A Constructive Neural Network for Evolving a Machine Controller in Real-Time

Andreas Huemer, David Elizondo, and Mario Gongora

Abstract. A novel method is presented to allow a machine controller to evolve while the machine is acting in its environment. The method uses a single spiking neural network with a minimum number of neurons and no initial connections. New connections and neurons are grown by evaluating reward values which can represent either the internal state of the machine or the rating of its task performance. This way the topology and the level of connectivity of the network are kept to a minimum. The method will be applied to a controller for an autonomous mobile robot.

Keywords: Constructive Neural Network, Spiking Neural Network, Reinforcement Learning, Growing Machine Controller.

1 Introduction

Typically constructive neural networks are used to solve classification problems. It has been shown that using this type of network results in less computation requirement, smaller topologies, faster learning and better classification [5, 9]. Additionally, [4] shows that certain constructive neural networks can always be evolved to a stage in which it can classify 100% of the training data correctly.

For machine controllers, classification is only a secondary issue, but still an important one, as will be discussed later. The main machine task is to select a suitable action in a certain situation. It need not be the best possible action but must certainly

Andreas Huemer
Institute Of Creative Technologies, De Montfort University, Leicester, UK
e-mail: ahuemer@dmu.ac.uk

David Elizondo
Centre for Computational Intelligence, De Montfort University, Leicester, UK
e-mail: elizondo@dmu.ac.uk

Mario Gongora
Centre for Computational Intelligence, De Montfort University, Leicester, UK
e-mail: mgongora@dmu.ac.uk

L. Franco et al. (Eds.): Constructive Neural Networks, SCI 258, pp. 225–242.
springerlink.com

be suitable. Much more important is that the action is selected in the required time: machines, especially mobile robots, must act in real-time. In [10] real-time systems are discussed and a definition is provided.

One approach to enable a machine acting in real-time is to create the controller in advance and then use it on the machine without further intervention. This is the traditional way, but it works only for simple applications or it needs much development time for more complex applications.

As machines are required to fulfil increasingly complex tasks, people have looked for various possible solutions to this problem. On the one hand, methods were sought to speed up the development of the controller, which is then used on the machine. The other idea was to improve the controller when already in use.

Both approaches are useful and can also be combined, and have resulted in methods inspired by nature. Evolutionary algorithms are inspired by the evolution of life an can be used to improve the controller. For example [23] presents a very effective method for evolving neural networks that can be used to control robots.

However, the power of this approach is limited, because it does not work with a single controller but with a whole population of controllers. More controllers require more computational power, which either may not be available or if available, may slow down the embedded computer thereby making this option too expensive. Of course there could also be a population of machines and not only a population of controllers, but the increasing expense in material and energy is obvious in this case.

Alternatively, machines or their controllers could be simulated on a computer, but this only transfers several problems from the robot to the computer and involves the additional problem of it being hard to simulate the complex environments that robots should often act in.

On-line learning, which means adapting the machine controller while it is running, can help with overcoming the problems of evolutionary methods, but it does not substitute them. The remainder of this chapter is dedicated to on-line learning. In Sect. 2 we summarise the history of on-line learning up to the current state-of-the-art methods and we identify remaining problems tackled in this chapter.

Section 3 shows the basic characteristics of a novel constructive machine controller. The crucial issue of how feedback can be defined, how it is fed into the controller and how it is used for basic learning strategies is discussed in Sect. 4. Section 5 shows how the feedback is used for the more sophisticated methods of growing new neurons and connections.

A simulation of a mobile robot was used to test the novel methodology. The robot learns to wander around in a simulated environment avoiding obstacles, such as walls. The results are presented in Sect. 6, and Sect. 7 draws some conclusions and outlines future lines of work.

2 History of Machine Controllers

This section will provide a brief overview of the history of machine controllers. A machine controller is needed as soon as a machine is required to behave differently

in different situations. The controller decides which parts of the machine to adjust (output) in which situation (input). The complexity of the controller increases with the number of different situations that can be sensed and the number of different actions that can be chosen.

We do not cover the history of control technologies (e.g. mechanical or electronic control) here but we discuss control concepts with respect to control methods. Also remote control is not given explicit consideration, because we interpret the remote signals as an input to the actual controller.

2.1 Fixed Behaviour

Early controllers were developed so that they behaved in a strictly predefined way as long as no error occurred. These controllers are still needed and it is hard to imagine why they should not be needed in the future.

However, each decision of the machine is planned and programmed manually and the number of situations that have to be considered grows exponentially:

$$N_I = \prod N_v \tag{1}$$

where the number of all input combinations N_I is the product of all states that have to be considered for the input variables N_v. Because of this, the development time or the number of errors increases dramatically with the number of states that have to be differentiated for an input variable and especially with the number of input variables. This is without even considering the output side.

Consequently, there have been different approaches to tackle this problem. One idea was to adapt the environment in which the machine is situated. For example by situating a mobile robot in a laboratory where the ground is level and there is constant standardized lighting the number of situations the controller has to consider is minimised.

The problem is that mobile robots and other machines are usually not needed in a quasi-perfect environment. Many robots are however required to fulfil increasingly complex tasks in our real and complex world. The good news is that many of those robots need not act in a perfect way but just in a way that is good enough. There is some room for error and it is possible to concentrate on minimising the time required to develop a controller.

Controllers that have a fixed behaviour for regular operations have one important advantage and one important drawback by definition. The advantage is that the machines do not change their behaviour in known situations: their behaviour is predictable. The drawback is that the machines do not change their behaviour in unknown situations either, which may result in intolerable errors. An increasing complexity of the environment increases the probability for intolerable errors, if the time for developing a controller that should work in it is not increased significantly.

2.2 Adaptive Behaviour

This problem has led to research in on-line adaptation processes for machines, especially for mobile robots used in real-world applications. Traditional methods work with controller parts which all have a certain predefined task. Adaptation processes can change the reactions to certain input, but the machine is still limited to the behaviour of preprogrammed modules of different complexity. An interesting example of these adaptation methods was presented in [20].

Increased flexibility can be achieved by keeping the modules as simple as possible and as similar as possible. This way it is not only possible to change the interactions between existing preprogrammed modules but it is also possible to change those modules themselves. Additionally, it is possible to create completely new modules.

Artificial neural networks can achieve a high level of flexibility and therefore are a useful tool for reducing development time to a minimum. For example Alnajjar and Murase [1] have shown that a controller consisting of a spiking neural network (SNN) enables a robot to learn certain tasks successfully.

Spiking neurons, unlike traditional artificial neurons, encode the information they send to other neurons, called postsynaptic neurons, in a sequence of spikes (Boolean signals) instead of a single value. Spiking neurons can have a lot of features that are impossible or at least difficult to implement with traditional neurons, which encode their information in a single value and send them to other neurons at a particular moment. In [12] some interesting features that can be implemented in different types of spiking neural networks are discussed.

Some advantages of spiking neural networks compared to traditional neural networks are:

- Single spikes are not very important to the overall behaviour of the network. This network behaviour is probabilistic and fuzzy and is therefore more robust to errors. For example, if a neuron sends spikes at a slightly higher frequency, it obtains slightly more influence; if a neuron sends a few additional spikes, its influence lasts a little longer; if a spike is sent a little late, it may already be compensated for by the next spike. There are no drastic effects as there would be with wrong single values that transport the whole information.
- Spiking neurons can react faster to certain time-dependent input patterns (e.g. a sound). Those input patterns can be easily implemented in spiking patterns because of their time-dependent nature. If a time-dependent input pattern has to be encoded in a single value, the controller has to wait some time before being able to calculate a statistic value.
- Probably the most interesting advantage regarding reinforcement learning is that feedback assignment is much easier with spiking neural networks. An action is not initiated at a certain moment and then left alone, but rather it is encoded in a continuous spike train. This way the neurons that are responsible for a certain action can still be active when the machine starts to receive feedback for that action. This issue is discussed in more detail in Sect. 4.3.

2.3 Constructive Machine Controllers

For full flexibility, to create completely new behaviour that reacts to completely new situations, the spiking neural network must be constructive. This means that new connections and new neurons need to be created in certain circumstances.

Alternative approaches that try to generate neurons at the beginning and later only change the connections of the network (e.g. "Echo State Networks" [14] and "Liquid State Machines" [19]) have two main disadvantages for adaptive control systems:

- Usually, not all parts of the network are needed for the control task. The unnecessary parts are processed anyway and decrease the performance of the system.
- It cannot be guaranteed that the network can learn all tasks successfully, because its general structure is fixed after its random initialisation.

A very flexible and effective approach for classification problems is the "Growing Neural Gas" algorithm [7, 8], which is an improvement of the "Neural Gas" algorithm [21]. Drawbacks of "Neural Gases" that are similar to the problems mentioned previously with "Echo State Networks" and "Liquid State Machines", could be solved successfully with the "Growing Neural Gas" algorithm.

Examples for constructive neural networks for classification problems without the need for recurrent connections are "Support Vector Machines" [22] and the "Recursive Deterministic Perceptron" [24].

In recent years some ideas for constructive methods have been published for control problems. A typical test application for the adaptive controllers is to enable a two-wheeled mobile robot to learn to run around while avoiding crashing into obstacles.

In [17, 18] a robot controller is presented that creates neurons to map a certain input pattern to an output using "pleasure cells". In [2] a two-level control system has been introduced. A low-level unit contains a number of spiking neural networks which are all trained to perform the robot's task in a local environment. The high-level unit switches between the SNNs depending on the current environment or it can create a new SNN, if it cannot find an SNN for the current environment.

The constructive neural network discussed in this chapter integrates the complete control task into a single neural network. The construction method enables the network to evolve from an initial stage with a minimum number of neurons and no connections. Even basic movements are learnt autonomously (in [18] "random cells" are used for random initial behaviour, in [2] the robot initially goes straight forward). The main goal of the work presented here is to minimise the time for developing a highly adaptive control system by making it as flexible and autonomous as possible.

3 A Constructive Machine Controller

Based on the issues we have discussed in the previous sections we present a constructive neural network that will enable a machine to learn to fulfil its tasks

autonomously. In addition to the main task, the action selection, we also discuss classification. In a machine controller, classification can be used to reduce the size of the network.

A novel method is presented which not only includes the classification task and the action selection task in a single neural network, but it is also capable of defining the classes autonomously and it can grow new neurons and connections to learn to select the correct actions from the controller's experience based on reward values. Positive and negative reward values build the feedback, which can be generated internally, without human intervention. Alternatively it is possible for a human to feed the controller with feedback during runtime acting as a teacher.

The initial neural network consists of a minimum number of neurons and no connections. The designer of the controller need not concern him/herself with the topology of the network. Only the interface from the controller to the machine has to be defined, which reduces the design effort considerably.

The interface includes:

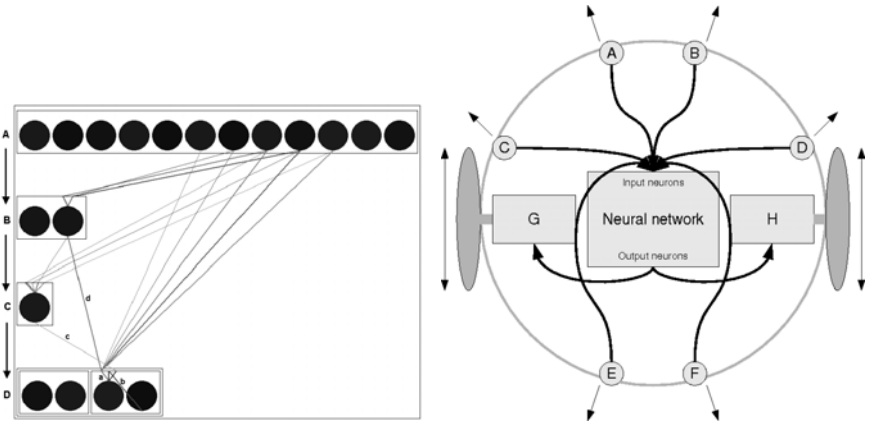
- *Input neurons*, which convert sensory input into values that can be sent to other neurons. For our experiments we used spiking neurons, so the sensory values are converted into spikes by the input neurons.
- *Output neurons*, which convert the output of the neural network into values that can be interpreted by the actuators.
- *A reward function*, which feeds feedback into the machine controller.

3.1 Controller Characteristics

Our "growing" machine controller consists of a layered spiking neural network as illustrated in Fig. 1.a, in which neurons send Boolean signals that transport the information depending on whether or not the presynaptic neuron (the first of two connected neurons) was activated and fired (a spike). The hidden layer "B" contains two neurons that store input combinations. One of them excites dendrite "a" via axon "d", which itself excites an output neuron. The neuron in layer "C" inhibits the same output neuron by axon "c". Also axon "b" inhibits that neuron but implements local inhibition. A neuron is activated when a certain threshold potential is exceeded. The neuron potential is increased by spikes arriving at excitatory connections and decreased by spikes arriving at inhibitory connections. A basic explanation of spiking neural networks can be found in [25].

For a controller system it is evident, that the action selection task needs to be fulfilled. Classification of input signals and recursive classification of classes are important issues, because this reduces the necessary number of neurons and connections.

The use of sparse neural networks has been discussed in [5] and [9]. These models result in fewer computational requirements and better development of the network as well as smaller topologies. The main reason for obtaining smaller topologies when using classification is obvious:



(a) (b)

Fig. 1 (a) Neural network topology after a short simulation period. For clarity not all connections are shown. Layer A consists of the input neurons which are connected to sensors A to F from the robot shown in (b). Layer D contains the motor neurons, connected to G and H in (b).

If there are no specialised neurons that represent a certain class in the subsequent layers in the network, all neurons of this class have to connect to the next layer separately and not with a single connection from the specialised "class neuron". There is no problem if one neuron connects only to one other neuron. In this case no additional connections are required. However, when a neuron connects to a second neuron, only one additional connection is made instead of more connections from all the neurons of a class. Also the total number of neurons is reduced if the neurons represent the possible combinations of neurons in the previous layer, because in a subsequent layer only the class neurons have to be considered.

To achieve an efficient topology along with action selection and classification, in a single neural network, we separate the connections into two parts: artificial dendrites and axons. An axon of the presynaptic neuron is connected to a dendrite of the postsynaptic neuron. Excitatory connections have to be used for operations that are similar to the logical AND and OR operations. For inhibitory connections this separation is not necessary, because they represent an operation similar to the logical NOT operation.

A dendrite has a low threshold potential and is activated when only one presynaptic neuron (or a few neurons) have fired a spike via their axons. All presynaptic neurons are handled equally at this point (logical OR operation) and represent neurons which are combined into one class. An axon weight defines to what degree one presynaptic neuron belongs to the class.

The neuron finally fires only if a certain combination of dendrites has fired (logical AND operation). For this operation, the threshold of the neuron potential, which is modified by the dendrites, is very high. This causes a neuron to select an action for a certain combination of input signals. The dendrite weight defines the importance

of a class in the action selection process. In our model a single neuron can fulfil both tasks, classification and action selection, and it learns both tasks automatically.

In the following we show the computations performed when signals travel from one neuron to another.

Input of a dendrite:

$$I_d = \sum O_{a+} \cdot w_{a+} \tag{2}$$

where I_d is the dendrite input. O_{a+} is the output of an excitatory axon, which is 1 if the presynaptic neuron has fired and 0 otherwise. w_{a+} is the weight of the same excitatory axon, which must be a value between 0 and 1.

Output of a dendrite:

$$O_d = \frac{1}{1 + e^{-b \cdot (I_d - \theta_d)}} \tag{3}$$

where O_d is the dendrite output and I_d its input. θ_d is a threshold value for the dendrite. b is an activation constant and defines the abruptness of activation.

Input of a neuron:

$$I_j = \sum O_d \cdot w_d - \sum O_{a-} \cdot w_{a-} \tag{4}$$

where I_j is the input of the postsynaptic neuron j, O_d is the output of a dendrite, w_d is the weight of this dendrite, O_{a-} is the output of an inhibitory axon and w_{a-} is the weight of this inhibitory axon. Dendrite weights and axon weights are in the range $[0, 1]$ and all dendrite weights add up to 1.

Change of neuron potential:

$$P_j(t+1) = \delta \cdot P_j(t) + I_j \tag{5}$$

where the new neuron potential $P_j(t+1)$ is calculated from the potential of the last time step t, $P_j(t)$, and the current contribution by the neuron input I_j. δ is a constant between 0 and 1 for recovering the resting potential with time (which is 0 in this case).

The postsynaptic neuron is activated when its potential reaches the threshold θ_j and becomes a presynaptic neuron for neurons which its own axons are connected to. After firing, the neuron resets its potential to the resting state. In contrast to similar neuron models that are for example summarised by [15], a refractory period is not implemented here.

4 Feedback and Reward Values

4.1 Calculation of Reward Values

The main challenge for the designer of a machine controller that uses the methods described in this chapter, is to define an appropriate reward function. Positive and negative reward values are fed into the neural network as explained below and are used for all adaptation processes, like basic learning (adaptation of

connection weights), creating new connections and creating new neurons (growing mechanisms).

In our experiments we have used a single global reward value, which represents a positive "rewarding" or negative "aversive" feedback depending on the machine's performance in the task that it has been assigned. The objective is to create a network which maximises positive feedback.

Depending on current measurements like fast movement, crashes, recovering from crashes, or the energy level, the reward value is set from -1 (very bad), to 1 (very good). The challenge is to provide a useful value in all situations. For example, as experiments have shown (see Sect. 6), a reward function that is not capable of providing enough positive feedback may result in a machine malfunction, because despite all of its effort to find a good action, it is not evaluated properly. Also uniform positive feedback may result in a similar situation because of a lack of contrast.

The reward value $\rho(t)$, which is the result of the reward function at time t, has to be back-propagated to all neurons, where it is analysed and used for the adaptation mechanisms. To do so, the neurons of each layer, starting with the output layer and going back to the input layer, calculate their own reward value. The value of the output neurons is equivalent to the global reward value. All other neurons calculate their reward as follows:

$$\rho_i(t) = \frac{\sum \rho_{j+}(t) - \sum \rho_{j-}(t)}{N_+ + N_-} \tag{6}$$

where $\rho_i(t)$ is the reward value of the presynaptic neuron i at time step t. $\rho_{j+}(t)$ is the reward value of a postsynaptic neuron that has an excitatory connection from neuron i, while $j-$ refers to a postsynaptic neuron that has an inhibitory connection from neuron i. N_+ is the number of postsynaptic neurons of the first kind and N_- is the number of the other postsynaptic neurons.

4.2 Adaptation of Connection Weights

First the reward value of a neuron is used to adapt the connection weights. This is done after the basic processes of Sect. 3 for each neuron. All calculations for all neurons are done once at each time step t.

The reward value can be added to a learning rule as an additional factor. Different authors, all of them using different neuron functions and learning functions, have shown that this surprisingly simple method can be used successfully to implement reinforcement learning in a neural network [3, 6, 13]. Networks no longer need an external module that evaluates and changes the connections after each processing step.

Activation Dependent Plasticity is used to adapt connection weights in the experiments. Activation Dependent Plasticity (ADP) is based on Hebb's ideas of strengthening connections that fire together [11]. As shown by [6, 13] reward can also be integrated into the more sophisticated Spike Time Dependent Plasticity (STDP) learning model.

Adaptation of an excitatory axon weight (axon connected to dendrite):

$$w_{a+}(t+1) = w_{a+}(t) + \eta_a \cdot \rho_j(t) \cdot \phi_i \cdot \phi_j \tag{7}$$

where $w_{a+}(t)$ and $w_{a+}(t+1)$ are the axon weights before and after the adaptation. η_a is the learning factor for axons and $\rho_j(t)$ is the current reward value of the postsynaptic neuron. ϕ_i and ϕ_j represent the recent activity of the presynaptic and the postsynaptic neuron. In our experiments ϕ_i was kept between -1 and 0 for very little activity and from 0 to 1 for more activity, ϕ_j is kept between 0 and 1. For positive reward much activity in the presynaptic neuron strengthens the axon weight if also the postsynaptic neuron was active but little presynaptic activity weakens the axon weight. A negative reward value reverses the direction of change.

Adaptation of a dendrite weight (always excitatory):

$$w_d(t+1) = w_d(t) + \eta_d \cdot \rho_j(t) \cdot \phi_d \cdot \phi_j \tag{8}$$

where $w_d(t)$ and $w_d(t+1)$ are the dendrite weights before and after the adaptation. η_d is the learning factor for dendrites. ϕ_d represents the recent dendrite activity (which joins the activity of the connected axons) and is kept between 0 and 1. $\rho_j(t)$ and ϕ_j are discussed with Equ. 7. When all dendrite weights of a neuron are adapted they are normalised to add up to 1 again because of the dependencies between the weights (see Sect. 3).

Adaptation of an inhibitory axon weight (axon connected to neuron):

$$w_{a-}(t+1) = w_{a-}(t) - \eta_a \cdot \rho_j(t) \cdot \phi_i \cdot \phi_j \tag{9}$$

where $w_{a-}(t)$ and $w_{a-}(t+1)$ are the axon weights before and after the adaptation. η_a, $\rho_j(t)$, ϕ_i and ϕ_j are discussed in Equ. 7. Axons that are part of local inhibition, were not changed in our experiments. Also if ϕ_i is negative, the weight was kept equal. An inhibitory axon is strengthened, if it was not able to prevent bad feedback, and it is weakened, if it tried to prevent good feedback.

4.3 Delayed Feedback

An important issue to consider when dealing with feedback from the environment and the resulting rewards is delayed feedback. When weights are adapted and, as discussed later, neurons are created based on the current reward, it may at first seem to be the wrong time to do so. Typically, the feedback is received *after* the action responsible is executed. In fact, the time difference can vary significantly.

However, because in spiking neural networks there is no single event that is responsible for an action, but a continuous flow of spikes, the discussed methods can be efficient anyway. The input pattern, and hence the spiking pattern, usually does not change rapidly if a certain feedback is received. Figure 2 shows an example situation for this issue.

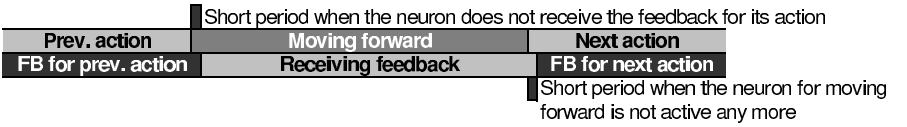


Fig. 2 When a spiking neural network controls a machine, there are two short periods where the assignment of the feedback to the corresponding action is problematic. Whenever an action is active for a longer period the feedback can be assigned correctly to the active neurons that are responsible for the current action.

Of course there remain situations in which it is difficult to assign the feedback correctly, for example if there is a big time difference between action and feedback, or if there are many competing actions or feedback values at the same time. However, even humans do not always arrive at the correct conclusions. They can deal with very complex relations but not with all of them.

5 Autonomous Creation of the Neural Network

The methods that were discussed in Sect. 4 tune a neural network. They are necessary to reinforce neural paths that were responsible for good actions and to weaken and finally remove connections that made a neuron output classify incorrectly.

There are different methods that result in new connections or even new neurons.

If a neuron has no input connections, it can connect to a random new predecessor in the previous layer. In our experiments the randomness was reduced by looking for neurons that have similar relative positions. Such a new connection is always excitatory and consists of a single axon and a single dendrite. This makes the postsynaptic neuron (or better: its single dendrite) represent a new class and it has the potential to carry out a new action when activated. Other neurons can be added to the class by creating new axons occasionally. An axon that does not fit into the class will be weakened by the mechanisms of Sect. 4.

New neurons are created to remember activation patterns that were responsible for good or bad actions. Liu, Buller and Joachimczak have already shown that correlations between certain input patterns and a certain reward can be stored by creating new neurons [17, 18]. For this task we add a new potential value to each neuron. The common "neuron potential" function defines when a neuron fires a spike. Our new "reward potential" function defines when a neuron has enough feedback to create a new neuron:

$$R_j(t+1) = \delta_R \cdot R_j(t) + \phi_j \cdot \rho_j(t) \tag{10}$$

where $R_j(t+1)$ is the new reward potential of neuron j while $R_j(t)$ is the old one. ϕ_j is the value for the recent activity that was also used in Sect. 4 and $\rho_j(t)$ is the current reward value of neuron j.

When $|R_j|$ reaches a certain threshold θ_R (different thresholds for positive and negative feedback are possible) all dendrites (excitatory connections) that were active recently are evaluated. Young dendrites are ignored, because they have not yet

proved themselves. For a dendrite with more than one axon it may be worth remembering an activation combination. A single axon may still be interesting if there was bad feedback, because the axon should have been inhibitory in this case. A list of axons, that had an influence on the activation of the neuron, is kept. If this combination is not yet stored in an existing preceding neuron, a new neuron is created and each axon of the list is copied. However, each of the new axons is connected to its own dendrite to record the combination. The new neuron will only be activated when the same combination is activated again. When the reward potential is positive, the new neuron is connected to the existing one by a new axon to the currently evaluated dendrite (neurons in layer B in Fig. 1.a and neurons in Fig. 3). A negative reward potential results in the addition of a new inhibitory axon to the existing neuron (neuron in layer C in Fig. 1.a).

Generally, the new neuron is then inserted into the layer previous to the existing postsynaptic neuron as shown in Fig. 3.a. The relative position of the new neuron will be similar to the relative position of the existing one.

However, if one of the axons to the new neuron has its source in the layer the new neuron should be inserted into, a new layer is created in front of the layer of the existing postsynaptic neuron as shown in Fig. 3.b. This way the feed-forward structure which our methods are based on can be preserved.

Once the new neuron is inserted, local inhibition will be generated. Experiments have shown that local inhibition makes learning much faster and much more reliable (see Sect. 6). Hence, new inhibitory axons are created to and from each new neuron. This inhibitory web has been automatically created within one layer in our experiments. In more sophisticated future developments this web should perhaps be limited to certain areas within a layer. Nested layers with neuron containers, which

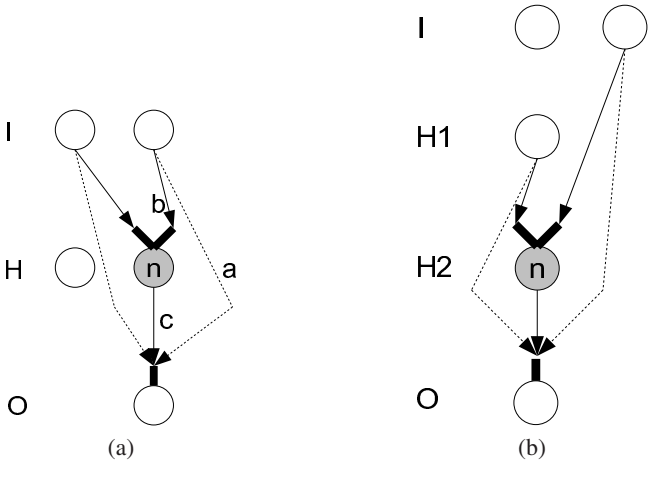


Fig. 3 (a) The new neuron n is created and active axons (a) are used to create new connections (b) to it. Then the new neuron is connected to the existing one (c). In (b) the new layer H2 is created before neuron n is inserted, because the output neuron already has a connection to a neuron in layer H1.

are basically supported by our implementation but are not used for the growing mechanisms yet, could help with this task.

6 Experiments

6.1 Setup

Our novel method for the autonomous creation of machine controllers was tested in a simulation of a mobile robot which moves using differential steering, as shown in figure Fig. 1.b. The initial neural network consists of 12 input neurons (2 for each sensor) and 4 output neurons (2 for each motor, see Fig. 1.a).

The input neurons are fed by values from 6 sonar sensors as shown in Fig. 1.b, each sensor feeds the input of 2 neurons. The sonar sensors are arranged so that 4 scan the front of the robot and 2 scan the rear as shown in the figure. The distance value is processed so that one input neuron fires more frequently as the measured distance increases and the other neuron connected to the same sensor fires more frequently as the distance decreases.

For the actuator control, the output connections are configured so that the more frequently one of the output neurons connected to each motor fires, the faster this motor will try to drive forward. The more frequently the other output neuron connected to the same motor fires, the faster that motor will try to turn backwards. The final speed at which each motor will drive is calculated by the difference between both neurons.

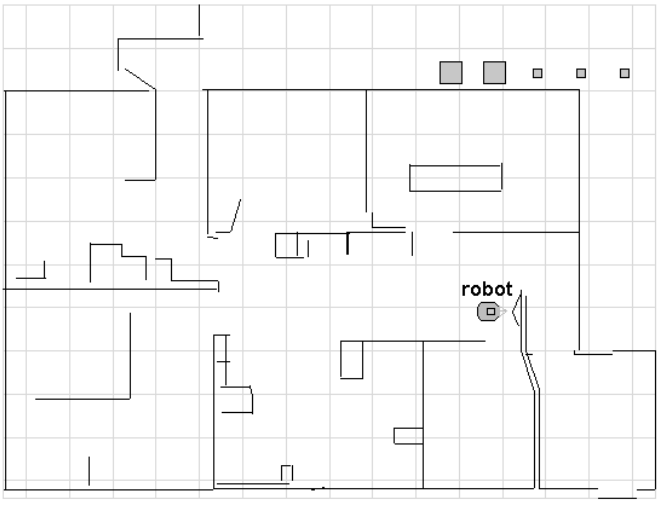


Fig. 4 A simulated Peoplebot is situated in this simulated office provided by MobileRobots/ActivMedia.

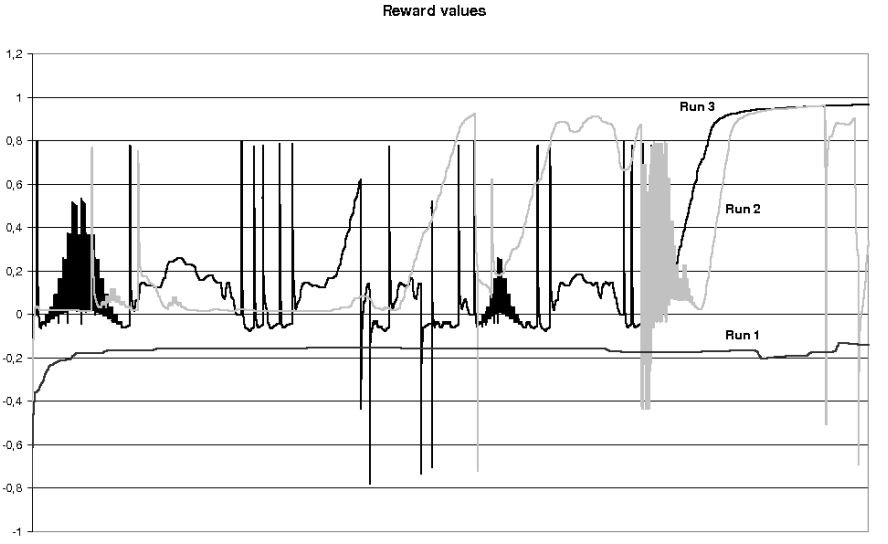


Fig. 5 The three curves show the reward values the robot received in simulation runs of the same duration and starting from the same position. In "Run 1" only forward movement is rewarded. The robot learns to hold its position to minimise negative feedback. In "Run 2" "activity reward" was introduced. In "Run 3" the same reward algorithm is used as in "Run 2", but in "Run 3" no or only small forward movement is punished. The robot learns to receive positive feedback with time which makes it more stable.

With this experimental setup the robot should learn to wander around in the simulated environment shown in Fig. 4 while avoiding obstacles.

The original robot's bumpers are included in the simulation and are used to detect collisions with obstacles, and are used to penalise significantly the reward values when such a collision occurs. The reward is increased continuously as the robot travels farther during its wandering behaviour. Backward movement is only acceptable when recovering from a collision, therefore it will only be used to increase the robot's reward value in that case, while it is used to decrease this value for all other cases. As time increases, linear forward movement will receive higher positive reward and this will discourage circular movement.

6.2 Results

This section discusses the challenges when finding an appropriate reward function. Additionally, we show the importance of local inhibition for the reliability of the learning methods. Finally, we present some results considering the performance and the topology of the constructive neural network.

The reward function that we have used in our experiments delivers -1 in the case where the robot crashes into an obstacle. Backward movement is punished

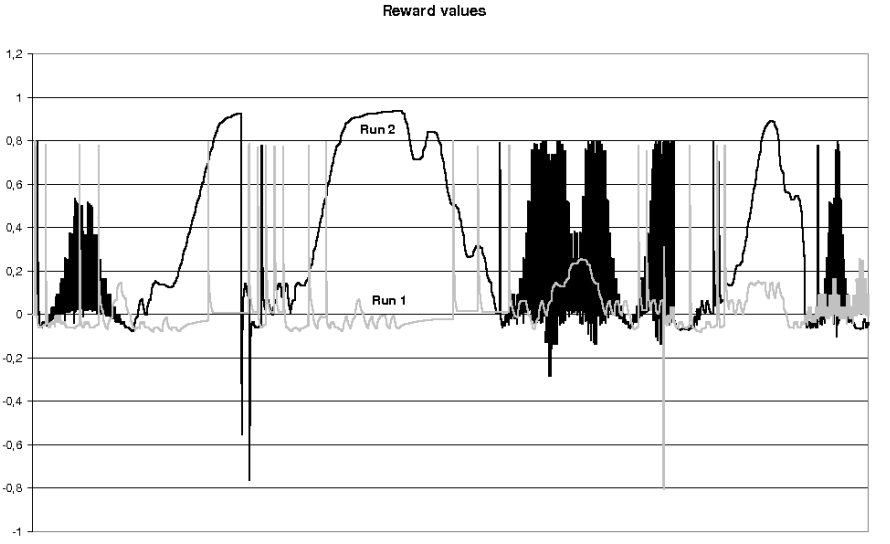


Fig. 6 "Run 1" shows the development of the reward value without local inhibition. This contrasting method increases the produced reward values significantly in "Run 2".

(negative value). There are two features that are not implemented in all of the three simulation runs of Fig. 5: First, no or only small forward movement is punished; second, backward movement is rewarded (positive value), if the robot received bad feedback for a while, to keep the robot active.

Figure 6 shows the importance of local inhibition. Without local inhibition the simulation run did not produce a single phase in which significant positive feedback was received. Only short periods of positive reward can be identified where the robot acted appropriately by chance. Local inhibition increases the contrast of spiking patterns, which makes single neurons and hence single motor actions more powerful and the assignment of reward to a certain spiking pattern more reliable.

Table 1 shows the results of a test of 50 simulation runs. In many cases the robot was able to learn the wandering task with the ability to avoid obstacles. Each run of the test sample was stopped after 20000 control cycles (processing the whole neural network in each cycle).

Table 1 Results of 50 simulation runs.

Performance of the controller			
	Min.	Max.	Avg.
Total reward	664.82	7486.24	4014.02
Average reward	0.03	0.37	0.20
Maximum speed	657.00	1056.00	1015.92
Average speed	102.74	971.63	260.37
Crashes	0.00	13.00	4.50

Topology of the controller			
	Min.	Max.	Avg.
Neurons	16.00	38.00	22.46
Excitatory axons	14.00	178.00	45.28
Excitatory dendrites	4.00	139.00	27.88
Inhibitory axons	4.00	7.00	4.36

The neural network contained no hidden neurons and no connections at the beginning. Connections for local inhibition were created when the controller was started. The speed values of the table are given in internal units of the simulation.

7 Conclusions and Further Work

We have shown that a neural network can be grown based on the reward measured by a feedback function which analyses the performance of a task in real-time. The neural network can control a machine such as a mobile robot in an unpredictable or unstructured environment.

Since the controller constructs itself, only the input layer, the output layer and a feedback function that measures the task performance of the machine and rewards the controller have to be defined. This means that the task of the designer involves only the definition of these elements, no effort is required for the actual design of the network.

Because controlling the machine and learning from experience continuously when running is integrated into a single and robust stage, the system can adapt to completely new situations without changing any part of the control structure manually. This involves three advantages in addition to the possibility of evolving a control system from scratch:

1. The machine can learn to react appropriately in situations that it has encountered previously but can use the experience it has gathered so far.
2. The machine can learn to handle changes to its hardware, for example if a sensor breaks.
3. When new machines are developed, it may be possible to use the neural network of established machines to reduce the time necessary for training them by starting from a semi-optimal state rather than from scratch. This could not only help to save time but also material, because a machine that has no knowledge at the beginning may easily damage itself.

Further analysis and improvements of the growing methodology are necessary to gain even better results from the growing methods for the neural network. For example it will be necessary to investigate the behaviour of the system with concurrent tasks and conflicting or noisy sensory data.

Also time dependent situations like action sequences and timed input (speech, objects moving through a visual field, ...) will be important issues for future research. Combining our results with other work may be very helpful for these issues, for example using Spike Time Dependent Plasticity (STDP) methods and recurrent connections. Another essential issue of further research is to investigate different reward functions for different tasks, because this is the key to effective evolutions of neural networks. Sophisticated reward functions will probably make a machine learn its tasks sooner and fulfil its tasks better (faster, more precisely, ...).

References

1. Alnajjar, F., Murase, K.: Self-organization of Spiking Neural Network Generating Autonomous Behavior in a Real Mobile Robot. In: Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation, vol. 1, pp. 1134–1139 (2005)
2. Alnajjar, F., Bin Mohd Zin, I., Murase, K.: A Spiking Neural Network with Dynamic Memory for a Real Autonomous Mobile Robot in Dynamic Environment. In: 2008 International Joint Conference on Neural Networks (2008)
3. Dauc, E., Henry, F.: Hebbian Learning in Large Recurrent Neural Networks. Movement and Perception Lab, Marseille (2006)
4. Elizondo, D., Birkenhead, R., Taillard, E.: Generalisation and the Recursive Deterministic Perceptron. In: International Joint Conference on Neural Networks, pp. 1776–1783 (2006)
5. Elizondo, D., Fiesler, E., Korczak, J.: Non-ontogenetic Sparse Neural Networks. In: International Conference on Neural Networks, vol. 26, pp. 290–295. IEEE, Los Alamitos (1995)
6. Florian, R.V.: Reinforcement Learning Through Modulation of Spike-timing-dependent Synaptic Plasticity. Neural Computation 19(6), 1468–1502 (2007)
7. Fritzke, B.: Fast Learning with Incremental RBF Networks. Neural Processing Letters 1(1), 2–5 (1994)
8. Fritzke, B.: A Growing Neural Gas Network Learns Topologies. In: Advances in Neural Information Processing Systems, vol. 7, pp. 625–632 (1995)
9. Gómez, G., Lungarella, M., Hotz, P.E., Matsushita, K., Pfeifer, R.: Simulating Development in a Real Robot: On the Concurrent Increase of Sensory, Motor, and Neural Complexity. In: Proceedings of the Fourth International Workshop on Epigenetic Robotics, pp. 119–122 (2004)
10. Greenwood, G.W.: Attaining Fault Tolerance through Self-adaption: The Strengths and Weaknesses of Evolvable Hardware Approaches. In: Zurada, J.M., Yen, G.G., Wang, J. (eds.) WCCI 2008. LNCS, vol. 5050, pp. 368–387. Springer, Heidelberg (2008)
11. Hebb, D.O.: The Organization of Behaviour: A Neuropsychological Approach. John Wiley & Sons, New York (1949)
12. Izhikevich, E.M.: Which Model to Use for Cortical Spiking Neurons? IEEE Transactions on Neural Networks 15(5), 1063–1070 (2004)
13. Izhikevich, E.M.: Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling. Cerebral Cortex 10, 1093–1102 (2007)
14. Jaeger, H.: The "Echo State" Approach to Analysing and Training Recurrent Neural Networks. GMD Report 148, German National Research Institute for Computer Science (2001)
15. Katic, D.: Leaky-integrate-and-fire und Spike Response Modell. Institut für Technische Informatik, Universität Karlsruhe (2006)
16. Kohonen, T.: Self-organization and Associative Memory. 3rd printing. Springer, Heidelberg (1989)
17. Liu, J., Buller, A.: Self-development of Motor Abilities Resulting from the Growth of a Neural Network Reinforced by Pleasure and Tension. In: Proceedings of the 4th International Conference on Development and Learning, pp. 121–125 (2005)
18. Liu, J., Buller, A., Joachimczak, M.: Self-motivated Learning Agent: Skill-development in a Growing Network Mediated by Pleasure and Tensions. Transactions of the Institute of Systems, Control and Information Engineers 19(5), 169–176 (2006)

19. Maass, W., Natschlaeger, T., Markram, H.: Real-time Computing without Stable States: A New Framework for Neural Computation Based on Perturbations. Neural Computation 14(11), 2531–2560 (2002)
20. Maes, P., Brooks, R.A.: Learning to Coordinate Behaviors. In: AAAI, pp. 796–802 (1990)
21. Martinetz, T.M., Schulten, K.J.: A "Neural-Gas" Network Learns Topologies. In: Kohonen, T., Mäkisara, K., Simula, O., Kangas, J. (eds.) Artificial Neural Networks, pp. 397–402 (1991)
22. Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. In: Adaptive Computation and Machine Learning. The MIT Press, Cambridge (2001)
23. Stanley, K.O., D'Ambrosio, D., Gauci, J.: A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks. Accepted to appear in Artificial Life journal (2009)
24. Tajine, M., Elizondo, D.: The Recursive Deterministic Perceptron Neural Network. Neural Networks 11, 1571–1588 (1998)
25. Vreeken, J.: Spiking Neural Networks, an Introduction. Intelligent Systems Group, Institute for Information and Computing Sciences, Utrecht University (2003)