

A Coevolutionary Approach to Learn Animal Behavior With Parallel Nash Memory

ABSTRACT

This paper proposes and analyzes an approach that allows a computer program to learn the behavior of an animal by itself. Rather than simply using coevolutionary approach to learn the behavior, this approach introduces the concept of Parallel Nash Memory [1], [2]. By using coevolutionary approach, the program can simultaneously evolve fake models to approximate the real animal and classifiers to classify between fake models and real animal [1]. By introducing Parallel Nash Memory concept, the program can memorize the results coming out from coevolutionary approach [2].

The combined approach with both coevolutionary approach and Parallel Nash Memory is able to produce a better performance than simply using coevolutionary approach [1], [2]. This paper firstly illustrates how coevolution with Parallel Nash Memory worked in theory and how they are designed and implemented. Then the paper gives clues that shows how well the performance is improved by using this combined approach. Moreover, evaluation of experiment results related with other issues that may affect the performance are provided. Furthermore, the limitations and the potential improvements of the approach are discussed.

1. INTRODUCTION

In specific, the project aims to reproduce to behavior of human drivers which is specified by a Car Following Model [3]. The Car Following Model contains 6 parameters in total to specify the driving behavior which the program tries to learn.

One way to do this is to apply coevolutionary approach directly. However, unless the algorithm is carefully designed the evaluation function of it may lead to various issues that prevent stable learning progress. This means optimized fake models learnt at early iterations is very likely to be discarded later which effects program's performance. One ideal solution to solve such problem is to apply Parallel Nash Memory that can be used to memorize the optimized fake models at early stage and make sure they will not be discarded later in the iteration.

In general, in order to learn from the Car Following Model, the program should be able to satisfy three assumptions which are:

- The program can observe actions from Car Following Model and fake models. In this paper, the speed, position and acceleration of both following car and leading car are tractable at discrete steps in time.
- The program is able to simulating actions of cars. In this paper, it means that given an acceleration, speed and position for a car at certain time step the program is able to simulate this car's speed and position at next time step.
- The program is able to memorize the optimized fake models. In this paper, it means that once a learned fake model which is consider to be optimized it will never be discarded until the end of the learning iteration.

The objectives of the project is to see how well coevolution with Parallel Nash Memory approach can learn from the Car Following Model and evaluate its performance in terms of efficiency, model convergence and noise handling compared with not using Parallel Nash Memory.

Although no model is learnt completely within reasonable time due to the expensive computational effort required, the experiment result still shows identical advantage for applying Parallel Nash Memory under various situation.

2. BACKGROUND

Inspire by the idea that machines could autonomously carryout scientific investigation, a coevolutionary approach is proposed to learn animal behavior [R.], [R.7]. In principle, this approach requires no prior knowledge about the animal at all and not even need to calculate the amount of differences between animals and fake models (animates) [R.]. As a result, it could even be an ideal solution to reproduce the behavior of humans.

Although the coevolutionary algorithm is proved to be able to reproduce animal behavior, the monotonic progress is not guaranteed as mentioned in section 1. Therefore, a solution that can guarantee the monotonic progress of coevolutionary algorithm is required. Since the unstable monotonic progress is mainly resulted from forgetting (discarding), a solution for this should be a method which provides a memory mechanism to memorize the results

coming out from coevolutionary algorithm. Inspired by this idea, the concept of Parallel Nash Memory introduced by Frans is exactly the solution that can be applied to improve the performance of coevolutionary approach [F].

2.1 Reading and Research

- **2.1.1 Coevolutionary Algorithm to Learn Animal Behavior**

This approach uses a coevolutionary algorithm consist of two populations[R]. The first population are animats that referred to fake models. The second population are classifiers[R]. These two populations coevolve competitively with each other through iterations[R]. The evaluation function of the algorithm is based on fitness[R]. The fitness of the fake models depends singly on how well they are able to mislead the judgment of classifiers[R]. The fitness of classifiers depends singly on how well they are able to distinguish fake models from real animal[R].

- **2.1.2 Parallel Nash Memory**

Nash Memory is a solution concept inspired by the concept of Nash equilibrium in game theory for 2-player games [F], [F.28]. Nash equilibrium specifies a pair of mixed-strategies each for a player and each player has no incentive to unilaterally deviate to another player's strategy [F]. This makes sure that both player are using the best strategy they perform to against each other and neither of the player could be better off using other strategies [F]. For symmetric zero-sum games, the monotonicity could be guaranteed under the concept of Nash equilibrium [F], [F.8, F.13].

However, since the two populations (models and classifiers) in coevolutionary algorithm are normally not symmetric, Nash Memory cannot be applied directly. Here we introduce an extension of Nash Memory called Parallel Nash Memory which is proposed by Frans in his paper [F]. It can widen the range of problems that Nash Memory can be applied. Parallel Nash Memory could be combined with any method that finding new strategies [F]. As a result, it would be an ideal choice to combine Parallel Nash Memory with coevolutionary algorithm to guarantee the monotonic progress. It allows the third assumption mentioned in section 1 to be satisfied that the program is able to memorize the optimized fake models.

- **2.1.3 Car Following Model**

Car Following Model is a mathematical model with certain parameters that can specify the behavior of follower-car in relation to the behavior of leader-car under certain traffic environment [Deign-6]. This is the real model that my algorithm aiming to learn. Fake models (generated by coevolutionary algorithm) and Car Following Model share exactly the same mathematical structure and the only difference between them is that the parameters are different. Model with different parameters will produce different behaviors. This satisfy the second assumption mentioned in section 1 that the program is able to simulating actions of cars.

- **2.1.4 Neural Network**

Neural Network has been widely applied in the area of Deep Learning and it has shown its powerful ability in pattern recognition and machine leaning in recent years [G-Academic].

Normally a Neural Network consist of an input layer, one or more hidden layer(s) and an output layer [G-Academic]. The input layer can take the behavior observed by classifiers as input. The hidden layer performs mathematic process based on the input layer. The output layer will output the final result which contributes to the judgment of classifiers. Thus it could be an ideal structure for my classifiers to observe and classify different behaviors between fake modes and Car Following Model. By using Neural Network the first assumption mentioned in section 1 can be satisfied that the program can observe actions from Car Following Model and fake models

2.2 Project Requirements

This project investigate a method called 'Nash memory' to make the paradigm of coevolutionary algorithm more effective which requires an understanding of basic game theory, evolutionary computation and linear programming.

3. DATA REQUIRED

There will be no human data or human participant required for this project. The only data

set required is a set of parameters contained in Car Following Model given in [Car].

4. DESIGN

4.1 Design of system

● 4.1.1 Basic

The system is implemented using Python language which means it is based on Object-oriented methodology.

The version of language is Python 3.6

The version of operating system is Linux-Ubuntu-16.04.

Extra python package required:

- Theano/Lasagne [Design] – For the implementation of Neural Network.
- Nvidia display driver/CUDA – This increases the processing speed of Theano/Lasagne by using GPU instead of CPU and the related install instructions could be found here [r1]
- Matplotlib [Design] – For the implementation of plots drawing.
- PuLP – For the implementation of linear programming [r3].

● 4.1.2 System overview

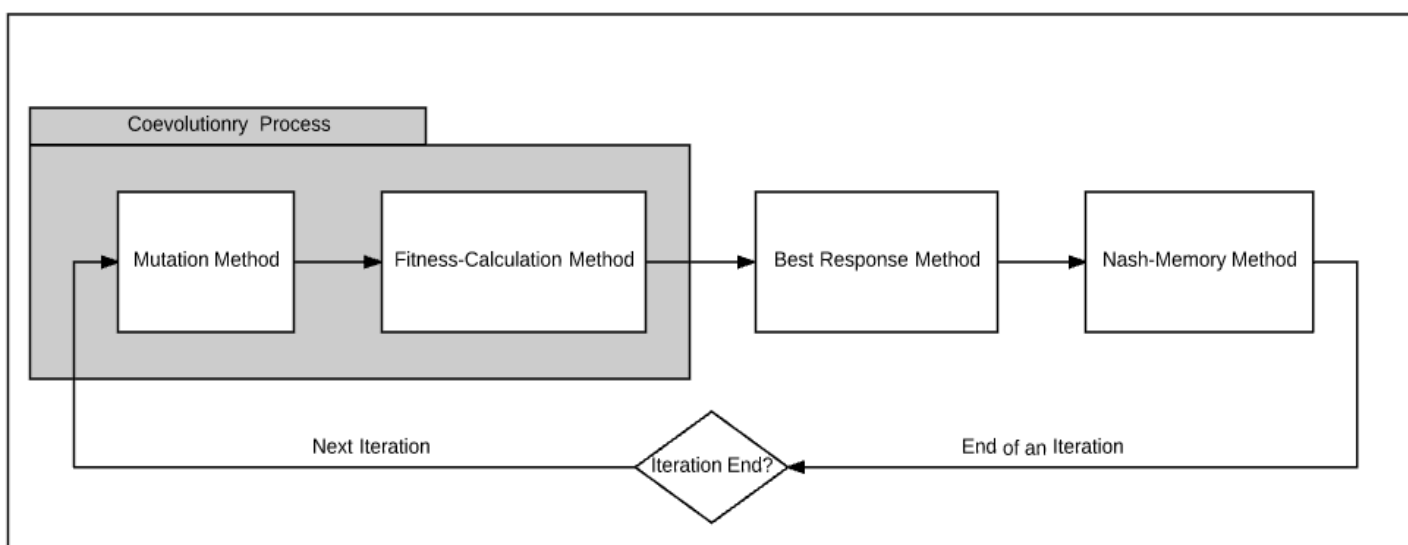


Figure 1 *System Overview*

In general the system is an iteration loop that consist of three main function modules.

- Coevolutionary Module

In this module, based on the result from last iteration, new fake models and classifiers will be generated from mutation method and added to the original population of fake models and classifiers. Then, within Fitness-Calculation method, the increased population of classifiers will try to distinguish between the increased population of fake models and Car Following Model. The performance of them will be reflected by their fitness. Finally, only individuals with higher fitness will be selected from population.

- Best Response Module

From here we introduce the idea of mixed strategy. Mixed strategy is a set of pure stratified each is assigned with a probability from 0 to 1 and the sum of probabilities is 1.

Similarly, we take each single classifier or model as a pure strategy and the mixed individuals of them will be a set of classifiers or models assigned with probabilities and the sum of probabilities is 1.

This module calculates the best mixed individuals of classifiers and fake models selected from coevolutionary process against the current support within Parallel Nash Memory (support means the best mixed individuals of classifiers and fake models come out from Nash-Memory Method since last iteration and the detail of it will be discussed later on in [4.1.3 System components]). The best responses for both classifiers and fake models will be return in the form of mixed individuals.

- (Parallel) Nash Memory Module

If best responses are not better than the current support within Parallel Nash Memory, the support will remain unchanged.

If best responses are better than the current support, the Parallel Nash Memory will be updated by those best responses and new support will be calculated for next iteration.

● 4.1.3 System components

▪ Main

The main function in controls of the general system settings and performs the running of system iteration.

It specifies the number of iteration, the number of initial populations, the number of mutated individuals during each iteration, the time step for model to simulate behavior, the total time steps for classifier to classify, the noise condition and the parameters for Car Following Model.

It passes all the specification above to a starting method which runs the iteration.

▪ Car Following Model

The Car Following Model will be used to evaluate the fitness for each classifier and the detail of it will be contained in the explanation of fitness-calculation method.

The Car Following Model I chose for my system to learn is called MITSIM [6]. The behavior of this model is specified in three regimes.

- Free Driving

When the time headway between follower and leader is larger than h_{upper} , the follower is not constrained by the leader and the model is in free driving regime [6].

In this regime the follower tries to get the desired speed. For example, if the current speed is lower than desired speed, it will use its maximum acceleration (a^+) to achieve desire speed. If the current speed is equal to desired speed, the acceleration will be 0. If the current speed is faster than desired speed, it will use its normal deceleration (a^-) to achieve desire speed. The acceleration of follower can be expressed as [6]:

$$a = a^+ \quad v < v^{desired}$$

$$a = 0 \quad v = v^{desired}$$

$$a = a^- \quad v > v^{desired}$$

- Car Following

When the time headway is between h -upper and h -lower, the follower is in the car following regime and the acceleration is constrained by the differences of speed and distance between [6]. The acceleration of follower- (a_{n-1}) can be expressed as [6]:

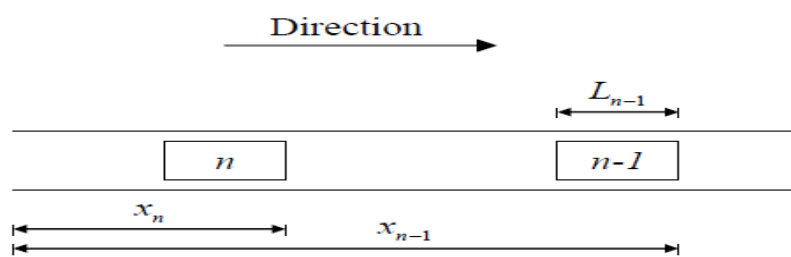
$$a_{n-1} = \alpha^{\pm} \frac{v_n^{\beta^{\pm}}}{(x_{n-1} - l_{n-1} - x_n)^{\gamma^{\pm}}} (v_{n-1} - v_n)$$


Figure 2 Car following Model notation [6]

Here α^{\pm} , β^{\pm} and γ^{\pm} are model parameters (which my system trying to learn). α^+ , β^+ and γ^+ are used when $(v_{n-1} - v_n) \geq 0$ which means the follower is accelerating. α^- , β^- and γ^- are used when $(v_{n-1} - v_n) < 0$ which means the follower is decelerating. [6]

- Emergency

When the time headway is lower than h -lower, the follower is in emergency regime [6].

In this regime, the follower tries to use a deceleration that prevents collusion and increase the time headway [6]. The acceleration of follower can be expressed as [6]:

$$a_n = \min \left\{ a_n^-, a_{n-1} - \frac{0.5(v_{n-1} - v_n)^2}{(x_{n-1} - l_{n-1} - x_n)} \right\} \quad v_{n-1} - v_n < 0$$

$$a_n = \min \{ a_n^-, a_{n-1} + 0.25a_n^- \} \quad v_{n-1} - v_n \geq 0$$

There will be only one Car Following Model in the system and the parameters of

it are shown in the Fig.3.

MITSIM

Parameter	Description	Speed [m/s]					Reference
		< 6.1	6.1–12.2	12.2–18.3	18.3–24.4	> 24.2	
a_n^-	Normal deceleration rate, vehicle n	8.7 m/s ²	5.2 m/s ²	4.4 m/s ²	2.9 m/s ²	2 m/s ²	(Yang, 1997)

Parameter	Description	Speed [m/s]			Reference
		< 6.096	6.096 – 12.192	> 12.192	
a_n^+	Maximum acceleration rate, vehicle n	7.8 m/s ²	6.7 m/s ²	4.8 m/s ²	(Yang, 1997)

Parameter	Description	Value	Reference
α^+	Car-following parameter, acceleration	2.15	(Yang, 1997)
β^+	Car-following parameter, acceleration	-1.67	(Yang, 1997)
γ^+	Car-following parameter, acceleration	-0.89	(Yang, 1997)
α^-	Car-following parameter, deceleration	1.55	(Yang, 1997)
β^-	Car-following parameter, deceleration	1.08	(Yang, 1997)
γ^-	Car-following parameter, deceleration	1.65	(Yang, 1997)
h_{upper}	Max following time headway	1.36 s	(Yang, 1997)
h_{lower}	Min following time headway	0.5 s	(Yang, 1997)

Figure 3 MITSIM parameters [6] [8]

- Model (also referred as fake model)

Models will be implemented the same structure as MITSIM. However, the value of parameters initialized for each model will be:

$$\alpha^{\pm} = 1, \beta^{\pm} = 1, \gamma^{\pm} = 1.$$

- Each model has an array of these 6 parameters.
- Each model has an array of mutation strength correspond to these 6 parameters. (For Mutation-Method)
- Models can be represented in a form of mixed models each assigned with a probability from 0 to 1 (total probabilities assigned is 1).
- Each model has a value of fitness. (For Fitness-Calculate-Method)
- Each model has a state that contains acceleration, speed and position of follower

car and leader car.

- Each model has an array that records its behavior sequence of follower car. The length of this array is the number of total time steps that the model will be observed by classifiers. There are three values from 0 to 3 each refers to a different regime. 0 referring to Free Driving, 1 referring to Car Following and 2 referring to Emergency. For example, behavior sequence [0, 1, 2] means that from time step 1 to 3 the follower car is firstly in Free Driving regime at time step 1 then in Car Following regime at time step 2 and then in Emergency regime at time step 3. This array will be useful to see how well the behavior of fake models compared to Car Following Model are converged.
- Given a certain acceleration of leader car, the model is able to record a state sequence of both follower car and leader car in to a matrix. The number of rows of this matrix refers to the total time steps and the number of columns refers to the state recorded. The state recorded is an array of length 5 refers to both cars' speed and position and the acceleration of follower car.
This matrix will then be send to classifiers for classification. Moreover, if the noise is set to true in main method, all values in this matrix would be timed by a random error factor range from 0.95 to 1.05.

▪ Classifier

Classifier can process on the state matrix returned from each model and give a judgment of whether the model is a Car Following Model or not. Both fake models and Car Following Model can be classified by classifiers.

Classifier will be based on neural network implemented by a Python library called Lasagne [D-9]. Lasagne is a lightweight open source Python library to build and train neural networks [D-7].

- Classifiers can be represented in a form of mixed classifiers each assigned with a probability from 0 to 1 (total probabilities assigned is 1).
- Each classifier has a Neural Network.

In general, the neural network structure consist of three layers:

1. Input layer

Input layer will take the state matrix returned from model as input.

2. Hidden layer

Hidden layer can specify the type of the layer, the number of hidden neurons and an activation function.

In default, we use a fully connected dense layer [r2] with 20 hidden neurons and use logistic sigmoid as activation function.

3. Output layer

Output layer can specify the number of output neurons and an activation function.

In default, we use a fully connected dense layer [r2] with 1 output neuron and use logistic sigmoid as activation function.

The logistic sigmoid activation function [D-7] used output layer will shape the output value in a range from 0 to 1. Then the classifier can make judgment on a model based on the output from output layer. If output value is less than 0.5, the classifier classifies the model as fake model. If output value is equal or greater than 0.5, the classifier classifies the model as Car Following Model.

- Each classifier has a set a value of fitness. (For Fitness-Calculate-Method)
- Each classifier can return a judgment of model

▪ Mutation Method

The method is based on a $(\mu + \lambda)$ evolution strategy which has self-adaptive mutation strengths [2], [11]. It will be applied for models and classifiers.

In this method, each parameter is related with a mutation strength (Initialized as 1) to form a pair. Each individual of model or classifier will have a set of these pairs. Here we take models as example. Assume a generation has a population of 60 models and each model has 6 parameters:

$$1) \text{ population}^g = \{m_1^g, m_2^g, m_3^g, \dots, m_{60}^g\}$$

$$2) M_k^g \text{ has } (p_k^g[i], ms_k^g[i]) \quad (0 \leq i \leq 6)$$

This means the k-th individual of model-m in generation-g has 6 parameters- $(p_k^g[i])$ each corresponds to a mutation strength- $(ms_k^g[i])$

In the population of first generation- $(population^0)$, all the parameters- $(p_k^g[i])$ are initialized as 0.0 and all the mutation strengths- $(ms_k^g[i])$ are initialized as 1.0.

Thereafter, assuming mutate 50 models each iteration, $population^g$ is used to create a mutated population of 50 models by recombination:

$$3) p_z^{g'}[i] = p_x^g[i] \text{ or } p_y^g[i] \quad (1 \leq z \leq 50; 1 \leq x \leq 60; 1 \leq y \leq 60)$$

Randomly choose 2 models from $population^g$ and recombinate their parameters to form new parameters- $(p_z^{g'}[i])$.

$$4) ms_z^{g'}[i] = \frac{(ms_x^g[i] + ms_y^g[i])}{2}$$

From 2 chosen models recombine their mutation strength to form new mutation strength- $(ms_z^{g'}[i])$.

$$5) ms_z^{g''}[i] = ms_z^{g'}[i] \exp(r'N_z(0,1) + rN_z[i](0,1))$$

Get mutated mutation strength. Here r' and r are learning rates set as $r' = \frac{1}{2\sqrt{2n}}$, $r = \frac{1}{2\sqrt{2\sqrt{n}}}$ with n refers to the number of parameters which is 6 for model. $N_z(0,1)$ and $N_z[i](0,1)$ are both random numbers generated from standard normal distribution [10] with the former generated once for each model and the latter generated for each pair of $(p_k^g[i], ms_k^g[i])$ of model m_k^g .

$$6) p_z^{g''}[i] = p_z^{g'}[i] + ms_z^{g''}[i]N_z[i](0,1)$$

Get mutated parameters.

$$7) \text{Now each mutated model-}(m_z^{g''}) \text{ has } (p_z^{g''}[i], ms_z^{g''}[i])$$

$$8) population^{g''} = \{m_1^{g''}, m_2^{g''}, m_3^{g''}, \dots, m_{50}^{g''}\}$$

Finally $population^{g''} \cup population^g$ will form a population with 50+60

models. This population will be evaluated by fitness method in order to give the new generation- $(population^{g+1})$ for next iteration.

- Fitness-Calculation Method

- Fitness of Model

The fitness of each model is given by evaluating it with each of the classifiers in the population that consist of both un-mutated and mutated classifiers. For example, if a generation contains 60 classifiers, after the mutation method, the total individual of classifiers will be 110. For every classifier that wrongly judges the model as being the Car Following Model, the model's fitness increases by 1. In the end, the fitness of each model will be range from 0 to 110.

- Fitness of Classifier

One part of the fitness of each classifier is given by using it to evaluate each of the models in the population that consist of both un-mutated and mutated models. For every correct judgment of the model, the classifier's fitness increases by 1. Similarly, if a generation contains 60 models, the fitness of each classifier for this part will be range from 0 to 110.

Additionally, another part of the fitness of each classifier is given by using it to evaluate the Car Following Model 110 times. In the end, the final fitness of each classifier will be range from 0 to 220.

After the fitness is calculated, fake models and classifiers will be ranked by their fitness and only the individuals with higher fitness will be selected as candidate solution. In default, only top 10 from 20 individuals will be selected.

- Best Response Method

This method is used to optimize the current candidate solution come out from coevolutionary process. In general, this method will calculate the best combination of fake models and classifiers in terms of mixed individuals that beats the current Nash approximation (the current support of mixed fake models and mixed classifiers in Parallel Nash Memory).

With given opponent classifiers and fake models provided by the support of Parallel Nash Memory, best response can be calculated using linear programming [].

	C1 (0.7)	C2 (0.3)
M1 (x1)	1	1
M2 (x2)	1	0
Real model (0.5)	1	0

Figure 4 *Best Response Matrix*

Here is an example showing how to calculate the best response for candidate fake models. Figure 1 shows the payoff matrix for candidate fake models against current support of mixed classifiers. Value 1 means a correct judgement made by classifier and 0 means a wrong judgement. Since fake models are trying to mislead the judgement made by classifiers, they will always trying their best to minimize the payoff of the matrix. Best response method used linear programming [] to calculate the best probabilities assigned for each candidate fake models that can minimize the total payoff of the matrix (trying best to mislead the current support of mixed classifiers). Meanwhile the total probability could be assigned for each candidate fake models is 0.5 which the same as probabilities assigned for Car Following Models. Such equal weight probabilities makes sure that the performance of classifiers is evaluated equally by how well they are able to classify fake models and how well they are able to classify Car Following Models.

Similarly, the best response for candidate classifiers can be calculated. That is using linear programming to calculate the best probabilities assigned for each candidate classifiers which can maximize the total payoff the matrix (trying best to give a correct judgement for both current support of mixed fake models and Car Following Model).

Once the pair of best responses for both fake models and classifiers are calculated, they will be evaluated against the current Nash approximation. The total payoff of the matrix will be separated into two parts for evaluation. The first part is the payoff refers to fake models and the second part is the payoff of refers to Car Following Model.

For the evaluation of best response for fake models, we only care about how well they are able to minimize the first-part-payoff of the matrix since fake models do not care about the payoff refers to Car Following Models. The range of first-part-payoff of the payoff matrix is from 0 to 0.5. If the first-part-payoff is less than 0.25, it means that the best response of fake models are in general good at misleading the judgement of classifiers provided by current Nash approximation.

For the evaluation of best response for classifiers, we care about how well they are able to classify both fake models and Car Following Models. So that both first-part-payoff and second-part-payoff will be evaluated. The range of second-part-payoff is the same as the first-part-payoff. If the first-part-payoff is larger or equal than 0.25, it means that the best response of classifiers are in general good at classifying fake models. If the second-part-payoff is larger or equal than 0.25, it means that the best response of classifiers are in general good at classifying Car

Following Model.

In my design, only when both best response of fake models and classifiers are better than ones provided by current Nash approximation, they will be selected by Nash Memory Method. Otherwise, the system will skip Nash Memory Method and start a new iteration. This not a necessary setting. An optional setting could be that as long as there exist a better best response of either fake models or classifiers, they will be selected by Nash Memory Method.

- (Parallel) Nash Memory Method

Nash Memory Method in general is a memory mechanism that maintains three sets represented by WMN. W is the set that contains the optimal individuals calculated from searching heuristic. In my design, W set will receive the support of best response calculated from Best Response Method. M is the set that maintains the individuals that are currently not in support (probabilities assigned for individuals not in support are 0). N is the set that maintains the individuals that are currently in support (probabilities assigned for individuals in support are larger than 0).

The Parallel Nash Memory maintains two sets of WMN set one for fake models and one for classifiers. Once W sets for both fake models and classifiers have been updated by the support of best responses calculated, the system will then use linear programming to calculate the new Nash approximation for the augmented parallel WMN set (augmented means that the original MN set augmented to WMN set). Once the linear programming is solved, new probabilities will be assigned for each individuals contained in (Parallel) WMN set for both fake models and classifiers. Those individuals with probabilities equal to 0 (not in support) will be maintained in M set and those individuals with probabilities larger than 0 (in support) will be maintained in N set. W set is then set to empty. Individuals in N set will be sent to coevolutionary process in next iteration.

Here is an example of how to solve linear programming for Parallel Nash Memory.

	C1 (y1)	C2 (y2)
M1 (x1)	1	1
M2 (x2)	1	0
Real model (0.5)	1	0

Figure 5 *Parallel Nash Memory Matrix*

Assume the matrix given by Figure 5 is the payoff matrix for all fake models in augmented WMN set against all classifiers in augmented WMN set. This matrix is very similar to the one in best response method. The only difference is that we calculate the probabilities for both fake models and classifiers the same time. Fake models (M1, M2) trying to calculate probabilities (x1, x2) that can minimize the matrix's payoff while the constrain is that classifiers (C1, C2) trying to calculate probabilities (y1, y2) that maximize the matrix's payoff. The result solved by linear programming here could considered as a Nash equilibrium between fake models and classifiers [Game theory].

In general, this method memorizes all the best responses of fake models and classifiers from every iteration and calculate the support of them for each iteration. Those fake models and classifiers maintained in Parallel Nash Memory can be either in support or not in support through iterations and they will never be discarded.

● 4.1.4 Objects, Attributes and Methods

- An UML-Class Diagram shows objects, attributes, methods and relations among them

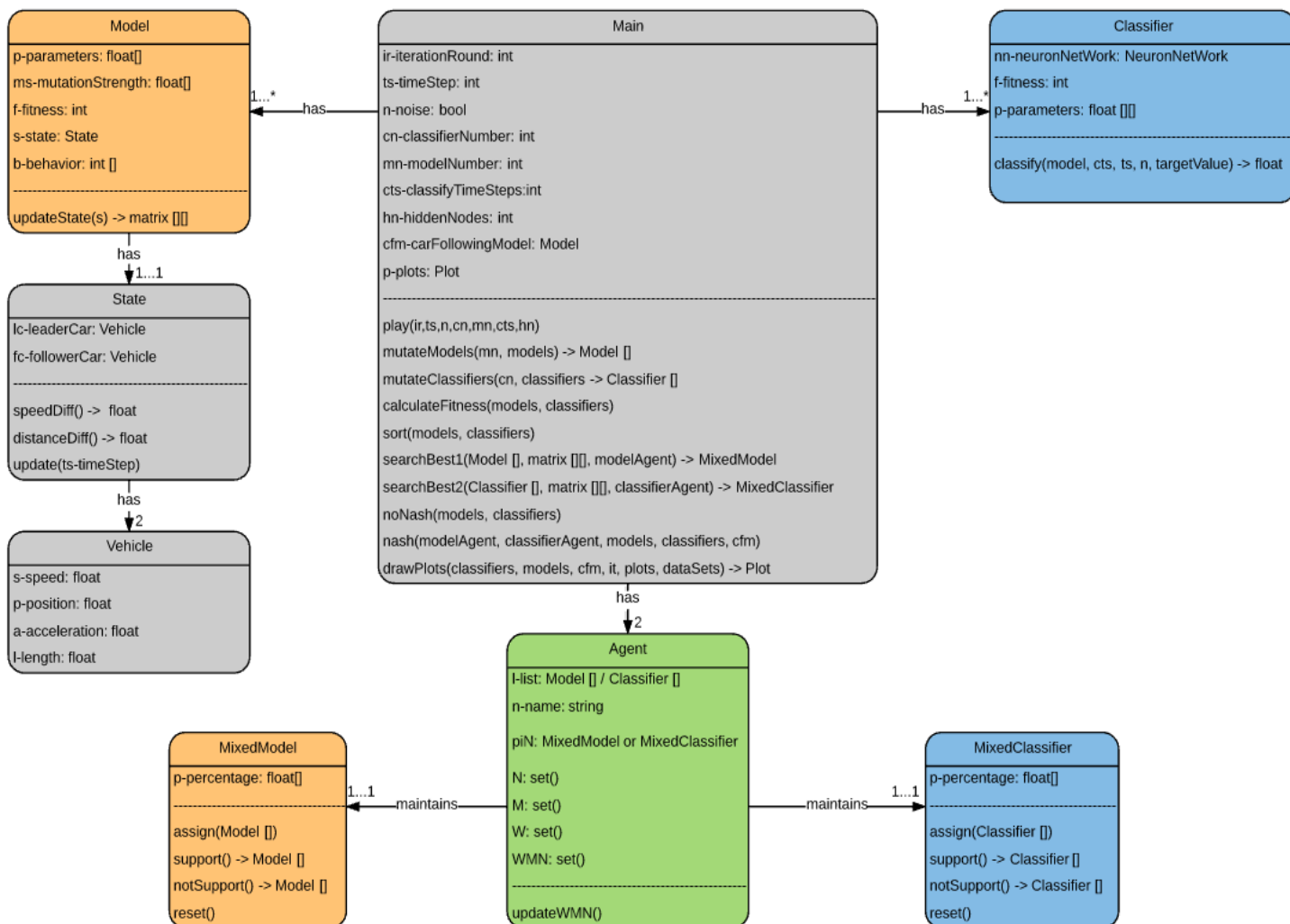


Figure 5 UML-Class Diagram

- Pseudo Code of main methods.
 - Class Main():


```

def main:
    Initialize fake model number: mn
    Initialize classifier number: cn
    Initialize iteration number: it
    Initialize time step: ts
          
```



```
Initialize classify time steps: cts
Initialize number of hidden nodes: hn
Initialize noise (Boolean): n
Initialize Car Following Model: cfm
start(mn,cn,it,ts,cts,hn,n,cfm)
```

```
def start:
```

```
def mutate_new_models(models):
```

```
    Select m1 and m2 from models
    Use m1 and m2 to mutate m3
    Models append m3
```

```
def mutate_new_classifiers(classifiers):
```

```
    Classifiers append new classifier (cts, hn)
```

```
def calculate_fitness(models, classifiers):
```

```
    for all m in models
        for all c in classifiers
            if c.judge(m) = true
                m.fitness += 1
            else
                c.fitness += 1
    for all c in classifiers
        if c.judge(cfm) = true
            c*.fitness += m.count
```

```
def sort(models,classifiers ):
```

```
    sort models by fitness
    sort classifiers by fitness
```

```
def payoff(modelSupport, classifierSupport, cfm):
```

```
    create a payoff matrix for modelSupport against classifierSupport
    payoff[0]= payoff for modelSupport against classifierSupport
    payoff[1]= payoff for cfm against classifierSupport
    return payoff
```

```
def searchBestModels(models_top10, matrix, classifierWMN):
```

```
    use linear programming to calculate the best response for models
    return best response
```

```
def searchBestClassifiers(classifiers_top10, Matrix, modelWMN):
```

use linear programming to calculate the best response for classifiers
return best response

```
def nash(modelWMN, classifierWMN, models, classifiers, cfm):  
    create a matrix for models_top10 against support of classifiers M1  
    create a matrix for classifiers_top10 against support of models M2  
  
    b1 = searchBestModels(models_top10, M1, cfm)  
    b2 = searchBestClassifiers(classifiers_top10, M2, cfm)  
  
    payoff1 = payoff (b1, classifierWMN, cfm)  
    payoff2 = payoff (b2, modelWMN, cfm)
```

use payoff1 and payoff2 to evaluate if b1 b2 is good

```
if good:  
    use linear programming to calculate the new support for
```

classifiers and models

```
        models = modelSupport  
        classifiers = classifierSupport  
    else:  
        skip to next iteration  
def noNash(models, classifiers):  
    models = models_top5  
    classifiers = classifiers_top5
```

```
def drawPlots(...):  
    draw experiment diagram
```

//Main Loop Start Here

Initialize settings

While count < it:

```
    count += 1
```

```
    for i from 0 to mn  
        mutate_new_model(models)  
        mutate_new_classifier(classifiers)
```

```
    calculate_fitness(models, classifiers)  
    sort(models, classifiers)  
    reset fitness
```

```
nash(modelWMN, classifierWMN, models, classifiers, cfm)
# noNash(models, classifiers)
```

● 4.1.5 Test Design

The test will be carried out as test driven. This means testing for each method will be considered firstly. Additionally, once code is modified or new method included all related classes and methods will be tested. However, I will not use any testing framework here since this project is more about algorithm research rather than software application. Which means I will check manually if certain method with given inputs can give the right outputs by printing them out.

For example, to test whether mutation method is working, I can send mutation method two fake models (m1 and m2) with predefined parameters and mutation strengths. After mutation method mutating m3 from m1 and m2, I can check the correctness of mutation method by printing out the parameters of m3.

● 4.1.6 Experiment Design

- Experiments to carry out:

- Experiment with/without noise
- Experiment with/without Nash-Memory method

When experiment with Nash-Memory method, the target fake models and classifiers are the support in Parallel Nash Memory from every iteration.

When experiment without Nash-Memory method, the target fake models and classifiers are individuals with higher fitness (e.g. top 5 fitness individuals) from coevolutionary process.

- Experiment with classifier using different neural network structures (e.g. different hidden nodes).

- Experiments representation:

- Draw plots figure that shows the variance of fake models' parameters every iteration. This will show how well the parameters convergent.
- Draw plots figure that shows the dynamic of the classifiers using the output value from neural network.
- Draw plots figure that shows the behavior difference between fake models and Car Following Model every iteration. This will show how well the parameters convergent.

● 4.1.7 Design of Evaluation

- Evaluate how well the system works with and without noise.
- Evaluate how well the coevolutionary process convergent.
- Evaluate how well the Nash-Memory works.

Comparing the experiment results from the system with and without Nash-Memory method.

- Evaluate how well the classifier's structure contributes to the system. Comparing the experiment results from the system using different classifier structures.

4.2 Revised Design

● 4.2.1 Memory Mechanism

In previous design, the memory mechanism was designed by adding a survival chance to each classifier and fake model individuals.

For example, assume each fake model has a survival chance of 3. If a fake model's fitness is in the TOP 50 of 100+ models then its survival chance will be kept for this iteration of optimization. If a fake model's fitness is out of the TOP 50 then its survival chance will be decreased by 1. As long as a model has a survival chance greater than 0, it will be able to join the next iteration of optimization. Once a model has a survival chance of 0, it will be considered as a poor model and discarded.

However, later on I realized that this design was just an overweight version of coevolutionary process without memorizing anything which was not even close to the concept of Nash Memory. So that I had to revise previous design and selected the concept of Parallel Nash Memory as a new memory Mechanism.

● 4.2.1 Changes of Classifier Structure

In previous design, the classifier was in control of the leader car's acceleration. Although controlled integration is proved to be able to provide a better leaning result in [R], it would be too computation expensive to be applied for my design. For instance, if the total classifying time steps is 20 then each classifier has to compute the neural network 20 times for classifying every single model (may spend several days to carry out a single experiment). So that I change the classifier structure to observing the models passively. Which means that the acceleration of leader car is given by default and models can send all their behavior over 20 time steps together to classifier. As a result, classifiers only need to compute the output of neural network once for classifying each single model.

5. REALIZATION

5.1 Implementation Stages

● 5.1.1 Stage 1- Implementation of Coevolutionary Process

▪ Implementation of vehicle class and state class

Both fake models and Car Following Model require a simulated environment to be applied for which could be provided by state class and vehicle class. The vehicle class is initialized with four parameters which refers to a vehicle's speed, position, acceleration, and acceleration. The state class is initialized with two vehicle objects refers to leader car and follower car. Three methods for state class are implemented:

- Return the speed difference of leader car and follower car.
- Return the distance difference of leader car and follower car.
- Update the state with given time step using acceleration formula for leader car and follower car. .

▪ Implementation of model class and classifier class

Model class and Classifier class are implemented the same as mentioned in '4.1.3 – System Components'.

Model class has one main method to be implemented which is to update the state object though given time steps. Based on the formula provided in '4.1.3 - Car Following Model' this method can record certain information from state object for each time step in to a matrix and return this matrix to classifiers.

Classifier class has one main method to be implemented which is to return the judgment value calculated by neural network with input matrix given by fake models or Car Following Model.

▪ Implementation in main class

For now, all the necessary classes for coevolutionary process are implemented. The next step is to implement iteration loop and necessary methods for coevolutionary process in main class. Here the necessary methods are mutation

method, fitness calculation method and sort method which are implemented exactly as specified in '4.1.3 – System Components'. Moreover, a 'noNash' method would be implemented to only remain fake models and classifiers with higher fitness and discard the rest of them (e.g. only remain individuals with top 5 fitness).

- **5.1.2 Stage 2- Implementation of Parallel Nash Memory**

- Implementation of mixed individuals and agent class

Mixed individuals is implemented for both model and classifier class. In terms of model, a class called mixedModel underneath the models class is implemented. This provides a group view of models each assigned with a probability as mentioned in '4.1.3 – System Components'. MixedClassifier is implemented the same way.

Agent class is a class that maintains the WMN set for fake models and classifier set. It can be initialized with a mixedModel object or a mixedClassifier object. This class uses a method called 'updateWMN' to refresh and maintain WMN sets. In terms of mixedModels, this method add fake models that are in support (probabilities > 0) to the set N and those fake models which are not in support (probabilities = 0) will be added to set M. If there are support individuals calculated from best response method, updateWMN method will also maintain them into W set.

- Implementation of Nash Memory method and Best Response method in main class

The implementation of Parallel Nash Memory method and Best Response method are implemented the same as mentioned in '4.1.3 – System Components'

- **5.1.3 Stage 3- Implementation of Experiment Recording**

- Implementation of experiment data recoding

This is implemented by using a set of list objects to recording the following values for each iteration.

- Variance between the average value of parameter-1 of fake models in support and the parameter-1 of Car Following Model. (e.g. if the total value of parameter-1 for 'm' fake models in support is 'a' and the value of parameter-1 for Car Following

Model is 'b', the variance is given by $(\frac{a}{m} - b)^2$

Variance for other 5 parameters are recorded the same way.

- Average variance for all 6 parameters recorded above.
- The average judgment value of classifiers in support classifying fake models in support. (e.g. if the total judgment value is 'a' and there are 'm' classifiers in support and 'n' fake models in support, the average judgment value is given by $\frac{a}{mn}$)
- The average judgment value of classifiers in support classifying Car Following Model. (e.g. if the total judgment value is 'a' and there are 'm' classifiers in support, the average judgment value is given by $\frac{a}{m}$)
- Behavior variance between fake models in support and Car Following Model. For instance, if there are 2 fake models in support with behavior sequences [0,1,1] and [1,1,2] and Car Following Models has behavior sequence [1,1,1], the behavior variance is given by following steps:
 - a) Calculate the behavior variance for each fake model against Car Following Model. Which is to calculate the behavior variance for [0,1,1] against [1,1,1] and [1,1,2] against [1,1,1]. The result for [0,1,1] against [1,1,1] is given by $(0 - 1)^2 + (1 - 1)^2 + (1 - 1)^2 = 1$. The result for [1,1,2] against [1,1,1] is given by $(1 - 1)^2 + (1 - 1)^2 + (2 - 1)^2 = 1$.
 - b) Calculate the average behavior variance which is $\frac{(1+1)}{2}=1$ (2 here refers to the number of fake models in support).

The lower the behavior variance the better the fake models can reproduce the behavior of Car Following Model.

5.2 Problem Encountered and Change Made to Design

One problem encountered during implementation stage was that due to lacking of knowledge about Theano/Lasagne package, I was not able to find a method that can retrieve all the parameters from neural network into a list object. As a result, the

mutation method for classifier is not applicable. So that when the system is introducing new classifiers, all the parameters are initialized randomly rather than using mutation method mentioned in '4.1.3 – System Components'.

5.3 Testing

In general, once system has been modified or updated, all the necessary information (including number of individuals, parameters, judgment value, etc.) for different components would be printed out in command line to check the system validation. Though some testing cases could be taken into consideration.

- Optional Test Case Setting for Coevolutionary Process
 - Initialize 5 fake models all with mutation strength to 1.
 - One of these fake models is initialized with the same parameters as Car Following Model.
 - Set iteration number to 1, mutation number to 5.
 - Initialize 5 randomly generated classifiers.

The expected result for the test case above would be 10 fake models with fitness range from 0 to 10 and 10 classifiers with fitness range from 0 to 20. The judgment value for each classifier classifying the fake model with real parameters should be the same as classifying Car Following Model. Meanwhile, parameters of 5 newly mutated fake models should not deviate too much compared to parameters of 5 original fake models.

- Optional Test Case Setting for Parallel Nash memory method
 - Initialize 5 fake models all with mutation strength to 0.
 - Set iteration number to 1, mutation number to 5.
 - Initialize 5 randomly generated classifiers.
 - Initialize modelAgent and classifierAgent each with 5 individuals assigned with random probabilities.
 - Select top 5 individuals from coevolutionary process.
 - Skip the evaluation function after best response method. This means the support

calculated from best response will always be added to Parallel Nash Memory. The expected result for the test case above would be 10 fake models with fitness range from 0 to 10 and 10 classifiers with fitness range from 0 to 20. Top 5 fake models and top 5 classifiers with higher fitness would be selected for best response method. As a result, the matrix used for best response method should contain 6 rows indicating top 5 fake models and a Car Following Model and 5 columns indicating top 5 classifiers. If the best response of fake models has 2 support individuals and the best response of classifiers has 1 support individual, then modelAgent should be extended to 7 individuals and classifierAgent should be extended to 6 individuals. After the new Nash approximation is calculated, the W set should be empty. In modelAgent, the total number of individuals in M and N set should be 7. In classifierAgent, the total number of individuals in M and N set should be 6.

5.4 Snapshot

6. EVALUATION

6.1 How Project Is Evaluated

The project is evaluated by three main criteria.

1. How well the system dealing with noise.
2. How well the classifier converged.
3. How well parameters of fake models converged.
4. How well the behavior of fake models converged.

There is no standard way for assessing those criteria since performance and settings of coevolutionary algorithm for learning various models could be very different. However, those criteria could be assessed by comparing the experiment results from the system using Parallel Nash Memory and the one not using Parallel Nash Memory.

6.2 Evaluation of Experiment Results

Default settings for all experiment are given below:

1. Initialized with 5 fake models and 5 classifiers.

2. Initialized with modelAgent and classifierAgent each with 5 individuals.
3. Each time step is set to 1 second.
4. Total time for classifying a model is set to 20 time steps (20 second).
5. Mutation number for each iteration set to 20 (for both fake models and classifiers).

The results are shown in figure consists of three parts:

1. Part 1 shows the parameter convergence of fake models evolved through iterations.
2. Part 2 shows the average judgment value for classifiers evolved evaluating both fake models evolved and Car Following Model (real model) through iterations.
3. Part 3 shows the behavior convergence of fake models evolved through iterations.

Some notification:

1. The running time for a 500-iteration-experiment is about 5 hours.
2. There is a spelling mistake in the subtitles for experiment figures below, it should be convergence not 'covergency'.

● 6.2.1 Experiment 1

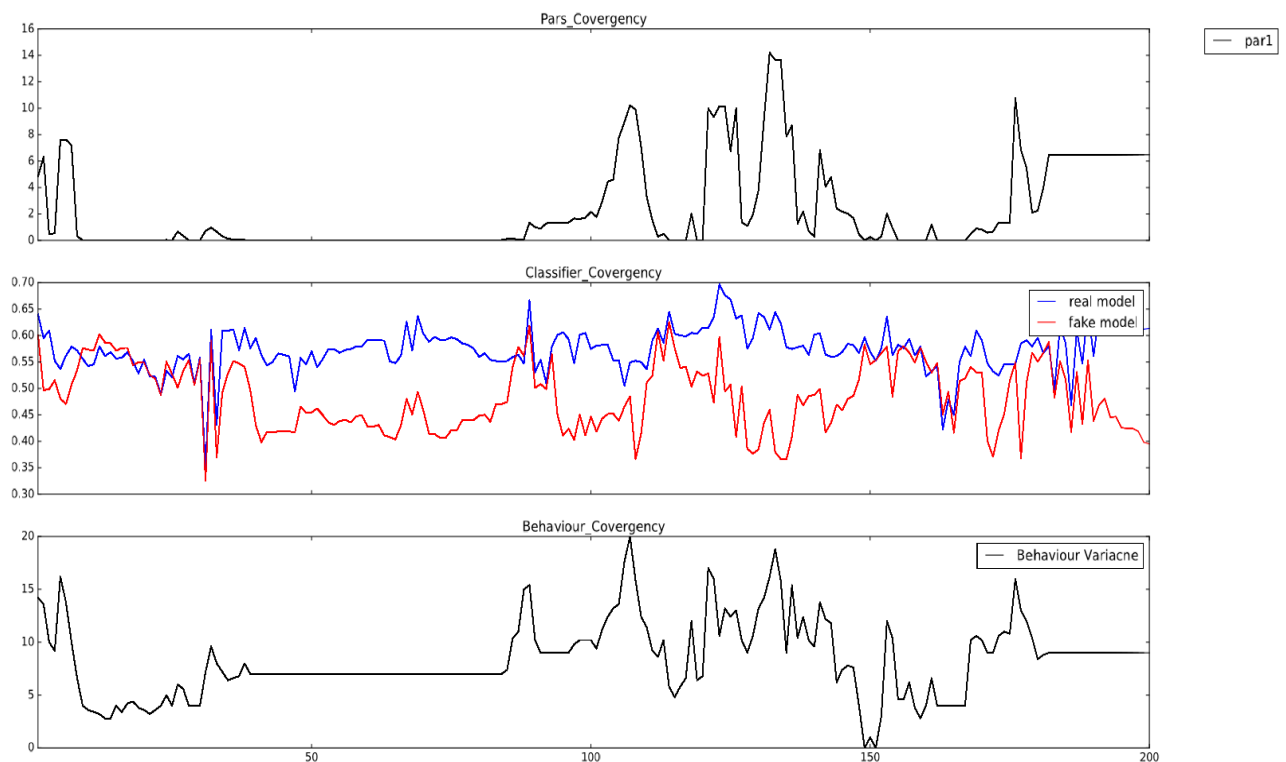


Figure 6 *No Nash Memory with noise (1 parameter)*

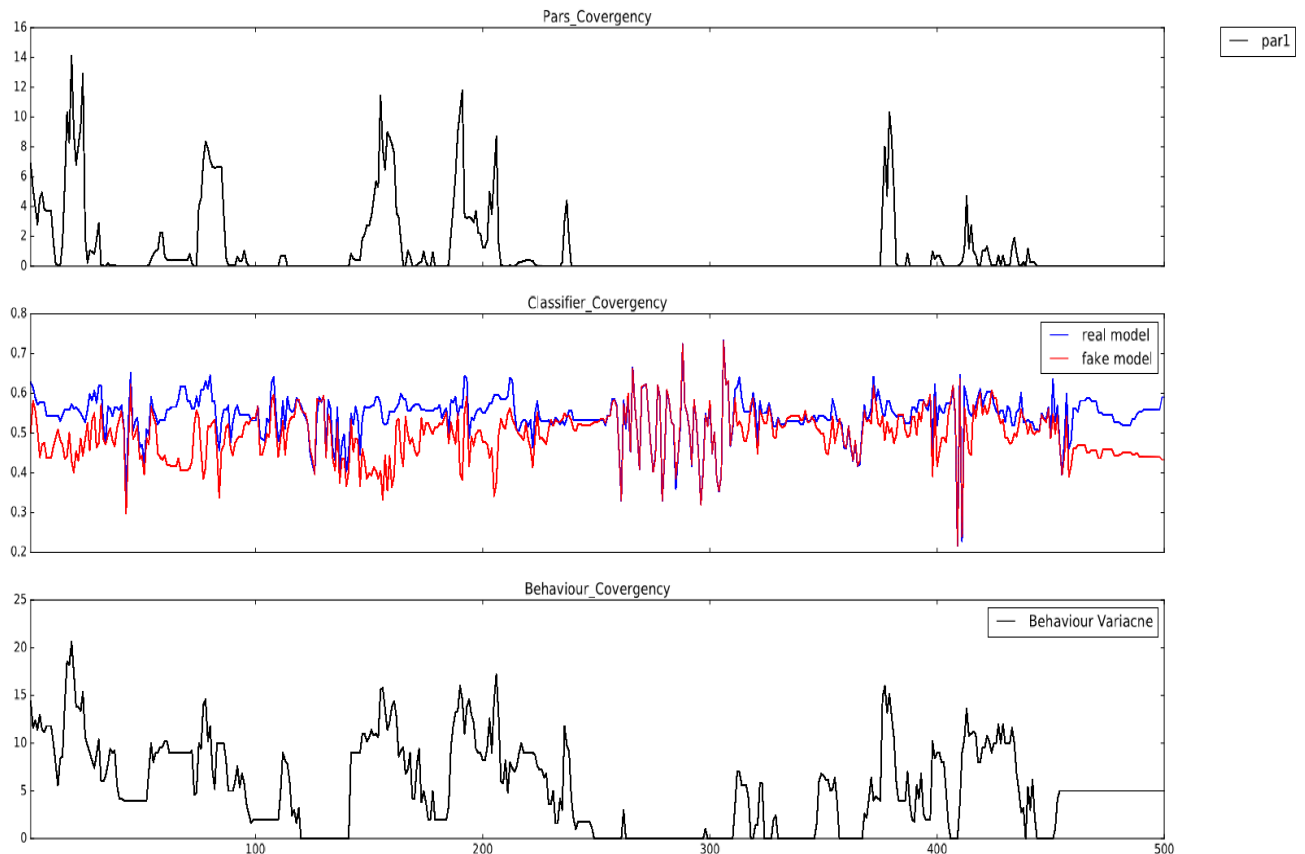


Figure 7 *No Nash Memory without noise (1 parameter)*

- Figure 6 shows the experiment result based on following settings:
 1. Only one parameter to learn. (other 5 parameters are set the same as Car Following Model)
 2. Iteration number is set to 200.
 3. Number of hidden nodes in classifier is set to 5.
 4. Noise is set to true.
 5. No Parallel Nash memory applied.
- Figure 7 shows the experiment result based on following settings:
 1. Only one parameter to learn.
 2. Iteration number is set to 500.
 3. Number of hidden nodes in classifier is set to 5.
 4. Noise is set to false.
 5. No Parallel Nash Memory applied.

In general, those two experiments shows how well coevolutionary process (without Nash memory) dealing with noise while the model to learn is simple. Identical result could be found by comparing the part of behavior convergence.

From the result in Figure 6 which dealing with noise, there is no obvious tendency shows the convergence of model behavior. Although later around iteration 150 there exists a point that behavior converging to 0, it rebounds back very quickly.

From the result in Figure 7 which not dealing with noise, there exist an obvious tendency shows the convergence of model behavior. It shows that the behavior is well converged around iteration 120 and kept converged for about 20 iterations. However, at the point around iteration 140, the behavior convergence rebounds again.

Such rebound phenomenon is mainly result from the fact that coevolutionary process without memory mechanism does not guarantee the monotonic process as mentioned before. Moreover, the reason for why converged fake models can be kept for several iterations in Figure 7 is that fake models with higher fitness are more likely to be maintained in the population. However, when the noise is added, it makes the coevolutionary process further more difficult to guarantee such maintenance. As a result very quick rebound is shown in Figure 6.

One more identical phenomenon in Figure 6 (from iteration 40 to 80) is that even very small variance of parameters can lead to a huge difference in behavior.

- **6.2.2 Experiment 2**

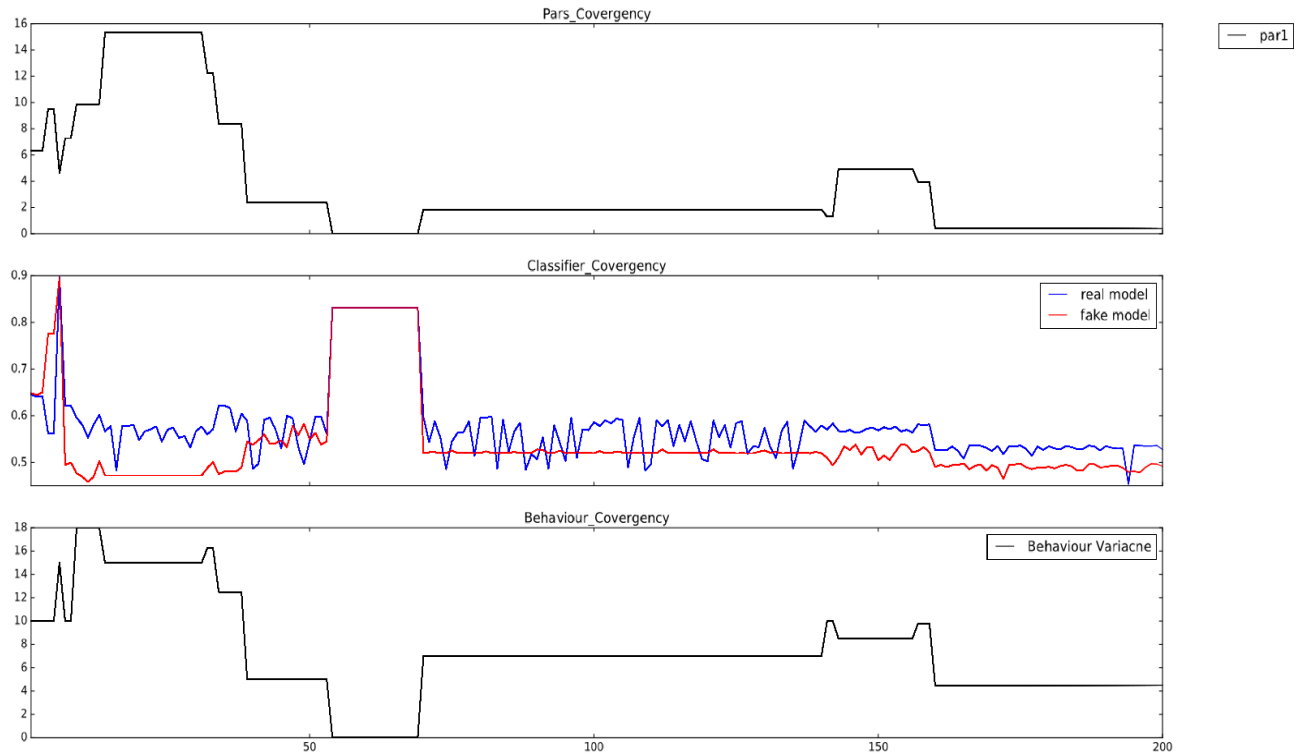


Figure 8 *Nash Memory with noise (1 parameter)*

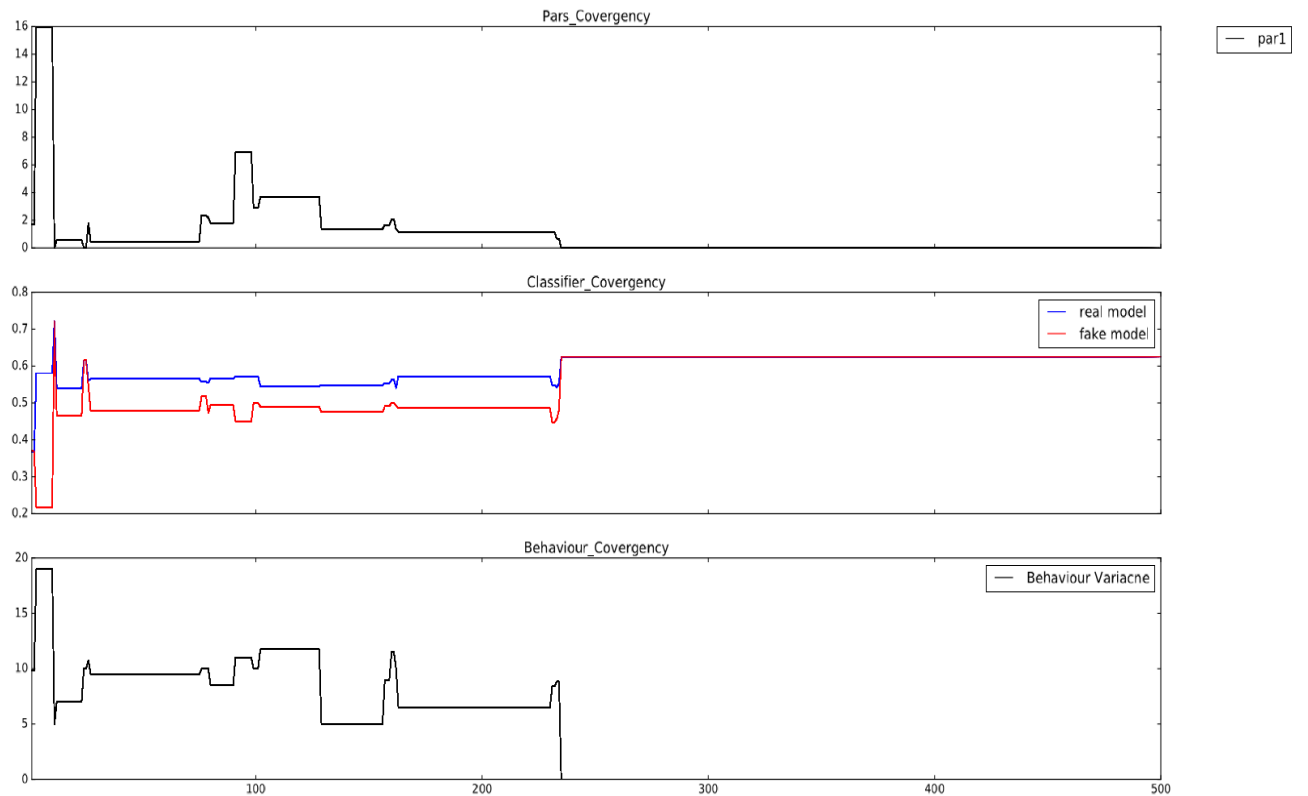


Figure 9 *Nash Memory without noise (1 parameter)*

- Figure 8 shows the experiment result based on following settings:

1. Only one parameter to learn.
 2. Iteration number is set to 200.
 3. Number of hidden nodes in classifier is set to 5.
 4. Noise is set to true.
 5. Parallel Nash Memory applied.
- Figure 9 shows the experiment result based on following settings:
 1. Only one parameter to learn.
 2. Iteration number is set to 500.
 3. Number of hidden nodes in classifier is set to 5.
 4. Noise is set to false.
 5. Parallel Nash Memory applied.

In general, those two experiments shows how well coevolutionary process (with Nash memory) dealing with noise while the model to learn is simple.

Compared to Figure 6 in Experiment 1, when Parallel Nash memory is applied, Figure 8 shows an identical convergent process even the noise is applied. For instance, both parameter and behavior convergent to 0 around iteration 55 and kept for about 20 iterations before they rebounded. This result shows that applying Parallel Nash memory can provide a significantly better convergent process while dealing with noise compared to normal coevolutionary process. However, Figure 6 shows that there still exists a rebound point around iteration 70. This is because when noise is applied, the judgment value for the same fake model made by same classifier can be different. As a result, converged fake model may still be classified from Car Following Model. Then the Nash memory method will temporally put that fake model in to M set which means not in support rather than discard it.

Compared to Figure 7 in Experiment 1, when Parallel Nash memory is applied, Figure 9 shows that there is no rebound happened when no noise is applied. This is because when there is no noise, the judgment value for the same fake model made by same classifier is exactly the same. As a result, converged fake model cannot be classified from Car Following model. Then the Nash memory method will remain that

fake model in to N set which means in support.

By comparing Figure 8 and 9, when there is no noise applied, classifiers can in general give right judgment values for both fake models and Car Following Model after the first 10 iterations. However, when there is noise applied, judgment values given by classifiers tends to be much more unstable.

In general, when there is only 1 parameter to learn, coevolutionary process with Parallel Nash memory did provide a better and more stable leaning progress than the one without Parallel Nash memory in both noise and no noise situation.

● 6.2.3 Experiment 3

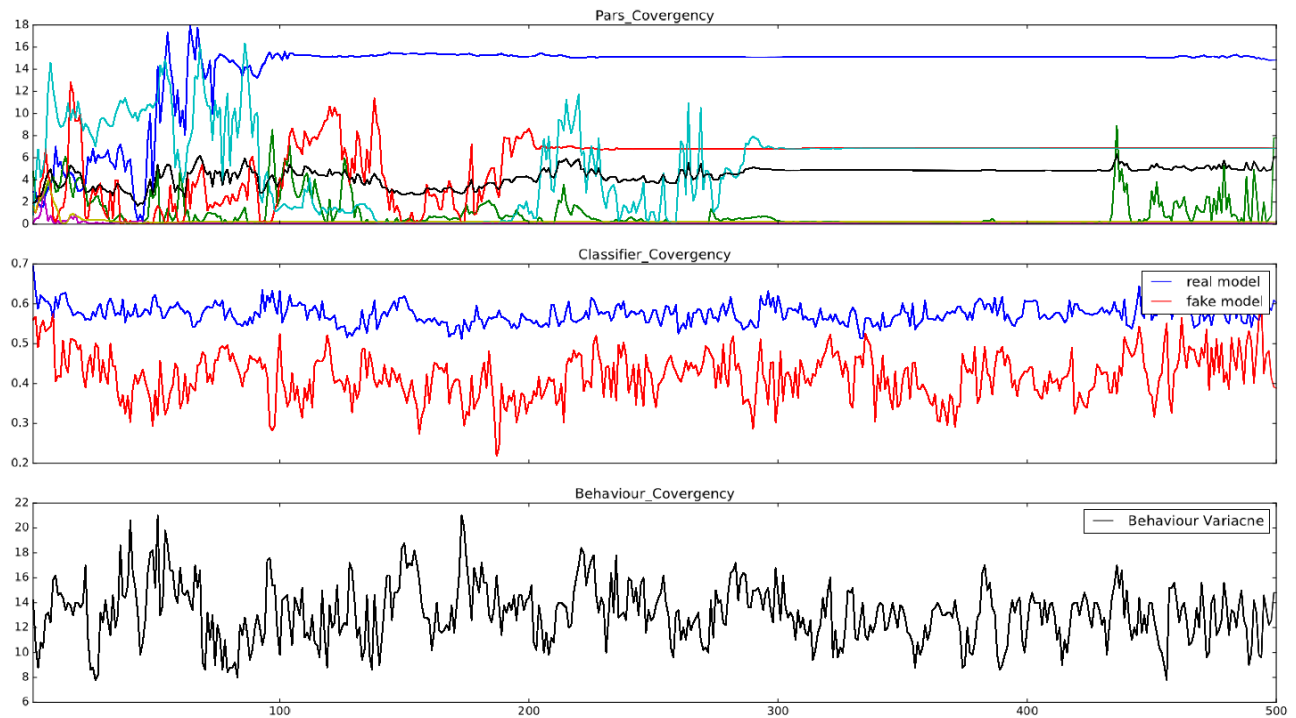


Figure 10 *No Nash Memory with noise (6 parameter)*

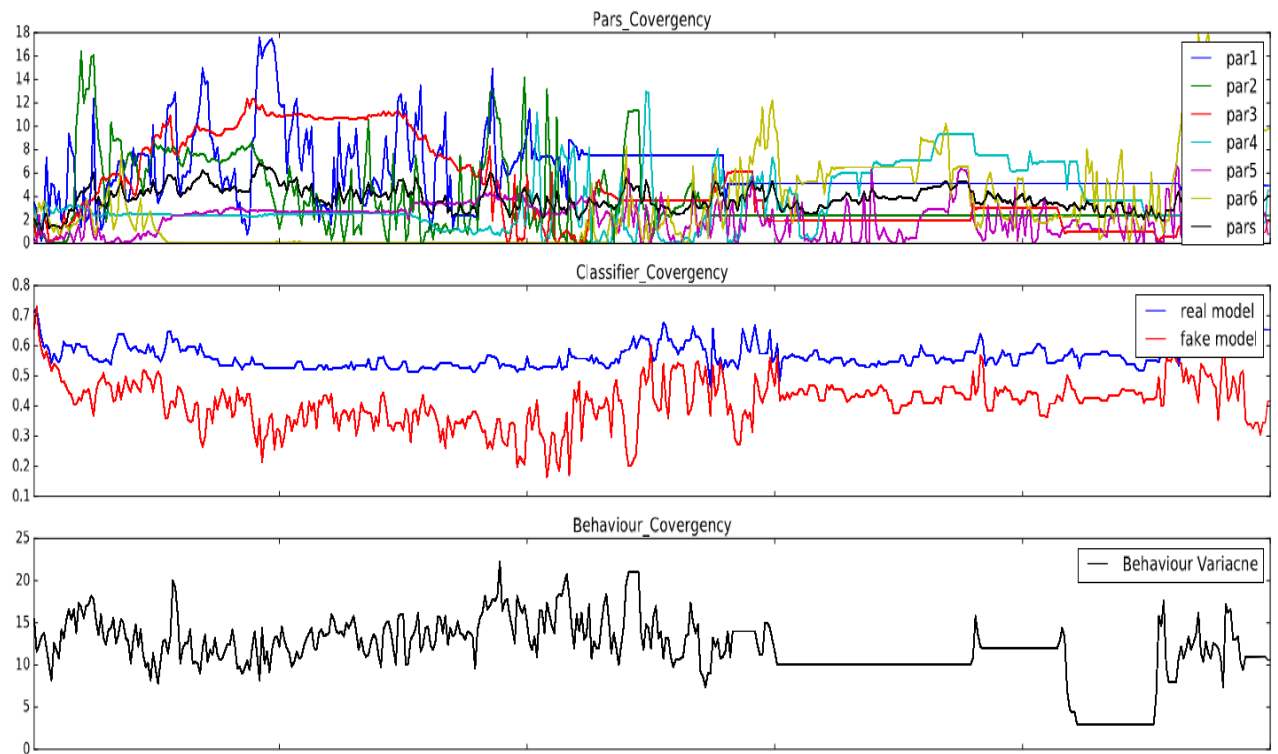


Figure 11 *No Nash Memory without noise (6 parameter)*

- Figure 10 shows the experiment result based on following settings:
 6. 6 parameters to learn.
 7. Iteration number is set to 500.
 8. Number of hidden nodes in classifier is set to 20.
 9. Noise is set to true.
 10. No Parallel Nash Memory applied.
- Figure 11 shows the experiment result based on following settings:
 6. 6 parameters to learn.
 7. Iteration number is set to 500.
 8. Number of hidden nodes in classifier is set to 20.
 9. Noise is set to false.
 10. No Parallel Nash Memory applied.

From Figure 10, it seems that when the model is complex, there is barely no tendency shows the convergence for both fake models' behavior and parameters

when noise is applied. This is because when the noise is applied to a more complex model, the uncertainty judgment value given by classifiers tends to be more unstable which leading to the situation that it is even harder for coevolutionary process to maintain optimized individuals of both fake models and classifiers. Or in other words, complexity of the model to learn and noise applied make the system much easier to forget what it have learnt so far.

Similarly conclusion could be found in Figure 11. Compered to noise applied, when noise is not applied there do exist some point that optimized fake models are evolved out and kept around iteration 420 to 450. However, different from the result of Experiment 1 showed in Figure 7, there is no clear tendency shows the convergence of fake models' behavior even the noise is not applied.

In general, the performance of coevolutionary process without Parallel Nash Memory is limited when model is complex.

● 6.2.4 Experiment 4

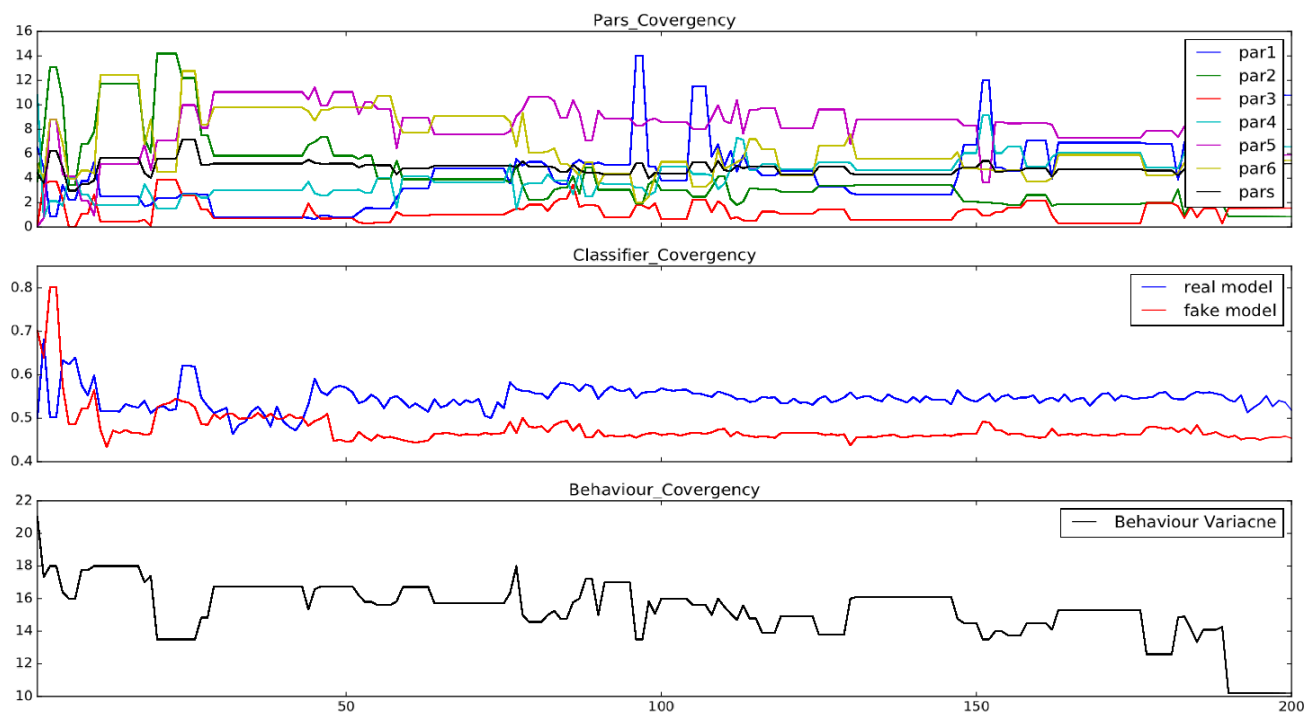


Figure 12 *Nash Memory with noise (6 parameter)*

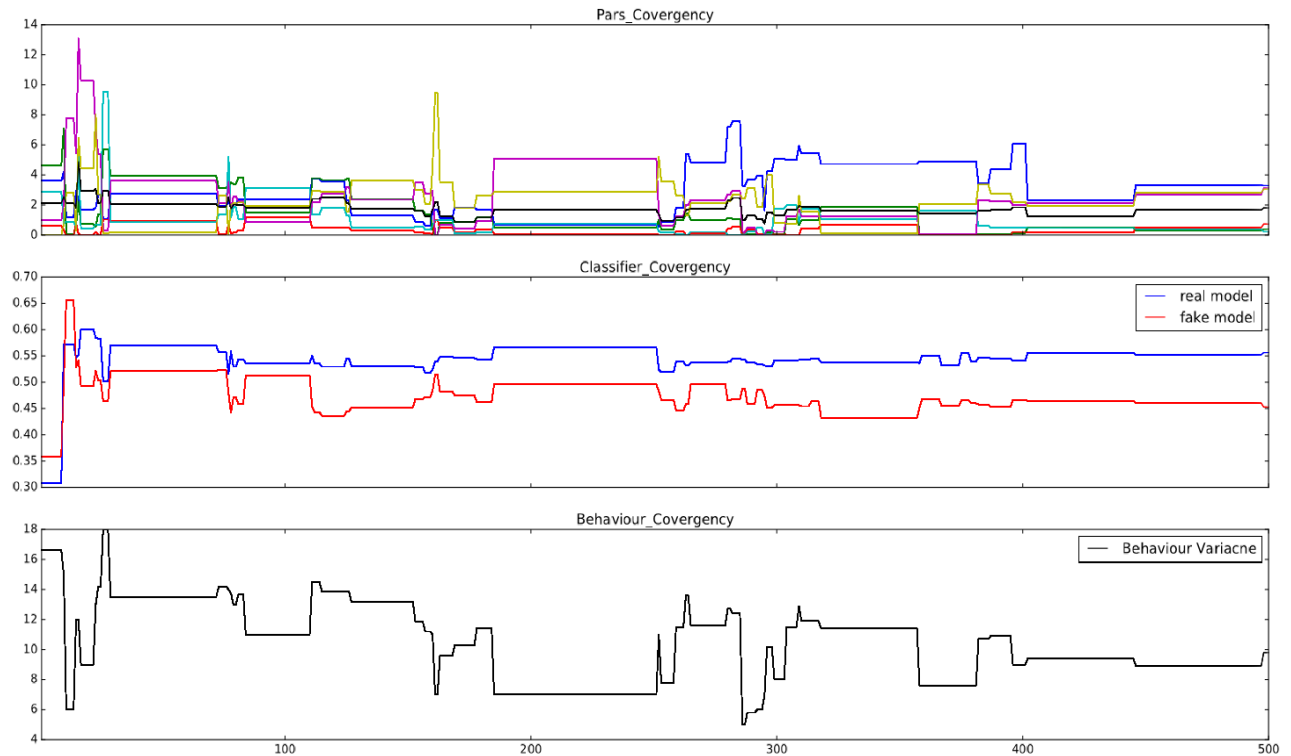


Figure 13 *Nash Memory without noise (6 parameter)*

- Figure 12 shows the experiment result based on following settings:
 1. 6 parameters to learn.
 2. Iteration number is set to 500.
 3. Number of hidden nodes in classifier is set to 20.
 4. Noise is set to true.
 5. Parallel Nash memory applied.
- Figure 13 shows the experiment result based on following settings:
 1. 6 parameters to learn.
 2. Iteration number is set to 500.
 3. Number of hidden nodes in classifier is set to 20.
 4. Noise is set to false.
 5. Parallel Nash Memory applied.

From Figure 12 and 13, it shows that when Parallel Nash Memory is applied there exist an identical convergence for fake models' behavior under both noise and no

noise situation. This means Parallel Nash Memory has the ability to hand noise even the model is complex.

Additionally, another benefit of Parallel Nash Memory can be find by comparing the classifier convergence between Experiment 2 and 4. It is obvious to see that the judgment values given by classifiers in Figure 12 and 13 are more stick to the value of 0.5 while ones given by classifiers in Figure 10 and 11 vibrate unstably around the value of 0.5. This because fake models and classifiers coevolved in a more competitive way when Parallel Nash Memory is applied. As a result, the quality of fake models and classifiers evolved can be guaranteed.

Moreover, rebound of behavior convergence can be observed frequently even Parallel Nash Memory is applied. This is because as long as fake models cannot behave the same as Car Following Model there will always exist classifiers to classify them out. When those temporally optimized fake models are classified by classifiers, they will be send to the M set (not in support). As a result, other fake models originally in M set have the chance to be selected to N set (in support) which might produce less optimized behaviors.

In general, Parallel Nash Memory provide a better performance in terms of noise handing and monotonic convergent progress. However, when the model is complex, far more iterations (or population) are required to be iterated in order to learn a completely converged model.

- **6.2.5 Experiment 5**

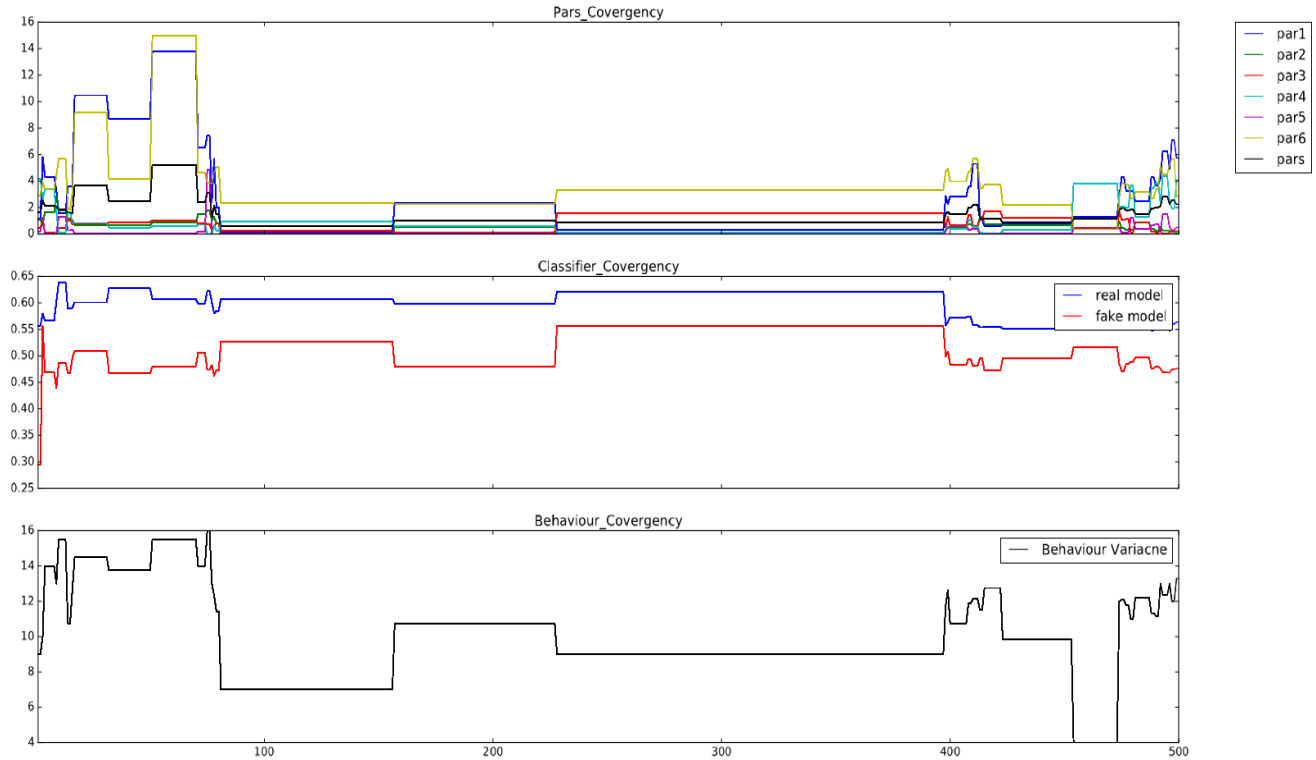


Figure 14 *Nash Memory without noise (6 parameter & 5 hidden nodes)*

- Figure 14 shows the experiment result based on following settings:
 1. 6 parameters to learn.
 2. Iteration number is set to 500.
 3. Number of hidden nodes in classifier is set to 5.
 4. Noise is set to false.
 5. Parallel Nash Memory applied.

By comparing Figure 14 and Figure 13 we can find the difference of using different number of hidden nodes in classifier.

When hidden nodes is set to 5, the frequency of finding optimized best response is lower than using 20 hidden nodes. Moreover, the convergent tendency is less obvious when using 5 hidden nodes. This is because neural network's expressivity is limited by small number of hidden nodes which directly affect the performance of classifiers.

6.3 Evaluation of Project Strength & Weakness

In conclusion, coevolutionary with Parallel Nash Memory can provide overall better performance than the one without Parallel Nash Memory. Strength could be found in both noise handling and monotonic progress guaranteeing.

There are several methods that can be taken into consideration in order to provide a better result.

- Simply iterate more iterations.
- Using classifier in controls of the environment using recurrent neural network mentioned in R's paper [1].
- Using self-learning classifier.

Create loss function that allows neural network to learn by itself.

However, no matter which method is selected, they all have to face the same problem that the coevolutionary algorithm itself is computationally expensive. Which is the main weakness of the project.

7. LEARNING POINTS

One significant point of this project for me is that it is my first time getting in touch with machine learning algorithm. If the learning of basic coding knowledge such as bubble sort algorithm helped me take the first step in software engineering, then this project helped me to take a very important step in the area of machine learning from scratch.

In terms of skills learnt, coding with Python, working with Linux system and using GitHub are main learning points. As Python is getting more and more popular these years, the learning of it will surely benefit my professional career. Learning of Linux system not only improved my command line skill but also provided me an ideal development environment which is much better than Windows system. Learning of GitHub provided me various benefits for source control.

In terms of knowledge learnt, coevolutionary algorithm, Parallel Nash Memory, linear programming and neural network are main learning points. Moreover, composing those knowledge together to solve certain problem is more than the meaning of knowledge itself. Actually, it is the art of knowledge combination and innovation. For example,

coevolutionary algorithm is inspired by the coevolutionary process of nature itself. Both of them follow the rule of 'survival of the fittest'. Moreover, Nash memory is the concept originated from game theory, linear programming is the knowledge from mathematics and deep learning neural network is inspired by the real neural network. At first glance, they might seem to be so unrelated. However, when I put all those concepts together, they have just provided a better way for model learning. From this point, I realized that the further meaning of this project is not just how machine learning algorithm can be implemented. The deeper meaning of the project giving to me is that innovation is nothing more than using what I have learnt to conquer new problems. I am sure this meaning will help me a lot later on the road to master degree.

Moreover the experience of carrying out a project from scratch is also very precious. From specification to evaluation, my abilities are improved in many aspects such as report writing, presentation making and experiment carrying.

8. PROFESSIONAL ISSUES

8.1 How Project is Related to Code of Practice

In terms of technical competence maintenance, I tried to improve my IT skills by attending all the courses offered from university and learned from other sources such as researcher's papers. I commit to continue my professional development programme and I will seek further studying and training on IT matters.

In terms of regulations adhering, I followed the project standard specified by university. All the content related with the appropriate application of the law or regulations are discussed at early stage of the project.

In terms of acting as professionally as a specialist, I kept in touch with the most recently developments related to the domain of my project. Moreover, a regular meeting was set with my academic supervisor to discuss about my project process in detail. Tools and methods used in my project are kept up to date.

In terms of research performing, although my project is related with animal behavior reproducing, no IT support of research was provided on human subjects nor animals. Meanwhile, I am very glad to share my experiment results to other students or

researchers in the university.

8.2 How Project is Related to Code of Conduct

In terms of public interest, while working on my project, I respected to the legitimate rights of Third Parties and took care about the effect to the privacy and environment.

In terms of professional competence and integrity, I developed my professional knowledge, skills and competence on a continuing basis. Meanwhile, I kept in touch the most advanced technological developments relevant to my domain.

In terms of duty to the profession, I took care about the actions within my project and none of them could bring the professional into disrepute. Meanwhile, I did encourage and support many of my schoolmates in their professional development.

9. BIBLIOGRAPHY

- [1] Wei.Li, M.Gauci and R.Gross. A coevolutionary approach to learn animal behaviour through controlled interaction. In Proceedings of the 15th annual conference on Genetic and evolutionary computation, pages 223-230. ACM, 2013.
- [2] F.A.Oliehoek, E.D.de Jong and N.Vlassis. The Parallel Nash Memory for Asymmetric Games. [GECCO '06](#) Proceedings of the 8th annual conference on Genetic and evolutionary computation, pages 337-344. ACM, 2006.
- [4] pyplot — Matplotlib 1.5.3 documentation, Matplotlib.org, 2016. [Online]. Available: http://matplotlib.org/api/pyplot_api.html. [Accessed: 16- Nov- 2016].
- [5] C. Darwin and G. Beer. The origin of species, 1st ed. Oxford: Oxford University Press, 1996.
- [6] J.Janson and O.A.Tapani. Comparison of Car-following models. In VTI meddelande 960A. Swedish National Road and Transport Research Institute, 2004.
- [7] Q.Zhang and K.C.Gupta. Neural networks for RF and microwave design, pages 61-77. Artech House, Inc., 2000.
- [8] Q.Yang. A Simulation Laboratory for Evaluation of Dynamic Traffic Management Systems. Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, page 193, 1997.

- [9] Lasagne — Lasagne 0.2.dev1 documentation, Lasagne.readthedocs.io, 2016. [Online]. Available: <http://lasagne.readthedocs.io/en/latest/>. [Accessed: 16- Nov- 2016].
- [10] What actually is bias unit? quora.com, 2016. [Online]. Available: <https://www.quora.com/In-artificial-neural-networks-what-actually-is-bias-unit>. [Accessed: 16- Nov- 2016].
- [11] H.G.Beyer. The theory of evolution strategies. Springer, Berlin, 2001.
- [12] Sumo, Sumo.dlr.de, 2016. [Online]. Available: http://sumo.dlr.de/wiki/Main_Page. [Accessed: 16- Nov- 2016].

10. APPENDICES