More Mathematics on Adaptive Tiled Neural Networks

M.Nokhbeh-Zaeem[†], *D.Khashabi, H.A.Talebi*
EE Department, Amirkabir University of Technology

Abstract

The paper includes some more details of ATNN which is recently introduced in [1]. This paper, gives more details on some obscure points and claims of it. As it is previously introduced, this approach is based on a combination of conventional Neural Networks and tile coding approximators. The proposed approach can maintain the desired features of both approaches while eliminates the deficiencies of each method.

1 Introduction

Training very expressive Neural Networks (NNs) have always had a high computational cost and memory usage. As it is generally known, the generalization property also decreases with the increase of the network size [2, 3, 4]. From another point of view, small size networks have problems in representing a wide range of functions. Several algorithms have been proposed to make the structure of NNs and tile coding discretization adaptive [5, 6, 7, 8], so that the structure can change whenever the network is not expressive enough. Another deficiency of conventional NNs is that the approximation of one point cannot change without changing the whole function, unless all training examples are used in the training. However, in sequel the training with all training examples, increases the computational cost and causes other problems such as memory shortage, etc. On the other hand, methods with lower cost that can change the accuracy/complexity of approximation on subspaces, such as tile coding [9, 10, 11] (which discretizes the state space and approximates the function piecewise on the discretizations) and hyperplane tile coding [11], cannot express functions very smoothly. Another significant drawback of conventional NNs is that the superviser cannot allocate a desired accuracy/complexity to the desired subspaces. The accuracy/complexity of the approximation in conventional NNs is the same over the whole subspace.
In this paper, we will give more details on performance and analysis of ATNN,

This document is a companion to the paper: *Nokhbeh-Zaeem et al, "Adaptive Tiled Neural Networks", IEEE SMC 2011.* This document is published for free. Copyright is reserved for the authors.

[†] Correspondent author: nokhbeh100@gmail.com

which is recently proposed in [1]. As it is comprehensively clarified, the new combination is proposed so that the approximator's structure can change during the training without the use of huge NNs leads to larger memory usage during the training. In addition, the approximator can change its approximation of a subspace without changing the approximation of the whole space, this property is called approximation memory. The is specially of high importance in applications that need local adjustment, e.g. Reinforcement Learning [12]. It can learn subspaces with different accuracies which is also a feature of adaptive tile coding. In addition, it never ends up with training or simulation of big NNs.
The rest of this paper is organized as follows. In Section 2 tile coding and some related methods are discussed. Then in Section 3 the proposed combination is introduced and some features are discussed. Finally in Section 4 the efficiency of the proposed approach is demonstrated by means of several simulation studies.

2 Backgrounds on continuous approximators

There are several methods used in function approximation; however, most of them have one of the two main framework ideas. First, considering a fixed structure and changing the coefficients, so that it converges to the desired function. Second, using a dynamic structure and changing the structure as well as the coefficients. NN-related issues and methods are considered here.

2.1 Fixed structure Neural Networks

NN approximation have been one of the most popular approximation methods in the recent years. Usually, the structure of the network is fixed and the represented function is fitted to the training examples with gradient descend method and Backpropagation [13].

2.1.1 Computational requirement of the Backpropagation method

Here the computational requirement of training a fixed structure NN using the Backpropagation algorithm is considered. The computational cost calculated here is used to compare computational cost of NN and ATNN methods. Let us assume that the network is a three-layered network, as it is proven that can express any arbitrary function with any arbitrary error having enough hidden units [14, 15]. Assume that the network has i inputs, n hidden units and o outputs. In such a structure there are ni input weights between the input and the hidden units and no weights between the hidden layer and the output units. According to the Backpropagation method [13], for each epoch, first each sample is fed to the network, then the output errors are calculated and backpropagated using the delta rule and weights are tuned. For each hidden unit there will be i products and i sums (including bias) and activation function is calculated, so there will be $c_pi + c_si + c_{ac}$ computation for each hidden unit, in which c_p, c_s, c_{ac} are respectively computational cost of product, sum and calculation of activation function. For each output there are n products and n sums, so there will be $c_p n + c_s n + c_{ac}$ computation for each output unit. Then errors in the output layer are calculated by applying the following error.

$$\delta_j = (t_j - o_j) \times o_j \times (1 - o_j) \tag{1}$$

For other layers but outputs the following error is used to propagate error.

$$\delta_h = o_h \times (1 - o_h) \times \sum_{k \in outputs} (w_{kh} \delta_k) \tag{2}$$

Where w_{kh} represents the weight from node h in the hidden layer to node k in the output layer. Weights are updated by the following equation.

$$w_{ij} = w_{ij} + \eta \delta_j x_{ji} \tag{3}$$

Where η represents the learning rate and x_{ji} is the ith input to the jth unit. The error term has $2c_p + 2c_s$ computation for each output. There are o weights from a hidden unit to the outputs. So we have $o + 2$ products and $(o - 1) + 1$ sums for each hidden unit. So there will be $c_p o + c_s(o+2)$ computation for each hidden unit. and we have a $c_p + 2c_s$ computation for updating of each weight. Total computation is $(n(c_p i + c_s i + c_{ac}) + o(c_p n + c_s n + c_{ac}) + n(c_p o + c_s(o+2)) + o(2c_p + 2c_s)) + n(i+o)(c_p + 2c_s))$. If $k_1 = 2c_p + 3c_s$, $k_2 = 3c_p + 4c_s$, $k_3 = c_{ac} + 2c_s$ and $k_4 = c_{ac} + 2c_p + 2c_s$, we have $n(k_1 i + k_2 o + k_3) + k_4 o$ computation per epoch per sample.
Almost all training methods based on the gradient-descend method are based on Backpropagation and hence their computational costs are similar.

2.2 Dynamic structure Neural Networks

Dynamic structure NNs are mainly divided into two main categories, constructive and pruning structures [16]. In the first category, a simple network is trained to learn the target function and when it fails to learn, the structure of the network is changed, usually by adding some hidden units; examples of such methods are Cascade-Correlation [6] and [7, 8, 17]. In the second category, a very expressive network is trained and then the network is pruned, some methods are expressed in [5, 8, 18, 19, 20].

2.2.1 Constructive networks

Most of the times when the performance of the network cannot reach the desired value, the structure of the network is changed, so that the network can express the target function. This change in structure will lead to an increase in the computational requirement. Usually this change of structure is done by adding some extra hidden units, therefore the network becomes more expressive. The computational requirement and other related topics are discussed in [21].
Since the Cascade-Correlation method [6] have some similarities with the proposed ATNN method, we discuss it further.

2.2.2 Pruned networks

As the name suggests, it generally starts with a bigger NN and prone it later [17, 21]. It is computationally more expensive than the constructive methods. Some of these methods are discussed in [19, 20, 22, 23, 24].

2.3 Tile coding

Tile coding [12] and related methods [25, 26, 27] for approximating functions are usually used for applications which need to retrieve information very fast and very often and do not need the most possible accuracy such as Reinforcement Learning [12]. In the conventional tile coding, the input space is discretized into several tiles and the target function is approximated piecewise on these tiles.

2.4 Adaptive tile coding

As considered in [9, 10, 28], it is often hard to decide how many tiles should the input space be split into. On the other hand, a constant tiling rate is not always preferred, and different tiling rates might be useful. Details about how and when the tiling rate is changed are given in [9]. Note that here, another condition is used to decide when and where to change the tiling rate.

2.5 CMAC

CMACs [29, 30] are special types of NNs whose output is sum of the weights of the tiles of the input. CMACs have the same problem with the sharpness of the edges which is discussed and some solutions are proposed in [31, 32, 33, 34]. However, similar to tile coding, CMACs are not expressive enough for some complex regression problems.

3 Adaptive tiled Neural Networks

As it is presented in [1], the pseudo-code of ATNN is as shown in Algorithm 1. After the training procedure, because of tiled structure of ATNN, we need to take care about the way that we need to retrieve the information from the trained network. Algorithm 2 shows how samples' values can be retrieved. For retrieving a set of samples' values, all tiles must be examined to find the relative membership of samples to them. Then the weighted average of all the results are calculated considering the weight of corresponding memberships. The details of this process are shown in Algorithm 2.

3.1 Computational requirement of ATNN

As we mentioned, the computational requirement of training networks with more than three layered is very high. On the other hand, two-layered NNs are not expressive enough, but in comparison a three-layered networks have enough

Algorithm 1 Pseudo-code of ATNN
T: Tiles of ATNN
S: Input samples
N: Base network

```
if $T.Bound$ does not exist then
    Initialize the tiles bounds $T.Bound$ so that it include all samples.
end if
for every tile $T$ that ($T.MSEerror > Threshold$) do
    $I$ = Find all corresponding samples
    if $N$ does not exist then
        $N$ = Initialize network
    end if
    $N$ = train the network with $I$
    if $N.MSEerror < Threshold$ then
        $T.MSEerror = N.MSEerror$
        Save the network $N$ for tile $T$
    else
        $T.Bound$ = split($T.Bound$)
    end if
end for
```

capacity to express any function with enough hidden units[14, 15]. Hence here only three-layered NNs are considered.
Let us assume function approximation problem for the case that we have o outputs, i inputs, d training samples and m input dimensions.
Here we calculate the computational requirement of finding the relative membership of samples to tiles. Assuming that there are d training samples, a logical comparison of the bounds of tiles with the sample inputs must be done. Hence there will be $2mdc_c$ computations per tile where c_c is the computational requirement of a single comparison. There is also an AND operator between the upper and the lower bounds' conditions results, so there are dm ANDs and dm ANDs on the different dimensions conditions and finally there will be d conditions for determining that each sample belongs to the kth tile or not. Therefore the computational cost is $(2mdc_c + dmc_a + mdc_a + dc_c)$ for determination of the samples membership to a tile where c_a is the computational cost for a single AND. However, in practice, because the process of training is usually repeated for many epochs, the calculated computation can be neglected as compared to the training computation.
Let us assume that the target function can be learned with an NN with at least n_t hidden units (three-layered) with some arbitrary error. The Computational requirement, as proven in 2.1.1, is $(d(n_t(k_1 i + k_2 o + k_3) + k_4 o))$ per epoch. Whereas in ATNN the algorithm tries to learn the target function by NNs with n_0 hidden units so that the computational requirement of training an NN tile will be $(d_i(n_0(k_1 i + k_2 o + k_3) + k_4 o))$ per epoch, in which d_i is the number of training example in a specific tile. Considering that $n_0 \gg o, i > 0$ and we can

neglect k_4. Hence the total computational requirement of training a set of tiles covering the whole instance space will be $(d(n_0(k_1 i + k_2 o + k_3) + k_4 o))$, since $\sum_{i=1}^{n} d_i = d$.
Assuming that the number of epochs in conventional NNs is the same as epochs of training of each tile, it can be concluded that the computation of training conventional NN structure is more than training a full covering set of tiles $\lfloor \frac{n_t}{n_0} \rfloor$ times. It is obvious that if one of the tiles become accurate enough, the computational cost will be less than what is calculated.
Assuming that after training each full covering set of tiles the tiles are split into two sub-tiles, after $\lfloor \frac{n_t}{n_0} \rfloor$ trainings, there will be $2^{\lfloor \frac{n_t}{n_0} \rfloor}$ tiles having $n_0 2^{\lfloor \frac{n_t}{n_0} \rfloor}$ localized hidden units. Hence, the computational requirement of a conventional NN is equal to ATNN with $2^{\lfloor \frac{n_t}{n_0} \rfloor}$ tiles.

Algorithm 2 Pseudo-code of retrieve ATNN

```
for every sample $S$ do
    $\vec{P}$ = Find all corresponding predictions of sample $S$
    $\vec{W}$ = Find the relative membership weight of sample $S$ to $T$
    $O(S) = \vec{W}.\vec{P}/\vec{W}.\vec{1}$‡
end for
```

3.2 Capacity

Although it was shown that the ATNN with $n_0 2^{n_t/n_0}$ localized hidden nodes have the same computational cost as an NN with n_0 hidden units, there are other ways that may increase the degree of freedom (DoF) of ATNN system. For instance, if for solving the discontinuity problem, the first solution is chosen, the system will have an additional smooth DoF between tiles. Meaning that different biases of networks can make smooth step changes along the axis. For example when the tile is a hyperplane expressed by a linear combination of the inputs, the tile edges have an average of both tiles coefficients.
Most of NN structures use units that have a linear combination and an activation function; Every linear combination is equivalent to a weighted distance of the point and a hyperplane. If the tiles are uniformly distributed, as we discussed, every hyperplane crosses at most $k^m - (k-1)^m$ tiles, in which m is the dimension of the approximation problem and k is the number of tiles spread in each dimension.
In this case, we have $k = \log_m 2^{\lfloor \frac{n_t}{n_0} \rfloor} = \frac{\lfloor \frac{n_t}{n_0} \rfloor}{\log_2 m}$ (for the NNs with the same computational requirement as the ATNN). Knowing that $x - 1 \le \lfloor x \rfloor \le x$, we can say that the capacity of a conventional NN is at most equal to an ATNN with $\frac{n_t}{n_0 \log_2 m}^m - (\frac{n_t}{n_0 \log_2 m} - 1)^m$ localized hidden units (note that these localized hidden units are not independent). Comparing $n_0 2^{\frac{n_t}{n_0}}$ with mentioned capacity, it is obvious that the capacity growth of the ATNN is exponential, with respect

‡ $\vec{1}$ is a vector whose elements are all 1.

to n_t, but capacity of NN grows polynomially. On the other hand, the capacity of NN falls with the growth of m for reasonable values of $\frac{n_t}{n_0}$ but the ATNN's capacity is independent of m. By comparing localized hidden units of the two methods mentioned above, it is obvious that with the same computational requirement, the ATNN is more expressive in high dimensional problems[1].

4 Practical results

In this section we are going to implement ATNN and analyse the results.

4.1 Implementations of ATNN

Here we aim to make comparisons between results derived from ATNN and NN. In these experiments we use MLP (Multi Layer Perceptron) in our both ATNN and NN structures. The implementations are trained with fixed *sharpness* parameter and Joining of tiles is not considered. All figures shown in this section include MSE (mean square error) of ATNN on training samples, number of hidden units of the base network (df), *threshold* (the parameter defined in Algorithm 1), number and region of samples.

4.1.1 The sine function

The result of approximating a sine function by ATNN is shown in Figure 1. The green bar bellow the figure shows rates of tiling along x-axis. Since the sine function can be approximated accurate enough by two NNs consisting of 2 hidden units each defined in $(-3, 0)$ and $(0, 3)$, so ATNN approximates it with only two tiles. It is important to remind that based on Algorithm 1 the splitting process is done by algorithm without involvement of supervisor.

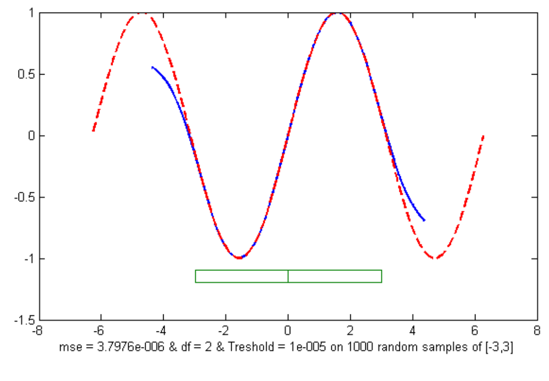


Fig. 1: The ATNN approximation of $sin(x)$ function in $(-3, 3)$ by base network consisting 2 hidden units. The green bar bellow the figure shows rates of tiling along x-axis.

4.1.2 The $sinc(\frac{1}{x})$ function

The approximation of $sinc(\frac{1}{x})$ function is shown in Figure 2. This figure shows how ATNN can fit the tiling rate to the target function. As it is shown, the tiling rate is proportionately much higher near 0 than near 2. The approximation shows how ATNN adaptively splits the region in order to make a proper approximation of a function. In other words, the more variations in a function, the more tiles we have.

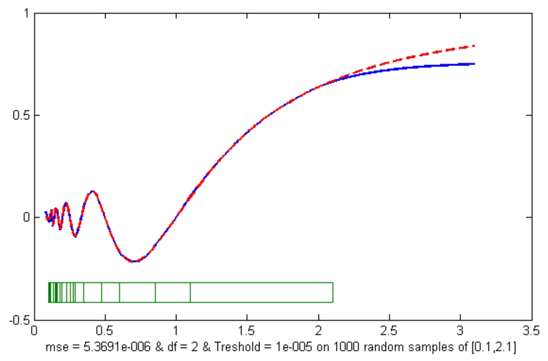


Fig. 2: Approximation of $sinc(\frac{1}{x}) = \frac{xsin(\pi\frac{1}{x})}{\pi}$ function in $(0.1, 2.1)$ by base network consisting 2 hidden units.

4.1.3 The $sin(\frac{1}{x})$ function

The approximation of $sin(\frac{1}{x})$ function is shown in 3. Like the previous experiment, the rate of tiles is proportionate to variations of function.

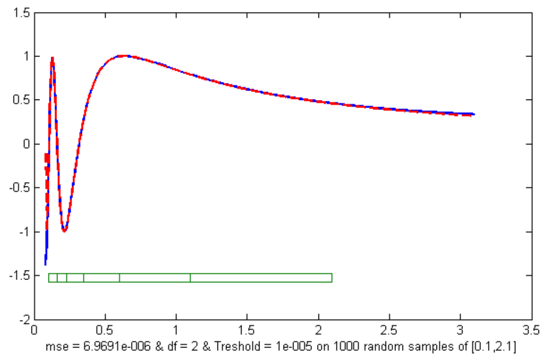


Fig. 3: Approximation of $sin(\frac{1}{x})$ function in $(0.1, 2.1)$ by base network consisting 2 hidden units.

4.1.4 The limited cosine function

The approximation of a limited cosine function, $cos(x)(u(x+\frac{\pi}{2})-u(x-\frac{\pi}{2}))$ is shown in Figure 4, where $u(x)$ stands for step function.

$$u(x) = \begin{cases} 0 & \text{if } x < 0 \\ \frac{1}{2} & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$$

The comparisons in the next section between NN and ATNN are based on this function.

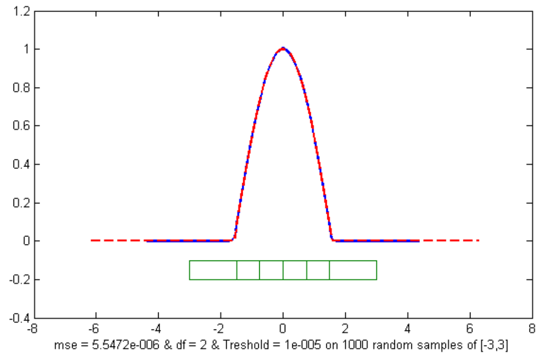


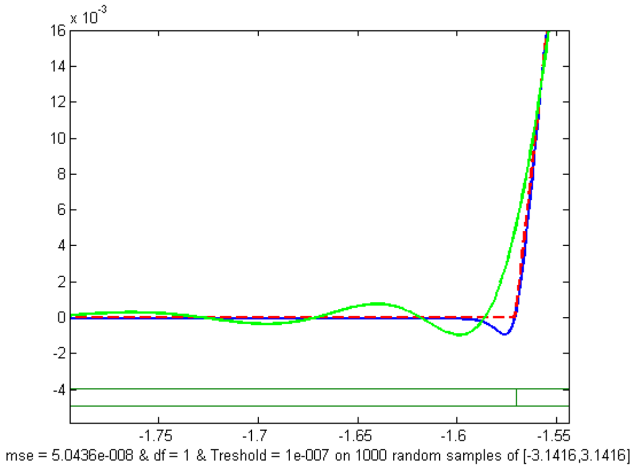
Fig. 4: The ATNN approximation of $cos(x)(u(x+\frac{\pi}{2})-u(x-\frac{\pi}{2}))$ function in $(-3,3)$ by base network consisting 2 hidden units.

4.2 More discussion on practical results

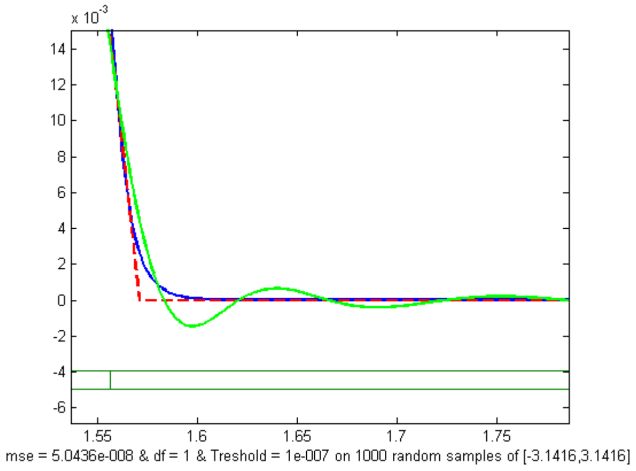
ATNN might have difficulties in approximating the function when its *sharpness* parameter is not tuned well. The following comparisons show that both NN and ATNN can achieve enough accuracy with acceptable computational effort. The Figure 5 shows the difference between ATNN and NN in sharp points. NN's approximation is usually smoother than that of ATNN, so in sharp points ATNN gives a better approximation. The Figure 6a shows how ATNN can be misled to a sharp point when there is a smooth extremum on the tile boundary[1].
As depicted in Figure 6b, ATNN's approximation might have insignificant error whereas NN's approach has overfitting problem. Regarding the figure, NN's output is only accurate in training points. So NN's approximation is usually overfitted to samples but ATNN have a generalization which is determined by *sharpness* parameter.

5 Future work

The *sharpness* parameter, as introduced have significant effect in approximation of keen of edges. One improvement for ATNN can be about discussing

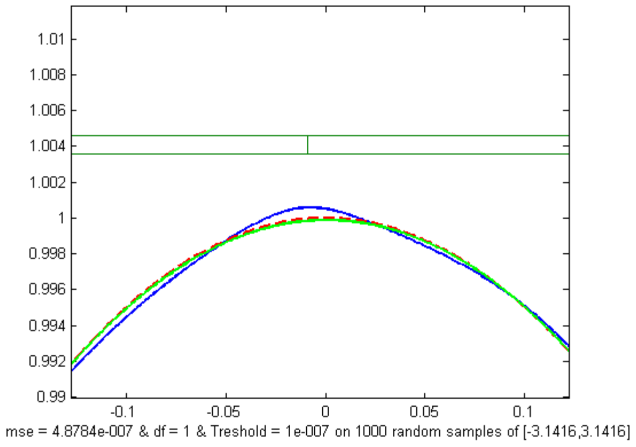


(a)

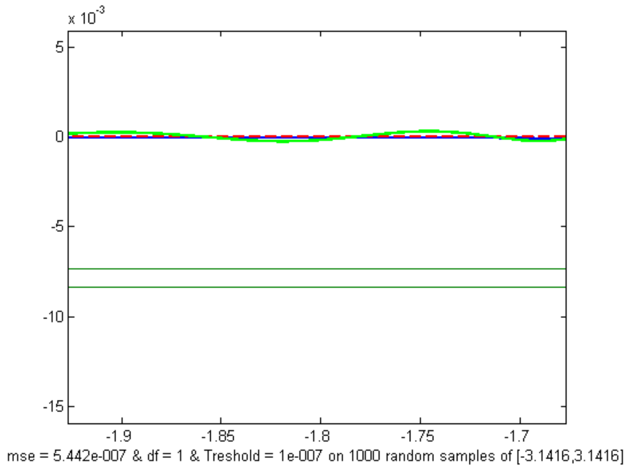


(b)

Fig. 5: The figure shows comparison between ATNN approximation (blue) and NN's approximation (green)



(a)



(b)

Fig. 6: The figure shows some advantages and disadvantages of ATNN in comparison to NN.

on methods for fine-tuning *sharpness* parameter, in order to find favourable approximation for wide range of functions and without direct interference of supervisor. Furthermore, in this paper our comparisons were completely based on MLP; but as we mentioned, we can used any function approximator in the ATNN structure. Even any other approximator can show different behaviours. However, here we showed applicability of ATNN by several functions, it is of high value to show its applicability in a more realistic application, e.g applications in Control Design, Reinforcement Learning, etc.

References

[1] M. Nokhbeh-Zaeem, D. Khashabi, H. Talebi, S. Navabi, and F. Jabbarvaziri, "Adaptive tiled neural networks," in *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on.* IEEE, 2011, pp. 2543–2548.

[2] E. B. Baum and D. Haussler, "What size net gives valid generalization?" *Neural Comput.*, vol. 1, no. 1, pp. 151–160, 1989. [Online]. Available: http://dx.doi.org/10.1162/neco.1989.1.1.151

[3] Y. Chauvin, "Generalization performance of overtrained back-propagation networks," in *Proceedings of the EURASIP Workshop 1990 on Neural Networks.* London, UK: Springer-Verlag, 1990, pp. 46–55. [Online]. Available: http://portal.acm.org/citation.cfm?id=646655.700117

[4] J. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, L. Jackel, and J. Hopfield, "Large automatic learning, rule extraction, and generalization," *Complex Systems*, vol. 1, pp. 877–922, 1987.

[5] G. Castellano, A. M. Fanelli, and M. Pelillo, "An iterative pruning algorithm for feedforward neural networks," *IEEE Trans. Neural. Networks*, pp. 519–531, 1997.

[6] S. E. Fahlman and C. Lebiere, *The Cascade-Correlation Learning Architecture*, 1991.

[7] S. Gallant, "Optimal linear discriminants," in *Eighth International Conference on Pattern Recognition.* Paris 1986: IEEE, New York, 1986, pp. 849–852.

[8] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*, ser. A Lecture Notes Volume in the Santa Fe Institute Studies in the Science of Complexity. Redwood City: Addison-Wesley, 1991.

[9] S. Whiteson, M. E. Taylor, and P. Stone, "Adaptive tile coding for value function approximation," University of Texas at Austin, Tech. Rep. AI-TR-07-339, 2007.

[10] A. A. Sherstov and P. Stone, “Function approximation via tile coding: Automating parameter choice,” in *SARA 2005*, ser. Lecture Notes in Artificial Intelligence, J.-D. Zucker and I. Saitta, Eds. Berlin: Springer Verlag, 2005, vol. 3607, pp. 194–205.

[11] D. Loiacono and P. L. Lanzi, “Recent advances in reinforcement learning,” S. Girgin, M. Loth, R. Munos, P. Preux, and D. Ryabko, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, ch. Tile Coding Based on Hyperplane Tiles, pp. 179–190.

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press, 1998.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D. E. Rumelhart and J. L. Mcclelland, Eds. Cambridge, MA: The MIT Press, 1986, vol. 1: Foundations, pp. 318–362.

[14] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, pp. 303–314–314, December 1989. [Online]. Available: http://dx.doi.org/10.1007/BF02551274

[15] K. Hornik, M. B. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators.” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[16] S. Waugh, “Dynamic learning algorithms,” 1994.

[17] T. Ash, “Dynamic node creation in backpropagation networks,” *Neural Networks*, vol. 2, p. 623, 1989.

[18] R. Reed, “Pruning algorithms: A review,” *IEEE Trans. Neural. Networks*, vol. 4, pp. 740–747, 1993.

[19] Y. L. Cun, J. S. Denker, and S. A. Solla, “Optimal brain damage,” in *Advances in Neural Information Processing Systems.* Morgan Kaufmann, 1990, pp. 598–605.

[20] B. Hassibi, D. G. Stork, and S. C. R. Com, “Second order derivatives for network pruning: Optimal brain surgeon,” in *Advances in Neural Information Processing Systems 5.* Morgan Kaufmann, 1993, pp. 164–171.

[21] T. yau Kwok and D. yan Yeung, “Constructive algorithms for structure learning in feedforward neural networks for regression problems,” *IEEE Trans. Neural Networks*, vol. 8, pp. 630–645, 1997.

[22] E. Drougge and J. Wroldsen, “A robust algorithm for pruning neural networks,” 1995.

[23] L. Prechelt, "Adaptive parameter pruning in neural networks," 1995.

[24] E. Cantu-Paz, "Pruning neural networks with distribution estimation algorithms," in *GECCO 2003, LNCS 2723, Springer-Verlag (2003) 790800.* Springer-Verlag, 2003, pp. 790–800.

[25] P. Lanzi, D. Loiacono, S. Wilson, and D. Goldberg, "Classifier prediction based on tile coding," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation.* ACM, 2006, pp. 1497–1504.

[26] L. Tokarchuk and Q. Mary, "Fuzzy and tile coding approximation techniques for coevolution in reinforcement learning," 2005.

[27] C. Wu and W. Meleis, "Function Approximation Using Tile and Kanerva Coding For Multi-Agent Systems," 2009.

[28] S. Lin and R. Wright, "Evolutionary Tile Coding: An Automated State Abstraction Algorithm for Reinforcement Learning," 2010.

[29] J. Albus, "Data storage in the cerebellar model articulation controller (CMAC)," *Journal of Dynamic Systems, Measurement and Control*, vol. 97, no. 3, pp. 228–233, 1975.

[30] ——, "A new approach to manipulator control: The cerebellar model articulation controller (CMAC)," *Journal of Dynamic Systems, Measurement, and Control*, vol. 97, p. 220, 1975.

[31] C. hung Lee and B. hang Wang, "Adaptive wavelet-based-cmac network predictor design for lossless image coding."

[32] S. Sayil, "A hybrid maximum error algorithm with neighborhood training for cmac," Citeseer, pp. 165–170, 2002.

[33] M. Eldracher, A. Staller, and R. Pompl, "Function approximation with continuous-valued activation functions in cmac," in CMAC, Tech. Rep., 1994.

[34] D. Cornforth and D. Newth, "The kernel addition training algorithm: Faster training for cmac based neural networks," in *Proc. Conf. Artificial Neural Networks and Expert Systems*, 2001.