Adaptive Tiled Neural Networks

M.Nokhbeh-Zaeem†, D.Khashabi†, H.A.Talebi‡, Sh.Navabi†, F.Jabbarvaziri†
Department of Electrical Engineering, Amirkabir University of Technology, Tehran, Iran
†{nokhbeh100,d.khashabi,shiva.navabi,faramarz.vaziri87}@gmail.com, ‡alit@aut.ac.ir

***Abstract*—In this paper, a novel function approximation approach based on a combination of conventional Neural Networks and tile coding approximators is proposed. The proposed approach can maintain the desired features of both approaches whiles eliminates the deficiencies of each method. The combination will reduce the sharpness of tile coding. It will also provide an easy way to adjust the accuracy/complexity of the approximation according to the function being approximated (adaptive tiling) and the subspace used on. In this algorithm, it is possible to construct the approximator with specified and various approximation accuracies in different subspaces. This feature enables us to allocate an arbitrary accuracy/complexity wherever a more accurate approximation is needed. Finally simulation studies are presented to show the efficiency of and applicability of the proposed approach.**

***Index Terms*—adaptive tiling, adaptive tile coding, tile coding, neural network, approximation memory**

I. Introduction

In function approximation tasks, training very expressive Neural Networks (NNs) have always had a high computational cost and memory usage. Moreover, the generalization property also decreases with the increase of the network size [1], [2], [3]. On the other hand, 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 [4], [5] and reference therein, 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 [6], [7], [8] and hyperplane tile coding [8], cannot express functions very smoothly.

In this paper, a new combination of NNs and adaptive tile coding is proposed. 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. One feature of the presented approach is that the approximator can change its approximation of a subspace without changing the approximation of the whole space (approximation memory). The proposed method can also learn subspaces with different accuracies which is also a feature of adaptive tile coding. In addition, simulation results show that tiling of an NN can act effectively in extrapolation if the target function can be locally approximated with considered base network (network of each tile). Simulation results show that the proposed combination has better generalization capability than the conventional neural networks over the training examples.

The rest of paper is organized as follows. In Section II tile coding and some related methods are discussed. Then in Section III the proposed combination is introduced and some features are discussed. Section IV gives practical aspects of the proposed Adaptive Tiles Neural Networks(ATNN). Finally in Sections V the efficiency of the proposed approach is demonstrated by means of several simulation studies.

II. 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.

A. Fixed structure and dynamic 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 [9]. Dynamic structure NNs are mainly divided into two main categories, constructive and pruning structures [10]. 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 [5] and [11]. Related topics are discussed in [12]. Since the Cascade-Correlation method [5] have some similarities with the proposed ATNN method, we discuss it further. In the second category, a very expressive network is trained and then the network is pruned, some methods are expressed in [4], [13] and reference therein.

B. Tile coding and adaptive tile coding

Tile coding [14] and related methods [15], [16] 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 [14]. In the conventional tile coding, the input space is discretized into several tiles and the target function is approximated piecewise on these tiles. As considered in [6], [7], 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 [6]. Note that here, another condition is used to decide when and where to change the tiling rate.

III. Adaptive tiled Neural Networks

The idea of tile coding is fast enough for some functions approximation tasks. However when the target function is varying very fast, the approximator's generalization and convergence can be less accurate and slower. On the other hand, training a full expressive NN is expensive. Hence a combination of both methods may be helpful. Here, the combination of the two methods are discussed. The pseudo-code of ATNN is shown in Algorithm 1. First, approximation region indicated by some boundaries are changed to include all samples. Considering that there might be some tiles saved in the structure, relative memberships of training samples to these tiles are determined.Then, each network is trained with the corresponding training examples. If the performance of all networks reaches a pre-defined threshold, then the algorithm will terminate; otherwise the corresponding tiles will split up and retrain until all networks have the desired performance.

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
```

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. As 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.

Algorithm 2 Pseudo-code of retrieve ATNN

```
for every sample S do
    P⃗ = Find all corresponding predictions of sample S
    W⃗ = Find the relative membership weight of sample S to T
    O(S) = W⃗.P⃗ / W⃗.1⃗ [1]
end for
```

A. Validation discussions on ATNN

Since NNs are universal approximators, proving that an ATNN retrieved values are universal approximations of the corresponding samples is just a manner of justification. As it is proven in [17], [18], and knowing that each tile training is only influenced by near samples and approximation is done with an NN, the base network's approximation is accurate enough on its tile. On the other hand, in retrieving data, it is clear that the influence of a tile is insignificant in far subspaces. Furthermore, neighbouring tiles might use some of the tiles samples, close enough to satisfy the membership condition, might be used in training. It is clear that the constant compared with relative membership can be changed such that all samples having significant effect from a specific tile are considered in the training of that tile. Hence, an ATNN is a universal approximator, similar to NN.

B. Approximation Capacity

It can be 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 (n_t is the least number of hidden units of a network to learn the target function). 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 $x - 1 \leq \lfloor x \rfloor \leq x$, 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 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] $\vec{1}$ is a vector whose all elements are 1.

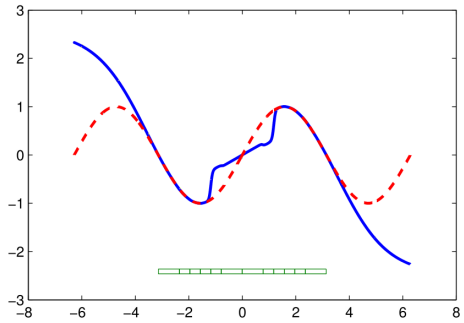


Fig. 1: The effect of memory when the tiled network is retrained with a new (sample,target) set. (first trained with sine function in $(-\pi, \pi)$ then with a line in $(-1, 1)$)

C. Approximation memory

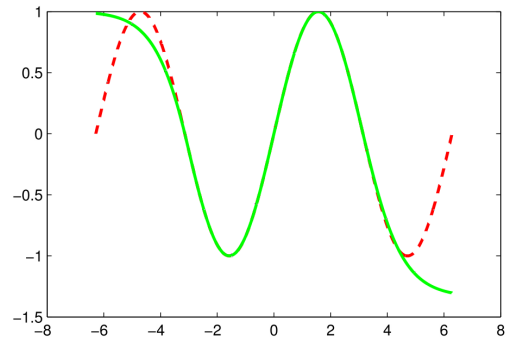
When an NN is trained, the weights corresponding to the network change, which affect the approximation of the NN in the whole instance space, a measurement of such phenomena in the conventional NNs is given in [19]. Such singular can crumble the whole approximation, unless all the previous training samples are considered in every training, which will cause an increase in the computational requirement.
On the other hand, if an ATNN is trained in a subspace and then it is trained on another subspace separate from the first one, the approximation of ATNN for the first subspace does not change and only the approximation of the second tile will change (see Figure. 1).
This feature of ATNN leads to a decrease in generalization on the untrained tiles, (the tiles having no training examples). This deficiency is solved by considering omitting or joining tiles. In such case the approximation will be based on the neighbouring tiles.

D. Generalization

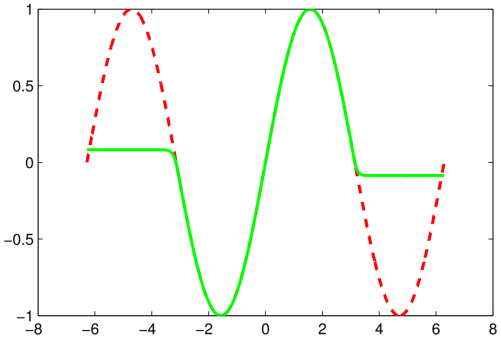
Adding memory to the approximator always reduces its generalization, since keeping old values means that the approximator also insist on keeping its previous not so accurate trainings; hence training with new samples do not help generalization of the target function. In the case of ATNN, approximation of the tiles having no samples do not change, so other samples do not help in correction of the tile predictions of the target function (decreasing the generalization between tiles). Of course, at the first training, when the tiles are being split the generalization is considered. For solving such problems we can consider joining the tiles.

E. Extrapolation

Usually expressive NN approximators cannot extrapolate, and the approximators output out of the system's training examples' subspace is usually constant. On the other hand, the ATNN tiles approximators are also NNs so the results of this approximator would be flat as shown in Figure 2, but as



(a) an NN with 5 hidden units trained to approximate sine function.



(b) an NN with 20 hidden units trained to approximate sine function.

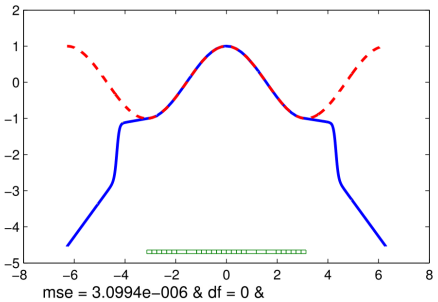
Fig. 2: The effect of the number of NN hidden units on the function learned. Both are trained with sine in $(-\pi, \pi)$.

we discussed before, with the decrease of the DoF (degree of freedom) of the NN the generalization increases. So if DoF of the base network is kept small enough, the approximator can be expressive enough and has the generalization quality for extrapolation (see Figure 3).

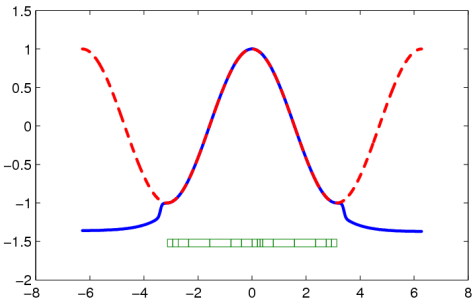
IV. More discussion on the proposed model

A. How and when to split a tile?

The problem of determining how and when to split a tile is one of the most challenging problems in tile coding. Here this problem will be "After how many epochs should the algorithm split a tile." The number of epochs depends on the smoothness of the target function and the Dof (degree of freedom) of the base network. Here, we split a tile whenever it fails to get to the desired accuracy after a fixed number of epochs. For a faster training, other conventional termination methods like gradient bounds might be useful.
The second problem is "How to split a tile." In one dimensional problems, there is only one answer, the tile will be split into two halfs. But in high dimensional problems, deciding on what dimension to split the tile, is difficult. Without any assumptions about the target function, two main methods might be used. he tile can be split along (i) the longer side or (ii) along a random dimension. In theoretical computations it is assumed that the



(a) Approximation of cosine function in $(-\pi, \pi)$ assuming the whole system can be learned linearly.



(b) Approximation of cosine function assuming the whole system can be learned with one hidden unit.

Fig. 3: The function is approximated locally and smoothly according to the assumption.

whole tiling is uniform, so the first method will be preferred for keeping the tiling uniform.

B. Tiles trained with few samples

When splitting the tiles, this question should always be considered: "If this tile is split, will the sub-tiles have enough samples to be trained?" Insufficient sampled tiles lead to approximations locally based on only few samples, decreasing the generalization of the approximator. Moreover, tiles are not allowed to have no samples because the localized information in such case would be zero and the respective networks would not have any samples to be trained with.
Hence, in splitting the tiles, the number of sub-tiles' samples must be considered. The tiles with no training samples might be omitted; In that case the approximation of the subspace would be based on neighbouring tiles.

C. Curse of dimensionality

It is obvious that if the *split* subroutine in the Algorithm 1 only splits the tiles in one dimension, it will take a long time in high-dimensional problems to shatter into small enough tiles for learning a very high-dimensional target function. This problem can easily be solved by changing *split* so that it splits tiles in all dimensions. Since it is possible that many of these tiles do not have sufficient samples, the generalization between tiles might be affected. However, if the tiles having no samples are omitted, the generalization will be based on near tiles (in Euclidean space).

D. Uniformed sample frequency of different functions in space

Assume that we have learned a specific function with a very high frequency of samples in space. Then, the target function is changed (for some reason), so some samples are fed into the ATNN and it is retrained. However the tiles are split too small to fit the first function and the samples for the modified function are less frequent than the first samples, some tiles might not have any samples and are not retrained. With some tiles still representing the first function, the approximation accuracy is critically decreased in those subspaces. The easy solution for such problem is to join or omit the tiles with no samples which will decrease the memory of the approximator. Joining the tiles with no samples will result in losing the data stored in those tiles so there must be a balance between joining and keeping the tiles.

E. The sharpness of edges

One of the problems faced in the tile coding methods, including ATNN, is that the approximation is not smooth. In order to smooth the edges of the tiles, the networks should be trained so that the edge is continuous. But smoothing the edges in such fashion will cause growth in the computational requirement, as will discussed later. Therefore, instead of forcing the NNs to have the same approximation on the edges, a fuzzy selection of NNs in retrieving data is considered. Since fuzzy membership of samples and tiles outputs are smooth, average of the tiles with respect to the fuzzy relative membership (the approximation) would be smooth. Toward this end, the process which determines the membership of samples to the tiles should be fuzzy. Below both methods mentioned above are discussed.

1) Overlapping bounds to force the networks to have the same approximation on edges: In order to fix sharpness problem, the bounds of tiles are set bigger than what they should be. Hence they will consider the samples of other neighbour networks in training, hence the network will have more likelihood with neighbour networks on the edges. This can be accomplished on *split* subroutine in Algorithm 1.

2) Fuzzy selection of networks: In order to find whether a sample is in a tile or not, some inequalities are examined. On each dimension the bound of the tile is compared to the point's values. If these inequalities are fuzzy, the whole process of finding the relative memberships would be fuzzy. In the following the structure of this fuzzy comparator is discussed. Fuzzy comparator must have 1 value when y is much larger than x and 0 value when y is much smaller than x. Also a fuzzy comparator value must only depend on the difference of x and y.

$$compare(x, y) = \frac{1}{1 + e^{sharpness \times (x-y)/Scale}} \qquad (1)$$

Sharpness is a constant which shows the fuzziness of the comparator, smaller *sharpness* correspond to fuzzier compara-

tors and ∞ *sharpness* correspond to digital comparator. It is obvious that a fixed *sharpness* is not suitable for all problems, for example in atomic size the differences are in order of 10^{-9} but in normal systems they are in order of 10^{0}, so the comparator must be scaled to the problem. In our problem the scale can be easily obtained with width of each tile.
Since being of a sample in training set a tile cannot be fuzzy, a threshold must be defined to be the criteria to train a tile with a specific sample.

F. Comparing ATNN with Cascade-Correlation method

As mentioned before the ATNN and the Cascade-Correlation have some similarities, and a comparison can make advantages and disadvantages of both methods clear. Consider the growth of a Cascade-correlation network. The first hidden unit output determines the position of the input relative to a hyperplane. The second hidden unit determines the position of input relative to another hyperplane and the first hidden unit's output. Such that for the same relative position to the second hyperplane and different relative position to the first hyperplane, the second hidden unit has different outputs. This attribute is like tiles in ATNN method, but two things are different: (i) in the Cascade-Correlation method these boundaries are not along axes but in ATNN all boundaries are along the axes and (ii) the gradient of approximated function in the Cascade-Correlation method on both sides of the boundary (in both tiles) are the same but in ATNN the gradient can differ in different tiles.

V. Practical results

A. 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.

1) The $sinc(x)$ function: The approximation of $sinc(x)$ function is shown in Figure 4. As mentioned before, ATNN's approximation generalizes the data assuming that the whole function can be learned with the base network. Since $sinc(x)$ function cannot be learned with an NN consisting of one hidden unit around its extremums the tiling rate is so high in that area comparing to other areas. As shown in the figure, the NN's approximation is flat in extrapolation.

2) The $sinc(\frac{1}{x})$ function: The approximation of $sinc(\frac{1}{x})$ function is shown in Figure 5. 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.

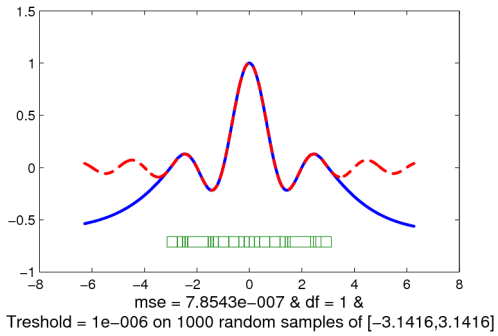


Fig. 4: Approximation of $sinc(x) = \frac{sin(\pi x)}{\pi x}$ function in $(-\pi, \pi)$ by base network consisting of a hidden unit.

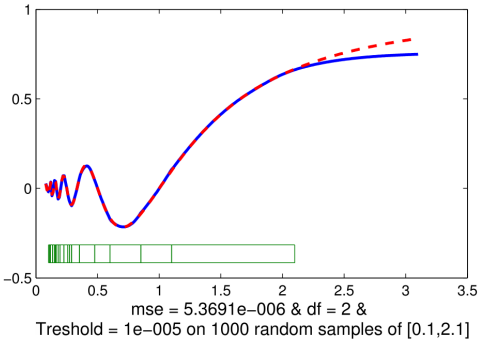


Fig. 5: Approximation of $sinc(\frac{1}{x}) = \frac{xsin(\pi \frac{1}{x})}{\pi}$ function in $(0.1, 2.1)$ by base network consisting 2 hidden units.

3) The stairs like function: The approximation of a stairs like function is shown in 6. This figure shows that by choosing an appropriate *sharpness* parameter, ATNN can approximate a discontinuous function with sudden changes with insignificant error.

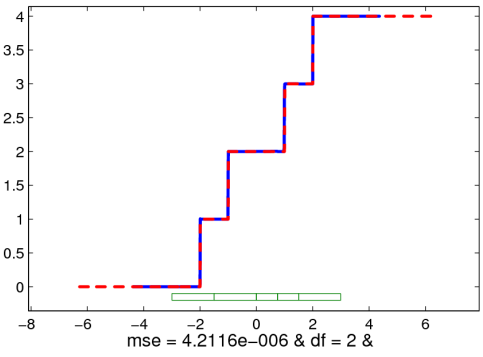


Fig. 6: Approximation of $u(x-2)+u(x-1)+u(x+1)+u(x+2)$ function in $(-3, 3)$ by base network consisting 2 hidden units.

B. More discussion on practical results

ATNN might have difficulties in approximating the function when its *sharpness* parameter is not tuned well.

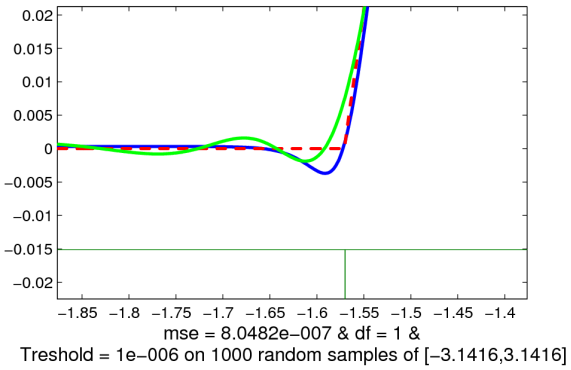


Fig. 7: The figure shows comparison between ATNN approximation (blue) and NN's approximation (green)

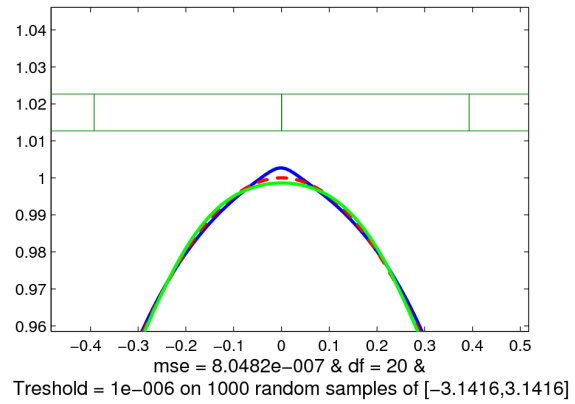


Fig. 8: The figure shows some advantages and disadvantages of ATNN in comparison to NN.

The following comparisons show that both NN and ATNN can achieve enough accuracy with acceptable computational effort. The Figure 7 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 8 shows how ATNN can be misled to a sharp point when there is a smooth extremum on the tile boundary.

VI. Conclusion and Future Works

In this paper we introduced ATNN as a new combinational function approximator based on conventional Neural Networks and tile coding approximators. We showed that this combination will help to make accurate enough approximations. We introduced *sharpness* and *threshold* parameter of structure to control the accuracy/complexity of approximation according to the function being approximated. Experimental results derived from comparing ATNN and NN show that ATNN approximation is usually more precise in sharp points. ATNN might be useful in tasks which keeping approximation of old trainings is needed. The important flaw with the *sharpness* parameter is that, ATNN might have some problems in the tiles' boundaries with different rate of variations. This can easily be solved by tuning the *sharpness* parameter manually by the supervisor. The *sharpness* parameter, as introduced have significant effect in approximation of keen of edges. One improvement for ATNN can be about discussing on methods for automatic fine-tuning *sharpness* parameter. Furthermore, here we showed applicability of ATNN by several functions, it is of high value to show its applicability in a more realistic application.

Acknowledgment

The authors would like to thank Shahab Karrari and Razieh Nokhbeh Zaeem for their reviews and helpful advices.

References

[1] E. B. Baum and D. Haussler, "What size net gives valid generalization?" *Neural Comput.*, vol. 1, no. 1, pp. 151–160, 1989.
[2] Y. Chauvin, "Generalization performance of overtrained backpropagation networks," in *Proceedings of the EURASIP Workshop 1990 on Neural Networks*. London, UK: Springer-Verlag, 1990, pp. 46–55.
[3] 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.
[4] G. Castellano, A. M. Fanelli, and M. Pelillo, "An iterative pruning algorithm for feedforward neural networks," *IEEE Trans. Neural. Networks*, pp. 519–531, 1997.
[5] S. E. Fahlman and C. Lebiere, *The Cascade-Correlation Learning Architecture*, 1991.
[6] 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.
[7] 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.
[8] 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.
[9] 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.
[10] S. Waugh, "Dynamic learning algorithms," 1994.
[11] T. Ash, "Dynamic node creation in backpropagation networks," *Neural Networks*, vol. 2, p. 623, 1989.
[12] 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.
[13] R. Reed, "Pruning algorithms: A review," *IEEE Trans. Neural. Networks*, vol. 4, pp. 740–747, 1993.
[14] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press, 1998.
[15] 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.
[16] C. Wu and W. Meleis, "Function Approximation Using Tile and Kanerva Coding For Multi-Agent Systems," 2009.
[17] 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.
[18] K. Hornik, M. B. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators." *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.
[19] S. Weaver, L. Baird, and M. Polycarpou, "An analytical framework for local feedforward networks," *Neural Networks, IEEE Transactions on*, vol. 9, no. 3, pp. 473–482, 2002.