ANAPLAN 1.5.7 USER MANUAL

Author: Hasan Can Beydili

ABOUT ANAPLAN:

Anaplan 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 Anaplan library were created by Author, and all rights are reserved by Author.

Anaplan is free to use for commercial business and individual users.

It is prohibited to copy or share the code and these documents by duplicating or using different names.

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.

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 anaplan" 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 anaplan". (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 anaplan --upgrade". The latest version was “1.0.3” 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: “from anaplan import plan”. Now, you can call the necessary functions from the plan module.

MAIN FUNCTIONS:

1. **fit ()**
2. **evaluate ()**
3. **save_model ()**
4. **load_model ()**
5. **predict_model_ssd ()**
6. **predict_model_ram ()**
7. **auto_balancer ()**
8. **synthetic_augmentation ()**
9. **get_weights ()**
10. **get_scaler ()**
11. **get_preds ()**
12. **get_acc ()**
13. **get_act_pot ()**
14. **encode_one_hot ()**
15. **split ()**
16. **metrics ()**
17. **decode_one_hot ()**
18. **roc_curve ()**
19. **confusion_matrix ()**
20. **plot_decision_space ()**
21. **standard_scaler()**
22. **manuel_balancer()**
23. **weight_normalization ()**
24. **learner()**
25. **activations_list ()**

The functions of the Anaplan modules, uses snake_case written style.

1. fit ()

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

```
a.      fit Args:
b.         x_train (list[num]): List or numarray of input data.
c.
d. y_train (list[num]): List or numarray of target labels. (one hot
   encoded)
e.
f. val (None or True): validation in training process ? None or True
   default: None (optional)
g.
h. val_count (None or int): After how many examples learned will an
   accuracy test be performed? default: 10=(%10) it means every
   approximately 10 step (optional)
i.
j. activation_potentiation (list): For deeper PLAN networks, activation
   function parameters. For more information please run this code:
   plan.activations_list() default: [None] (optional)
k.
l. x_val (list[num]): List of validation data. default: x_train (optional)
m.
n. y_val (list[num]): (list[num]): List of target labels. (one hot encoded)
   default: y_train (optional)
o.
p. show_training (bool, str): True or None default: None (optional)
q.
r. LTD (int): Long Term Depression Hyperparameter for train PLAN neural
   network default: 0 (optional)
s.
t. interval (float, int): frame delay (milisecond) parameter for Training
   Report (show_training=True) This parameter effects to your Training
   Report performance. Lower value is more diffucult for Low end PC's
   (33.33 = 30 FPS, 16.67 = 60 FPS) default: 100 (optional)
u.
v. decision_boundary_status (bool): If the visualization of validation and
   training history is enabled during training, should the decision
```

```
   boundaries also be visualized? True or False. Default is True.
   (optional)
w.
x. train_bar (bool): Training loading bar? True or False. Default is True.
   (optional)
y.
z. auto_normalization(bool): Normalization process during training. May
   effect training time and model quality. True or False. Default is True.
   (optional)
aa.
bb.neurons_history (bool, optional): Shows the history of changes that
   neurons undergo during the CL (Cumulative Learning) stages. True or
   False. Default is False. (optional)
cc.
dd.    Returns:
ee.        numpyarray([num]): (Weight matrix).
```

The output of this function Weight matrix of model.

2. evaluate ()

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. x_test (list[num]): Test input data.
b.
c. y_test (list[num]): Test labels.
d.
e. W (list[num]): Weight matrix list of the neural network.
f.
g. activation_potentiation (list): For deeper PLAN networks, activation
   function parameters. For more information please run this function:
   'plan.activations_list()' default: ['linear']
h.
i. loading_bar_status:  Evaluate progress have a loading bar ? (True or
   False) Default: True.
j.
k. show_metrics (bool): (True or None) (optional) Default: None
```

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

3. save_model ()

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. Function to save a potentiation learning model.
b.
c.     Arguments:
d.     model_name (str): Name of the model.
e.     model_type (str): Type of the model. default: 'PLAN'
f.     test_acc (float): Test accuracy of the model. default: None
g.
h. weights_type (str): Type of weights to save (options: 'txt', 'pkl',
   'npy', 'mat'). default: 'npy'
i.
j. WeightFormat (str): Format of the weights (options: 'f', 'raw').
   default: 'raw'
k.
l. model_path (str): Path where the model will be saved. For example:
   C:/Users/beydili/Desktop/denemePLAN/ default: ''
m.
n. scaler_params (list[num, num]): standard scaler params list: mean,std.
   If not used standard scaler then be: None.
o.
p.     W: Weights of the model.
q.
r. activation_potentiation (list): For deeper PLAN networks, activation
   function parameters. For more information please run this code:
   plan.activations_list() default: ['linear']
s.
t.     Returns:
u.     str: Message indicating if the model was saved successfully or
   encountered an error.
```

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

4. load_model ()

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

```
a. model_name (str): Name of the model.
b. model_path (str): Path where the model is saved.
```

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

5. predict_model_ssd ()

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 (list or ndarray): Input data for the model (single
   vector or single matrix).
b. model_name (str): Name of the model.
```

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

6. predict_model_ram ()

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.)

```
a.     Input (list or ndarray): Input data for the model (single vector or
   single matrix).
b.
c.     W (list of ndarrays): Weights of the model.
d.
e.     scaler_params (list): standard scaler params list: mean,std.
   (optional) Default: None.
f.
g.     activation_potentiation (list): ac list for deep PLAN. default:
   [None] ('linear') (optional)
```

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

7 . auto_balancer ()

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.

```
a.    x_train (list): Input data for training.
b.    y_train (list): Labels corresponding to the input data.
```

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 ()

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

```
a.      x -- Input dataset (examples) – array format
b.      y -- Class labels (one-hot encoded) - array format
```

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_test, y_test)

W = test_model[plan.get_weights()]

10. get_scaler ()

Returns scaler_params of the selected model For exp:

model = plan.load_model(model_name=”example”, model_path=””)

scaler_params = model[plan.get_scaler()]

11. get_preds()

Returns predictions list of the selected model

12. get_acc()

Returns accuracy of the selected model

13. get_act_pot()

Returns activation potential of the selected model.

14. encode_one_hot(y_train, y_test)

```
a.          y_train (numpy.ndarray): Labeled train data.
b.          y_test (numpy.ndarray): Labeled test data.
```

Returns one hot encoded labels.

15. split ()

This function splits all data for train and test

```
a.          X (numpy.ndarray): Features data.
b.
c.          y (numpy.ndarray): Labels data.
d.
```

```
e.          test_size (float or int): Proportion or number of samples for
   the test subset.
f.
g.          random_state (int or None): Seed for random state.
```

Returns: x_train, x_test, y_train, y_test

16. metrics ()

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

```
a.          y_ts (list or numpy.ndarray): True labels.
b.          test_preds (list or numpy.ndarray): Predicted labels.
c.          average (str): Type of averaging ('micro', 'macro', 'weighted').
```

Returns: precision, recall, f1.

17. decode_one_hot (y_test)

```
a. encoded_data (numpy.ndarray): One-hot encoded data with shape
   (n_samples, n_classes).
```

Returns: decoded y_test given input

18. roc_curve ()

```
a.      y_true : (array), shape = [n_samples]
b.          True binary labels in range {0, 1} or {-1, 1}.
c.
d.      y_score : (array), shape = [n_samples]
e.
```

```
f.          Target scores, can either be probability estimates of the
   positive class,
g.          confidence values, or non-thresholded measure of decisions
   (as returned
h.          by decision_function on some classifiers).
```

Returns: fpr, tpr, thresholds

19. confusion_matrix (y_test, y_preds, class_count)

```
a.          y_true (numpy.ndarray): True class labels (1D array).
b.          y_pred (numpy.ndarray): Predicted class labels (1D array).
c.          num_classes (int): Number of classes.
```

Returns: confusion matrix

20. plot_decision_space (x, y)

Plots decision boundary like data distrubition.

21. standard_scaler ()

```
a. train_data: numpy.ndarray
b.
c. test_data: numpy.ndarray (optional)
d.
e. scaler_params (optional for using model)
```

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 ()

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

```
a.      x_train -- Input dataset (examples) - NumPy array format
b.
c.      y_train -- Class labels (one-hot encoded) - NumPy array format
d.
e.      target_samples_per_class -- Desired number of samples per class
```

Returns: x_train, y_train

23. weight_normalization ()

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

Mostly in unbalanced training cases.

```
a.              W (list(num)): Trained weight matrix list.
b.              class_count (int): Class count of model.
```

Returns: W

24. learner ()

Optimizes the activation functions for a neural network by leveraging train data to find the most accurate combination of activation potentiation for the given dataset.

This next-generation generalization function includes an advanced learning feature that is specifically tailored to the PLAN algorithm.

It uniquely adjusts hyperparameters based on test accuracy while training with model-specific training data, offering an unparalleled optimization technique.

Designed to be used before model evaluation. This called TFL(Test Feedback Learning).

```
    Args:
        x_train (array-like): Training input data.

        y_train (array-like): Labels for training data.

        x_test (array-like, optional): Test input data (for improve next gen generilization). If test data is not given then train feedback learning active

        y_test (array-like, optional): Test Labels (for improve next gen generilization). If test data is not given then train feedback learning active

        strategy (str, optional): Learning strategy. (options: 'accuracy', 'loss', 'f1', 'precision', 'recall', 'adaptive_accuracy', 'adaptive_loss', 'all'): 'accuracy', Maximizes test accuracy during learning. 'f1', Maximizes test f1 score during learning. 'precision', Maximizes test preciison score during learning. 'recall', Maximizes test recall during learning. loss', Minimizes test loss during learning. 'adaptive_accuracy', The model compares the current accuracy with the accuracy from the past based on the number specified by the patience value. If no improvement is observed it adapts to the condition by switching to the 'loss' strategy quickly starts minimizing loss and continues learning. 'adaptive_loss',The model adopts the 'loss' strategy until the loss reaches or falls below the value specified by the patience parameter. However, when the patience threshold is reached, it automatically switches to the 'accuracy' strategy and begins to maximize accuracy. 'all', Maximizes all test scores and minimizes test loss, 'all' strategy most strong and most robust strategy. Default is 'accuracy'.

        patience ((int, float), optional): patience value for adaptive strategies. For 'adaptive_accuracy' Default value: 5. For 'adaptive_loss' Default value: 0.150.

        depth (int, optional): The depth of the PLAN neural networks Aggreagation layers.
```

```
o.
p.          batch_size (float, optional): Batch size is used in the
   prediction process to receive test feedback by dividing the test data
   into chunks and selecting activations based on randomly chosen
   partitions. This process reduces computational cost and time while
   still covering the entire test set due to random selection, so it
   doesn't significantly impact accuracy. For example, a batch size of
   0.08 means each test batch represents 8% of the test set. Default is
   1. (%100 of test)
q.
r.          auto_normalization (bool, optional): If auto
   normalization=False this makes more faster training times and much
   better accuracy performance for some datasets. Default is True.
s.
t.          early_shifting (int, optional): Early shifting checks if the
   test accuracy improves after a given number of activation attempts
   while inside a depth. If there's no improvement, it automatically
   shifts to the next depth. Basically, if no progress, it’s like,
   “Alright, let’s move on!”  Default is False
u.
v.          early_stop (bool, optional): If True, implements early
   stopping during training.(If test accuracy not improves in two depth
   stops learning.) Default is False.
w.
x.          show_current_activations (bool, optional): Should it display
   the activations selected according to the current strategies during
   learning, or not? (True or False) This can be very useful if you want
   to cancel the learning process and resume from where you left off
   later. After canceling, you will need to view the live training
   activations in order to choose the activations to be given to the
   'start_this' parameter. Default is False
y.
z.          show_history (bool, optional): If True, displays the training
   history after optimization. Default is False.
aa.
bb.         loss (str, optional): For visualizing and monitoring. PLAN
   neural networks doesn't need any loss function in training(if
   strategy not 'loss'). options: ('categorical_crossentropy' or
   'binary_crossentropy') Default is 'categorical_crossentropy'.
cc.
dd.         interval (int, optional): The interval at which evaluations
   are conducted during training. (33.33 = 30 FPS, 16.67 = 60 FPS)
   Default is 100.
ee.
ff.         target_acc (int, optional): The target accuracy to stop
   training early when achieved. Default is 2 (off).
gg.
hh.         target_loss (float, optional): The target loss to stop
   training early when achieved. Default is -1 (off).
```

```
ii.
jj.         except_this (list, optional): A list of activations to
   exclude from optimization. Default is None. (For avaliable activation
   functions, run this function: plan.activations_list())
kk.
ll.         only_this (list, optional): A list of activations to focus on
   during optimization. Default is None. (For avaliable activation
   functions, run this code: plan.activations_list())
mm.
nn.         start_this (list, optional): To resume a previously canceled
   or interrupted training from where it left off, or to continue from
   that point with a different strategy, provide the list of activation
   functions selected up to the learned portion to this parameter.
   Default is None
oo.
pp.         neurons_history (bool, optional): Shows the history of
   changes that neurons undergo during the TFL (Test Feedback Learning)
   stages. True or False. Default is False.
qq.
rr.     Returns:
ss.         tuple: A list for model parameters: [Weight matrix, Test
   loss, Test Accuracy, [Activations functions]].
tt.
uu.
```

Returns: model(list)

25. activations_list()

This function includes all avaliable activations list in the modüle

Returns: all avaliable activations

LAST PART:

Despite being in its early stages of development, Anaplan 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 Anaplan, 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 Anaplan library. For your suggestions, lists and feedback, my e-mail address is: tchasancan@gmail.com

And finally, trust the PLAN…