Implementation of Finite State Automata using fLIF Neurons

Yulei Fan & Christian Huyck
Middlesex University of London, UK
y.fan@mdx.ac.uk c.huyck@mdx.ac.uk

Abstract – *This paper presents a novel approach to implementation of finite state automaton (FSA) using fatiguing Leaky Integrate and Fire (fLIF) neurons. This approach uses fLIF neurons because they are a model of biological neurons that closely approximates the basic functions of neurons. FSAs are an important mechanism for processing information. In this paper, the creation and implementation of two FSA models is described; the results show that all input sentences are correctly marked as acceptable or unacceptable.*

Keywords: Fatiguing Leaky Integrate and Fire Neurons, Finite State Automata, Parser

1 Introduction

A Finite State Automata (FSA) is a standard mechanism for processing information [Lewis and Papadimitriou, 1981]. FSAs are not Turing complete, but they can be (and are) used for a wide range of software and hardware applications.

A fatiguing Leaky Integrate and Fire (fLIF) neuron is a model of a biological neuron that is reasonably accurate [Huyck, 2007]. This model has been used to learn Cell Assemblies (CAs) [Hebb, 1949]. CAs are a widely agreed upon neural basis for concepts and are thus quite important. However, a CA is an attractor state. While recognising objects and moving to attractor states is important, a complete processing system needs to do more than this; it needs to move to new states for processing.

A games agent has been implemented in fLIF neurons for the CABot project ($http://www.cwa.mdx.ac.uk/CABot1/CANT.html$); the agent views the environment, takes one of several commands from the user, parses [Huyck and Fan, 2007] the commands into semantic frames, sets goals based on the semantics and executes actions. For example, when the user types "turn toward the pyramid", the agent parses it, sets up a goal of "turn toward pyramid" and executes an action of either "turn left" or "turn right" depending on the location of the pyramid. This agent uses an informal processing system.

In this paper, a more formal processing system, FSAs, are described. The core pieces are described, along with implementations of several particular FSAs.

2 Basic fLIF Model

The basic fLIF model is an idealized model of a biological neuron, which is a modification of the integrate and fire neuron [McCulloch and Pitts, 1943, Hopfield, 1982]; this has been extended to the commonly used Leaky Integrate and Fire model [Maass and Bishop), 2001].

2.1 Leaky Integrate & Fire Neurons

The integrate and fire neuron is a spiking neuron, i.e., each neuron collects energy from other connected neurons. When the energy (activation, or potential) passes a threshold, the neuron fires and sends out energy to other neurons. A mathematical model of the neuron is given by Equation 1

$$E_i(t) = \frac{1}{d}E_i(t-1) + \sum W_{ji} \quad (1)$$

where, W_{ji} is the connection weight between neuron-i and neuron-j, and neuron-j is fired in the prior step; $E_i(t)$ is the new energy of neuron-i, $E_i(t-1)$ is the old energy of neuron-i, and d is a decay constant (leak) that is greater than 1. That means if a neuron does not fire, its energy is reduced by a decay parameter d; some of its energy leaks away. If a neuron fires, it loses all its energy.

2.2 Fatigue and Fatigue Recovery

Each neuron has a fatigue value associated with it. When a neuron fires, its fatigue is increased, and when it does not fire the fatigue is reduced. A neuron fires if it has enough activation to surpass the firing threshold plus the neurons fatigue. This means a neuron becomes more difficult to fire the more it is fired.

Equation 2 shows the change of fatigue level depending on the neurons firing behaviour. F_t^i and F_{t-1}^i are the fatigue values of neuron i at times t and $t-1$; F_c is the fatigue constant for increasing the fatigue value of the fired neuron; F_r is the fatigue recovery for reducing the fatigue value of the unfired neuron; both are positive.

$$F_t^i = \begin{cases} F_{t-1}^i + F_c & \text{fire} \\ F_{t-1}^i - F_r & \text{not fire} \end{cases} \quad (2)$$

Equation 3 represents the effective change in firing threshold at each cycle, a combination of Equation 1 and

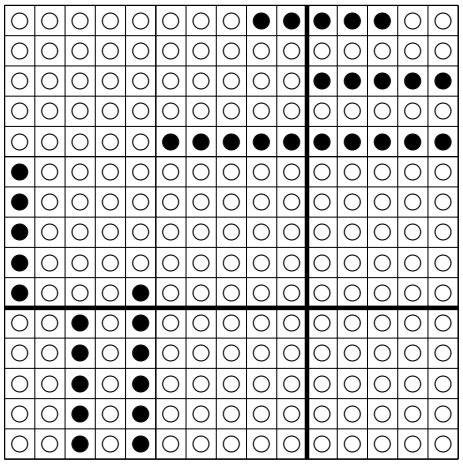

Figure 1: A Sample CA Network.

Equation 2.

$$\theta_t^i = \theta + F_t^i \tag{3}$$

θ_t^i is the new threshold for neuron i and θ is the initial threshold. Again θ is a constant for all neurons in a subnet.

Based on Equations 1, 2 and 3, when $E_t^i \geq \theta_t^i$, the neuron fires, otherwise it does not fire. Firing behaviour can be restated as Equation 4.

$$\begin{cases} E_t^i - F_t^i \geq \theta & \text{fire} \\ E_t^i - F_t^i < \theta & \text{not fire} \end{cases} \tag{4}$$

2.3 Topological Structure of the Network

A network of neurons can be described by a two dimensional matrix. An example net is shown in figure 1, which is a 15 × 15 net containing 225 neurons, where each circle represents a neuron, a filled circle shows a fired neuron and a white circle shows an unfired neuron. If the activation of a neuron surpass a threshold, the neuron is activated or fired, it spreads energy to other connected neurons, then loses all energy. Each neuron is a fLIF neuron.

3 Finite State Automaton

The FSA technique is a 'language recognition device' or a translator, like a compiler [Lewis and Papadimitriou, 1981]. An FSA contains a finite number of states, and its mathematical model can be represented by: (S, Σ, T, S_0, A), where:

S is a finite non empty set of states.

Σ is the input alphabet (a finite non empty set of symbols).

T is a transition function (T: S×Σ→S).

S_0 is start state, an element of S.

A is a set of accept states, a (possibly empty) subset of S.

A state indicates information about the past; i.e. it reflects the changes of state due to earlier input. A transition indicates a change of state due to the current input.

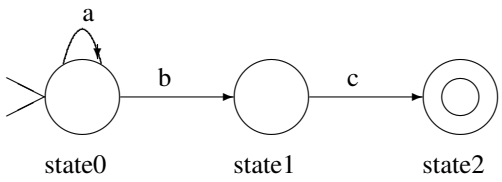

Figure 2: A Simple FSA for a*bc.

An FSA is commonly represented or described by three methods: state diagram, state transition table and state logic. For instance, figure 2 shows a simple example of FSA drawn in a state diagram.

Where, 'a', 'b' and 'c' are the inputs, state0 is the start state, and state2 is the final state. When the FSA is in state0 and the input is 'a', the FSA remains in state0; when the input is 'b', the FSA switches to state1. When the FSA is in state1 and the input 'c', the final state2 comes on, i.e., the goal has been achieved. That is, for this FSA, the input string *abc* is an element of the language it recognizes. In general, there are many routes for reaching the final state, depending on the particular FSA and on the input sequence. FSAs only accept the correct sequences of input reading, so that the final goal can be achieved.

4 Structures

There are many ways to implement FSAs with fLIF neurons. In this paper, three particular structures are used to implement FSAs. The first is called a persistent net, which continues to fire until shut down by another net; the second is called an instant net, which only fires for a few cycles and dies off. Instant nets and persistent nets are sufficient to implement many FSAs. The third type of structure is called a resistant net, which acts as a stub for the end of an arc to avoid problems that some FSAs cause for systems based solely on the first two structures.

4.1 Persistent net

The main feature of a persistent net is that, once started, the net continues to fire without external input from outside the net. This persistence is a basic function of CAs, but, for clarity, in this paper, it is done with only six neurons. The parameters and topology determine the performance of a fLIF net. The main parameters are threshold, decay, fatigue, fatigue recovery, the connections and connection weights.

Figure 3 shows the topology of a persistent net. A circle represents a neuron and they are labelled A to F. Connections are weighted and the bidirectional arrow (e.g. A⟷B) represents two uni-directional connections. The values of threshold, decay, fatigue and fatigue recovery are 4, 2, 1 and 2 respectively. The Activity is governed by equation 1, fatigue by 2 and firing by 4. A persistent net has a repeatable firing pattern, which makes each state of an FSA model deterministic and stable. If ABC fire, the system will oscillate between ABC and DEF. This particular

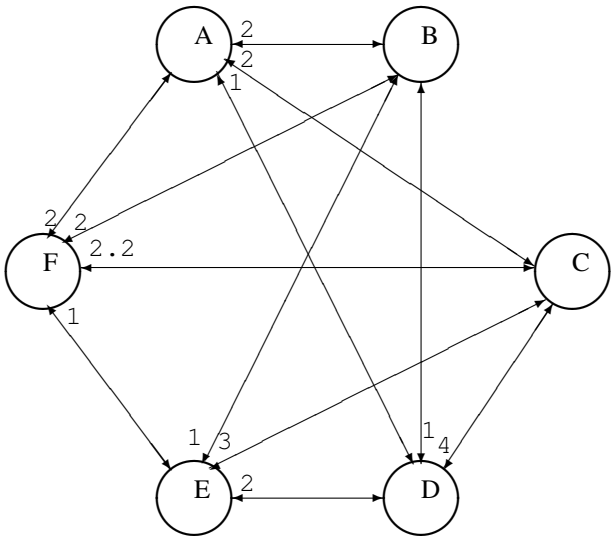

Figure 3: The Topology of a Persistant Net of fLIF Neurons.

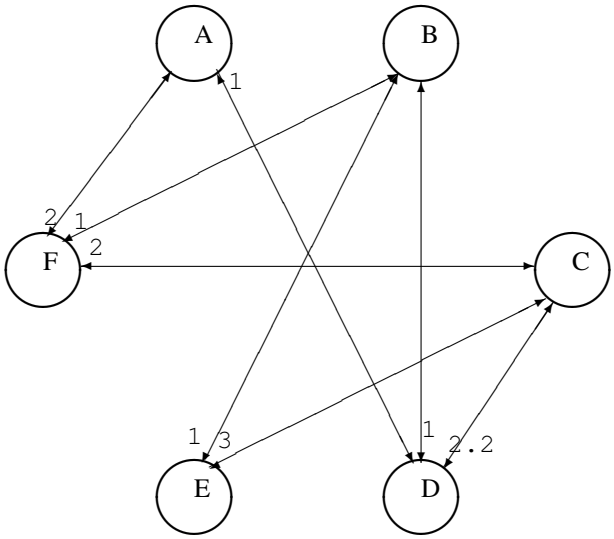

Figure 4: The Topology of an Instant net of fLIF neurons.

system more or less ignores the fatigue behaviour by having each neuron fire every other cycle and having fatigue, F_c, less than fatigue recovery, F_r.

4.2 Instant net

The instant net fires for a few cycles and then dies off. An instant net can be easily obtained by modifying the connections of a persistent net. Figure 4 shows an instant net; it is a simple modification of the persistent net by changing some connections and weights. The instant net is used to represent the input of an FSA model. If ABC are fired, DEF will fire in the next cycle, then C and then nothing.

4.3 Resistant Net

A resistant net is used in simulating particular FSAs. FSA transitions are represented by resistant nets and once activated, the resistant nets pass on energy to the destination state. The resistant net has the same topology as the persistent net, but there are different connections between structures.

5 FSA Implementation

These basic components can be combined to implement FSAs. Input is represented by instant nets, and states are represented by persistent nets. This is enough to implement many but not all FSAs, and one that can be implemented is described next. After that the resistant net is introduced to account for two problems, and thus all remaining FSAs.

5.1 FSA Components

Each state is represented by a persistent net. The system starts by turning on the initial state. Neurons ABC of figure 3 are fired by external activation (outside the network). This state will remain active until there is an input. After this initial firing, the state nets never receive external activation. Aside from during state transition, only one state is active at a time.

Each input is represented by an instant net. When there is an input, the appropriate input instant net is started. It is assumed that inputs come at least 10 cycles apart, but anything longer than that is acceptable. The input net is started by firing ABC of figure 4 by external activation.

The combination of activity from the current state and the input neurons causes a new state to become active. This new state suppresses the old state, and only one state remains on. The activity is propagated by connections from the current state to all states that are immediately after it.

The connection weights between states are supported neurally by one to one connections with weight 1.5; e.g. neuron A in state S_0 is connected to neuron A in state S_1, B is connected to B. Alone, these connections do not cause the neurons in the next state to fire. There are also connections from each input to all states that are transited to on a particular input. The connection weights of input to state are again neuron to neuron (A to A ... F to F) with weight 2.5. Again, alone these connections are insufficient to cause the neurons to fire.

The combination of connections from the state and the input is sufficient to cause the neurons in the next state to surpass their firing threshold and fire. This starts the cycle in the new state.

There are also inhibitory back connections from state to state. The connections are one neuron to one neuron and the weight is -10. An example of this process is shown next.

5.2 Implementation of FSA 1

The neural implementation of the FSA model in figure 5 is described in this section. The 'A choice-free version' recognises simple noun phrases like 'a big cat', or 'the big black book'.

There are five states in this model, S_0, S_1, S_2, e_1, and e_2 with S_0 being the start state and S_2 the final state (goal). C_1, C_2 and C_3 are the inputs; C_1 is either 'a' or 'the'; C_2

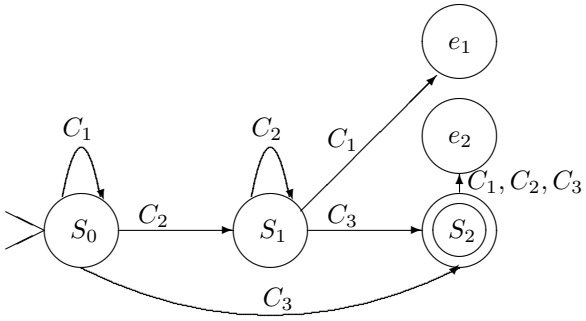

Figure 5: The FSA model-1, 'A choice-free version'.

is 'big', 'fat' or 'black'; and C_3 is 'cat', 'dog' or 'book'. Each state is represented by a persistent net and each of the 8 inputs is represented by an instant net.

Figure 6 shows the coarse connections. Solid arrows are the 'excitatory connections', which show an active state or input spreading energy to another state following the arrow; dashed arrows are the 'inhibitory connections', which show an active state switching off another state. The inputs are 'a', 'the', 'big', 'fat', 'black', 'cat', 'dog' and 'book'. The error states, e_1 and e_2, show an error message if the input reading cannot be accepted by the FSA model. The implementation has been tested on several different sentences. The correct test sentences are 'cat', 'fat dog', 'a big cat', 'the big black book' and 'a big fat black cat'; the incorrect test sentences are 'a black fat the cat', 'big fat the book', 'a the book' and 'book big'.

When the input reading is 'the big black book', the firing procedure of the FSA model is as below:

1. At the beginning, the S_0 persistent net is activated. While there is no input, S_0 continues to fire.
2. When the input is 'the', the FSA stays in S_0.
3. When the input is 'big', S_1 starts to fire due to excitation from S_0 and the 'big' instant net. At the same time, S_0 is shut down by S_1.
4. When the input is 'black', S_1 does not change and keeps firing.
5. When the input is 'book', S_2 starts to fire because of S_1 and the input 'book'; S_1 is shut down by S_2.

When the input is the incorrect sentence 'a black fat the cat', e_1 is the final state. After input of 'a', 'black' and 'fat', the state is S_1; the input of 'the', causes the e_1 net to fire because both of S_1 and 'the' spread energy to it. The test results show that it recognizes all the sentences correctly.

5.3 Implementation of FSA 2

The keen observer will note that there are two types of problems that the above method does not handle. The first is

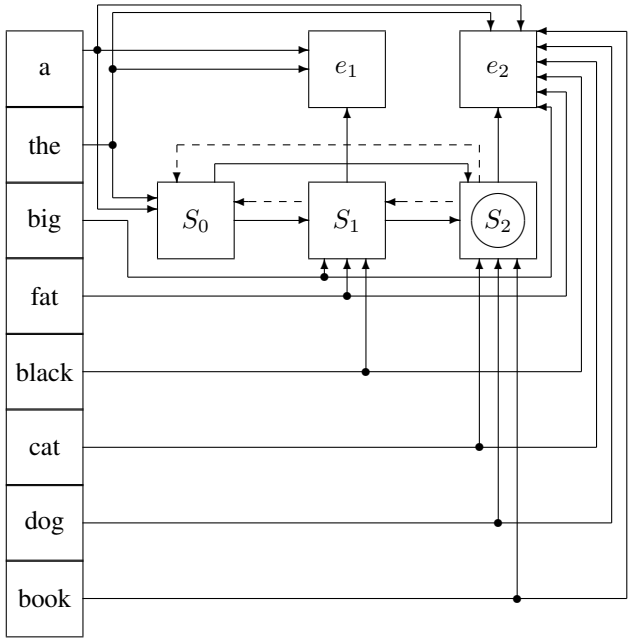

Figure 6: the detail of connections in FSA model-1.

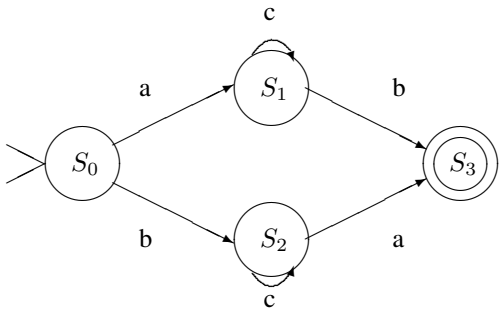

Figure 7: the FSA model-2, which works for a-c-b or b-c-a; dose not work for a-c-a or b-c-b.

the backward transition, an FSA with a transition from one state S_m to another S_n and another transition back from S_n to S_m. They both need to excite and inhibit each other. The second problem is the exclusive-or problem. An example of the exclusive-or problem is shown in figure 7. The problem is that the final state can be reached via two paths. For example, if the system is in S_1, an input of 'a' causes it to incorrectly transit to S_3. In order to overcome this problem, two resistant nets are used to delay spreading energy to S_3 from condition 'a' or 'b'.

The connections to and from the resistant net are again all one to one. The weights to the resistant net are the same as those to the persistent net, from predecessor state 1.5, from input 2.5, and from the successor state -10. The weight from the resistant net to the successor state are 3.5 enabling an automatic transition in 4 cycles. The weight from resistant net to the predecessor state is -10.

Figure 8 shows the coarse connections between nets. The solid arrows are 'excitatory connections', and the dashed arrows are the 'inhibitory connections'. The inputs are 'a', 'b' and 'c'; S_0 is the start state, S_3 is the final state, and the

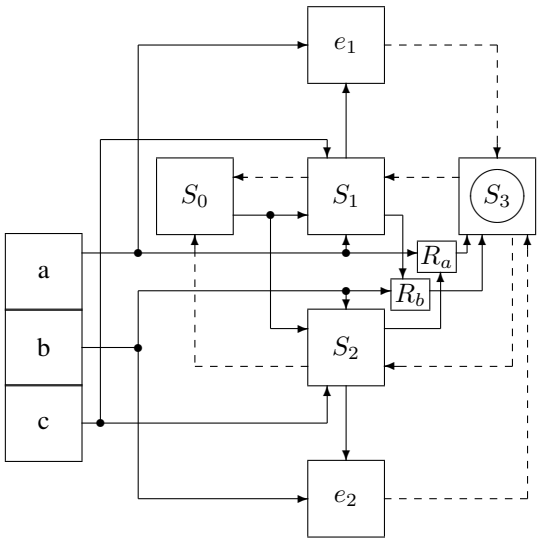

Figure 8: the detail of connections of FSA model-2.

e_1 and e_2 are the error states. R_a and R_b are the resistant nets.

For example, the input is 'acb'. The start state is S_0; state S_1 will fire because 'a' and S_0 are active; e_1 will not fire because 'a' alone does not activate it. When 'c' is read there is no change of state. When 'b' is presented, R_b starts to fire because of energy from S_1 and 'b'; state S_3 is then activated by energy from R_b alone.

Four main input readings of 'abc', 'aca', 'bca', and 'bcb' were tested, and the test results show that the FSA model-2 accepts the input readings of 'abc' and 'bca', and does not accept input reading of 'aca' and 'bcb'.

In general, each transition can be replace by a resistant net. A transition S_m, σ, S_n would be represented by two persistent nets S_m and S_n and a resistant net R_x. There would be input from σ and S_m to R_x, and R_x would stimulate S_n. S_n would inhibit both R_x, and R_x would inhibit S_m. This solves both the exclusive-or problem, and the backward connection problem.

6 Conclusion

The FSAs have been successfully implemented using nets of fLIF neurons, and the experiments show that the neural implementations recognize the acceptable sentences correctly and note unacceptable sentences. It should be obvious how any FSA can be implemented in this fashion. This shows that a wide range of traditional programs – such as language parsing system, environment control, decision-making and control system – can be readily implemented by simulated neurons. Perhaps more importantly, this shows how a standard programming paradigm, FSAs, can be used to implement process in a neural system. These neural FSAs can then be used as a skeleton to integrate neural components that implement more sophisticated behaviour such as learning. This has been the approach described elsewhere in these proceedings [Huyck, 2008].

Acknowledgements

This work is supported by EPSRC grant EP/D059720 - Natural Language Parsing using Cell Assemblies: Computational Linguistics with Attractor Nets.

References

[Hebb, 1949] Hebb, D. O. (1949). *The Organization of Behavior*. J. Wiley & Sons.

[Hopfield, 1982] Hopfield, J. (1982). Neural nets and physical systems with emergent collective computational abilities. *Proc. of the Nat. Academy of Sciences USA*, 79:2554–8.

[Huyck, 2007] Huyck, C. (2007). Creating hierarchical categories using cell assemblies. *Connection Science*, 19:1:1–24.

[Huyck, 2008] Huyck, C. (2008). CABot1: a videogame agent implemented in fLIF neurons. In *IEEE Systems, Man and Cybernetics Society*.

[Huyck and Fan, 2007] Huyck, C. and Fan, Y. (2007). Parsing with flif neurons. In *IEEE Systems, Man and Cybernetics Society*, pages 35–40.

[Lewis and Papadimitriou, 1981] Lewis, H. and Papadimitriou, C. (1981). *Elements of the theory of computation*. Prentice-Hall, Inc, Englewood Cliffs, New Jersey.

[Maass and Bishop), 2001] Maass, W. and Bishop), C. (2001). *Pulsed Neural Networks*. MIT Press Cambridge MA.

[McCulloch and Pitts, 1943] McCulloch, W. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.