Pruning Product Unit Neural Networks

A Ismail† and AP Engelbrecht‡

† Department of Computer Science, University of Western Cape, South Africa,
aismail@uwc.ac.za
‡ Department of Computer Science, University of Pretoria, South Africa,
engel@driesie.cs.up.ac.za

Abstract - Selection of the optimal architecture of a neural network is crucial to ensure good generalization by reducing the occurrence of overfitting. While much work has been done to develop pruning algorithms for networks that employ summation units, not much has been done on pruning of product unit neural networks. This paper develops and tests a pruning algorithm for product unit networks, and illustrates its performance on several function approximation tasks.

I. Introduction

Multilayer neural networks have shown to be successful in many applications, including environments such as engineering, medicine, econometrics, and computer science, to name a few. Application of a neural network (NN) to solve a real-world problem is not a straight forward process, but requires careful planning of the network architecture and training process. With reference to the training process, careful consideration must be given, amongst others, to the type of optimization algorithm used to train the network, the mode of training (batch or on-line), training set manipulation techniques, active learning strategies, and optimal training parameters. Planning of the NN architecture includes selection of the number of layers and the number of neurons in each layer.

Architecture selection is important to ensure good generalization performance. Undersized networks fail to approximate the underlying function that relates inputs to desired outputs, thus causing bad generalization. Oversized networks, on the other hand, have too many degrees of freedom (number of weights), which allows the network to exactly approximate the mapping as reflected in the training set. This causes the network to overfit the training data and to memorize noise components, thereby reducing the generalization ability.

Several techniques have been developed to deal with the architecture selection problem. These include

- regularization, where a penalty is added to the objective function to penalize architectural complexity [1], [2];
- network growing, where the size of the network grows incrementally [3], [4]; and
- pruning, where an initial oversized network is reduced to an optimal size through deletion of weights, input and hidden units [5].

These architecture selection techniques have been developed specifically for networks with summation units (SU). While summation unit neural networks (SUNN) have been very successful, networks that make use of product units (PU) have the added advantage of increased information capacity [6]. That is, smaller product unit neural network (PUNN) architectures can be used than for SUNNs. Several studies have been done to study the training and performance of PUNNs [6]-[9], while architecture selection of PUNNs has not received much attention.

This paper presents a pruning algorithm to determine optimal PUNN architectures. The algorithm is based on the variance nullity algorithm developed by Engelbrecht [10], where the significance of a unit is measured as the sensitivity of the network output to perturbations in that unit.

The rest of the paper is organized as follows: Section II overviews PUNNs. The variance nullity pruning approach is summarized in section III, and the PUNN pruning algorithm is presented. Experimental results are provided in section IV.

II. Product Unit Neural Networks

Standard neural networks (NN) use summation units to compute the net input signal to hidden and output units. For SUs, the net input signal is calculated as the weighted sum of the inputs connected to that unit. Research has shown that these summation unit neural networks (SUNNs) can approximate any continuous function to an arbitrary degree of accuracy, provided that the hidden layers contain a sufficient number of hidden units [11], [12]. However, these networks require a large number of summation units (SUs) when approximating complex functions that involve higher order combinations of its inputs [8]. When approximating polynomials, higher-order combinations of inputs, such as x^3y^7, are often re-

quired. Networks that utilize higher-order combinations of its inputs will greatly reduce the number of processing units required to represent these complex functions [7].

A product unit neural network is such a higher-order network, where the net input is now a product of terms; each term consisting of an input raised to a weight. Advantages of PUNNs are increased information capacity and the ability to form higher-order combinations of inputs. Durbin and Rumelhart determined empirically that the information capacity of product units PUs (as measured by their capacity for learning random boolean patterns) is approximately *3N*, compared to *2N* of a SU network for a single threshold logic function, where N denotes the number of inputs to the network [6]. Durbin and Rumelhart suggested two types of networks incorporating PUs [6]. In the one network type each SU is directly connected to the input units, and also connected to a group of dedicated PUs. The other network consists of alternating layers of product and summation units, terminating the network with a SU. This paper concentrates on three-layer PUNNs where only the hidden layer consists of PUs, while the output layer has SUs. Both layers use linear activation functions.

Under these assumptions, the net input signal to hidden unit y_j for pattern p is calculated as [9]

$$\begin{aligned} net_{y_{j,p}} &= \prod_{i=1}^{I+1} z_{i,p}^{v_{ji}} \\ &= e^{\sum_{i=1}^{I+1} v_{ji} \ln |z_{i,p}|} \\ &= e^{\rho_j} \cos(\pi \phi_j) \end{aligned}$$

where

$$\begin{aligned} \rho_j &= \sum_{i=1}^{I+1} v_{ji} \ln |z_{i,p}| \\ \phi_j &= \sum_{i=1}^{I+1} v_{ji} \mathcal{I}_i \\ \mathcal{I}_i &= \begin{cases} 0 & \text{if } z_{i,p} \geq 0 \\ 1 & \text{if } z_{i,p} < 0 \end{cases} \end{aligned}$$

with I the total number of input units, $z_{i,p}$ the input value of the i-th input unit for pattern p, and v_{ji} the weight between input unit z_i and hidden unit y_j. Note that no bias is included for hidden units, but a distortion unit is included as part of the product. Unit z_{I+1}, with a constant input of -1, refers to the distortion unit, and has the objective to dynamically shape the activation function to more accurately approximate the target function (refer to [13] for a presentation on the distortion unit).

The net input signal to output unit o_k is

$$net_{o_{k,p}} = \sum_{j=1}^{J+1} w_{kj} y_{j,p}$$

where J is the total number of hidden units, $J+1$ refer to a bias unit with a constant input of -1 (the weight leading from the bias unit to the output unit serves as the bias), w_{kj} is the weight between hidden unit y_j and output unit o_k, and $y_{j,p}$ is the activation value of hidden unit y_j for pattern p.

Training of PUNNs present problems due to the increased number of local minima in the search space and the exponential terms in the weight update equations. Local optimization algorithms such as gradient descent therefor fails to train PUNNs in general [9]. Successful training of these networks requires the use of global optimization algorithms such as genetic algorithms and particle swarm optimization [9]. For this paper, particle swarm optimization [14] is used to train PUNNs.

III. Variance Nullity Pruning Algorithm

The variance nullity pruning algorithm of Engelbrecht is based on NN output sensitivity analysis, where the relevance of parameters (input and hidden units) is based on parameter sensitivity information [10]. A variance nullity is computed for each parameter. The objective of the variance nullity measure is to test whether the variance in parameter sensitivity for the different patterns is significantly different from zero. If the latter is not the case, it indicates that the corresponding parameter has little or no effect on the output of the NN over the entire set of patterns presented to the network. A hypothesis testing step uses these variance nullity measures to statistically test if a parameter should be pruned, using the χ^2 distribution.

The variance nullity measure for a single parameter θ_i is a function of the sensitivity, $S^{(p)}_{O\theta,ki}$ of output o_k to changes in parameter θ_i for a single pattern p, where

$$S^{(p)}_{O\theta,ki} = \frac{\partial o_k}{\partial \theta_i^{(p)}}$$

In the case of PUNNs, the sensitivity of output unit o_k to changes in hidden unit y_j is, assuming linear activations,

$$S^{(p)}_{OY,kj} = \frac{\partial o_k}{\partial y_j^{(p)}} = w_{kj}$$

The $S^{(p)}_{OY,kj}$ values are then used by the pruning algorithm to remove irrelevant hidden units (refer to [10] for more detail on the algorithm). For pruning of input units,

the sensitivity with respect to changes in input unit z_i is expressed as

$$\begin{aligned} S^{(p)}_{OZ,ki} &= \frac{\partial o_k}{\partial z_i^{(p)}} \\ &= \sum_{j=1}^{J} w_{kj} \frac{v_{ji}}{\left|z_i^{(p)}\right|} (e^{\rho_j} \cos(\pi\phi_j)) \end{aligned}$$

The statistical nullity in parameter sensitivity, Υ_{θ_i}, of a parameter θ_i over patterns $p = 1, ..., P$ is then defined as follows [10]:

$$\Upsilon_{\theta_i} = \frac{(P-1)\sigma^2_{\theta_i}}{\sigma_0^2} \tag{1}$$

where $\sigma^2_{\theta_i}$ is the variance of the sensitivity of the network to perturbations in parameter θ_i, σ_0^2 is a value close to zero and P the number of patterns in the pruning set. The variance in parameter sensitivity, $\sigma^2_{\theta_i}$, is computed as

$$\sigma^2_{\theta_i} = \frac{\sum_{p=1}^{P} (\aleph^{(p)}_{\theta_i} - \overline{\aleph}_{\theta_i})^2}{P-1}$$

where

$$\aleph^{(p)}_{\theta_i} = \frac{\sum_{k=1}^{K} S^{(p)}_{O\theta,ki}}{K}$$

and $\overline{\aleph}_{\theta_i}$ is the average parameter sensitivity over all patterns $p = 1, ..., P$, i.e.

$$\overline{\aleph}_{\theta_i} = \frac{\sum_{p=1}^{P} \aleph^{(p)}_{\theta_i}}{P}$$

The null hypothesis is then defined as

$$\mathcal{H}_0 : \sigma^2_{\theta_i} = \sigma_0^2$$

This hypothesis can however not be used, since equation (1) does not allow $\sigma_0^2 = 0$, and therefor it cannot be hypothesized that the variance in parameter sensitivity over all patterns is exactly zero. To alleviate this problem a small value close to zero is chosen for σ_0^2, and the alternative hypothesis,

$$\mathcal{H}_1 : \sigma^2_{\theta_i} < \sigma_0^2$$

is tested. The variance nullity measure defined in equation (1) has a $\chi^2(P-1)$ distribution in the case of P patterns. The critical value, Υ_c, can therefor be obtained from χ^2 distribution tables, i.e.

$$\Upsilon_c = \chi^2_{v;1-\alpha}$$

where $v = P - 1$ is the number of degrees of freedom and α is the level of significance. Using the critical value defined above, if $\Upsilon_{\theta_i} \leq \Upsilon_c$, the alternative hypothesis $\mathcal{H}_1$ is accepted and parameter θ_i is pruned.

IV. Experimental Results

This section illustrates the performance of the PUNN pruning algorithm. For this purpose, a particle swarm optimization (PSO) algorithm has been used to learn each of the following functions (refer to [9] for the implementation of the PSO):

- The quadratic function $f(z) = z^2$, with $z \sim U(-1, 1)$. The training, test and validation sets consisted of 50 distinct randomly selected patterns.
- The cubic function $f(z) = z^3 - 0.04z$, with $z \sim U(-1, 1)$. The training, test and validation sets consisted of 50 distinct randomly selected patterns.
- The henon time series $z_t = 1 + 0.3z_{t-2} - 1.4z^2_{t-1}$, with $z_1, z_2 \sim U(-1, 1)$. The training, test and validation sets consisted of 200 distinct randomly selected patterns.
- The surface $f(x, y) = y^7x^3 - 0.5x^6$, with $x, y \sim U(-1, 1)$. The training, test and validation sets consisted of 300 distinct randomly selected patterns.
- The paraboloid $f(x, y) = x^2 + y^2$, with $x, y \sim U(-2, 2)$. The training, test and validation sets consisted of 300 distinct randomly selected patterns.
- The function $f(x, y) = \sin(x^2) + \sin(y^2)$, with $x, y \sim U(-2, 2)$. The training, test and validation sets consisted of 300 distinct randomly selected patterns.
- The camel function $f(x, y) = 4 - 2.1x^2 + \frac{x^3}{3}x^2 + xy + (4y^2 - 4)y^2$, with $x \sim U(0, 10)$ and $y \sim U(0, 10)$. The training, test and validation sets consisted of 500 distinct randomly selected patterns.
- The function $f(x, y) = \sin(x)\sin(y)\sqrt{xy}$, with $x, y \sim U(0, 10)$. The training, test and validation sets consisted of 500 distinct randomly selected patterns.

This section reports results as averages over 30 simulations together with a 95% confidence interval as obtained from the t-distribution. For all the results, networks were pruned as soon as overfitting has been detected. That is, training stopped when

$$\mathcal{E}_V > \overline{\mathcal{E}}_V + \sigma_{\mathcal{E}_V}$$

where $\mathcal{E}_V$, $\overline{\mathcal{E}}_V$ and $\sigma_{\mathcal{E}_V}$ are respectively the current mean squared error (MSE) on the validation set, the average MSE since training started, and the standard deviation in the validation MSE.

Table I summarizes the results obtained for pruning of product hidden units for each of the functions listed above. The table provides the number of hidden units used for the initial oversized networks and the average number of hidden units after applying the variance analysis and brute force pruning. The MSEs for each problem are given for the training and test sets. The first

thing to notice from table I is the significant improvements in training accuracy and generalization for all the functions, except for the Camel function. In this case, the training error is a little worse for the pruned network, but generalization is on average better. For most of the problems, the variance analysis pruning algorithm did not manage to produce the same numbers of hidden units as obtained by the brute force approach. However, note that the training errors and generalization performances of the pruned networks for the variance analysis approach are much better than those obtained from the brute force pruning - an indication that the brute force approach overpruned the networks.

Table II compares the pruned PUNNs with pruned SUNNs, where the SUNNs were pruned using the variance analysis algorithm in [10]. Note that for all problems the pruned PUNN architectures are smaller than the pruned SUNN arechitectures, supporting the fact that PUNNs have a higher information capacity than SUNNs. It is only for the Henon map, Camel function and $f(x, y) = x^2 + y^2$ that the pruned SUNNs have better accuracy than the pruned PUNNs.

Table III illustrates the performance of the pruning algorithm under noisy conditions. For the functions listed in table III, 10% noise was added to the training set. For all the functions, the average number of pruned hidden units does not differ significantly from those obtained without noise (also refer to table I). For these functions, the performance of the variance analysis pruning algorithm does not degrade under these noisy conditions.

To illustrate the pruning of input units, irrelevant input parameters were introduced for functions $f(x) = x^2$, $f(x) = x^3 - 0.04x$ and $y^7x^3 - x^6$. These irrelevant parameters have random values. The number of hidden units as determined from the pruning algorithm has been used. Table IV summarizes the results for input pruning. In all three cases the network succeeded in removing the irrelevant input units. Both functions $f(x) = x^2$ and $f(x) = x^3 - 0.04x$ have only one relevant input unit. Three input units must therefore be pruned, which is the case. For the last function, the pruning algorithm succeeded to remove the two irrelevant parameters to remain with the two relevant inputs. Note the improvement in both the training accuracy and generalization performance for the pruned networks.

V. Conclusions

This paper presented a pruning algorithm for the pruning of input and hidden units of product unit neural networks, where product units are used within the hidden layer of the network. The pruning algorithm is based on the variance nullity pruning approach for summation units [10], where the significance of a unit is determined based on the sensitivity of the network output to perturbations of that unit.

Experimental results presented in this paper showed that the PUNN variance nullity pruning algorithm successfully removes irrelevant input and hidden units. Further extensions of the algorithm can be done to also prune network weights.

References

[1] F Girosi, M Jones, T Poggio, *Regularization Theory and Neural Networks Architectures*, Neural Computation, Vol 7, 1995, pp 219-269.

[2] PM Williams, *Bayesian Regularization and Pruning Using a Laplace Prior*, Neural Computation, Vol 7, 1995, pp 117-143.

[3] B Fritzke, *Incremental Learning of Local Linear Mappings*, International Conference on Artificial Neural Networks, Paris, October 1995.

[4] T-Y Kwok, D-Y Yeung, *Constructive Feedforward Neural Networks for Regression Problems: A Survey*, Technical Report HKUST-CS95-43, Department of Computer Science, The Hong Kong University of Science & Technology, 1995.

[5] R Reed, *Pruning Algorithms - A Survey*, IEEE Transactions on Neural Networks, 4(5), 1993, pp 740-747.

[6] R Durbin and D Rumelhart, *Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks*, Neural Computation, Vol 1, pp 133-142, 1989.

[7] DJ Janson and JF Frenzel, *Training Product Unit Neural Networks with Genetic Algorithms*, IEEE Expert Magazine, pp 26-33, October 1993.

[8] LR Leerink, C Lee Giles, BG Horne and MA Jabri, *Learning with Product Units*, Advances in Neural Information Processing Systems, Vol 7, pp 537, 1995.

[9] AP Engelbrecht, A Ismail, *Training Product Unit Neural Networks*, Stability and Control: Theory and Applications, Vol 2, No 1/2, pp 59-74, 1999.

[10] AP Engelbrecht, *A New Pruning Heuristic Based on Variance Analysis of Sensitivity Information*, IEEE Transactions on Neural Networks, Vol 12, No 6, November 2001.

[11] KI Funahashi, *On the Approximate Realization of Continuous Mappings by Neural Networks*, Neural Networks, 2, pp 183-192, 1989.

[12] K Hornik, M Stinchcombe and H Whit *Universal Approximation of an Unknown Mapping and its Derivatives Using Multilayer Feedforward Networks*, Neural Networks, 3, pp 551-560, 1989.

[13] A Ismail, AP Engelbrecht, *Improved Product Unit Neural Networks*, submitted to IJCNN, 2002.

[14] RC Eberhart, J Kennedy, *A New Optimizer using Particle Swarm Theory*, Proceedings of the Sixth International Symposium on Micro Machine and Human Science, pp 39-43, Nagoya, Japan, 1995.

TABLE I

PRUNING OF HIDDEN UNITS

	Oversized network			**Variance Analysis Pruning**			**Brute Force Pruning**		
Function	No of hidden units	MSE on Training set	MSE on Test set	No of hidden units	MSE on Training set	MSE on Test set	No of hidden units	MSE on Training set	MSE on Test set
$f(x) = x^2$	8	0.01450 $\pm$0.00340	0.04149 $\pm$0.03512	1.6	0.00054 $\pm$0.00028	0.00064 $\pm$0.00036	1	0.00034 $\pm$0.00011	0.00033 $\pm$0.00011
$f(x) = x^3 - 0.04x$	8	0.00449 $\pm$0.00099	0.0542 $\pm$0.00140	1.5	0.00015 $\pm$0.00018	0.00018 $\pm$0.00027	1	0.000018 $\pm$0.000004	0.000016 $\pm$0.000004
Henon	10	0.00391 $\pm$0.00415	0.0382 $\pm$0.04519	6.8	0.00014 $\pm$0.00007	0.00056 $\pm$0.00020	4	0.00317 $\pm$0.00175	0.00788 $\pm$0.00446
$f(x,y) = y^7x^3 - 0.5x^6$	10	0.00166 $\pm$0.00055	0.00232 $\pm$0.00062	3.8	0.00028 $\pm$0.00014	0.00132 $\pm$0.00071	2	0.00091 $\pm$0.00047	0.00121 $\pm$0.00062
$f(x,y) = x^2 + y^2$	10	0.03269 $\pm$0.00744	0.03501 $\pm$0.00653	3.3	0.01024 $\pm$0.00299	0.01122 $\pm$0.00281	2	0.00883 $\pm$0.002895	0.00831 $\pm$0.00271
$f(x,y) = sin(x^2) + sin(y^2)$	10	0.00396 $\pm$0.00127	0.01431 $\pm$0.01039	5.1	0.00047 $\pm$0.00022	0.00076 $\pm$0.00036	4	0.00211 $\pm$0.00095	0.00411 $\pm$0.00319
Camel	15	0.03058 $\pm$0.00208	0.04252 $\pm$0.00498	5.9	0.03273 $\pm$0.00457	0.03880 $\pm$0.00512	6	0.03169 $\pm$0.00267	0.03987 $\pm$0.00329
$f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$	12	0.00052 $\pm$0.00027	0.00506 $\pm$0.00325	7.1	0.00009 $\pm$0.00004	0.00068 $\pm$0.00077	7	0.00056 $\pm$0.00040	0.00079 $\pm$0.00056

TABLE II

COMPARISON OF OPTIMAL SUNN AND PUNN USING PSO

	SUNN			**PUNN**		
Function	Best configuration	MSE on Training set	MSE on Test set	Best configuration	MSE on Training set	MSE on Test set
$f(x) = x^2$	1 : 2 : 1	0.001945 $\pm$0.00172	0.001873 $\pm$0.001355	1 : 1 : 1	0.000344 $\pm$0.000112	0.000334 $\pm$0.000112
$f(x) = x^3 - 0.04x$	1 : 3 : 1	0.000095 $\pm$0.000025	0.000165 $\pm$0.000049	1 : 1 : 1	0.000018 $\pm$0.000004	0.000016 $\pm$0.000004
Henon	1 : 5 : 1	0.00042 $\pm$0.000049	0.00040 $\pm$0.000045	1 : 4 : 1	0.003173 $\pm$0.001754	0.007881 $\pm$0.004455
$f(x,y) = y^7x^3 - 0.5x^6$	1 : 6 : 1	0.001086 $\pm$0.000089	0.003604 $\pm$0.000953	1 : 2 : 1	0.000919 $\pm$0.000476	0.001213 $\pm$0.000625
$f(x,y) = x^2 + y^2$	1 : 4 : 1	0.001860 $\pm$0.000776	0.002164 $\pm$0.000753	1 : 2 : 1	0.008836 $\pm$0.002895	0.008312 $\pm$0.002709
$f(x,y) = sin(x^2) + sin(y^2)$	1 : 6 : 1	0.008542 $\pm$0.000572	0.011924 $\pm$0.000804	1 : 4 : 1	0.002119 $\pm$0.000956	0.004106 $\pm$0.003188
camel	1 : 8 : 1	0.001228 $\pm$0.000217	0.002044 $\pm$0.000344	1 : 6 : 1	0.031696 $\pm$0.002673	0.03987 $\pm$0.003291
$f(x,y) = sin(x) \cdot sin(y) \cdot \sqrt{x \cdot y}$	1 : 9 : 1	0.010991 $\pm$0.000368	0.012457 $\pm$0.000309	1 : 7 : 1	0.000568 $\pm$0.000400	0.000795 $\pm$0.000569

TABLE III

PRUNING OF HIDDEN UNITS WITH NOISE ADDED TO DATA

	Oversized network			**Pruned network**			
Function	No of hidden units	MSE on Training set	MSE on Test set	No of hidden units	MSE on Training set	MSE on Test set	MSE on Validation set
$f(x) = x^2$	8	0.01716 $\pm$0.00360	0.01921 $\pm$0.00493	1.7	0.00091 $\pm$0.00045	0.00118 $\pm$0.00070	0.00264 $\pm$0.00340
$f(x) = x^3 - 0.04x$	8	0.00677 $\pm$0.00110	0.01039 $\pm$0.00218	1.6	0.00073 $\pm$0.00026	0.00075 $\pm$0.00035	0.00087 0.00016
$f(x,y) = y^7x^3 - 0.5x^6$	10	0.00149 $\pm$0.00033	0.00311 $\pm$0.00082	3.9	0.00026 $\pm$0.00016	0.00044 $\pm$0.00017	0.00089 $\pm$0.00041

TABLE IV
PRUNING OF INPUT UNITS

	Oversized network			**Pruned Network**		
Function	No of input units	MSE on Training set	MSE on Test set	No of input units	MSE on Training set	MSE on Test set
$f(x) = x^2$	4	0.01970 ±0.00327	0.02223 ±0.00477	1	0.00186 ±0.00278	0.00171 ±0.00257
$f(x) = x^3 - 0.04x$	4	0.00199 ±0.00142	0.00256 ±0.00185	1	0.00003 ±0.00002	0.00004 ±0.00003
$f(x, y) = y^7 x^3 - 0.5x^6$	4	0.00680 ±0.00084	0.00950 ±0.00108	2	0.00163 ±0.00097	0.00269 ±0.00172