Building Subcomponents in the Cooperative Coevolution Framework for Training Recurrent Neural Networks: Technical Report

Rohitash Chandra, Marcus Frean, Mengjie Zhang

School of Engineering and Computer Science,
Victoria University of Wellington, Wellington, New Zealand

Christian Omlin

Middle East Technical University,
Northern Cyprus Campus, Turkish Republic of Northern Cyprus

Abstract

Cooperative coevolution decomposes a large problem into its subcomponents and uses evolutionary algorithms for solving them in order to gradually solve the large problem. This paper uses cooperative coevolution framework for training recurrent neural networks for grammatical inference problems. In the past, different encoding schemes were used to build subcomponents from the neural network for the cooperative coevolution framework. This work proposes a new encoding scheme for building subcomponents which is based on the functional properties of a neuron and compares it with the best encoding scheme from literature. All subcomponents in their respective cooperative coevolution framework employ the G3-PCX evolutionary algorithm. The results show the the proposed encoding scheme for building subcomponents achieves better performance, although, it has a lower level of modularity when compared to the CC framework used from literature. The level of modularity of the proposed encoding scheme further enables it to have smaller number of function evaluations in the initialisation stage when compared to their previous counterparts. The approach is further used for long-term dependency problems and demonstrates to learn from strings lengths of up to 500 time lags.

Key words: Cooperative coevolution, neuro-evolution, recurrent neural networks, grammatical inference, and genetic algorithms.

1. Introduction

The ability of neural networks to approximate complex functions and model any open dynamical system has well been praised (Schaefer and Zimmermann, 2007; Hornik et al., 1989; Scarselli and Tsoi, 1998; Giles et al., 1999). However, the search for its optimal training algorithm is still an open problem. Gradient descent based training paradigms are unable to guarantee a good or acceptable solution in difficult problems and those involving long-term dependencies (Bengio et al., 1994; Frasconi et al., 1993). Therefore, evolutionary algorithms(EA) have been used in neural network training and design in order to achieve a better solution when compared to traditional gradient descent based approaches (Yao, 1999). The paradigm of using evolutionary computation for evolving neural networks is known as *neuro-evolution*.

Cooperative coevolution (CC) is an evolutionary computation framework which divides a larger problem into smaller subcomponents and solves those subcomponents independently in order to solve the larger problem. The CC framework has been used for function optimisation and neural network training(Potter and Jong, 1994; Potter and De Jong, 2000; Garcia-Pedrajas et al., 2003; García-Pedrajas and Ortiz-Boyer, 2007). A major advantage of the CC framework in neuro-evolution is that it provides a mechanism for encoding regions of network weights into different subpopulations. In this way, state information in the network is preserved during learning whereas in standard neuro-evolution, there is a tendency for internal state information to be lost due to reproduction as the crossover operator affects weights of the entire network encoded in a single population.

The CC framework has been used effectively in training recurrent neural networks in order to preserve information associated with the recurrent state neurons. Pioneering work was done in their deployment as *enforced subpopulations* (ESP) for training recurrent neural networks in solving the double pole balancing problem without velocity information(Gomez and Mikkulainen, 1997; Gomez, 2003). A sophisticated version of ESP known as *Evolino* has been used to evolve the LSTM network where it was shown that the framework outperformed gradient based LSTM and learned tasks that were unlearn-able by Echo State Networks (Schmidhuber et al., 2007). Recently, the same framework with a different encoding scheme has been used for training RNN for the double pole balancing problem without velocity information. The approach was called cooperatively coevolved synapse (CoSyNE) and has shown better performance than ESP and standard neuro-evolution methods (Gomez et al., 2008).

This work is motivated by the network encoding schemes used in ESP and CoSyNE. The advantage of CoSyNE over ESP is that it achieves a higher level of modularity with a greater number of subpopulations. However, this is costly as the number of function evaluations increases with a greater number of subpopulations. In the initialisation state, CoSyNE requires a higher number of function evaluations as it consists of a higher number of subcomponents in comparison to ESP. This is one major weakness of the CoSyNE over ESP. There is a need for an encoding scheme which can train RNN as well as CoSyNE and at the same time requires a lower level of modularity. A smaller number of subcomponents would further reduce the number of function evaluations in the initialisation stage. Note that in the initialisation stage, $s \times P$ number of function evaluations are required, where s is the number of subcomponents and P is the size of the subpopulations.

The neuron based subpopulation (NSP) encoding scheme for building subcomponents is proposed which is based on the functional properties of a neuron. The performance of the NSP is compared with CoSyNE.

In the past, most CC paradigms used outdated evolutionary algorithms when compared with their recent counterparts. It has been argued that there is a need to use more recent and efficient evolutionary algorithm in the subpopulations (Yang et al., 2008). Therefore, this work employs the G3-PCX evolutionary algorithm (Deb et al., 2002) in the subpopulations of the CC frameworks. This evolutionary algorithm has outperformed CMA-ES evolutionary strategy (Hansen and Ostermeier, 2001) and other real-coded genetic algorithms for some function optimisation problems shown in (Deb et al., 2002). Furthermore, it has also been used for training feedforward neural networks in the past (CantuPaz and Kamath, 2005).

We choose a specific grammatical inference problem taken from (Blanco et al., 2001), in order to demonstrate the effectiveness of the proposed scheme. We also use two different language of average learning difficulty from the Tomita grammar (Tomita, 1982) as it has been a major benchmark for training RNN's on grammatical inference problems (Castao et al., 1995). We use the Elman style first-order RNN (Elman, 1990; Manolios and Fanelli, 1994) in order to evaluate the CC training paradigms. Furthermore, the proposed NSP CC framework is also used in learning long term dependency problems generated by a finite state automata to learn from long strings of up-to 500 time lags.

The rest of the paper is organised as follows. In Section 2, the general cooperative coevolution framework and standard neuro-evolution with a survey of CC frameworks and encoding schemes for training RNN is presented. Section 3 presents the encoding scheme in the proposed neuron based subpopulation while in Section 4, grammatical inference and their usage for studying fundamental properties of RNN are discussed. Section 5 presents the results and section 6 concludes the work with a discussion on future work.

2. Background

2.1. Neuro-Evolution and Cooperative Coevolution

Neuro-evolution has been popular for training neural networks as it employs evolutionary algorithms which are meant to handle the global search problem. It can also be easily deployed in any neural network optimisation problem without being constrained to a particular network architecture. The neuro-evolution paradigm has been used for evolving feedforward and recurrent network architectures in the past (Yao, 1999; Delgado and Pegalajar, 2005; Blanco et al., 2001; Sexton and Dorsey, 2000; CantuPaz and Kamath, 2005; Chandra and Omlin, 2007). The optimisation strategy in neuro-evolution does not depend on the back-propagation of gradient information. This is handy when gradient information is difficult to obtain as in the case for learning long-term dependencies in recurrent neural networks through back-propagation through-time (BPTT)(Bengio et al., 1994). It has been shown that evolutionary computation performs better than gradient descent in terms of recognition of unseen data in some selected problems(Sexton and Dorsey, 2000).

The application of RNN for finite state automata induction via evolutionary algorithms has been explored in the past where Wright's heuristic crossover was used in a genetic algorithm which achieved better training performance than gradient based techniques (Blanco et al., 2001). Standard neuro-evolution has also been used in evolving RNN for standard grammatical inference problems (Delgado and Pegalajar, 2005).

Cooperative coevolution decomposes a bigger problem into smaller subcomponents and employs standard evolutionary optimisation in solving those subcomponents in order to gradually solve the bigger problem. The subcomponents are also known as species and are represented as subpopulations. The subpopulations are evolved separately and the co-operation only takes place for fitness evaluation for the respective individuals in each subpopulation. The CC framework has been successfully applied to optimise high dimensional problems (Liu et al., 2001; Potter and Jong, 1994; Sofge et al., 2002) with independent variables, non-separable or dependant variable problems of high dimension (Yang et al., 2008) and multi-objective optimisation (Goh et al., 2009). Moreover, the CC framework has also been used in real-world problems such as the pedestrian detection system (Xu et al., 2006) and neural network based classifier systems (Zhu and Guan, 2008). The use of CC framework in evolving recurrent neural networks is discussed in the following subsection.

2.2. *Network Encoding Schemes for Cooperative Coevolution*

Rather than standard recurrent neuro-evolution, where the weights, biases, and context weights were encoded in a single population, the CC framework decomposes the problem into subpopulations and encodes the network in a way which would least affect the information gained in the recurrent state neurons in the evolutionary search process.

There have been two major encoding schemes based on the CC framework for training recurrent neural networks. The first scheme proposes a neuron level encoding where the each neuron in the hidden layer is used as a major reference point for each subpopulation in the CC framework. Therefore, the number of hidden neurons is equal to the the number of subpopulations. This encoding was used in *Enforced Subpopulation* (ESP) paradigms (Gomez and Mikkulainen, 1997; Gomez, 2003) where a particular neuron h_i in the hidden layer would encode the following weight links in its subpopulation:

1. The weights links connecting from the input layer to h_i
2. The weight links connecting from h_i to each context neurons
3. The weight links connected from h_i to each output layer

4. The bias associated with h_i

In this encoding scheme, the sizes of all individual subpopulations are the same for the entire framework. This encoding is shown in Figure 1.

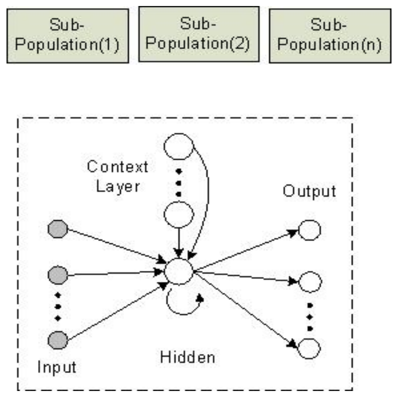

Figure 1: The ESP encoding scheme taken from (Gomez, 2003).

The second encoding scheme was presented in the cooperatively coevolved synapse neuro-evolution (CoSyNE) algorithm. This encoding scheme decomposes the network into its lowest level, where each weight link (synapse) in the network is part of a single subpopulation. Therefore, the number of subpopulations depends on the number of weights and biases. The CoSyNE demonstrated better performance than ESP on the two pole balancing problem without velocity information (Gomez et al., 2008).

3. A New Encoding Scheme: Neuron Based Subpopulation (NSP)

In ESP, a single sub-population encodes the incoming, outgoing and recurrent connection in reference to a hidden neuron. The NSP breaks down this encodings scheme into a lower level and achieves a higher level of modularity. Each subcomponent in the NSP consists of incoming weight links associated with a neuron in the hidden, state(recurrent), and output later. The CoSyNE has a greater level of modularity than NSP as it has a separate subpopulation for each weight and bias of the network. The CoSyNE has shown better performance than ESP (Gomez et al., 2008), therefore, we use it for comparison with NSP.

The neuron based subpopulation (NSP) is motivated by the properties of a single neuron which computes the weighted sum of incoming weight links associated with it. It does not include the outgoing weight links in this computation unlike ESP. Each neuron in the hidden and output layer is a reference point for a subpopulation. Each hidden neuron also acts as a reference point for the context weight links connected to it. Therefore, each subpopulation for a layer is composed of the following:

1. Hidden layer subpopulations: weight-links from each neuron in the $hidden_j(t)$ layer connected to all $input_i(t)$ neurons and the bias of $hidden_j(t)$, where t is time.
2. State (recurrent) neuron subpopulations: weight-links from each neuron in the $hidden_j(t)$ layer connected to all hidden neurons in previous time step $hidden_j(t-1)$
3. Output layer subpopulations: weight-links from each neuron in the $output_k(t)$ layer connected to all $hidden_j(t)$ neurons and the bias of $output_k(t)$

The general CC framework for NSP in training RNN is summarised in Algorithm 1. Figure 2 shows a detailed diagram of the NSP encoding scheme.

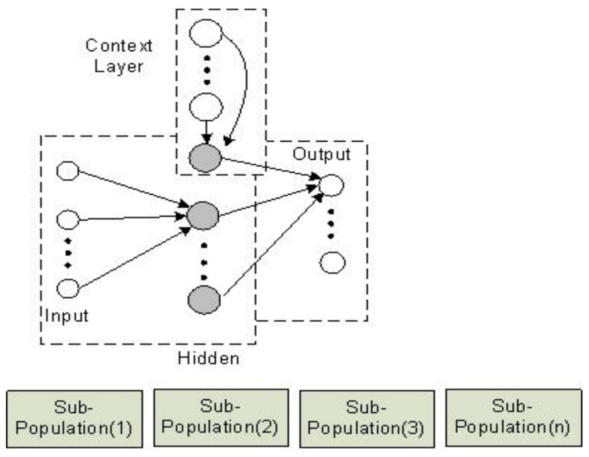

Figure 2: The NSP encoding scheme. Each neuron in the hidden and output layer acts as a reference point to its subcomponents given as subpopulations. The subpopulation for the context weights are also shown. The same encoding scheme is used in the rest of the neurons in the hidden and output layer.

In Algorithm 1, a large problem of network evolution is decomposed in k subcomponents where k is equal to the number of hidden neurons, plus the number of context neurons, plus the number of output neurons. Each subpopulation contains all the weight links from the previous layer connecting to a particular neuron. A *Cycle* is completed when all the subpopulations are evolved for a fixed number of function evaluations or generations. At the end of each Cycle, the best components of each subpopulation is cooperatively evaluated. The algorithm halts if the termination condition is satisfied.

A major concern in the general paradigm is the cooperative evaluation of each subcomponent in every subpopulation, especially in the initialisation and evaluation phases shown in Step 3 of Algorithm 1. This is simply done by assigning random best fitness individuals for each subpopulation in the initialisation phase. In order to evaluate the ith individual of the kth subcomponent, the best individuals from the rest of the subpopulations would be combined with the chosen individual and cooperatively evaluated. However, the fitness of the individual is not divided among the rest of the subpopulation (Potter and De Jong, 2000).

Algorithm 1 The NSP CC Framework for Training RNN

```
Step 1) Decompose the problem into $k$ subcomponents according to the number of Hidden,
State, and Output neurons
Step 2) Encode each subcomponent in a subpopulation in the following order:
  1. Hidden layer subpopulations
  2. State(recurrent) neuron subpopulations
  3. Output layer subpopulations
Step 3) Initialise and cooperatively evaluate each subpopulation
for each Cycle until termination do
  for each Subpopulation do
    for $n$ Generations do
      i) Select and create new offspring
      ii) Cooperatives Evaluate the new offspring
      iii) Add new offspring's to the subpopulation
    end for
  end for
end for
Get the best solution
```

Cooperative evaluation is done by combining or concatenating the chosen individual from a subpopulation k with the best individuals from the rest of the subpopulations. Note that the position of the particular subpopulation in the chromosome is retained during concatenation. The final chromosome is encoded into the RNN and the training data is used to calculate the network error which becomes the inverse of the fitness value. The goal of the evolutionary process is to increase the fitness which tends to decreases the network error. In this way, the fitness of each subcomponent in the network is evaluated until the Cycle is completed.

A major disadvantage in the encoding scheme used in ESP is that it assumes that the fitness of the recurrent state neurons should be evaluated together with incoming and outgoing weight links. The NSP evaluates the fitness of weights associated with recurrent state neurons independently. It achieves a higher level of modularity than ESP and contains smaller number of subpopulations than CoSyNE.

Other CC frameworks such as ESP and CoSyNE evaluates the fitness of each individual in n trial runs and takes the average or best fitness. This is done only in the initialisation phase. The NSP framework will use the fitness evaluation given in (Potter and De Jong, 2000) for the initialisation of subpopulations since this initialisation methodology uses smaller function evaluations than doing n trial runs. In order to show a favourable comparison with NSP, the CoSyNE will use the same method for subpopulation initialisation.

4. Grammatical Inference for the Study of RNN

The knowledge acquired in recurrent neural networks corresponds well with the dynamics of finite-state automata expressed as grammatical inference problems. They have also been used to study knowledge representation in recurrent neural networks and it has been demonstrated through knowledge extraction that RNNs can represent dynamical systems (Omlin and Giles, 1996; C. W. Omlin and Giles, 1998; Watrous and Kuhn, 1992; Chandra and Omlin, 2006). There is no feature extraction necessary in order for recurrent neural networks to learn these languages. Therefore, grammatical inference is used as an appropriate test bed for the investigation of the performance of learning algorithms for recurrent neural networks. The Tomita Grammar (Tomita, 1982) has been used as a benchmark in order to evaluate RNN training algorithms and architecture(Castao et al., 1995). Recently, it has been used to show the performance of real-time recurrent-learning algorithm (RTRL) in training a generalised recurrent network architecture (Gabrijel and Dobnikar, 2003) and for optimisation of the RNN architecture during training by an evolutionary algorithm (Delgado and del Carmen Pegalajar Jimnez, 2005).There are seven language in the Tomita grammar. This work will use two languages from the Tomita grammar (Tomita 3 and Tomita 4) given in (Gabrijel and Dobnikar, 2003). It has been shown that the first-order RNN can learn and generalise the Tomita grammars well in comparison with second-order RNNs (Miller and Giles, 1993). The goal of this paper is not to demonstrate the ability of the RNN architecture to represent the Tomita grammar, but to show the comparative performance of the proposed training paradigm in learning them. A formal definition on deterministic and fuzzy finite-state automata is given as follows.

Definition 1: A deterministic finite-state automata (DFA) is defined as a 5-tuple $M = (Q, \Sigma, \delta, q_1, F)$, where Q is a finite number of states, Σ is the input alphabet, δ is the next state function $\delta : Q \times \Sigma \rightarrow Q$ which defines which state $q' = \delta(q, \sigma)$ is reached by an automaton after reading symbol σ when in state q, $q_1 \in Q$ is the initial state of the automaton (before reading any string) and $F \subseteq Q$ is the set of accepting states of the automaton.

The language $L(M)$ accepted by the automaton contains all the strings that bring the automaton to an accepting state. The languages accepted by DFAs are called regular languages. Figure 3 shows the DFAs selected from the Tomita grammar which will be used for training the recurrent network in this study.

Definition 2: A fuzzy finite-state automaton M is a 6-tuple,
$M = (\Sigma, Q, R, Z, \delta, \omega)$, where Σ and Q are the input alphabet and the set of finite states, respectively, $R \in Q$ is the automaton's fuzzy start state, Z is a finite output alphabet, $\delta : \Sigma \times Q \times [0, 1] \rightarrow Q$ is the fuzzy transition map, and $\omega : Q \rightarrow Z$ is the output map.

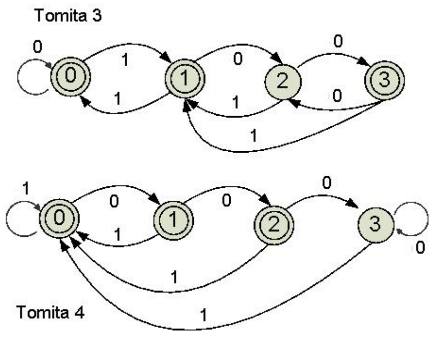

Figure 3: Deterministic Finite-State Automata from the Tomita grammar: Double circles in the figure show accepting states while rejecting states are shown by single circles. State 1 is the automatons start state.

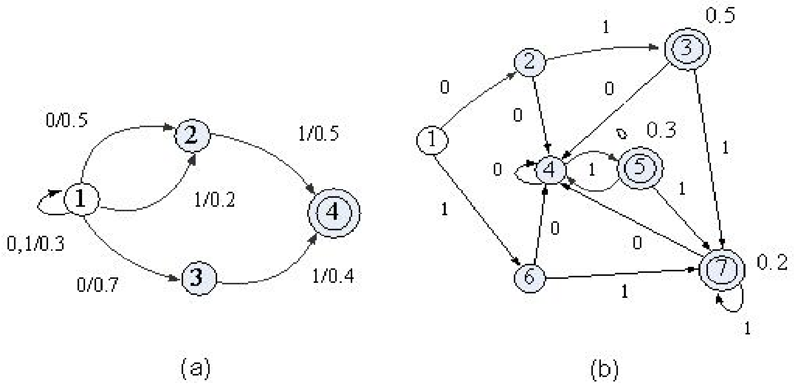

Figure 4: The fuzzy finite-state automata(a) and its equivalent deterministic acceptor(b) . The accepting states are labeled with a degree of membership. State 1 is the automatons start state; accepting states are drawn with double circles.

A restricted type of fuzzy automata is considered whose initial state is not fuzzy, and ω is a function from F to Z, where F is a non-fuzzy set of states, called finite states. Any fuzzy automaton as described in *Definition 1* is equivalent to a restricted fuzzy automaton. The transformation of a fuzzy automaton to its corresponding deterministic acceptor is discussed in (Giles et al., 1999). Figure 4 shows an example of a FFA with its corresponding deterministic acceptor which is used for training recurrent neural networks. This FFA has been used by (Blanco et al., 2001) to show that RNNs can be training by evolutionary algorithms.

5. Simulation and Analysis

This section presents a experimental study of the proposed neural based subpopulation (NSP) and compares it with CoSyNE discussed in the previous sections. CoSyNE (Gomez et al., 2008) has shown better performance than ESP and standard neuro-evolution. Therefore, it is suitable to compare the performance of NSP with CoSyNE. The G3 PCX (Deb et al., 2002) evolutionary algorithm was used in standard neuro-evolution and all the respective CC frameworks. The Elman recurrent network (Elman, 1990) with one hidden layer was uses in all experiments. The dynamics of the change of hidden state neuron activations in the Elman recurrent network is given by Equation(1).

$$S_i(t) = g\left(\sum_{k=1}^{K} V_{ik} S_k(t-1) + \sum_{j=1}^{J} W_{ij} I_j(t-1)\right) \tag{1}$$

where $S_k(t)$ and $I_j(t)$ represent the output of the context state and input neurons, respectively and $V_i k$ and $W_i j$ represent their corresponding weights. $g(.)$ is a sigmoidal discriminant function.

The G3-PCX algorithm uses a population size of 100, a pool size of 2 kids and a family size of 2 parents for all the respective CC frameworks. This set-up has been used in (Deb et al., 2002) and has shown good results for general optimisation problems. The same configuration is also used in the standard neuro-evolution for evolving the RNN. The subpopulations were initialised with random real numbers in the range of [-5, 5] in all experiments.

The FFA shown in Figure 4 is used as a benchmark problem. We generate a training dataset by presenting strings of increasing lengths of 1 to 7 to the FFA and record the corresponding output for each sample. Note that for every string length, all the possible bits are generated. Therefore, we have a training set of 255 samples. Similarly, we generate testing dataset with string lengths of 8 to 14 using the same FFA.

The recurrent neural network topology for the FFA is as follows: 1) one neuron in the input layer, 2) two output neurons in the output layer representing the 4 fuzzy output states of the FFA. The RNN is trained until 100 percent of the training sample were correctly classified. The training would halt if the number of function evaluation exceeds the maximum. The maximum number of function evaluations for all the experiments in this paper is 50000.

The Tomita grammar was also used in order to show the comparative performance of the respective encoding schemes. Tomita 3 and Tomita 4 were chosen since they are problems of average difficulty in the Tomita grammar set. The training data was generated by presenting 250 strings of random length to the respective Tomita grammars given in Figure 3 and Figure 4. The maximum string length was 30. The testing set contained 5000 strings. The RNN topology for the Tomita grammar had one neuron in the input layer and two neurons in the output layer representing the accepting and rejecting states.

The training dataset for Tomita 3 consisted of 53 negative and 197 positive strings while the testing dataset consisted 1020 negative and 3080 positive strings, respectively. Similarly, the training dataset for Tomita 4 consisted of 24 negative and 226 positive strings while the testing dataset consisted 560 negative and 4440 positive strings, respectively.

Note that the results in the paper do not include the number of function evaluations done in the initialisation state for each CC framework.

5.1. Number of generations in subpopulation

In NSP CC framework shown in Algorithm 1, each subpopulation is evolved for a fixed number of generations. We proceed our study after determining the optimal number of generations needed for the subpopulation. Note that all subpopulations are meant to evolve for the same number of *n* generations which must be fixed beforehand.

The FFA used in this experiment has 7 states and in order to make the problem more difficult, we used only 3 neurons in the hidden layer of the RNN to represent 7 states.

The results are given in Table 1 which reports the number of function evaluations (FuncEval) and the number of successful runs (Success) out 50 runs. The number of generations used for all subpopulations is also shown. Note that in the NSP column; the rows with 1 and 5 generations show similar results in terms of function evaluations. We choose 1 generation as the optimal as it has a greater number of successful runs. The results show that the optimal performance is given when the CC framework uses 1 generation of fixed evolution for all subpopulations for NSP and CoSyNE. This is shown in bold in the first entry of Table 1 which reports the least number of function evaluations with the best number of successful runs.

The results in Table 1 reveal that each subpopulation should be evolved for one generation only for the entire Cycle.The neural network training paradigm is a non-separable optimisation problem. The output of the neurons in the third layer are linked to the weights in the second layer and also dependant on the weights in the first layer. Therefore, a deep greedy search for a large number of generations is not helpful for each subcomponent. This is shown in Table 1 which reports that only 1 generation used for each subpopulation is enough to provide good results.

Generations	NSP		CoSyNE	
	FuncEval	Sucess	FuncEval	Sucess
1	**5013** $_{\mathbf{3662}}$	48	**8429** $_{\mathbf{5030}}$	**49**
3	4664 $_{1564}$	46	9720 $_{4718}$	48
5	5118 $_{2434}$	43	12849 $_{8263}$	45
7	6454 $_{6915}$	48	18983 $_{8828}$	47
9	6342 $_{7479}$	47	17190 $_{8057}$	46
11	6342 $_{7479}$	47	21183 $_{8963}$	44
13	6742 $_{8193}$	39	20904 $_{4264}$	46
15	6240 $_{6606}$	46	22185 $_{6899}$	46

Table 1: Results determine the optimal number of generations for all subpopulations in the NSP CC framework. The optimal run is determined by the least number of function evaluations (FuncEval) and greater number of successful (Success) runs in 50 experiments. Note that results from the unsuccessful runs are not included in the mean and the standard deviation. All results included in the table give a 100 percent training and generalisation performance.The best results are shown in bold. Note that the mean is reported and the number in the subscript represents the standard deviation.

5.2. Comparison with other CC frameworks

In this section, we compare the performance of the NSP CC framework with CoSyNE. Note that the original CoSyNE employed an older genetic algorithm in their subpopulation. In our implementation, the G3-PCX is used. We use 1 generation for all subpopulations in the NSP and CoSyNE CC frameworks taken from the results in Table 1.

At this stage, our experiments are not concerned in evolving the architecture while training the network. We are only concerned in finding out on the performance of the respective CC paradigms in training the weights and bias of the network. The CC framework has been used for evolving the topology of the network during learning in the past. The ESP for instance uses the "Burst Mutation" topology evolution scheme which adapts the number of hidden neurons with respect to overall network error observed over a fixed number of fitness evaluations. The implementation of "Burst Mutation" as done in ESP can be further done for the NSP CC framework in future experiments.

The main goal of this experiment was to observe the performance of the respective paradigms in relation to a particular topology; i.e. fixed number of hidden neurons. Note that the number of hidden neurons directly influences the difficulty of the learning problem. It is more difficult to learn the problem if enough neurons are not present in the hidden layer.

Table 2 shows the relationship between the number number of function evaluations and the number of hidden neurons used in NSP and CoSyNE. The RNN topology has one input neuron and two output neurons. The results show that the number of function evaluations given in terms of population size *P* increases as the size of the networks increases in terms of "Hidden" neurons which directly influences the number of subcomponents represented by the subpopulations. Note that the NSP uses fewer number of function evaluations shown in Table 2 as it has lesser number of subpopulations when compared to CoSyNE. Therefore, the initialisation of subpopulations in evaluating different subcomponent encoding schemes for the cooperative coevolution framework is an important measure. Note that the size of each subpopulation in CoSyNE is 1.

The comparative results during evolution are given in Table 3 where the attribute “Hidden” represents the number of hidden neurons. These results do not include the number of function evaluation during the initialisation phase given in Table 2 for each CC framework.

The best performance of individual experiments in Table 3 are shown in bold. They show the mean and standard deviation of the least number of function evaluations with respect to the most number of successful runs in 50 experiments. A run was considered successful if the network could learn 100 percent of the training data before the maximum number of function evaluation is reached.

Hidden	NSP	CoSyNE
3	8P	23P
4	10P	34P
5	12P	47P
6	14P	62P
7	16P	79P

Table 2: A comparison of the NSP and CoSyNE based on the number of function evaluation required during initialisation. This is for a RNN with one input neuron and two output neurons which is used in all our experiments. P is the size of the population which is 100 in our experiments.

The comparative performance given in Table 3 shows that the proposed NSP CC framework achieved better performance than the CoSyNE for 3-6 neurons in the hidden layer for the FFA problem. The CoSyNE shows good performance for 7 neurons in the hidden layer and the NSP scheme outperforms the rest of the experiments. The results for Tomita 3 also show that the NSP has performed better or similar for the given number of neurons in the hidden layer. The RNN training in Tomita 4 has also shown that the NSP has performed better than CoSyNE. Note that the best or acceptable results are shown in bold. Note that the number of function evaluation in the initialisation stage has not been included for both paradigms. If the results in Table 2 are included, the NSP will further outperform the CoSyNE as it solves the same task with smaller number of subcomponents. The results show that the NSP encoding suits grammatical inference problems where one neuron is used in the input layer. Note that NSP has been effective in giving the best performance of the most difficult situation, i.e the RNN had only 3 neurons in the hidden layer for all three problems.

5.3. Learning Long Term Dependency

A major advantage of neuro-evolution is their ability to optimise the recurrent network without acquiring gradient information as done with back-propagation through-time (BPTT). Stan-

		NSP		CoSyNE	
Problem	Hidden	FuncEval	Success	FuncEval	Success
FFA	3	**4336** **1574**	47	8868 6852	46
	4	**5199** **5944**	50	6307 3557	48
	5	**3909** **1038**	50	5762 2488	50
	6	**4431** **1866**	50	6179 2862	49
	7	6952 3201	50	**6152** **3397**	50
Tomita 3	3	**7252** **4856**	50	10869 6929	46
	4	**7362** **2784**	49	**7306** **2986**	49
	5	**6062** **2458**	49	7484 6098	48
	6	**6632** **1877**	50	**6764** **2030**	49
	7	**9206** **4074**	50	7615 1714	45
Tomita 4	3	**2760** **840**	50	5892 5156	50
	4	**3126** **1327**	50	4254 1508	49
	5	**3120** **1117**	49	6446 4811	50
	6	**4756** **3419**	50	8846 5477	50
	7	**6589** **3348**	50	8219 5515	50

Table 3: A comparison of the NSP CC framework with CoSyNE. A total of 50 experiments with different random initialisation were done.The best results are shown in bold.

dard RNN's have major limitations in learning long term dependency problems, as outlined by (Bengio et al., 1994; Frasconi et al., 1993) which suggest that simple RNN would have difficulty to learn time lags greater than 10. It was noted that the error information is not-useful for weight update in long term dependency problems. The strength of cooperative coevolution is that they provide a mechanism for retaining the knowledge gained in state neuron from crossover operator, and at the same time, it allows the RNN to learn a particular problem without the back-propagation of errors.

In order to show the contribution of CC in learning long term lags, we generate training and testing data of 500 and 5000 samples, respectively. We test the RNN to learn from random string lengths of maximum length of: 100, 200, 300, 400 and 500, respectively. We train the RNN on strings generated from the FFA in Figure 4. The results in Table 4 show that the NSP CC paradigm has demonstrated to learn long-term dependency problems in reasonable time (FuncEval) and also provides good generalisation performance (Generalise).

5.4. *Discussion*

Note that the CC framework has been used for optimisation of dependant variables (non-separable problems) and independent or separable problems, respectively. The use of the CC framework for the non-separable problem was altered in order to improve its performance on non-separable problems(Yang et al., 2008). The CC framework performs better in separable problems as the general framework employs an independent subcomponent greedy search given

Length.	FuncEval	Generalise (%)	Success/10
100	7407 $_{1461}$	99.99 $_{0.01}$	10
200	7008 $_{1960}$	99.97 $_{0.03}$	9
300	5778 $_{1669}$	99.83 $_{0.07}$	8
400	6055 $_{1760}$	99.95 $_{0.05}$	8
500	7078 $_{1560}$	99.85 $_{0.05}$	9

Table 4: Results showing the use of NSP CC paradigm in learning long-term dependency problem of respective string lengths. Note that all successful runs gave a 100 percent training and generalisation performance.

a larger number of generations for each subpopulation.

The results in Table 1 reveal that optimal results are achieved when each subpopulation is evolved for one generation only for the entire Cycle. This is because the neural network training is a non-separable problem. Therefore, a deep greedy search for a large number of generations is not helpful for each subcomponent as shown in the results.

The comparative performance given in Table 3 shows that the proposed NSP CC framework shows better performance than CoSyNE. Note that these results are specific for grammatical inference problems where one neuron is used in the input later for string input. The NSP encoding schemes has shown to have the relevant ability to effectively form the required states in the RNN during the learning process. The major advantage of the NSP is that is can represent the same problem in smaller number of subcomponents than the CoSyNE and at the same time it provides similar or better training performance. This advantage further enables the NSP to have fewer function evaluations in the initialisation phase due to lesser subcomponents when compared to CoSyNE. This is verified by the results in Table 2.

Furthermore, the general CC RNN framework has shown to learn long term dependency problems according to the results in Table 4. This success is due to the non-gradient requirement of neuro-evolution in optimising the weights of the network. Therefore, the length of the strings or time lags does not matter to neuro-evolution. Furthermore, the success of this experiment shows that standard RNN's without and enhancements in their architecture have the ability to learn long term lags. Their failure to do so in the past was due to gradient based graining algorithms and not to their architectural properties.

6. Conclusions and Future Work

The NSP CC framework has shown better properties than their previous counterparts for grammatical inference problems in general. A major advantage of the NSP over CoSyNE is that it requires a smaller number of function evaluations during the initialisation stage. Our results show, that NSP, can achieve similar of better results in grammatical inference problems although it has a lower level of modularity than CoSyNE.

The general paradigm has been successful for learning long term dependency problems and shows that standard RNN's have the architectural capability of learning problems of this nature.

Future work can examine the implementation of the NSP CC paradigm in evolving the weights and the network architecture at the same time during training. A paradigm where the different encoding schemes can be combined during training can also be explored. This would be implemented by increasing the modularity by increasing the number of subcomponents as the algorithm is progressing towards the final solution. Moreover, the need to reduce function evaluation by some means of fitness sharing among subpopulations can also be examined.

7. Acknowledgement

The first author expresses his gratitude towards Dr. Peter Andre from Victoria University of Wellington for the helpful discussions during the course of this project.

References

Bengio, Y., Simard, P., Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Networks 5 (2), 157–166.

Blanco, A., Delgado, M., Pegalajar, M. C., 2001. A real-coded genetic algorithm for training recurrent neural networks. Neural Netw. 14 (1), 93–105.

C. W. Omlin, K. K. T., Giles, C. L., 1998. Fuzzy finite state automata can be deterministically encoded into recurrent neural networks. IEEE Trans. Fuzzy Syst. 6, 76–89.

CantuPaz, E., Kamath, C., 2005. An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems. IEEE Transactions on systems, man and cybernetics-Part B: Cybernetics 35 (5), 915–933.

Castao, M. A., Vidal, E., Casacuberta, F., 1995. Finite state automata and connectionist machines: A survey. In: Mira, J., Hernndez, F. S. (Eds.), IWANN. Lecture Notes in Computer Science. Springer, pp. 433–440.

Chandra, R., Omlin, C. W., 2006. Training and extraction of fuzzy finite state automata in recurrent neural networks. In: Proc. of International Conference on Computational Intelligence. pp. 274–279.

Chandra, R., Omlin, C. W., 2007. The comparison and combination of genetic and gradient descent learning in recurrent neural networks: An application to speech phoneme classification. In: Proc. of International Conference on Artificial Intelligence and Pattern Recognition. pp. 286–293.

Deb, K., Anand, A., Joshi, D., 2002. A computationally efficient evolutionary algorithm for real-parameter optimization. Evol. Comput. 10 (4), 371–395.

Delgado, M., del Carmen Pegalajar Jimnez, M., 2005. A multiobjective genetic algorithm for obtaining the optimal size of a recurrent neural network for grammatical inference. Pattern Recognition 38 (9), 1444–1456.

Delgado, M., Pegalajar, M., 2005. A multiobjective genetic algorithm for obtaining the optimal size of a recurrent neural network for grammatical inference. Pattern Recognition 38 (9), 1444 – 1456.

Elman, J. L., 1990. Finding structure in time. Cognitive Science 14, 179–211.

Frasconi, P., Gori, M., Tesi, A., 1993. Successes and failures of backpropagation: a theoretical investigation. In: Progress in Neural Networks. Ablex Publishing. Ablex Publishing, pp. 205–242.

Gabrijel, I., Dobnikar, A., 2003. On-line identification and reconstruction of finite automata with generalized recurrent neural networks. Neural Netw. 16 (1), 101–120.

Garcia-Pedrajas, N., Hervas-Martinez, C., Munoz-Perez, J., 2003. COVNET: a cooperative coevolutionary model for evolving artificial neural networks. IEEE Transactions on Neural Networks 14 (3), 575–596.

García-Pedrajas, N., Ortiz-Boyer, D., 2007. A cooperative constructive method for neural networks for pattern recognition. Pattern Recogn. 40 (1), 80–98.

Giles, C. L., Omlin, C., Thornber, K. K., 1999. Equivalence in knowledge representation: Automata, recurrent neural networks, and dynamical fuzzy systems. Proceedings of the IEEE 87 (9), 1623–1640.

Goh, C. K., Tan, K. C., Liu, D. S., Chiam, S. C., May 2009. A competitive and cooperative co-evolutionary approach to multi-objective particle swarm optimization algorithm design. European Journal of Operational Research.

Gomez, F., Mikkulainen, R., 1997. Incremental evolution of complex general behavior. Adapt. Behav. 5 (3-4), 317–342.

Gomez, F., Schmidhuber, J., Miikkulainen, R., 2008. Accelerated neural evolution through cooperatively coevolved synapses. J. Mach. Learn. Res. 9, 937–965.

Gomez, F. J., 2003. Robust non-linear control through neuroevolution. Technical Report AI-TR-03-303, PhD Thesis, Department of Computer Science, The University of Texas at Austin.

Hansen, N., Ostermeier, A., 2001. Completely derandomized self-adaptation in evolution strategies. Evol. Comput. 9 (2), 159–195.

Hornik, K., Stinchcombe, M., White, H., 1989. Multilayer feedforward networks are universal approximators. Neural Networks 2 (5), 359–366.

Liu, Y., Yao, X., Zhao, Q., Higuchi, T., 2001. Scaling up fast evolutionary programming with cooperativecoevolution. In: Evolutionary Computation, 2001. Proceedings of the 2001 Congress on. Vol. 2. Seoul, South Korea, pp. 1101–1108.

Manolios, P., Fanelli, R., 1994. First-order recurrent neural networks and deterministic finite state automata. Neural Comput. 6 (6), 1155–1173.

Miller, C., Giles, C., 1993. Experimental comparison of the effect of order in recurrent neural networks. International Journal of Pattern Recognition and Artifical Intelligence 7 (4), 849–872, special Issue on Neural Networks and Pattern Recognition, editors: I. Guyon , P.S.P. Wang.

Omlin, C. W., Giles, C. L., 1996. Constructing deterministic finite-state automata in recurrent neural networks. J. ACM 43 (6), 937–972.

Potter, M. A., De Jong, K. A., 2000. Cooperative coevolution: An architecture for evolving coadapted subcomponents. Evol. Comput. 8 (1), 1–29.

Potter, M. A., Jong, K. A. D., 1994. A cooperative coevolutionary approach to function optimization. In: PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature. Springer-Verlag, London, UK, pp. 249–257.

Scarselli, F., Tsoi, A. C., 1998. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. Neural Networks 11 (1), 15–37.

Schaefer, A., Zimmermann, H., 2007. Recurrent neural networks are universal approximators. Int. J. Neural Systems 17 (4), 253–263.

Schmidhuber, J., Wierstra, D., Gagliolo, M., Gomez, F., 2007. Training recurrent networks by evolino. Neural Comput. 19 (3), 757–779.

Sexton, R. S., Dorsey, R. E., 2000. Reliable classification using neural networks: a genetic algorithm and backpropagation comparison. Decision Support Systems 30 (1), 11–22.

Sofge, D., De Jong, K., Schultz, A., 2002. A blended population approach to cooperative coevolution fordecomposition of complex problems. In: Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on. Vol. 1. Honolulu, HI, USA, pp. 413–418.

Tomita, M., 1982. Dynamic construction of finite automata from examples using hill-climbing. In: Proceedings of the Fourth Annual Cognitive Science Conference. Ann Arbor, MI., pp. 105–108.

Watrous, R. L., Kuhn, G. M., 1992. Induction of finite-state languages using second-order recurrent networks. Neural Comput. 4 (3), 406–414.

Xu, Y. W., Cao, X. B., Qiao, H., 2006. A low-cost pedestrian detection system with a single optical camera. In: Intelligent

Control and Automation, 2006. WCICA 2006. The Sixth World Congress on. Vol. 2. pp. 8759–8763.

Yang, Z., Tang, K., Yao, X., 2008. Large scale evolutionary optimization using cooperative coevolution. Inf. Sci. 178 (15), 2985–2999.

Yao, Z., 1999. Evolving artificial neural networks. Proceedings of the IEEE 87 (9), 1423–1448.

Zhu, F., Guan, S.-U., 2008. Cooperative co-evolution of ga-based classifiers based on input decomposition. Engineering Applications of Artificial Intelligence 21 (8), 1360 – 1369.