Optimal Neuron Selection and Generalization: NK Ensemble Neural Networks

Darrell Whitley[1(✉)], Renato Tinós[2], and Francisco Chicano[3]

[1] Colorado State University, Fort Collins, CO 80523, USA
whitley@colostate.eu
[2] University of São Paulo, Ribeirão Preto, SP, Brazil
[3] University of Málaga, Málaga, Spain

Abstract. This paper explores how learning can be achieved by turning on and off neurons in a special hidden layer of a neural network. By posing the neuron selection problem as a pseudo-Boolean optimization problem with bounded tree width, an exact global optimum can be obtained to the neuron selection problem in $O(N)$ time. To illustrate the effectiveness of neuron selection, the method is applied to optimizing a modified Echo State Network for two learning problems: (1) Mackey-Glass time series prediction and (2) a reinforcement learning problem using a recurrent neural network. Empirical tests indicate that neuron selection results in rapid learning and, more importantly, improved generalization.

1 Introduction to Optimal Neural Selection

Programmed cell death, also known as neuronal apoptosis, is known to be an important part of normal brain development and mechanisms behind neuronal apoptosis have been extensively studied [12,15]. Along with synaptic pruning [3], neuronal apoptosis helps to shape the size and configuration of different neural processing centers in the brain. This is also thought to represent a very basic form of learning. Thus, it is natural to ask what are the benefits of neuron selection and how might neuron selection be utilized in artificial neural networks.

The proposed method converts a form of the neuron selection problem into a k-bounded pseudo Boolean optimization problem, with the goal of identifying useful combinations of neurons. A k-bounded pseudo-Boolean optimization problem [2] can be expressed in the following form:

$$f(\mathbf{x}) = \sum_{i=1}^{M} f_i(\mathbf{x}) \tag{1}$$

where $\mathbf{x} \in \{0,1\}^N$ is a bit vector, each subfunction f_i can output any real value, and $f_i(\mathbf{x})$ is evaluated using a subset of k bits drawn from the bit vector $\mathbf{x}$. Each subfunction f_i identifies which bits are the correct inputs to f_i. MAX-kSAT is a classic example of a k-bounded Boolean optimization problem, where

© Springer Nature Switzerland AG 2018
A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 449–460, 2018.
https://doi.org/10.1007/978-3-319-99259-4_36

each subfunction f_i corresponds to a clause that evaluates to 0 or 1. Spin glass systems and NK Landscapes are also well known k-bounded pseudo Boolean optimization problems.

In this paper, we will restrict our attention to neural networks with a single output neuron that learn a single continuous real valued output; however, the learning method can generalize to multiple outputs. The bit vector $\mathbf{x} \in \{0,1\}^N$ will be used to indicate if a neuron should be turned on or turned off.

In order to create M subfunctions, the single output neuron is converted into an ensemble of M output neurons, all of which attempt to learn the same task. Furthermore, only a subset of other neurons in the neural network (k to be precise) will be allowed to connect to a particular output neurons. Thus, optimizing Eq. 1 results in the selection of a subset of neurons from vector $\mathbf{x}$ that contribute in a positive fashion to an ensemble of M outputs attempting to learn the same task. The neuron selection method proposed here only acts on the set of neurons that are directly connected to an output neuron.

Each subfunction f_i might minimize the mean squared error under supervising learning, or it might maximize a performance metric in the case of reinforcement learning. Because the problem is posed as a k-bounded pseudo-Boolean optimization problem, each of the output neurons (corresponding to a subfunction f_i) receives input from only k other neurons. If k neurons were randomly selected to connect to an output f_i, it would probably be necessary to use a heuristic method to optimize the neuron selection problem. However, there are advantages to choosing a localized and structured pattern when connecting neurons to outputs. In the current paper, neurons are connected to outputs in such a way that the neuron selection problem can be solved in $O(N)$ time using dynamic programming. The resulting solution is globally optimal relative to the starting architecture, and the input vector $\mathbf{x}$. Obviously, different initial architectures would nevertheless yield different results.

We apply this new learning method to two problems. The first problem is the Mackey-Glass time series prediction problem [10]. The second problem is the reinforcement learning problem of balancing two poles on a cart while providing only cart position and the two pole angles as input [20]; this configuration makes it necessary to learn to compute velocity information and makes this classic control problem much more difficult. For both learning problems, we utilize a recurrent neural network in the form of an Echo State Network.

Echo State Networks are one form of *reservoir computing* [13,14]. Reservoir computing networks use a reservoir of sparsely and recurrently connected artifical neurons that have randomly generated weighted connections. Both the input neurons and output neurons are outside of the reservoir. The weights inside of the reservoir of neurons are not adjusted by learning. To determine which neurons are useful, learning is typically used to adjust the weights that connect artificial neurons in the reservoir to the outputs. In our experiments, we use neuron selection to determine which neurons connected to the reservoir are useful.

For the Mackey-Glass problem, neuron selection improves generalization using similar computation time when compared to the standard Echo State

Network. For the two pole balancing problem with no velocity inputs, neuron selection learns more rapidly and produces dramatically better generalization than any other method that has been reported in the literature. It achieves these results with no policy iteration and no back propagation. Neuron selection is the *only* form of learning that is utilized.

Although we use an Echo State Network as the foundation for our experiments, in principle the same technique might be applied to multi-layered perceptrons or deep learning networks. It therefore provides a new means of automatically configuring a neural network architecture to fit a particular problem.

2 Optimization by Dynamic Programming

We will very briefly outline how and when dynamic programming can used to optimize k-bounded pseudo-Boolean functions. We construct a Variable Interaction Graph, G, to model the interaction between variables in a k-bound pseudo-Boolean optimization problem. If two variables x_q and x_j appear together in subfunction f_i then there is an edge between x_q and x_j in G. We next define the concepts of tree width and tree decomposition of a graph [6].

Definition 1. *[6] A tree decomposition of any graph $G(V, E)$ is a pair $D = (S, T)$ where $S = \{X_i, i \in I\}$ is a collection of I subsets of the vertices of G and T is a tree with one node for each subset in the collection S, such that:*

1. $\bigcup_{i \in I} X_i = V$,
2. *for all the edges $(u, w) \in E$ there exists a subset $X_i \in S$ such that both u and w are in X_i,*
3. *for each vertex v, the set of nodes that contain v, $\{i | v \in X_i\}$, form a subtree of T.*

The tree width, denoted by w, of the decomposition $D = (S, T)$ is given by $w = max_{i \in I} (|X_i| - 1)$.

Dynamic programming can be used to find the global optimum of any pseudo-Boolean optimization problem (i.e., Eq. 1) in time $O(2^w N)$, where w is the tree width of graph $G(V, E)$ [4] and N is the number of variables. In effect, a tree decomposition D provides an ordering of the variables and of the subfunctions so that only w variables are *active* at a time. A variable is active if it appears in a subfunction that is currently being probed by dynamic programming. Once a variable is active, it stays active until a global solution is found for all of the subfunctions that utilize that variable. In general, for NP-Hard problems (such as MAXSAT) the runtime cost of dynamic programming is exponential. However, problems where the tree width is bounded by a constant can be solved in linear time. Thus, there is an advantage in creating a neuron selection architecture that limits the size of the tree width of the variable interaction graph. In the next section we show that if the neuron connections are sufficiently localized and regular, the tree width is automatically limited in size.

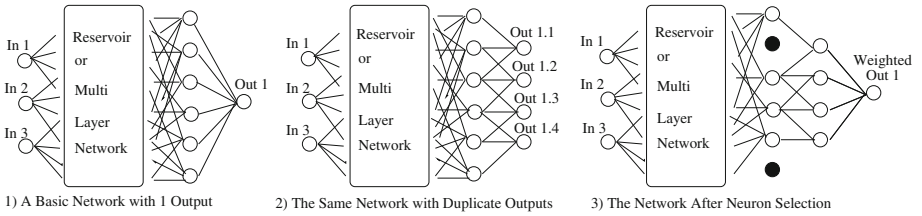

Fig. 1. On the left is a basic network with one output. In the middle figure, the one output neuron is replaced by an ensemble of output neurons, all with the same target output and learning objective. Each output neuron in the ensemble is connected to exactly $k = 3$ neurons in the probe filter layer ($k \ll N$). Normally, there are N neurons in both the probe filter layer and an ensemble of N outputs. Here, to aid visualization, there are 6 neurons in the probe filter layer and only 4 outputs in the ensemble (x_1 is not adjacent to x_6), and $k = 3$. Each output Out 1.i is different in performance because it connects to different neurons in the probe filter layer. The neurons of the probe filter layer are turned on or off by optimizing binary vector $\mathbf{x}$ so as to maximize performance summed across all outputs. In the figure on the right, the black neurons have been turned off, optimizing $\mathbf{x}$ and modifying the architecture. The ensemble of outputs are again collected into a single output weighted by the relative performance of each output in the ensemble.

3 Converting a Neural Network into an NK Landscape

One type of pseudo-Boolean optimization problem that automatically controls for tree width is the *Adjacent NK Landscape* [11,19]. N refers to the number of Boolean variables in vector x, $M = N$ is the number of subfunctions, and $k = K + 1$ is the number of variables that appear in each subfunction f_i. In an Adjacent NK Landscape, Boolean variable x_i appears in subfunction f_i as well as the variables $x_{i+1}, x_{i+2}, \ldots x_{i+K}$. If the Adjacent NK Landscape allows variables to wrap around such that x_1 and x_N are adjacent, then the tree width is $2K$. If variables do not wrap and x_1 and x_N are not adjacent in the Adjacent NK Landscape, then the tree width is K [21].

We will use Fig. 1 to explain how the neuron selection problem can be expressed as a k-bounded pseudo-Boolean optimization problem with bounded tree width. On the left in Fig. 1, we start with a basic neural network with a single output neuron. In this example, there are 3 inputs and 1 output. We assume there is a hidden layer of neurons that feed into the output, or if a reservoir is used, we create a hidden layer of N neurons that feed into the output. We will refer to this layer as the *probe filter* layer. This layer probes the other neurons in the network that feed into this final hidden layer. The probe filter layer must directly connect to the output layer.

We will refer to our architecture as an *NK Ensemble Network*. The *NK Ensemble Network* has an ensemble of N outputs, and each output receives inputs from k neurons in the probe filter layer. All weights in the NK Ensemble Network remain fixed during neuron selection. Only the bit vector $\mathbf{x} \in \{0,1\}^N$ is

optimized. If $x_i = 1$, the i^{th} neuron of the probe filter is turned on, i.e., its activation is used as input for neurons in the output layer connected to it; if $x_i = 0$, the i^{th} neuron is turned off.

We denote the evaluation of the i^{th} output neuron as $f_i(\mathbf{x})$. The subfunction f_i automatically accesses the correct k bits when passed an input of length N. It is also convenient to assume that f_i can take an input of length k or length $N = |\mathbf{x}|$. For example, assume $k = 3$ and that $f_i(\mathbf{x}) = f_i(011)$; this means that $k = 3$ probe filter neurons feed into output neuron i, but the 1st neuron (numbering bits left to right) is currently turned off in $\mathbf{x}$. All of the inputs to f_i must be evaluated once (and only once). Thus, for $k = 3$ we must evaluate $f_i(000), f_i(001), \ldots f_i(111)$. This is done by turning off the correct neurons in the probe filter layer, then doing an evaluation that processes the training data just once, or (e.g., for reinforcement learning) a simulation is used to evaluate the network performance. This means that each subfunction requires 2^k presentations of the training data, or 2^k performance based evaluations. This needs to be done for all N subfunctions.

At most $2^k N$ presentations of the training data are needed to convert the neuron selection problem into an Adjacent NK Landscape. For recurrent networks where the output at time t impacts the input at time $t + 1$, the total number of online evaluations will be exactly $2^k N$; this represents the worst case runtime cost, which is still $O(N)$ for fixed k. For other classes of learning problems this cost might be reduced because subfunction f_i might be evaluated simultaneously and in parallel with other subfunctions (e.g., f_i and f_{i+k} do not interact) using the same presentation of the training data.

Because k is a small constant, each function f_i can be expressed as a lookup table with 2^k entries. Dynamic programming can then be done offline because the neuron selection objective function is now fully captured by the lookup tables. In practice, we have found that the runtime cost of dynamic programming is less than 1% of the entire computation for small k (e.g., $k \leq 6$) and usually takes about the same amount of time as a single feed-forward pass over the training data.

The algorithm for neuron selection follows the 3 illustrations in Fig. 1.

(1) Step 1. Start with a basic network. The weights in the network might be optimized with a weight training algorithm (e.g., back propagation), or weights might be generated randomly. The network must include a probe filter layer, and only neurons in the probe filter layer connect to the output.
(2) Step 2. Assume there are N neurons in the probe filter layer: create an ensemble of N outputs. Let x_i reference the i^{th} neuron in the probe filter. Output neuron "Out 1.i" receives inputs from neurons x_i to x_{i+K}. The performance of output neuron Out1.i is condensed into a single number, P_i, where $f_i(x) = P_i$. This makes is possible to storage each function f_i as a look-up table of size 2^k.
(3) Step 3. Optimize the function: $f(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} f_i(\mathbf{x})$ using dynamic programming. Use the optimal solution $\mathbf{x}*$ to select neurons in the probe filter

layer. In this formulation we average over the subfunctions but this has no impact on the form of the optimization problem.

Let z be an input to the neural network. Let $S_{out.i}(z)$ denote the state of ensemble output neuron (e.g. Out 1.i in Fig. 1) after input z is propagated through the network; let $Out_{ensemble}(z)$ denote the weighted output obtained by combining the ensemble:

$$Out_{ensemble}(z) = \frac{1}{N}\sum_{i=1}^{N} \alpha_i S_{out.i}(z) \qquad \text{where} \qquad \alpha_i = \frac{f_i(\mathbf{x}^*)}{\sum_{i=1}^{N} f_i(\mathbf{x}^*)} \tag{2}$$

The weighting vector α is calculated after optimizing the vector $\mathbf{x}$. Thus α_i depends on the performance associated with $f_i(x*)$ and output neurons with better results have higher weights.

4 Experimental Results

All of the experiments in this paper use "Echo State Networks" as a foundation. One motivation for using Echo State Networks is that the neurons in the reservoir have randomly generated weights. Schiller and Steil [17] show that when gradient methods are used to train recurrent neural networks, most of the weight changes occur in the weights that connect to outputs, even if the methods are being used to change all of the weights in the network. We explore the idea that neural networks can be trained using little or no weight optimization.

The term "NK Ensemble Network" will be used to denote networks that have been enhanced by neuron selection.

4.1 Problem One: Mackey-Glass Time Series Prediction

The Mackey-Glass time series problem is a supervised learning problem and a classic benchmark for chaotic time series prediction. The original Echo State Network was successfully applied to this problem [10]. The vector of weights $\mathbf{w_{out}}$ in a given output neuron can be trained by solving a system of linear equations:

$$\mathbf{y_d} = \mathbf{H}\mathbf{w_{out}} \tag{3}$$

where $\mathbf{H}$ is a matrix composed by the inputs of the output layer for each training example and $\mathbf{y_d}$ is the vector of desired values for the output neuron. Equation 3 can be solved [14] by:

$$\mathbf{w_{out}} = (\mathbf{H}^{\mathrm{T}}\mathbf{H})^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{y_d} \tag{4}$$

In order to avoid numerical instability, a regularization term can be added to the term inside the parentheses in Eq. 4.

The Echo State Network for this problem has only one input and only one output. These respectively correspond to the points of the time series at instants t and $t + 1$ (the Mackey-Glass time series with delay 17 was used). There are

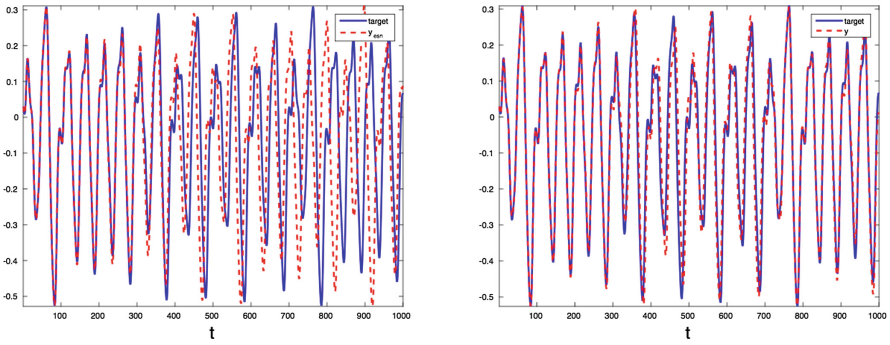

Fig. 2. The leftmost figure illustrates the performance of the standard echo state network for the Mackey-Glass problem. The rightmost figure illustrates the performance of the NK ensemble network. In both figures, the desired output is denoted by the solid blue line, and the actual output is denoted by the dashed red line. The NK echo state network error is 4 times lower than the standard echo state network. (Color figure Online)

two hidden layers between the reservoir and the output. The weights associated with the two hidden layers were obtained using the same weight optimization reported by Jaeger [10]. The inputs of the neurons of the hidden layer 2 receive inputs from 90% of the neurons in hidden layer 1. A bias neuron is used. A linear activation function is used in the neurons of the hidden layer 2 (in order to apply Eq. 4 to train the output weights) and output layer. The neurons of the reservoir use the hyperbolic tangent activation function.

The NK Ensemble Network uses exactly the same standard configuration except an ensemble of N output neurons is used. The second hidden layer functions as the probe filter layer. The weights between the probe filter layer and the ensemble of outputs are generated randomly, then rescaled so that the sum of the weights is equal to 1.

The output neurons of the NK Ensemble Network for inputs z are given by:

$$S_{out.i}(z) = \sum_{q=1}^{N} w^s_{q,i} S_q(z) x_q \tag{5}$$

where $S_q(z)$ is the output of neuron q of the probe filter layer for input z, x_q indicates if the neuron q is turned on (1) or off (0), and the scaled weight $w^s_{q,i}$ is given by:

$$w^s_{q,i} = \frac{w_{q,i}}{\sum_{j=1}^{N} w_{j,i} x(j)} \tag{6}$$

where $w_{q,i}$ are randomly generated in the interval between 0.0 and 1.0.

The weights of the output neurons are then adapted using Eq. 4 (for each output neuron and each combination given by vector $\mathbf{x}$). Finally, look-up tables are generated for each subfunction f_i. Next, the bit vector $\mathbf{x} \in \{0,1\}^N$ is optimized using dynamic programming.

The initial 1000 time steps of the series are used to stabilize the reservoir before the training phase for each learning algorithm. The standard Echo State Network is trained for 2000 time steps of the Mackey-Glass series and tested for additional 300 points. During the test phase, the input of the network at time t is given by the output of the network at time $t-1$.

Learning for the NK Ensemble Network is broken into three phases. In Phase 1, the weights are adapted for 1800 time steps in exactly the same way in which they were for the standard Echo State Network. In Phase 2, the subfunctions f_i are generated. Each subfunction is evaluated for 2^k configurations, and each configuration is evaluated for 200 time steps. This can be thought of as a validation phase where the optimization of the probe filter layer corresponds to a type of model selection. The input of the network at time t is given by the output of the network at time $t-1$. In the validation phase, the terms f_i are computed by the mean squared error for the NK Ensemble Network during 200 time steps. The vector $\mathbf{x}$ is then optimized by the dynamic programming procedure; the cost of the dynamic programming is minimal and less than 1% of the total runtime. In Phase 3, the NK Ensemble Network is tested for generalization.

Both the standard Echo State Network and the NK Ensemble Network were allocated 2000 time steps of the data for learning and 1000 time steps for testing. (The 2000 steps for the NK Ensemble network includes the time needs for dynamic programming.) Both networks used exactly the same reservoir. Both networks were trained and tested using a sample size of 30. During testing, generalization was measured by the function:

$$g = \frac{1}{1 - e_{mse}}$$

where e_{mse} is the mean squared error. $g = 1.0$ represents perfect generalization.

Both networks yield reasonably good prediction during the first 300 steps of testing. However, an examination of Fig. 2 shows that the standard Echo State Network yields poorer performance after time step 300 during the testing phase: the predictions become increasingly worse with time. The NK Ensemble Network continues to make good predictions across all of the testing phase. Overall, the error of the standard Echo State Network is 4 times larger than the error of the NK Ensemble Network. Thus, the NK Ensemble Network is able to improve generalization with little or no additional training cost.

4.2 Problem Two: Double Pole Balancing Without Velocity Inputs

The NK Ensemble Network is next tested on the double pole balancing problem without velocity information [20]. No back propagation was used. No policy iteration was used. The only form of learning was neuron selection. All of the weights in the network were generated randomly.

The reservoir utilizes 60 neurons, with recurrent connections between neurons. Each neuron in the reservoir has recurrent connections to 10% of the neurons in the reservoir. All weights and bias of the NK Ensemble Network are

fixed, being randomly generated between $[-0.6, 0.6]$. After the initialization, the recurrent weights in each reservoir are scaled with a spectral radius equal to 0.95. All neurons use the hyperbolic tangent function as the activation function.

When no velocity information is provided, this problem is difficult; it has also been widely studied [5,7–9,18]. The 3 inputs to the artificial neural network at step t are the scaled cart position and the angles of the two poles:

$$\mathbf{u}(t) = [p_c(t)/p_c^{max}, \quad \theta_1(t)/\theta_1^{max}, \quad \theta_2(t)/\theta_2^{max}]^{\mathrm{T}}$$

where $p_c(t)$ is the cart position, $\theta_i(t)$ is the angle of the i-th pole, and p_c^{max} and θ_i^{max} are the maximum allowed values used to scale the inputs between -1 and $+1$. All neurons use the hyperbolic tangent function with outputs between -1 and $+1$ as the sigmoidal squashing function.

The following objective function has been used by a number of researchers [5,8,9,18].

$$f = t/t_{max} + 9f_{stable}$$

$$f_{stable} = \begin{cases} 0, & \text{if } t < 100 \\ \frac{0.75}{\sum_{i=t-100}^{t}(|x_c(i)|+|\dot{x_c}(i)|+|\theta_1(i)|+|\dot{\theta}_1(i)|)}, & \text{otherwise,} \end{cases}$$

where t is the number of time steps that the system is successfully controlled (up to a limit of $t_{max} = 1000$ steps).

The output in the problem posed in this paper is continuous, allowing for greater control. The force (in Newtons) applied to the cart at iteration t when evaluating the i-th output neuron is given by:

$$action(t) = 10S_{out.i}(\mathbf{u}(t)) \tag{7}$$

The state of the neuron selection vector $\mathbf{x}$ is included in the calculation of S_i (Eq. 5). The track length is given by $p_c \in [-2.4, 2.4]$ meters; beyond this range the cart crashes into the ends of the track. The system must keep both poles within $\theta_i \in [-36, 36]$ degrees of vertical. The function f_1 indicates how long the cart and pole system has avoided a failed state (where a pole falls, or the cart crashes). An overall evaluation greater than 1.0 generally means that the system avoided failure for t_{max} time. However, because $t_{max} = 1000$ is small, a bang-bang control strategy might be learned so that even if the controller avoids failure for t_{max} time steps, the system will become increasingly unstable and eventually fail when the system is run for more than t_{max} time steps. The second function f_{stable} indicates the stability of the system during the last 100 time steps if $t \geq 100$. A higher value of f_{stable} means that the system is staying close to the ideal state: close to the center of the track, with small pole angles close to vertical (zero), and with low velocities.

During learning, the system always starts from the state $p_c(0) = \theta_2(0) = \dot{p_c}(0) = \dot{\theta}_1(0) = \dot{\theta}_2(0) = 0$ and $\theta_1(0) = 4.5°$. The mass of cart is 1 kg, the mass of pole 1 is 0.1 kg, the mass of pole 2 is 0.01 kg: length of pole 1 equal to 1 m,

length of pole 1 equal to 0.1 m, coefficient of friction of the cart on the track is 0.0005, the coefficient of friction of the poles equal to 0.000002 [5]. The 4th order Runge-Kutta method with integration step equal to 0.01 was used.

4.3 Comparative Results

Learning was successful 100% of the time across all experiments. To test generalization, the final network was evaluated 625 times, each time with different initial settings for cart position, cart velocity, pole 1 angle, and pole 1 velocity. The angle and velocity for pole 2 are set to zero. The combination of five different initial settings for each variable is considered: 5, 25, 50, 75, and 95% of a reduced range of the variables. With 5 settings and 4 variables, $5^4 = 625$. The evaluation of the generalization test counts the number of positions from which the system is successfully controlled for 1000 steps. This test of generalization has been widely used for the last 20 years [5,7–9].

In Table 1, we report results for the NK Ensemble Network for several different configurations, as well as previously published results. There appears to be no new significant results since 2008.

Gomez, Schmidhumber and Miikkulainen [8] have shown that a wide range of standard reinforcement learning methods do not work well on the problem of balancing two poles on a cart given no velocity information. They used Q-learning with a Multi-Layer Perceptron that mapped state-action pairs to Q-values. They also compared to methods such as Sarsa(λ) with Case Based Function Approximators as well as Sarsa(λ) with a Cerebellar Model Articulation Controller [16]. They concluded these methods were less effective and less efficient compared to neuroevolution based methods such as NEAT [18], ESP [7] and CoSyNE [8]. In this paper and the above studies, a continuous output is learned. The most recent reinforcement work on pole balancing [1] looked at a discrete "bang-bang" controller and provided velocity information as inputs; this work also did not test for generalization.

The NK Ensemble Networks included networks with N = 20 and N = 100, and K = 2, 3, 4, 5. Using just 320 evaluations, the NK Ensemble Networks with $N = 20$ and $K = 3$ yields an average generalization of 304 successes from the 625 possible start states. This level of generalization is similar to the best results previously reported in the literature as reported in Table 1. Increasing N and K improved generalization at the cost of additional evaluations. The best generalization was achieved by setting $N = 100$ and $K = 4$ and then selecting only the "Top 20" best output neurons (based on α_i from Eq. 2) to be included in the ensemble. This was done at no additional runtime cost. This configuration used 3200 evaluations, but the NK Ensemble Network was able to successfully balance the double pole from 490 of the 625 start states on average with a relatively low standard deviation. These generalization results greatly improve on results previously reported in the literature. A runtime analysis shows that 99% of the runtime was spent on feedforward evaluations of the neural network; less than 1% of the time was spent on the dynamic programming optimization.

Table 1. Evaluation results for the NK ensemble network with different values of N and K. The results are also compared to other results in the literature.

Algorithm		Evaluations	Generalization
CE 1996, reference [9]		840,000	300
ESP 1999, reference [7]		169,000	289
ESP 2008, reference [8]		26,342	Not Given
NEAT 2002, reference [18]		33,184	286
NEAT 2008, reference [8]		6,929	Not Given
CoSyNE 2008, reference [8]		3,416	Not Given
NK Ensemble Network,	N=20, K=2	160	229 ±160 s.d.
NK Ensemble Network,	N=20, K=3	320	304 ±154 s.d.
NK Ensemble Network,	N=20, K=4	640	321 ±151 s.d.
NK Ensemble Network,	N=20, K=5	1,280	377 ±126 s.d.
NK Ensemble Network,	N=100, K=2	800	323 ±115 s.d.
NK Ensemble Network,	N=100, K=3	1,600	396 ±108 s.d.
NK Ensemble Network,	N=100, K=4	3,200	437 ±83 s.d.
NK Ensemble Network, "Top 20"	N=100, K=2	800	450 ±79 s.d.
NK Ensemble Network, "Top 20"	N=100, K=3	1,600	478 ±56 s.d.
NK Ensemble Network, "Top 20"	N=100, K=4	3,200	490 ±48 s.d.

5 Conclusions

This paper explores the idea that learning can be achieved by turning on and turning off neurons in an artificial neural system. By posing the neuron selection problem as a pseudo-Boolean optimization problem with bounded tree width, an exact global optimum can be obtained to the neuron selection problem in $O(N)$ time. In this paper neuron selection is empirically evaluated when used in combination with the Echo State Network. However, the method could be used with other multi-layer networks. It should also be noted that the NK Ensemble Network does not require significant tuning to achieve the "right network configuration" in order to learn.

On the Mackey-Glass time series prediction problem the NK Ensemble Network improved generalization and reduced variance across runs compared to the standard Echo State Network.

The NK Ensemble Network is able to learn the control task of balancing two poles on a fixed track with no velocity information. Learning was 100% successful. No back propagation was used. No policy iteration was used. All of the weights in the network were generated randomly. The only form of learning was neuron selection. Learning was much faster compared to other algorithms. But more important, generalization dramatically improved as N and K were increased, and variance in the generalization results decreased.

References

1. Anderson, C., Elliott, D.: Faster reinforcement learning after pretraining deep networks to predict state dynamics. In: International Joint Conference on Neural Networks (2015)

2. Boros, E., Hammer, P.: Pseudo-boolean programming revisited. Discrete Appl. Math. **123**(1), 155–225 (2002)
3. Chechik, G., Meilijson, I., Ruppin, E.: Neuronal regulation: a mechanism for synaptic pruning during brain maturation. Neural Comput. **11**(8), 2061–2080 (1999)
4. Crama, Y., Hansen, P., Jaumard, B.: The basic algorithm for pseudo-boolean programming revisited. Discrete Appl. Math. **29**(2–3), 171–185 (1990)
5. Dürr, P., Mattiussi, C., Floreano, D.: Neuroevolution with analog genetic encoding. In: Runarsson, T.P., Beyer, H.-G., Burke, E., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 671–680. Springer, Heidelberg (2006). https://doi.org/10.1007/11844297_68
6. Gao, Y., Culberson, J.: On the treewidth of NK landscapes. In: Cantú-Paz, E., et al. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 948–954. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45105-6_106
7. Gomez, F., Miikkulainen, R.: Solving non-Markovian control tasks with neuroevolution. In: IJCAI. Morgan Kaufmann (1999)
8. Gomez, F., Schmidhuber, J., Miikkulainen, R.: Accelerated neural evolution through cooperatively coevolved synapses. J. Mach. Learn. Res. **9**, 937–965 (2008)
9. Gruau, F., Whitley, D., Pyeatt, L.: A comparison between cellular encoding and direct encoding. In: Genetic Programming Conference. Morgan Kaufmann (1996)
10. Jaeger, H., Haas, H.: Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. Science **304**(5667), 78–80 (2004)
11. Kauffman, S.A.: The Origins of Order: Self-organization and Selection in Evolution. Oxford University Press, Oxford (1993)
12. Kristiansen, M., Ham, J.: Programmed cell death during neuronal development: the sympathetic neuron model. Cell Death Differ. (Nature Publishing Group) **21**(7), 1025–1035 (2014)
13. Lukoševičius, M.: A practical guide to applying echo state networks. In: Montavon, G., Orr, G.B., Müller, K.-R. (eds.) Neural Networks: Tricks of the Trade. LNCS, vol. 7700, pp. 659–686. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35289-8_36
14. Lukoševičius, M., Jaeger, H.: Reservoir computing approaches to recurrent neural network training. Comput. Sci. Rev. **3**(3), 127–149 (2009)
15. Roth, K.A., D'Sa, C.: Apoptosis and brain development. Ment. Retard. Dev. Disabil. Res. Rev. **7**, 261–266 (2001)
16. Santamaria, J., Sutton, R., Ram, A.: Experiments with reinforcement learning in problems with continuous state and actions spaces. Adapt. Behav. **6**(2), 163–217 (1998)
17. Schiller, U.D., Steil, J.J.: Analyzing the weight dynamics of recurrent learning algorithms. Neurocomputing **63**, 5–23 (2005)
18. Stanley, K., Miikkulainen, R.: Efficient reinforcement learning through evolving neural network topologies. In: Genetic and Evolutionary Computation Conference (GECCO), pp. 569–577, Morgan Kaufmann (2002)
19. Tomassini, M., Verel, S., Ochoa, G.: Complex-network analysis of combinatorial spaces: the NK landscape case. Phys. Rev. E **78**, 066114 (2008)
20. Wieland, A.P.: Evolving neural network controllers for unstable systems. In: Proceedings of the 1991 International Joint Conference on Neural Networks (IJCNN), vol. 2, pp. 667–673. IEEE (1991)
21. Wright, A.H., Thompson, R.K., Zhang, J.: The computational complexity of N-K fitness functions. IEEE Trans. Evolut. Comput. **4**(4), 373–379 (2000)