Experiments Regarding Fusion Adaptive Resonance Theory Software

Patrick Tjahjadi

13 June 2020

Table of Contents

Abstract

Machine learning has been used in a variety of purposes, such as classification, clustering, and regression, using both supervised and unsupervised learning methods. The problem that arises within machine learning is the necessity to acquire new knowledge while retaining previously learnt information. This paper introduces an elegant application of machine learning that uses multiple learning methods which aims to balance old and new information called Fusion Adaptive Resonance Theory (Fusion ART). In addition, several experiments using code to simulate Fusion ART models are conducted to illustrate the process involved and the viability of the models. These experiments include basic, abstract experiments and a concrete experiment taken from a dataset. One objective of this experiment is to develop a model that can store and retrieve data using machine learning principles. Future improvements can then be suggested after experimenting through the code.

Acknowledgements

I would like to express my appreciation to Prof. Tan Ah Hwee of the Singapore Management University, who introduced me to Adaptive Resonance Theory and its generalisation to fusion models through various research articles, and who encouraged me to write this paper.

In addition, I would like to thank Dr. Budhitama Subagdja of Nanyang Technological University for creating the API to develop the Fusion ART models used in this paper. I also would like to thank him for answering my questions regarding his API and Adaptive Resonance Theory in general.

This paper would not have been possible without their guidance.

1. Introduction

Adaptive Resonance Theory (ART), a theory regarding cognitive information processing and an application of machine learning, is prevalent across technology, biology, and psychology. Many machine learning algorithms experience a problem where they forget past information while learning new information. ART arises from the need to balance acquiring new information and retaining previously acquired ones. This problem is called the stability-plasticity dilemma, in which ART aims to alleviate (Grossberg, 2013). Particularly, ART concerns a type of neural network that uses unsupervised machine learning methods.

There are many variations of ART models, including ART 1 that accepts binary inputs, ART 2 that accepts continuous inputs and Fuzzy ART that implements fuzzy logic (Carpenter & Grossberg, 2002). Fusion ART is a generalisation of ART architecture that uses multiple pattern channels instead of one (Nguyen, Woon & Tan, 2008). Unlike ART, Fusion ART falls within several machine learning paradigms, such as supervised learning, unsupervised learning, reinforcement learning, etc. that learns in an online and incremental manner (Tan, Subagdja, Wang & Meng, 2019).

The search cycle of ART concerns an input layer of nodes *F1* that accepts an input from a pattern. The *F1* layer then transfers the input to the recognition layer of nodes *F2* for comparison. This comparison is in the form of a hypothesis test. In this report, the aforementioned hypothesis test is called the vigilance test: for each node *J* in F2, a level of matching against the input is generated and if the match is close enough to the input pattern, the node *J* will learn from the mentioned input pattern (Carpenter & Grossberg, 2002). The vigilance test compares the level of matching against a defined vigilance parameter *rho*. The test succeeds if the level of matching is higher than *rho*. As such, as *rho* increases, the nodes tend to produce more detailed information. If the vigilance test fails, the process then iterates through all nodes in *F2* in sequential order until a successful test is found. If there are no successful tests, the uncommitted node in *F2* becomes committed, and a new uncommitted node is added which grows the network (Tan, Subagdja, Wang & Meng, 2019). The parameters of the Fusion ART model will be discussed in more detail in Chapter 3.

This report aims to present the findings from experiments regarding Fusion ART software, created by Dr. Budhitama Subagdja. The main goal of this software is to select the node in *F2* that most closely resembles a defined input pattern through a process called resonance search. Then, the effectiveness of Fusion ART models can be tested by retrieving data stored in the model by means of a query. Chapter 3 explains more details of the process involved.

2. Software and Dataset Materials

The software concerns two types of Fusion ART models in Python:

1. Basic: This software develops a Fusion ART model that accepts plain numerical vectors as input. Here, the number of *F1* fields and the length of each field is specified. The F1 layer would then all be initialised to 0s. The code is named “basic-FusionART-example”.

2. Schema-based: This software develops a Fusion ART model that defines and specify fields with respect to a Python dictionary with attributes and values. The schema-based model allows each node in the *F1* layer to have an attribute name. The code is named “schemabased-FusionART-example”.

The functions in which the basic and schema-based Fusion ART models are developed can be found within the API “fusionART.py”. These include the processes to create the model, such as code competition, resonance search and top-down *F1* search. The code “fusionART.py” also takes the functions in “ARTfunc.py”. Whereas “fusionART.py” contains functions to develop the Fusion ART models, “ARTfunc.py” provides the computational methods necessary for the functions in “fusionART.py” to work.

In order to test the software, example cases have been provided in “fartcase.py” and “fartcaseSchema.py” which corresponds to cases for basic and schema-based Fusion ART models, respectively. Test cases can be further developed by modifying the model parameters and input patterns.

The application of the Fusion ART model can then be further realised by providing a real-world example taken from a dataset. For this experiment, the dataset concerns the names of 83 professors, along with their group, universities, and research interests. The dataset is named “SCSE ProfProfile”.

3. Experimental Analysis

3.1. Structure and Parameters

With respect to the code, the Fusion ART model concerns several underlying structures:

- Input layer *F1*: Let $I = 0, \ldots, numspace–1$ denote the input vector of size *numspace* with each vector of size defined in the vector *lengths*, with each element of $I \in [0, 1]$. In the basic model, the value *numspace* can be defined. The schema-based model depends on the number of values in *attrib*.

- Competitive layer *F2*: Let $J = 0, \ldots, N–1$ denote the activity vector of size *N*. Initially, *F2* only contains 1 uncommitted node for learning. In itself, the value F2 of a particular node in the code denotes its level of matching from bottom-up activation, which is part of the resonance search process. The uncommitted node of an *F2* layer has an F2 value of 0.

- Weight vector: The weight vector, shown in the code as 'weights', corresponds to the weight in which the J^{th} node in *F2* learns the input patterns in *F1* for all *I*. Hence, the weight vector has the same dimensions as the *F1* layer. Initially, all elements of the weight vector are set to 1.

- Complement-coding: Specific to the schema-based model, complement-coding augments the input vector *I*, with a complement vector I^c such that $I^c = 1 – I$ (Nguyen, Woon & Tan, 2008). As a result, they are both presented together in both the input and weight vectors of the model. This is defined as a Boolean-value "*compl*" in the code. Upon setting *compl* to True, a vector *vcompl* is generated that is the complement of the vector *val*.

- Parameters: The Fusion ART code contains several parameters that determine the behaviour of the model, such as how strict should the model accept patterns from inputs. For all *I*, there exists choice parameters *alpha* > 0, learning rate parameters *beta* $\in [0, 1]$, contribution parameters *gamma* $\in [0, 1]$ and vigilance parameters *rho* $\in [0, 1]$ (Tan, Carpenter & Grossberg, 2007). The parameter *numspace* defines the number of nodes in F1, with *lengths* defining the length of each node.

The ART architecture also introduces a reset mechanism, that checks the similarity between the weight vector and the input vector. The reset mechanism takes effect during the vigilance test and decides whether a node in *F2* learns the input pattern. A reset occurs when the level of matching of a node in *F2* is smaller than the defined vigilance parameter. The reset mechanism would then call the next sequential node in the *F2* layer for the vigilance test (Ignizio Burke, 1991).

3.2. Experimentation and Discussion

The experiment involves running the code to develop a Fusion ART architecture for both basic and schema-based models. The first input pattern will be used to train the model. Training is done to train and fit the Fusion ART model, which will learn from the input. Some of the sample cases found in “fartcase.py” and “fartcaseSchema.py” will be used for testing. This will be done to evaluate the model’s fit and performance. Moreover, further testing includes modifying the model parameters and developing new inputs.

As a background, the code aims to select a node in the *F2* layer that has a high level of matching against the input. Each node in *F2* is assigned a matching value and the node’s index that has the highest matching value will be selected for learning. This process is called resonance search and is defined as the function resSearch in the Python code.

The observation is in the form of a table that measures the input and output of the Fusion ART model. A slight modification of the code includes measuring the execution time for the resonance search process using Python’s *timeit* library. It is to note that the execution time depends on many factors such as hardware specifications. Graphs are modelled using Python’s *matplotlib* library and measure the average selected node’s F2 value and layer size for all test patterns.

The way the code is run is by initialising the *F1* layer and parameters. Then, the model is trained by running through to the training pattern. Finally, the model is tested by running through one test pattern. For each test pattern, the model is rebuilt and trained by the same training pattern.

3.2.1 Basic Fusion ART Experimentation

Suppose we initialise an *F1* layer with the following parameters:

numspace	*lengths*	*alpha*	*beta*	*gamma*	*rho*
3	[4, 4, 2]	[0.1, 0.1, 0.1]	[1, 1, 1]	[1, 1, 1]	[0.2, 0.2, 0.5]

Table 1: Parameters and structure of the *F1* layer of the first basic model.

The experiments involve several input patterns. Since the node status may change to *committed* after another iteration, each input pattern is undergone resonance search more than once. Node and F2 represent the node’s index and level of matching, respectively. Size denotes the number of nodes in the *F2* layer. Node Status determines if there exists an uncommitted node.

Input Pattern	Node	F2	Time (ms)	Size	Node Status
[1, 0, 0, 1], [0, 0, 0, 1], [0.6, 0.4]	0	1.2079	1.1157	2	Uncommitted
	0	2.77056	1.7842	2	Committed

Table 2: Training results of the first basic model

Input Pattern	Node	F2	Time (ms)	Size	Node Status
[0, 1, 1, 0], [1, 0, 0, 0], [0.3, 0.7]	1	1.2079	1.3398	3	Uncommitted
	1	2.77056	1.5769	3	Committed
[1, 0, 0, 1], [0, 0, 0, 0], [0.5, 0.5]	0	1.7706	1.475	2	Committed
	0	1.8524	1.35	2	Committed
[0, 1, 1, 0], [0.5, 0.5, 0.5, 0.5], [0.5, 0.5]	1	1.4518	1.275	3	Uncommitted
	1	2.81385	1.2766	3	Committed
[1, 1, 1, 1], [1, 1, 1, 1], [1, 1]	1	2.9036	1.1791	3	Uncommitted
	1	2.9036	1.2209	4	Uncommitted
	1	2.9036	1.6369	5	Uncommitted

Table 3: Test results of the first basic model.

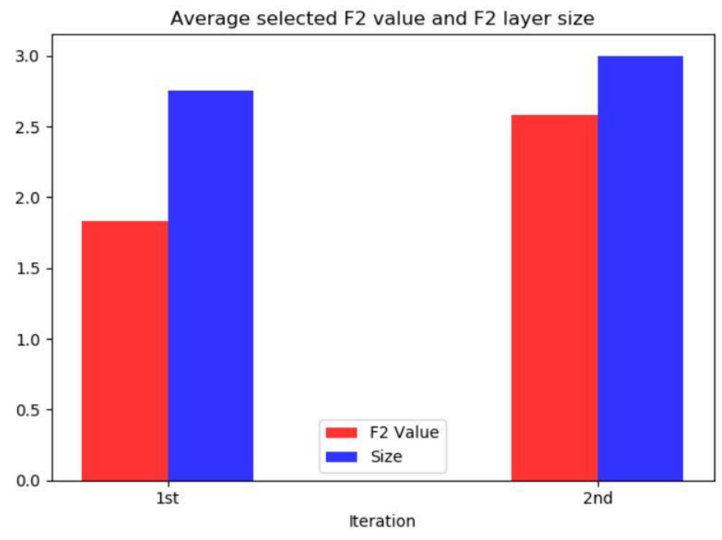

Figure 1: Bar chart measuring the first basic model

Usually, performing resonance search twice would suffice in finding the *F2* node used for learning. Testing shows that after two iterations, model convergence would occur and there would be no change in output. The matching value for each node also increases after each iteration but would remain unchanged after convergence.

Testing the extremities of the input pattern would yield peculiar results. For instance, having all elements of the input pattern to be 1 would always grow the *F2* layer with each iteration. The F2 value for all the nodes sans the uncommitted node would always remain the same, presumably because the nodes have reached the highest possible matching value. Consequently, there will be an uncommitted node generated with each iteration ad infinitum.

Suppose we modify the parameters further. In particular, the vigilance parameter *rho* is reduced. We would expect to see less detailed knowledge learnt by each *F2* node. Let the *F1* layer be initialised along with the following parameters:

numspace	*lengths*	*alpha*	*beta*	*gamma*	*rho*
3	[3, 2, 2]	[0.2, 0.2, 0.2]	[0.5, 0.5, 0.5]	[0.5, 0.4, 0.6]	[0.2, 0.1, 0.2]

Table 4: Parameters and structure of the *F1* layer of the second basic model.

Again, the first input pattern is used as training. The result would then be as follows:

Input Pattern	Node	F2	Time (ms)	Size	Node Status
[1, 0, 1], [0, 1], [0.6, 0.4]	0	0.767	1.1833	2	Uncommitted
	0	1.2879	1.1235	2	Committed

Table 5: Training results of the second basic model.

Input Pattern	Node	F2	Time (ms)	Size	Node Status
[0, 1, 0], [1, 0], [0.4, 0.6]	1	0.6108	1.642	3	Uncommitted
	1	1.25	1.4619	3	Committed
[0.3, 0.7, 0.2], [0, 0], [0.5, 0.5]	0	0.5636	1.4296	2	Committed
	0	0.642	1.294	2	Committed
[0.5, 0.5, 0.5], [0.5, 0.5], [0.5, 0.5]	0	0.844	1.4742	2	Committed
	0	0.974	0.866	2	Committed
[0, 0, 0], [0, 0], [0, 0]	0	0	2.0788	2	Committed
	0	0	1.857	2	Committed
	0	0	1.4257	2	Committed

Table 6: Results from the second basic model.

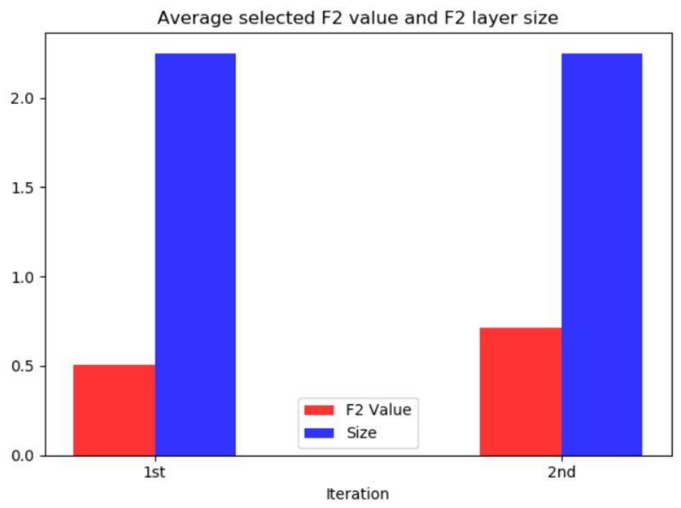

Figure 2: Bar chart measuring the second basic model.

Several experiments suggest that a lower value of the contribution parameter *gamma* and a higher value of the choice parameter *alpha* would decrease the F2 value of nodes in the *F2* layer.

A trivial observation is, contrary to the input vector of all 1s, an input vector of all 0s would not generate anything productive. That is, there will never be a new uncommitted node, and the F2 value would always remain zero.

3.2.2 Schema-based Fusion ART Experimentation

As discussed, the *F1* layer of the schema-based Fusion ART model comprises of a Python dictionary containing attributes and values. Suppose we initialise a model with the following parameters:

alpha	beta	gamma	rho
[0.1, 0.1, 0.1]	[1, 1, 1]	[1, 1, 1]	[0.2, 0.2, 0.5]

Table 7: Parameters of the first schema-based model

Moreover, suppose the *F1* layer comprises of the following schema:

```
[{'name': 'state',
  'compl': True,
  'attrib': ['s1', 's2'],
  'val': [0.0, 0.0],
  'vcompl': [0.0, 0.0]},
 {'name': 'action',
  'attrib': ['a1', 'a2', 'a3', 'a4'],
  'compl': False,
  'val': [0.0, 0.0, 0.0, 0.0]},
 {'name': 'Q', 'compl': True, 'attrib': ['q'], 'val': [0.0], 'vcompl':
[0.0]}]
```

Figure 3: *F1* schema of the first schema-based model.

Note that the *compl* attribute is set to False by default unless coded otherwise. The attributes *val* and *vcompl* are automatically included from the expandInputwSchema() function from "fusionART.py", with *vcompl* present only if *compl* is True. Hence, the only defined fields are *name* and *attrib*.

The first input schema will be used to train the model, while the subsequent input schemas will be used for testing.

Similar to the basic Fusion ART model, the outputs of the schema-based model are as follows, through the following input schemas:

name	*compl*	*val*
state	False	[1, 0]
action	False	[0, 0, 0, 1]
Q	False	[0.6]

Table 8: Training input schema of the first schema-based model

Iteration	Node	F2	Time (ms)	Size	Node Status
1	0	1.2079	1.4355	2	Uncommitted
2	0	2.77056	1.4395	2	Committed
3	0	2.77056	1.2471	2	Committed

Table 9: Results from the training input schema of the first schema-based model

name	*compl*	*val*
state	False	[0, 1]
action	False	[1, 0, 0, 0]
Q	False	[0.3]

Table 10: First test schema of the first schema-based model

Iteration	Node	F2	Time (ms)	Size	Node Status
1	1	1.2079	1.7127	3	Uncommitted
2	1	2.77056	1.6498	3	Committed
3	1	2.77056	1.1234	3	Committed

Table 11: Results from the first test schema of the first schema-based model

name	*compl*	*val*
state	True	[0, 1]
action	False	[1, 0, 0, 0]
Q	True	[0.3]

Table 12: Second test schema of the first schema-based model

Iteration	Node	F2	Time (ms)	Size	Node Status
1	1	1.2079	1.3688	3	Uncommitted
2	1	2.77056	1.439	3	Committed
3	1	2.77056	1.2372	3	Committed

Table 13: Results from the second test schema of the first schema-based model

name	*compl*	*val*
state	False	[1, 1]
action	False	[1, 1, 1, 1]
Q	False	[1]

Table 14: Third test schema of the first schema-based model

Iteration	Node	F2	Time (ms)	Size	Node Status
1	1	2.41954	1.905	3	Uncommitted
2	1	2.85244	2.846	3	Committed
3	3	2.85244	2.2998	3	Committed

Table 15: Results from the third test schema of the first schema-based model

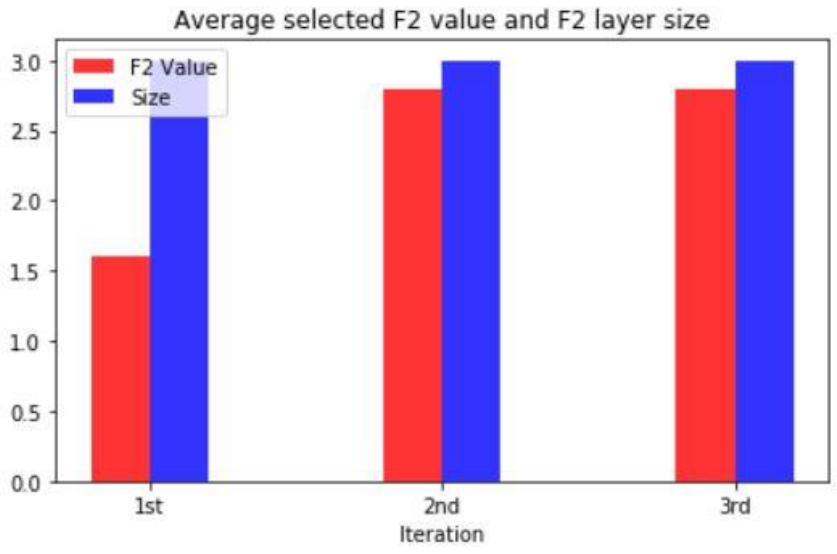

Figure 4: Bar chart measuring the first schema-based model.

Again, it is expected that the schema-based Fusion ART model behaves similarly to the basic ART model. The F2 value of each node increases through each iteration and converges after several iterations. Figure 4 suggests that two iterations are sufficient to achieve the highest F2 value. The uncommitted *F2* node which has an F2 value of 0 will become committed in the next iteration.

A peculiar observation is that maximising the *val* elements to 1 would also generate a very high F2 value for the selected *F2* node used for learning. This is also similar to having an input pattern of all 1s in the basic Fusion ART model. It does not, however, expand the *F2* layer indefinitely by generating more nodes.

Modifying the *compl* value does not seem to affect the results of the model. That is, regardless of whether *compl* is set to True or False, the selected node, its F2 value and the size of the *F2* layer remains unchanged.

Next, suppose we initialise a model with the following parameters:

alpha	beta	gamma	rho
[0.3, 0.3, 0.3]	[1, 1, 1]	[0.6, 0.4, 0.1]	[0.1, 0.1, 0.3]

Table 16: Parameters of the second schema-based model

Moreover, suppose the *F1* layer comprises of the following schema:

```
[{'name': 'state',
  'attrib': ['s1', 's2', 's3'],
  'compl': False,
  'val': [0.0, 0.0, 0.0]},
 {'name': 'action',
  'compl': True,
  'attrib': ['a1', 'a2', 'a3'],
  'val': [0.0, 0.0, 0.0],
  'vcompl': [0.0, 0.0, 0.0]},
 {'name': 'Q', 'attrib': ['q1', 'q2'], 'compl': False, 'val': [0.0, 0.
0]}]
```

Figure 5: *F1* schema of the first schema-based model.

This would yield the following results:

name	*compl*	*val*
state	False	[1, 0, 1]
action	False	[0, 0, 1]
Q	False	[0.6, 0.4]

Table 17: Training input schema of the second schema-based model

Iteration	Node	F2	Time (ms)	Size	Node Status
1	0	0.5976	1.4029	2	Uncommitted
2	0	0.9623	1.7844	2	Committed
3	0	0.9623	1.037	2	Committed

Table 18: Results from the training input schema of the second schema-based model

name	*compl*	*val*
state	False	[0, 1, 0]
action	False	[1, 0, 0]
Q	False	[0.5, 0.5]

Table 19: First test schema of the second schema-based model

Iteration	Node	F2	Time (ms)	Size	Node Status
1	1	0.416	1.5569	3	Uncommitted
2	1	0.9021	2.1579	3	Committed
3	1	0.9021	1.513	3	Committed

Table 20: Results from the first test schema of the second schema-based model

name	*compl*	*val*
state	False	[0.1, 0.1, 0.1]
action	True	[0.2, 0.2, 0.2]
Q	False	[0.3, 0.3]

Table 21: Second test schema of the second schema-based model

Iteration	Node	F2	Time (ms)	Size	Node Status
1	0	0.31651	1.7396	2	Uncommitted
2	0	0.6495	1.3708	2	Committed
3	0	0.6495	1.6273	2	Committed

Table 22: Results from the second test schema of the second schema-based model

name	*compl*	*val*
state	False	[0, 0, 0]
action	False	[0, 0, 0]
Q	False	[0, 0]

Table 23: Third test schema of the second schema-based model

Iteration	Node	F2	Time (ms)	Size	Node Status
1	0	0.190476	1.5584	2	Uncommitted
2	0	0.363636	1.229	2	Committed
3	0	0.363636	1.1183	2	Committed

Table 24: Results from the third test schema of the second schema-based model

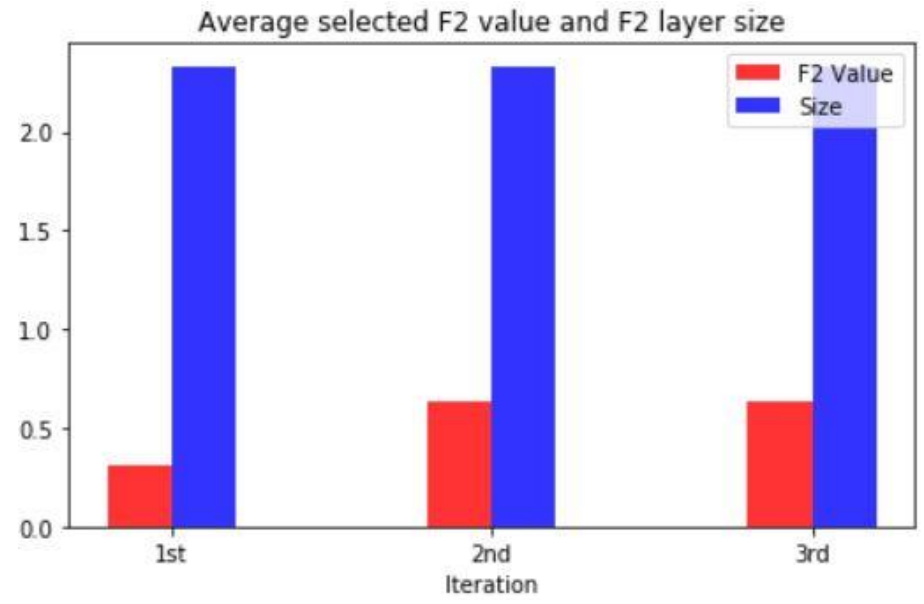

Figure 6: Bar chart measuring the second schema-based model.

The pattern shown in Figure 6 also resembles Figure 4 such that there is no distinct difference between the 2nd and 3rd iteration. Hence, iterating more than twice would not affect anything in these test cases.

As previously observed from the basic Fusion ART model, a lower value of the contribution parameter *gamma* and a higher value of the choice parameter *alpha* would decrease the F2 value of nodes in the *F2* layer. Moreover, lowering the values of *val* would also decrease the F2 value of the nodes, but not to the extent such that the F2 values would be 0 as shown in the second basic model.

On a side note, there also exists a top-down F1 operation in the code named *TopDownF1()* for schema-based Fusion ART models. This is done in order to update the *F1* field after the resonance search process. When a node in the *F2* layer learns from an input pattern, the corresponding schema is updated based on the pattern using *TopDownF1()*.

3.2.3 Model Experimentation for the Professor Dataset

We would like to assess the effectiveness of the Fusion ART model in storing and retrieving data. For this experiment, the dataset "SCSE ProfProfile" is used that contains 83 professors, with their group, universities, and research interests.

Before developing the Fusion ART model, several data pre-processing strategies are implemented:

1. Each attribute of the dataset (name, group, university and research interest) is converted to separate Python lists.
2. The dataset contains a semicolon ";" that separates values of universities and research interests. Since each professor can have multiple universities and research interests, the universities and research interests of each professor are split into lists.
3. Since the model accepts numerical values, the current data could not be input to the model. The data is transformed from categorical data to numerical data using one-hot encoding.
4. For each professor, their universities and research interests are transformed into one-hot arrays that can contain multiple one-hot values. For example, if a professor has three research interests out of ten possible research interests, their research interest vector may look like [0, 1, 0, 0, 1, 0, 0, 0, 1, 0].

Next, the schema for the Fusion ART model is defined as the list of unique names, groups, universities and research interests sorted alphabetically. This will ensure that the one-hot vectors associated with each attribute correspond to the values in the schema since the one-hot vectors are also sorted alphabetically.

name	*compl*	*attrib*
name	False	["AS Madhukumar", "Alexei Sourin", …]
Group	False	["Biomedical Informatics", "Computational Intelligence", …]
University	False	["Anna University", "Birla Institute of Technology", …]
Research Interests	True	["Ad Hoc and Mobile Networks", "Agent Oriented Software Engineering", …]

Table 25: Input schema of the professor model

The research interests attribute is complement-coded because we would like to use research interests as a query to retrieve professor data. More information regarding querying is explained below.

Suppose we initialise this model with the following parameters:

alpha	beta	gamma	rho
[0.1, 0.1, 0.1, 0.1]	[1, 1, 1, 1]	[0.25, 0.25, 0.25, 0.25]	[0.2, 0.2, 0.5, 0.5]

Table 26: Parameters of the professor model

Then, the model is updated by inserting each professor's full data, including their names, groups, universities, and research interests using a for loop. This is done by calling the function updateF1bySchema and inserting their respective one-hot lists. By performing resonance search for each node, the network can grow to 83 committed nodes, with each node containing each professor's full data along with one uncommitted node. This brings the network to a total size of 84 nodes.

Once the model has successfully stored the data, we would like to test the model whether it can retrieve data using an input query. This experiment concerns two queries:

1. Query by name: Given an input of a professor name, convert the name into the form of an input vector to be read by the model. Then, retrieve the full data of that professor, including their group, universities, and research interests.
2. Query by research keywords: Given an input of a list of research interests, convert the keywords into the form of an input vector to be read by the model. Then, retrieve the professor that best matches the list of research interests.

For query by name, the model's gamma parameters are set to [1, 0, 0, 0] to solely consider the professor's name. The model's schema is updated by inputting the *val* vector that corresponds to the professor's name. For instance, to find the full details of Zinovi Rabinovich, the name vector [0, 0, …, 0, 1] is called to updateF1bySchema. To launch the query, the function compChoice is called that calculates the F2 value of each node. The query algorithm would retrieve the node that has the highest F2 value since the weight vectors of that node best match the query. The data would then be extracted from that node. For each index that has a value of 1 in the node's weight vector, the data is retrieved by printing the values from the model's F1 field.

Input	Output

Alexei Sourin	`Name:`
	`Alexei Sourin`
	
	`Group:`
	`Graphics and Interactive Computing`
	
	`Universities:`
	`Moscow Engineering Physics Institute`
	
	`Research Interests:`
	`Computer Graphics`
	`Cybermedicine`
	`Scientific Visualization`
	`Shape Modelling`
	`Virtual Reality`
	`Visualization on the Grid`
	`Web Visualization`

Bo An	`Name:`
	`Bo An`
	
	`Group:`
	`Computational Intelligence`
	
	`Universities:`
	`Chongqing University`
	`University of Massachusetts Amherst`
	
	`Research Interests:`
	`Game Theory`
	`Intelligent  E-commerce`
	`Multi agent Systems`
	`Optimization`

Dusit Niyato	`Name:`
	`Dusit Niyato`
	
	`Group:`
	`Computer Networks and Communications`
	
	`Universities:`
	`King Mongkut's Institute of Technology Ladkrabang`
	`University of Manitoba`
	
	`Research Interests:`
	`Economics Modeling of Wireless Communications and Ne`
	`tworking`
	`Network Performance Analysis`
	`Radio Resource Management`

Lee Bu Sung Francis	Name: Lee Bu Sung Francis Group: Computer Networks and Communications Universities: Loughborough University Research Interests: Cloud Computing Grid Computing Quality of Service SDN in Computer Network Wireless Ad Hoc Network
Smitha Kavallur Pisharath Gopi	Name: Smitha Kavallur Pisharath Gopi Group: Hardware and Embedded Systems Universities: Anna University Calicut University Nanyang Technological University Research Interests: Brain Computer Interface Embedded Systems Low Power Reconfigurable Computing Signal Processing Applications
Zinovi Rabinovich	Name: Zinovi Rabinovich Group: Computational Intelligence Universities: Hebrew University of Jerusalem Research Interests: Artificial Intelligence Modelling of Manipulative Interaction Multi agent Systems
zinovi rabinovich	Professor not found!
Patrick Tjahjadi	Professor not found!

Table 27: Experiment results for querying by name.

The model successfully queried by name with 100% accuracy. It is to note, however, that the query is case-sensitive, and the input must match the name exactly as stored in the dataset.

To query by research keywords, a batch file is created that contains the list of professor names and their research keywords, separated by a tab. The batch file is named “research_query.bat”. For instance, the first five lines of the batch file are:

```
Alexei Sourin      Computer Graphics;Shape Modelling;Virtual
Reality;Web Visualization;Visualization on the
Grid;Cybermedicine;Scientific Visualization

Anupam Chattopadhyay               High level Synthesis;Application
specific Processors;Heterogeneous MPSoC;Synthesis for Emerging
Technology

Anwitaman Datta            Distributed Systems;Security and
Privacy;Data Analytics;Algorithms;Self organization

Arijit Khan           Graphs Querying and Mining;Databases;Data
Mining;Algorithms;Machine Learning

Arvind Easwaran              Real time Systems;Cyber physical
Systems;Formal Methods
```

Figure 7: Sample lines of the query by research keywords batch file

For each professor, their research keywords are used as input for the query. For this query, the model’s gamma parameters are set to [0, 0, 0, 1] to solely account for research interests. The research interests are converted into an input vector. Similar to querying by name, the compChoice function retrieves the node with the highest F2 value.

In this experiment, noise is introduced with varying probabilities from $N = \{0, 10, 20, 30, 40, 50\}$. With a probability of N%, each bit of the research keyword vector is toggled from 0 to 1 (a research keyword is added) or from 1 to 0 (a research keyword is removed) before being fed to the model.

The accuracy of the model is determined by the number of queries where the output of the query matches the name stated in the batch file. Therefore, it is expected that 0% noise would yield 100% retrieval accuracy. The query is labelled as correct if the correct professor’s name is output when multiple professors are returned, due to a tie in F2 values. Since the probability of bit toggling is random, the model’s accuracy differs for each attempt. Hence, the experiment concerns 4 attempts. For each attempt and level of noise, a text file is output that includes the research keywords and the retrieved professor name(s). The sample mean ($\bar{X}$) and standard deviation (s) is calculated for each level of noise.

$$\bar{X} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Equation 1: Formula for sample mean. x_i denotes the retrieval accuracy for each attempt and n signifies the number of attempts.

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{X})^2}{n - 1}}$$

Equation 2: Formula for sample standard deviation. x_i denotes the retrieval accuracy for each attempt and n signifies the number of attempts.

The experiment is separated into two trials. The first trial concerns querying by research keywords without complement-coding. This means that the *compl* value for Research Interests is set to False in Table 25. The results are as follows:

Noise	Retrieval Accuracy	$\bar{X}$	s
0%	100%	100%	0
10%	90.36%, 90.36%, 93.98%, 95.18%	92.47%	2.485
20%	63.86%, 51.81%, 59.04%, 51.81%	56.63%	5.903
30%	15.66%, 22.89%, 13.25%, 30.12%	20.48%	7.621
40%	4.82%, 6.02%, 4.82%, 7.23%	5.72%	1.153
50%	1.2%, 0%, 2.41%, 0%	0.9%	1.153

Table 28: Query by research keywords results without complement-coding.

The results as shown in Table 28 were peculiar. There is a sudden notable drop of accuracy when the noise level is set beyond 20%. There seems to be a high amount of variability in the accuracy for 20% and 30% noise, as shown with their high value of sample standard deviation.

The second trial uses complement-coding for research interests. By inserting both *val* and *vcompl* vectors to updateF1bySchema, the model can query using complement-coding. For this trial, each bit of both *val* and *vcompl* vectors have a probability of N% to flip.

By default, updateF1bySchema automatically sets *vcompl* to be the complement of *val*. By setting refresh = False in updateF1bySchema, the *vcompl* vector can be manually set, since the vector has undergone bit toggling. The results are as follows:

Noise	Retrieval Accuracy	$\bar{X}$	*s*
0%	100%	100%	0
10%	100%, 100%, 100%, 100%	100%	0
20%	96.39%, 95.18%, 97.59%, 92.77%	95.48	2.059
30%	59.04%, 60.24%, 65.06%, 61.45%	61.45	2.602
40%	22.89%, 15.66%, 28.92%, 20.48%	21.99	5.513
50%	4.82%, 1.2%, 4.82%, 2.41%	3.31	1.809

Table 29: Query by research keywords results with complement-coding.

Interestingly, the model performs better when research interests are complement-coded. The increase in accuracy can be attributed to the complement vector *vcompl*. Querying with complement-coding involves verifying whether the values in the *vcompl* vector is truly the complement of their *val* vector counterparts. Therefore, if there is an index in the *val* vector that has its bit toggled, that research keyword is ignored from the query if the same index in the *vcompl* vector does not undergo bit toggling because the indexes do not complement each other. This is referred to as the "don't-care" condition. This condition may attribute to the high sample standard deviation on higher levels of noise than the non-complement-coded experiment.

However, the "don't-care" condition has a notable weakness. The query can still include or remove noisy or actual research keywords, respectively if both indexes from the *val* and *vcompl* vectors undergo bit toggling. This weakness becomes increasingly noticeable for higher levels of noise, as there is a higher probability for both indexes to go through bit toggling.

4. Reflections

4.1. Initial Expectations and Limitations

After several considerations for the experiments in Chapter 3, several issues were expected that did not match initial expectations.

4.1.1 Basic and Schema-based Model Experiment

One expectation for the experiment is that the *F2* layer would continuously grow for each iteration. However, the experiments thus far suggest that there are no changes in the overall network and output beyond the second iteration, except when the input vector is all 1s. Several reasons may explain this anomaly.

Firstly, the cases may not be comprehensive enough to cover inputs and parameters in which the model may grow beyond 2 iterations. Further trial and error may be required to find models that grow with multiple iterations.

Another possibility is due to the limitation of the code or the techniques used. The resonance search process, and thus the main purpose of the code, is only to select a node in the *F2* layer that has the highest F2 value. As discussed in Chapter 3, the way the code is run in this experiment is, for each test pattern, build the Fusion ART model and train using the given input pattern. Then, each test pattern is run multiple times. This does not necessarily grow the network.

4.1.2 Professor Model Experiment

It is expected that the Fusion ART model used to store and retrieve professor data can be used for a wider variety of applications. The current research keywords query could retrieve data in a relatively short period of time, since $n = 83$, where n is the number of rows in the dataset. However, the query becomes computationally intensive for large values of n. For instance, there is already a noticeable processing time of 14.1 seconds for $n = 83$ for a noise level of 10%. Although the current model is applicable for other data, querying larger datasets would require more graphically intensive hardware.

Another expectation for this experiment is that the research keywords query would be able to retrieve the correct professor despite noises in the research keywords vector. In essence, the retrieval accuracy is lower than expected when varying levels of noise are introduced to the query. To alleviate this problem, implementing complement-coding to the research interests vector proved beneficial in increasing the retrieval accuracy.

4.2. Suggested Improvements

Several improvements can be made in this experiment. For instance, if we would like to test a particular input pattern, we may train the model using all patterns that were previously meant for testing as training patterns. This will yield different results. Suppose we build a Fusion ART model with parameters stated in Table 1, with the following training patterns in order:

1. [1, 0, 0, 1], [0, 0, 0, 1], [0.6, 0.4]
2. [0, 1, 1, 0], [1, 0, 0, 0], [0.3, 0.7]
3. [1, 0, 0, 1], [0, 0, 0, 0], [0.5, 0.5]

After training using the following patterns, suppose we wish to test the input pattern [0, 1, 1, 0], [0.5, 0.5, 0.5, 0.5], [0.5, 0.5]. As opposed to the result in Table 3, the result becomes:

Input Pattern	Node	F2	Time (ms)	Size	Node Status
[0, 1, 1, 0], [0.5, 0.5, 0.5, 0.5], [0.5, 0.5]	1	2.1342	1.808	3	Committed
	1	2.6746	1.5585	3	Committed
	1	2.6746	1.0554	3	Committed

Table 30: Result of testing using the first basic model, after training using 3 previous input patterns in order.

Interestingly, the F2 value of the selected node is high in the first iteration. However, the F2 value is not as high in the second iteration as compared to when training using only one input pattern (see Table 3).

All in all, there are different approaches that can be used to build the Fusion ART models. However, there are considerations to determine how much data should be used for training and testing. The aim is to create a Fusion ART model that generalises well from training data to test data, that is, how well can the patterns learnt by the model apply to test patterns that were not previously used for learning. Too much data used for training would cause the model to overfit and yield high accuracy solely on training data. In contrast, too little training data would cause the model to underfit, such that the model did not learn enough to predict accurately on all data (Ghasemian, Hosseinmardi & Clauset, 2019). Improvements can be done by finding the balance between training and test data.

With respect to the code, there are several suggestions for improvement. Dr. Subagdja mentioned that the current code does not show the F2 value consistently with the reset process during resonance search. The current code also does not show what happens "under the hood" when building a Fusion ART model, unless one checks manually at "fusionART.py". While building and testing a Fusion ART model, showing the function values of the model's processing cycle (e.g. code activation, code competition, etc.) would clarify the process in creating the model.

For the querying experiment, since the professor dataset only contains 83 rows, it is not sufficient to generalise the effectiveness of the model from querying this dataset to other datasets. A larger test dataset would add higher variability to the retrieval accuracy and could give a better indication of the effectiveness of Fusion ART models in retrieving data overall.

Finally, this experiment does not consider Episodic Memory-Adaptive Resonance Theory (EM-ART), a three-layer ART model that combines two Fusion ART models: one that encodes events and one that encodes episodes (Tan, Subagdja, Wang & Meng, 2019). Since the code supports EM-ART, further experiments can be done by developing EM-ART models.

5. Conclusion

This paper has presented an introduction to Adaptive Resonance Theory and its extension to Fusion Adaptive Resonance Theory, a type of neural network that learns from multiple pattern channels and machine learning paradigms in an online and incremental manner. Able to be used for a variety of applications, Fusion ART is paramount in developing advancements regarding machine learning and cognitive information processing.

Through several basic experiments using code, we can explore the processes behind Fusion ART models. This is portrayed using numerical vectors and schemas as input. A distinction of Fusion ART is that unlike other neural networks, Fusion ART networks can grow by creating new uncommitted nodes in the *F2* layer, as observed in the experimentation. While it is not possible to show the processes of these Fusion ART models in detail, such as the function values for the model's processing cycle, the experiment was successful in showing the results of the two aforementioned types of Fusion ART models. Further experiments and improvements can be done to obtain more significant results and investigate these models in more detail.

The Fusion ART model was also proven to be successful in storing and retrieving data. Using the professor dataset as an example, the model was able to store the details of each professor and retrieve those data using a query. The Fusion ART API contains functions that allow such processes to work. This model can be used for a wider variety of data.

References

Carpenter, G., & Grossberg, S. (2002). Adaptive Resonance Theory. The Handbook Of Brain Theory And Neural Networks, 2.

Ghasemian, A., Hosseinmardi, H., & Clauset, A. (2019). Evaluating Overfit and Underfit in Models of Network Community Structure. IEEE Transactions On Knowledge And Data Engineering. doi: 10.1109/tkde.2019.2911585

Grossberg, S. (2013). Adaptive Resonance Theory: How a brain learns to consciously attend, learn, and recognize a changing world. Neural Networks, 37, 1-47. doi: 10.1016/j.neunet.2012.09.017

Ignizio Burke, L. (1991). Clustering characterization of adaptive resonance. Neural Networks, 4(4), 485-491. doi: 10.1016/0893-6080(91)90044-6

Nguyen, L., Woon, K., & Tan, A. (2008). A Self-Organizing Neural Model for Multimedia Information Fusion.

Tan, A., Carpenter, G., & Grossberg, S. (2007). Intelligence Through Interaction: Towards a Unified Theory for Learning.

Tan, A., Subagdja, B., Wang, D., & Meng, L. (2019). Self-organizing neural networks for universal learning and multimodal memory encoding. Neural Networks, 120, 58-73. doi: 10.1016/j.neunet.2019.08.020