Robotica (2015) volume 33, pp. 1551–1567. © Cambridge University Press 2014
doi:10.1017/S026357471400085X

Training oscillatory neural networks using natural gradient particle swarm optimization

Hamed Shahbazi*, Kamal Jamshidi, Amir Hasan Monadjemi and Hafez Eslami Manoochehri

Faculty of Engineering, University of Isfahan, Isfahan, Iran

(Accepted March 7, 2014. First published online: April 15, 2014)

SUMMARY

In this paper, a new design of neural networks is introduced, which is able to generate oscillatory patterns in its output. The oscillatory neural network is used in a biped robot to enable it to learn to walk. The fundamental building block of the neural network proposed in this paper is O-neurons, which can generate oscillations in its transfer functions. O-neurons are connected and coupled with each other in order to shape a network, and their unknown parameters are found by a particle swarm optimization method. The main contribution of this paper is the learning algorithm that can combine natural policy gradient with particle swarm optimization methods. The oscillatory neural network has six outputs that determine set points for proportional-integral-derivative controllers in 6-DOF humanoid robots. Our experiment on the simulated humanoid robot presents smooth and flexible walking.

KEYWORDS: Central pattern generators; Humanoid robots; Neural networks; Policy gradient learning; Walking; Sensory feedbacks.

1. Introduction

Neural networks (NN), as used in artificial intelligence, have been usually viewed as simplified models of neural processing in the brain, even though the relation between this model and biological architecture of the brain is debated as it is not clear as to what degree artificial neural networks resemble the brain function.[2] A neural network, in the case of artificial neurons called artificial neural network (ANN) or simulated neural network, is an interconnected group of natural or artificial neurons that use a mathematical or computational model for information processing based on a connectionist approach to computation. These can be used to model complex correlation between inputs and outputs or to find patterns in data. Theoretical and computational neuroscience is the field concerned with the theoretical analysis and computational modeling of biological neural systems. Since neural systems are intimately related to cognitive processes and behavior, the field is closely related to cognitive and behavioral modeling. In this paper, a new model for artificial neural networks based on the concept of oscillation in neurons is designed.

Many different tasks in human body originate from oscillatory neural networks (ONNs) called central pattern generators (CPGs).[11] CPGs are neural circuits located at the ending parts of the brain and the beginning parts of the spinal cord of a large number of animals and are responsible for generating rhythmic and periodic patterns of locomotion in different parts of body.[15] Although these pattern generators use very simple sensory inputs imported from sensory systems, these can produce high dimensional and complex patterns for walking, swimming, jumping, turning, and other types of locomotion. The origin of many movements in animals is the CPG, which was discovered by Brone in the early decades of the 20th century.[5] Many implementations of CPGs are based on coupling some special neurons to make them work together to produce a desired rhythm.

* Corresponding author. E-mail: shahbazi@eng.ui.ac.ir

Coupled oscillator-based CPG implementations offer miscellaneous features such as the stability properties of the limit-cycle behavior (i.e., ability to overcome perturbations and compensate their effects), smooth online modulation of trajectories through changes in the parameters of the dynamical system, and entrainment phenomena when the CPG is coupled with a mechanical system.[3] Examples of CPGs applied to biped locomotion are given in refs. [12, 18, 22]. Peters and Schaal[16] have discussed a CPG-based method for biped walking combined with policy gradient learning. A disadvantage of the CPG approach is that most of the time CPGs have to be custom-made for a specific application. There are few techniques to be applied in the construction of a CPG for generating an arbitrary input trajectory. Righetti and Ijspeert[18] presented a construction model for a generic model of CPG. This method is Programmable Central Pattern Generator (PCPG) and is used by applying dynamical systems and some differential equations for developing a training algorithm. The learner model is based on the works of Righetti *et al.*,[19] a Hebian learning method in dynamical Hopfs oscillators. The PCPG was used in generating walking patterns for a Hoap2 robot. This Hoap2 robot can increase its speed without falling on ground. Using this type of generic CPGs, they trained PCPGs with sample trajectories of walking patterns of a Hoap-2 robot. Each trajectory is a teaching signal to the corresponding CPG controlling associated joints.

Hackenberger initiated some proceedings[10] on a PCPG model included in ref. [18] to use a nonlinear feedback policy for balancing a humanoid robot during a walking gait. This system consists of two modules: a polar-based PCPG, which reproduces a walking trajectory, and a reinforcement learning agent responsible for modifying walking patterns. This paradigm can use PCPGs and can enable them to incorporate gyro feedbacks into the system definitions that generate walking trajectories. Degallier[4] defined a modular generator of movements called the Unit Pattern Generators (UPGs) and combined it to build CPGs for some robots with high degrees of freedom (DOF). He applied his framework to interactive drumming and infant crawling in an iCub humanoid robot.

Gams *et al.*[8] discussed a system (called CDS-ODS) for learning and encoding a periodic signal with no knowledge of its frequency and waveform, and this system was able to modulate an input periodic trajectory in response to some external events. Their system is used to learn periodic tasks on the arms of a humanoid HOAP-2 robot for the task of drumming. The canonical dynamical system is actually a polar implementation of PCPGs included in ref. [10]. Another online method of trajectory learning for a seven-link biped robot is discussed in ref. [6], which is based on a fuzzy system.

In this paper, we introduced a new type of neural networks to build CPGs. The fundamental building block of a neural network is o-neurons that can generate oscillations in its transfer functions. Since natural policy gradient learning is used in training a CPG paradigm, we called these as ONNs. O-neurons are connected and coupled with each other to shape a network, and their unknown parameters are found by a natural policy gradient learning algorithm. The outline of a higher level controller and the methods used for the stability are discussed in our previous work.[20]

In the next section the fundamental block of ONNs is introduced. The structure and functions of O-neurons are widely explained in this section. Section 3 is dedicated to the proposed model of neural network presented in this work. It explains the topology and the learning algorithm of the proposed method on a humanoid robot locomotion. The experimental results and analysis are discussed in Section 4. Section 5 includes conclusions and future works.

2. O-Neurons in Oscillatory Neural Networks

The fundamental building block of ONNs is O-neurons, a new design of oscillatory neurons introduced here. This neuron model is inspired by the realistic models of neurons such as the Hodgkin–Huxley model.[9] In these realistic models, the dynamics of electrophysiological transaction on neuron's membrane is modeled through some nonlinear ordinary differential equations (ODEs) for demonstrating behavioral changes during time. Figure 1 illustrates the schematic diagram of O-neurons. The internal parts of O-neurons are shown in this figure. Each O-neuron has an internal state, which is saved in a three-element vector [a,b,c]. This internal state is being updated during the time by neurons dynamics. O-neurons are biased by two biases, v_1 and v_2, which initiate the dynamics of neurons. These biases determine the initial frequency and initial phase of the oscillation. These are trained in the learning procedure of the network. An O-neuron receives P_a, P_b, P_c, and P_{dc} inputs. The P_a, P_b, and P_c inputs enter sensory feedbacks to the network and modify output patterns of the network, while the P_{dc} input is used in synchronizing the neuron with other neurons in the network.

https://doi.org/10.1017/S026357471400085X Published online by Cambridge University Press

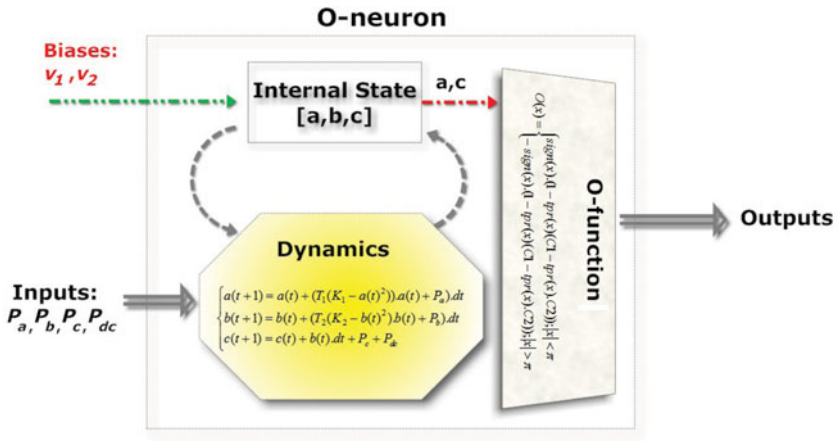

Fig. 1. Schematic diagram of the O-neurons. Each neuron receives some inputs; P_a, P_b, and P_c enter sensory feedbacks to the network and modify output patterns of the network. Gyro and Foot sensors are connected to these inputs. P_{dc} is used in synchronizing the neuron with other neurons in the network. Bias values initiate the state vector [a,b,c]. Dynamics described in Eqs. (1)–(4) change the internal states. Inputs enter the dynamics and change the output, which is produced from [a,c]. These values enter an O-function transfer function. O-function makes a desirable low-computational oscillation.

Synchrony is one of the most important properties of O-neurons; it can learn to generate specific component of a complicated pattern and accelerate or decelerate itself to change the total pattern. This model consists of a particular transfer function called O-function. O-function receives two of the three internal states, [a, c], and computes an oscillation based on these values. The c value is another output of the neuron and is used for synchronization within the network.

The internal dynamics of the neuron are described in Eqs. (1)–(4):

$$a(t+1) = a(t) + (T_1(a(0)^2 - a(t)^2)a(t) + P_a)dt, \tag{1}$$

$$b(t+1) = b(t) + (T_2(b(0)^2 - b(t)^2)b(t) + P_b)dt, \tag{2}$$

$$c(t+1) = c(t) + b(t).dt + P_c + P_{dc}, \tag{3}$$

$$a(0) = 1, b(0) = 1.v_1, c(0) = 1.v_2. \tag{4}$$

Here $a(t)$, $b(t)$, and $c(t)$ are three variables of internal dynamics of neurons, and T_1 and T_2 are two timing constants that determine the sensitivity level of neurons. The initial values of variables $a(t)$, $b(t)$, and $c(t)$ are constructed by the bias values v_1 and v_2. For simplicity, $a(0)$ is set to be 1 for all neurons in the network. Since there are some synaptic weights connected to the output of O-neurons, this setting does not reduce the generality of the system. The dynamical system generates new state variables $a(t+1)$, $b(t+1)$, and $c(t+1)$ in the next time episode and send $a(t+1)$ and $c(t+1)$ to O-function. The O-function is the transfer function of the neuron which computes the neuron output pattern based on its internal state. This function is described in Eq. (5).

$$O(a,c) = \begin{cases} \text{sign}(c).a.(1 - \Phi(c)(C1 - \Phi(c).C2)); & |c| < \pi \\ -\text{sign}(c).a.(1 - \Phi(c)(C1 - \Phi(c).C2)); & |c| > \pi \end{cases}. \tag{5}$$

In this equation $C1 = 0.5$ and $C2 = 0.0416$ are two pre-computed constants that shape a desirable oscillation in this function. The "sign" function determines the sign of its input c. The $\Phi(c)$ is a limiter function which bounds the input x to a linear interval and generates a periodic behavior for it. Eq. (6) presents this function.

$$\Phi(x) = \begin{cases} (x_{\text{mod}(2\pi)} - \frac{\pi}{2})^2; & \left|(x_{\text{mod}(2\pi)}\right| < \pi \\ (x_{\text{mod}(2\pi)} - \frac{3\pi}{2})^2; & \left|(x_{\text{mod}(2\pi)}\right| > \pi \end{cases}. \tag{6}$$

https://doi.org/10.1017/S026357471400085X Published online by Cambridge University Press

This phi-function computes the remainder of its input x by dividing it by 2π, then reduces it and calculates its square. For each input x, the $\Phi(c)$ is computed once, saved, and used twice in computing the O-function. This procedure reduces the computational complexity of the function and allows for numerous implementations of this function during the test period. These neural networks can be run on different controller platforms as well.

Each O-neuron is able to learn a specific periodic signal and regenerates it during the time according to some sensory inputs. Sensory inputs originated from different stimulations on the neural system of vertebrate animals are the main source of changes in their neural pattern generators. In Eqs. (1) and (2), the dynamics is designed to damp the inserted perturbations. In fact, O-neuron forms a stable limit cycle that can attract all close trajectories in the phase plane by making it possible for the robot to bypass rough surfaces.[7] This damping property is very useful in walking robot applications.[11]

In Eq. (3), the dynamics of c is a linear one combined with some instantaneous inputs P_c and P_{dc}. Input P_c is a sensory input that can be originated from the sensory feedbacks of the system, and moves the phase of pattern to a new phase. P_{dc} is a phase difference that comes in the network from another neuron. This phase difference is a means for the synchronization of neurons within the network. Using this option in the network, a change in one neuron phase will be transmitted to other neurons.

2.1. Properties of O-neurons

Here some of the characteristics of O-neurons are discussed. As described earlier, each O-neuron has three sensory inputs that can regenerate its output value. P_a affects the amplitude of the output, P_b affects the frequency of the oscillation, and P_c changes the phase of the pattern. In Fig. 2, an example of these effects on O-neurons is shown. In this example, $a(0)$, v_1, and v_2 are set to 1, 1, and 2 respectively. Sensory inputs P_a, P_b, and P_c are generated as some pulses. These pulses exhibit some of the hidden properties of O-neurons.

Each O-neuron has two-limit cycle on the amplitude and frequency of the oscillation. It is designed to lock on a specific amplitude and frequency, and it keeps on generating patterns with that frequency and amplitude. P_a and P_b can induce changes on the output pattern, but attractor limit cycles attract new trajectories to the original one. This phenomenon is illustrated in Fig. 3.

Another important characteristic of O-neuron is coupling. O-neurons can be coupled with each other and share their phase. This sharing property enables them to be synchronized and cooperate with each other. Coupling between neurons is a very important natural property in neural networks.[13] In Fig. 4, there are two O-neurons which are coupled with each other. The bias value of each neuron is written on it. The first one exports its c state to the other. P_{dc} is generated and imported to the equations of the second neuron based on this c value.

There are two tests shown in Fig. 5 that exhibit the coupling properties between two O-neurons. In the first one the effect of change in P_b of the first neuron on the second one is examined. Figure 5 illustrates output of the second neuron (output2) based on the two pulses of P_b. It can be observed that oscillation before and after these pulses are synchronous. In fact, pulses change the first output, and the coupling term makes the second neuron coupled with the first one. So the final outputs would be still synchronous. The second test is designed to test the effect of change in P_c of the first neuron on the second one. The neurons would also be synchronized after the perturbation pulses. Here the oscillation phase of the first neuron is changed, but the second neuron made itself adapted with the first one by changing its phase using the coupling term. Coupled neurons in a network can work with each other and generate a rhythmic synchronized pattern.

3. Network of O-Neurons

In order to make the neural network, O-neurons are connected and coupled and a network of neurons is stabilized. This neural network can have multiple layers. In Fig. 6, a network with only two layers is presented. In the first layer some O-neurons are coupled with one another. Coupling can be done by sending out the c state to the next neuron in the network. This phase will be added to a phase difference value and inserted into the input P_{dc} at next neuron. O-neurons can be connected to a synaptic neuron in the next layer. Synaptic neuron computes a weighted sum on the outputs of O-neurons and sends this sum to a limiter transfer function. The limiter transfer function can bound the upper and lower outputs of synaptic neuron. Each synapse is modeled by a weight link between

https://doi.org/10.1017/S026357471400085X Published online by Cambridge University Press

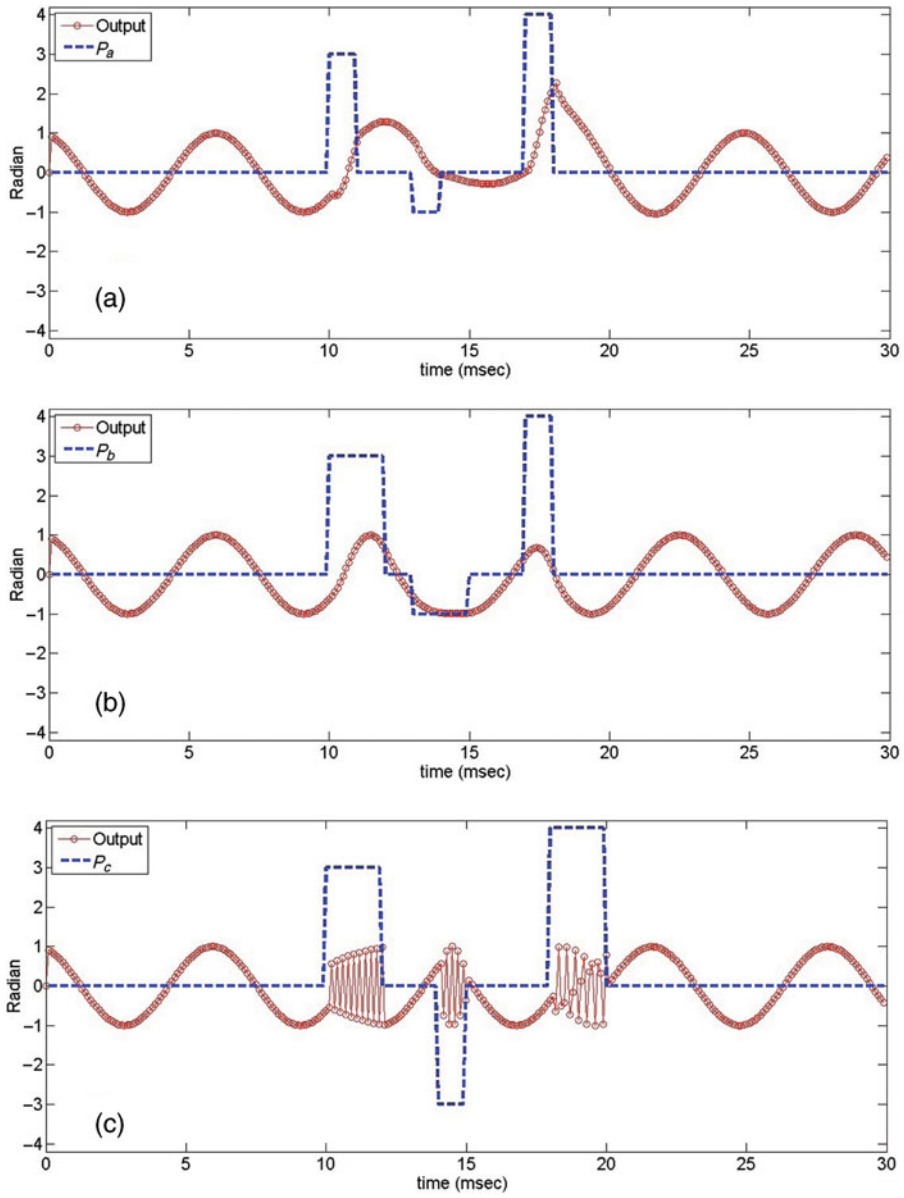

Fig. 2. Effect of P_a, P_b, and P_c on the output of O-neurons. (a) P_a is generated as two pulses and each pulse changes the amplitude of the pattern and makes it higher or lower. When the pulse gets to zero, the neuron tries to regenerate the original waveform of the pattern. The system has been built based on an attractor limit cycle which is able to attract all close trajectories. (b) Pulses of P_b change the frequency of the pattern. When these pulse gets to zero, the neuron generates the original waveform with basic frequency. Equations here are also built based on limit cycle behavior. (c) Pulses of P_c demonstrate that there is no limit cycle for c equation. This value is the phase of the neuron. Phase of oscillation will be changed whenever P_c is not zero. This is used to reset the phase of pattern generation.

O-neurons and synaptic neuron. The weights should be trained in the learning procedure to shape a particular wave form. In the next section the procedure of training these weights would be discussed. Function F is described in Eq. (7),

$$F(ow) = \sum w_i.O(a_i, c_i) + b_i) = \begin{cases} ow; & |ow| < M1 \\ \text{sign}(ow).M1; & |ow| > M1 \end{cases}. \tag{7}$$

https://doi.org/10.1017/S026357471400085X Published online by Cambridge University Press

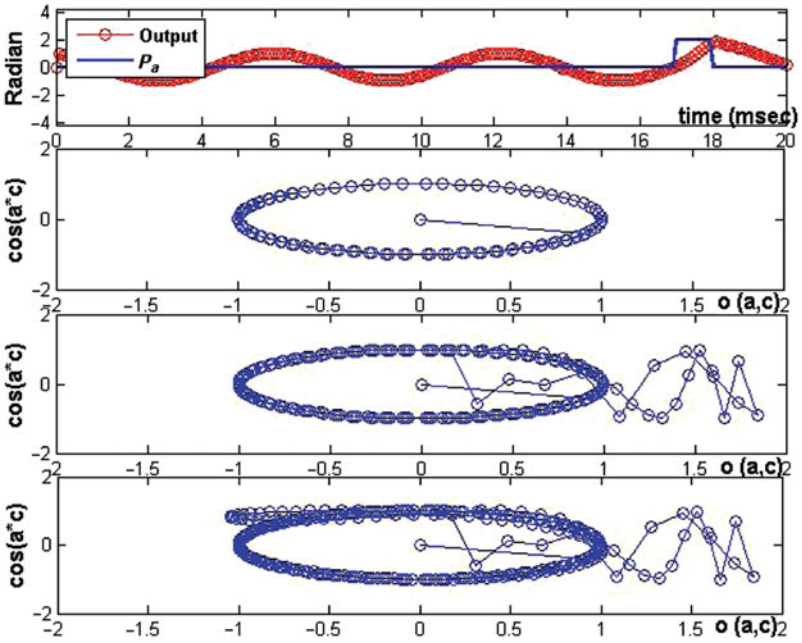

Fig. 3. Response of the limit cycle of O-neurons to the changes in P_a.

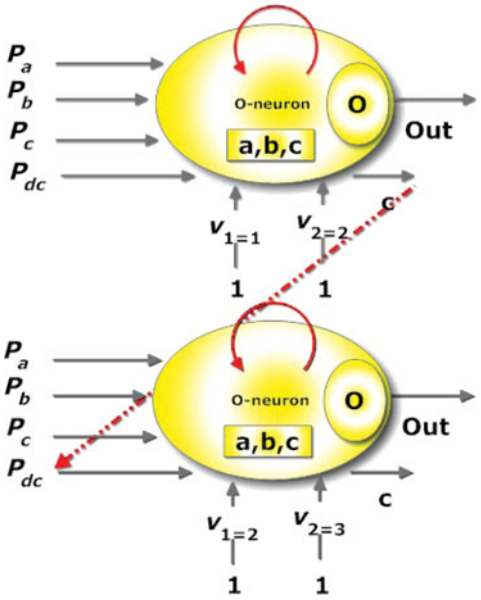

Fig. 4. Coupling between two neurons. Here, in the first neuron, v_1 is set to 1 and v_2 is set to 2. In the second neuron, v_1 is set to 2 and v_2 is set to 3.

Function F is purely linear for the range between $-M1$ and $M1$. For inputs bigger or smaller than this maximum threshold, its output is $-M1$ or $M1$, depending on the sign of the input. The function prevents the output size from becoming bigger than a predefined value. This would be useful in the control application of walking. b_i is the bias value in the synaptic neuron and is able to bias the output on a specific level. This constant should be trained in the first step of learning in order to simplify next steps.

A one-dimensional network of neurons can be coupled with another network to shape a multi-output network of O-neurons. In this application a six-dimentional network is necessary to generate suitable walking patterns for all six joints involved in the walking of a humanoid robot. Figure 7 illustrates the connection between networks that make a n-dimensional network. In this figure, a

https://doi.org/10.1017/S026357471400085X Published online by Cambridge University Press

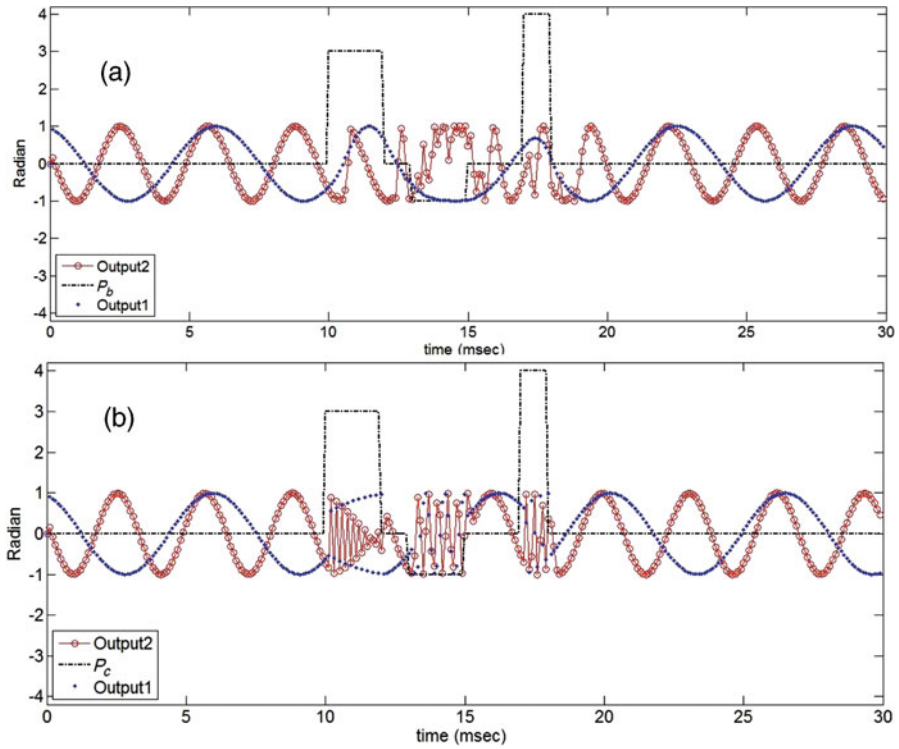

Fig. 5. Coupling between two neurons and perturbation on (a) P_b and (b) P_c.

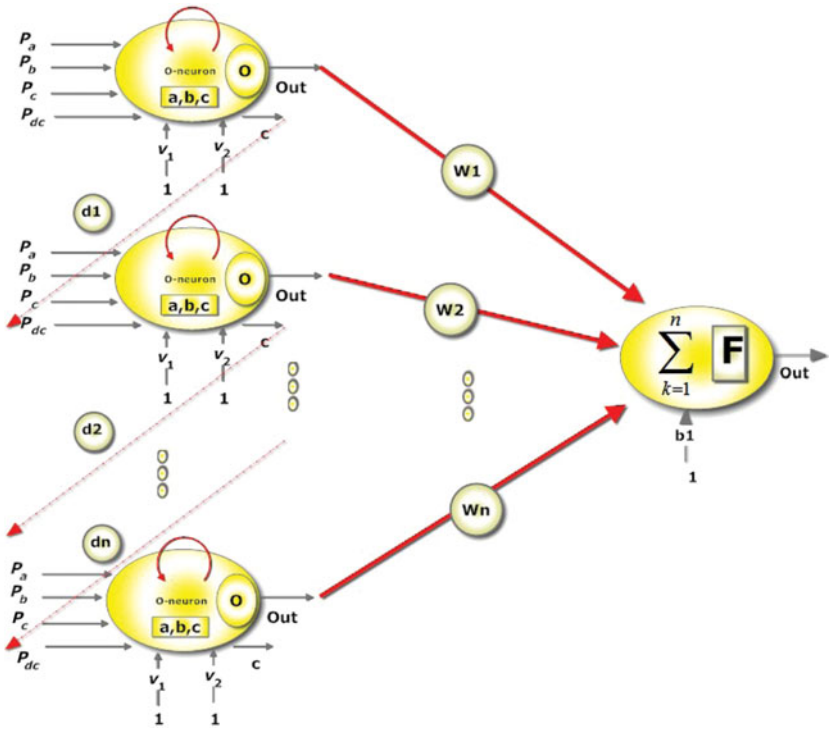

Fig. 6. One-dimensional network of O-neurons. In the first layer there are usually some O-neurons coupled with one another. Coupling can be done by sending out the c state to the next neuron in the network. This phase will be added to a phase difference value and enter to the input P_{dc} at the next neuron. O-neurons can be connected to a synaptic neuron in the next layer. Synaptic neuron computes a weighted sum on the outputs of O-neurons and sends this sum to a limiter transfer function. The limiter transfer function can bound the upper and lower outputs of synaptic neuron. Each synapse is modeled through a weight link between O-neurons and synaptic neuron.

https://doi.org/10.1017/S026357471400085X Published online by Cambridge University Press

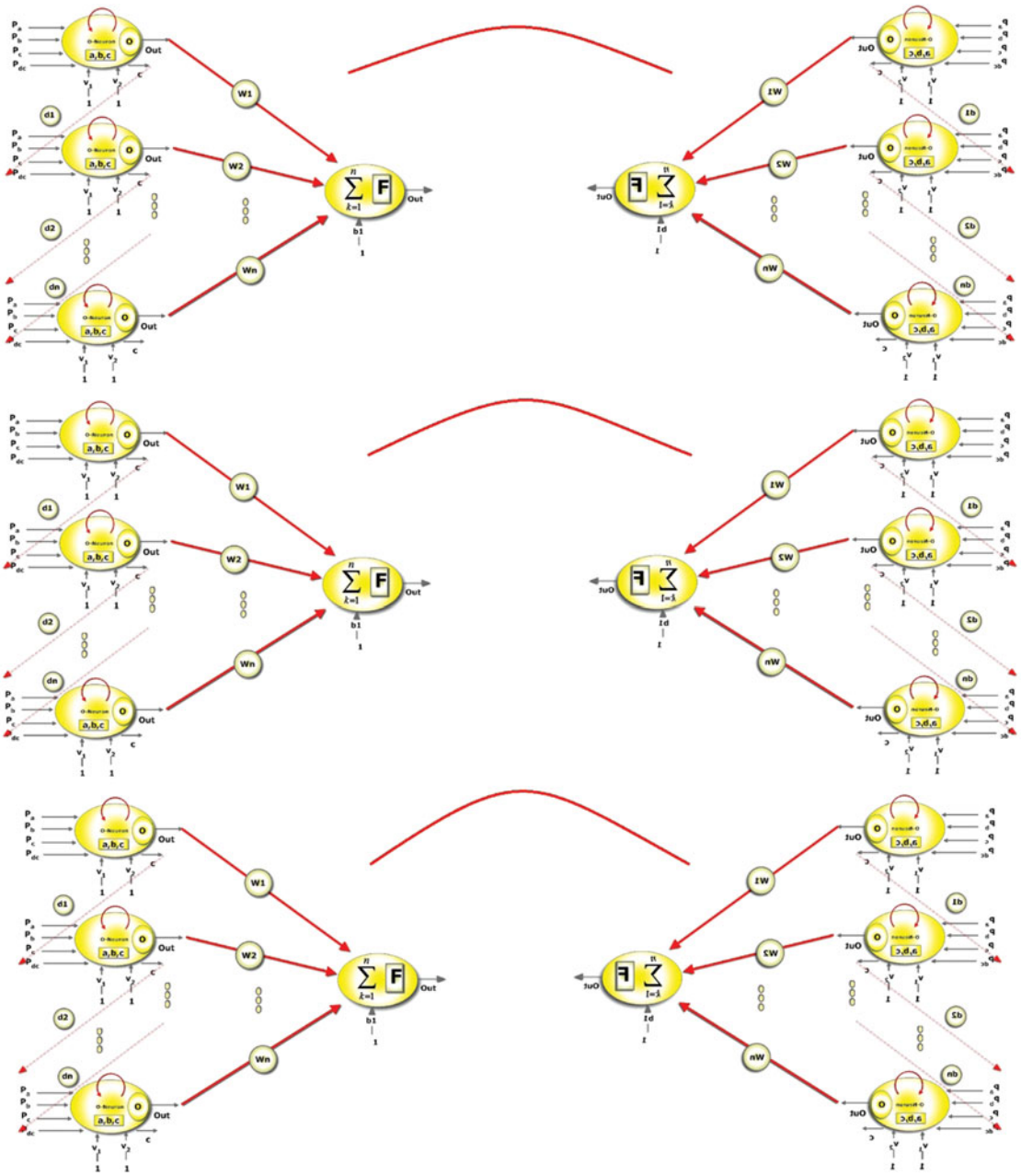

Fig. 7. Connection of some one-dimensional neural networks. The circle between one-dimensional network contains an external difference *dij* between the one-dimensional network. These *dij*s are trained in the last step of the learning algorithm.

circle containing an external difference *dij* between the one-dimensional network within the bigger network is observed. These *dij*s are trained in the last step of the learning algorithm.

To coordinate several joints, each joint in each leg is coupled with the opposite leg. The hip joint in the left leg is coupled with the hip in the right and the same is true for the knee and ankle joints. This coupling on the robot is illustrated in Fig. 8.

3.1. Training the oscillatory network

In order to train the unknown parameters in the presented neural network, the particle swarm optimization (PSO)[14] method is used, which is combined with the natural policy gradient method.[16] PSO is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. In this application the measure of quality is the

https://doi.org/10.1017/S026357471400085X Published online by Cambridge University Press

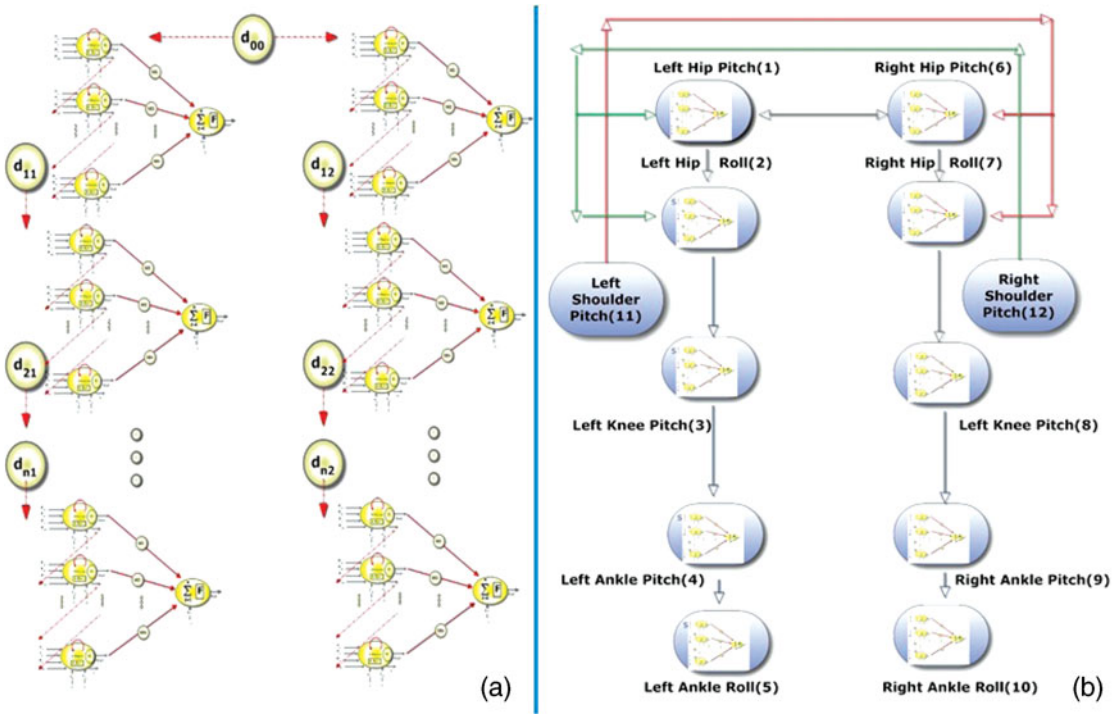

Fig. 8. Coupling between joints in a simulated humanoid robot.

root mean square error of the outputs with respect to some pre-recorded walking trajectories. These predefined trajectories in this work are walking trajectories produced by some other methods of walking for NAO humanoid robot. In ref. [1], a method based on inverted pendulum and inverse kinematic is discussed to generate walking pattern for NAO. PSO optimizes a problem by having a population of candidate solutions and moving these particles around in the search space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position and is also guided toward the best known positions in the search space, which are updated as better positions found by other particles. This is expected to move the swarm toward the best solutions.

In the modified version of PSO in this paper, particle's movement is also influenced by the natural gradient of their current positions. In this way, each particle can compute the natural gradient of its positions and update its new velocity using Eq. (8). Position update is like the original version presented in Eq. (9),

$$v_i(t+1) = v_i(t) + c_1.\text{rand}().(p_i^{\text{best}} - p_i(t)) + c_2.\text{rand}().(p_{g\text{best}} - p_i(t)) + c3.NG(p_i(t)), \quad (8)$$

$$p_i(t+1) = p_i(t) + v_i(t), \quad (9)$$

where $v_i(t+1)$ is the new velocity for the ith particle; c_1, c_2, and c_3 are the weighting coefficients for the personal best and global best positions respectively; $p_i(t)$ is the ith particle's position at time t; p_i^{best} is the ith particle's best known position; and $p_{g\text{best}}$ is the best position known to the swarm. The rand() function generates a uniformly random variable $\in [0, 1]$. Variants on this update equation consider best positions within a particle's local neighborhood at time t. The natural gradient in Eq. (8) is computed using Algorithm 2. Each particle in each learning stage is modeled as an actor that needs a policy to generate its pattern. The policy of the network π_θ is parameterized by a $\theta = [\theta_1, \theta_2, \ldots, \theta_n]$ vector which determines its behavior. Each policy in policy space determines how the states correspond to the actions. The objective in reinforcement learning is to find the optimal policy that contains the best correspondence. The Actor-Critic methods use both branches of methods for finding this optimal policy: value iteration (VI) and policy iteration (PI). In a natural version of the Actor-Critic, the actor updates are archived using stochastic policy gradient employing Amari's natural gradient approach, while the critic obtains both natural policy and additional parameters of a

https://doi.org/10.1017/S026357471400085X Published online by Cambridge University Press

value function simultaneously by linear regression.[17] It is shown in ref. [16] that actor improvement with natural policy gradients is particulary appealing as they are independent of the coordinate frame of the chosen policy representation and can be estimated more efficiently than regular policy gradients. A stochastic Gaussian policy π_θ function can be defined for the network as:

$$\pi_\theta(x,u) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{1}{2}(u - g_\theta(x))\right), \tag{10}$$

where x is the input state of the network, which is *determined* by the current time, and u is the action of the *network*, which is the network output potential. σ is a constant used for standard deviance, and $g_\theta(x)$ is the mean behavior of the neural network, which is defined in Eq. (11):

$$g_\theta(x) = \sum_{\forall j=3k,k=1}^{n} \theta_j.O(\theta_{j+1}.t_i + \theta_{j+2}), \tag{11}$$

where n is the number of neurons and θ_j is the jth parameter of the vector. For $j = 3k$, $\theta_j = w_i$; $j = 3k{+}1$, $\theta_j = v_1^i$, and $j = 3k{+}2$, $\theta_j = v_2^i$.

In such a policy gradient problem, the following parameter vector θ is defined in the system as some unknown parameters of $g_\theta(x)$ function,

$$\theta = [w_1, v_1^1, v_2^1, w_2, v_1^2, v_2^2, \ldots, w_n, v_1^n, v_2^n]. \tag{12}$$

In this vector, w_i is the synaptic weight between neuron i and synaptic neuron, and v_1^i and v_2^i are two biases used as initial states in each O-neuron. In this function, O is the O-function which is used as transfer function in O-neurons. To use this stochastic policy in the natural Actor-Critic method, $\nabla\log(\pi_\theta(x,u))$ should be computed as in Eq. (13):

$$\nabla_\theta\log(\pi_\theta(x,u)) = \frac{(u - g_\theta(x))}{\sigma^2}\nabla g_\theta(x). \tag{13}$$

The most important aspect of Eq. (14) is computation of gradient $\nabla g_\theta(x)$. In this approach, gradient $\nabla g_\theta(x)$ is computed analytically:

$$\nabla^i g_\theta(x) = \frac{\partial g_\theta(x)}{\partial\theta_i}. \tag{14}$$

Gradient of $g_\theta(x)$ with respect to θ_i is:

$$\nabla^i g_\theta(x) = \begin{cases} O(\theta_{j+1}.t_i + \theta_{j+2}); & \text{CD1,} \\ t_i.\theta_{j-1}.\Psi(\theta_j.t_i + \theta_{j+1}).[4.C2.(\Psi(\theta_j.t_i + \theta_{j+1}))^2 - 1]; & \text{CD2} \\ ,\theta_{j-2}.\Psi(\theta_{j-1}.t_i + \theta_j).[4.C2.(\Psi(\theta_{j-1}.t_i + \theta_j))^2 - 1]; & \text{CD3,} \\ -t_i.\theta_{j-1}.\Psi(\theta_j.t_i + \theta_{j+1}).[4.C2.(\Psi(\theta_j.t_i + \theta_{j+1}))^2 - 1]; & \text{CD4,} \\ -\theta_{j-2}.\Psi(\theta_{j-1}.t_i + \theta_j).[4.C2.(\Psi(\theta_{j-1}.t_i + \theta_j))^2 - 1]; & \text{CD5.} \end{cases} \tag{15}$$

In this equation, $C2 = 0.0416$ is a pre-computed constant. $\Psi(x)$ is defined as in Eq. (16),

$$\Psi(x) = \begin{cases} (x_{\text{mod}(2\pi)} - \frac{\pi}{2}); & \left|x_{\text{mod}(2\pi)}\right| < \pi \\ (x_{\text{mod}(2\pi)} - \frac{3\pi}{2}); & \left|x_{\text{mod}(2\pi)}\right| > \pi \end{cases}. \tag{16}$$

https://doi.org/10.1017/S026357471400085X Published online by Cambridge University Press

The $\Psi(x)$ function is the derivation of the $\Phi(x)$ function. CD1, CD2, CD3, CD4, and CD5 are five different conditions of derivation. These conditions are described below:

- CD1 : $j_{\text{mod}3} = 0$,
- CD2 : $j_{\text{mod}3} = 1$ and $\left|\theta_j.t_i + \theta_{j+1}\right| < \pi$;
- CD3 : $j_{\text{mod}3} = 2$ and $\left|\theta_{j-1}.t_i + \theta_j\right| < \pi$;
- CD4 : $j_{\text{mod}3} = 1$ and $\left|\theta_j.t_i + \theta_{j+1}\right| > \pi$;
- CD5 : $j_{\text{mod}3} = 2$ and $\left|\theta_j.t_i + \theta_{j+1}\right| > \pi$.

Algorithm 1 describes the whole training algorithm of the one-dimensional oscillatory network.

Algorithm 1 Training the Oscillatory Neural Network

for each particle pi: $i \leftarrow 1$ to S **do** Initialize the particle's position with a uniformly distributed random vector.
  Initialize the particle's best known position to its initial position
  if $error(pi)$ is less than $error(global)$ **then**
    update the swarm's best known position
  end if
  Initialize the particle's velocity
end for
$it \leftarrow 0$;
repeat
  $gbest(it) \leftarrow gbest(it-1)$
  for each particle pi: $i \leftarrow 1$ to S **do** :
    Update the particle's velocity randomly;
    $NG(pi) \leftarrow$ *ComputeNaturalGradient* using Algorithm 2;
    Update vi the particle's velocity base on Eq. (8);
    Update xi the particle's position base on velocity;
    $error(xi) \leftarrow$ *TestTheNeuralNetwork*(xi);
    if ($error(xi)$ is less than $error(gbest)$) **then**
      Update the particle's best known position
      if ($error(pi)$ is less than $error(gbest)$) **then**
        update the swarm's best known position
      end if
    end if
  end for
  $it \leftarrow it + 1$;
until the desired precision is met

4. Experimental Results

In this section the experiments of walking of a simulated humanoid robot are discussed. The learning process includes the training of basic walking trajectories to ONNs. This should be done to find the initial state values of O-neurons within the network. The proposed learning algorithm in this study is explained in Algorithm 1. In Fig. 9 an example of a trained ONN in Simulink is illustrated which was programmed by sample trajectories. This example clarifies the design of an ONN consisting of some O-neurons. The internal design of an O-neuron is presented at the top of this figure. The system is trained with a sample input, and the biases and synaptic weights are set in the system using some constant blocks. It can be seen that the first neuron sends out its phase to other neurons as a synchronization criteria. These neurons have learned the most important harmonics of the input trajectory.

The limit cycle behavior of this network is shown in Fig. 10. The top part here shows the sequence of shaping the limit cycle. The trajectory of the system starts from the initial states in (1) and makes some cycles in the phase plane. In (3) the first cycle is completed and the second cycle is started. The end of the second cycle is shown in (6). The third and fourth cycles in (7), (8), and (9) can be observed. These inward cycles shape a global limit cycle for the system, which is attractor one. The

https://doi.org/10.1017/S026357471400085X Published online by Cambridge University Press

Algorithm 2 Natural Policy Gradient Computation

repeat
  Sample $t^{0:H}$, $y_{teach}^{0:H}$, $dy_{teach}^{0:H}$
  Set $x \leftarrow t$, $u \leftarrow y_{teach}$, $r \leftarrow (dy_{teach}.dy_{teach}^{T})$.
  Obtain the sufficient statistics:
    Policy derivatives: $\psi_k \leftarrow \nabla_\theta log(\pi_\theta(x_k, u_k))$
    Eligibility: $\phi \leftarrow E\left\{(\sum_{k=0}^{H} \psi_k)\right\}$
    Average reward: $\overline{r} \leftarrow (\sum_{k=0}^{H} r_l)^T$
  Fisher Matrix: $F_\theta \leftarrow E\left\{(\sum_{k=0}^{H} \psi_k)(\sum_{l=0}^{H} \psi_l)^T\right\}$
  Vanilla Gradient: $g \leftarrow E\left\{(\sum_{k=0}^{H} \psi_k)(\sum_{l=0}^{H} r_l)^T\right\}$
  Obtain Natural Gradient by computing:
    $Q \leftarrow (1 + \phi^T(F_\theta - \phi\phi^T)^{-1}\phi)$
    Baseline $b \leftarrow Q(\overline{r} - \phi^T F_\theta^{-1} g)$
    Natural Gradient $g_{NG} \leftarrow F_\theta^{-1}(g - \phi b)$
until gradient estimate g_{NG} converged
Output g_{NG}
end;

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
Left hip	1.2281	0.5816	0.6776	1.4503	−0.7610	1.9333	0.0954	0.0684	0.0416
Left knee	−1.2301	0.6741	0.5775	0.3289	0.8544	1.2954	−0.0750	0.0788	−0.1004
Left ankle	0.6095	1.9313	1.2198	0.6208	1.0824	0.9331	0.1622	−0.0266	−0.1034
Right hip	0.2995	−1.2305	0.6100	−1.0783	0.3565	−0.0610	0.0381	−0.0778	0.1171
Right knee	−0.5960	1.2200	1.8092	−0.9703	1.1285	−0.5220	0.1309	−0.1086	0.0166
Right anke	1.8340	−0.6116	1.2143	−0.6903	1.7512	−1.0000	−0.0764	0.0768	−0.0946

Table I. Final parameters of ONNs in the walking application obtained by PSO. The nine vectors for each joint represent parameters of three O-neurons connected together.

attractor limit cycle of a network can absorb close trajectories. Changing the sensory inputs can put new trajectories near this limit cycle and the limit cycle absorbs them. In other words, when some sensory inputs are inserted, the system can autonomously regenerate the intermediate samples of the output signal until it returns to its natural initial behavior state. This phenomenon is applied in our neural networks to control the walking behavior. The sensory signals (the gyro and foot pressure) are entered into the system and its natural behavior is to walk on the surface smoothly as fast as possible.

As shown in Fig. 8, a six-dimensional oscillator neural network is used in walking robots. Each output of the network is sent to each joint in humanoid robot. Pre-recorded walking trajectories are used for the learning in the learning algorithm. Using Algorithm 1, these trajectories are fed into the learning system and the error of the learning is decreased to a minimum point. These errors (for six learning episodes) are illustrated in Fig. 11. The learning is performed rapidly for all input trajectories. Fig. 12 presents the input training trajectories and the final outputs of the neural network. Here it can be observed that the neural network could learn input walking patterns very well. Final parameter values of the network are shown in Table I. Using the final parameters obtained in the learning algorithms, we have simulated the humanoid robot in the WebotsTM Simulator. A higher level controller is also needed to maintain stability. Design of this high-level controller is not discussed is this paper, but the reader can refer to ref. [21]. Snapshots of the final walking on the humanoid robot are presented in Fig. 13.

4.1. Method comparisons

To illustrate the efficiency of the proposed learning algorithm (ONNs), this method is compared with two older methods used in learning periodic patterns. These methods are the learning methods

https://doi.org/10.1017/S026357471400085X Published online by Cambridge University Press

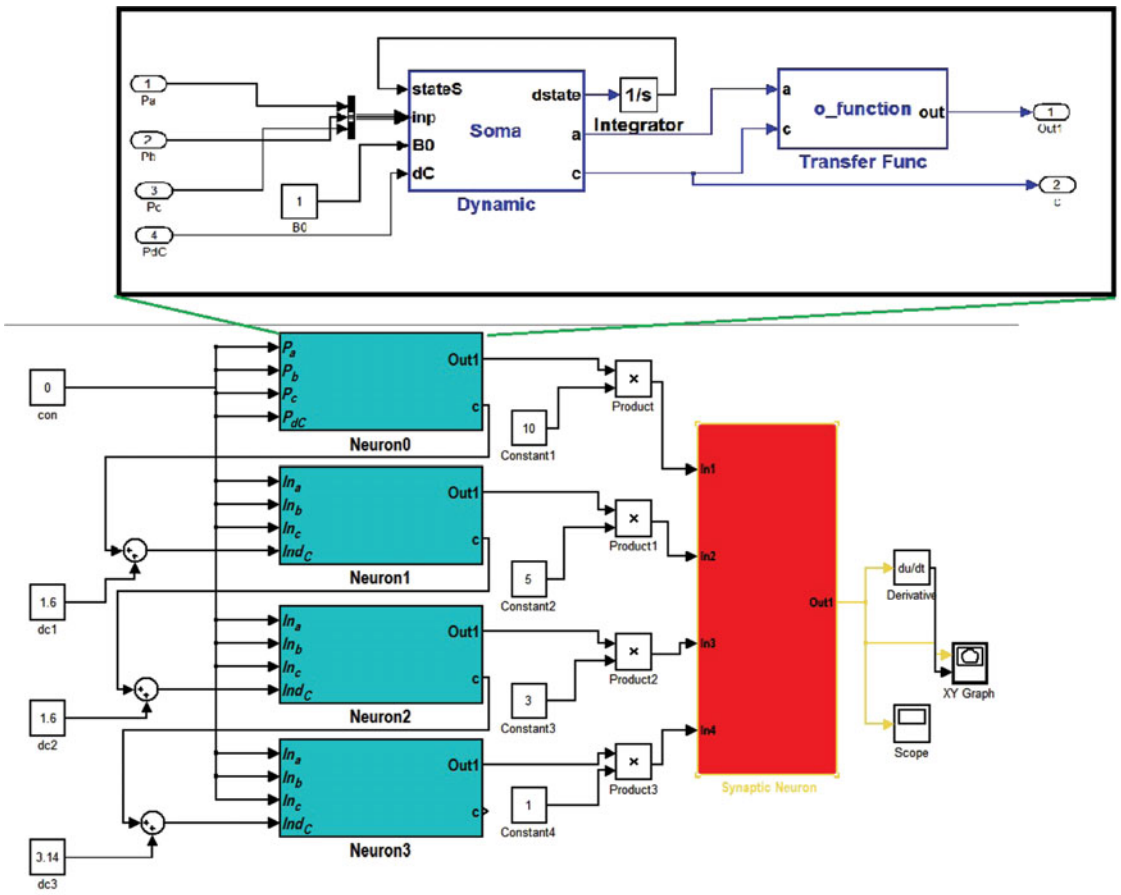

Fig. 9. An example of trained ONNs in Simulink is illustrated, which can be programmed by a sample training trajectories. Design of an O-neuron is presented in the top of this figure.

used in ref. [18] and are called PCPGs (optimized version is in ref. [10]), andtwo-stage learning algorithms given in ref. [8] are called the CDS-ODS method. Righetti and Ijspeert[18] represented a model for the construction of a generic model of CPG. This method is the PCPG method and is used in dynamical systems and some differential equations to build up a training algorithm. The leaner model, based on the works of Righetti *et al.*,[19] is the Hebian leaning method in dynamical Hopfs oscillators. The PCPG has been used to generate walking patterns for a Hoap2 robot. Using this type of generic CPGs, they trained the generic CPGs with sample trajectories of walking patterns of Hoap-2 robot. Gams *et al.*[8] discussed a system for learning and encoding a periodic signal with no knowledge on its frequency and waveform, which was able to modulate an input periodic trajectory in response to some external events. Their system is not used for learning to walk but for some other periodic tasks, such as the task of drumming, on the arms of a humanoid HOAP-2 robot. This model uses two layers of trajectory generation. The first layer, the Canonical Dynamical System (CDS), is actually a polar implementation of generic CPGs included in ref. [10]. The second layer, the Output Dynamical System (ODS), is responsible for learning and regenerating the waveform of the input signal.

These methods are compared in Table II. The reference signals are the pre-defined trajectories for walking. In this comparison four different efficiency criteria are defined. The first one is the *average convergence time*. This indicates the average time (proportional to the number of training epochs) required for the convergence of algorithm (i.e., reaching at an acceptable training error). The second criterion is the *average convergence rate*. This parameter determines the ability of the learning system to learn different patterns. Since many complicated patterns are not found from the initial points of a learning system, the system fails to find suitable weights to generate that pattern. The third one is the *average training error*, which is the average error during the training process. The fourth criterion is

https://doi.org/10.1017/S026357471400085X Published online by Cambridge University Press

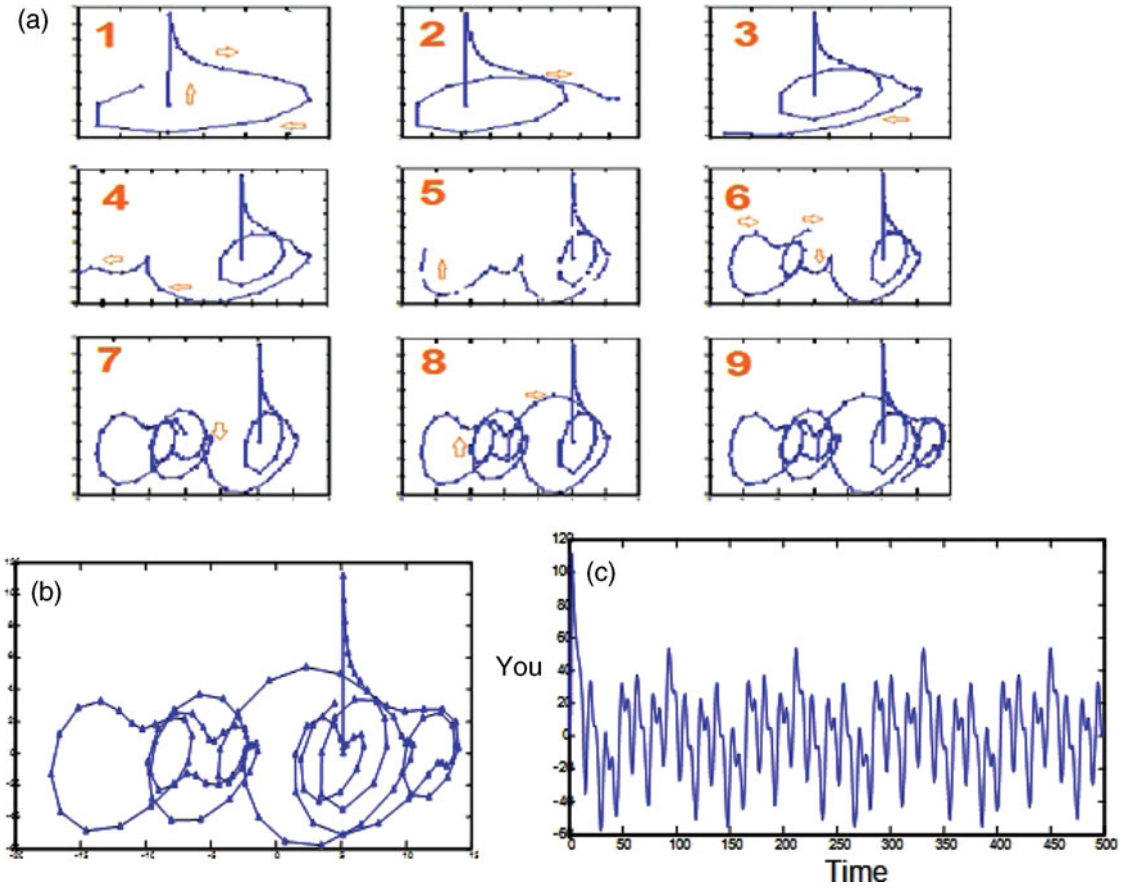

Fig. 10. The limit cycle behaviors of the network. The sequence of shaping the limit cycle during the time is shown in (a). The trajectory of the system starts from the initial states in (1) and makes some cycles in the phase plane. In (3) the first cycle is completed and the second cycle is started. (6) shows the end of the second cycle. The third and fourth cycles are seen in (7), (8), and (9). These inward cycles shape a global limit cycle for the system, which is attractor one. (b) Complete form of the limit cycle, (c) waveform of the limit cycle.

the *average testing error*, which presents the average error of the system after all the parameters are found and fixed. This criterion shows the quality of the learning.

As illustrated in the Table II, the proposed method has the maximum convergence time between the other methods. This means that our algorithm can obtain necessary parameters slower than others. In spite of slower convergence time, the average convergence rate in ONNs is higher than the other methods. In other words, ONNs can learn many different patterns that the other two methods are not able to learn. This is because the learning capabilities in PCPGs and CDS-ODSs are so much dependent on the initial values of their oscillators. They usually fail to learn a complicated pattern from many initial points. So the user should try many initial points to train its desired pattern in PCPGs and CDS-ODSs. On the other hand, ONNs start from different initial vectors and simultaneously search from these initial vectors. This leads to a successful learning in long convergence time.

Since ONNs start from many different initial vectors, they make very big errors during the training stage, but when they find fundamental frequency and suitable initial points, they rapidly converge to the best final parameters. In other words, this method makes many different wrong guesses in the parameter space, but can quickly find the right answer. So the average testing error in our method is higher than in the other methods. This fact is illustrated in Fig. 14. The average testing error of ONNs is lower than in the other methods. This indicates that this method can find its required parameters better than the other methods and is able to regenerate teaching trajectories very well.

https://doi.org/10.1017/S026357471400085X Published online by Cambridge University Press

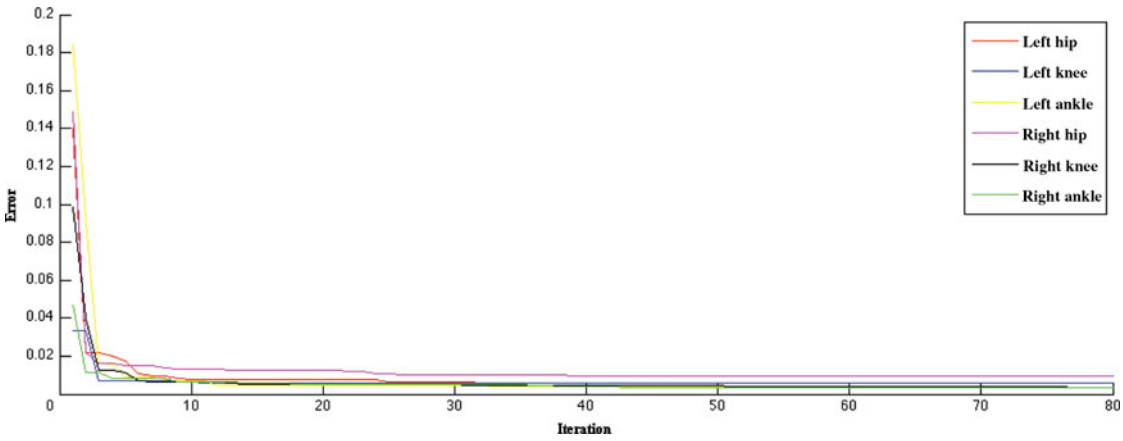

Fig. 11. Error of the proposed neural network during the time. The ONN is trained for six trajectories of six joints in the robot using Algorithm 1.

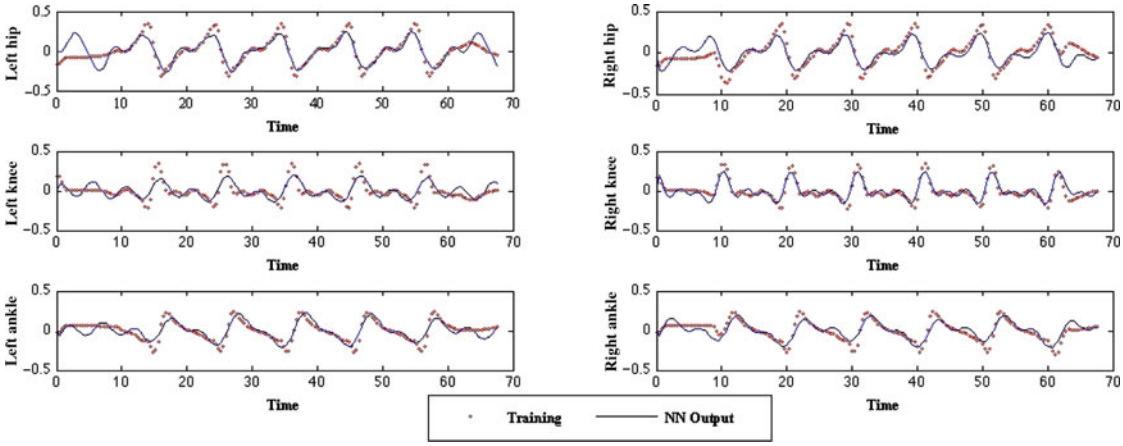

Fig. 12. Final results of the training of ONNs for walking application.

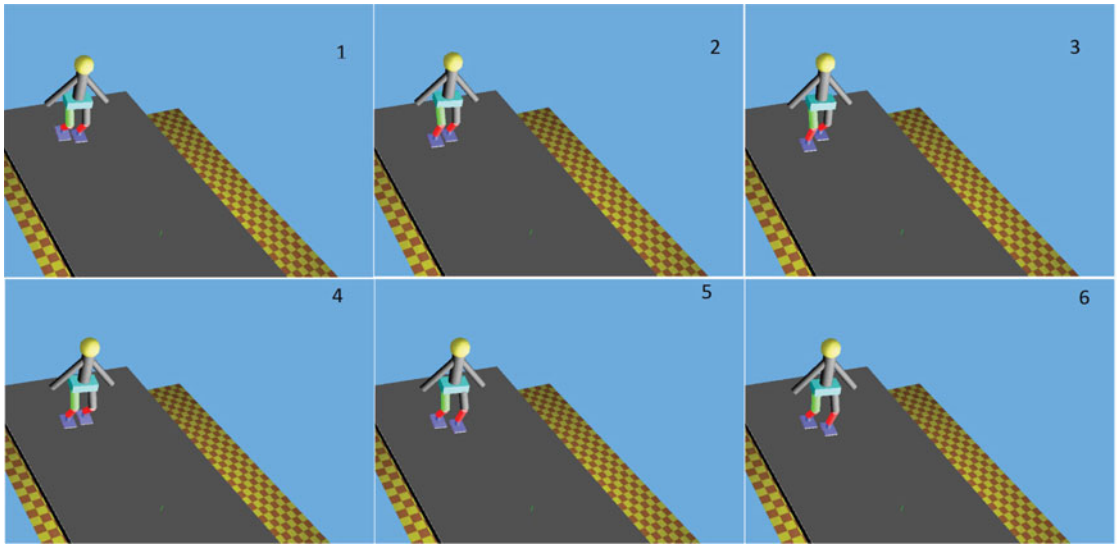

Fig. 13. Snapshots of the trained walking on the simulated robot.

https://doi.org/10.1017/S026357471400085X Published online by Cambridge University Press

Method	Average convergence time	Average convergence rate	Average training error	Average training error
PCPG	7.5	35%	66.38	21379.10
ODS-CDS	14.5	45%	82.97	69.54
ONNs	45	93%	1321.8	10.212

Table II. Comparison of learning behavior of three different methods. In this comparison four different efficiency criteria are defined. Average convergence time indicates the average time (proportional to the number of training epochs) required for the convergence of algorithm. Average convergence rate determines the ability of the learning system to learn different patterns. Average training error is the average error during the training process. Average testing error presents the average error of the system after all the parameters are found and fixed in the system.

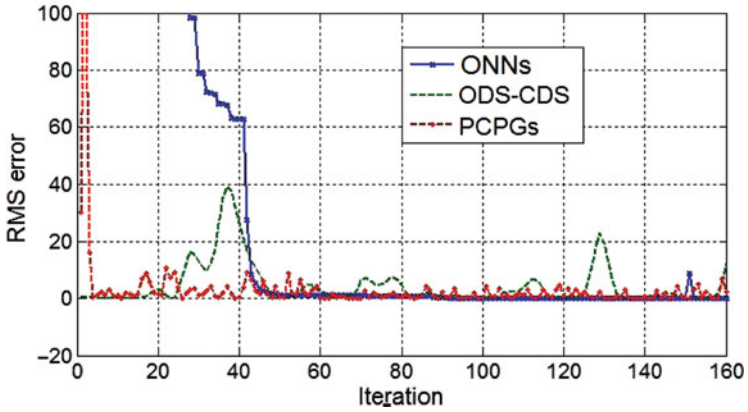

Fig. 14. Learning curve of three different methods. ONNs are converged slower than other two methods. The average testing error in ONNs is higher than in the other methods, while the average testing error in ONNs is lower than the other methods. This indicates that this method can find its required parameters better than other methods and is able to regenerate teaching trajectories very well.

5. Conclusions

In this paper, a new training algorithm for ONNs is proposed. This network is able to generate oscillation for different types of applications. We focused on learning locomotion and used the proposed model for a humanoid robot to teach it to walk on ground. Walking is a very complicated problem for humanoid robots, which many researchers have tried to solve. We choose this problem to show the abilities of our neural network to learn complicated patterns. The network used particle swarm optimization to learn input walking trajectory. One of the most important characteristics of this neural network is the ability of coupling between different neurons and dimensions. This coupling is used to synchronize outputs for the walking application and maintain robot's stability.

Acknowledgment

We ought to thank Abbas Abdolmaleki and Dr Amin Mahnam for their guidance and help. In addition, we should thank Dr Masood Fazeli, who helped us use contributions of his PhD thesis.

References

1. J. J. Alcaraz-Jimenez, D. Herrero-Perez and H. Martınez-Barberá, "Motion planning for omnidirectional dynamic gait in humanoid soccer robots," *J. Phys. Agents* **5**(1), 25–34 (2011).
2. I. A. Basheer and M. Hajmeer, "Artificial neural networks: Fundamentals, computing, design, and application," *J. Microbiol. Methods* **43**(1), 3–31 (2000).
3. M. R. Clark, G. T. Anderson and R. D. Skinner, "Coupled oscillator control of autonomous mobile robots," *Auton. Robots* **9**(2), 189–198 (2000).

https://doi.org/10.1017/S026357471400085X Published online by Cambridge University Press

4. S. Degallier, L. Righetti, S. Gay and A. Ijspeert, "Toward simple control for complex, autonomous robotic applications: Combining discrete and rhythmic motor primitives," *Auton. Robots* **31**(2–3), 155–181 (2011).
5. J. Duysens and H. W. A. A. Van de Crommert, "Neural control of locomotion; part 1: The central pattern generator from cats to humans," *Gait Posture* **7**(2), 131–141 (1998).
6. Y. Farzaneh, A. Akbarzadeh and A. A. Akbari, "Online bio-inspired trajectory generation of seven-link biped robot based on T–S fuzzy system," *Appl. Soft Comput.* **14**, 167–180 (2014).
7. A. F. Filippov, "A sufficient condition for the existence of a stable limit cycle for an equation of the second order," *Matematicheskii Sbornik* **72**(1), 171–180 (1952).
8. A. Gams, A. J. Ijspeert, S. Schaal and J. Lenarčič, "On-line learning and modulation of periodic movements with nonlinear dynamical systems," *Auton. Robots* **27**(1), 3–23 (2009).
9. S. Grillner, "Biological pattern generation: The cellular and computational logic of networks in motion," *Neuron* **52**(5), 751–766 (2006).
10. F. Hackenberger, "Balancing central pattern generator based humanoid robot gait using reinforcement learning," Graz University of Technology (2007).
11. A. J. Ijspeert, "Central pattern generators for locomotion control in animals and robots: A review," *Neural Netw.* (special issue) **21**(4), 642–653 (2008).
12. A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Comput.* **25**(2), 328–373 (2013).
13. E. M. Izhikevich and Y. Kuramoto, "Weakly coupled oscillators," *Encyclopedia Math. Phys.* **5**, 448 (2006).
14. J. Kennedy, "Particle Swarm Optimization," **In**: *Encyclopedia of Machine Learning* (Springer, New York, NY, 2010), pp. 760–766.
15. O. Kiehn and S. J. B. Butt, "Physiological, anatomical and genetic identification of CPG neurons in the developing mammalian spinal cord," *Prog. Neurobiol.* **70**(4), 347–361 (2003).
16. J. Peters and S. Schaal, "Policy Gradient Methods for Robotics," IEEE/RSJ International Conference on Intelligent Robots and Systems (IEEE 2006) (2006) pp. 2219–2225.
17. J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing* **71**(7), 1180–1190 (2008).
18. L. Righetti and A. J. Ijspeert, "Programmable Central Pattern Generators: An Application to Biped Locomotion Control," Proceedings of 2006 IEEE International Conference on Robotics and Automation (ICRA 2006) (2006) pp. 1585–1590.
19. L. Righetti, J. Buchli and A. J. Ijspeert, "Dynamic Hebbian learning in adaptive frequency oscillators," *Physica D.* **216**(2), 269–281 (2006).
20. H. Shahbazi, K. Jamshidi and A. H. Monadjemi, "Modeling of mesencephalic locomotor region for NAO humanoid robot," *Ind. Robot. Int. J.* **39**(2), 136–145 (2012).
21. J. Strom, G. Slavov and E. Chown, "Omnidirectional Walking Using Zmp and Preview Control for the NAO Humanoid Robot," **In**: *RoboCup 2009: Robot Soccer World Cup XIII* (Springer, New York, NY, 2010) pp. 378–389.
22. T. Zielińska, "Biological inspiration used for robots motion synthesis," *J. Physiol.* (Paris) **103**(3), 133–140 (2009).