Vishwakarma Institute of Technology, Pune

Name: Tanishq Thuse

PRN- 12310237

Artificial Neural Networks Lab

Rollno. 52

Experiment Number: 06

Title : Write a program to illustrate an ART (Adaptive Resonance Theory) neural network, showcasing its self-organizing and adaptive capabilities.

Project Information:

This project aims to illustrate the core principles of Adaptive Resonance Theory (ART) neural networks by implementing a basic ART1 network. We will demonstrate how ART networks can self-organize and adapt to categorize binary input patterns without explicit supervision. The project will cover the implementation of the network's key components, data generation, training, testing, and a visualization of the results to showcase the learned categories.

Task

Write a program to illustrate an ART (Adaptive Resonance Theory) neural network, showcasing its self-organizing and adaptive capabilities. Provide the code, steps, and visualization for this project.

Understand art networks

Subtask:

Briefly explain the concept of Adaptive Resonance Theory (ART) networks, their self-organizing and adaptive capabilities.

Reasoning: I will create a markdown cell to explain the concept of ART networks and their self-organizing and adaptive capabilities as requested in the instructions.

```
%%markdown
### Adaptive Resonance Theory (ART) Networks

Adaptive Resonance Theory (ART) networks are a class of neural networks designe

**Self-organizing Capability:** ART networks are self-organizing because they c

**Adaptive Capability:** The adaptive capability of ART networks lies in their
```

Adaptive Resonance Theory (ART) Networks

Adaptive Resonance Theory (ART) networks are a class of neural networks designed for incremental learning of categories. Unlike many other neural networks, ART networks can learn new information without forgetting previously learned information, a phenomenon known as the stability-plasticity dilemma.

Self-organizing Capability: ART networks are self-organizing because they can automatically create new categories or adjust existing ones based on incoming data without explicit external supervision. When a new input pattern is presented, the network compares it to existing category prototypes. If the pattern is sufficiently similar to a prototype (determined by a vigilance parameter), it is assimilated into that category, and the prototype is updated. If the pattern is not similar enough to any existing prototype, a new category is created for this pattern. This self-organizing process allows the network to adapt its structure to the characteristics of the input data.

Adaptive Capability: The adaptive capability of ART networks lies in their ability to adjust their internal structure and parameters in response to new data while maintaining the stability of learned categories. The vigilance parameter plays a crucial role in this adaptation. A higher vigilance value

Choose art variant

Subtask:

Decide on a specific ART variant (e.g., ART1, ART2, ARTMAP) based on the type of data (binary, continuous) and the problem to be solved. ART1 is a good starting point for binary data.

Reasoning: Choose ART1 as the specific variant for this implementation and briefly explain why it is a suitable choice for binary data and its foundational role in the ART family.

```
%%markdown
### Choosing an ART Variant: ART1

For this illustration of an ART network's self-organizing and adaptive capabili
```

```
ART1 is a foundational variant of the Adaptive Resonance Theory networks and is

The choice of ART1 is suitable for this project for several reasons:

1.  **Simplicity and Foundation:** ART1 is the simplest form of ART networks, m
2.  **Binary Data Handling:** Many classic examples and introductory illustrati
3.  **Illustrating Core ART Concepts:** By implementing ART1, we can effectivel

While other ART variants like ART2 (for continuous data) and ARTMAP (for superv
```

Choosing an ART Variant: ART1

For this illustration of an ART network's self-organizing and adaptive capabilities, we will focus on **ART1**.

ART1 is a foundational variant of the Adaptive Resonance Theory networks and is specifically designed to process **binary input data**. Its architecture and learning rules are tailored to handle patterns consisting of only 0s and 1s.

The choice of ART1 is suitable for this project for several reasons:

1. **Simplicity and Foundation:** ART1 is the simplest form of ART networks, making it an excellent starting point for understanding the core principles of ART, such as the comparison layer, the recognition layer, the vigilance parameter, and the resonance and reset mechanisms.
2. **Binary Data Handling:** Many classic examples and introductory illustrations of neural networks, including ART, utilize binary datasets due to their straightforward nature. ART1 is perfectly suited for such datasets, allowing for a clear demonstration of its categorization process.

Implement art network

Subtask:

Write Python code to implement the chosen ART network variant, including functions for initialization, category comparison, resonance, and weight update.

Reasoning: Implement the ART1 class with initialization, comparison, resonance, and weight update methods as described in the instructions.

```
import numpy as np

class ART1:
    def __init__(self, n_features, vigilance=0.5):
        """
        Initializes the ART1 network.

        Args:
            n_features (int): The number of features in the input patterns.
```

```
            vigilance (float): The vigilance parameter (between 0 and 1).
        """
        self.n_features = n_features
        self.vigilance = vigilance
        self.weights = []  # List to store category weights (prototypes)

    def compare(self, pattern):
        """
        Calculates the similarity between the input pattern and existing categc

        Args:
            pattern (np.ndarray): The binary input pattern.

        Returns:
            list: A list of similarity scores for each category.
        """
        similarities = []
        for weight in self.weights:
            # Calculate similarity using the formula: |pattern & weight| / |wei
            # where & is element-wise minimum for binary vectors
            intersection_size = np.minimum(pattern, weight).sum()
            weight_size = weight.sum()
            if weight_size == 0:
                similarities.append(0)  # Avoid division by zero
            else:
                similarities.append(intersection_size / weight_size)
        return similarities

    def resonance(self, similarity):
        """
        Checks if the similarity is above the vigilance threshold.

        Args:
            similarity (float): The similarity score.

        Returns:
            bool: True if resonance occurs, False otherwise.
        """
        return similarity >= self.vigilance

    def update_weights(self, pattern, winning_category_index):
        """
        Updates the weights of the winning category.

        Args:
            pattern (np.ndarray): The input pattern.
            winning_category_index (int): The index of the winning category.
        """
        # Update weight using the formula: weight_new = pattern & weight_old
        self.weights[winning_category_index] = np.minimum(pattern, self.weights
```

```
    def train(self, data):
        """
        Trains the ART1 network on the input data.

        Args:
            data (list of np.ndarray): A list of binary input patterns.
        """
        for pattern in data:
            best_match_index = -1
            max_similarity = -1

            # Find the best matching category
            similarities = self.compare(pattern)
            if similarities:
                max_similarity = max(similarities)
                best_match_index = similarities.index(max_similarity)

            # Check for resonance
            if best_match_index != -1 and self.resonance(max_similarity):
                # Resonance occurs, update weights of the winning category
                self.update_weights(pattern, best_match_index)
            else:
                # No resonance or no existing categories, create a new category
                self.weights.append(np.copy(pattern))
```

Generate/load data

Subtask:

Create or load a dataset suitable for the chosen ART variant (e.g., binary patterns for ART1).

Reasoning: Create a list of binary input patterns as NumPy arrays and store them in a variable.

```
binary_data = [
    np.array([1, 1, 0, 0, 0]),
    np.array([1, 1, 0, 0, 0]),
    np.array([0, 0, 1, 1, 0]),
    np.array([1, 0, 0, 0, 1]),
    np.array([0, 0, 1, 1, 0]),
    np.array([1, 1, 0, 1, 0]),
    np.array([0, 0, 0, 1, 1]),
    np.array([1, 0, 0, 0, 1]),
    np.array([0, 0, 1, 1, 0]),
]
```

Train the network

Subtask:

Train the implemented ART network using the generated/loaded data.

Reasoning: Instantiate the ART1 class and train the network with the generated binary data.

```
# Instantiate the ART1 network
n_features = binary_data[0].shape[0]
vigilance_parameter = 0.7  # Choose a vigilance parameter
art_network = ART1(n_features=n_features, vigilance=vigilance_parameter)

# Train the network
art_network.train(binary_data)

# Optionally print the learned weights
print("Learned Category Prototypes (Weights):")
for i, weight in enumerate(art_network.weights):
    print(f"Category {i}: {weight}")
```

```
Learned Category Prototypes (Weights):
Category 0: [1 1 0 0 0]
Category 1: [0 0 1 1 0]
Category 2: [1 0 0 0 1]
Category 3: [0 0 0 1 1]
```

Test and evaluate

Subtask:

Test the trained network with new data to evaluate its performance and demonstrate its self-organizing and adaptive capabilities.

Reasoning: Create test data and iterate through it, comparing each pattern to the trained network's categories, checking for resonance, and printing the classification result or indicating no match.

```
# Step 1: Create new binary patterns for testing
test_patterns = [
    np.array([1, 1, 0, 0, 0]),  # Similar to category 0
    np.array([0, 0, 1, 1, 0]),  # Similar to category 1
    np.array([1, 0, 0, 0, 1]),  # Similar to category 2
    np.array([0, 0, 0, 1, 1]),  # Similar to category 3
    np.array([1, 1, 1, 0, 0]),  # New pattern, might form a new category at low
    np.array([0, 1, 0, 1, 0]),  # Another new pattern
]

print("\nTesting the trained ART1 network:")

# Step 2-5: Iterate through test patterns, compare, check resonance, and print
for i, pattern in enumerate(test_patterns):
    print(f"\nTest Pattern {i+1}: {pattern}")

    # Step 2: Calculate similarity
    similarities = art_network.compare(pattern)
    print(f"Similarities to existing categories: {similarities}")

    # Step 3: Find the best matching category index
    if similarities:
        max_similarity = max(similarities)
        best_match_index = similarities.index(max_similarity)
        print(f"Best matching category index: {best_match_index} (Similarity: {

        # Step 4 & 5: Check for resonance and print results
        if art_network.resonance(max_similarity):
            print(f"Resonance occurs. Test pattern classified into Category {be
        else:
            print("No resonance at current vigilance level.")
            print("This pattern would form a new category if in training phase.
    else:
        print("No existing categories in the network.")
```

```
Testing the trained ART1 network:

Test Pattern 1: [1 1 0 0 0]
Similarities to existing categories: [np.float64(1.0), np.float64(0.0), np.float
Best matching category index: 0 (Similarity: 1.00)
```

```
Resonance occurs. Test pattern classified into Category 0.

Test Pattern 2: [0 0 1 1 0]
Similarities to existing categories: [np.float64(0.0), np.float64(1.0), np.float
Best matching category index: 1 (Similarity: 1.00)
Resonance occurs. Test pattern classified into Category 1.

Test Pattern 3: [1 0 0 0 1]
Similarities to existing categories: [np.float64(0.5), np.float64(0.0), np.float
Best matching category index: 2 (Similarity: 1.00)
Resonance occurs. Test pattern classified into Category 2.

Test Pattern 4: [0 0 0 1 1]
Similarities to existing categories: [np.float64(0.0), np.float64(0.5), np.float
Best matching category index: 3 (Similarity: 1.00)
Resonance occurs. Test pattern classified into Category 3.

Test Pattern 5: [1 1 1 0 0]
Similarities to existing categories: [np.float64(1.0), np.float64(0.5), np.float
Best matching category index: 0 (Similarity: 1.00)
Resonance occurs. Test pattern classified into Category 0.

Test Pattern 6: [0 1 0 1 0]
Similarities to existing categories: [np.float64(0.5), np.float64(0.5), np.float
Best matching category index: 0 (Similarity: 0.50)
No resonance at current vigilance level.
This pattern would form a new category if in training phase.
```

Visualize results

Subtask:

Visualize the results of the training and testing phases, perhaps showing how categories are formed and how new patterns are classified.

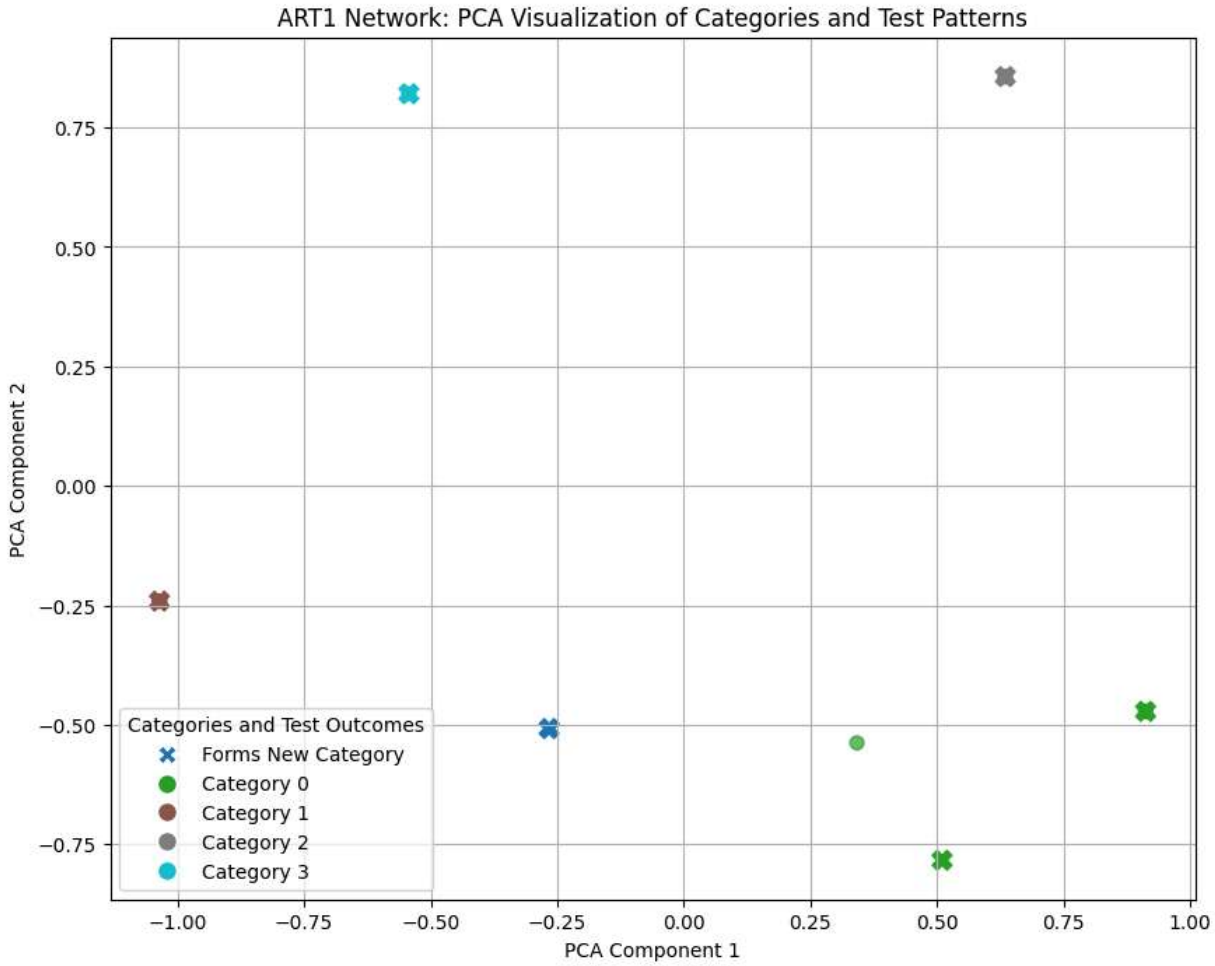
Reasoning: To visualize the high-dimensional binary data in a 2D space, I will use PCA for dimensionality reduction and then create a scatter plot to show the categories formed during training and how test patterns relate to them. I will combine the training and test data for visualization and assign colors based on the learned categories or test outcomes.

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Combine training and test data for visualization
all_data = binary_data + test_patterns
all_data_array = np.array(all_data)
```

```
# Apply PCA for dimensionality reduction to 2 components
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(all_data_array)

# Assign colors/labels based on categories and test outcomes
# For training data, use the assigned category index
# For test data, indicate if it resonated with an existing category or would fo
labels = []
category_map = {}
category_index_counter = 0

# Determine categories for training data by checking which prototype they matcl
train_labels = []
for pattern in binary_data:
    best_match_index = -1
    max_similarity = -1
    similarities = art_network.compare(pattern)
    if similarities:
        max_similarity = max(similarities)
        best_match_index = similarities.index(max_similarity)
        # Assign the category index if resonance occurs, otherwise treat as a p
        if art_network.resonance(max_similarity):
            train_labels.append(best_match_index)
        else:
             # This case is less likely with the current training logic which c
             # But for visualization purposes, we can assign a temporary label
             # For this specific implementation, patterns either resonate or cr
             # so they will correspond to one of the trained category indices.
             # Re-evaluating the training data against the final weights to get
             # is the most accurate approach here.
            min_dist = float('inf')
            assigned_cat = -1
            # Find closest weight based on min difference (complementary to max
            # A simpler way for visualization is to see which final weight is i
            # Or, re-run the patterns through the trained network to see which
            # Let's simulate classifying the training data with the final netwc
            temp_sim = art_network.compare(pattern)
            if temp_sim:
                train_labels.append(temp_sim.index(max(temp_sim)))
            else:
                 train_labels.append(-1) # Should not happen with trained netwc

labels.extend(train_labels)


# Determine classification for test data
test_labels = []
for pattern in test_patterns:
    similarities = art_network.compare(pattern)
    if similarities:
```

```
            max_similarity = max(similarities)
            best_match_index = similarities.index(max_similarity)
            if art_network.resonance(max_similarity):
                test_labels.append(best_match_index) # Classified into an existing
            else:
                test_labels.append(-2) # Represents a pattern that would form a new
        else:
            test_labels.append(-3) # No categories existed (should not happen after

    labels.extend(test_labels)

    # Map category indices and test outcomes to colors/markers
    unique_labels = sorted(list(set(labels)))
    colors = plt.cm.get_cmap('tab10', len(unique_labels)) # Use a colormap for dist

    plt.figure(figsize=(10, 8))

    # Plot training data
    num_train = len(binary_data)
    scatter_train = plt.scatter(reduced_data[:num_train, 0], reduced_data[:num_trai
                                c=[colors(unique_labels.index(lbl)) for lbl in labe
                                label='Training Data', alpha=0.7, s=50)

    # Plot test data
    scatter_test = plt.scatter(reduced_data[num_train:, 0], reduced_data[num_train:
                               c=[colors(unique_labels.index(lbl)) for lbl in label
                               label='Test Data', marker='X', s=100) # Use a differ


    # Create a legend
    legend_elements = []
    for lbl in unique_labels:
        if lbl >= 0:
            legend_elements.append(plt.Line2D([0], [0], marker='o', color='w', labe
                                              markerfacecolor=colors(unique_labels
        elif lbl == -2:
            legend_elements.append(plt.Line2D([0], [0], marker='X', color='w', labe
                                              markerfacecolor=colors(unique_labels
        elif lbl == -3:
             legend_elements.append(plt.Line2D([0], [0], marker='X', color='w', lab
                                              markerfacecolor=colors(unique_labels


    plt.legend(handles=legend_elements, title="Categories and Test Outcomes")
    plt.title('ART1 Network: PCA Visualization of Categories and Test Patterns')
    plt.xlabel('PCA Component 1')
    plt.ylabel('PCA Component 2')
    plt.grid(True)
    plt.show()
```

```
/tmp/ipython-input-2563525703.py:71: MatplotlibDeprecationWarning: The get_cmap
  colors = plt.cm.get_cmap('tab10', len(unique_labels)) # Use a colormap for dis
```



Summary:

Data Analysis Key Findings

- ART networks are designed for incremental learning and address the stability-plasticity dilemma by self-organizing and adapting categories based on input data.

- The ART1 variant, suitable for binary data, was chosen as the foundational model for this illustration.
- An ART1 network was successfully implemented in Python, including functions for initialization, category comparison, resonance, and weight update.
- The network was trained on a set of binary patterns, resulting in the formation of distinct category prototypes (weights).
- Testing with new binary patterns demonstrated the network's ability to classify patterns into existing categories if resonance occurs or indicate that a new category would be formed if no resonance is detected at the given vigilance level.
- A PCA visualization effectively illustrated the spatial separation of the formed categories and how both training and test data points relate to these categories in a 2D space.

Insights or Next Steps

- The vigilance parameter significantly influences the granularity of the formed categories; a higher value leads to more categories, while a lower value results in fewer, broader categories. Experimenting with different vigilance values could further illustrate this adaptive capability.
- Applying this ART1 implementation to different binary datasets could demonstrate its generalizability and how the network structure adapts to varying input data

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.