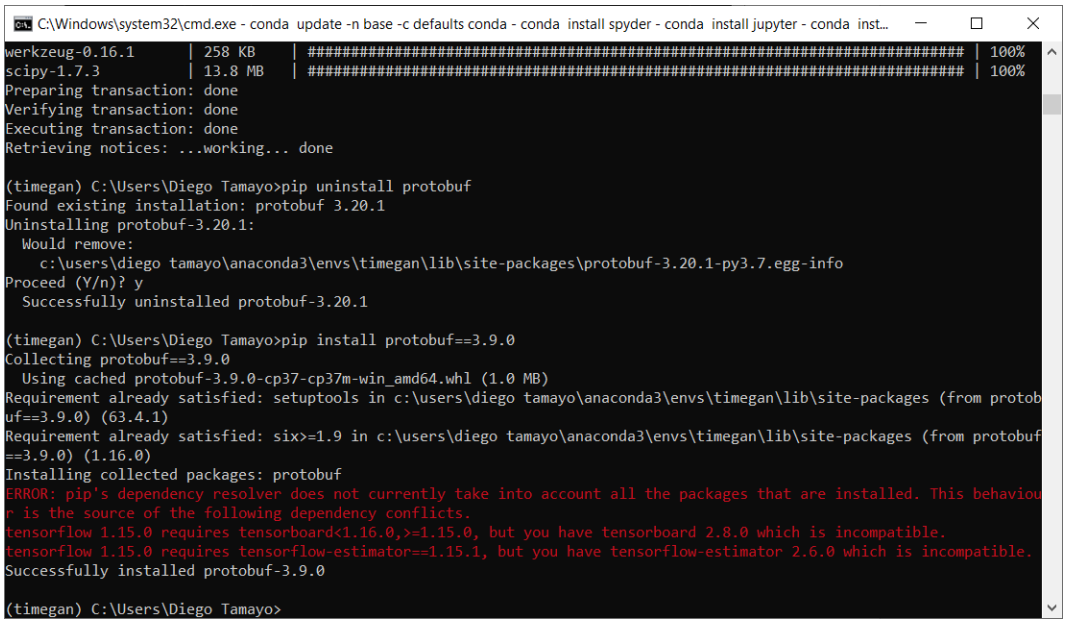

Figure 11: *protobuf* package installation

To find out which version of TensorFlow is installed, use the following command:

```
python -m pip show tensorflow (16)
```

Figure 12 shows the respective output after running the command (16).

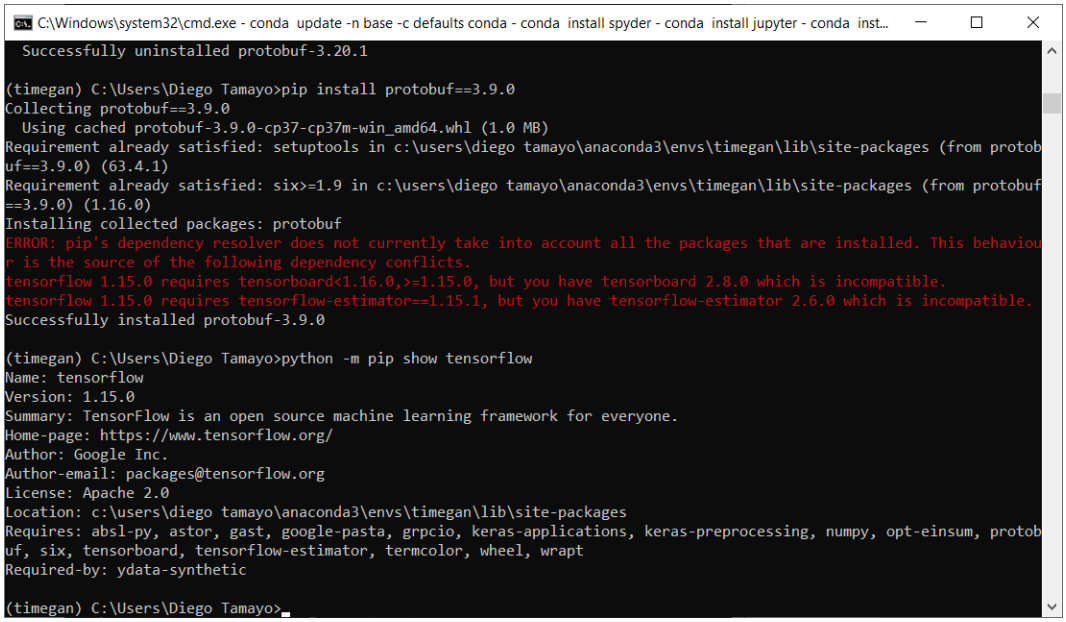

Figure 12: Tensorflow version information

It is also possible to find out the installed version of Tensorflow from the console of a Python interpreter. To do this we first run the following code:

```
import tensorflow as tf
tf. version
```

In order to be able to use the specific Python kernel for the environment created in Anaconda, which is associated with Tensorflow, the *ipykernel* package is required. Specifically, this package is required when working with *Jupyter Notebook*, as it associates the kernel with the corresponding environment. The *ipykernel* package must be installed with the following command:

```
conda install -c anaconda ipykernel (17)
```

Figure 13 shows the respective output after running the command (17).

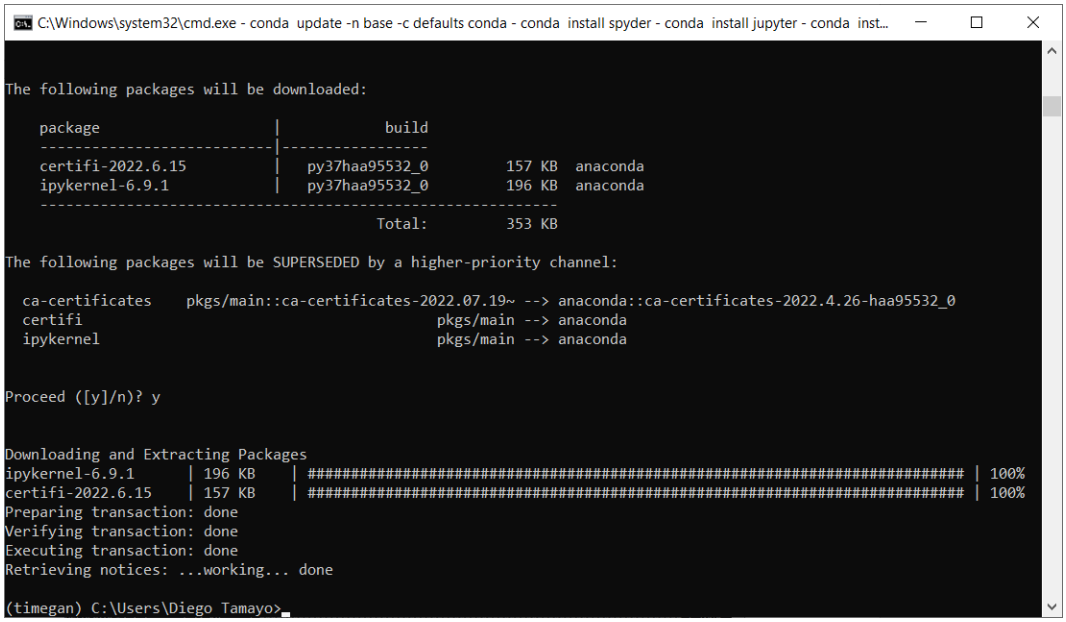

Figure 13: *ipykernel* package installation

In case it is necessary to uninstall a particular kernel, it can be done with the following command:

```
jupyter kernelspec uninstall name-unwanted-kernel (18)
```

Figure 14 shows the respective output after running the command (18).

Once the ipykernel package is installed, it is possible to create a kernel to associate with the environment created for TensorFlow, which will be called *timegan*. The command used for is:

```
python -m ipykernel install --user --name=timegan (19)
```

Figure 15 shows the respective output after running the command (19).

The *pandas* package is very useful because of its methods for manipulating datasets. This package is installed with the following command:

```
pip install pandas (20)
```

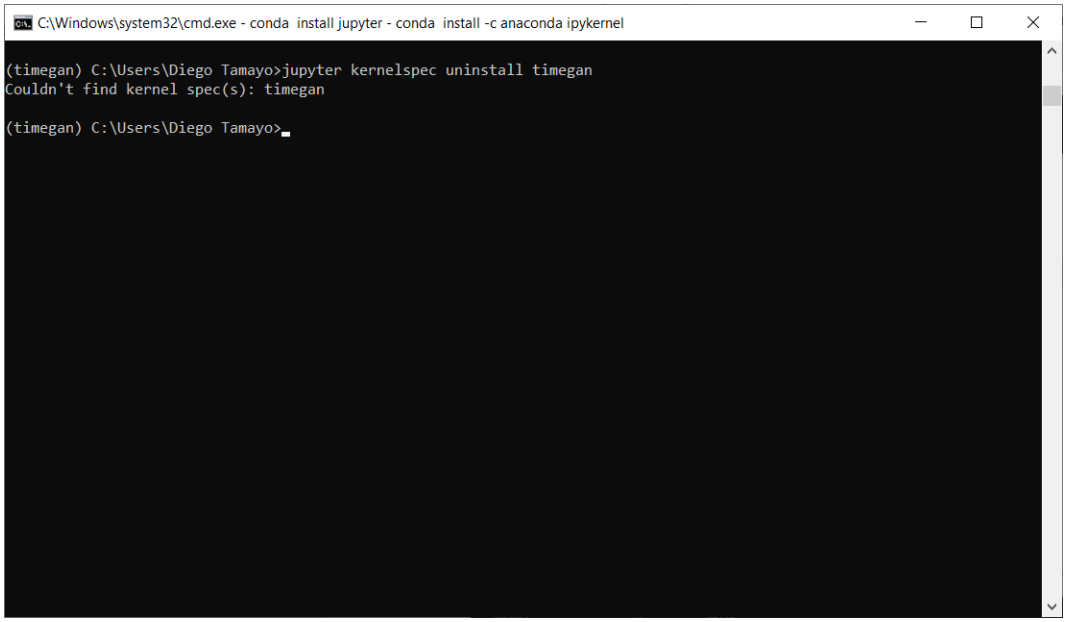

Figure 14: Kernel uninstallation command

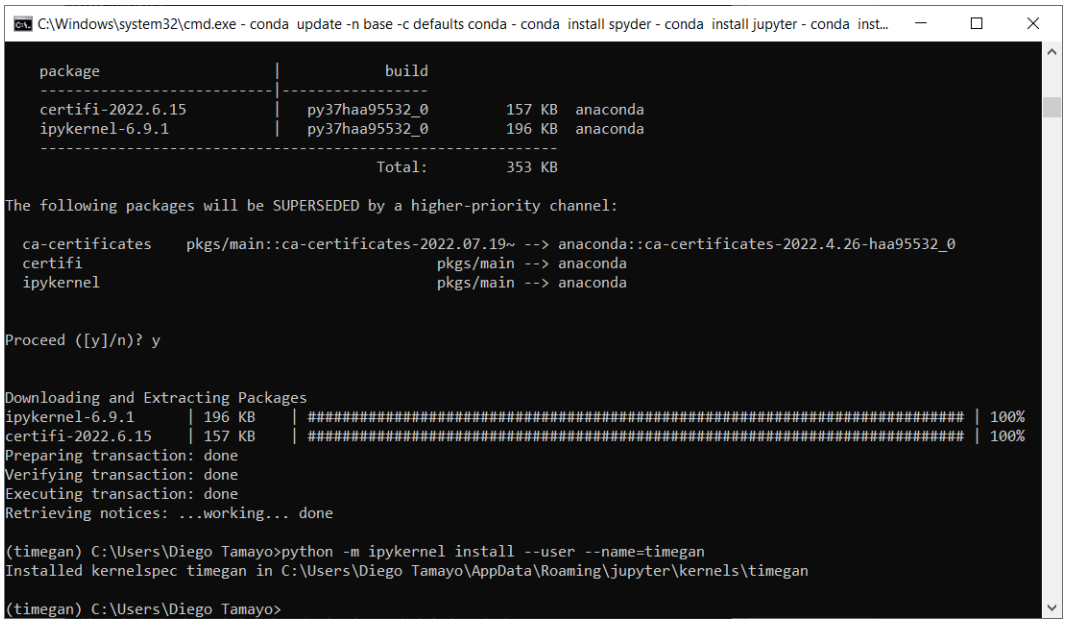

Figure 15: Kernel creation using *ipykernel*

Figure 16 shows the respective output after running the command (20).

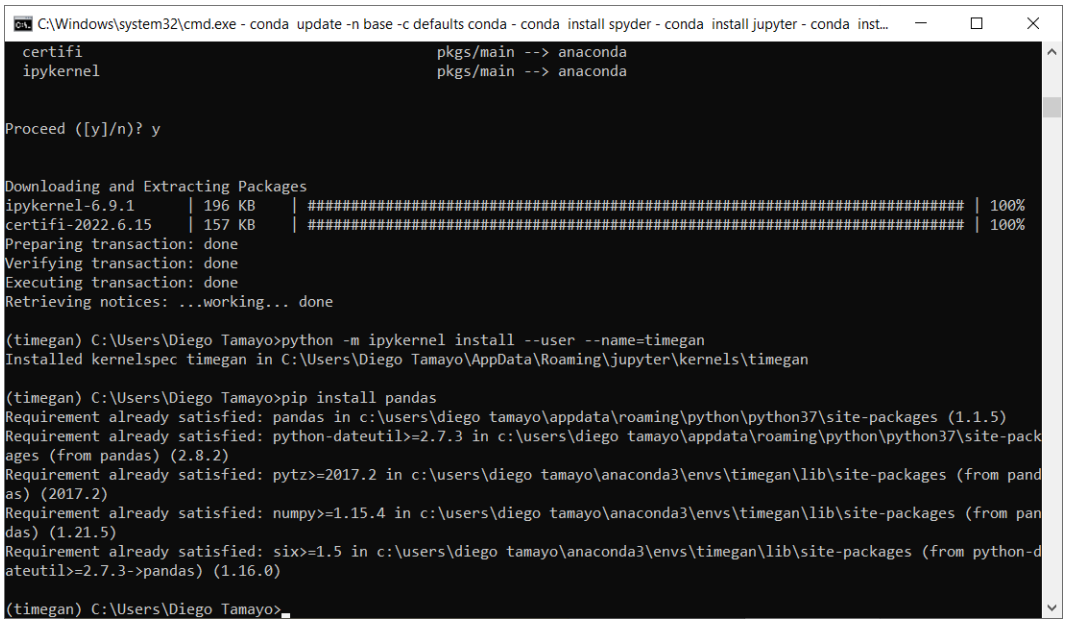

Figure 16: *pandas* package installation

A very useful library for displaying graphs is *matplotlib*, which can be installed with the following command:

```
pip install matplotlib (21)
```

Figure 17 shows the respective output after running the command (21).

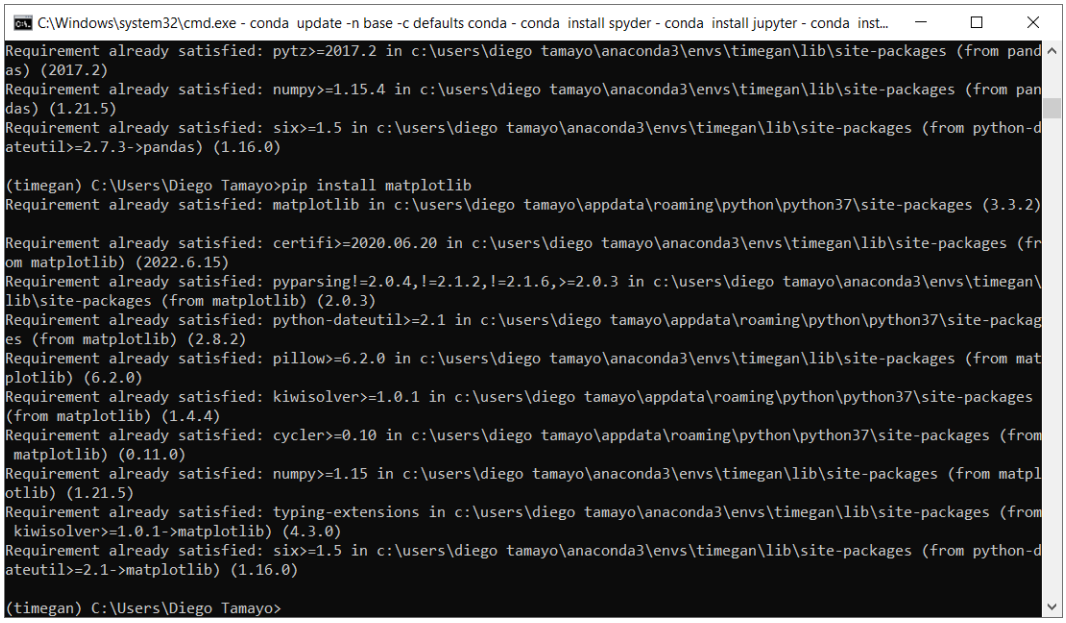

Figure 17: *matplotlib* package installation

A very useful package for data analysis is *scikit-learn*, which can be installed with the command:

```
pip install -U scikit-learn (22)
```

Figure 18 shows the respective output after running the command (22).

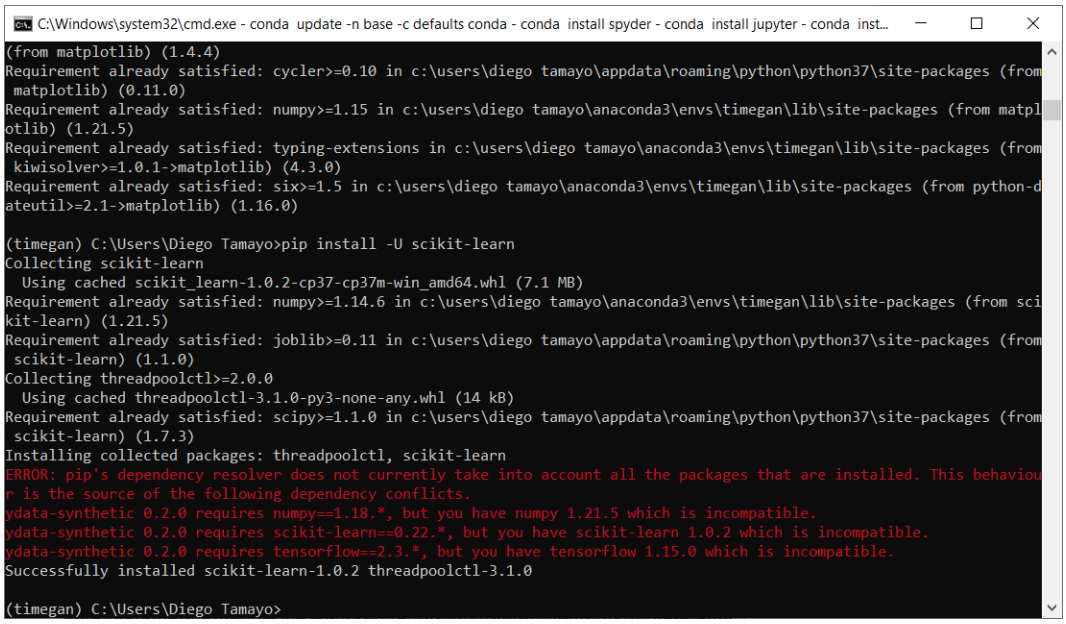

Figure 18: *scikit-learn* package installation

Another library for data visualization is *seaborn*, which is based on matplotlib, but has higher quality than matplotlib. To install this package, the following command is executed:

```
pip install seaborn (23)
```

Figure 19 shows the respective output after running the command (23).

4. Code execution

An example of a command to execute the code is shown in Figure 20.

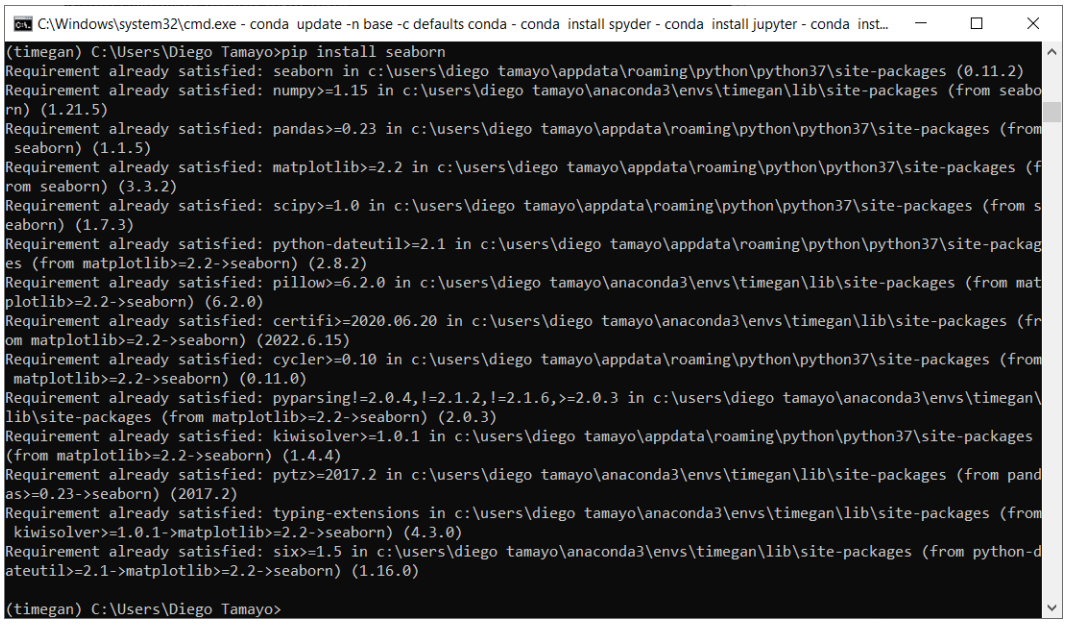

Figure 19: *seaborn* package installation

```
python main_timegan.py --data_name serie_temporal
                       --seq_len 24
                       --module lstm
                       --hidden_dim 80
                       --num_layer 3
                       --iteration 100
                       --batch_size 128
                       --metric_iteration 10
```

Figure 20: Python code execution

where:

- data_name is the name of the data file.
- seq_len is the length of the sequence.
- module is the type of RNN network (LSTM or GRU).
- hidden_dim is the number of units in a LSTM or GRU cell.
- num_layer is the number of layers of the TimeGAN networks.
- iteration is the number of iterations for training.
- batch_size is the number of samples or sequences from each batch.
- metric_iteration is the number of iterations for the calculation of the metrics.