International Journal of Pattern Recognition and Artificial Intelligence

An Incremental Framework Based on Cross-Validation for Estimating the Architecture of a Multilayer Perceptron

Oya Aran, Olcay Taner Yıldız, Ethem Alpaydın

Department of Computer Engineering, Boğaziçi University,
TR-34342, Istanbul, Turkey
aranoya@boun.edu.tr, olcaytaner@isikuniv.edu.tr, alpaydin@boun.edu.tr

We define the problem of optimizing the architecture of a multilayer perceptron (MLP) as a state space search and propose the MOST (Multiple Operators using Statistical Tests) framework that incrementally modifies the structure and checks for improvement using cross-validation. We consider five variants that implement forward/backward search, using single/multiple operators, and searching depth-first/breadth-first. On 44 classification and 30 regression datasets, we exhaustively search for the optimal and evaluate the goodness based on: (1) Order, the accuracy with respect to the optimal, and (2) Rank, the computational complexity. We check for the effect of two resampling methods (5×2, 10-fold cv), four statistical tests (5×2 cv t, 10-fold cv t, Wilcoxon, sign) and two corrections for multiple comparisons (Bonferroni, Holm). We also compare with Dynamic Node Creation (DNC) and Cascade Correlation (CC). Our results show that: (1) On most datasets, networks with few hidden units are optimal, (2) forward searching finds simpler architectures, (3) variants using single node additions (deletions) generally stop early and get stuck in simple (complex) networks, (4) choosing the best of multiple operators finds networks closer to the optimal, (5) MOST variants generally find simpler networks having lower or comparable error rates than DNC and CC.

Keywords: Model selection; cross-validation; statistical testing; growing/pruning methods

1. Introduction

Every learning algorithm starts by assuming a certain model structure and once the structure is fixed, the parameters are trained to optimize the fit to the training data. Since the number of possible structures is large, the unsystematic way of trying a number of arbitrary structures and selecting the best rarely finds an optimal or a near optimal result and there is a need for a methodology to optimize the structure.

For a feed-forward multilayer perceptron (MLP), the model is defined by the network structure such as the number of hidden layers, the number of units in these layers and the connectivity between them, and once a network structure is fixed, backpropagation trains the weights of the connections in the network. In this paper, we use MLP with sigmoid hidden units and softmax/linear output units for classification/regression problems respectively. We assume the general and most-widely used connectivity pattern where each node feeds to all units in the next

layer, connections do not skip layers, and there are no recurrent connections.

The increasing number of data mining applications in the industry increased the number of end users, most of whom are not experts and lack the knowledge of the algorithms and the models used. Specifically for the case of MLP, it is hard to estimate the number of hidden units or layers even for an expert user. There is therefore need for a methodology to optimize the network structure in a manner invisible to the end user, which, though may not necessarily return the optimal, returns a good enough structure in reasonable time. In this paper, we propose the MOST framework to optimize the architecture without any user interference. A rudimentary version of this work was presented at ICANN 2003[1].

The optimal architecture is a network that is large enough to learn the underlying function and is small enough to generalize well. A network smaller than the optimal architecture can not learn the problem well; on the other hand, a large network will overlearn the training data resulting in poor generalization. The trade-off between bias and variance is the key[2]: A small network underfits and fails to learn the data (bias is high and variance is low). A large network overfits; it learns the data (bias is low) but also learns the noise (variance is high). The optimal architecture is the one with low bias and low variance so that the network learns the function underlying the data and not the added noise.

Approaches that modify network structure are basically a variant or a combination of two greedy strategies, growing and pruning[3]. If the initial model is a small network and the network grows during learning, the algorithm is called a *growing/constructive* algorithm[4]. If the initial network is large and the network shrinks during learning, then the algorithm is called a *pruning/destructive* algorithm[5]. There are also hybrid algorithms which can both add and remove. The constructive approach is generally preferred over the destructive approach: Specifying the initial network is easier in constructive methods; in destructive methods, one has to decide how big the initial network must be. Since constructive methods start small, they train simpler networks at each iteration and the total computation time is less. Constructive methods are also more likely to end up with smaller networks.

To check for improvement after a change in the network structure, one needs an evaluation function to assess the goodness of a network. In early studies, this used to be the training error. To check for overfitting, it is better to use the error on a separate validation set. To average over randomness on small datasets, it is nowadays customary to use cross-validation by resampling[6,7] and bootstrap[8,9]. A comparison of cross-validation and bootstrap techniques for model selection can be found in Ref. 10.

Most work on incremental algorithms consider networks with a single hidden layer, reducing the problem to the estimation of the number of hidden units[11,12,13,7]. In theory, networks with multiple hidden layers are not more powerful than ones with a single hidden layer[14], but it is also known empirically that networks with multiple hidden layers sometimes generalize better, with fewer parameters[15].

Among the constructive methods proposed, there are those that add hidden

nodes to a single hidden layer one by one; examples are dynamic node creation[12], upstart algorithm[13], and feedforward neural network construction using cross validation[7]. Cascade correlation[16] also adds hidden nodes but the added node becomes a new hidden layer with a single hidden node. Modified versions of cascade correlation are also proposed[17,18,19].

Resource allocating network is a constructive method for RBF nodes[20]. In Ref. 21, an algorithm for incremental neural network construction that is capable of learning new information without forgetting the old knowledge is proposed. This algorithm is based on the combination of RBF modules and fuzzy systems. An incremental training method for the probabilistic RBF is presented in Ref. 22. In Ref. 23, authors propose a growing network training strategy based on Hermite polynomial activation functions instead of sigmoid activation functions.

Methods have been proposed to improve convergence while adding hidden units one at a time by freezing previous weights or by training input or output layer weights separately[24,25,26]. In Ref. 27 authors propose the incremental extreme learning machine, which randomly generates hidden nodes and analytically determines the output weights. In the enhanced version of their algorithm[28] several hidden nodes are generated and only the hidden node leading to the highest improvement is added to the network at each step. There are also algorithms that do not fix the connectivity, such as group method of data handling[29], where the number of incoming connections to a hidden unit is fixed but their sources may change.

In pruning methods[30,31,32], the effect of each weight to the error can be used to determine whether it is necessary. After training a large network, weights are set to zero one by one and the change in the error is observed. A weight can be removed from the network if the change in error is small. However, evaluating the effect of each weight is cumbersome and will be very slow for a large network. Weight decay forces weights to decay to zero by adding a penalty term to the error function, after which connections with too small weights can be pruned. A survey of pruning methods can be found in Ref. 5.

Hybrid methods, such as generate and test procedure[33], allow both addition and deletion of hidden units and layers. These methods generally apply a pruning strategy following the constructive step as proposed in Refs. 34, 35, and 25. Hybrid algorithms which uses dynamic decay adjustment together with pruning strategies are proposed for RBF networks[36,37]. In Ref. 38, a growing and pruning strategy is applied to RBF networks. Learning accuracy is linked to the significance of each neuron, which is defined as the average information content of that neuron. In Ref. 39, a hybrid algorithm that makes use of splitting, pruning and merging is proposed to jointly determine the structure and the parameters.

Besides such greedy techniques, evolutionary programming[40] and genetic algorithms[41,42,43] are also used for learning the network structure. A good survey that covers both evolutionary algorithms and constructive/destructive methods can be found in Ref. 44. Random search methods such as genetic algorithms are suc-

cessful in finding networks that may have any arbitrary topology. For these very complex problems, the state space is not continuous and thus is not suitable for the use of an incremental technique. However, for networks with a fixed domain in which the solution space is continuous, it is possible to apply an incremental strategy and perform a guided search. In problems with a continuous solution space, constructive/destructive strategies that add/delete neurons or layers work well. Genetic algorithms, on the other hand, require longer execution times without necessarily outperforming incremental methods[44].

In Ref. 45, the number of hidden neurons are estimated for three-layered and four-layered MLPs using the geometrical interpretation of the weights, the biases, and the number of hidden neurons and layers. Regularization techniques and Bayesian methods are also used for model selection for neural networks[46,47,48]. Constructive or destructive strategies are applied to a wide range of applications, such as image compression[49], feature selection[50], robot control[51,52], forecasting[53], incident detection[54], and online character recognition[55].

In this paper, we propose a framework, named MOST, which intelligently searches the space of all possible MLP networks. Following a search strategy, starting from an initial state, we navigate through the state space with the help of operators that change the architecture by adding/removing units/layers. The comparison and selection of visited states are done via cross-validation and statistical tests. We present different MOST instantiations with different initial states and operators, leading to constructive or destructive MOST variants that implement depth or breadth-first search. We also discuss the effect of the statistical test used for architecture comparison on the actual and estimated optimal architectures.

In the following section, we present and discuss our proposed MOST framework. We present the experimental results in Section 3 where we compare in detail five different variants of MOST on 44 classification and 30 regression datasets. Section 4 gives the conclusions and discusses possible future directions.

2. MOST: A meta learning algorithm for architecture selection

2.1. *Structure learning as state space search*

We view optimizing the architecture of the MLP as a search in the state space of all possible architectures[1]. In our case where the connectivity graph and the activation function is fixed, the search space contains all possible combinations of hidden layers and hidden nodes. The search space is infinite and it is not possible to train/validate all architectures and select the best one. There is hence need for a heuristic strategy which finds a good solution visiting only a small part of the search space.

We define operators that modify the MLP structure and allow moving from one state to another. For example, there is the operator ADD-1 which adds one hidden unit and applying this operator takes us from state MLP_4 to MLP_5, that is from an MLP with four hidden units to one with five hidden units. Constructive/growing

algorithms therefore can be considered as implementing *forward search* which start from a simple initial state and use ADD operators. Destructive/pruning algorithms implement *backward search* by starting from a complex state and using REMOVE operators to prune unit(s)/layer. If both ADD and REMOVE operators are allowed, the algorithm implements a *floating search*. The MOST operators and their application details are given in Section 2.2.

After an operator application, the state evaluation function compares the goodness value of the next state with that of the current state and accepts/rejects the operator depending on whether the goodness value is improved or not. This state evaluation function trains and validates the network corresponding to the next state and uses a model selection criterion, which takes into account the generalization accuracy together with a measure of complexity, so that we favor architectures that are accurate and simple. The details of the model selection criterion is given in Section 2.3.

When we define the problem of finding the optimal architecture as a state space search, there are five choices to be made:

(1) *Initial state.* If we select the linear perceptron (LP) or another simple architecture as the initial state, the algorithm is mainly constructive and adds hidden nodes and/or layers to the architecture. If we start from a MLP with N hidden units (MLP_N), with large N, the algorithm is destructive and removes hidden nodes.
(2) *State transition operators.* Operators add unit/layer or remove unit/layer. To allow for faster convergence, there are also operators that make longer jumps in the state space by adding/removing multiple units, e.g., ADD-5 adds five hidden units whereas ADD-1 adds only a single hidden unit.
(3) *Search beam.* We can have a single operator which gives us a single candidate architecture or we can apply multiple operators to get multiple candidates. In the case of multiple operators, breadth-first, depth-first, best-first search, or any variant thereof can be used.
(4) *State evaluation function.* One can use cross-validation and an associated statistical test, or some other model selection criterion, such as Akaike's information criterion (AIC), minimum description length (MDL), Bayesian information criterion (BIC)[56], or structural risk minimization (SRM)[57].
(5) *Termination condition.* A trivial condition is to stop the search when no candidate improves on the current best. Another possibility is to stop when the error falls below a certain level, or when a fixed number of iterations are made.

2.2. *MOST operators*

MOST supports the following operators (ordered by their effect on the complexity):

(1) REMOVE-n: Remove n hidden units from a layer.
(2) REMOVE-1: Remove a single hidden unit from a layer.

(3) ADD-1: Add a hidden unit to a layer.
(4) ADD-n: Add n hidden units to a layer.
(5) ADD-L: Add a new layer before the output layer and set the number of units in this new layer as:

    H1: Number of nodes in the upper layer
    H2: Number of nodes in the upper layer $\times$ 2
    H3: Average of the number of nodes in the upper and lower layers
    H4: Average of the number of nodes in the upper and lower layers / 2

The operators above are selected to enable MOST to search widely in the search space. Applying REMOVE-n, ADD-n, or ADD-L results in large jumps in the search space whereas operators REMOVE-1 and ADD-1 are used for finetuning in a small neighborhood. The value of n in ADD-n or REMOVE-n can be determined as any percentage of the number of hidden units in that layer. To create large jumps in the search space, we set n as half of the hidden units in that layer. With ADD-L, when a new layer is to be added to the network, the challenging point is to determine the number of hidden units in each layer. A popular heuristic is using the average of the number of nodes in the upper and the lower layers (H3). However this can lead to misleading results if the number of units in one of the layers (e.g., input) is very large or very small. We use four heuristics: H1 and H3 are the base heuristics. H2 and H4 are useful when H1 and H3 return either too small or too large values. Again depending on their complexity, the candidates are tried ordered by their complexities and simpler ones are favored.

2.3. *The role of accuracy and complexity: The MultiTest algorithm*

We define the complexity of a network by the number of hidden layers and nodes. The number of layers is the first criterion and complexity increases as the number of layers increase. Given two networks having the same number of layers, the number of hidden nodes is considered and complexity increases as the number of hidden nodes increase. Other complexity measures, such as the number of connections, can also be used; note however that if the number of connections is used as the complexity measure, MLP with a few hidden nodes can be simpler than a LP depending on the number of inputs and outputs.

The pseudocode of MOST is given in Figure 1. It starts by selecting an initial network as current BEST *(Line 2)*. To generate candidate models, C_i, operators are applied *(Lines 4–5)* ordered by their complexities, and that is why candidate models are sorted *(Line 6)*. Once generated, a candidate model is first trained and validated over cross-validation folds *(Line 8)* and its expected validation error is compared with that of the current best using a one-sided statistical test.

The idea is to keep the network simple, unless we know that the additional complexity decreases the error significantly, and in applying the test, we take com-

```
1  BestModel MOST
2      BEST = initial network
3      while BEST changed
4          for each applicable operator OPER_i
5              C_i ← OPER_i(BEST)
6          Sort candidates C_i in the order of complexity
7          for i = 1 to number of candidates
8              Train and validate C_i on k folds
9              if C_i is more complex than BEST
10                 Test H_0 : μ_BEST ≤ μ_C_i
11                 if H_0 is rejected
12                     BEST ← C_i
13                     break
14             else
15                 Test H_0 : μ_C_i ≤ μ_BEST
16                 if H_0 is accepted
17                     BEST ← C_i
18                     break
19     return BEST;
```

Fig. 1. The pseudocode of MOST.

plexity into account as follows[58]: When comparing two models i and j where i is simpler than j, we test if model i has an expected error rate less than or equal to the expected error of model j:

$$H_0 : \mu_i \leq \mu_j \text{ vs. } H_1 : \mu_i > \mu_j$$

We set a prior preference of i over j because it is simpler. By assuming a prior ordering, we would like to test whether it is supported by the data, the hypothesis follows the prior and is one-sided. If the test accepts, we favor i: Either $\mu_i < \mu_j$, that is, the simpler model indeed has less error and we choose it because it is more accurate; or, $\mu_i = \mu_j$ and we prefer the simpler model. We favor model j only if the test rejects, i.e., when the additional complexity is justified by a significant decrease in error and the test (data) overrides our prior preference. That is, accuracy is checked first and given equal accuracies, the simpler model is favored.

Hence, in the case of MOST, if candidate C_i is more complex than the current best, it must have significantly less error in order to replace the best model *(Lines 9–13)*. Similarly, if C_i is simpler than the current best, it replaces the best unless it has significantly larger error *(Lines 14–18)*. If the best model is changed, the algorithm continues by generating new candidates *(Line 3)*. If none of the candidates replace the best, the algorithm stops and returns the current best *(Line 19)*.

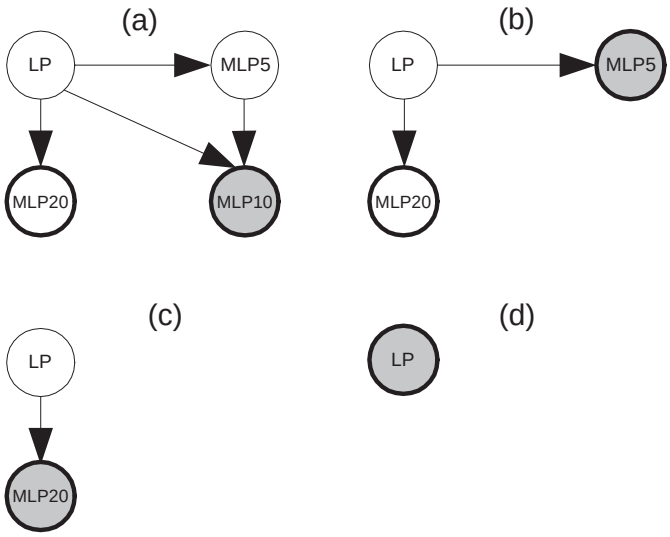


Fig. 2. Sample execution of MultiTest on four networks after the application of six pairwise tests. Nodes with thick lines indicate candidates at each step and among them, the simplest one (the most preferred) is taken (shown shaded). The best one is MLP_{10} and if we continue iterating the ordering found is $\mathbf{MLP_{10}} < MLP_5 < MLP_{20} < LP$.

If at any state we have multiple operators to apply and want to do breadth-first search, we generate *all* the next states and need to choose the best among all new candidates and the current best. Choosing the best of two models taking into account expected error and complexity by using a statistical test can be generalized to choosing the best of an arbitrary number of models using the *MultiTest algorithm*[58]:

To order $K > 2$ models, MultiTest uses a $K(K-1)/2$ pairwise tests and represents the results of these tests as a directed graph which it then sorts topologically. The graph has K vertices corresponding to the K models (Fig. 2). Assume that the models are sorted such that $i < j$ if model i is simpler than j. For all $i < j$ where $i, j = 1, \ldots, K$, we test $H_0 : \mu_i \leq \mu_j$, and if the test *rejects*, we place a directed edge from i to j to indicate that we are overriding the prior order. This corresponds to a binary relation R defined on the set of models where jRi implies that $j > i$ (more complex) and j has significantly less expected error than i. If we have $j \not R i$, this means that the test is accepted and our prior choice of i over j due to simplicity stands. The resulting directed graph has thus directed edges where the test is rejected for its incident vertices. The number of incoming edges to a node j is the number of models that are preferred over j but have higher expected error. The number of outgoing edges from a node i is the number of models that are less preferred (more complex) than i but have less expected error. The resulting graph need not be connected.

Once the directed graph is formed by the use of the pairwise tests, we choose the "best":

(i) We find the nodes with no outgoing edges to produce the set L_k. If there is no outgoing edge from a node, there is no other model that has less expected error.
(ii) From the elements of L_k, we select the simplest node to report. This selected model is the one that is the most preferred among all with the least expected error.

This calculates the "best" node; if we want to order in terms of "goodness," we iterate steps (i) and (ii) removing the best and the edges incident to it at each iteration to get a topological sort (Fig. 2).

To find the best of K models, MultiTest makes $K(K-1)/2$ tests, and in order to have a confidence level of $(1-\alpha)$ for the final best model, the confidence level of each one-sided test should be corrected. The two correction methods are Bonferroni[59] and Holm[60].

2.4. *Five MOST variants*

Different search algorithms can be obtained by changing the choices in the MOST framework. One can produce a constructive or a destructive algorithm by changing the initial state, operator set, and the order of trying the operators. Changing the search variant affects the performance of the solution network as well as the complexity of search until a solution is found. We investigate the following five MOST variants (Fig. 3):

(1) One-step forward (**1-Fwd**): Starts with LP and uses ADD-1 until there is no improvement.
(2) One-step backward (**1-Bwd**): Starts with MLP_{50} and uses REMOVE-1 until no improvement.
(3) Forward MOST (**Fwd**): Starts with LP and applies all operators above in increasing order of complexity in a depth-first manner, starting with the simplest one.
(4) Backward MOST (**Bwd**): Starts with MLP_{50} and applies all operators plus an operator for layer removal in decreasing order of complexity in a depth-first manner, starting with the most complex.
(5) Forward MOST with MultiTest (**MultiFwd**): Starts with LP and instead of applying one by one, in a breadth-first manner, applies all operators at once and chooses the best among these new candidates and the current best using MultiTest. (i.e. If there are m operators, MultiTest chooses the best of $m+1$, that is, m new candidates and the current best).

1-FWD and 1-BWD are the basic constructive and destructive algorithms. FWD and BWD allow multiple operators and use floating search, that is, allow both addi-

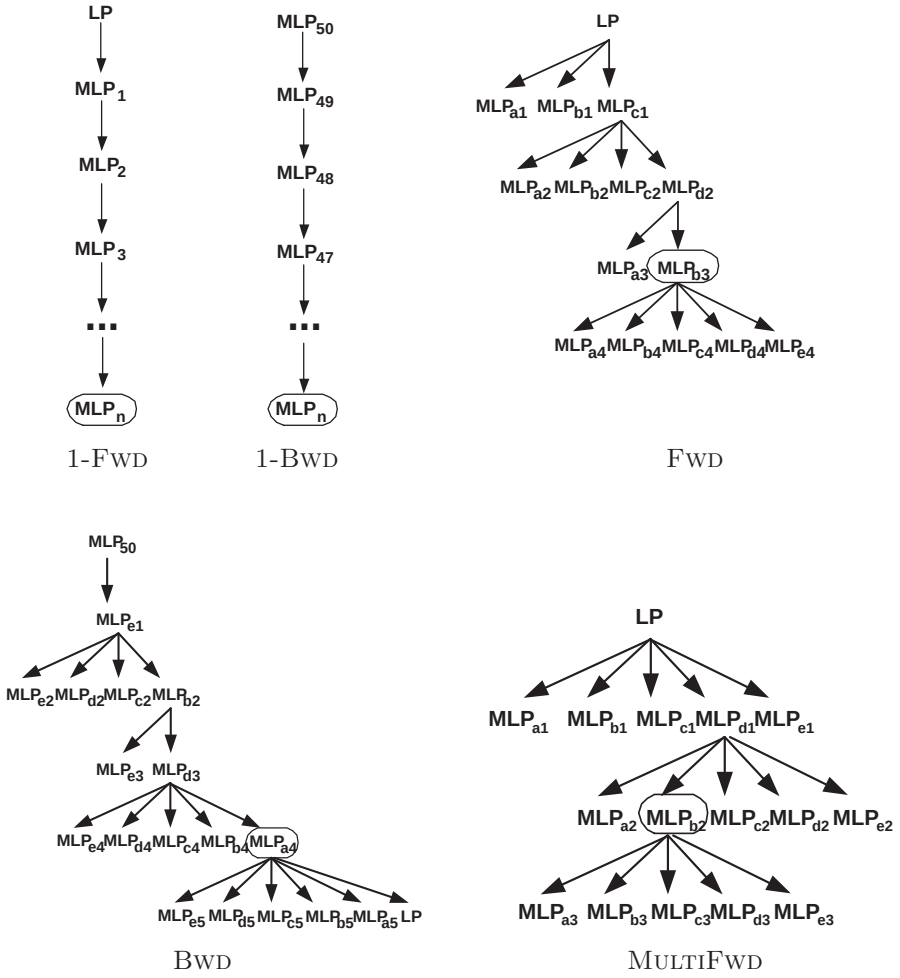


Fig. 3. Example search paths of the five MOST variants. For FWD, MULTIFWD and BWD, at step i, five candidate networks ($MLP_{ai}, MLP_{bi}, MLP_{ci}, MLP_{di}, MLP_{ei}$) are produced by applying operators. The complexity of these candidate networks increase from MLP_{ai} to MLP_{ei}. Note that in FWD, candidate networks are processed from simple to complex whereas the order is from complex to simple in BWD. The difference between MULTIFWD and FWD is that in MULTIFWD, instead of using an order and processing one by one, all candidate networks are processed at once and the best is selected. The circled states are the final optimal states selected by the algorithm.

tions and removals. MULTIFWD does breadth-first search evaluating all candidates at any intermediate state. FWD, MULTIFWD and 1-FWD can be defined as constructive and BWD, 1-BWD as destructive. It is important to note that MOST framework is general and by using different initial states and operator sets, one can

achieve a spectrum of MOST variants.

3. Experiments

3.1. *Experimental factors and evaluation criteria*

In the following experiments, we investigate the effect of the following factors:

- *MOST variant used in search.* These are 1-FWD, 1-BWD, FWD, BWD, and MULTIFWD, implementing forward vs. backward search, using single vs multiple operators, and searching depth-first vs breadth-first.
- *Type of application.* It can be classification or regression. We use 44 classification datasets and 30 regression datasets mainly from UCI Machine Learning repository[61], Delve[62], Statlib[63] and Statlog[64] datasets archives (Table 1). Discrete attributes with n possible values are converted to numeric using 1-of-n encoding.
- *Resampling done in cross-validation.* We use 5×2 and 10-fold cross-validation. Although cross validation is very time consuming, it generalizes better than information criteria such as AIC, BIC or SRM, because it makes no a priori assumptions, though is better to use such criteria when time complexity is crucial and lower execution times can be traded for a lower generalization error. The problem of neural network model selection is generally a one time process. Hence, higher accuracy should be preferred.
- *Statistical test used in comparison.* We use two parametric tests, 5×2 cv paired t test, 10-fold paired t test, and two nonparametric tests, Wilcoxon test and sign test[65].
- *Confidence level* $(1 - \alpha)$ *of the test.* We use 0.95, 0.99, 0.995, and 0.999.
- *Correction used when applying multiple tests.* We use Bonferroni and Holm corrections and compare with no correction.

Networks are trained using backpropagation based on stochastic gradient-descent with adaptive learning rate and momentum. The retraining of the network, when an operator is applied, is done by completely re-initializing all the weights and retrain from scratch. No weights are kept or frozen. The learning rate and the number of epochs are determined by cross-validation for each dataset separately, to maximize accuracy on the validation set. For each dataset, we determine learning rate-epoch pairs for LP and MLP networks and also for each resampling method, 10-fold and 5×2 cross validation, resulting in four different learning rate - epoch pairs per dataset. We determine the number of epochs by choosing among five different values (50 - 100 150 - 200 - 250) and the learning rate by choosing among (0.001 - 0.005 - 0.01 - 0.05 - 0.1). On each setup, the learning rate - epoch pair that gives the highest accuracy on the validation set is selected. Parameter selection for the MLP networks is performed on a network of 10 hidden units.

The architectures found by MOST variants are compared with the optimal architecture. The optimal architecture of each dataset is found by applying an exhaustive

Table 1. d: Inputs, C: Classes, N: Sample size. (U)CI, (D)elve, (S)tatlib, Stat(L)og, (A)T&T labs, s(Y)nthetic.

Classification	d	C	N	Source
artificial	10	2	320	U
australian	42	2	690	L
balance	20	3	625	U
breast	9	2	699	U
bupa	6	2	345	U
car	21	4	1728	U
cmc	24	3	1473	U
credit	46	2	690	U
cylinder	69	2	540	U
dermatology	34	6	366	U
ecoli	7	8	336	U
flags	118	8	194	U
flare (sonar)	23	3	323	U
glass	9	6	214	U
haberman	3	2	306	U
heart	13	2	270	U
hepatitis	19	2	155	U
horse	97	2	368	U
iris	4	3	150	U
ionosphere	34	2	351	U
letter	16	26	20000	U
monks	6	2	432	U
mushroom	116	2	8124	U
nursery	27	5	12960	U
ocr	256	10	600	U
optdigits	64	10	3823	U
pageblock	10	5	5473	U
pendigits	16	10	7494	U
pima	8	2	768	U
postoperative	23	3	90	U
ringnorm	20	2	7400	D
segment	19	7	2310	U
spambase	57	2	4601	U
tae	56	3	151	U
thyroid	47	4	2800	U
tictactoe	27	2	958	U
titanic	8	2	2201	D
twonorm	20	2	7400	D
vote	32	2	435	U
wave	21	3	5000	U
wine	13	3	178	U
yeast	8	10	1484	U
zipcodes	256	10	7291	A
zoo	16	7	101	U

Regression	d	N	Source
abalone	7	4177	U
add10	10	9792	D
bank32fh	32	8192	D
bank32fm	32	8192	D
bank32nh	32	8192	D
bank32nm	32	8192	D
bank8fh	8	8192	D
bank8fm	8	8192	D
bank8nh	8	8192	D
bank8nm	8	8192	D
boston	13	506	U
california	8	20640	S
comp	21	8192	D
kin32fh	32	8192	D
kin32fm	32	8192	D
kin32nh	32	8192	D
kin32nm	32	8192	D
kin8fh	8	8192	D
kin8fm	8	8192	D
kin8nh	8	8192	D
kin8nm	8	8192	D
puma32fh	32	8192	D
puma32fm	32	8192	D
puma32nh	32	8192	D
puma32nm	32	8192	D
puma8fh	8	8192	D
puma8fm	8	8192	D
puma8nh	8	8192	D
puma8nm	8	8192	D
sine	1	8000	Y

search over LP, all MLPs with a single hidden layer up to 50 hidden nodes, and MLP with two hidden layers up to 50 hidden nodes on both layers. Hence, there are $1 + 50 + 50 \times 50 = 2551$ different architectures to choose from. Then by using a statistical test, we compare each of these 2551 architectures with the other 2550 architectures and find the optimal architecture using MultiTest[58] (MultiTest also gives an ordering of these models and we use this ordering to calculate the "distance" between an architecture found by MOST and the optimal architecture, as we see below. Also see Section 2.3). We see that a two hidden layer MLP is selected as the best model in only a single dataset out of 74. Therefore throughout the rest of this paper, we restrict the search space to LP and MLP with a single hidden layer where the number of hidden units range from 1 to 50, hence having a search space of 51 states

We compare the architectures found by the five MOST variants in terms of three criteria:

(1) The accuracy of the estimated architecture,
(2) The complexity of the estimated architecture, and

Table 2. An example for calculation of ranks.

	Fwd	Bwd	MultiFwd	1-Fwd	1-Bwd
Order	3	2	1	1	2
Number of states visited	10	5	3	4	5
Rank	5	3.5	1	2	3.5

(3) Computational complexity of the search until an architecture is found.

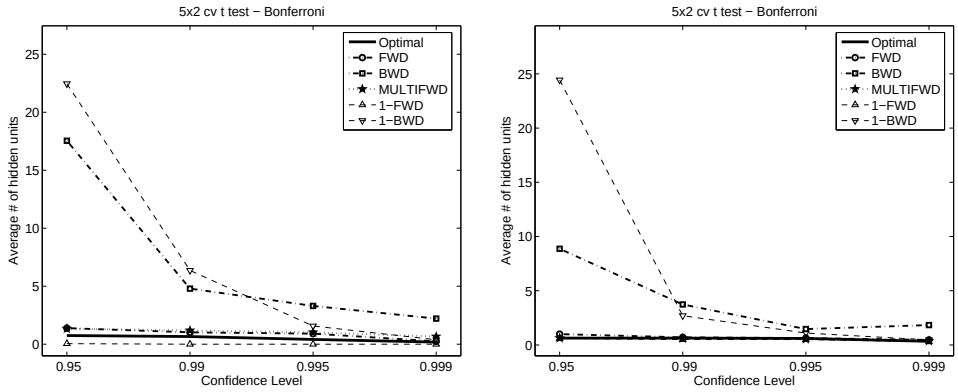
To measure the goodness of an architecture found, we define the measure of *order*, which, found by MultiTest (specific for a dataset), has the optimal architecture in position one, second best architecture in position two, and the worst architecture in the last position. Both accuracy and complexity are used for finding this order by MultiTest: A low order indicates that we either find an architecture that is close to the optimal architecture in terms of accuracy, or if the accuracies are comparable, we find an architecture that is simpler than an architecture with higher order.

Additional to the goodness of the final state found, we are also interested in how fast we get there, and for this, we define an additional measure of *rank*, which uses the number of states visited as a measure of the complexity of search. It is important to keep this low because this corresponds to the number of architectures that should be trained and validated (over multiple folds). The architectures found by the algorithms are ordered first using MultiTest to determine their orders and then ranked by their proximity to the optimal architecture. The search complexity is then taken into account if two or more MOST variants find the same architecture. In that case, ranks replace the order and the one which visits less number of states takes a lower rank. If the number of states visited is also equal, the average of the ranks is taken. An example for calculating the ranks is given in Table 2. A MOST variant with a low rank indicates that it finds an architecture that is close to the optimal architecture and in doing this, visits few states, when compared with another MOST variant with higher rank.

3.2. *Initial results*

Before reporting the full set of results, we first give results using 5×2 cv t test with $1-\alpha = 0.95, 0.99, 0.995, 0.999$ using Bonferroni correction, to get a quick, first idea. 5×2 cv t test has low type I error and reasonable power[65] and Bonferroni correction is the most frequent approach used in corrections. We discuss how results vary using other choices for resampling, test, and correction in the next subsection.

The effect of the confidence level on the average complexity of the architectures found by the five MOST variants, compared with the optimal architecture is shown in Figure 4. We see that on most datasets, LP or MLP with few hidden units tend to be considered optimal. Our experiments show that as the confidence level increases,



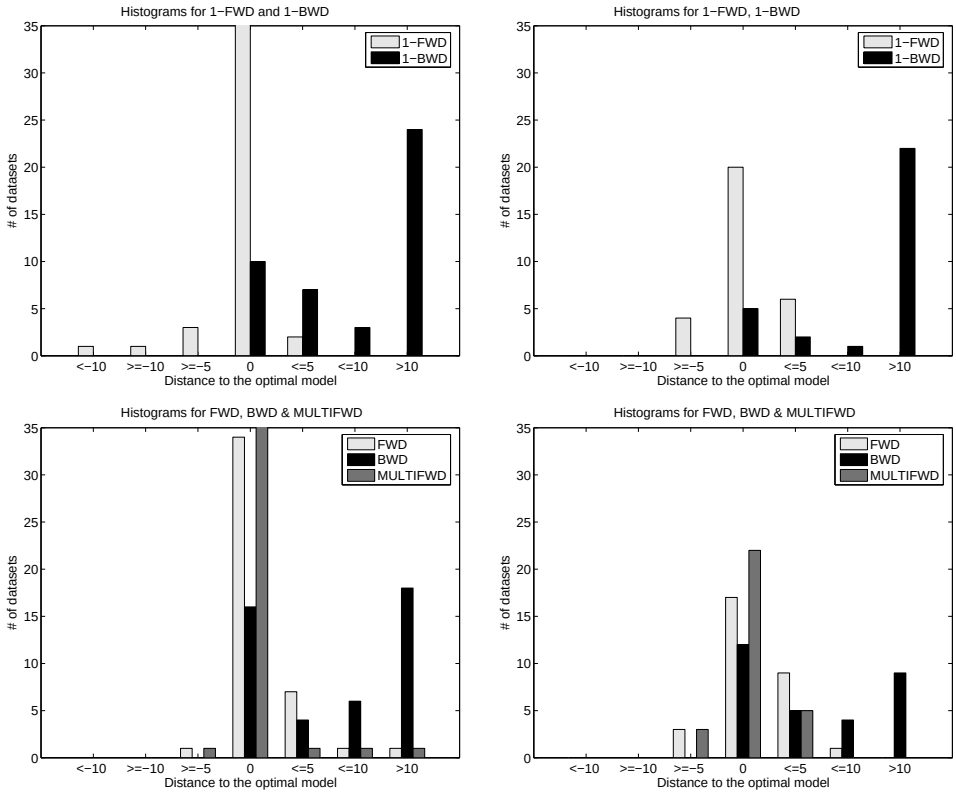
(a) Classification (b) Regression

Fig. 4. The average number of hidden units of the found architectures with 5×2 cv paired t test using Bonferroni correction.

the complexity of the optimal architecture as well as the complexity of the architectures found by MOST variants decrease. This is because when the confidence value is high, confidence intervals get larger and the differences between expected errors should be larger for two models to be considered significantly different; hence tests tend to accept equality of error rates more as $(1 - \alpha)$ increases, in which case the simpler one is preferred. As we see in the experiments, 5×2 cv t is a conservative test and Bonferroni correction has the effect of further increasing the confidence level.

The destructive variants, Bwd and 1-Bwd, are more sensitive to changes in the confidence level (note the steep decrease, e.g. from 0.95 to 0.99). Regardless of the confidence level, Fwd and MultiFwd get closest to the optimal architecture where MultiFwd is slightly better. 1-Fwd finds the simplest, and the two destructive variants, Bwd and 1-Bwd, find the most complex architectures. The behavior of MOST variants on classification and regression datasets are similar.

Figure 5 shows the histograms of the number of datasets according to the distance of the models found by MOST variants to the optimal architecture. The distance between two architectures is taken as the difference of the number of hidden nodes. Bin 0 of the histogram contains the number of datasets where the models are correctly found (distance to the optimal architecture is zero). Bins on the right contains the number of found models which are more complex and bins on the left contains models which are simpler than the optimal architecture. The models found by the constructive variants (1-Fwd, Fwd, MultiFwd) are generally simpler than the optimal architecture and for the destructive variants (1-Bwd, Bwd), models are more complex. The exact, optimal architecture is found more frequently by 1-Fwd than 1-Bwd. Fwd and MultiFwd find the optimal architecture more



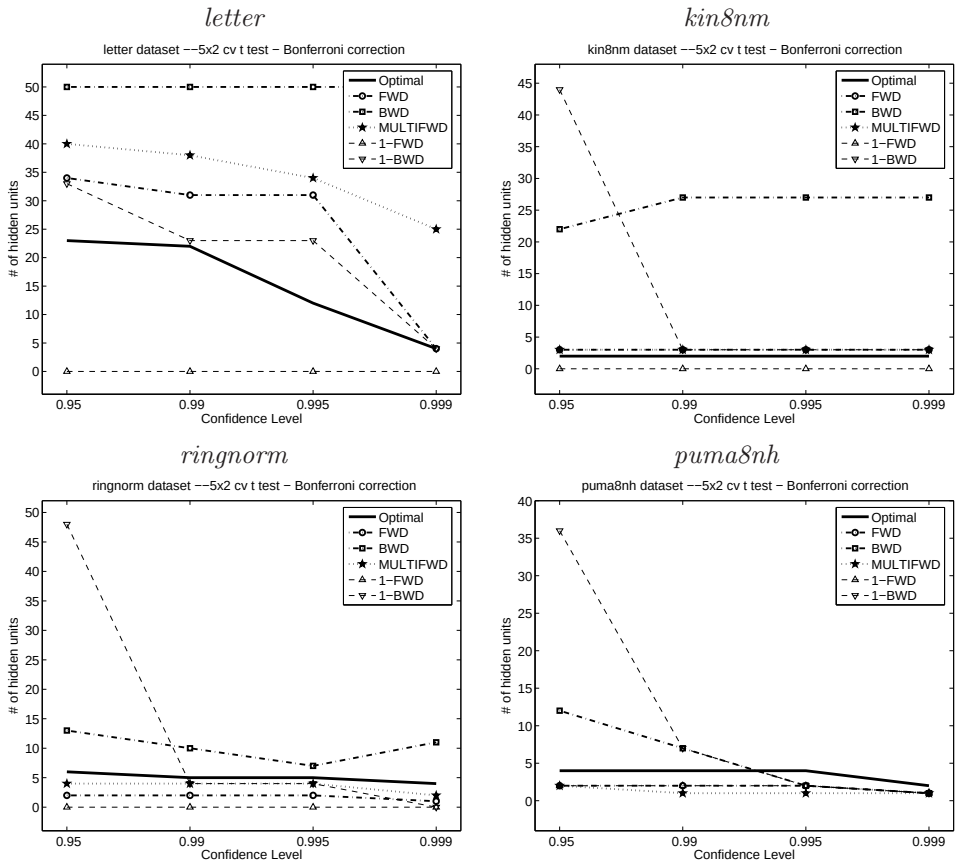
(a) Classification (b) Regression

Fig. 5. Histogram of number of hidden units found by 1-FWD, 1-BWD (top row) and FWD, BWD, MULTIFWD (bottom row). These results are obtained with 5×2 cv T test, Bonferroni correction and 0.95 confidence.

frequently than BWD.

The performances of the MOST variants, as a function of confidence level, on two example classification datasets, *letter* and *ringnorm*, are given in Figure 6a. The optimal architecture for the *letter* is the most complex one among the 44 classification datasets used in this work. 1-FWD find very simple architectures, which is an expected result due to its limited constructive nature. 1-BWD has an unexpected behavior and finds simpler architectures than FWD and MULTIFWD. For this dataset, 1-BWD finds the closest architectures to the optimal for each confidence level. FWD and MULTIFWD find simpler architectures as the confidence level increases. BWD gets stuck in the initial architecture and finds the most complex architecture for this dataset. The optimal architecture of *ringnorm* has a moderate complexity. For many of the datasets and for the average values, as confidence level increases, the complexity decreases, however in *ringnorm*, the confidence level on

BWD has an unusual effect. The complexity of the model found by BWD is higher for 0.999 than lower confidence levels. MULTIFWD finds the closest architecture to the optimal architecture.

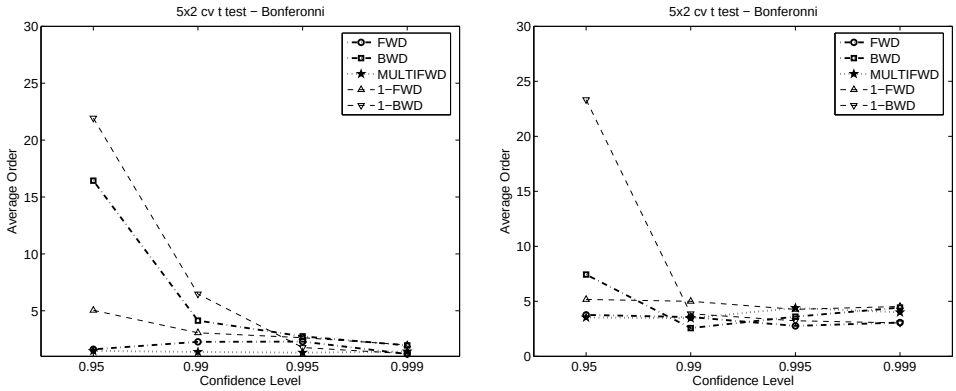


(a) Classification (b) Regression

Fig. 6. Results on two example (a) classification and (b) regression datasets: The architectures found by the MOST variants are compared with the optimal architecture

The performances of the MOST variants on three example regression datasets, *kin8nm* and *puma8nh*, are given in Figure 6b. For *kin8nh* dataset, although the architectures found by MULTIFWD and FWD are very close to the optimal, none of the variants is successful at finding the exact architecture. The architecture found by BWD is more complex than the optimal and is not affected by the confidence level. For *puma8nh* dataset, exact architecture is not found by any variant. Constructive variants, 1-FWD, FWD and MULTIFWD, find close results for all confidence levels. Destructive variants, 1-BWD and BWD, find closer results as the confidence level

increases.



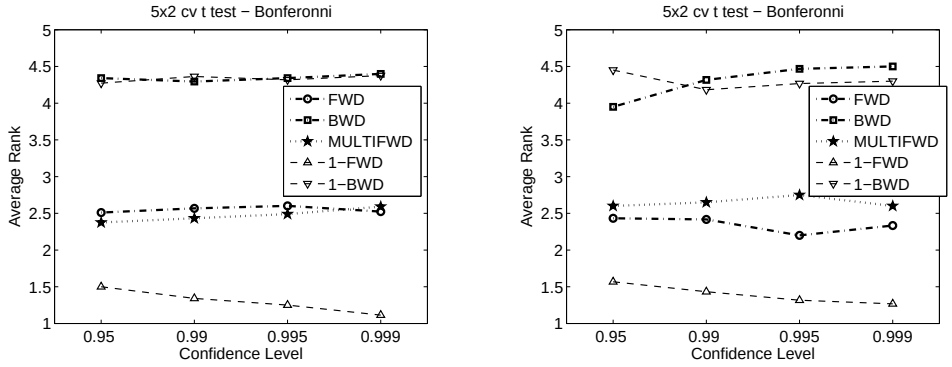
(a) Classification (b) Regression

Fig. 7. The average order of the MOST variants with 5×2 cv paired t test using Bonferroni correction.

Figure 7 shows the average *order* of the five MOST variants on all datasets, where, *order*, as we discuss above corresponds to the distance to the optimal architecture. It can be seen that for classification problems, FWD and MULTIFWD have smaller order whereas BWD and 1-BWD have higher order and 1-FWD is in the middle. For regression problems, the same behavior can be seen at 0.95 confidence. For both classification and regression problems, the differences between MOST variants tend to disappear for high confidence levels, when confidence intervals gets so large that the differences between error rates are considered insignificant.

Figure 8 shows the average *rank* of MOST variants for different confidence levels, where *rank*, as we discuss above, corresponds to the time complexity of the search. The two destructive variants, BWD and 1-BWD, have high rank, whereas the constructive variants, FWD, MULTIFWD and 1-FWD, have low ranks, with 1-FWD having the lowest rank.

Both the order and the rank of destructive variants, BWD and 1-BWD, are the highest. This shows that the architectures found by these are far from optimal and that their computational complexities are high. However for constructive variants, the orders of FWD and MULTIFWD are lower than 1-FWD whereas their ranks are higher. This implies that FWD and MULTIFWD can find architectures that are closer to the optimal but need to visit more intermediate states for this. Unlike FWD and MULTIFWD, variants that perform one hidden node additions (1-FWD and 1-BWD) perform well only if the optimal architecture is close to the initial architecture (too small or too large respectively).



(a) Classification (b) Regression

Fig. 8. The average rank of the MOST variants with 5×2 cv paired t test using Bonferroni correction.

3.3. *The effect of the confidence level, the statistical test, and the correction methodology*

The optimal architecture found and the performances of the MOST variants change when a different correction methodology or a different statistical test is used. We use four tests; two parametric tests, 5×2 cv t and 10-fold cv t tests, and two non-parametric tests, Wilcoxon and sign tests. For these tests, we check for the effect of varying the confidence at four levels, 0.95, 0.99, 0.995, 0.999. For comparison purposes, we also show the results when no statistical test is used; in this case, when two architectures are compared, the one with the higher average validation fold accuracy (doing 10-fold cv) is selected, without any regard for statistical significance. To observe the effect of the correction methodology, we also compare the results with no correction against applying Bonferroni or Holm correction. In this section, we will only report results on classification datasets as very similar behaviors are observed on the regression datasets.

Figure 9 shows the change of the average number of hidden units of the optimal architectures according to the confidence level $1-\alpha$ and the correction method. The average complexity drops down as the confidence level increases. This is expected because as the confidence level increases, the differences between the errors of some of the complex models and some of the simple models become insignificant. In that case, since complex models are not significantly better than the simple models, the simple models will be selected as optimal.

Statistical tests have a similar drop down effect on the complexity according to how conservative they are. We see that 5×2 cv t and Wilcoxon tests are more conservative (define larger confidence intervals) than 10-fold cv t and sign tests, since the complexity (the number of hidden units) of the optimal architecture is always higher in the two latter. When we use no statistical test and select the

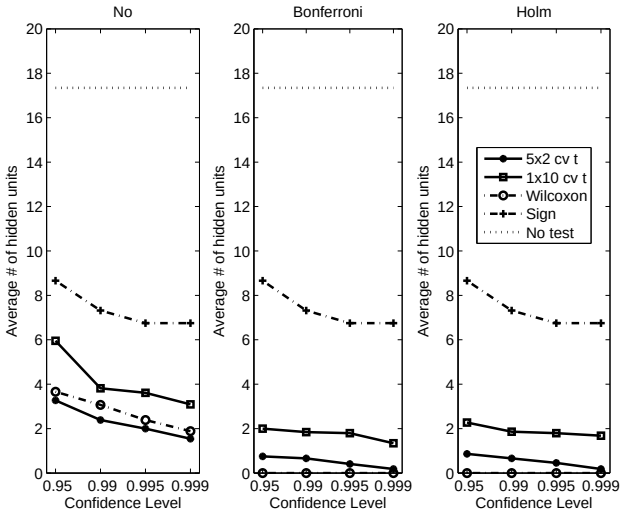


Fig. 9. Effect of confidence level, statistical test, and correction method on the average number of hidden units of the optimal network architecture on classification datasets.

architecture with highest accuracy, the average complexity increases, justifying the need for model selection, e.g., through a statistical test.

The sign test accepts or rejects the null hypothesis by counting how many times the first model is more accurate than the second model in k folds, where the minimum is 0 and the probability of this case is $1/2^k$. Because of this, the sign test can not accept or reject null hypotheses for confidence levels higher than $1 - 1/2^k$. In our case where $k = 10$, we have $1 - 1/2^{10} = 0.999$. For both Bonferroni and Holm corrections, the minimum confidence level is $1 - 0.05/(50 \cdot 51/2) = 0.99996$, which is higher than the maximum confidence level we can reject. Therefore for both correction types, we can never reject the null hypothesis and the simplest model, LP, is always selected. To prevent this effect, in sign test, when the second model wins against the first in all folds, we reject the hypothesis. This procedure cancels the effect of confidence when there is an absolute win in all folds. In Figure 9, we see that the results of the sign test are more or less the same when different correction methodologies are applied.

Similar results occur also with the Wilcoxon test, which orders the validation errors of both models and assigns rank one to the minimum result, rank two to the second minimum result, etc. After assigning the ranks, Wilcoxon test defines a test statistic based on the sum of the ranks of the first model. The best/worst case occurs when all of the error rates of the first model are smaller/larger than the error rates of the second model. For our problem ($k = 10$), this case has a probability of 0.0001 (confidence level of 0.9999). As explained before, the minimum confidence level for Bonferroni and Holm corrections is higher than this confidence level. Therefore for

both correction types, we can never reject the null hypothesis and the simplest model, LP, is always selected.

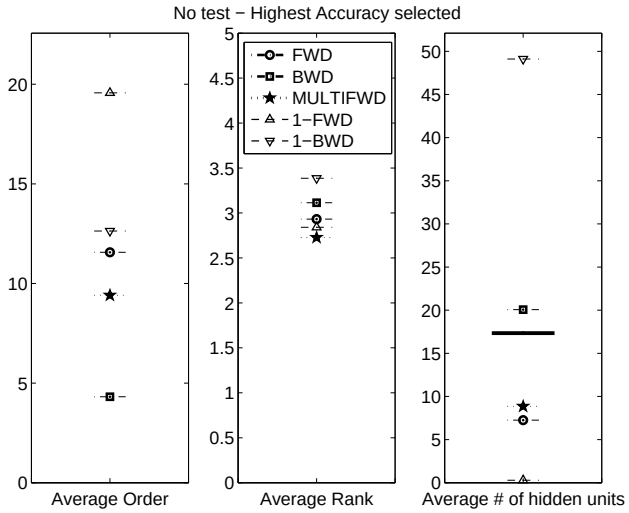


Fig. 10. Average order, rank, and the number of hidden units of architectures found by MOST variants on classification datasets when no statistical test is used and simply the model with the highest average validation accuracy is selected. The bold line shows the average optimal number of hidden units for this case.

Figure 10 shows the average order, rank, and the number of hidden units of the models found by MOST variants without applying any statistical test but by just selecting the model with the highest average validation set accuracy. On the classification datasets, BWD finds the closest architecture to the optimal in terms of accuracy and model complexity (order) and 1-FWD is the worst. When computational complexity is also taken into account (rank), MULTIFWD is the best whereas BWD has a high rank, meaning that although it finds the closest architecture to the optimal on the average, its computational complexity is very high. The average number of hidden units found by MOST variants are also compared to that of the optimal model. It can be seen that the results of 1-FWD and 1-BWD are lower and upper bounds respectively, as expected. The average model found by BWD is the closest to the average optimal model. As shown in Figure 9, the optimal models in this case, compared to when a statistical test is used, are very complex. Therefore, the performance of BWD is expected to be good since when the optimal model is complex, backward search algorithms perform better.

The reason of obtaining architectures with high complexity when no statistical tests are used is that, whenever candidate architecture produces error rates less than the current one, it is automatically accepted. However, by using statistical

tests to compare the error rates of the architectures, even when the average error of some architecture is less than the other, the statistical test may indicate that there is no significant difference between them. When there is no significant difference, MOST gives the priority to the simpler architecture. This makes it hard to accept candidate architectures with large number of hidden units when we apply statistical tests.

Figure 11 shows the average number of hidden units found by MOST variants as the statistical test, the confidence level, and the correction method are varied on classification datasets. In all cases except when Wilcoxon test is used with correction, FWD and MULTIFWD find architectures closest to the optimal. The confidence level of Wilcoxon test with correction is very high and it selects LP as the optimal architecture for all datasets. BWD and 1-BWD find simpler architectures as the confidence level increases.

Figure 12 shows the effect of statistical test and correction methodology on the order of MOST variants on classification datasets as a function of the confidence level. When a correction method is applied (either Bonferroni or Holm), the average order of the constructive algorithms, FWD, MULTIFWD, 1-FWD, decrease and get closer to the optimal, whereas for destructive algorithms, BWD and 1-BWD, the behavior is variable. For the sign test, the correction method has no effect as a result of the slight modification in the test procedure in which we reject the hypothesis when the second model wins against the first in all folds. In general, FWD and MULTIFWD have the smallest order and they find architectures that are close to the optimal in terms of accuracy and also less complex than architectures with comparable accuracy.

When algorithm ranks are considered (see Figure 13), we see that the effect of the type of statistical test or correction on the rank is very small. In all cases, destructive algorithms, BWD and 1-BWD, have the highest rank. Although they have small order especially when the confidence level is high, they have the disadvantage of starting from a complex architecture. This shows that when the computational complexity of the search is also taken into account, destructive algorithms have poor performance. Among the constructive algorithms, 1-FWD has the lowest rank and FWD and MULTIFWD have slightly higher ranks. Again, the lowest rank of 1-FWD follows from the fact that most datasets have LP or MLP with few hidden units as the optimal architecture and 1-FWD start from very close to the goal. With less conservative tests and smaller confidence levels, the optimal model starts to move further away from LP and the ranks of constructive variants start to increase.

3.4. *Comparison with other incremental methods*

We compare MOST with two well-known incremental methods, namely Dynamic Node Creation and Cascade Correlation.

Dynamic Node Creation (DNC)[12] builds MLP networks with a single hidden layer. It starts with small number of hidden units and incrementally adds hidden

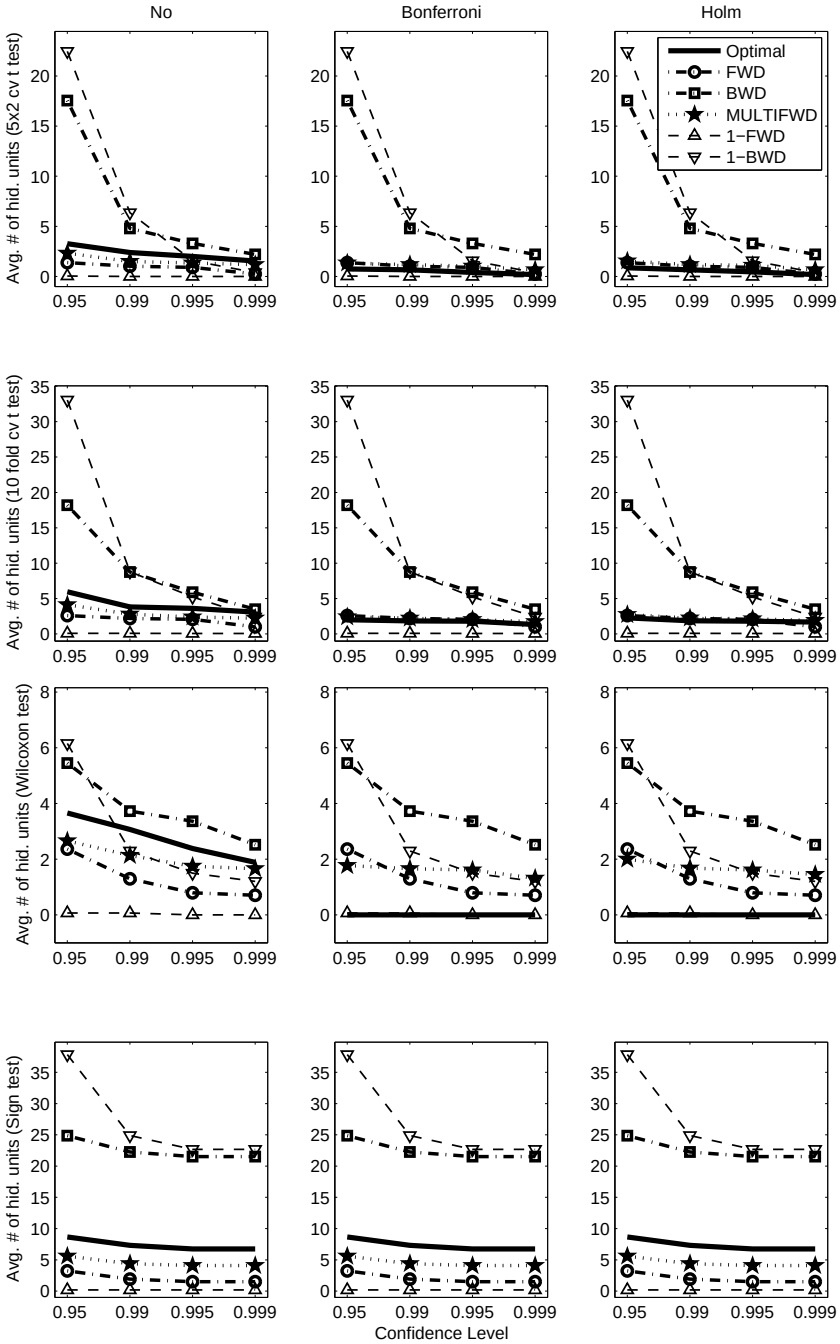


Fig. 11. Effect of confidence level, test, and correction on the number of hidden units for classification datasets.

nodes to the network until a satisfactory solution is found. Hidden nodes are added one at a time and to the same hidden layer. The weights of the newly added hidden node are initialized randomly to a small number. The whole network (both the old hidden nodes and the new one) is re-trained after each addition. The decision of adding a new hidden node is made by considering the flatness of the average error curve. Although both DNC and 1-Fwd variant of MOST construct MLP networks in a very similar way, they differ in the details. The main difference is the decision of adding a new node: 1-Fwd uses a statistical test whereas DNC checks the flattening of the error curve.

Cascade Correlation (CC)[16] constructs a network with multiple hidden layers, each layer containing a single hidden unit. This method uses the constructive approach and starts with an initial network and incrementally adds hidden nodes to the network until a satisfying solution is found. In cascade correlation, the new nodes are added as a one-unit hidden layer. The network formed is different in that the inputs are directly connected both to the outputs and to the hidden units. A hidden unit is connected to the inputs, to the outputs and to all the hidden units in the preceding layers. When a hidden unit is added, first its input weights are calculated until its contribution to the network error is minimum and then is added to the network. The calculated input side weights are frozen in training the output weights.

We compare the average error rate and the complexity of the architecture found by DNC and CC with two MOST variants, 1-Fwd and MultiFwd. Table 3 shows the complexities (in terms of the number of hidden units) and the error rates of the architectures found by DNC, CC, 1-Fwd, and MultiFwd on classification datasets. The error rates are averaged over 10 fold validation sets. The models found by DNC and CC differ from one fold to the other, thus for these methods we report the average number of hidden units over all folds.

The last column in Table 3, denotes the number of architectures visited/trained during the search of MultiFwd. For DNC, CC and 1-Fwd, the number of architectures visited during the search is the same as the complexity of the architecture found since these methods start with an MLP with a single hidden unit and adds units/layers one by one. The time complexity of a method is directly related to the number of architectures visited and the model complexity of each visited architecture.

We present a summary of these results via pairwise comparisons of these four methods. In Table 4a, we give a pairwise comparison in terms of accuracies on classification datasets. We see that on the classification datasets DNC and CC lose to MOST variants more than they win against them, with MultiFwd being more accurate than 1-Fwd.

In Table 4b, we give a pairwise comparison in terms of the complexities (number of hidden units). We see that both DNC and CC find more complex networks than 1-Fwd and MultiFwd. 1-Fwd and MultiFwd find the same architectures in general, with MultiFwd finding more complex ones. We see that the more complex

Table 3. Error rates of DNC, CC, 1-Fwd and MultiFwd on classification datasets. C denotes the architecture complexity in terms of the number of hidden nodes. S denotes the number of architectures visited during the search of MultiFwd.

	DNC		CC		1-Fwd		MultiFwd		
Dataset	**C**	**Err±std**	**C**	**Err±std**	**C**	**Err±std**	**C**	**Err±std**	**S**
artificial	2.0	0.00±0.00	0.0	0.00±0.00	0	0.00±0.00	0	0.00±0.00	5
australian	7.4	14.49±2.04	12.4	21.59±5.28	0	13.20±3.94	0	13.20±3.94	5
balance	1.0	0.00±0.00	2.4	1.168±1.24	0	1.76±2.54	0	1.76±2.54	4
breast	1.9	3.85±2.58	6.9	5.50±1.60	0	2.85±1.63	0	2.85±1.63	5
bupa	2.2	36.23±8.52	12.8	30.83±4.31	0	29.54±5.86	0	29.54±5.86	3
car	4.4	1.91±0.99	8.0	0.98±0.41	0	6.42±1.55	5	2.84±2.27	10
cmc	3.0	46.78±5.90	4.5	39.51±2.54	0	48.54±4.73	3	45.01±5.03	8
credit	7.3	15.79±2.79	16.9	25.41±9.95	0	13.46±3.14	0	13.46±3.14	5
cylinder	3.0	24.29±7.66	12.4	38.24±3.29	0	25.22±6.64	0	25.22±6.64	5
dermatology	3.1	5.20±3.574	0.0	1.09±0.47	0	1.68±2.69	0	1.68±2.69	5
ecoli	24.4	18.28±5.32	4.3	22.63±40.79	0	12.29±5.67	0	12.29±5.67	4
flags	6.4	43.40±12.10	0.0	9.77±2.35	0	32.54±10.15	0	32.54±10.15	5
flare	1.6	11.39±2.25	13.4	17.14±29.15	0	10.78±1.92	0	10.78±1.92	4
glass	1.9	46.87±8.24	24.2	13.85±2.36	0	35.93±12.23	0	35.93±12.23	5
haberman	8.9	25.48±4.26	7.4	24.33±3.45	0	24.20±5.69	0	24.20±5.69	5
heart	2.8	18.15±9.31	10.6	19.44±7.16	0	14.07±9.85	0	14.07±9.85	4
hepatitis	1.8	18.23±9.48	8.9	22.57±28.16	0	15.41±9.14	0	15.41±9.14	5
horse	2.0	11.46±4.57	10.9	18.10±6.71	0	11.15±4.60	0	11.15±4.60	5
iris	1.9	4.67±6.33	4.1	3.78±4.45	0	3.33±4.71	0	3.33±4.71	5
ironosphere	4.3	7.70±3.33	3.5	11.02±5.79	0	11.06±4.90	0	11.06±4.90	5
letter	19.1	15.40±1.17	30.0	1.56±0.07	0	22.60±0.96	46	8.05±1.17	13
monks	3.8	0.00±0.00	2.0	0.11±0.36	3	3.92±4.16	3	3.92±4.16	5
mushroom	1.2	0.10±0.14	0.0	0.03±0.04	0	0.00±0.00	0	0.00±0.00	5
nursery	5.3	0.60±0.29	11.3	0.05±0.03	0	7.18±0.49	5	1.10±0.85	12
ocr	5.6	7.50±2.97	0.0	2.17±0.61	0	3.00±3.12	0	3.00±3.12	4
optdigits	8.0	4.79±0.77	22.0	1.32±0.17	0	3.11±0.95	0	3.11±0.95	5
pageblock	9.1	2.98±0.62	13.9	21.64±41.30	0	3.58±0.51	4	3.45±0.33	8
pendigits	14.1	1.74±0.32	28.6	2.65±2.79	0	3.35±0.64	13	1.64±0.50	8
pima	1.4	23.56±5.27	13.8	34.82±8.36	0	22.12±5.46	0	22.12±5.46	4
postoperative	5.5	40.09±15.93	4.5	68.75±40.65	1	28.27±4.73	1	28.27±4.73	7
ringnorm	17.0	7.68±0.89	30.0	6.71±0.62	0	23.38±1.77	10	7.99±1.21	10
segment	4.0	14.11±3.30	20.5	7.37±3.89	0	4.68±1.08	0	4.68±1.08	5
spambase	5.5	6.61±1.24	14.9	11.73±4.54	0	7.13±1.16	1	6.89±1.53	9
tae	13.6	34.35±11.85	15.1	42.16±40.58	0	40.50±8.17	0	40.50±8.17	5
thyroid	2.8	1.57±0.56	12.5	0.97±0.26	0	1.68±0.74	0	1.68±0.74	5
tictactoe	2.1	2.72±1.34	4.2	3.40±1.91	0	1.67±0.74	0	1.67±0.74	5
titanic	9.3	21.31±2.07	3.5	25.72±2.12	0	21.17±2.18	0	21.17±2.18	4
twonorm	4.0	2.19±0.38	0.3	80.47±41.17	0	2.15±0.40	0	2.15±0.40	5
vote	3.9	4.57±3.53	21.5	4.00±3.22	0	3.20±2.87	0	3.20±2.87	5
wave	17.3	14.76±1.23	7.0	12.27±0.99	0	12.90±1.27	0	12.90±1.27	4
wine	2.0	3.44±3.96	10.9	14.29±6.00	0	1.14±2.41	0	1.14±2.41	5
yeast	13.4	41.85±5.36	6.7	10.57±0.76	0	40.78±3.39	0	40.78±3.39	5
zipcodes	6.3	8.86±1.09	7.9	21.41±41.42	0	5.07±0.98	15	4.83±0.93	10
zoo	4.1	8.17±8.12	0.0	1.96±2.13	0	4.97±7.45	0	4.97±7.45	5

architectures that MultiFwd finds also have less error rates, as in classification datasets such as *ringnorm, pendigits, spambase, nursery, letter*.

The results indicate that MOST variants find architectures that are both simple and accurate with 1-Fwd running fast and MultiFwd running slow but leading to more accurate networks.

4. Conclusions

We propose the MOST framework which considers the optimization of the number of hidden units of a MLP as a search problem in the space of all possible networks. Starting from an initial state, operators allow addition/removal of units/layer and we use cross-validation and a statistical test to compare the goodness of states to accept/reject operators. Depending on the initial state and operators, different MOST instantiations are possible leading to constructive or destructive variants,

Table 4. Pairwise comparison of (a) algorithm accuracies in terms of the number of *wins-losses-ties*, and (b)complexities of networks found by algorithms in terms of the number of hidden units, *less-higher-equal*, over 44 classification datasets

(a) Accuracy

	DNC	CC	1-FWD	MULTIFWD
DNC		23-20-1	14-29-1	9-34-1
CC			18-25-1	17-26-1
1-FWD				2-7-35
MULTIFWD				

(b) Complexity

	DNC	CC	1-FWD	MULTIFWD
DNC		29-15-0	0-44-0	2-41-1
CC			1-38-5	3-36-5
1-FWD				9-0-35
MULTIFWD				

implementing depth- vs breadth-first search.

Of the five variants we compare, MULTIFWD and FWD are the best performing algorithms where MULTIFWD is slightly better. Although they both use the same operators, the difference comes from the fact that MULTIFWD compares all candidate models with the current best model simultaneously whereas FWD compares them one by one. Hence MULTIFWD selects architectures that are as accurate as or more accurate than ones FWD selects. The MOST variants which add (1-FWD) or remove (1-BWD) one hidden node at a time usually stop early. One exception is when the optimal architecture is LP; in that case, since 1-FWD starts with LP and there are no other architectures better than LP, 1-FWD finds the optimal architecture immediately. The performances of the destructive variants (1-BWD and BWD) are highly affected by the confidence level; as the confidence level increases, the architectures found by 1-BWD and BWD tend to get closer to the optimal. As expected, the constructive versions (FWD, MULTIFWD, 1-FWD) find simpler architectures and destructive versions (BWD, 1-BWD) find more complex ones.

Regardless of the search strategy, as the confidence level of the statistical test increases, the complexities of the optimal architecture and the architectures that MOST variants find decrease. A similar effect is observed when Bonferroni and Holm corrections are used, which has the same effect of increasing the overall confidence level. Although the number of models used is large (51), the results for Bonferroni and Holm correction are quite similar, if not the same, leading us to conclude that for our case, Bonferroni is not more conservative than Holm. Among the four statistical tests used in our experiments, 5×2 cv t and Wilcoxon tests are more conservative than 10-fold cv t and sign tests, and because of this, tend to find simpler architectures.

The computational complexities of destructive variants are higher than those of constructive variants. This is expected because the computational complexity of a search problem depends on the distance between the initial state and the final state and the optimal architectures of the datasets in this study tend to be simple.

Our proposed MOST variants 1-FWD and MULTIFWD, both on classification and regression datasets generate networks that use simpler and on the average more accurate than those generated by well-known incremental algorithms, dynamic node

creation and cascade correlation.

Here we use cross-validation and a statistical test but other model selection methods can also be used and one can propose MOST variants using AIC, BIC, MDL, or SRM. Though our discussion in this paper focuses on optimizing the complexity of feed-forward MLP, a similar approach can be applied to structure learning in a wide range of learning architectures, e.g., recurrent networks, Bayesian networks, hidden Markov models, etc. These are possible future research directions.

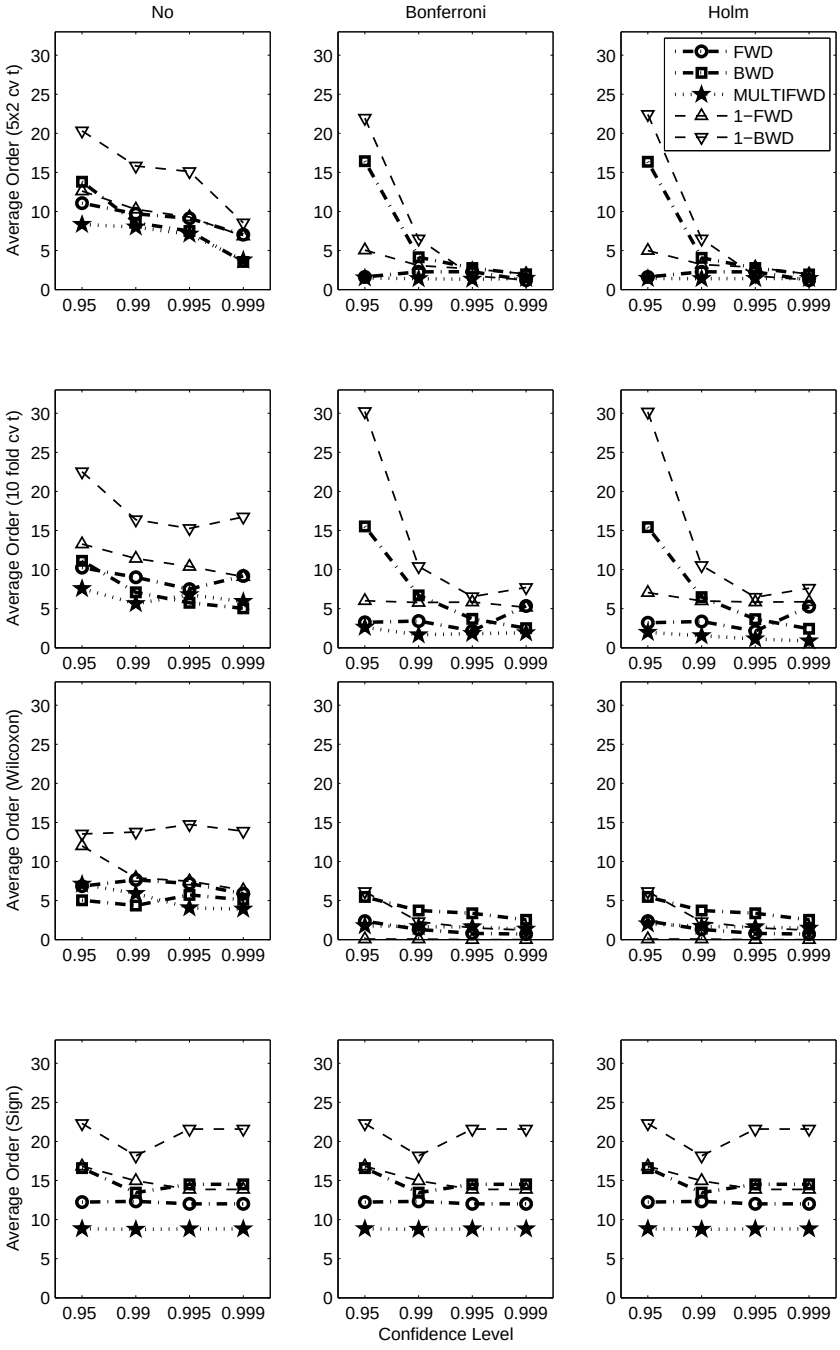


Fig. 12. The average order of MOST variants for classification datasets.

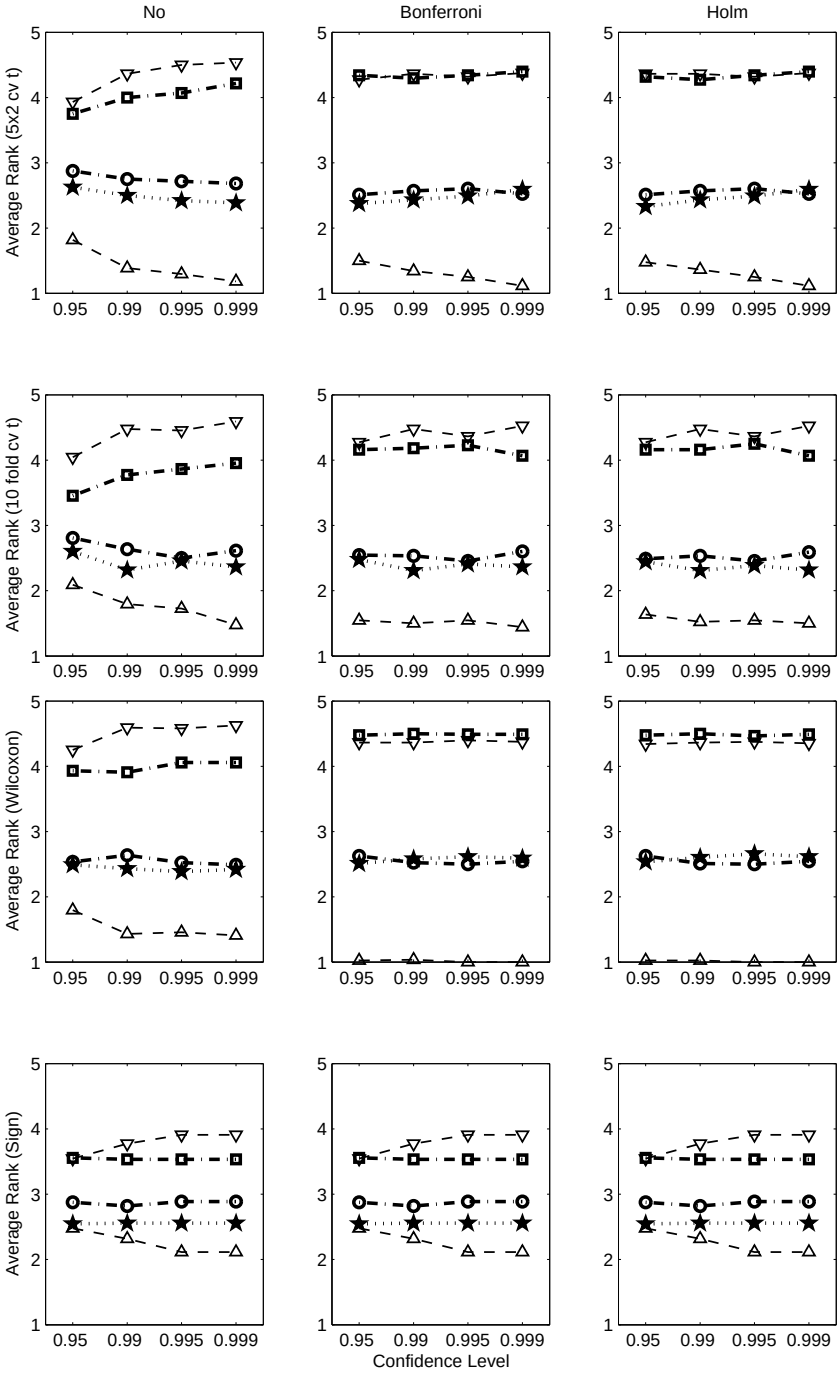


Fig. 13. The average rank of MOST variants for classification datasets.

5. Acknowledgments

All the software used for the simulations, except cascade correlation, is developed by the authors. For cascade correlation, we have used the code of Matt White from Carnegie Mellon University, which is a re-engineered version of the C port, by Scott Crowder, of the original Lisp code by Scott Fahlman[16].

This work has been supported by the Turkish Academy of Sciences in the framework of the Young Scientist Award Program (EA-TÜBA-GEBİP/2001-1-1), Boğaziçi University Scientific Research Project 05HA101 and Turkish Scientific Technical Research Council TÜBİTAK EEEAG 104E079.

References

1. O. Aran and E. Alpaydın, "Incremental neural network construction algorithms for training multilayer perceptrons," in *Artificial Neural Networks and Neural Information Processing - ICANN/ICONIP'03, Istanbul, Turkey*, 2003.
2. S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, no. 1, pp. 1–58, 1992.
3. E. Alpaydın, "GAL: Networks that grow when they learn and shrink when they forget," Tech. Rep. TR-91-032, ICSI, Berkeley, CA, 1991.
4. T. Kwok and D. Yeung, "Constructive algorithms for structure learning in feedforward neural networks for regression problems," *IEEE T. Neural. Networ.*, vol. 8, no. 3, pp. 630–645, 1997.
5. R. Reed, "Pruning algorithms - A survey," *IEEE T. Neural. Networ.*, vol. 4, no. 5, pp. 740–747, 1993.
6. Ulrich Anders and Olaf Korn, "Model selection in neural networks," *Neural Networks*, vol. 12, no. 2, pp. 309–323, 1999.
7. R. Setiono, "Feedforward neural network construction using cross validation," *Neural Computation*, vol. 13, no. 12, pp. 2865–2877, 2001.
8. R. Kallel, M. Cottrell, and V. Vigneron, "Bootstrap for neural model selection," *Neurocomputing*, vol. 48, no. 1-4, pp. 175–183, 2002.
9. A. Lendasse, G. Simon, V. Wertz, and M. Verleysen, "Fast bootstrap methodology for regression model selection," *Neurocomputing*, vol. 64, pp. 161–181, 2005.
10. R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Fourteenth International Joint Conference on Artificial Intelligence (IJCAI), San Mateo, CA*, 1995, pp. 1137–1145.
11. J. H. Friedman and W. Stuetzle, "Projection pursuit regression," *Journal of the American Statistics Association*, vol. 76, no. 376, pp. 817–823, 1981.
12. T. Ash, "Dynamic node creation in backpropogation networks," *Connection Science*, vol. 1, no. 4, pp. 365–375, 1989.
13. M. R. Frean, "The upstart algorithm: A method for constructing and training feedforward neural networks," *Neural Computation*, vol. 2, no. 2, pp. 198–209, 1990.
14. Guang-Bin Huang, Yan-Qiu Chen, and Haroon A. Babri, "Classification ability of single hidden layer feedforward neural networks," *IEEE T. Neural. Networ.*, vol. 11, no. 3, pp. 799–801, 2000.
15. Guang-Bin Huang, "Learning capability and storage capacity of two-hidden-layer feedforward networks," *IEEE T. Neural. Networ.*, vol. 14, no. 2, pp. 274–281, 2003.
16. S. E. Fahlman and C. Leibiere, "The cascade-correlation learning architecture," *Advances In Neural Information Processing Systems*, vol. 2, pp. 524–532, 1990.

17. M. Lehtokangas, "Fast initialization for cascade-correlation learning," *IEEE T. Neural. Networ.*, vol. 10, no. 2, pp. 410–414, 1999.
18. M. Lehtokangas, "Modified cascade-correlation learning for classification," *IEEE T. Neural. Networ.*, vol. 11, no. 3, pp. 795–798, 2000.
19. J. J. T. Lahnajarvi, M. I. Lehtokangas, and J. P. P. Saarinen, "Evaluation of constructive neural networks with cascaded architectures," *Neurocomputing*, vol. 48, no. 1-4, pp. 573–607, 2002.
20. J. Platt, "A resource allocating network for function interpolation," *Neural Computation*, vol. 3, no. 2, pp. 213–225, 1990.
21. Mu-Chun Su, Jonathan Lee, and Kuo-Lung Hsieh, "A new ARTMAP-based neural network for incremental learning," *Neurocomputing*, vol. 69, no. 16–18, pp. 2284–2300, 2006.
22. C. Constantinopoulos and A. Likas, "An incremental training method for the probabilistic RBF network," *IEEE T. Neural. Networ.*, vol. 17, no. 4, pp. 966–974, 2006.
23. Liying Ma and K. Khorasani, "Constructive feedforward neural networks using hermite polynomial activation functions," *IEEE T. Neural. Networ.*, vol. 16, no. 4, pp. 821–833, 2005.
24. T. Kwok and D. Yeung, "Objective functions for training new hidden units in constructive neural networks," *IEEE T. Neural. Networ.*, vol. 8, no. 5, pp. 1131–1148, 1997.
25. Liying Ma and K. Khorasani, "New training strategies for constructive neural networks with application to regression problems," *Neural Networks*, vol. 17, no. 4, pp. 589–609, 2004.
26. Guang-Bin Huang and Lei Chen, "Convex incremental extreme learning machine," *Neurocomputing*, vol. 70, pp. 3056–3062, 2007.
27. Guang-Bin Huang, Lei Chen, and Chee-Kheong Siew, "Universal approximation using incremental networks with random hidden nodes," *IEEE T. Neural. Networ.*, vol. 17, no. 4, pp. 879–892, 2006.
28. Guang-Bin Huang and Lei Chen, "Enhanced random search based incremental extreme learning machine," *Neurocomputing*, 2008.
29. M. F. Tenorio and W. T. Lee, "Self-organizing network for optimum supervised learning," *IEEE T. Neural. Networ.*, vol. 1, no. 1, pp. 100–110, 1990.
30. G. Castellano and A. M. Fanelli, "An iterative pruning algorithm for feedforward neural networks," *IEEE T. Neural. Networ.*, vol. 8, no. 3, pp. 519–531, 1997.
31. P. V. S. Ponnapalli, K. C. Ho, and M. Thomson, "A formal selection and pruning algorithm for feedforward artificial neural network optimization," *IEEE T. Neural. Networ.*, vol. 10, no. 4, pp. 964–968, 1999.
32. P. Lauret, E. Fock, and T. A. Mara, "A node pruning algorithm based on a fourier amplitude sensitivity test method," *IEEE T. Neural. Networ.*, vol. 17, no. 2, pp. 273–293, 2006.
33. T. M. Nabhan and A. Y. Zomaya, "Toward generating neural network structures for function approximation," *Neural Networks*, vol. 7, no. 1, pp. 89–99, 1994.
34. R. Parekh, J. Yang, and V. Honavar, "Constructive neural-network learning algorithms for pattern classification," *IEEE T. Neural. Networ.*, vol. 11, no. 2, pp. 436–451, 2000.
35. Shimon Cohen and Nathan Intrator, "On different model selection criteria in a forward and backward regression hybrid network," *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, vol. 18, no. 5, pp. 847–865, 2004.
36. J. Paetz, "Reducing the number of neurons in radial basis function networks with dynamic decay adjustment," *Neurocomputing*, vol. 62, pp. 79–91, 2004.

37. A. L. I. Oliveira, B. J. M. Melo, and S. R. L. Meira, "Improving constructive training of RBF networks through selective pruning and model selection," *Neurocomputing*, vol. 64, pp. 537–541, 2005.
38. Guang-Bin Huang, "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," *IEEE T. Neural. Networ.*, vol. 16, no. 1, pp. 57–67, 2005.
39. J. I. Arribas and J. Cid-Sueiro, "A model selection algorithm for a posteriori probability estimation with neural networks," *IEEE T. Neural. Networ.*, vol. 16, no. 4, pp. 799–809, 2005.
40. Ajith Abraham, "Meta learning evolutionary artificial neural networks," *Neurocomputing*, vol. 56, pp. 1–38, 2004.
41. F. H. F. Leung, H. K. Lam, S. H. Ling, and P. K. S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE T. Neural. Networ.*, vol. 14, no. 1, pp. 79–88, 2003.
42. Qin-Yu Zhu, A.K. Qin, P.N. Suganthan, and Guang-Bin Huang, "Evolutionary extreme learning machine," *Pattern Recognition*, vol. 38, no. 10, pp. 1759–1763, 2005.
43. Kang Li and Jian-Xun Peng, "System oriented neural networks - problem formulation, methodology and application," *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, vol. 20, no. 2, pp. 143–158, 2006.
44. C. Macleod and G. M. Maxwell, "Incremental evolution in ANNs: Neural nets which grow," *Artificial Intelligence Review*, vol. 16, no. 3, pp. 201–224, 2001.
45. C. Xiang, S. Q. Ding, and T. H. Lee, "Geometrical interpretation and architecture selection of MLP," *IEEE T. Neural. Networ.*, vol. 16, no. 1, pp. 84–96, 2005.
46. J. P. Vila, V. Wagner, and P. Neveu, "Bayesian nonlinear model selection and neural networks: A conjugate prior approach," *IEEE T. Neural. Networ.*, vol. 11, no. 2, pp. 265–278, 2000.
47. L. Xu, "BYY learning, regularized implementation, and model selection on modular networks with one hidden layer of binary units," *Neurocomputing*, vol. 51, pp. 277–301, 2003.
48. J. Ma, T. Wanga, and L. Xu, "A gradient BYY harmony learning rule on gaussian mixture with automated model selection," *Neurocomputing*, vol. 56, pp. 481 – 487, 2004.
49. Liying Ma and K. Khorasani, "Application of adaptive constructive neural networks to image compression," *IEEE T. Neural. Networ.*, vol. 13, no. 5, pp. 1112–1126, 2002.
50. Anil Kumar Ghosh and Smarajit Bose, "Feature extraction for classification using statistical networks," *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, vol. 21, no. 7, pp. 1103–1126, 2007.
51. A. Carlevarino, R. Martinotti, G. Metta, and G. Sandini, "An incremental growing neural network and its application to robot control," in *International Joint Conference on Neural Networks (IJCNN'00)*, 2000, vol. 5, p. 5323.
52. Kyung-Joong Kim and Sung-Bae Cho, "Evolved neural networks based on cellular automata for sensory-motor controller," *Neurocomputing*, vol. 69, no. 16–18, pp. 2193–2207, 2006.
53. M. Ghiassi and H. Saidane, "A dynamic architecture for artificial neural networks," *Neurocomputing*, vol. 63, pp. 397–413, 2005.
54. D. Srinivasan, X. Jin, and R. L. Cheu, "Adaptive neural network models for automatic incident detection on freeways," *Neurocomputing*, vol. 64, pp. 473–496, 2005.
55. Cheng-An Hung and Sheng-Fuu Lin, "An incremental learning neural network for pattern classification.," *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, vol. 13, no. 6, pp. 913–928, 1999.

56. Hanen Borchani, Nahla Ben Amor, and Fedia Khalfallah, "Learning and evaluating bayesian network equivalence classes from incomplete data," *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, vol. 22, pp. 253–278, 2008.
57. T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Springer, New York, 2001.
58. O. T. Yıldız and E. Alpaydın, "Ordering and finding the best of $K > 2$ supervised learning algorithms," *IEEE T. Pattern. Anal.*, vol. 28, no. 3, pp. 392–402, 2006.
59. A. Dean and D. Voss, *Design and Analysis of Experiments*, Springer, New York, 1999.
60. S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, pp. 65–70, 1979.
61. C. L. Blake and C. J. Merz, "UCI repository of machine learning databases," .
62. "Delve datasets," http://www.cs.toronto.edu/ delve/data/datasets.html.
63. "Statlib datasets," http://lib.stat.cmu.edu/.
64. Ellis Horwood, *Machine Learning, Neural and Statistical Classification*, 1994, See also http://www.ncc.up.pt/liacc/ML/statlog/datasets.html.
65. T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms," *Neural Computation*, vol. 10, no. 7, pp. 1895–1923, 1998.