A Bayesian CMAC for High Assurance Learning

James L. Carroll and Kevin D. Seppi
3350 TMCB Brigham Young University, Provo UT 84602, USA
jlcarroll@gmail.com and kseppi@cs.byu.edu

Abstract— **We analyze the drawbacks to using ANNs in high assurance systems and propose a solution based upon a Bayesian approach with a specific network topology that can be solved in closed form. The Bayesian approach leads to better answers in the traditional sense, while also allowing us to quantify risk and deal with it in a reasonable manner. We demonstrate this approach on several synthetic functions and the Abalone data set.**

I. INTRODUCTION

Artificial Neural Networks (ANNs) are a common technique for learning classification and control from examples. However, there are several drawbacks to using ANNs in high assurance systems. These drawbacks involve issues with output uncertainty, weight interpretability, output generalization, expert knowledge inclusion, and the sufficiency of training data. We will discuss each of these problems in greater detail, and then propose solutions based on a Bayesian adaptation of the CMAC [1] network topology.

The most common complaint about ANNs in high assurance systems is *output uncertainty*. It is tempting to try and remove all uncertainty in situations where the wrong response can have disastrous consequences. Unfortunately, this is not always possible. There are many sources of uncertainty including: an inappropriate learning algorithm, intrinsic non-determinism in the function to be learned, and insufficient or inappropriate training data. Since it is usually impossible to remove all uncertainty (whether or not an ANN is being used) we believe that the best solution to these problems is to reduce uncertainty where possible through user input, and to quantify the uncertainty that can not be removed. Then it is possible to compute the risk involved in each decision when made in the context of that uncertainty using the principles of decision theory.

Traditional ANNs return only the answer that they consider most probable, but do not provide measures of their confidence in that answer. This makes traditional ANNs especially inappropriate for high assurance systems. For example, if an ANN were trained on the mushroom dataset [2] to predict whether a mushroom will be edible or poisonous given its appearance, how could that network be used to make decisions about eating a given mushroom? This is an example of a high assurance system since if you ate a poisonous mushroom you can get very sick, even die. If the network predicts that the mushroom is edible should you eat it? Of course that depends on how hungry you are and on how certain the network is in its prediction. Thus, in order to be effective at aiding decision making ANNs should provide more than just the output that they consider most probable, they should return a distribution over possible outputs.

Another major drawback to using ANNs for high assurance control involves *weight interpretability*. ANNs form function outputs by passing input values through a network of nodes, each with a set of interconnections and weights. The output is formed by summing the weights of interconnected nodes. Such a structure can be extremely difficult to interpret, and although we can query the network with a specific input to see its output, it is difficult to understand exactly why the network is generating the specific output. Thus, it can be difficult to look at a given network and determine that it has found a "correct" or "safe" policy.

The next major drawback to using ANNs for high assurance control involves *output generalization*. The weights of an ANN are learned through training examples rather than by specifically specifying what the outputs should be. This can actually result in improved performance, since hand coded rules can be very brittle to unexpected inputs. If the input to the network is not in its training set, the weights will hopefully return a reasonable output based on the similar examples that it has seen. The process of determining appropriate outputs in areas of the function for which there is no training data is known as generalization. Yet, this can also be problematic in high assurance control, since the network's behavior is often unpredictable when given unexpected inputs. Because the network's policy is not easily interpretable by looking at the weights, it can be difficult to predict exactly how the network will respond in each situation.

This could be solved by *expert knowledge inclusion*. The user could provide some simple safety steps such as "never do a when b is observed" this could greatly reduce the uncertainty of the system and thereby improve the applicability of ANNs to high assurance control. Unfortunately, the very problem that made the network weights un-interpretable, also makes it difficult to provide the network with such rules. Since it is difficult to determine the behavior of the network given its weights, it is equally difficult to determine a set of weights that will guarantee a certain behavior. Thus, traditional ANN learning is limited by information in the data, and it can be difficult to provide external information to the system.

The next difficulty involves the *sufficiency of training data*. ANN learning improves with training data, however it is difficult to determine how much data is necessary to be certain that the network is likely to perform within safety limits. This is also due to the lack of confidence reporting in the network outputs. Attempts to answer these questions with PAC learnability and VC dimensions have provided some loose bounds, but have failed to answer these questions in an average case.

Our proposed solution to the above problems is to recast the supervised learning paradigm in terms of decision theory and a graphical model[3]. When ANNs are built using the model as a guide, prior distributions can be used as a mechanism for providing input from a human expert, and the network will produce a true posterior distribution as its output. This provides a measure of confidence in the network outputs. In addition, our solution uses the CMAC network topology. This topology for an ANN is appealing since the weights are trivially interpretable, and since this particular network topology allows the Bayesian approach to be solved in closed form. The interpretability of the weights allows the user to provide additional information in an intuitive manner, and allows the learned policy of the network to be easily analyzed.

We will explain the graphical model of supervised learning, the CMAC topology for ANNs, and then derive a Bayesian ANN based on this network topology. We will compare the traditional CMAC with our new BCMAC and analyze the BCMAC's ability to express confidence in its results, and discuss the implications of the technique for high assurance systems.

II. The Unified Bayesian Decision Theoretic Model (UBDTM)

The supervised learning problem can be modeled as a set of random variables. These variables include feature vectors $\mathbf{x}$ that are mapped to an output value y. A list of $\mathbf{x}$, y pairs constitutes a training set. Similarly we will represent the test set with $\mathbf{x}'$, y' pairs. Usually there is also an unknown function f that either maps feature vectors to output vectors in the deterministic case $f : \mathbf{x} \to y$, or from feature vectors to a distribution over output values in the stochastic case $f : \mathbf{x} \to p(y)$. Of special interest is that we are treating the unknown function itself as a random variable, and this will have significant implications in how the model is used.

Several different relationships between the random variables in a supervised learning problem are possible, but perhaps the simplest would be the relationship represented by the graphical model in figure 1 [4]. This set of relationships is very intuitive, and implies that function outputs y and y' are dependent on the feature vectors $\mathbf{x}$ and $\mathbf{x}'$ and the function f that maps them. Note that this model of variable relationships considers $\mathbf{x}$ and f to be conditionally independent and therefore does not model the relationship captured by unsupervised and semi-supervised learning which is based on this relationship.

Supervised learning can be seen as the problem of determining $p(y'|\mathbf{x}', \mathbf{x}, y)$, that is, of determining the probability of a class in the test set given its features and the training data. If we accept the relationships among the random variables given in the graphical model of figure 1, then the rules of probability provide an optimal technique for classification. It can be shown that:

$$p(y'|\mathbf{x}', \mathbf{x}, y) = \int p(y'|\mathbf{x}', f) p(f|\mathbf{x}, y) df \, . \qquad (1)$$

Using Bayes law and the independence assumptions of the model it can also be shown that:

$$p(f|\mathbf{x}, y) = \frac{p(y|\mathbf{x}, f)p(f)}{\int p(y|\mathbf{x}, f)p(f)df} \, . \qquad (2)$$

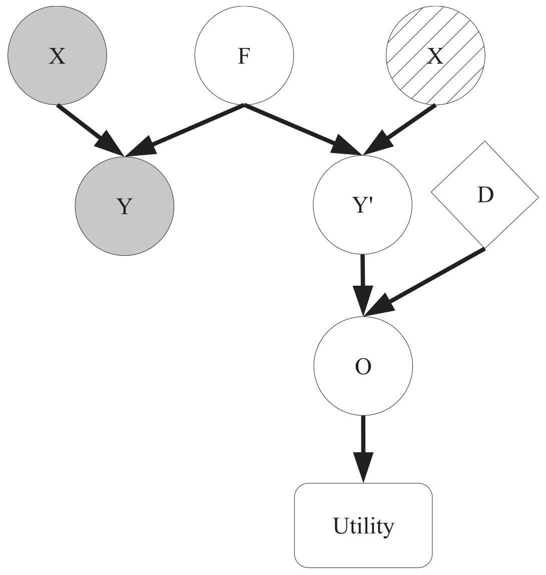

Fig. 1. Complete decision theoretic model for supervised learning.

Equations 1 and 2 form the backbone of our Bayesian model for supervised learning [4]. This formulation is surprisingly versatile. To create a learning system with a desired bias the user only has to specify $p(f)$, a prior distribution over the functions of interest, and the rules of probability dictate the rest. Specifying $p(f)$ usually consists of two parts, specifying the representation and the priors for the parameters of this representation. It is possible to select $p(f)$ so that we match the representational bias of a wide variety of classical learning techniques. and has been applied to several, including neural networks [5], support vector machines [6], and evolutionary computation [7][8]. In all of these cases this approach has lead to improved performance over many test cases.

It has been traditional to separate the ideas of classification from control/decision theory; however, in practice, classification is not performed for its own sake. Usually we do not classify things for the intrinsic value of putting things in classes, rather, we classify things to aid in decision making. As shown in figure 1, the entire decision process can be represented as a decision network tied to the graphical model. Y' is the node most commonly tied to decisions. In the simplest case, there is some outcome O that will depend upon the decision D and the value of Y' (see figure 1). Then the utility U is determined from the outcome O. Many more complex examples could be envisioned, for example, the output could depend on many more things that just Y', however, we will only focus on this case.

If the posterior distribution for $y'|\mathbf{x}', y, \mathbf{x}$ is computed as shown in equation 1, then it is possible to compute the expected utility of any decision d as:

$$E[U(d, O)] = \sum_{y' \in Y} U(o)p(o|y', d)p(y'|\mathbf{x}', \mathbf{x}, y)\,. \quad (3)$$

and the optimal decision $d \in D$ that will maximize the expected utility can be found as follows:

$$\underset{d \in D}{\operatorname{argmax}} \sum_{y' \in Y} U(o)p(o|y', d)p(y'|\mathbf{x}', \mathbf{x}, y)\,. \quad (4)$$

The degree to which the system is risk averse, risk seeking, or risk neutral is determined by the utility function chosen. Once a utility function is chosen, these equations allow us to quantify the risk of using the network's outputs for control. This is especially important for high assurance systems. Furthermore, this value will change as more data is acquired and the learner becomes more confident in its outputs. We call this graphical/decision model, and the learning algorithm based on it the Unified Bayesian Decision Theoretic Model or UBDTM [4].

Notice that in order to effectively use a classification technique to make general decisions that technique must provide a full distribution over $p(y'|\mathbf{x}', \mathbf{x}, y)$. Techniques that simply report the most probable class are not useful for making optimal decisions from a maximum expected utility perspective. Returning to our mushroom example, without a full distribution over y' it is impossible to quantify the risk of eating the mushroom. Since traditional ANN's only report what they consider to be the most likely class without reporting the entire distribution they are dangerous to use in high assurance systems. However, an ANN based on the UBDTM could report a full distribution, and could be more effective in high assurance systems.

We will propose an ANN based on UBDTM, but which has a very specific network topology, namely the CMAC ANN. This network is especially interesting because the values in the network can be solved in closed form for this network, and because its values weights are especially easy to interpret.

III. The Traditional CMAC

The CMAC is a well known ANN topology which has been shown to be useful for many applications [1]. It is modeled on the human cerebellum, and functions by mapping weights $w[i]$ to tiles which are interpreted spatially (see figure 2). Inputs are mapped to the correct bins by means of an association function $b[i](x)$, where $b[i](x) = 0$ when x does not fall within the spatial region assigned to bin i and where $b[i](x) = 1$ when it does.

The output of the system can be simply computed by

$$f_{CMAC}(x) = \sum_i w[i]b[i](x).$$

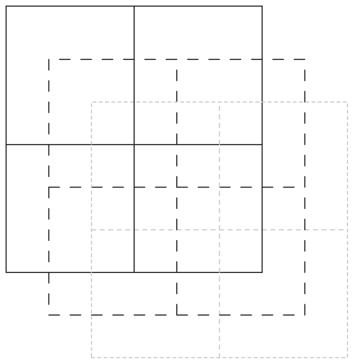

Fig. 2. Tile structure for a CMAC with three layers, and four tiles per layer.

The error at location x is:

$$e(x) = f_{CMAC}(x) - f_{observed}(x).$$

Traditionally the weights can then be intuitively updated as follows:

$$\Delta w[i] = \alpha \frac{e(x)}{\sum_i b[i](x)},$$

where α is a learning rate. The output y of the network at any position x is the sum of the weights for the tiles that overlap that position. It can be shown that this update rule iteratively approximates a maximum likelihood solution for the weights given the training data. Unfortunately, like any maximum likelihood technique, this approach does not produce a full distribution over outputs.

IV. A Bayesian Learning Procedure for the CMAC

We will create a Bayesian technique for computing the parameters of F where F is a CMAC. First, we assume that the function that we are trying to approximate is stationary, and that our observations $\mathbf{y}$ have linear Gaussian noise with covariance $\mathbf{\Sigma}_y$. For this work we will assume that all observations are of equal quality. We initialize $\mathbf{\Sigma}_y$ as $\mathbf{\Sigma}_y = \sigma_y^2\mathbf{I}$ where σ_y is a constant which models how much we should expect the output of each y to vary from the true value of the function (for example, this can account for sensor noise). We also assume that the training and test sets are identically distributed, and thus $P(y|x, f) = P(y'|x', f)$, a common assumption in Machine Learning.

Since a CMAC models functions using a sum of weights, it is convenient to model the prior for F as follows:

$$p(f) = \begin{cases} p(\mathbf{w}) & \text{if } f \in \{\lambda\mathbf{x}.g(\mathbf{x}, \mathbf{w}) | \forall \mathbf{x} \in \mathbb{R}^D.g(\mathbf{x}, \mathbf{w}) = \\ & \sum_{i \in \mathcal{T}} w[i]b[i](x)\} \\ 0 & \text{otherwise} \end{cases}$$

$$p(\mathbf{w}) = N(\mathbf{w}|\boldsymbol{\mu}_0, \mathbf{\Sigma}_0)$$

where $\mathcal{T}$ is a list of all tiles in the CMAC and where $\mathbf{w}$ is a vector-valued random variable with a multivariate normal distribution with prior means $\boldsymbol{\mu_0}$ (corresponding to the weights of each tile) and prior covariance $\boldsymbol{\Sigma}_0$.

The distribution of $\mathbf{X}$ does not need to be modeled since its values are given. Then, the relationship between $\mathbf{X}$, Y, and F can be modeled as follows:

$$p(\mathbf{y}|\mathbf{x}, f) = N(\mathbf{y}|f(\mathbf{x}), \boldsymbol{\Sigma}_y).$$

At this point it is helpful to note that $p(\mathbf{y}|\mathbf{x}, f)$, and similarly the predictive distribution $p(\mathbf{y}'|\mathbf{x}, f)$, can be rewritten as follows:

$$p(\mathbf{y}|\mathbf{x}, f) = N(\mathbf{y}|\mathbf{H}\mathbf{w}, \boldsymbol{\Sigma}_y),$$

where $\mathbf{H}$ can be thought of as an association matrix. $\mathbf{H}_{i,j} = 1$ if tile j influences the training example i. That is, $\mathbf{H}_{i,j} = b[j](x_i)$. Notice that $\mathbf{x}$ is used to construct $\mathbf{H}$, and then drops out of the equation. Arbitrarily complex kernels can be represented by simply modifying the $\mathbf{H}$ matrix such that $\mathbf{H_{i,j}} = k[j](x_i)$, where $k[j](x_i)$ is not binary but rather is a function of the distance from x_i to the center of tile j. Thus $\mathbf{H}$ encodes the amount of influence each tile has on a given x_i.

The weights are related to our observations according to a multivariate normal model [9] with prior parameters $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_0$. The parameters of the posterior distribution are then

$$\boldsymbol{\mu}_1 = \boldsymbol{\mu}_0 + \mathbf{K}_1(\mathbf{y} - \mathbf{H}\boldsymbol{\mu}_0),$$

and

$$\boldsymbol{\Sigma}_1 = (\mathbf{I} - \mathbf{K}_1\mathbf{H})(\boldsymbol{\Sigma}_0),$$

where

$$\mathbf{K}_1 = (\boldsymbol{\Sigma}_0)\mathbf{H}^T(\mathbf{H}(\boldsymbol{\Sigma}_0)\mathbf{H}^T + \boldsymbol{\Sigma}_y)^{-1}.$$

In the more complex case where the function can change over time in a linear manner with Gaussian noise these equations become identical to the Kalman Filter Equations.

This observation means that, given a prior over CMAC weights and some training data, a well known and widely studied filtering technique can be applied to solve in closed form for both the posterior distribution over the CMAC weights, and for the posterior predictive distribution for the CMAC outputs. We call this technique the Bayesian CMAC or BCMAC. Learning in theBCMAC is simply Bayesian inference.

Perhaps most importantly, this technique can simply produces an actual posterior distribution over possible outputs given the training data, $p(y'|\mathbf{x}', \mathbf{x}, y)$. Equation 1 tells us that this is a large and often difficult integral. However, because in this case we are dealing with the sum of a set of normal random variables this can be solved simply in closed form:

$$p(y'|\mathbf{x}', \mathbf{x}, y) = N(y'|\mathbf{H}\mathbf{w}, \mathbf{H}^t\boldsymbol{\Sigma}\mathbf{H}). \quad (5)$$

This makes it possible to know how much to trust the BCMAC's outputs in each part of the function. This can reveal when more data is needed, and where such data would be most useful. Such a confidence is also essential in order to use the outputs of the algorithm to make optimal decisions from a decision theoretic standpoint.

TABLE I

SUM^2 ERROR OF THE ACTUAL FUNCTION OUTPUTS AND THE TRADITIONAL CMAC OUTPUTS, OR THE MEAN BCMAC OUTPUTS OVER THE TEST SET. WE ALSO PRESENT THE RATIO BETWEEN THESE ERRORS

Run	Simple CMAC Sum^2 Error	BCMAC Sum^2 Error	CMAC/ BCMAC
1	1,857.654	1,459.136	1.27
2	1,917.688	1,412.675	1.36
3	1,323.653	1,141.215	1.16
4	1,593.757	1,326.135	1.20
5	1,728.339	1,401.232	1.23
6	1,660.846	1,327.345	1.25
7	1,621.135	1,292.013	1.25
8	1,472.336	1,273.971	1.16
9	1,271.723	1,121.361	1.13
...	...	...	...
100	1,950.859	1,534.583	1.27
Average	1,581.205	1,305.917	1.21

Priors can come from many places. If detailed information is known a-priori concerning the function, then a detailed prior can be provided, allowing the system to perform well with low amounts of data. This is possible because of the interpretability of weights in the CMAC topology. If less is known, then a wide subjective prior can be chosen. If the prior is sufficiently wide, it will have little effect on the posterior. This will cause the system to require more sample points before it becomes confident in its outputs. Although the data in a specific tile may be sparse, the cumulative data over the entire function will often be more rich. By analyzing the cumulative data, it is possible to use that information to produce an "empirical" prior for each tile which can often improve performance over a wide subjective prior. Theoretically a hierarchical model would be more statistically correct, but at the expense of a more complex model, which is unwarranted in this case.

V. RESULTS

To test the power of the BCMAC, we performed several experiments. First, we attempted to determine if the BCMAC outperformed the standard CMAC on the sorts of functions we might expect to encounter by comparing their results on several synthetic functions, and on a variant of the UCI Abalone data set. We also explored the ability of the algorithm to express confidences in its outputs.

A. BCMAC and CMAC Compared

First, we chose to illustrate the algorithm's behavior with a simple two dimensional function called step2d because the results could easily be visualized. The function was designed to be difficult for a CMAC representation. If $x1^2 + x2^2 < 10$ the function returns 1 otherwise it returns -1. This causes a steep transition, and a curved boundary to that transition, both of which are difficult features for a CMAC to capture. Points in the function fall in the domain between 0 and 5 for both x1 and x2. We created 100 different training sets by drawing 100 different sets of 500 examples each from the function, drawn uniformly. We then trained both a traditional CMAC and the BCMAC (with a wide subjective prior mean $\mu_o = 1$ and covariance $\Sigma_o = 20I$) on these training sets, and computed the difference between the mean of their outputs to a single test set of size 20,000. The results are summarized numerically in Table I. Although the error can vary substantially from one run to another depending on the quality of the small random training set chosen, the ratio of the standard CMAC error over the Bayesian CMAC error is consistent. This means that although the quality of the training set is an important factor in obtaining a good result on this function, the BCMAC always outperformed the standard CMAC no matter what training set it is given.

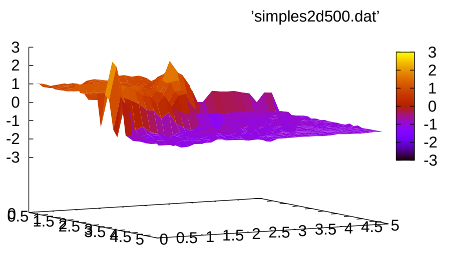

Fig. 3. CMAC output, viewed from the side to show the jagged overshoot artifacts.

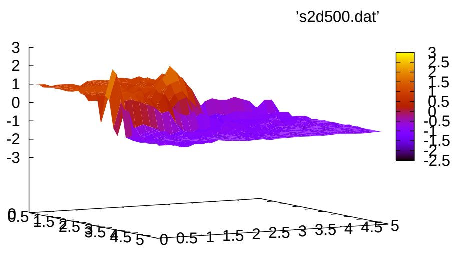

Fig. 4. Bayesian CMAC output, notice that there are less jagged overshoot artifacts.

This process was then repeated for a variant of the two dimensional egg carton function $y = sin(x1*2.5)+sin(x2*2.5)$, 2dEgg, and its three dimensional variant $y = sin(x1*2.5) + sin(x2*2.5) + sin(x3*2.5)$, 3dEgg. These results are summarized in Table II. In the interests of space, only the averages will be reported here, however, the full tables were much like those shown for step2d, where the individual values varied greatly depending on the random training set chosen, but where the ratio between the results of the various methods was consistent. Parameters for these functions were tuned using the "empirical" technique discussed above.

We also compared the CMAC with the BCMAC on the UCI Abalone data set [2]. This problem was chosen because of its continuous features and ordinal output which can be

TABLE II

AVERAGE SUM^2 ERRORS FOR THE SIMPLE CMAC AND THE BAYESIAN CMAC AVERAGED OVER 100 DIFFERENT TRAINING SETS ON SEVERAL DIFFERENT BENCHMARK FUNCTIONS.

Function	CMAC	BCMAC
step2d	1581.205	1305.917
2dEgg	510.409	499.885
3dEgg	483.814	481.560
Abalone	0.9063	0.7543

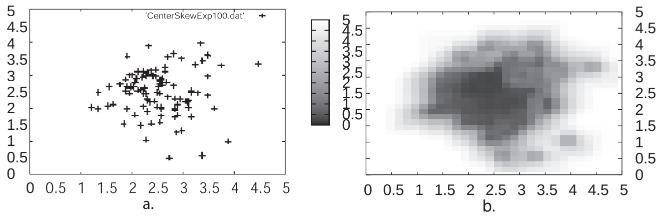

Fig. 5. a) Training points for the BCMAC, selected from a function exponentially biased towards the center. b) Tile variance after training with data points in a, darker areas represent regions of less variance. Note that the algorithm is more confident in areas where more data has been observed.

treated as continuous. All nominal features were dropped, as well as the many redundant features, leaving the most informative weight measurements (shell weight) and the most informative size measurements (diameter). The first 1500 examples were used as test data, while the remaining examples were used as training data (see Table II). The BCMAC priors were again set "empirically."

B. Expressing Confidence

We have seen that when the BCMAC reports its most likely output values it consistently has a lower mean squared error than the traditional CMAC on all the functions that we tested. However, the real power of this technique is not in its lower error alone. Because the training technique for the BCMAC is Bayesian in nature, a full distribution is available for the predictive posterior. If more data has been encountered in one region, then the algorithms will be more confident in their answers in that region. If less data has been encountered in another region they will be less confident in their answers in that region.

To illustrate, we again used the step2d function to demonstrate BCMAC's ability to return confidences in its results. We generated data that is exponentially skewed towards the center of the domain. After training we then graphed the algorithm's confidence (see figure 5). This ability to report confidence in a statistically meaningful way is an important advantage of our algorithms. Such a confidence is essential in decision theory in order to make optimal decisions with respect to maximum expected utility [9]. Because this value quantifies risk, it is an essential part of any high assurance system. Furthermore, the distributional output can be extremely useful in situations such as active learning [10].

VI. Discussion

Notice that the posterior predictive of the BCMAC (equation 5) is a normal distribution. Having a full distribution is extremely important for high assurance systems. Returning to our mushroom example, assume that mushrooms with toxicity levels below 10 are harmless, but with levels over 10 their negative effects begin to grow rapidly. Assume that we have been lost in the forest for 10 hours, and are beginning to be hungry. Now we come across a patch of mushrooms. Whether or not we should eat the mushrooms depends on our utility function. Because we are hungry there is a slight negative utility (-1) if we don't eat the mushroom. If we eat the mushroom, the utility will depend on the toxicity level of the mushroom as follows:

$$U(notEat, tox) = -1$$

$$U(eat, tox) = \begin{cases} 1 & \text{if } tox < 10 \\ -5 * tox + 51 & \text{if } tox > 10 \end{cases}$$

Now, if our mushroom toxicity predicting algorithm only returns the maximum likelihood result, as most neural networks do, and if it returns 10.1 as the most likely toxicity. If this were the true toxicity, the utility of eating these slightly toxic mushrooms would be .5. The utility of not eating the mushrooms is -1. At first glance it would seem that the best thing to do would be to eat the mushrooms. However, this all depends on how sure the algorithm is in its prediction. If the most likely toxicity is 10.1, then there is some chance that the actual toxicity is 15, or perhaps even 20, in which case eating the mushrooms would be very dangerous. The problem here is not that we are uncertain, but that we are unable to quantify our uncertainty and risk. It is impossible to effectively make this decision without more information.

If we used our BCMAC, the posterior predictive value would be a normal distribution. Suppose that our algorithm returned a mean of 10.1 with a variance of 3 for the toxicity of the mushroom. Now the expected utility of eating the mushroom can be computed by integrating as given in equation 3. The expected utility of eating the mushroom can now be computed and is -5.2, and the best action is to not eat the mushroom even though the utility at the maximum likelihood value would seem to indicate that we should eat the mushroom. If we had been lost longer, the utility of not eating the mushroom could drop considerably as the likelihood of starving increases. Once this value drops below -5.2, the optimal decision will be to eat the mushrooms despite the risk.

If we were designing the ANN for mushroom toxicity detection, we might have noted that for many common mushrooms the variance returned was simply too high in too many common cases. This could be cause by a naturally noisy function, but if the amount of intrinsic noise in the function is such that the designer believes that the ANN should be able to do better, then this would indicate that either more data is needed, or else the resolution of the BCMAC needs to be increased. Furthermore, if there is more variance in one part of the output than in another, then this could indicate not only that more data is needed, but where data would be most useful. This can be helpful in active learning situations. Thus, because the network reports its confidence, the risk of using the system can be evaluated beforehand and improvements made when appropriate.

Another way of lowering the variance of the predicted outputs would be to provide the algorithm with expert human knowledge. If a human expert knows that dark mushrooms of a certain size are always very poisonous, while light mushrooms of the same size are never poisonous, then that information can easily be introduced into the priors for the weights. Because the weights are spatially interpretable, it is clear which weight priors need to be adjusted to achieve the desired effect. Furthermore, the strength of the prior is determined by the variance on the weights, allowing our experts to express how confident they are in their information.

This ability is illustrated by the above step2d experiments. In those experiments we set the mean to 1 and the variance to 20 because we did not want to give the BCMAC an unfair advantage. However, since we knew beforehand that the values would vary between -1 and 1 we could have selected a more informed prior with a mean of 0 and a standard deviation of 1. It would have been possible to do even better by setting the prior mean of all tiles with centers $x1^2 + x2^2 < 10$ to 1 with all others set to -1, and then lowering the variance accordingly, in which case little or no learning would have been necessary because of the accuracy of the prior.

VII. Conclusions

By modeling the regression and classification process as a graphical model we have produced a Bayesian variant of the CMAC that can be solved in closed form. We call this technique the BCMAC. We have shown that these techniques outperform the traditional training techniques for learning the weights of a CMAC for several example functions including step2d, 2dEgg and 3dEgg, as well as for the Abalone dataset. Furthermore, the BCMAC not only can determine the most likely weights, but it can also give a statistically correct estimate of how certain it is of its values at every position, which is essential for making good decisions from a decision theoretic perspective.

We have shown that the BCMAC has several advantages in high assurance systems. Its weights are interpretable, the priors can be used to introduce expert knowledge, and the distributional output can be used to quantify risk, has enough data, and determine if the algorithm is sufficiently accurate to be used in a given context. This can greatly improve the applicability of ANNs to high assurance situations.

References

[1] J. S. Albus, "A new approach to manipulator control: The cerebellar model articulation controller (CMAC)," *Journal of Dynamic Systems, Measurement, and Control*, vol. 97, no. 3, pp. 220–227, 1975.

[2] D. J. Newman, S. Hettich, C. Blake, and C. Merz, “UCI repository of machine learning databases,” 1998. [Online]. Available: http://www.ics.uci.edu/∼mlearn/MLRepository.html

[3] F. V. Jensen, *Bayesian Networks and Decision Graphs*. New York: Springer-Verlag, 2001.

[4] J. L. Carroll and K. Seppi, “No free-lunch and bayesian optimality,” *IJCNN Workshop on Meta-Learning*, 2007.

[5] J. de Freitas, M. Niranjan, A. Gee, and A. Doucet, “Sequential monte carlo methods for optimisation of neural network models,” 1998. [Online]. Available: citeseer.ist.psu.edu/freitas98sequential.html

[6] C. M. Bishop and M. E. Tippling, “Bayesian regression and classification,” *Advances in Learning Theory: Methods, Models and Applications*, vol. 190, pp. 267–285, 2003.

[7] C. K. Monson, “No free lunch, bayesian inference, and utility: A decision-theoretic approach to optimization,” Ph.D. dissertation, Brigham Young University, Department of Computer Science, 2006.

[8] C. K. Monson, K. D. Seppi, and J. L. Carroll, “A utile function optimizer,” in *The Proceedings of the IEEE Congress on Evolutionary Computation (CEC) (accepted)*. IEEE Press, Sept 2007.

[9] M. H. DeGroot, *Optimal Statistical Decisions*. New York, NY: McGraw-Hill Book Company, 1970.

[10] H. S. Seung, M. Opper, and H. Sompolinsky, “Query by committee,” *proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pp. 287–294, 1992.