A Constructive Spiking Neural Network for Reinforcement Learning in Autonomous Control

Toby Lightheart, Steven Grainger and Tien-Fu Lu
School of Mechanical Engineering
University of Adelaide, North Terrace, Adelaide, SA 5005
Australia
toby.lightheart@adelaide.edu.au

Abstract

This paper presents a method that draws upon reinforcement learning to perform autonomous learning through the automatic construction of a spiking artificial neural network. Constructive neural networks have been applied previously to state and action-value function approximation but have encountered problems of excessive growth of the network, difficulty generalising across a range of problems and a lack of clarity in the operation of resultant networks. The results presented here demonstrate that rapid learning of the control of an inverted pendulum can be achieved with automatic construction of an efficient spiking neural network with internal reward-value associations. This provides a new approach to reinforcement learning and automatic neural network construction for autonomous learning.

1 Introduction

Artificial neural networks (ANNs), inspired by biological neural networks, have been applied to a wide range of control problems. ANNs have the ability to handle continuous domains through function approximation and a large number of associated learning techniques have been developed. The latest generation of neuron models output "spikes" rather than real-value outputs that represent neuron mean firing rates. Using spiking neuron models to produce spiking neural networks (SNNs) increases biological accuracy and dynamic operational capabilities [Stratton and Wiles, 2007]. Despite these advancements, long standing issues surrounding ANNs persist, including difficulty interpreting internal functions of the network and imprecise methods for selection of the size and structure of the network.

ANNs that grow neurons during training or operation are referred to as *constructive neural networks* (CoNNs). Constructive algorithms and techniques tackle the problem of network size selection and have been applied to classification and function approximation in a similar manner to non-growing ANN variants [Nicoletti and Bertini Jr., 2007]. However, a common drawback of many CoNN techniques is a propensity towards excessive growth and over-fitting. This can be caused by processes designed to minimise error during learning that can be particularly problematic when learning data is noisy, such as is common in real-world applications.

Standard learning algorithms for ANNs and CoNNs are described as either *supervised* (training data has inputs and desired outputs) or *unsupervised* (training data only has inputs) [Haykin, 1994]. For autonomous control purposes training examples might be difficult to produce or simply unavailable. Instead a different machine learning paradigm known as *reinforcement learning* is frequently used.

Reinforcement learning (RL) uses evaluation of states and actions during operation (or after each episode or operation) to perform learning updates for the future selection of successful actions. The fundamental components of RL are; the *policy*, for selecting actions; the reward or *reinforcement function*, used to measure and provide feedback on immediate performance; and the *value function*, that stores the learned expected future rewards from states or actions [Sutton and Barto, 1998]. The primary goal of RL is to accurately learn the state-value or action-value functions, which can then be used to update the policy to choose the most successful actions. As many real-world control problems have a continuous state space (represented by continuous values), storage of value functions is often most easily achieved using function approximation.

ANNs have been widely applied to the function approximation of value functions in RL agents operating in continuous domains. Supervised learning techniques are commonly used for training ANNs for value function approximation with training examples provided from experiential data and reward feedback [Anderson, 1989]. However, few techniques exist that construct or train ANNs utilising components and processes of RL.

This paper presents a novel CoNN that utilises reinforcement learning techniques for online learning and growth. Emphasis has been placed on producing rapid learning in an ANN during operation from high-level reward feedback with minimal prior knowledge or assumptions. These properties are expected to support general application of the algorithm to autonomous control problems with low complexity. Simulated experiments with an inverted pendulum demonstrate the learning capabilities of the novel algorithm.

2 Constructive Neural Networks and Reinforcement Learning

Past research using CoNNs and RL has assessed various combinations of these techniques to perform value function approximation. Additionally, there exists a small number of CoNNs that use combinations of sensor and reward feedback to trigger the production of neurons with the aim of developing functional neuro-controllers.

2.1 Value Function Approximation

Ollington *et al*. [2009] summarise two general methods of applying CoNNs to value function approximation for RL, with the majority of approaches described as either *locally responsive constructive reinforcement learning* or *globally responsive constructive reinforcement learning*. In each case the CoNN is used to produce a function approximation of the value function, the difference lies in the method of storage and identification of the state space.

Locally responsive constructive reinforcement learning methods create neurons that are each associated with specific regions of the state or state-action space. New neurons are produced in response to the detection of any state that is not sufficiently associated with any present neuron. The CoNNs primarily used for this approach are growing self-organising maps [Huemer *et al*., 2010] and radial basis function networks [Bruske *et al*., 1997; Li and Duckett, 2005].

Globally responsive constructive reinforcement learning methods produce neurons that are connected to all inputs, with processing distributed across the network to calculate the action value. The cascade-correlation algorithm uses this approach and has received some attention as a CoNN for performing state-value function estimation [Rivest and Precup, 2003]. Nissen [2007] further investigated the performance of incremental and batch training for cascade-correlation neural networks, showing that action-value functions can be adequately approximated for reinforcement learning of problems such as pole-cart balancing and backgammon.

The use of CoNNs, whether locally or globally responsive, to produce value function approximations does not stray far from early implementations of ANNs for reinforcement learning, such as those described by Touzet [1997]. The primary advantage of CoNNs is that the neural network size need not be constrained prior to operation, allowing the system greater adaptability to unexpected levels of complexity. This strength however can also cause operational problems. Many CoNNs are designed to minimise approximation errors, and as such noise or stochasticity can easily cause production of an excessive or divergent number of neurons, resulting in over-fitting and reduced computational efficiency.

2.2 Reward Feedback-Based Construction

Contrasting with value function approximation methods, reward-feedback based constructive techniques [Liu and Buller, 2005; Huemer *et al*., 2008, 2009] largely neglect the storage of value functions. These methods present new CoNNs using reinforcement feedback as a primary mechanism for neuron growth.

Liu and Buller [2005] developed rules specifying sensor and reward conditions for triggering the growth of neurons and connections, directing a simulated robot to reproduce those conditions. The simulated mobile robot was shown to be able to learn to approach a ball when given rules for neuron construction that would activate upon visually detecting and touching the ball. An additional rule was used to learn obstacle avoidance. While successful for the object approaching task, it is not apparent if this approach is capable of learning without specific descriptive neuron construction rules being supplied. Furthermore, the results did not indicate the size of networks produced through this learning system.

A generalised approach to constructing neurons with reward feedback was developed in early work by Huemer *et al*. [2008, 2009]. This CoNN would initially produce random connections between the input and output neuron layers of a spiking neural network to perform action exploration. Positive and negative reinforcement feedback was attributed to each neuron and stored as “reward potential.” New neurons were produced when the reward potential of a neuron exceeded its threshold. The new neuron would replicate the input connections responsible for producing the reward potential and create an output connection to the old neuron. This output connection would encourage or discourage future activation depending on whether the reward potential was positive or negative.

The storage of reward within individual neurons for direct use in triggering the constructive process, while a novel development, had some shortcomings in implementation. Reward modulated synaptic weight adaptation [Florian, 2007] and random neuron connection growth included as online learning mechanisms would prevent new neurons from behaving consistently. Also a neuron produced in response to positive reward would typically produce further positive reward, resulting in a positive feedback loop that could lead to explosive growth of the CoNN. Rules for limiting neuron production, such as preventing new neurons from immediately producing further neurons and tests to prevent the production of redundant neurons, were used to combat excessive growth. However these added complexity to the system and were not thoroughly investigated for their effectiveness.

Results from reward feedback-based constructive methods have suggested that a CoNN can be used directly as a policy while making use of a reinforcement function to create neurons and guide learning. However, the reviewed methods did not adequately address issues including excessive neuron growth or generality of application to autonomous learning for control.

3 Network Construction with Neuron Associated Reward-Values

The approach proposed in this paper uses principles from reinforcement learning to induce the automatic production of neurons that are directly responsible for the actions of the agent. Rather than explicitly learn a value function, reward feedback is associated with individual neurons and updated for learning the appropriate selection of actions. Significant features of the network include the use of a simple spiking neuron model, the association of specific reward-values with “action neurons” and the construction of neurons for both storing states and as a result of action exploration.

3.1 Neuron Model and Network Architecture

The function of any constructive algorithm for a neural network is dependent on the neuron model used and the network structure that is produced. Among the most common models of spiking neurons is the leaky integrate-and-fire (I&F) neuron. While incapable of certain firing modes that have been observed in biological neurons [Stratton and Wiles, 2007], leaky I&F neurons are suitable for real-time control due to their low computational requirements. This neuron model is also easy to implement in CoNNs due to its simplicity and homogeneity in operation and design parameters.

The basic operation of a leaky I&F neuron is, as the name suggests, "integrating" incoming signals until the neuron "potential" exceeds a threshold value, upon which the neuron "fires" a spike. To further mimic the behaviour of a biological neuron, the potential is reset (to zero) after a spike and the potential decays or "leaks" over time. In simulation this process must be performed in discrete time, using update rules such as Eqs. 1 and 2.

$$P_j(t) = \sum_{i=1}^{N} w_{ij}.y_i(t) + k.P_j(t-1) \tag{1}$$

$$y_j(t) = \begin{cases} 1 & \text{, if } P_j(t) > \Theta_j \\ 0 & \text{, otherwise} \end{cases} \tag{2}$$

Where $P_j(t)$ and $P_j(t–1)$ is the potential of neuron j at time step t and $(t–1)$ respectively, w_{ij} is the weight of the input connection from neuron i to j, $y_i(t)$ is the output of neuron i, k is a constant between 0 and 1 representing the leak of potential, and Θ_j is the potential threshold of neuron j. Subsequent to the firing of neuron j, the potential $P_j(t)$ is reset to zero prior to the next time step.

The rate at which the neural network is updated is a factor in operation, as it is typical of a neuron to require a rapid succession of input spikes to cause an output spike. To reduce the need for synchronisation within the network, the time-steps used for simulation are coarse, related to the time constant of the control problem rather than a biological system. Any spikes within the time-step are treated as though they were simultaneous.

The architecture of the constructed neural network follows the layout of a basic three layer multi-layer perceptron, with an input layer, one middle or "action" layer and an output layer. States are detected by neurons in the action layer through connections with spiking input neurons. The neurons in the network are sparsely connected, thus emphasis is on the number of connections and the arrangement rather than connection weights. Connections are therefore directly representative of the state and action with which the neuron is associated (Fig. 1). Only excitatory forward connections exist in the network, inhibiting connections are not explicitly present.

The input layer and output layer neurons are defined prior to operation, designed according to the available state parameters or sensor inputs and the available actions or actuator outputs respectively. This task may be somewhat different to the usual process of normalising inputs and outputs as done for real-valued ANNs. This is due to the use of spiking neurons, allowing input neurons to be designed to spike for specific regions of a sensors input (e.g. when the sensor input is negative or positive) or for increments in sensor value (giving an indication of the sensor signal rate-of-change).

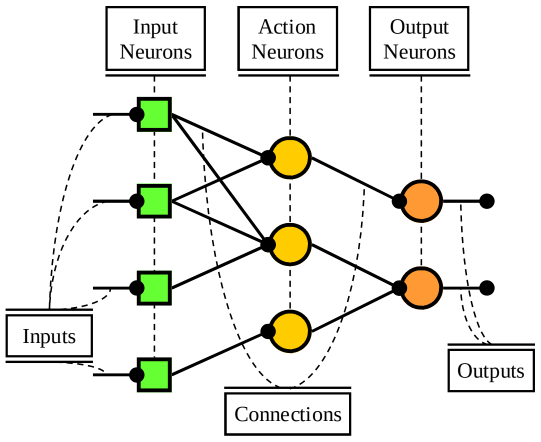

Figure 1: A sparsely connected artificial neural network. Neurons only connect to a selection of neurons in the following layer.

The use of spiking neurons within the network is an important factor within the proposed CoNN. Since different values are expressed by changes in the timing or rate of neuron spikes rather than minute changes in a real-value, states cannot be easily identified by average rates of activity. Instead the dynamic nature of network signals produce rapid changes in active neurons, achieving gradation in control with network outputs comparable to pulse-width modulated signals, relieving the need for fine signal measurements.

3.2 Implementation of Reinforcement Learning

The primary elements of RL (the policy, reinforcement function and value function) are incorporated into the proposed CoNN. The policy is enacted by the neural network that is produced by the constructive algorithm. A reinforcement function is defined prior to operation, and provides non-descriptive or high-level feedback, i.e. only success or failure conditions. The value function is not explicitly approximated, instead the value of actions is represented by reward-values associated with each of the action neurons and updated as they operate.

Each individual neuron that is created is associated with an action by its connections to output neurons. By controlling the activation of these neurons it is possible to assign each action neuron credit for reward received. This is stored as a reward-value parameter associated to each action neuron, in a similar manner to the neuron potential. The reward-value is calculated using local parameters (parameters directly associated with the neuron) and global data in the form of reinforcement feedback.

The reward-value is treated in a similar manner to the action-value, storing a time-weighted average of rewards received after an action is selected, thereby giving an indication of the effectiveness of performance of the action neuron. When a group of action neurons fire the individual reward-values indicates the best performing neuron of that group. This neuron is selected to be the only neuron that has an effect on output neurons (produce an action) in the time-step it is selected. In this way, the reward-value associated with each neuron is used directly to determine its selection. All neurons that are not selected to produce an action may be considered as "inhibited" (Fig. 2). Updates in the reward-values immediately affect the selection of actions (the policy).

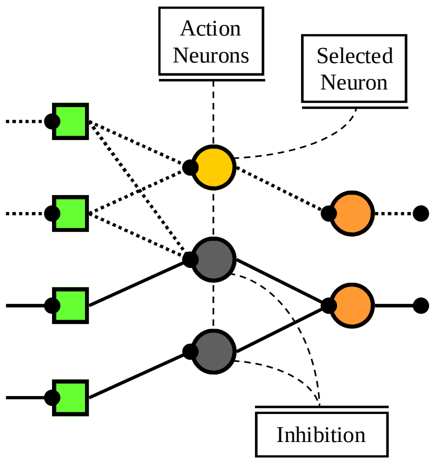

Figure 2: A neural network operating with action selection. Neurons that are not selected are inhibited (dark), only the selected neuron transmits spikes (dotted lines) to the outputs.

With the selection of an individual neuron it is possible to attribute a portion of the responsibility for reward received subsequent to its activation to that neuron. As reward signals continue to be received in later time-steps and during the subsequent selection of other action neurons, the update of the reward-value of a neuron continues at a discounted rate. This process is elaborated by Eqs. 3 and 4.

$$\Pi_j(t) = \Pi_j(t-1) + \alpha_j(t).[r(t) - \Pi_j(t-1)] \qquad (3)$$

$$\alpha_j(t) = \begin{cases} a & \text{, if neuron } j \text{ is selected} \\ \gamma.\alpha_j(t-1) & \text{, otherwise} \end{cases} \qquad (4)$$

Where $\Pi_j(t)$ and $\Pi_j(t–1)$ are the stored reward-values at time-steps t and $(t–1)$ respectively, $\alpha_j(t)$ and $\alpha_j(t–1)$ are the learning rate at time-steps t and $(t–1)$ respectively, $r(t)$ is the global reward, a is a constant for the maximum learning rate, and γ is a discount factor for rewards occurring subsequent to selection of the action neuron.

Action exploration is an important element of many RL techniques used to search for rewarding actions and avoid sub-optimal action choices. In the proposed algorithm action exploration is implemented using the ε-greedy method, i.e. random actions are selected with a probability of ε (a number between 0 and 1) at each time-step [Sutton and Barto, 1998]. As the CoNN does not have prior knowledge stored of possible actions and states it must discover them, therefore action exploration can result in the creation of new action neurons for discovered state-action mappings.

3.3 Neural Network Construction

Initially the neural network starts with only input and output neurons and has no internal connections. This empty ANN has no stored state-action mappings. As a constructive neural network, neurons are produced during operation and are placed between the input and output layers. The constructive algorithm has two processes that are capable of producing neurons: one operates in response to the detection of a new input pattern (state) and the other for exploration of a new output pattern (action).

As the system uses I&F spiking neurons the discrete occurrence of output spikes is used to measure input activation. As the CoNN commences operation input neurons will begin to spike. In each time-step where an unstored pattern of input spikes appear, the production of a new neuron is triggered. This new neuron is created with connections to all input neurons that spiked in the unique combination. This process is limited to input spike patterns that have not already been stored, thus preventing redundancy and excessive growth of the network. Furthermore, the discrete nature of spiking inputs means there is an inherent limit to the number of unique input-patterns that are possible. For each neuron created, connections to output neurons are simultaneously constructed to produce an action, thereby storing a single state-action mapping. The algorithm for neuron creation to store new input states is summarised in Fig. 3.

```
For all input neurons
    Update neuron potential
    If the potential threshold has been exceeded
        Produce an output spike
    End if
End for
For all action neurons
    Compare input neuron spike pattern with action
        neuron input connections
End for
If no action neuron has matching input connections
    Create a new action neuron
    Create connections to all input neurons that
        spiked
    Create random output neuron connections
    Select the new action neuron
End if
```

Figure 3: Algorithm for the creation of action neurons in response to unstored input neuron spike patterns.

Note that this constructive process is essentially a form of unsupervised learning conceptually similar to Hebbian learning, with neurons and connections created that "fire together." The algorithm is also related to locally responsive constructive approaches, as the CoNN creates neurons and input connections in response to the detection of unique inputs represented by the pattern of activation of input neurons.

The mechanism that produces neurons in response to input spike patterns will only create one action neuron for each state. As multiple actions may be possible within each of these states, a mechanism has also been developed for action exploration that can create further action neurons. In this design of the CoNN, action selection is performed using the ε-greedy method, thus random actions are intermittently selected to perform action exploration.

After the input pattern has been detected the action exploration algorithm selects an output connection pattern at random. The algorithm then searches the active action neurons for this output neuron connection pattern. If the combination is found, that neuron is selected regardless of the reward-value associated with the neuron. If the combination cannot be found a new neuron is created for the current input state and created with connections for the action to be explored. The creation of new neurons for each action allows separation of responsibility for actions and the rewards received, as well as allowing action neurons to be selected individually. The algorithm for neuron creation resulting from action exploration is summarised in Fig. 4.

```
Generate a random output connection pattern (action)
**For** all active action neurons
    Compare output connection pattern to the
        randomly generated pattern
    **If** action neuron has matching output connections
        Select that neuron
    **End if**
**End for**
**If** no action neuron has matching output connections
    Create a new action neuron
    Create connections to all input neurons that
        spiked
    Create the randomly generated output neuron
        connections
    Select the new action neuron
**End if**
```

Figure 4: Algorithm for the creation of action neurons resulting from action exploration.

4 Inverted Pendulum Simulation

To demonstrate the learning capability of the proposed constructive algorithm, a simulation of a standard inverted pendulum balancing problem was used. Brownlee [2005] provides a review of common practices in the simulation of an inverted pendulum, or “cart-pole balancing,” as a benchmark control problem. The same parameters and equations of motion were used as those specified by Brownlee, with friction ignored and a “bang-bang” force control value of ±10N. At the start of each simulation run the initial direction of the control force was chosen at random, as were the starting position of the cart and angle of the pendulum.

4.1 Inverted Pendulum-CoNN Interface and Parameters

To interface the inverted pendulum simulation with the constructive neural network it was necessary to define input neuron spike-encoding transforms and output neurons. The input neurons were responsible for providing the constructive neural network with system information regarding the angle and angular velocity of the pendulum as well as the position and velocity of the cart. Each parameter was encoded using a pair of neurons, representing the positive magnitude and the negative magnitude of their associated parameters separately. This gave a total of eight input neurons for the inverted pendulum simulation.

To achieve state parameter encoding within a spiking neuron paradigm, neurons responsible for pendulum angle and cart position would spike with frequency proportional to the associated value (pendulum angle, θ, or cart position, x). This was achieved by increasing the potential of the input neuron at each time-step proportional to the associated value (Eqs. 5 and 6). Neurons associated with velocities were encoded to spike in response to increments in the value of the cart position, x, and pendulum angle, θ, calculated using velocity values as shown in Eqs. 7 and 8.

$$P_1(t) = \begin{cases} P_1(t-1) + a.x(t) & \text{, for } x > 0 \\ 0 & \text{, otherwise} \end{cases} \tag{5}$$

$$P_2(t) = \begin{cases} P_2(t-1) - a.x(t) & \text{, for } x < 0 \\ 0 & \text{, otherwise} \end{cases} \tag{6}$$

$$P_5(t) = \begin{cases} P_5(t-1) + b.\tau.\dot{x}(t) & \text{, for } \dot{x} > 0 \\ 0 & \text{, otherwise} \end{cases} \tag{7}$$

$$P_6(t) = \begin{cases} P_6(t-1) - b.\tau.\dot{x}(t) & \text{, for } \dot{x} < 0 \\ 0 & \text{, otherwise} \end{cases} \tag{8}$$

Where $P_1(t)$, $P_2(t)$, $P_5(t)$ and $P_6(t)$ are the activation potentials of input neurons 1, 2, 5 and 6 respectively, $x(t)$ and $\dot{x}(t)$ are the cart position and cart velocity respectively, and τ is the time-step length (0.02s, as is typical of inverted pendulum simulations). Parameters a and b are tuned input sensitivity factors for detecting cart position (a = 1.5) and cart velocity (b = 8) respectively. These same functions were used for calculating the activation potentials of pendulum angle and angular velocity input neurons (neurons 3, 4, 7 and 8) with separately tuned sensitivity factors (a = 20, b = 100).

Similar to real-valued neurons with normalised outputs, this implementation of spiking neurons will saturate at an activation level. This forms a trade-off with the sensitivity of the neuron to small changes in the measured variable. Mechanisms for adapting the sensitivity automatically would simplify this design step and would facilitate greater system adaptivity.

Two output neurons were used, with a neuron associated with the positive and negative directions in the one controllable degree of freedom (force applied left or right to the cart). Using bang-bang control the output neurons were only required to spike once to adjust the direction of the applied force to the cart. Smooth movements were achievable by the network through rapid switching in the application of the force between positive and negative directions. To simplify the operation of the CoNN, action exploration was restricted to producing one output neuron connection. Preventing multiple output connections and zero connection cases reduced the action search space to actions allowed for bang-bang control.

Maintaining the emphasis on learning with minimal feedback, the reinforcement signal generated for the constructive neural network was +1 at all times except in the event of failure when –1 was received for that one time-step. After each failure of the inverted pendulum the system was reset, while the constructive neural network was maintained with only the learning rate of neuron associated reward-values reset to zero. This is only necessary for episodic tasks to prevent the neurons from continuing to update across a reset of the system and taking credit for reinforcement signals that those neurons played no part in obtaining.

4.2 Simulation Results

Simulated trials of the inverted pendulum with the proposed reinforcement learning CoNN were performed. The results presented here give an overview of the performance with averages and examples of individual results indicative of achieved learning. Of specific interest within the results are the rate of learning, the convergence and reliability of operation, and the size and topology of artificial neural networks produced.

The first plot (Fig. 5) shows time of successful inverted pendulum balancing for each successive trial averaged across ten simulated learning runs. Note the upwards trend of the average confirming that learning takes place. The average result of the learning system exceeded balance times of 10 seconds in fewer than 50 trials of balancing the pendulum.

The second plot (Fig. 6) shows the average number of neurons that were produced over the course of ten learning runs. Note that the CoNN did not grow uncontrollably or diverge, with the average just exceeding 120 neurons including input and output neurons.

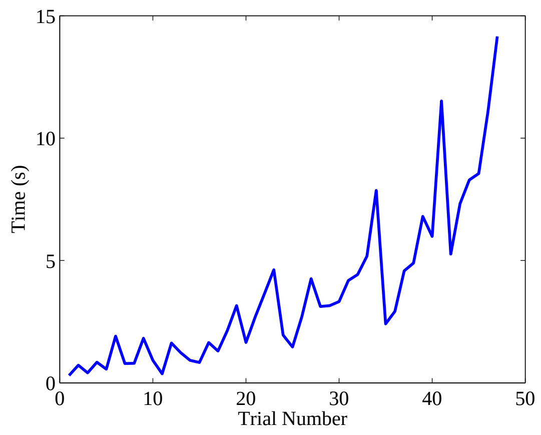

Figure 5: Average time of success with inverted pendulum control. Average over ten learning simulations at each trial of inverted pendulum control.

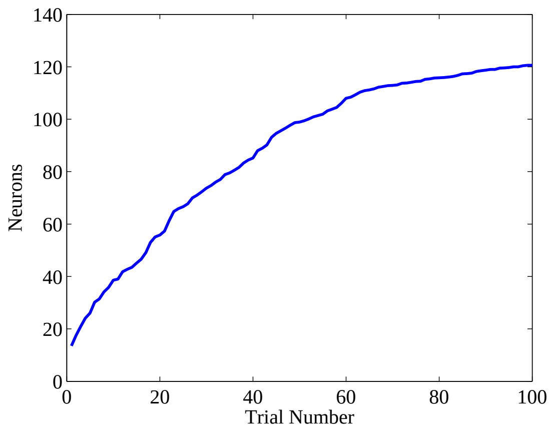

Figure 6: Average number of neurons. Average over ten learning simulations at each trial of inverted pendulum control.

Examples of neurons produced, including their connections and stored reward-values, have been extracted and interpreted (Fig. 7). Pendulum angle and angular velocity (θ and $\dot{\theta}(t)$) were positive in the clockwise direction and negative in the counter-clockwise direction. Cart position and velocity ($x(t)$ and $\dot{x}(t)$) were positive towards the right and negative towards the left. Force (F) was positive when applied towards the right and negative when applied towards the left.

By comparing their reward-values it is possible to determine trends in the control outputs that would be generated for those given states. The example neurons 11 and 17, and 29 and 117 were competing pairs. Neuron 17 would get selected in preference to neuron 11, therefore in the state where the inverted pendulum is rotating counter-clockwise the force exerted on the cart will be towards the left. The second example shows that when the pendulum is rotating clockwise and simultaneously moving to the left, that the favoured action is to exert force towards the right. These are intuitively sound outcomes.

Neuron ID: 11	**Neuron ID:** 17
Input: $-\dot{\theta}(t)$	**Input:** $-\dot{\theta}(t)$
Output: $+F$	**Output:** $-F$
Reward-Value: 0.979	**Reward-Value:** 0.991
Neuron ID: 29	**Neuron ID:** 117
Inputs: $+\dot{\theta}(t), -\dot{x}(t)$	**Inputs:** $+\dot{\theta}(t), -\dot{x}(t)$
Output: $-F$	**Output:** $+F$
Reward-Value: 0.975	**Reward-Value:** 0.980

Figure 7: Examples of the neuron parameters values created for balancing an inverted pendulum.

5 Discussion and Future Work

The proposed algorithm aimed to produce rapid learning by constructing an ANN using reinforcement learning. This was achieved with the use of high-level rewards and a limited amount of prior knowledge in parameter selection and system design. The performance of the algorithm in the inverted pendulum balancing task has shown that the proposed algorithm is capable of improving performance during operation.

In the simulated experiments presented in this paper autonomously developed neuro-controllers demonstrated learning of state-action mappings and the attribution of reward-values for action selection. The learned action selection produced rapid oscillation of network outputs suitable for bang-bang control and was capable of producing smooth control and successful balancing of the inverted pendulum. The performance of the proposed learning algorithm can be compared with other learning systems for inverted pendulum learning and more generally with other CoNNs used for RL.

Anderson [1989] demonstrated inverted pendulum balancing with similar emphasis on minimal use of prior knowledge and learning bias. The comparison is apt due to the use of ANNs, though non-growing and with back-propagation used for offline weight adjustment. The algorithm successfully learned inverted pendulum balancing however was recorded taking in excess of 5000 trials to achieve similar success times.

While the proposed algorithm has demonstrated faster learning of basic balancing, the system was also prone to less consistent long-term operation. Difficulties were encountered in associating cart position feedback with the comparatively rare track-limit failure. Anderson [1989] noted similar unreliable learning in avoiding the boundaries of the cart track, but was remedied through use of a wide range of random starting positions.

The inconsistency displayed by the learning system presented in this paper is explicable and effective improvements are plausible. The input sensitivity parameters were tuned but not optimised, this was especially evident for learning the failure mode of reaching the cart track limit. Additional potential causes of failure came from perturbations introduced internally from random action exploration, as well as unfortunate combinations of starting position, pendulum angle and the initialised direction of the control force. This was evident when the controller would reach the maximum allowed simulated balance time of 60 seconds, but then immediately following this show difficulty in obtaining a balanced position from the random starting conditions. In the event of one of these possible failures, the resultant reward-value adjustments could cause undesirable changes in the policy for operation. This caused bouts of poor performance until learning updates corrected the erroneous adjustments.

CoNNs have previously been applied to learning inverted pendulum balancing, with the application predominantly being value function approximation. Nissen [2007] presented results from the application of a cascade correlation algorithm, combined with a cache for offline training, to the cart-pole problem. With dissimilar reward feedback (additional reward for keeping the cart and pole within a small margin of the origin) and trial conditions (trials ended after 300 time-steps), balancing performance is difficult to compare. However the performance criterion of greatest interest is the growth of neurons. While certain parameter combinations did not show clear converging trends in growth, given a large enough training cache, the network produced a successful learning algorithm that in one instance balanced the pendulum using one hidden neuron. However in simulated experiments involving other learning problems, such as the mountain-car, the same parameter combination caused significant divergence in network growth.

For further comparison, the CoNN for reinforcement learning developed by Huemer *et al*. [2010] was also evaluated using an inverted pendulum simulation. While the learning algorithm shows a learning trend with the increase of balance times, improvements appear to be constrained by episodes of erratic operation. Results presented indicate that the averaged time of pendulum balancing did not exceed four seconds and similar inconsistent performance. This suggests issues in learning stability and possibly challenges learning the infrequent track-limit failure. Additionally from a CoNN standpoint a large number of neurons and connections were produced (approximately 1500 and ten thousand respectively), although whether there were issues with convergence was not clear.

At this point it should be raised that the purpose of developing CoNNs for RL is at present not to achieve a new standard in inverted pendulum balancing. Rather, improving the generality in application and the scalability of RL are among potential outcomes. To this end, determining the strengths and weaknesses of constructive algorithms and analysing performance in specific applications may give insight for design improvements. With this in mind, intermediate goals for systems combining CoNNs and RL, such as preventing excessive neuron growth, are pragmatic.

The algorithm presented produces neural networks that will be limited in their size relative to the number of input and output neurons regardless of the application. This is largely due to the use of spiking neurons as inputs dynamically representing a range of input and output values, instead of using precise real values. The emphasis has therefore been shifted to the dynamic switching and modulation of neuron activation to perform control.

The example neuron connections and reward values extracted from the CoNN (Fig. 7), can be understood with little effort, and present the potential to interface with other real-time diagnostic programs and control systems. In this manner the process of selecting neurons may be controlled using mechanisms external to the network, allowing for a direct interface between hybrid controllers and developed CoNNs. The ability to produce neuro-controllers that may be externally interfaced through direct coupling with internal components is not common within ANNs and potentially has further benefits.

It should be noted that similar to the reviewed CoNN and RL algorithms, parameters that have a significant bearing on performance must be set prior to operation. In the presented algorithm this is prevalent in the input transformation functions that are defined for input neuron activation prior to operation, and the reward-value learning rate and learning rate decay. This requires some knowledge (or assumptions) regarding the required sensitivity and range of the state-space or sensor operation, and effective learning given a speed-stability trade-off. Inclusion of effective methods for parameter adjustment during operation would improve the online adaptivity and the performance of the learning algorithm.

As a new conceptual development many avenues for future work exist. Investigation of the generality and scalability of the algorithm within the target domain of online learning during real-time control will provide more insight into the applicability of the algorithm as a machine learning tool. Furthermore, algorithm performance in noisy real-world conditions as well as for continuous online learning with response to environmental changes will be evaluated. Applicability of the algorithm to learning mobile robot behaviours is a desired outcome and will likewise be the subject of further study.

Additional algorithm developments could include; mechanisms for adapting input neurons; implementation of alternative action exploration algorithms for neuron growth; extraction of the best performing neurons to form smaller network topologies; implementation within, or design of, a behaviour-based control architecture that may integrate multiple CoNNs; and exploration of learning capabilities in the learning of multiple behaviours in such a control architecture.

6 Conclusion

Constructive neural networks have had limited development both for application in reinforcement learning or as a beneficiary of reinforcement learning techniques. The proposed novel algorithm utilises principle elements of RL for the automatic construction and operation of an ANN. Spiking neurons are produced to be individually responsible for state-action mappings and reward-values received subsequent to their activation are stored. Action selection is performed using these neuron associated reward-values by inhibiting of all but the best performing neurons.

This CoNN has been demonstrated through experimental simulations as an alternative to traditional methods of using CoNNs for RL value function approximation. The proposed algorithm overcomes problems with excessive network growth by using spiking neurons as inputs with control performed as a dynamic process related to spike frequencies. The initial requirements of the learning system do not exceed those typical of RL (the prior definition of the reinforcement function, learning rate and the learning rate decay) and ANN interfaces (input and output neuron signal transform functions). As such, this learning system is generally applicable to autonomous online learning and real-time control suited to neuro-controllers.

References

[Anderson, 1989] Charles W. Anderson, Learning to control an inverted pendulum using neural networks, *IEEE Control Systems Magazine*, Vol 9(3), pp. 31-37, April 1989.

[Brownlee, 2005] Jason Brownlee, The pole balancing problem: A benchmark control theory problem, Technical Report 7-01, Swinburne University of Technology, July 2005.

[Bruske *et al*., 1997] Jörg Bruske, Ingo Ahrns, and Gerald Sommer, An integrated architecture for learning of reactive behaviors based on dynamic cell structures, *Robotics and Autonomous Systems*, Vol. 22(2), pp. 87-101, November 1997.

[Florian, 2007] Răzvan V. Florian, Reinforcement learning through reward modulation of spike-timing-dependent synaptic plasticity, *Neural Computation*, Vol 19(6), pp. 1468-1502, June 2007.

[Haykin, 1994] Simon Haykin, *Neural Networks: A Comprehensive Foundation*, Maxwell-Macmillan International, New York, 1994.

[Huemer *et al*., 2008] Andreas Huemer, Mario A. Góngora and David A. Elizondo, Evolving a neural network using dyadic connections, *IEEE International Joint Conference on Neural Networks*, pp. 1019-1025, June 2008.

[Huemer *et al*., 2009] Andreas Huemer, David A. Elizondo and Mario A. Góngora, A constructive neural network for evolving a machine controller in real-time, *Constructive Neural Networks*, SCI 258, pp. 225-242, Springer-Verlag, 2009.

[Huemer *et al*., 2010] Andreas Huemer, Mario A. Góngora and David A. Elizondo, A robust reinforcement learning based self constructing neural network, *International Joint Conference on Neural Networks*, pp. 426-432, July 2010.

[Li and Duckett, 2005] Jun Li and Tom Duckett, Q-Learning with a growing RBF network for behaviour learning in mobile robotics, *Proc. IASTED Conference on Robotics and Automation*, Vol. 498, October-November 2005.

[Liu and Buller, 2005] Juan Liu and Andrzej Buller, Self-development of motor abilities resulting from the growth of a neural network reinforced by pleasure and tensions, *Development and Learning, The 4th International Conference on*, pp. 121-125, July 2005.

[Nicoletti and Bertini Jr., 2007] Maria de Carmo Nicoletti and João R. Bertini Jr., An empirical evaluation of constructive neural networks algorithms in classification tasks, *Int. J. Innovative Computing and Applications*, Vol. 1(1), pp. 2-13, 2007.

[Nissen, 2007] Steffen Nissen, Large scale reinforcement learning using *Q*-SARSA(λ) and cascading neural networks, Masters Thesis, University of Copenhagen, October 2007.

[Ollington *et al*., 2009] Robert Ollington, Peter Vamplew and John Swanson, Incorporating expert advice into reinforcement learning using constructive neural networks, *Constructive Neural Networks*, SCI 258, pp. 207-224, Springer-Verlag, 2009.

[Rivest and Precup, 2003] François Rivest and Doina Precup, Combining TD-learning with cascade-correlation networks, *20th Int. Conf. on Machine Learning*, *Proc. of*, Washington DC, pp. 632-639 August 2003.

[Stratton and Wiles, 2007] Peter Stratton and Janet Wiles, Why spiking neurons?, Technical Report TS-2007001, University of Queensland, 2007.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, Mass., 1998.

[Touzet, 1997] Claude F. Touzet, Neural reinforcement learning for behaviour synthesis, *Robotics and Autonomous Systems*, Vol. 22(3-4), pp. 251-281, December 1997.