Prediction of Pull-out capacity of Suction Caissons Using Self-Evolving Neural Networks

Abdussamad Ismail and Dong-Sheng Jeng

Abstract— A self-evolving neural network is developed using a combination of PSO and JPSO algorithms to predict the pull-out capacity of suction caissons in clay. The algorithm is proposed with the aim of reducing the network complexity without compromising accuracy. A database consisting of experiments performed on suction caissons is used to construct and validate the network model. The performance comparisons indicate that the proposed self-evolving neural network predicts more the capacity of suction caissons accurately than neural networks developed using conventional methods.

I. INTRODUCTION

Suction caissons serve as cost effective alternatives to conventional offshore foundations such as driven piles. They are favorably used in deep ocean and gas developments due to construction difficulty associated with the installation of foundation in such environment. By virtue of their larger diameter, suction caissons give a better capacity to withstand lateral loads than piles. The construction of caissons involves allowing the caisson to sink into the sea bed under its own weight, and then subsequently undergo an assisted penetration through pumping out of water from inside the caisson. Suction caissons usually function as anchors to hold the offshore installations, subject to severe environmental conditions, in place. Thus, there is a tendency for pullout movement to occur due to tensile forces exerted by the chain attached to the caisson (see Figure 1). Accurate evaluation of the pull-out capacity of suction caissons is therefore necessary for a reliable geotechnical design of this kind of foundation. Various attempts to improve the understanding of the behaviour of suction anchors through physical and numerical modelling have been reported in the literature [1]-[2]-[2]-[4]-[5]. However, due to the limited information about the complex nature of failure mechanism involved the reliability of conventional methods of analysis in accurately predicting the capacity of suction anchors is challenged. In an attempt to improve the accuracy of pull out capacity estimation, Rahman et al. [6] developed an empirical model using BPN networks. Based on their finding, neural network models gave reasonably accurate results in comparison with observed capacities and FEM based predictions. The downside of developing models using such a conventional neural network design procedure is that there is tendency to end up with a sub-optimal network with undesirably large network size, which could undermine its ability to generalize.

In this the present work, an approach to simultaneous optimization of network topology and parameters is proposed. The aim is to minimize the size of the network while still maintaining accuracy and generalization capability. The proposed algorithm is used to develop an empirical model to predict the pull-out capacity of suction anchors. The model depends on parameters such as the caisson geometry (depth and width), un-drain she strength of soil around the caisson tip, the depth of the load application point, direction of the pull-out force and loading rate.

II. NEURAL NETWORK MODELLING

Design of neural networks is a complex multi-dimensional optimization problem, involving not only choosing the optimum synaptic weights but also choosing a suitable processing function as well as an optimum network topology. The discrete, complex and multi-modal nature of the topology space, it is extremely challenging to optimize network architecture and the network parameters at the same time [7]. The classical topology optimization techniques include Network pruning [8][9], a top to bottom approach to network development, where the learning process begins with a large network, then subsequently trimmed to a smaller size by deleting redundant nodes and connections. Incremental learning algorithm [10][11] is a more convenient method in which the network size is increased by a gradual addition of nodes as training goes on. Near zero values are initially assigned to the synaptic weights associated with the newly added node to minimize the loss of knowledge. The
drawback of both pruning and incremental learning algorithms is their tendency to get the network entrapped in the topology space local minima [12].

Later developments in topology optimization methodology are mainly associated with evolutionary concepts of combinatorial optimization. These include the genetic algorithm (GA) based topology optimization algorithms such as EPNet [13] and NEAT [14]. In EPNet, the population of networks is initialized by randomly generating the topology and synaptic weights of the networks. The networks are then subjected to a series of parametric and structural mutation steps. Parametric mutations take the form of partial training using back-propagation and simulated annealing, while the structural mutations involve addition of node or connection (growing) or removal of node of connection (pruning). The mutations cycles are repeated until a satisfactory network is obtained. While in ESPNet, inefficient cross over operation is avoided, the manner in which the topology population is developed makes it inferior, in terms of computational efficiency, to NEAT, which start with a population of smallest possible networks, then gradually increasing the complexity as training progresses. The weak point of NEAT lies in the intricate cross-over procedure involved while updating the network topology. The use of a combined PSO and DPSO algorithms in evolving neural networks have also been reported in the literature [15][16]. Although PSO based algorithms are simpler to use and computationally more efficient than GA based methods, the inefficient procedure of topology generation undermines the capability of the algorithms to arrive at the optimum solution.

In this research, a population based self evolving network is proposed, where the initial topology begins with a single hidden node, then gradually evolving in size as the training progresses. The self evolution process begins by generating a population of neural nets with each having a random set of connection and synaptic parameters. The connection parameters are binary, assuming a value of 1 if there is a connection between two nodes and 0 if otherwise (Figure 2). They are updated using a jumping particle swarm optimization (JPSO) procedure in the cause of optimization process. JPSO algorithm, developed by Martinez-Garcia and Moreno-Pérez [17] is discrete optimization technique that turned out to be more efficient than the discrete version of particle swarm optimization algorithm (DPSO) proposed by Kennedy and Eberhart [18]. In Jumping PSO (JPSO) algorithm, the particle jumps from its current position to a new position under the influence of particle’s experience, global best position or explorative tendency which make a particle to make a random explorative search (see Figure. 3). Whether a particle jumping is influenced by previous experience or by explorative tendency depends on chance. The particle’s position is updated as follows:

\[
x_{i+1} = c_1 \otimes x_i \oplus c_2 \otimes b \oplus c_3 \otimes g
\]

where \( x_i \) and \( x_{i+1} \) are the vectors of current and future particle positions in the discrete search space. The parameters \( c_1 \), \( c_2 \) and \( c_3 \) are probabilities of jumping particle randomly, towards the best particle position and to the best swarm position respectively. B and g are, respectively, the particle best and global best positions. Equation (1) is implemented as follows:

\[
\begin{align*}
  x_{i+1} = \sum_{i \neq j} x_{i,j} \otimes &\begin{cases} 
  \rho & \text{if } i &\text{is connected to } j \\
  0 & \text{if } i &\text{is not connected to } j 
\end{cases} \\
  &+ \sum_{i \neq j} x_{i,j} \otimes \begin{cases} 
  b_{i,j} & \text{if } i &\text{is connected to } j \\
  0 & \text{if } i &\text{is not connected to } j 
\end{cases} \\
  &+ \sum_{i \neq j} x_{i,j} \otimes \begin{cases} 
  g_{i,j} & \text{if } i &\text{is connected to } j \\
  0 & \text{if } i &\text{is not connected to } j 
\end{cases} \\
  &+ c_1 + c_2 + c_3 = 1 
\end{align*}
\]

\[\rho\] represents a random value. The * operator is implemented by a stochastic modification of the features of the current particle with some features of its attractor. The updated position determined using equation (2) could be worse than the current one, therefore a random local search is carried out to find a better solution. In this research due to the mixed nature of optimization problem involving both continuous and discrete variables, the local search is carried out using few steps of back-propagation algorithm. Also, due to random resetting of the position of a fraction of swarm population at intervals during the training, the \( c_1 \) is reduced to zero, and the values of \( c_2 \) and \( c_3 \) sum up to 1.
The proposed JPSO algorithm is represented by the flowchart in Figure 4.

![Flowchart describing JPSO algorithm](image)

The synaptic weights of individual networks in the population are updated using a combination of PSO and BP algorithm. For the PSO part, the network parameters are updated using the following equations as proposed by Clerc and Kennedy [19]:

\[ v_{i,t+1} = \chi [ v_{i,t} + c_1 r_1 (b_i - x_{i,t}) + c_2 r_2 (g - x_{i,t}) ] \]

\[ x_{i,t+1} = x_{i,t} + v_{i,t+1} \]  \hspace{1cm} (3)

where \( \chi = \frac{2}{2 - \sqrt{\phi - 4\phi}} \)

where \( \phi = c_1 + c_2 > 4 \). The advantage of putting together the two techniques is to take the advantage of global search capability of the former and the ability of the later to perform local search. The proposed hybrid optimization algorithm is based on PSO technique and BP algorithm, whereby, both algorithms are used successively as training progresses. The idea is to get the best out of the two powerful algorithms by developing such algorithm which integrates more efficiently the PSO and BP techniques. The algorithm involves initially training the network parameters using PSO for a certain number of iterations, then training some selected (best performing) particles among the swarm population using BP algorithm for few number of iterations. The results of the local search by BP algorithm are then used to update the positions of relevant particles and the PSO takes over again. The cycle is repeated until a sufficiently accurate is obtained. In PSO, there is a possibility of particles to cluster around one co-ordinate, thereby, causing a stagnation in the search progress. To avoid this problem, duplicate particles have their positions reset randomly at the end of each cycle of PSO iterations. The positions of least performing fraction of the swarm population are also randomly reset in order to improve the topology search, having removed the random jumping aspect of JPSO.

When no further improvement is observed, the complexity of the network is increased by adding more nodes, one node at a time. To prevent the destruction of the so far acquired knowledge, the previous best particle positions (both topology and synaptic weights) are retained while adding one more node to the members of the swarm population. In this way, having to deal with unnecessary large network size is avoided as the case is with the models proposed by Kiranyaz et al. [15] and Xian-Lun et al. [16], while at the same time avoiding the danger of getting stuck in the local minima of topology space as in the case of classical pruning and incremental learning techniques. The algorithm of self-evolving network is summarised in the following steps:

1. Initialize a population of N neural networks with a single hidden node and randomly generated set of synaptic weights and connection parameters. Regard the population as particle swarm of N size with each network as a particle.
2. Select the best particles and update their positions for few iterations using BP algorithm.
3. Evaluate the fitness of each particle and update the best particle and global positions.
4. Use PSO to update the weight vector of each particle.
5. Use JPSO to update the binary connection parameters of each particle.
6. Update the particle best position and the best swarm position.
7. Use PSO/JPSO to update particle co-ordinates for certain number of iterations in the following sub-steps:
   a. Use PSO to update the weight vector of each particle
   b. Use JPSO to update the binary connection parameters of each particle.
   c. Update the particle best position and the best swarm position.
8. If convergence is sufficient then stop. Else continue.
9. Reset randomly the binary and continuous parameters of duplicate particles. Also, reset in the same manner, the binary parameters of certain fraction of the swarm with poor fitness.
10. Select best particles and update their continuous parameters using some steps of BP. If the training is satisfactory go to step 11. Else continue.
11. If number of iterations < maximum then go back to step 3. Else continue.
12. Generate N particles with one additional node over the current number of nodes. Replace all current particles with newly generated particles while retaining the current particle best positions (topology and synaptic weight). Then go back to step 3.

13. Terminate algorithm and return result.

Furthermore, a parallel swarm population with of fully connected networks but with the same number of nodes is optimized alongside the swarm with partial connections. The purpose of the fully connected swarm is to assist the partially connected swarm in the search for best network. The partially connected swarm can therefore learn from the fully connected swarm whenever the best swarm position in later is more accurate than the former.

Activation function
The choice of suitable activation function is pivotal to a successful development of neural networks. Sigmoid function has been the most widely used model for ANN development due to its stability. However, despite its popularity, it is not the optimum for all circumstances [19][20]. In this research, a combination of linear and product unit functions are used as processing functions. The idea behind selecting the two functions is to come up with a relatively simple and tractable for the relationship between input and output parameters at the end of the network training the processing function used is expressed in the following equation as:

\[
f(x) = k_c f(k_w x) + k_c \prod_i c_i
\]  

where \( n \) is the number of inputs; \( c_i \) is an adaptive coefficient, while \( k_i \) is a binary coefficient; \( x \) is the vector of inputs to the node; \( w \) represents the vector of synaptic weights of input signals. The binary coefficient assumes the value of 0 when function is switched off and a value of 1 when the function is turned on. In the training process the binary coefficient is updated together with connection parameters, while the adaptive parameter is updated alongside the synaptic weights using PSO-BP hybrid algorithm. In this manner, the topology, the synaptic weights and the activation functions are simultaneously optimized.

III. NETWORK DEVELOPMENT

A. Suction caisson data
The data used to in this research consist of 62 pull out test data sets compiled by Rahman et al. [6] from various sources in the literature. The data consists of caissons of various dimensions embedded into clayey soils, subject to pull-out forces in vertical, horizontal and inclined directions. Table 1 contains the database summary.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Training</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L/d )</td>
<td>Average value</td>
<td>1.56</td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>0.7818</td>
</tr>
<tr>
<td></td>
<td>Range</td>
<td>0.23 - 4</td>
</tr>
<tr>
<td>( s_c )</td>
<td>Average value</td>
<td>12.1</td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>10.3187</td>
</tr>
<tr>
<td></td>
<td>Range</td>
<td>1.8 - 38</td>
</tr>
<tr>
<td>( T_s )</td>
<td>Average value</td>
<td>0.0024</td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>0.0092</td>
</tr>
<tr>
<td></td>
<td>Range</td>
<td>1E-05 -0.04</td>
</tr>
<tr>
<td>( \Theta )</td>
<td>Average value</td>
<td>67.7</td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>37.594</td>
</tr>
<tr>
<td></td>
<td>Range</td>
<td>0 – 90</td>
</tr>
<tr>
<td>( D/L )</td>
<td>Average value</td>
<td>0.0781</td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>0.1883</td>
</tr>
<tr>
<td></td>
<td>Range</td>
<td>0 - 0.69</td>
</tr>
<tr>
<td>( q_e )</td>
<td>Average value</td>
<td>87.67</td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
<td>79.9813</td>
</tr>
<tr>
<td></td>
<td>Range</td>
<td>12.9 - 387.2</td>
</tr>
</tbody>
</table>

B. Input parameters
The pull-out capacity of suction caisson in embedded in clay deposit depends on various parameters such as undrained shear strength in the case of clays \( (s_c) \), depth of embedment \( (L) \), caisson diameter \( (d) \), the direction of pull-out force \( (\theta) \), depth of load application from the top of the caisson \( (D) \) and the non dimensional loading rate parameter \( (T_s) \), which is a function of soil permeability and the velocity of pull-out. The aforementioned parameters are organised into a set of five parameters which control the ultimate capacity of suction anchor \( (q_e) \) as represented by equation (7).

\[
q_e = f \left( \frac{L}{d}, \frac{D}{L}, T_s, s_c, \Theta \right)
\]  

(7)

The parameters on the right hand side of equation 5 serve as inputs to the network, while the ultimate resistance to pull-out is the output of the network.

C. Networks Training and validation
The database was partitioned into training and testing sets. A total of 37 data sets were used for training, while the remaining 25 sets were reserved for testing. To enhance the ability of the network to generalize, the data is split in such a way that both the training and testing data, in a statistical
sense, belong to the same population. For the purpose of comparison, several network the conventional BPN network was also trained, alongside the proposed self-evolving network. The training was brought to a termination when the quality of prediction cease to improve with further training effort with the view to avoiding over-fitting. The parameters used in assessing the prediction quality in the case of both training and testing are the root mean square error (RMSE) and the coefficient of determination ($R^2$).

The optimized network can be represented more simply by the following empirical relationship:

$$q_u (kPa) = 21.2801 \left( \frac{L}{d} \right)^{-0.1018} s_{uv}^{1.0474} T_i^{0.3357} \left( 1 + \sin \theta \right)^{0.3357} \left( 1 + \frac{D}{L} \right)^{-0.3357}$$

$$- 3.6474 \left( \frac{L}{d} \right)^{-1.0874} (1 + \sin \theta)^{1.0874} + 150.517 \left( \frac{L}{d} \right) - 31.891 \theta + 428.241 T_i + 142.791 \left( 1 + \sin \theta \right) - 316.42$$

(5) (8)

Figures 5(a)-(b) display the prediction results of the optimized network plotted against the training and testing data respectively. It can be clearly seen from the figures that the network gives a good correlation with both training and testing data. Most data points seem to fall within 15% envelope. The performance of the optimized network is compared with sigmoid network (BPN), product-unit network (PUNN) and a fully connected network with a combination of linear and product unit processing functions in Table 2. The proposed model seems to outperform all the networks considered with the highest value of $R^2$ (0.9810) with respect to testing data despite being the smallest network. This shows the proposed algorithm is capable of knocking out redundant nodes and synaptic links that could undermine generalization. It is also noteworthy that BPN had to use almost twice the number of parameters used the self-evolving network but still does not achieve accuracy of the later.

<table>
<thead>
<tr>
<th>Type of network</th>
<th>No of nodes</th>
<th>Number of network parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-evolving</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>BPN</td>
<td>4</td>
<td>28</td>
</tr>
<tr>
<td>PUNN</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>Lin+PUNN</td>
<td>2</td>
<td>18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type of network</th>
<th>Training</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-evolving</td>
<td>N-RMSE</td>
<td>0.025704</td>
</tr>
<tr>
<td></td>
<td>$R^2$</td>
<td>0.9851</td>
</tr>
<tr>
<td>BPN</td>
<td>N-RMSE</td>
<td>0.01644</td>
</tr>
<tr>
<td></td>
<td>$R^2$</td>
<td>0.9939</td>
</tr>
<tr>
<td>PUNN</td>
<td>N-RMSE</td>
<td>0.031641</td>
</tr>
<tr>
<td></td>
<td>$R^2$</td>
<td>0.9774</td>
</tr>
<tr>
<td>PUNN+Lin</td>
<td>N-RMSE</td>
<td>0.029101</td>
</tr>
<tr>
<td></td>
<td>$R^2$</td>
<td>0.981</td>
</tr>
</tbody>
</table>
To further assess the quality of predictions, the ratio of predicted capacity to measured capacity ($\lambda = \frac{q_{\text{predicted}}}{q_{\text{measured}}}$) is used. The mean ($\mu_\lambda$) and standard deviation ($\sigma_\lambda$) of ratio $\lambda$ give a great deal of insight about the reliability of model prediction. The mean value of $\lambda$ ratio indicates whether a model, on average, underestimates or overestimates the value in question. The standard deviation gives a measure of scatter in the prediction. A perfect model with 100% accuracy will have a mean value of 1.0 and a standard deviation of zero. From the bar chart in Figure 6, it can be seen that the optimized model gives the best estimate of ultimate capacity on average. The model has a slightly higher value of scatter than PUNN model. However, on the overall the optimized model yields the best result as the PUNN over estimates the capacity by over 25% (against the optimized model with only 5% overestimation).

Figure 6: Comparison of the models based on the mean and standard deviation of $\lambda$ ratio

To examine the influence of various parameters involved in the modeling of ultimate capacity, a sensitivity analysis is carried out by removing a parameter from the input set and evaluating the model performance without the parameter. The procedure is repeated until all input parameters considered are covered. The results are shown in Figure 7. It can be seen from the figure that the soil shear strength $su$ is the most significant parameter affecting the pull-out capacity. The least significant variable is the loading factor $Tk$.

Figure 7: Model sensitivity to various input parameters

IV. CONCLUSIONS
The simultaneous optimization of topology and synaptic weights of neural networks is a desirable but highly challenging task. While the traditional methods are inefficient, the bio-inspired population based algorithms are lacking in computational efficiency due to the random generated topology population with possibly many redundant connections and nodes. In this paper, a self-evolving network capable of growing from small to more complex network is developed. The key features of the algorithm are the ability to grow from a very small network to a complex without a loss of information while maintaining the capability of exploring the search space.

The proposed algorithm is implemented to predict the ultimate pull-out capacity of suction caisson penetrating into clay. The soil shear strength, the caisson’s geometry, the loading conditions are used as inputs to the model. Based on the performance comparisons, the proposed model, with smaller network size, gives a more reliable estimate of ultimate capacity of suction anchors than BPN, PUNN and the combination of PUNN and linear models.

REFERENCES


