In *Proceedings of the 1997 Sian Ka'an International Workshop on Neural Networks and Neurocontrol.*

Real-Valued Schemata Search Using Statistical Confidence

D. Randall Wilson[1] *and Tony R. Martinez*[2]

Abstract. Many neural network models must be trained by finding a set of real-valued weights that yield high accuracy on a training set. Other learning models require weights on input attributes that yield high leave-one-out classification accuracy in order to avoid problems associated with irrelevant attributes and high dimensionality. In addition, there are a variety of general problems for which a set of real values must be found which maximize some evaluation function. This paper presents an algorithm for doing a schemata search over a real-valued weight space to find a set of weights (or other real values) that yield high values for a given evaluation function. The algorithm, called the *Real-Valued Schemata Search* (RVSS), uses the BRACE statistical technique [Moore & Lee, 1993] to determine when to narrow the search space. This paper details the RVSS approach and gives initial empirical results.

1. Introduction

There are many situations in which it is desirable to find a set of values that maximize some evaluation measure. For example, attribute weights for radial basis function networks [Broomhead & Lowe, 1988] or instance-based learning algorithms [Aha, Kibler & Albert, 1991] can be used to reduce sensitivity to irrelevant attributes if attribute weights can be found that yield high leave-one-out classification accuracy. Multilayer perceptrons [Rummelhart & McClelland, 1986] are trained in such a way as to yield high training set accuracy (as indicated by low total sum-squared error) given a good set of real-valued weights. Many other problems also require a set of real-valued quantities that yield high accuracy or result in good scores according to some other real-valued evaluation measure.

Genetic algorithms [Spears et al., 1993] can be used to find sets of values without necessarily knowing what the gradient is at each sampled point in the weight space. However, genetic algorithms have many parameters that affect how successful they are in a particular domain. If not tuned carefully, they can quickly generate a set of homogeneous individuals that are very similar and thus ignore much of the search space.

Moore & Lee [1993] presented an algorithm called BRACE ("blocking race") which is used to determine when sufficient statistical evidence has been gathered to determine whether one "model" is worse than (or at least no better than) another. The algorithm can be used in situations when it is necessary to choose between two or more learning models, parameter settings, hypotheses, etc., and to know how much testing must be done before one of the competitors can be excluded from consideration.

Moore & Lee applied the BRACE algorithm to the problem of *attribute selection,* using a schemata search to avoid problems associated with the more standard approaches of *forward selection* and *backward elimination* [Miller, 1990]. Attribute selection involves deciding which of the available attributes in a set of training data should be used by a learning algorithm. This problem is equivalent to searching for a set of binary-valued attribute weights for use in some other learning algorithm. Their algorithm had success in finding such binary-valued weights, but it is not applicable to finding real-valued weights.

This paper presents a technique called the *Real-Valued Schemata Search* (RVSS) for doing a schemata search on a set of real-valued quantities by using statistical techniques similar to those used by BRACE in order to decide when to narrow the search space. Though the remainder of the paper refers to the real-valued quantities as *weights*, the same technique can be used to search for good settings for a wide variety of real-valued quantities for which there is a realiable evaluation function.

Section 2 presents the RVSS algorithm, including how the real-valued search proceeds and how statistical tests determine when to narrow the search space. Section 3 discusses empirical observations and further potential applications of the algorithm. Section 4 provides conclusions and suggestions for future research directions.

2. Real-Valued Schemata Search

This section presents a brief overview of the *Real-Valued Schemata Search* (RVSS) algorithm, followed by a more detailed exposition of the algorithm.

[1]*fonix* Corporation
180 W. Election Road
Draper, UT 84020, USA
E-mail: WilsonR@fonix.com
WWW: http://axon.cs.byu.edu/~randy

[2]Neural Network & Machine Learning Laboratory
Computer Science Department
Brigham Young University
Provo, UT 84602, USA
E-mail: martinez@cs.byu.edu
WWW: http://axon.cs.byu.edu

Given a vector $\boldsymbol{w}$ of m weights $w_1...w_m$, an evaluation function f (such as sum-squared error of the outputs of a learning algorithm using the weights in $\boldsymbol{w}$), and a training set T of n training instances, RVSS attempts to find values for the weights such that $f(\boldsymbol{w})$ yields as high a value as possible. The maximum and minimum possible values for each weight w_i are denoted as max_i and min_i, respectively.

The RVSS algorithm begins with an initial range $min_i....max_i$ for each weight, such as 0...1 or -1...1. It then divides each range into d subranges, where d is a small number such as 2 or 4. (Overhead of the RVSS algorithm increases by a factor of d^2, so small values are preferable.)

During each iteration, a random value is picked for all of the weights, and then each individual weight is optimized at a time. Each subrange of the weight competes in a greedy search to see if values in the subrange tend to yield higher values for $f(\boldsymbol{w})$ than other subranges. Whenever a subrange is found (with high confidence) to be no better than any other subrange (i.e., significantly worse or significantly similar), it is dropped from the race. If only one subrange remains then the *max* and *min* for the weight are narrowed to cover only the winning subrange, and this new smaller range is subdivided again unless the winning subrange is smaller than some threshold *minWidth* (e.g., .0001), at which point the center of the winning subrange is chosen as the permanent value for this weight, and no further alternates are considered. Once all the weights have been permanently chosen, the algorithm is finished.

The remainder of this section provides a more detailed explanation of the RVSS algorithm, including a description of the statistical tests needed to determine whether one range is no better than another. The algorithm proceeds as follows.

A random instance t is chosen from the training set T. Random values are chosen for each weight that is still in the race, with the restriction that the random value for each weight must be in one of the subranges that is still racing. This becomes the “global” weight vector $\boldsymbol{w}$ that will be used in testing each individual weight.

Next, each individual weight w_i is changed to be a random value in each of its subranges r that are still racing, while holding the values of all other weights equal to the “global” vector chosen above. The new weight vector is evaluated on the training instance t using $f(\boldsymbol{w})$, and the result is saved as $score_r$. The value of w_i for each subrange $r = 1...d$ is given by

$$w_{i,r} = \min_{i,r} + \mathrm{width}_{i,r} * v_i \qquad (1)$$

where

- $min_{i,r} = min_i + width_i * (r\text{-}1)$, and is the minimum value of subrange r
- $width_{i,r} = (max_i - min_i) / d$, and is the width of subrange r
- v_i is a random value, $0 \le v_i < 1$. While a different value of v_i are used for each weight, the same value of v_i is used for all subranges of a single weight so that the statistics are better correlated.

As an example, consider a weight w_i with min_i=0, max_i=1. If d=2, the two ranges would be 0 to .5 and .5 to 1. If a random value v_i is chosen to be .4, then for the first subrange, $w_{i,r}$ would be set to 0+.5*.4=.2, and for the second subrange, the value would be .5+.5*.2=.7.

Once $score_r$ has been found for all subranges of a weight, these scores are used to update statistics kept between each pair of subranges so that the algorithm can determine when one subrange can be dropped from the race. Specifically, for each subrange r and each subrange $s > r$ of weight i, the following statistics are kept:

- $num_{i,r,s}$, the number of times the statistics have been updated (which is equal to the number of training instances that have been processed since the race began or was restarted)
- $sum_{i,r,s}$, the sum of *differences* between scores for subranges r and s
- $sumSquared_{i,r,s}$, the sum of the squares of the differences.

For each training instance, *num* is incremented by one, the difference $score_r - score_s$ is added to *sum*, and $(score_r - score_s)^2$ is added to *sumSquared*. These quantities all start out at zero, and are reset to zero whenever one subrange wins its race and the race is restarted.

After these statistics are updated, they are used to help determine if one or more subranges can be dropped from the race. If one subrange results in significantly lower scores than another, it can be dropped from the race. Also, if two ranges are significantly identical, the lower one can be dropped from the race. Thus if we are confident that one range is *not* higher than another, it can be dropped.

This test can be made by seeing if

$$P(\mu_{s,r} < -\gamma) < \alpha \qquad (2)$$

where $\mu_{s,r}$ is the true mean of the difference $score_r - score_s$, γ is an error term indicating the amount of allowable error (e.g., γ=.0001), and α

is the probability of getting the observed mean by accident (e.g., α=.001, or 99.9% confident that the observed mean *could not* have happened by chance). If range s is actually better than range r, then the average difference (i.e., the observed mean, or $sum_{i,r,s} / num_{i,r,s}$) should be positive, and once enough samples have been seen, the probability of having a true mean less than $-\gamma$ will become very low. Once this probability is smaller than α, we are sufficiently confident that range s is not worse than range r, and thus range r can be dropped.

If, on the other hand, the average difference is negative, then we test to see if $P(\mu_{s,r} > \gamma) < \alpha$ (or, equivalently, if $P(-\mu_{s,r} < -\gamma) < \alpha$). If this is true, then there is very little chance that the true mean is positive given the statistical evidence that it is negative, and thus we conclude with 100*(1-α)% confidence that range r is not worse than range s, so s can be dropped from the race.

To see the need for the error term γ, consider what happens when the true mean is zero, i.e., the two ranges yield identical results. In this case $P(\mu_{s,r} < 0)$ will remain at about .5 indefinitely, while $P(\mu_{s,r} < -\gamma)$ will drop as the number of samples increases and the variance decreases. Thus the introduction of this error term allows the elimination of both worse subranges and identical subranges.

In order to calculate these probabilities, a student's t-distribution can be used, with the value of t set to:

$$t = \frac{|\mu| - (-\gamma)}{\sqrt{\mathrm{var}/n}} = \frac{\left|sum_{i,r,s} / num_{i,r,s}\right| + \gamma}{\sqrt{\mathrm{var}_{i,r,s} / num_{i,r,s}}} \tag{3}$$

where $\mathrm{var}_{i,r,s}$ is the variance of the observed differences. (The absolute values are used to handle the case where the mean is negative). The variance of a set of values x_i, i=1..n, is given as:

$$\mathrm{var} = \frac{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}{n-1} \tag{4}$$

which uses the average value of x during each part of the summation. In order to process the observed differences incrementally it is necessary to rework the above formula using the following derivation.

$$\mathrm{var} = \frac{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}{n-1} = \frac{\sum_{i=1}^{n}\left(x_i^2 - 2 \cdot x_i \cdot \bar{x} + \left(\bar{x}\right)^2\right)}{n-1}$$

$$= \frac{\sum_{i=1}^{n} x_i^2 - 2 \cdot \bar{x} \cdot \sum_{i=1}^{n} x_i + \sum_{i=1}^{n}\left(\bar{x}\right)^2}{n-1} \tag{5}$$

In terms of our three stored statistics, *num*, *sum*, and *sumSquared*, the variance can thus be expressed as

$$\mathrm{var} = \frac{sumSquared - 2 \cdot sum \cdot avg + num \cdot avg^2}{n-1} \tag{6}$$

where $avg = sum / num$ and is the running average, or the average so far.

Once the value of t is found, an interpolated table look-up is done to find $t_{1-\alpha,n-1}$, where (1-α) is the confidence level, and n-1 is the number of *degrees of freedom*. If $t > t_{1-\alpha,n-1}$ then there is a very small probability that the observed difference is from chance (i.e., $P<\alpha$), so the difference is statistically significant and the lower of the two sugranges can be thrown out. Minimum necessary values of $t_{1-\alpha,n-1}$ for several values of α and n are given in the appendix of this paper.

When all but one subrange has been thrown out of the race for a weight, that weight's range is reduced to span only the winning subrange, which is then subdivided into d smaller subranges. The statistics for this weight and all other weights are then cleared (i.e., *sumSquared*, *sum*, and *num* are all set to zero). If the new smaller range is small enough (e.g., when the width < *minWidth* = .0001), then the weight is fixed at the center point of the new range, and is no longer changed by the randomization for each training instance, nor does the algorithm try any further alternative values for the weight.

If, on the other hand, the probability P is not less than α, the weight w_i is reset to the "global" random value picked above, and the algorithm continues with the next weight. After all weights have been tested, a new training instance is chosen, new random weights are chosen (except for those weights that have been completely dropped from the race), and the process repeats until the range of all of the weights drops below *minWidth*, thus causing all the weights to become fixed.

Figure 1 shows pseudo-code for this algorithm. The algorithm takes three parameters, a, g, and *minWidth*, in addition to a training set and an evaluation function used to evaluate a given weight vector on a single training instance. It returns a weight vector upon completion.

```
LearnRVSS(T, alpha, gamma, minWidth):w[1..m]
In:  training set T
     alpha, the probability of an erroneous decision
     gamma, a small allowable error
     minWidth, the smallest width of weight range that is subdivided.
Out: w[1..m]

For i=1..m, r=1..d, and s=r+1..d
     set num[i][r][s], sum[i][r][s] and sumSquared[i][r][s] to 0

While there are still weights left in the race
     Let inst = a random instance from the training set T.
     # Pick random weights:
     For each weight i that is still in the race
          let w[i]=random value in any one of its divisions that is still racing.
     # Test individual weight settings
     For each weight i that is still not permanently fixed
          Pick a random value v in the range 0...1
          For each range r of weight i that is still racing
               Let w[i]=min[i][r]+width[i][r]*v
               Let score[r]=f(w[1..m], inst), i.e., evaluate w[1..m] on training instance 'inst'
          For each range r=1..d of weight i that is still racing
               For each other range s=(r+1)..d of weight i that is still racing
                    Add 1 to num[i][r][s]
                    Add (score[r]-score[s]) to sum[i][r][s]
                    Add (score[r]-score[s])^2 to sumSquared[i][r][s]
                    Let avg = sum[i][r][s] / num[i][r][s]
                    Let var = (sumSquared[i][r][s]-2*sum[i][r][s]*avg+num[i][r][s]*avg*avg)
                                   / (num[i][r][s]-1)
                    Let t = (|avg|+gamma) / sqrt(var/num[i][r][s])
                    If t > TDistrib(alpha,n-1), (i.e., P(true_avg < -gamma) < alpha)
                         then if avg ≥ 0 then remove range s from the race
                         else remove range r from the race
          If there are not at least two ranges still in the race for weight i
               Set min[i]=min[i][winning r] and max[i]=max[i][winning r]
               If the new width < minWidth
                    then set w[i]=(min[i]+max[i])/2, and fix it there permanently
               Clear statistics for all weights to restart their races
EndWhile
Return w[1..m]
```

Figure 1. Pseudo-code for RVSS learning algorithm.

3. Applications of RVSS

The Real-Valued Schemata Search algorithm addresses a very general problem, i.e., that of finding a vector of real values that yields a high value on some evaluation function. Thus, its applications are potentially quite numerous.

As an example of how this technique can be used, we have modified an instance-based learning algorithm [Wilson & Martinez, 1997] such that it uses attribute weights generated by RVSS in order to help it to be more robust in the presence of irrelevant attributes and redundant attributes. In experiments on 31 datasets from the Machine Learning Database Repository at the University of California Irvine [Merz & Murphy, 1996], four irrelevant input attributes were added to each dataset, which can degrade generalization accuracy [Wilson & Martinez, 1996]. For each dataset, 10-fold cross-validation was used to obtain the average generalization accuracy for a *k*-nearest neighbor classifier with equal attribute weights (i.e., all weights=1). An identical classifier that used RVSS-generated attribute weights was also tested, and the average accuracy for each dataset for both of these algorithms is given in Table I.

Dataset	kNN	kNN/RVSS
Anneal	90.5	**92.0**
Australian	83.3	83.3
Breast Cancer (Wi)	95.9	95.9
Bridges	**54.3**	50.6
Crx	83.9	83.9
Echocardiogram	93.2	93.2
Flag	59.6	59.6
Glass	**50.5**	49.5
Heart	83.3	83.3
Heart.Cleveland	77.2	77.2
Heart.Hungarian	**82.6**	82.3
Heart.Long Beach	71.0	**72.0**
Heart.More	74.4	74.4
Heart.Swiss	93.5	93.5
Hepatitis	83.9	**85.2**
Horse Colic	67.8	**71.8**
Image Segmentation	86.0	**88.3**
Ionosphere	83.7	**86.0**
Iris	82.0	**90.7**
Led-Creator	**69.2**	68.5
Led+17	68.5	**69.2**
Liver (Bupa)	55.1	55.1
Pima Indians Diabetes	70.4	70.4
Promoters	**84.9**	84.0
Sonar	**79.3**	78.8
Soybean (Large)	85.7	85.7
Vehicle	59.2	60.0
Voting	96.1	**96.3**
Vowel	56.6	**74.2**
Wine	92.2	**93.3**
Zoo	94.4	94.4
Avg	**77.7**	**78.8**

Table I. Example application of RVSS. Generalization accuracy of a *k*-nearest neighbor classifier on with added irrelevant attributes.

As can be seen from the table, the attribute weights provided by the RVSS algorithm yielded higher generalization accuracy than the non-weighted scheme in 11 cases and lower accuracy in only 6. Using the RVSS-generated weights resulted in over 1% higher accuracy on average than using fixed weights.

The RVSS technique was also applied somewhat less successfully to the problem of finding weights for multilayer perceptrons. The weights generated by RVSS did yield higher generalization accuracy and lower total sum-squared error as the algorithm progressed, but the algorithm was not able to achieve the levels of accuracy attained by the backpropagation learning algorithm.

However, there are many optimization problems for which a robust algorithm such as error backpropagation does not exist for tuning parameters, weights, or other values. In such situations there is still a potential for success using the RVSS algorithm or variations thereof.

4. Conclusions

The real-valued schemata search (RVSS) algorithm extends the binary-valued schemata search described by Moore & Lee [1993] to real-valued domains. It uses a "blocking race" (BRACE) statistical test to decide when one portion of the search space can be eliminated so as to narrow the search space.

The work presented here is still in its preliminary stages, but the RVSS algorithm has been applied with some success to weighting of input attributes in an instance-based learning algorithm.

Future work necessary for the RVSS algorithm includes:

- Incorporation of the ability to back up and re-widen the search space,
- Direction on what values of α, γ, and $minWidth$ to use,
- Careful analysis of how many subranges to use,
- Identification of additional problems that could benefit from this technique, such as finding weights for radial basis function networks [Broomhead & Lowe, 1988], and
- Incorporation of the statistical tests into other techniques such as genetic algorithms.

References

Aha, David W., Dennis Kibler, Marc K. Albert, (1991). "Instance-Based Learning Algorithms," *Machine Learning*, vol. 6, pp. 37-66.

Broomhead, D. S., and D. Lowe (1988). Multi-variable functional interpolation and adaptive networks. *Complex Systems*, Vol. 2, pp. 321-355.

Merz, C. J., and P. M. Murphy, (1996). *UCI Repository of Machine Learning Databases*. Irvine, CA: University of California Irvine, Department of Information and Computer Science. Internet: http://www.ics.uci.edu/~mlearn/MLRepository.html.

Moore, Andrew W., and Mary S. Lee, (1993). "Efficient Algorithms for Minimizing Cross Validation Error," In *Machine Learning: Proceedings of the Eleventh International Conference*, Morgan Kaufmann.

Rumelhart, D. E., and J. L. McClelland, *Parallel Distributed Processing*, MIT Press, 1986.

Spears, William M., Kenneth A. De Jong, Thomas Bäck, David B. Fogel, and Hugo de Garis, (1993). "An Overview of Evolutionary Computation," *Proceedings of the European Conference on Machine Learning*, pp. 442-459.

Wilson, D. Randall, and Tony R. Martinez, (1996). "Instance-Based Learning with Genetically Derived Attribute Weights," *International Conference on Artificial Intelligence, Expert Systems and Neural Networks (AIE'96),* pp. 11-14.

Wilson, D. Randall, and Tony R. Martinez, (1997). "Distance Weighting and Confidence in Instance-Based Learning," submitted to *Computational Intelligence*.

Appendix. Student's *t* Distribution.

	(1−α)									
n-1	**0.6**	**0.7**	**0.8**	**0.9**	**0.95**	**0.975**	**0.99**	**0.995**	**0.999**	**0.9995**
1	0.325	0.727	1.376	3.078	6.314	12.710	31.820	63.660	318.300	636.600
2	0.289	0.617	1.061	1.886	2.920	4.303	6.965	9.925	22.330	31.600
3	0.277	0.584	0.978	1.638	2.353	3.182	4.541	5.841	10.220	12.940
4	0.271	0.569	0.941	1.533	2.132	2.776	3.747	4.604	7.173	8.610
5	0.267	0.559	0.920	1.476	2.015	2.571	3.365	4.032	5.893	6.859
6	0.265	0.553	0.906	1.440	1.943	2.447	3.143	3.707	5.208	5.959
7	0.263	0.549	0.896	1.415	1.895	2.365	2.998	3.499	4.785	5.405
8	0.262	0.546	0.889	1.397	1.860	2.306	2.896	3.355	4.501	5.041
9	0.261	0.543	0.883	1.383	1.833	2.262	2.821	3.250	4.297	4.781
10	0.260	0.542	0.879	1.372	1.812	2.228	2.764	3.169	4.144	4.587
11	0.260	0.540	0.876	1.363	1.796	2.201	2.718	3.106	4.025	4.437
12	0.259	0.539	0.873	1.356	1.782	2.179	2.681	3.055	3.930	4.318
13	0.259	0.538	0.870	1.350	1.771	2.160	2.650	3.012	3.852	4.221
14	0.258	0.537	0.868	1.345	1.761	2.145	2.624	2.977	3.787	4.140
15	0.258	0.536	0.866	1.341	1.753	2.131	2.602	2.947	3.733	4.073
16	0.258	0.535	0.865	1.337	1.746	2.120	2.583	2.921	3.686	4.015
17	0.257	0.534	0.863	1.333	1.740	2.110	2.567	2.898	3.646	3.965
18	0.257	0.534	0.862	1.330	1.734	2.101	2.552	2.878	3.611	3.922
19	0.257	0.533	0.861	1.328	1.729	2.093	2.539	2.861	3.579	3.883
20	0.257	0.533	0.860	1.325	1.725	2.086	2.528	2.845	3.552	3.850
21	0.257	0.532	0.859	1.323	1.721	2.080	2.518	2.831	3.527	3.819
22	0.256	0.532	0.858	1.321	1.717	2.074	2.508	2.819	3.505	3.792
23	0.256	0.532	0.858	1.319	1.714	2.069	2.500	2.807	3.485	3.767
24	0.256	0.531	0.857	1.318	1.711	2.064	2.492	2.797	3.467	3.745
25	0.256	0.531	0.856	1.316	1.708	2.060	2.485	2.787	3.450	3.725
26	0.256	0.531	0.856	1.315	1.706	2.056	2.479	2.779	3.435	3.707
27	0.256	0.531	0.855	1.314	1.703	2.052	2.473	2.771	3.421	3.690
28	0.256	0.530	0.855	1.313	1.701	2.048	2.467	2.763	3.408	3.674
29	0.256	0.530	0.854	1.311	1.699	2.045	2.462	2.756	3.396	3.659
30	0.256	0.530	0.854	1.310	1.697	2.042	2.457	2.750	3.385	3.646
40	0.255	0.529	0.851	1.303	1.684	2.021	2.423	2.704	3.307	3.551
50	0.255	0.528	0.849	1.298	1.676	2.009	2.403	2.678	3.262	3.495
60	0.254	0.527	0.848	1.296	1.671	2.000	2.390	2.660	3.232	3.460
80	0.254	0.527	0.846	1.292	1.664	1.990	2.374	2.639	3.195	3.415
100	0.254	0.526	0.845	1.290	1.660	1.984	2.365	2.626	3.174	3.389
200	0.254	0.525	0.843	1.286	1.653	1.972	2.345	2.601	3.131	3.339
500	0.253	0.525	0.842	1.283	1.648	1.965	2.334	2.586	3.106	3.310
inf	0.253	0.524	0.842	1.282	1.645	1.960	2.326	2.576	3.090	3.291

Table II. Student *t* distribution. Each entry is $t_{1-\alpha,n-1}$ where α is the probability of getting a value of *t* larger than the entry given *n*-1 degrees of freedom.