PYERUALJETWORK 3.0 USER MANUAL

Author: Hasan Can Beydili

ABOUT PYERUAL JETWORK:

Pyerual Jetwork is a machine learning library written in Python for professionals, incorporating advanced, unique, new, and modern techniques. Its most important component is the PLAN (Potentiation Learning Artificial Neural Network).

Both the PLAN algorithm and the PyerualJetwork library were created by Author, and all rights are reserved by Author.

Pyerual Jetwork is free to use for commercial business and individual users.

As of 08/21/2024, the library includes only the PLAN module, but other machine learning modules are expected to be added in the future.

The PLAN algorithm will not be explained in this document. This document focuses on how professionals can integrate and use Pyerual Jetwork in their systems. However, briefly, the PLAN algorithm can be described as a classification algorithm. PLAN algorithm achieves this task with an incredibly energy-efficient, fast, and hyperparameter-free user-friendly approach. For more detailed information, you can check out 'PLAN.pdf' file.

The functions of the Pyerual Jetwork modules, uses snake_case written style.

HOW DO I IMPORT IT TO MY PROJECT?

Anaconda users can access the 'Anaconda Prompt' terminal from the Start menu and add the necessary library modules to the Python module search queue by typing "pip install pyerualjetwork" and pressing enter. If you are not using Anaconda, you can simply open the 'cmd' Windows command terminal from the Start menu and type "pip install pyerualjetwork". (Visual Studio Code reccomended) After installation, it's important to periodically open the terminal of the environment you are using and stay up to date by using the command "pip install pyerualjetwork --upgrade". As of 06/04/2024, the most current version is "2.0.5". This version is the successor to version "1.0" of the library. Previous versions were for testing purposes. The latest version was “3.3.2” at the time this document was written

- After installing the module using "pip" you can now call the library module in your project environment. Use: “import plan”. Now, you can call the necessary functions from the plan module.

MAIN FUNCTIONS:

1. **fit (2Args - 8Args)**
2. **evaluate (5Args -6Args)**
3. **save_model (8Args - 9Args)**
4. **load_model (2Args)**
5. **predict_model_ssd (3Args)**
6. **predict_model_ram (3Args - 4Args)**
7. **auto_balancer (2Args)**
8. **synthetic_augmentation (2Args)**
9. **get_weights ()**
10. **get_df ()**
11. **get_preds ()**
12. **get_acc ()**
13. **get_pot ()**
14. **encode_one_hot (2Args)**
13. **split (4Args)**
14. **metrics (3Args)**
15. **decode_one_hot (1Args)**
16. **roc_curve (2Args)**
17. **confusion_matrix (3Args)**
18. **plot_decision_space (2Args)**
19. **standard_scaler(1Args - 3Args)**
20. **manuel_balancer(3Args)**
21. **weight_normalization (2Args)**

Yellow labeled parameters is not necessary.

Like this labeled functions highly recommended to use.

Green labeled parameters necessary.

1. fit (x_train, y_train, show_training, val, val_count, x_val, y_val, activation_potentiation)

The purpose of this function, as the name suggests, is to train the model.

a. **x_train**: A list consisting of input vectors or matrices representing training examples.
b. **y_train**: One-hot encoded list of train labels.
c. **show_training**: show plan training process ? (True or None) (val parameter must: True.)
d. **val:** validation during training process ? None or True Default: None (optional)
e. **val_count:** After how many examples learned will an accuracy test be performed? Default: 0.1 (approximately
f. %10) (optional)
g. **x_val**: list of validation data. Default: x_train (optional)
h. **y_val**: list of validation labels. Default: y_train (optional)
i. **activation_potentiation:** hyperparameter for ‘Deep PLAN’ algorithm activation functions list. Default is: None(‘linear activation’). For full list enter this code: help(plan.activations_list)

The output of this function Weight matrix of model.

2. evaluate (x_test, y_test, show_metrics, bar_status, W, activation_potentiation)

This function calculates the test accuracy of the model using the inputs and labels set aside for testing, along with the weight matrices and other model parameters obtained as output from the training function.

a. W: A numpy weight matrix of model. (Descriptions of other parameters are the same as Function 1)
b. show_metrics: Show_metrics or not show_metrics test process. May be (True or False)

c. bar_status: Evaluate progress have a loading bar ? (True or None) Default: True.

The outputs of this function are, in order: weights of test process, a list of test predictions, and test accuracy rate.

3. save_model (model_name, model_type, test_acc, weights_type, weights_format, model_path, scaler_params, W, activation_potentiation)

This function creates log files in the form of a pandas DataFrame containing all the parameters and information of the trained and tested model, and saves them to the specified location along with the weight matrices.

a. model_name: Specify the name of the model to be saved, e.g.: 'Model1'.
b. model_type: Specify the type of the model to be saved, e.g.: 'PLAN'.
c. test_acc: Test accuracy rate.
d. weights_type: Specify the file extension of the weight matrices to be saved, e.g.: 'txt' or 'mat'.
e. weights_format: Specify the numeric data type of the weight matrices to be saved, e.g.: 'd' = integer, 'f' = floating-point, 'raw' = saves as is. (STRONG RECCOMENDATION: enter: ‘f’ or ‘raw’.
f. model_path: Specify the file path where the model will be saved. (Note: Use '/' when specifying the location and use '/' again at the end.) (W and other parameters have been mentioned above.)
g. scaler_params: If trained model scaled, give the first return of “standard_scaler” function. Othercase give: None

This function returns messages such as 'saved' or 'could not be saved' as output.

4. load_model (model_name, model_path)

This function retrieves everything about the model into the Python environment from the saved log file and the model name.

This function returns the following outputs in order: W, activation_potentiation, df (Data frame of model).

5. predict_model_ssd (Input, model_name, model_path)

This function loads the model directly from its saved location, predicts a requested input, and returns the output. (It can be integrated into application systems and the output can be converted to .json format and used in web applications.)

a. Input: The real-world example to be predicted.

This function returns the last output layer of the model as the output of the given input.

6. predict_model_ram (Input, scaler_params, W, activation_potentiation)

This function predicts and returns the output for a requested input using a model that has already been loaded into the program (located in the computer's RAM). (It can be integrated into application systems and the output can be converted to .json format and used in web applications.) (Other parameters are information about the model and are defined as described and listed above.)

This function returns the last output layer of the model as the output of the given input.

7 . auto_balancer (x_train, y_train)

This function aims to balance all training data according to class distribution before training the model. All data is reduced to the number of data points of the class with the least number of examples.

This function returns the following outputs in order: a list containing the balanced training data and a list containing the balanced training labels.

8 . synthetic_augmentation (x_train, y_train)

This function creates synthetic data samples with given data samples for balance data distribution.

This function returns the following outputs in order: a list containing the balanced training data and a list containing the balanced training labels. or testing labels.

9. get_weights()

This function returns wight matrices list of the selected model. For exp:

```
test_model = plan.evaluate(x_train, y_train)
W = test_model[plan.get_weights()]
```

10. get_df ()

Returns pandas data frame of the selected model

11. get_preds()

Returns predictions list of the selected model

12. get_acc()

Returns accuracy of the selected model

13. get_pot()

Returns activation potential of the selected model.

14. encode_one_hot(y_train, y_test)

Returns one hot encoded labels.

15. split (X, y, test_size, random_state)

This function splits all data for train and test

a. **test_size**: Size of test data (0 – 1 float) .
b. **random_state**: Seed information (exmpl: 42)

Returns: x_train, x_test, y_train, y_test

16. metrics (y_test, y_preds, average)

This function calculates precision, recall and f1 score of model.

a. **y_preds**: One hot decoded test prediction list.
b. **average:** ‘micro’ or ‘macro’ or ‘weighted’. Default is ‘weighted’.

Returns: precision, recall, f1.

17. decode_one_hot (y_test)

Returns: decoded y_test given input

18. roc_curve (y_test, y_preds)

Returns: fpr, tpr, thresholds

19. confusion_matrix (y_test, y_preds, class_count)

Returns: confusion matrix

20. plot_decision_space (x, y)

Plots decision boundary like data distrubition.

21. standard_scaler (x_train, x_test, scaler_params)

a. scaler_params: **Standard scaled x_train parameters for model predictions. (mean, std)**

Returns: If x_test given then returns: standart scaled parameters, standard scaled x_train, standard scaled y_test. If x_test is not given then returns: standard scaled parameters, standard scaled x_train.

22. manuel_balancer (x_train, y_train target_samples_per_class)

Same operation of auto_balacner, but this function gives the limit of sample addition to user.

Returns: x_train, y_train

23. weight_normalization (W, class_count)

Some cases need to normalize neurons. This function does this operation.

Mostly in unbalanced training cases.

Returns: W

ERROR MESSAGES:

100 Type Errors: Value Errors.

200 Type Errors: Variable type errors.

300 Type Errors: List length errors.

LAST PART:

Despite being in its early stages of development, Pyerual Jetwork has already demonstrated its potential to deliver valuable services and solutions in the field of machine learning. Notably, it stands as the first library dedicated to PLAN (Potentiation Learning Artificial Neural Network), embracing innovation and welcoming new ideas from its users with open arms. Recognizing the value of diverse perspectives and fresh ideas, I, Hasan Can Beydili, the creator of Pyerual Jetwork, am committed to fostering an open and collaborative environment where users can freely share their thoughts and suggestions. The most promising contributions will be carefully considered and potentially integrated into the Pyerual Jetwork library. For your suggestions, lists and feedback, my e-mail address is: tchasancan@gmail.com

And finally, trust the PLAN…