README
Data repository for the paper "Binary Classification as a Phase Separation Process"

Rafael Monteiro

Mathematics for Advanced Materials - Open Innovation Laboratory,
AIST, c/o Advanced Institute for Materials Research,
Tohoku University, Sendai, Japan
monteirodasilva-rafael@aist.go.jp, rafael.a.monteiro.math@gmail.com

September 18, 2020

Abstract

This is a README file with general instructions for users of the data repository [1]. The latter contains simulations used in the paper "Binary Classification as a Phase Separation Process", by Rafael Monteiro.

This guide explains what is stored in the data repository and how simulation data and computational statistics can be retrieved. We explain what is stored, mostly refraining from explaining how such a data is generated: for that, you should either read the paper or read the computer code (or read both).

All the data we refer to is available at [1]: https://dx.doi.org/10.5281/zenodo.4005131.

All the code we refer to is available at [2]: https://github.com/rafael-a-monteiro-math/Binary_classification_phase_separation. For matters of reproducibility, the code does not require proprietary software: you simply need Python, which is freely available online.

Contents

Folder trees: branches (paths) and leaves (folder)

List of dictionary contents

Part I
Preliminaries

1 How to open the data

We will assume that you have some basic knowledge: how to run some python code on a jupyter-notebook and how to use the shell in an UNIX based system.

1.1 Decompressing tarballs

All the data are inside folders as compressed tarballs, with extension ".tar.gz". For this you will need the dependency GNU tar.

For example, if **example.tar.gz** is the name of the file you want to decompress, you run the following command on your terminal[1]

```
tar -xzvf example.tar.gz
```

with a similar command for the other cases.

1.2 Running scripts

Assuming that you are using an Unix based Operating System, we now explain how to run a script **script_example.sh**. Scripts are essentially a series of commands that the computer runs on a terminal. Now, using a terminal, go to the folder that contains the script. Then you type the following commands

```
chmod +x script_example_PSBC.sh
```

Afterwards, to run the script, you do

```
./script_example.sh
```

1.3 How to use download_PSBC.sh to automatically download the data

To make matters a bit simpler, I wrote a bash script **download_PSBC.sh** which you can download on the Github page for this project [2]. It contains a series of commands that the computer will perform sequentially: create a folder, download the data, decompress it inside the folder, etc.

If you run the script **download_PSBC.sh** it gives you this:

```
You have the following options:

Enter 1 to download all examples;
 Enter 2 to download all statistics;
 Enter 3 to download all jupyter notebooks;
 Enter 4 to download all raw data;
 Enter any other value to finish.
```

Since the script will download data, I assume that you have the main dependency for its use:

- GNU Wget

used to retrieve data using an HTTP protocol.

Now let's go back to the script and explain the output you see above:

Option 1) It will create a folder **Examples**, download the compressed data in a tarball **PSBC_Examples.tar.gz** using wget, decompressing it inside the folder **Examples**. This folder's content is described in Part III in this guide.

This data has only been used to generate the heatmaps of layers (Figure 13 in the paper) and also to test the "0" and "1" data that I posted in the markdown website (you can play with your own figures too if you'd like).

[1] I'm assuming an UNIX based system, like Linux or MAC.

Option 2) It will create a folder **Statistics**, to be described below in Part IV of this guide. Then it downloads the compressed data in a tarball **PSBC_computational_statistics.tar.gz** using wget, decompressing it inside the folder **Statistics**.

Option 1 gives you a smaller file (less than 500Mb), with all that you need to recreate most of the figures in the paper. If you would like the code to generate the figure you need the files in options 2 and 3 as well.

Option 3) It downloads jupyter-notebooks and related modules in a tarball **PSBC_Notebooks.tar.gz** using wget, decompressing it inside the current folder where the script **download_PSBC.sh** is. It contains the files

- **Notebook_PSBC_1D.ipynb**
- **Notebook_PSBC_MNIST.ipynb**
- **Notebook_PSBC_examples.ipynb**

besides auxiliary modules

- **binary_phase_separation.py**
- **aux_fnts_for_jupyter_notebook.py**

Option 4) It downloads four tarballs

- **PSBC_1D_and_Extras.tar.gz**
- **PSBC_MNIST_Neumann.tar.gz**
- **PSBC_MNIST_Periodic.tar.gz**
- **PSBC_MNIST_Non_diff.tar.gz**

using wget, decompressing them inside a folder **PSBC**, reconstructing subfolders and decompressing files so that the folder tree respects the design of Trees 2, 3, 4, and 5. Raw data's content is thoroughly explained in Part V.

Remark 1.1 (Figures reproducibility) *Note that in order to reproduce the figures and the results in the website the folder where the script **download_PSBC.sh** runs has to contain the jupyter notebooks, the folder **Statistics**, and the folder **Examples**.*

Folders have to obey the folder structure given in the trees as described here, otherwise the code in the jupyter-notebooks will not be able to access the data correctly.

1.4 Reading pickled files

All the data has been recorded in dictionaries, statically stored as ".p" files. You can access these files using python in the following manner: first, you need to open python and import the following package

```
try:
    import cPickle as pickle
except ImportError: # if you use python 3.x
    import pickle
```

Then, assuming that you want to read a file **data.p** in the folder **Statistics/MNIST/**, you run

```
with open("Statistics/MNIST/parameters_MNIST_Neumann.p", "rb") as fp:
      import_dictionary = pickle.load(fp)
```

and now you have the dictionary **import_dictionary** available to you. More examples of dictionaries usage can be found in the git-hub repository for this paper, at [2].

1.5 Main python dependencies

The whole code was run using python 3.7.0. Modules version are given below:

Versions used were as follows:

- sklearn 0.22.2.post1
- matplotlib 3.2.1
- numpy 1.18.1
- scipy 1.3.0
- sympy 1.3
- pandas 1.0.1
- keras 2.3.1
- tensorflow 1.14.0
- shutil 1.14.0
- copy 1.14.0
- pickle 4.0

2 How to run the model

This section is complemented by the examples given in the tutorial companion website to this paper that can be found in [2]. All the python files in this section can be downloaded from the same github.

2.1 Importing the main module

A few things need to be done to run the model. First, the module **binary_phase_separation.py** with the PSBC's implementation has to be imported as

```
from  binary_phase_separation import *
```

2.2 Reading docstrings and asking for help

The module **binary_phase_separation.py** is endowed with extensive documentation, which can be accessed either by reading the comments in it, or as it is imported. We will explain how to use help functions and read docstrings (function documentation), and later on what are the main classes that the user needs to know in order to use the model.

Assume that we have defined an element of the class Binary_Phase_Separation()

```
Model = Binary_Phase_Separation()
```

Then two things can be done: one can directly access the docstring for that class, typing

```
print(Model.__doc__)
```

which gives the following short output:

```
This is the main class  of the Phase Separation Binary Classifier (PSBC).
    With its methods one can, aong other things, train the model and
    predict classifications (once the model has been trained).
```

A second choice consistis of using the help() method as follows,

```
print(help(Model))
```

which gives a much larger output, essentially printing out all the methods and attributes the class has. Part of the output is reproduced below:

```
Help on Binary_Phase_Separation in module binary_phase_separation object:
class Binary_Phase_Separation(builtins.object)
 |  Binary_Phase_Separation(cost=None, par_U_model=None, par_P_model=None,␣
↪par_U_wrt_epochs=None, par_P_wrt_epochs=None)
 |
 |  This is the main class  of the Phase Separation Binary Classifier (PSBC).
 |  With its methods one can, aong other things, train the model and
 |  predict classifications (once the model has been trained).
 |
 |  Methods defined here:
 |
 |  __init__(self, cost=None, par_U_model=None, par_P_model=None, par_U_wrt_epochs=None,␣
↪par_P_wrt_epochs=None)
 |      Class initializer.
 |
 |      Parameters
 |      ----------
 |      cost : bool, True, optional
 |      par_U_model : dictionary, None, optional
 |          Dictionary containing initialization parameters for the U component
 |          of the PSBC.
 |      par_P_model : dictionary, None, optional
 |          Dictionary containing initialization parameters for the P component
 |          of the PSBC.
 |      par_U_wrt_epochs : dictionary, None, optional
 |          Dictionary containing dictionaries of U parameters throughout the
 |          training.
 |      par_P_wrt_epochs : dictionary, None, optional
 |          Dictionary containing dictionaries of P parameters throughout the
 |          training.
 |
 |      Attributes
 |      ----------
 |      cost, par_U_model, par_P_model, par_U_wrt_epochs, par_P_wrt_epochs
 |
 |      Returns
 |      -------
 |      Class initializer. No returned value.
 |      Parameters are accessible as instance variables.
 |
 |  predict(self, X, par_U_model, par_P_model, with_phase=True, subordinate=True)
 |      'predict' method.
 |
 |      This method predicts the labels of X for a PSBC with parameters given
 |      by par_U_model and par_P_model.
 |      No accuracy is computed.
 |
 ...
```

In this output we can see, that the object Model has two methods predict and predict. They can be used, for example, if one types

```
Model.predict(X, par_U_model, par_P_model)
```

2.3 How to feed parameters and data to the PSBC

Assume that you have the following set of hyperparameters:

```
learning_rate = (.1,.08,.93)
patience = float("inf")
sigma = .1
epochs, dt, dx, eps, Nx, Nt = 600, .1, 1, 0, 1, 20
weights_k_sharing = Nt
ptt_cardnlty = 1
batch_size = None
subordinate, save_parameter_hist, orthodox_dt, with_phase = True, True, True, True
```

Which means that one has a PSBC classifying data in a 1D feature space (because $Nx = 1$). Part of the model (for example, trainable weights initialization) can be initialized by doing

```
Init = Initialize_parameters()
data = Init.dictionary(Nx, eps, dt, dx, Nt, ptt_cardnlty, weights_k_sharing, sigma = sigma )
```

Now we extend the data, the dictionary with hyperparameters, adding other features to the model:

```
data.update('learning_rate' :  learning_rate, 'epochs' :  epochs,\
            'subordinate' :  subordinate,"patience" : patience,\
            "orthodox_dt" : orthodox_dt,'with_phase' : with_phase,
            "batch_size" : batch_size, "save_parameter_hist" : save_parameter_hist )
```

We remark that each of these hyperparameters is explained in the methods documentations. For instance, 'subordinate' is a boolean "True" or "False", concerning to the model having subordinate phase or not (see paper, Section 4), 'patience' is related to the patience parameter p^* in the Early stopping method (in the present case, the model takes "infinite" patience'), 'orthodox_dt' is also a boolean taking "True"/"False" values, concerning the model adopting or not the Invariant Region Enforcing Condition (1.14) (in all the simulations we did comply with such a constraint: it has been added here in case someone wants to explore the model without satisfying the constraint; evidently, the model is a bit cheaper in such a case, since the quantities in (1.14) do not need to be computed.), and 'with_phase' indicates whether the model has phase or not.[2]

More information about each entry can be found in the documentation of each function, accessible using the instructions in Section 2.2.

2.4 How to train the model

```
Model.train(
    X_train, Y_train, X_train, Y_train, learning_rate, dt, dx, Nt,\
        weights_k_sharing, eps = eps, epochs = epochs, \
        subordinate = subordinate, with_phase = with_phase,\
        drop_SGD = drop_SGD, sigma = sigma,\
        orthodox_dt = orthodox_dt, print_every = 300,\
            save_parameter_hist = save_parameter_hist
)
```

[2]This is just a technical artifact: the model should have phase, and it is not necessary to add it here, for it takes this entry as "True" by default. We have only added the variable to illustrate the bad behavior of the model whenever the phase parameter is not there; this is investigated in the Section 2 of the paper.

It is clear by the name of the variables that X_train corresponds to sample feature and Y_train to sample' labels. Two important observations are worthwhile to be made: (i) Y_train should not be a rank one vector, but a vector of size $1 \times$ batch size , and X_train should have size $Nx \times$ batch size ; (ii) the quantities X_train and Y_train are used twice in the model's input slots. The reason is due to Early stopping, because the data in the second occurrence is used to create a monitored quantity M_q at each epoch q, as explained in the section optimization in Appendix A. For example, if one wants to train the model in the training set, but test it on the test set (here, written as (X_test, Y_test), it would be acceptable to run the following code instead

```
Model.train(
    X_train, Y_train, X_test, Y_test, learning_rate, dt, dx, Nt,\
        weights_k_sharing, eps = eps, epochs = epochs, \
        subordinate = subordinate, with_phase = with_phase,\
        drop_SGD = drop_SGD, sigma = sigma,\
        orthodox_dt = orthodox_dt, print_every = 300,\
            save_parameter_hist = save_parameter_hist
)
```

Once this step is complete properties of the trainable model are readily available as atributes of the object Model. For instance,

```
diameter_history = Model.diameters_hist
```

yield a dictionary with keys "U" and "P", where diameter_history["P"] is a vector with the description of the quantity $(\mathscr{P}_\alpha)_q$ over epochs; the role of diameter_history["U"] is similar, concerning the quantity $(\mathscr{P}_\beta)_q$.

Other attributes are also available, and can be seen using the help function (as explained in Section 2.2). Other examples are given in the tutorial website written as a companion to the github, and also in the jupyter notebook **Notebook_PSBC_examples.ipynb**.

Part II
Jupyter-notebooks, bash scripts, and python scripts

There are 3 jupyter-notebooks:

- **Notebook_PSBC_1D.ipynb** : generates the plots and tables in Sections 2, 3, and 3.6 of the paper.
- **Notebook_PSBC_MNIST.ipynb** : generates plots and tables in Sections 4, 5, and 6.3 of the paper.
- **Notebook_PSBC_examples.ipynb** : jupyter-notebook with the example shown in the website.

They are available to download on [2], as a tarball **PSBC_Notebooks.tar.gz**.

Remark 2.1 *It is important to emphasize that if you have the jupyter-notebooks and want to redo the figures, they have to be in the same directory as the folders* ***Statistics*** *and* ***Examples****.*

Besides that, there are several python scripts and modules. They have been kept in the folders with the raw data, in the way they were used, therefore we will not explain which one to use for each simulation: for that, the user just need to go to the folder and check the scripts therein.

Names and roles of scripts and modules are below.

- **binary_phase_separation.py**: this is the main python module, with an implementation of the PSBC.
- **aux_fnts_for_jupyter_notebook.py**: this is an auxiliary module for plotting and saving figures, used by jupyter-notebooks.
- **generate_statistics.py**: script to generate statistics which works if you have the raw data respecting the folder structure in the trees represented in this file (it has to be in the same directory as folders **PSBC** and **Extras**). It will create a folder **Statistics**, with the data that is already available as a tarball **PSBC_computational_statistics.tar.gz** in [1]; see Part IV.
- **Main_script.py**: script that calls the PSBC function at each set of parameters, manages outputs, save them, etc. This script's name may have some variations, and also vary its extension, depending on the context it has been used (for example, **Main_script_learning.py** when used for model selection of learning rates, etc).
- **Generate_k_folds.py**: python script to generate k-fold splitting for model selection.
- **Parameter_generator.py**: a script that generates folders within which simulations take place, or generates pickled lists of values where a parameter will assume values. For instance, upon doing model selection for the learning rate, this script would generate a pickled file **grid.p** containing a vector with values for the parameter **learning_rate**. This file may vary in name, with an extension "learning" or "varying", to distinguish which case it is used for.
- **submit-qsub.sh** : bash script that coordinates the order in which python scripts are run. This is the starting point once you want to run a simulation. This is an auxiliary bash script that sends information to another bash script ("submit.sh", below). Names may have changes in the middle (**submit_MNISt-qsub.sh** etc) but the prefix "**submit**" and suffix "**-qsub.sh**" are always the same.
- **submit.sh**: bash script that submits the batch job to a supercomputer using a PBS file. Names may have changes in the middle (**submit_MNISt-qsub.sh** etc) but the prefix "**submit**" and suffix "**.sh**" are always the same (in the latter case, no "**qsub**" in the suffix).
- **Data_generator_MNIST_normalized.py**: you can find it in the subfolders of **Extras/Normalizations/**. It is used to download the whole MNIST database, select the subset of digits "0" and "1", normalize them in the range $[0.5, 0.7]$ (see why in Appendix 6.10 of the paper), and save them as csv files. See Part III for further information.

Part III

Summary of examples in the folder Examples

This folder is contained in a tarball **PSBS_Examples.tar.gz** in [1]. It contains a copy of the normalized subset of handwritten digits "0" and "1" of the MNIST database as well as trained PSBC models on the MNIST database.

The normalized dataset is split in two .csv files:

- **data_train_normalized_MNIST.csv**
- **data_test_normalized_MNIST.csv**

which can be generated using the script **Data_generator_MNIST_normalized.py**. They have been saved in such a way that each row corresponds to an individual, and has 785 entries: the first 784 are features, the last entry is the tag, specifying whether this is a "0" or a "1" individual. An example of its use can be found in several scripts, or in the jupyter-notebook **Notebook_PSBC_MNIST.ipynb**, where a sample image of these numbers is depicted.

Inside **Examples** we also find one example of each parameter configuration for the PSBC. The name of each folder corresponds to a PSBC model with certain parameters, as shown in the next table.

Folder name	PSBC's parameters
W1S-NS	Non-diffusive, Weights-1-sharing, non-subordinate
WNtS-NS	Non-diffusive, Weights-N_t-sharing, non-subordinate
W1S-S	Non-diffusive, Weights-1-sharing, subordinate
WNtS-S	Non-diffusive, Weights-N_t-sharing, subordinate
W1S-Nt2	Diffusive, Weights-1-sharing, subordinate, Neumann B.C.'s, $N_t = 2$
W1S-Nt4	Diffusive, Weights-1-sharing, subordinate, Neumann B.C.'s, $N_t = 4$
W1S-Nt8	Diffusive, Weights-1-sharing, subordinate, Neumann B.C.'s, $N_t = 8$
WNtS-Nt1	Diffusive, Weights-N_t-sharing, subordinate, Neumann B.C.'s, $N_t = 1$
WNtS-Nt2	Diffusive, Weights-N_t-sharing, subordinate, Neumann B.C.'s, $N_t = 2$
WNtS-Nt4	Diffusive, Weights-N_t-sharing, subordinate, Neumann B.C.'s, $N_t = 4$
WNtS-Nt8	Diffusive, Weights-N_t-sharing, subordinate, Neumann B.C.'s, $N_t = 8$
Per_W1S-Nt2	Diffusive, Weights-1-sharing, subordinate, Periodic B.C.'s, $N_t = 2$
Per_W1S-Nt4	Diffusive, Weights-1-sharing, subordinate, Periodic B.C.'s, $N_t = 4$
Per_W1S-Nt8	Diffusive, Weights-1-sharing, subordinate, Periodic B.C.'s, $N_t = 8$
Per_WNtS-Nt1	Diffusive, Weights-N_t-sharing, subordinate, Periodic B.C.'s, $N_t = 1$
Per_WNtS-Nt2	Diffusive, Weights-N_t-sharing, subordinate, Periodic B.C.'s, $N_t = 2$
Per_WNtS-Nt4	Diffusive, Weights-N_t-sharing, subordinate, Periodic B.C.'s, $N_t = 4$
Per_WNtS-Nt8	Diffusive, Weights-N_t-sharing, subordinate, Periodic B.C.'s, $N_t = 8$

Table 1: Subfolders in the folder **Examples**.

Clearly, there are no folders **Per_W1S-Nt1** nor **W1S-Nt1**: these models are the same as **Per_WNtS-Nt1** and **WNtS-Nt1**, respectively.

Part IV
Summary of simulations: the Statistics folder

Most of the data used in plots are contained in the folder **Statistics**, compressed as the tarball **PSBC_computational_statistics.tar.gz** in [1]. When the latter decompressed, one obtains the folder **Statistics** with the following tree structure:

Tree 1: Contents in the folder **Statistics**.

Each leaf is a folder, within which one finds summarized data in pickled files. We describe them next.

3 What is summarized in Statistics/Extras

In this folder you will find two pickled files

- **path_in_the_tree_Extras.p**
- **parameters_Extras.p**

with data used in Section 3.6 of the paper, which are plotted using the jupyter-notebook **Notebook_PSBC_1D.ipynb**. Both consist of pickled dictionaries, with keys given in Table 2.

Dictionary key	PSBC's parameters
ANN	Summary of simulations for ANNs with 3 layers and K hidden units, with $K \in \mathbb{G}_{10}$. In each case, 10 simulations are run.
KNN	Summary of simulations KNNs with 2 and 3 neighbors.
PSBC	Summary simulations for PSBS with parameters $N_t = 2$,

Table 2: Keys in dictionary pickled as **path_in_the_tree_Extras.p** and **parameters_Extras.p**, with their corresponding meaning in terms of PSBC's parameters. They contain data used in Section 3.6 of the paper.

Assuming that **path_in_the_tree_Extras.p** has been unpickled as a dictionary **path_Extras** (using instructions in Section 1.4 of this guide), if you type

```
path_Extras["ANN"]
```

you will obtain the path in the tree for the raw data, namely

```
Extras/Comparison_ANN_KNN_PSBC/1D-gamma_star_beta_star/N_0/
```

parameters_Extras.p is a pickled dictionary. When unpickled as a dictionary **parameters_Extras**, at each key in Table 2 we find another (sub) dictionary. For instance, **parameters_Extras["PSBC"]** has the following keys:

Dictionary key	Description
Avr_train	Average of best accuracies for the PSBC on training set
Avr_test	Average of best accuracies for the PSBC on test set
Std_test	Standard deviation of best accuracies for the PSBC on test set
Std_train	Standard deviation of best accuracies for the PSBC on training set

Table 3: Keys in the dictionary **parameters_Extras["PSBC"]**, with data used in Section 3.6 of the paper.

par_Extras["ANN"] is also a dictionary, with the following keys:

Dictionary key	Description
Avr_train	Average of best accuracies for ANNs on training set
Avr_test	Average of best accuracies for ANNs on test set
Std_test	Standard deviation of best accuracies for ANNs on test set
Std_train	Standard deviation of best accuracies for ANNs on training set

Table 4: Keys in the dictionary **parameters_Extras["ANN"]**, with data used in Section 3.6.

Last, **par_Extras["KNN"]** is a dictionary, with the following keys:

Dictionary key	Description
Avr_train2	Accuracy of KNN with 2 neighbors on training set
Avr_test2	Accuracy of KNN with 2 neighbors on test set
Avr_train3	Accuracy of KNN with 3 neighbors on training set
Avr_test3	Accuracy of KNN with 3 neighbors on test set

Table 5: Keys in the dictionary parameters_Extras["KNN"], with data used in Section 3.6 of the paper.

4 What is summarized in Statistics/1D

Inside the folder 1D you will find two pickled files

- **path_in_the_tree_1D.p**
- **parameters_1D.p**

They are actually pickled dictionaries that can be opened using instructions in Section 1.4 of the paper. They contain data used in Sections 2 and 3 of the paper, which are plotted using the jupyter-notebook **Notebook_PSBC_1D.ipynb**.

Each one of these dictionaries, when unpickled, have the following keys and respective PSBC model correspondence,

Dictionary key	PSBC's parameters
W1S-NP	1D, Weights-1-sharing, No phase with label (2.9)
WNtS-NP	1D, Weights-N_t-sharing, No phase with label (2.9)
WNtS-NP_BC	1D, Weights-1-sharing, No phase with label (2.12)
WNtS-WP_BC	1D, Weights-1-sharing, No phase with label (2.12)
WkS-WP_vary	1D, Weights-k-sharing for $k \in \mathbb{G}_{N_t}$, $N_t = 20$, No phase with label (2.12)

Table 6: Keys in dictionary pickled as **path_in_the_tree_1D.p** and **parameters_1D.p**, with their corresponding meaning in terms of PSBC's parameters.

Now, following the instructions in Section 1.4 of this guide, assume that you unpickle **path_in_the_tree_1D.p** as a dictionary with name **path_in_the_tree_1D_dict**. Then, at each key shown in table 6 you will find the location in the path to that folder in the folder tree. For instance, if you type

```
path_in_the_tree_1D_dict["W1S-NP"]
```

the output will be

```
'PSBC/1D/no_phase/weights-1-sharing/1D-Weights-1-sharing-no_phase'
```

which gives you the path of the raw data in the tree and also the name of the leaf name (folder name) it corresponds to. That is, the leaf path using Trees 3 and 4.

The dictionary pickled as **parameters_1D.p** also has the same keys, and in each of them you will find a summary of the 5 experiments that illustrate Sections 2 and 3 of the paper, which we describe next.

To begin with, let's assume that **parameters_1D.p** has been unpickled as the dictionary **parameters_1D_dict**; it has the five keys given in Table 6; at each of them one finds a sub dictionary. For instance, **parameters_1D_dict["W1S-NP"]** is also a dictionary (nested in the dictionary **parameters_1D_dict**), with three keys

gamma6, gamma7, gamma8,

corresponding, respectively, to the experiments in Section 2 and 3 of the paper for different values of $\gamma^* = 0.6$, $\gamma^* = 0.7$, and $\gamma^* = 0.8$. That is, if you type

```
parameters_1D["W1S-Np"]["gamma6"]
```

you get the summary of data for the simulation, with $\gamma^* = 0.6$, also stored as a dictionary. Its keys are given in Table 7-8.

We remark that each one of the keys **W1S-NP, WNtS-NP, WNtS-NP_BC , WNtS-WP_BC** have a similar structure, therefore it suffices to discuss just one of them. The only thing we have left then is describing **WkS-WP_vary**, which we shall discuss afterwards.

Dictionary key	Description
data	dictionary with basic information about the simulated model, with • dimension of parameter space (in Nx); • diffusion (in eps); • number of layers (in N_t); • partition cardinality (in ptt_cardnlty); • number K corresponding to weights-K-sharing property (in weights_K_sharing); • Boundary condition (in Neumann); • batch size (in batch_size); • number of epochs (in epochs); • if the phase is subordinate or not (in Subordinate, which is a boolean variable); • if the model has a phase in (with_phase, which is a boolean variable); • what - if any - parameter is being varied (in what_is_varying, which is a string).
best_accuracy_train	model's accuracy when evaluated at the point of best parameters, for all the simulations.
best_accuracy_test	accuracy of the model at best_par_U_model and best_par_P_model , for all the simulations.
cost	cost as evaluated at each epoch (realization of a single simulation, for illustrative purposes only).
diam_hist	dictionary with two entries, one for the evolution of $\mathscr{P}_\alpha$'s diameter, the other for the evolution of $\mathscr{P}_\beta$'s diameter; see Invariant Region Enforcing Constraint (1.14) in the paper (realization of a single simulation, for illustrative purposes only).
dt_hist	dictionary with two entries, one for the evolution of Δ_t^u along epochs, the other for the evolution of Δ_t^p along epochs (realization of a single simulation, for illustrative purposes only).
par_U_model	copy of parameters of trainable weights W_U as of the last epoch epoch (realization of a single simulation, for illustrative purposes only).
par_P_model	copy of parameters of trainable weights W_P as of the last epoch epoch (realization of a single simulation, for illustrative purposes only).
par_U_hist	only available at the 1D simulations, where parameters have been saved throughout the whole training. It is also a dictionary, where each key is a number with a copy of the trainable weights W_U at each epoch along training.
par_P_hist	only available at the 1D simulations, where parameters have been saved throughout the whole training. It is also a dictionary, where each key is a number with a copy of the trainable weights W_P at each epoch along training.

Table 7: Keys in the dictionary parameters_1D[A]["gamma"+B], with $A \in$ {"W1S-NP", "WNtS-NP", "WNtS-NP_BC", "WNtS-WP_BC"} and $B \in$ {"6","7","8"}. (part 1 of 2.)

Dictionary key	Description
prediction_train	vector with predicted labels for the training set at best_par_U_model and best_par_P_model.
prediction_test	Vector with predicted labels for test set at best_par_U_model and best_par_P_model.
cost_train	evaluation of cost function for training set over epochs (realization of a single simulation, for illustrative purposes only).
cost_test	evaluation of cost function for test set over epochs (realization of a single simulation, for illustrative purposes only).
alpha_all	values of training weights W_U for all simulations, stacked as a matrix, with the following distinctions • If the model is weights-1-sharing, it contains a copy of all trainable weights at the point of stop of the algorithm; • If the model is weights-N_t-sharing, it contains the evolutions of simulations.
accuracy_train	accuracy at the training set as measured when the model stopped, for all simulations.
accuracy_test	accuracy at the test set as measured when the model stopped, for all simulations.

Table 8: Keys in the dictionary parameters_1D[A]["gamma"+B], with $A \in$ {"W1S-NP", "WNtS-NP", "WNtS-NP_BC", "WNtS-WP_BC"} and $B \in$ {"6","7","8"}. (part 2 of 2.)

Last, we describe the content of the dictionary parameters_1D_dict["WkS-WP_vary"]. It contains data used in Section 6.6 of the paper, Figure 17. This is a much smaller dictionary with only a few keys:

Dictionary key	Description
data	dictionary with basic information about the simulated model, with • dimension of parameter space (in Nx); • diffusion (in eps); • number of layers (in N_t); • partition cardinality (in ptt_cardnlty); • batch size (in batch_size); • number of epochs (in epochs); • if the phase is subordinate or not (in Subordinate, which is a boolean variable); • if the model has a phase in (with_phase, which is a boolean variable); • what - if any - parameter is being varied (in what_is_varying, which is a string).
Acc_train	matrix with size 20×100, where each row corresponds to each accuracy of the model at the training set, at the last epoch (not at the best epoch, which would give even better results)
Acc_test	matrix with size 20×100, where each row corresponds to each accuracy of the model at the test set, at the last epoch (not at the best epoch, which would give even better results)
Best_Acc_train	matrix with size 20×100, where each row corresponds to each accuracy of the model at the training set, at the best epoch (not at the best epoch, which would give even better results)
Best_Acc_train	matrix with size 20×100, where each row corresponds to each accuracy of the model at the test set, at the best epoch (not at the best epoch, which would give even better results)

Table 9: Keys in the dictionary parameters_1D_dict["WkS-WP_vary"].

5 What is summarized in Statistics/MNIST

In the folder MNIST you will find six other pickled files. The first two that we discuss are

- path_in_the_tree_MNIST_nondif.p,
- parameters_MNIST_nondif.p.

which are pickled dictionaries with the following keys with content related to the MNIST database simulations using the non-diffusive PSBC. Like the other pickled files discussed in this section, they contain data used in Sections 4, 5, and 6.3 of the paper, which are plotted using the jupyter-notebook Notebook_PSBC_MNIST.ipynb.

The keys to the unpickled dictionaries are given below

Dictionary key	PSBC's parameters
W1S-NS	Non-diffusive, Weights-1-sharing, non-subordinate
WNtS-NS	Non-diffusive, Weights-N_t-sharing, non-subordinate
W1S-S	Non-diffusive, Weights-1-sharing, subordinate
WNtS-S	Non-diffusive, Weights-N_t-sharing, subordinate

Table 10: Keys in dictionary pickled as path_in_the_tree_MNIST_nondif.p and parameters_MNIST_nondif.p, with their corresponding meaning in terms of PSBC's parameters.

The content of each one of these dictionaries can be accessed in the same way as in the case of 1D computational statistics; see Section 4 of this guide.

In the same folder MNIST we also have the summary of data corresponding to the MNIST database, using the diffusive PSBC with Neumann boundary conditions shown in Section 5 of the paper.

- path_in_the_tree_MNIST_Neumann.p
- parameters_MNIST_Neumann.p

which are pickled dictionaries with the following keys,

Dictionary key	PSBC's parameters
W1S-Nt1	Diffusive, Weights-1-sharing, subordinate, Neumann B.C.'s, $N_t = 1$
W1S-Nt2	Diffusive, Weights-1-sharing, subordinate, Neumann B.C.'s, $N_t = 2$
W1S-Nt4	Diffusive, Weights-1-sharing, subordinate, Neumann B.C.'s, $N_t = 4$
W1S-Nt8	Diffusive, Weights-1-sharing, subordinate, Neumann B.C.'s, $N_t = 8$
WNtS-Nt1	Diffusive, Weights-N_t-sharing, subordinate, Neumann B.C.'s, $N_t = 1$
WNtS-Nt2	Diffusive, Weights-N_t-sharing, subordinate, Neumann B.C.'s, $N_t = 2$
WNtS-Nt4	Diffusive, Weights-N_t-sharing, subordinate, Neumann B.C.'s, $N_t = 4$
WNtS-Nt8	Diffusive, Weights-N_t-sharing, subordinate, Neumann B.C.'s, $N_t = 8$

Table 11: Keys in dictionary pickled as path_in_the_tree_MNIST_Neumann.p and parameters_MNIST_Neumann.p, with their corresponding meaning in terms of PSBC's parameters.

The content of each one of the dictionaries pickled above can be accessed in the same way as in the case of 1D computational statistics.

And finally, the last two pickled files correspond to data for the Periodic boundary conditions case

- path_in_the_tree_MNIST_Periodic.p
- parameters_MNIST_Periodic.p

whose data can be accessed in a similar way as before. They contain summarized data for the Diffusive PSBC with Periodic Boundary conditions shown in Section 6.3 of the paper, which are pickled dictionaries with the following keys,

Dictionary key	PSBC's parameters
Per_W1S-Nt1	Diffusive, Weights-1-sharing, subordinate, Periodic B.C.'s, $N_t = 1$
Per_W1S-Nt2	Diffusive, Weights-1-sharing, subordinate, Periodic B.C.'s, $N_t = 2$
Per_W1S-Nt4	Diffusive, Weights-1-sharing, subordinate, Periodic B.C.'s, $N_t = 4$
Per_W1S-Nt8	Diffusive, Weights-1-sharing, subordinate, Periodic B.C.'s, $N_t = 8$
Per_WNtS-Nt1	Diffusive, Weights-N_t-sharing, subordinate, Periodic B.C.'s, $N_t = 1$
Per_WNtS-Nt2	Diffusive, Weights-N_t-sharing, subordinate, Periodic B.C.'s, $N_t = 2$
Per_WNtS-Nt4	Diffusive, Weights-N_t-sharing, subordinate, Periodic B.C.'s, $N_t = 4$
Per_WNtS-Nt8	Diffusive, Weights-N_t-sharing, subordinate, Periodic B.C.'s, $N_t = 8$

Table 12: Keys in dictionary pickled as path_in_the_tree_MNIST_Periodic.p and parameters_MNIST_Periodic.p, with their corresponding meaning in terms of PSBC's parameters.

Now that we know what the content of each folder is, we describe what is summarized in each one of these dictionaries.

When unpickled, you will find a dictionary with the keys given in the tables associated with them. At each one of these keys you will find a sub-dictionary with keys given in the next table.

Dictionary key	Description
data	dictionary with basic information about the simulated model, with • dimension of parameter space (in Nx); • diffusion (in eps); • number of layers (in N_t); • partition cardinality (in ptt_cardnlty); • number K corresponding to weights-K-sharing property (in weights_K_sharing); • Boundary condition (in Neumann); • batch size (in batch_size); • number of epochs (in epochs); • if the phase is subordinate or not (in Subordinate, which is a boolean variable); • if the model has a phase in (with_phase, which is a boolean variable); • what - if any - parameter is being varied (in what_is_varying, which is a string).
best_accuracy_train	model's accuracy when evaluated at the point of best parameters, for all the simulations.
best_accuracy_test	accuracy of the model at best_par_U_model and best_par_P_model , for all the simulations.
cost	cost as evaluated at each epoch (realization of a single simulation, for illustrative purposes only).
diam_hist	dictionary with two entries, one for the evolution of $\mathscr{P}_\alpha$'s diameter, the other for the evolution of $\mathscr{P}_\beta$'s diameter; see the Invariant Region Enforcing Constraint (1.14) in the paper (realization of a single simulation, for illustrative purposes only, used in Figure 19 of the paper).
dt_hist	dictionary with two entries, one for the evolution of Δ_t^u along epochs, the other for the evolution of Δ_t^p along epochs (realization of a single simulation, for illustrative purposes only).
max_alpha	vector, where each entry contains the maximum value of $\mathscr{P}_\alpha$ over epochs for that simulation;
max_beta	vector, where each entry contains the maximum value of $\mathscr{P}_\beta$ over epochs for that simulation;
value_of_parameter_varying	Value of the parameter being varied in that set of simulations (either N_{ptt} or ε).

Table 13: Common keys to both dictionaries summarizing data for MNIST and 1D problems. This dictionary has more keys, described in each section corresponding to the type of problem analyzed.

Part V
Raw data: how it is stored and how to access them

Even though only statistical pickled files are useful to assert qualitative and quantitative properties of the PSBC, all the data generated in simulations is also available for reproducibility and comparison purposes.

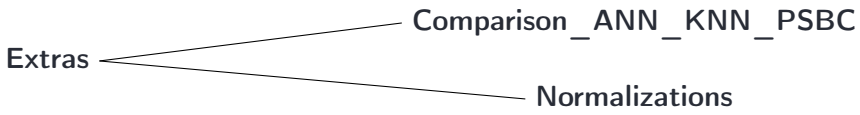
There are two main folders for data:

PSBC and **Extras**,

whose content we now explain

6 Extras

Extras is subdivided in two subfolder:



Tree 2: Contents in the folder **Extras**.

The Comparison_ANN_KNN_PSBC folder contains the simulations shown in Section 3.1 of the paper. Its content is made up of pickled files, as described below.

(i) For each value of N in $\{1, \dots 10\}$ it contains the results of 10 simulations of ANNs with 3 layers, 1 node in input and output layers, and N nodes in the hidden layer. Accuracies are stored.

(ii) A KNN method with 2 and 3 neighbors was also run. Accuracies are stored (only KNN with 2 neighbors has been used in the paper).

(iii) The PSBC in the form (2.1) is run 10 times. Model parameters, including its Accuracies, are stored.

The Normalizations folder contains the simulations shown in Section 6.10 of the paper, related to different types of normalizations that satisfy the Invariant Region Enforcing Condition 1.14.

Code is run as follows:

(i) A bash script downloads the MINIST data and normalizes it accordingly (σ equals to 0.1 or 0.2).

(ii) A python script generates a 5-fold split of the training set.

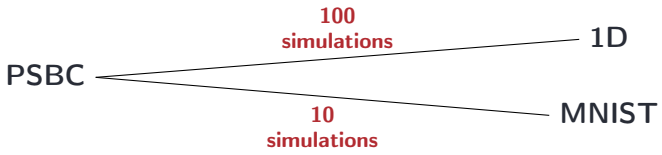
(iii) Another python script generates parameters to the model, besides a pickled file "grid.p" which contains a grid for learning rate grid search

(iv) A Main file runs the model on the grid, testing which learning rate is the best.

(v) Best accuracies are printed in a txt file. This data is shown in table 6, in Section 6.10 of the paper.

7 PSBC

The **PSBC** folder is subdivided in two other folders. Data is organized as shown in Tree 3; each folder has a similar - if not same - name, and each branch denotes that the leaf is a subfolder of the parent folder.



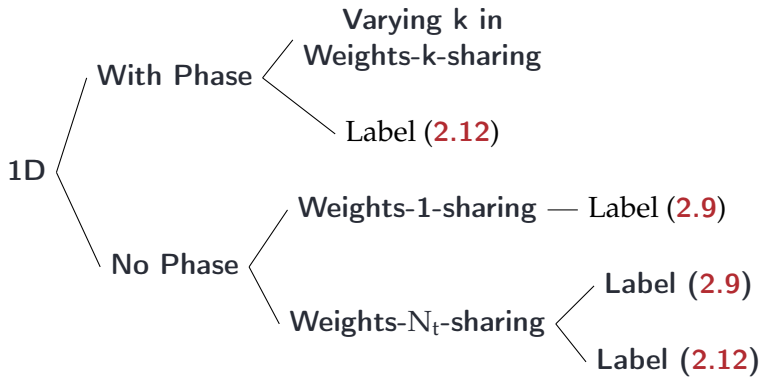
Tree 3: Contents in the folder **PSBC**.

In the arrows branching out of the PSBC node, we denote in red the number of simulations run in each case: as remarked in Section 1.2 of the paper, this is done to assert statistical properties and also due to the random initialization of trainable weights. We remark that the simulations in the 1D models are much bigger (in terms of storage) because we have saved the whole parameter story throughout the training.

Below, I explain the content of each subfolder - 1D and MNIST - in detail.

7.1 PSBC/1D

The contents of the 1D folder are described as shown in the Tree 4. As mentioned above, they concern the simulations for the toy problem of Sections 2 and 3 of the paper.



Tree 4: Contents in the folder **1D**.

The final leaf in the tree contains the corresponding PSBC simulation, with parameters as defined by the folder's name. For instance, following the branch

$$\textbf{PSBC} > \textbf{1D} > \textbf{No Phase} > \textbf{Weights-1-sharing} > \textbf{Label (2.9)} \tag{1}$$

where simulation data for for each one of the cases $\gamma^* \in \{0.6, 0.7, 0.8\}$ are stored, each in a separate folder. Folder names are the same as keys' names in Table 6; for instance, the folder path corresponding to (1) is **PSBC/1D/no_phase/weights_1_sharing/W1S-NP**.

For each case, 100 simulations were run, corresponding to the PSBC with 1 feature, as described in the toy problem with labels (2.9).

The final folder contains the output of such simulation, with a file **Full_model_parameters.p**; the latter contains a pickled dictionary with keys given in Tables 14.

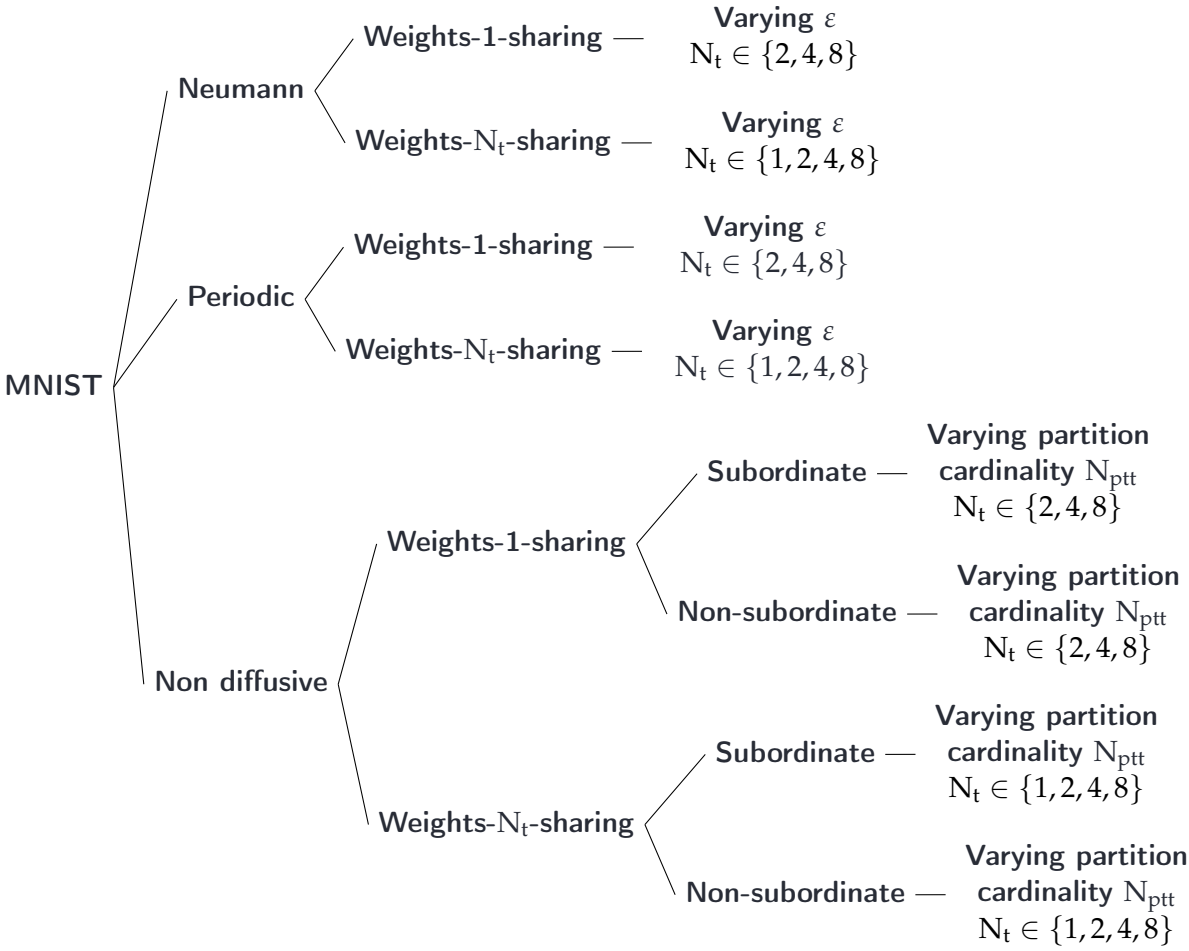
Dictionary key	Description
par_U_model	copy of parameters of trainable weights W_U as of the last epoch epoch.
best_par_U_model	copy of parameters of trainable weights W_U at epoch with highest accuracy.
par_P_model	copy of parameters of trainable weights W_P as of the last epoch epoch.
best_par_P_model	copy of parameters of trainable weights W_P at epoch with highest accuracy.
cost_non_averaged	cost as evaluated in every minibatches.
cost	cost as evaluated at each epoch.
diam_hist	dictionary with two entries, one for the evolution of $\mathscr{P}_\alpha$'s diameter, the other for the evolution of $\mathscr{P}_\beta$'s diameter (see the Invariant Region Enforcing Constraint in the paper).
dt_hist	dictionary with two entries, one for the evolution of Δ_t^u along epochs, the other for the evolution of Δ_t^p along epochs.
best_epoch	epoch in which the highest accuracy was obtained.
eps_now	if we are varying partition cardinality, this is the parameter being varied at the moment; for instance, if one is varying partition cardinality, the value of eps_now contains the partition cardinality. If diffusion is being varied, it will contain the diffusion value.
accuracy_train	evolution of model's accuracy when evaluated at the training set.
best_accuracy_train	model's accuracy when evaluated at the point of best parameters.
best_predic_vector_train	vector with predicted labels for the training set at best_par_U_model and best_par_P_model.
best_accuracy_test	accuracy of the model at best_par_U_model and best_par_P_model.
best_predic_vector_test	vector with predicted labels for test set at best_par_U_model and best_par_P_model.
par_U_wrt_epochs	only available at the 1D simulations, where parameters have been saved throughout the whole training. It is also a dictionary, where each key is a number with a copy of the trainable weights W_U at each epoch along training.
par_P_wrt_epochs	only available at the 1D simulations, where parameters have been saved throughout the whole training. It is also a dictionary, where each key is a number with a copy of the trainable weights W_P at each epoch along training.

Table 14: Correspondence table for PSBS parameters and keys used in the dictionary pickled as Full_model_parameters.p, containing the output of the non-diffusive PSBC applied to the 1D toy problem.

7.2 PSBC/MNIST

This folder contains many different simulations for the PSBC with different parameters. In this case you might notice some asymmetries in the branches of the tree: for instance, No weights-1-sharing model has a leaf $N_t = 1$. That's just a matter of labeling, because weights-1-sharing and weights-N_t-sharing are exactly the same.

The folder tree is given in Tree 5.



Tree 5: Contents in the folder **MNIST**.

Remark 7.1 (Companion folders: model selection for learning rates) *Except for the subfolders of the branch Periodic, each leaf has a companion folder where we do model selection for the leaning rate. This part is standard, and won't be discussed here. For information, see the code in [2] or the discussion in Section A of the paper.*

As before, each branch in the above tree is read as a PSBC parameter: for instance, the uppermost branch, with $N_t = 2$ corresponds to a diffusive PSBC with Neumann boundary conditions, weights-1-sharing, and $N_t = 2$. In this case, the term "Varying ε" means that inside this folder one finds several subfolders each of them with a PSBC with different value of ε; such a folder is mostly represented with an ending "**eps***", where * varies from 1 to 13.

Similarly, the lowermost branch with $N_t = 2$ corresponds to a non-diffusive PSBC with weights-1-sharing, Non-subordinate, and $N_t = 2$. In this case, the term "Varying partition cardinality " means that inside this folder one finds several subfolders each of them with a PSBC with different value of N_{ptt}; such a folder is mostly represented with an ending "**ptt***", where * varies from 1 to 10.

For each case, 10 simulations were run. Each simulation's output is stored in a dictionary as a pickled file, with name **Full_model_parameters.p**, in a Folder with name **simulation***/, where * varies from 1 to 10. **Full_model_parameters.p** contains a pickled dictionary with keys given in Table 15.

Dictionary key	Description
par_U_model	copy of parameters of trainable weights W_U as of the last epoch epoch.
best_par_U_model	copy of parameters of trainable weights W_U at epoch with highest accuracy.
par_P_model	copy of parameters of trainable weights W_P as of the last epoch epoch.
best_par_P_model	copy of parameters of trainable weights W_P at epoch with highest accuracy.
cost_non_averaged	cost as evaluated in every minibatches.
cost	cost as evaluated at each epoch.
diam_hist	dictionary with two entries, one for the evolution of $\mathscr{P}_\alpha$'s diameter, the other for the evolution of $\mathscr{P}_\beta$'s diameter (see the Invariant Region Enforcing Constraint in the paper).
dt_hist	dictionary with two entries, one for the evolution of Δ_t^u along epochs, the other for the evolution of Δ_t^p along epochs.
best_epoch	epoch in which the highest accuracy was obtained.
eps_now	if we are varying partition cardinality, this is the parameter being varied at the moment; for instance, if one is varying partition cardinality, the value of eps_now contains the partition cardinality. If diffusion is being varied, it will contain the diffusion value.
accuracy_train	evolution of model's accuracy when evaluated at the training set.
best_accuracy_train	model's accuracy when evaluated at the point of best parameters.
best_predic_vector_train	vector with predicted labels for the training set at best_par_U_model and best_par_P_model.
best_accuracy_test	accuracy of the model at best_par_U_model and best_par_P_model.
best_predic_vector_test	vector with predicted labels for test set at best_par_U_model and best_par_P_model.

Table 15: Correspondence table for PSBS parameters and keys used in the dictionary pickled as Full_model_parameters.p, containing the output of the PSBC applied to the MNIST database.

References

[1] R. Monteiro. Data repository for the paper "Binary classification as a phase separation process". https://dx.doi.org/10.5281/zenodo.4005131, September 2020.

[2] R. Monteiro. Source code for the paper "Binary classification as a phase separation process". https://github.com/rafael-a-monteiro-math/Binary_classification_phase_separation, September 2020.