Aim: Write a Python program to illustrate ART Neural Network.

Software requirement for Python: Jupyter Notebook

Theory:

Adaptive Resonance Theory ART networks, as the name suggests, is always open to new learning adaptive without losing the old patterns resonance. Basically, ART network is a vector classifier which accepts an input vector and classifies it into one of the categories depending upon which of the stored pattern it resembles the most.

The main operation of ART classification can be divided into the following phases −

Recognition phase − The input vector is compared with the classification presented at every node in the output layer. The output of the neuron becomes "1" if it best matches with the classification applied, otherwise it becomes "0".

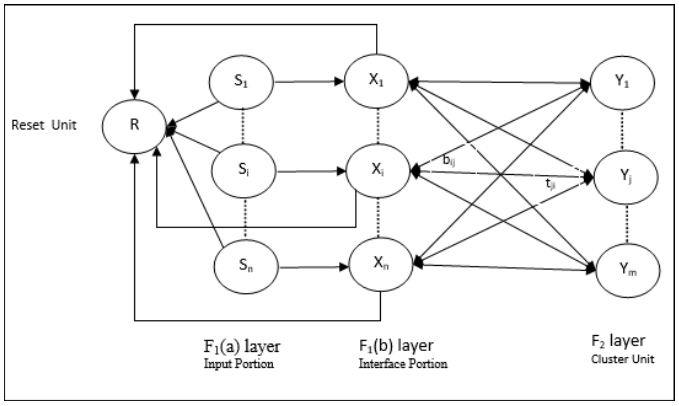
Comparison phase − In this phase, a comparison of the input vector to the comparison layer vector is done. The condition for reset is that the degree of similarity would be less than vigilance parameter.

Search phase − In this phase, the network will search for reset as well as the match done in the above phases. Hence, if there would be no reset and the match is quite good, then the classification is over. Otherwise, the process would be repeated and the other stored pattern must be sent to find the correct match.

ART1

It is a type of ART, which is designed to cluster binary vectors. We can understand about this with the architecture of it.

Architecture of ART1



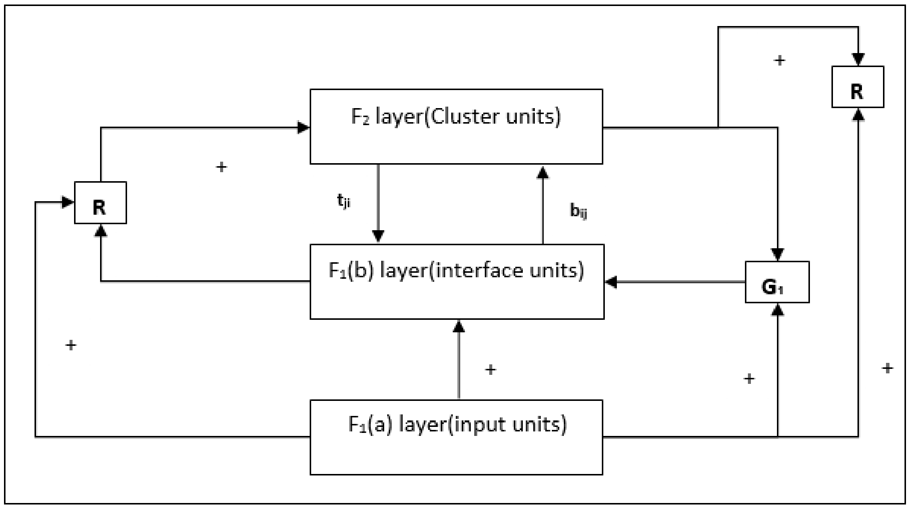


Fig.1-ART1 ARCHITECTURE

The ART1 Architecture (Fig.1) consists of the following two units –

Computational Unit − It is made up of the following −

Input unit (F1 layer) − It further has the following two portions −

F1a layer Input portion

− In ART1, there would be no processing in this portion rather than having the input vectors only. It is connected to F1b layer interface portion.

F1b layer Interface portion

− This portion combines the signal from the input portion with that of F2 layer. F1b layer is connected to F2 layer through bottom-up weights bij and F2 layer is connected to F1b layer through top-down weights tji.

Cluster Unit (F2 layer) – This is a competitive layer. The unit having the largest net input is selected to learn the input pattern. The activation of all other cluster unit is set to 0.

Reset Mechanism – The work of this mechanism is based upon the similarity between the top-down weight and the input vector. Now, if the degree of this similarity is less than the vigilance parameter, then the cluster is not allowed to learn the pattern and a rest would happen.

Supplement Unit – Actually the issue with Reset mechanism is that the layer F2 must have to be inhibited under certain conditions and must also be available when some learning happens. That is why two supplemental units namely, G1 and G2 is added along with reset unit, R. They are called gain control units. These units receive and send signals to the other units present in the network. '+' indicates an excitatory signal, while '−' indicates an inhibitory signal.

Procedure:

This code defines a class called ARTNetwork which is an implementation of an Adaptive Resonance Theory neural network. The network is trained on a set of input vectors represented by a numpy array X and can make predictions on new input vectors.

The network has a fixed number of categories specified by num_categories and a vigilance parameter specified by vigilance. The categories list holds the category vectors learned by the network during training.

During training, the network tries to fit each input vector to an existing category vector or create a new category vector if no match is found. This is done by first computing the similarity between the input vector and each existing category vector. If the similarity is above the vigilance threshold, the input vector is added to the category after updating it. If not, the input vector is checked for complementarity with the existing category vectors. If a complementary category is found, a new category vector is created by taking the maximum element-wise values between the input vector and the complementary category vector.

The predict method of the network returns the index of the matching category vector for a given input vector or None if no match is found.

The code tests the network on three input vectors and prints the predicted category index for each.

Conclusion: Thus, we have studied ART Neural Network with Python Programming.

Program Code and Output

```
import numpy as np
class ARTNetwork:
    def __init__(self, num_categories, vigilance):
        self.num_categories = num_categories
        self.vigilance = vigilance
        self.categories = []

    def _initialize_categories(self, input_vec):
        self.categories.append(input_vec)

    def _compute_similarity(self, input_vec, category):
        return np.dot(input_vec, category) / np.sum(input_vec)

    def _get_matching_category(self, input_vec):
        for category in self.categories:
            if self._compute_similarity(input_vec, category) >= self.vigilance:
                return category
        return None

    def _get_complementary_category(self, input_vec):
        for category in self.categories:
            if np.all(category + input_vec <= 1):
                return category
        return None

    def _update_category(self, input_vec, category):
        return np.maximum(input_vec, category)
    def _create_new_category(self, input_vec, complementary_category):
        return np.maximum(input_vec, complementary_category)

    def learn(self, input_vec):
        if not self.categories:
            self._initialize_categories(input_vec)
        else:
            matching_category = self._get_matching_category(input_vec)
            if matching_category is not None:
                self.categories.remove(matching_category)
                self.categories.append(self._update_category(input_vec, matching_category))
            else:
                complementary_category = self._get_complementary_category(input_vec)
                if complementary_category is not None:
                    self.categories.append(self._create_new_category(input_vec, complementary_category))
                else:
                    self.categories.append(input_vec)

    def predict(self, input_vec):
        matching_category = self._get_matching_category(input_vec)
        if matching_category is not None:
            return self.categories.index(matching_category)
        else:
            return None

# Define the input data
X = np.array([[1, 0, 0, 1],
              [1, 1, 0, 0],
              [0, 0, 1, 1]])

# Initialize the ART network
art = ARTNetwork(num_categories=2, vigilance=0.5)

# Train the ART network
for input_vec in X:
    art.learn(input_vec)

# Test the ART network
print("Prediction for [1, 1, 1, 0]:", art.predict(np.array([1, 1, 1, 0])))
print("Prediction for [0, 1, 1, 0]:", art.predict(np.array([0, 1, 1, 0])))
print("Prediction for [0, 0, 1, 0]:", art.predict(np.array([0, 0, 1, 0])))
```

OUTPUT OF THE CODE

```
Prediction for [1, 1, 1, 0]: 0

Prediction for [0, 1, 1, 0]: 0

Prediction for [0, 0, 1, 0]: 0
```

The output of 0 for all three predictions indicates that the ART network has categorized each input vector into the first category it created during initialization. This may be due to the chosen vigilance value of 0.5, which is relatively high and may lead to a lower number of categories being formed. Additionally, the input vectors in the X array are quite similar, which may have contributed to the ART network grouping them together into a single category. To improve the accuracy of the predictions, the vigilance parameter and input vectors could be adjusted to create more distinct categories.