Mathematical Findings on Artificial Neural Nets and Their Physical Meaning

Han-gyoo Kim
EECS Department
University of California, Berkeley

January 14, 1992

Abstract

This paper at first applies Stone-Weierstrass theorem to prove the existence of universal approximating capability of feedforward multilayer neural nets, and at second discusses the construction of such approximators. Since the fact that approximators can be built efficiently without using neural net structures provides insight of great importance into understanding the characteristics of neural nets, this paper further investigates the limit of neural net capability. Finally, feasibility of utilizing general approximators to solve AI problems is discussed.

1 Introduction

The capability of feedforward multilayer neural nets to approximate any functions quite well in real world applications has been recognized for the time being, and some of the recent research efforts have been devoted to find the approximation capability of neural nets. For example, Hornik et al. [1] proved that neural nets *can* approximate any Borel measurable functions from one finite dimensional space to another to any desired degree of accuracy. Their proof is, however, a direct application of Stone-Weierstrass theorem to the function class produced by the neural nets. Their work is nothing but a mathematical proof of existence of approximation capability in neural nets which has long been noticed in the academia of neural net research. Stone-Weierstrass theorem could, however, provide us with more insight about the nature of neural nets than just the guarantee of the existence of approximation capability.

This paper discusses mainly two important mathematical findings that resulted from applying Stone-Weierstrass theorem to the neural nets and inspecting the function class of neural net output. In section 2, we start with Stone-Weierstrass theorem to show that there exist functions which approximate any given real continuous function and those approximating functions take the form

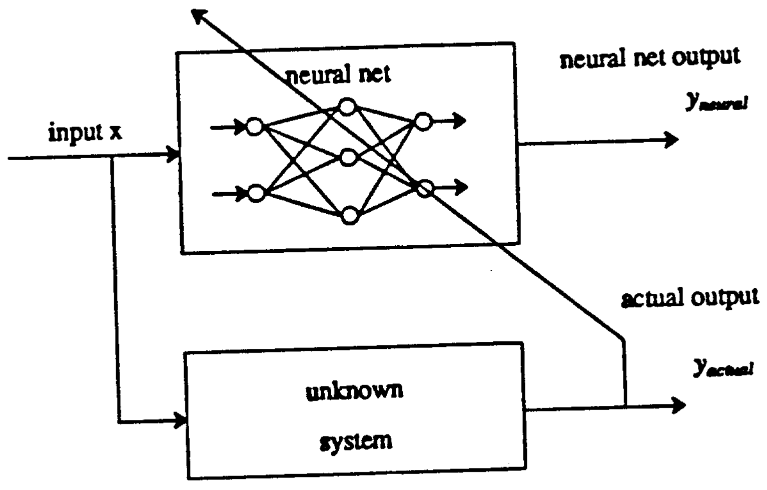


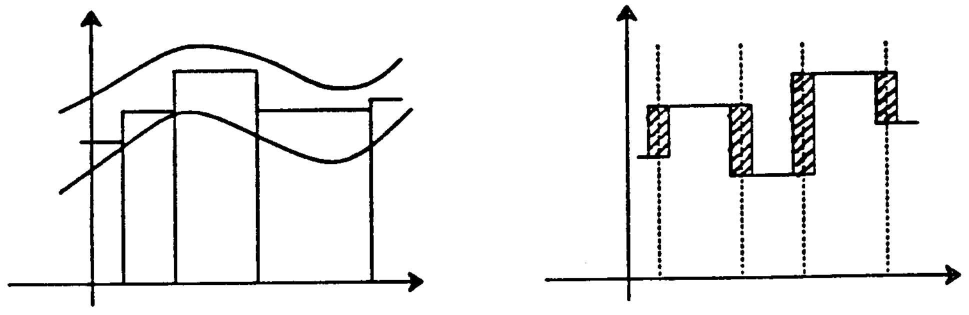
Figure 1: Typical Neural Net Application

of functions that *can* be generated by neural nets. This theorem, however, is nothing but an existence proof, and does *not* provide the design methodology how to build the approximating functions. Nonetheless, the proof tells us the fact that the approximation is the main characteristic of neural nets.

As is illustrated in Figure 1, most typical example of neural net application would be to use a neural net structure to find an approximate relation between the input and the output of a system of interest. As the neural net is getting learned, that is, adjusting its weights by comparing neural net's outputs with actual outputs, it is expected that the neural net eventually produces outputs close enough to the actual desired values.

In practice, neural nets would be used to find approximation in the applications that do usually not require precise approximation, i.e., the applications in which neural nets are only demanded to approximate already obscure function. See Figure 2(a). Other important area of neural net application would be the input classifiers or pattern recognizers equivalently. In other words, given a set of inputs neural nets are asked to classify those inputs into subsets of inputs such that the input elements of a classified subset are considered to share commonality while the boundaries among the classified subsets are not clearly cut. See Figure 2(b). In any case those applications only relax the accuracy requirement but do not change the requirement of approximation itself.

This fact that neural nets are approximators, in turn, provides us with important insight about what the neural nets are actually doing and raises a question: Why do we need neural nets to find an approximator *if* we have better approximators otherwise? Or more constructively, what kinds of algorithms can we use to build working neural nets? Section 3 contains the mathematical findings that



(a) loose approximator (b) input classifier

Figure 2: Relaxed Approximation

led us to the above question by closely looking at the function class of neural net output.

Throughout the paper, we are concerned about the feedforward multilayer neural net as is depicted in Figure 3 without loss of generality of discussing the properties of neural nets. Input to a hidden node takes the form of affine function $A(x) = w \cdot x + b$ where x is input vector, w is weight vector, and b is scalar bias. Output from a hidden node is the value of a certain type of squashing function[1], $G(\cdot)$, that is, $G(A(x))$. Therefore, overall output of a neural net constitutes the class of functions that take the form of $\sum_j \beta_j G(A_j(x))$ where β_j's are scalar weights from hidden nodes to output node. There have been suggested various learning algorithms to adjust the weights and biases.

2 Neural Net as Approximator

For the ease of mathematical manipulation, we consider single output neural nets, but all the results shown in this paper also pertain to multi-output neural nets.

Before we start with Stone-Weierstrass theorem (S-W theorem in the sequel), we present some mathematical definitions with explanation as appropriate to understand the S-W theorem.

Definition 1 (algebra) A linear space A of real functions defined on a set E is an *algebra* if the product of any two elements in A is again in A. Thus A is an algebra if for any two functions f and g in A and any real numbers a and b we have $af + bg$ in A and fg in A. A family A of functions on E is said to *separate* points if given distinct points x and y of E there is an f in A such that

[1]To be defined later.

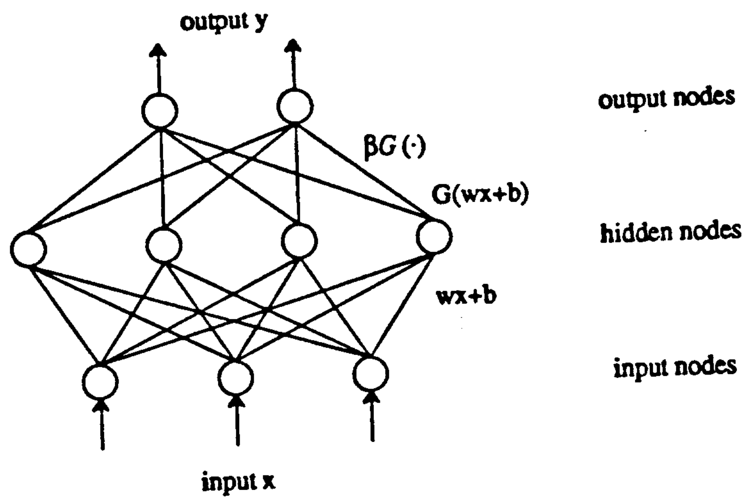


Figure 3: Feedforward Multilayer Neural Net

$f(x) \neq f(y)$. The family A *vanishes at no point* of E if for each x in E there exists f in A such that $f(x) \neq 0$. □

Definition 2 (dense) A subset S of a metric space (X, ρ) is ρ-dense in a subset T if for every $\epsilon > 0$ and for every $t \in T$ there is an $s \in S$ such that $\rho(s,t) < \epsilon$. □

That is, ρ-dense means that we always have at least one function that is close enough to any given function to any degree of accuracy.

Here follows the S-W theorem.

Theorem 1 (Stone-Weierstrass) [2] Let X be a compact space and A an algebra of continuous real-valued functions on X which separates the points of X and which contains the constant functions. Then given any continuous real-valued function f on X and any $\epsilon > 0$ there is a function g in A such that for all x in X we have $\mid g(x) - f(x) \mid < \epsilon$. In other words, A is ρ_X-dense. □

The S-W theorem tells that certain family of continuous functions satisfying the conditions of S-W theorem contains a function that approximates an arbitrarily given function to any degree of accuracy on any given input domain.

Let's now look at the function family generated by neural nets. As is shown in the previous section, the output of neural nets is a family of:

$$\sum_j \beta_j G(A_j(x))$$

where $G(\cdot)$ is sqaushing function and $A_j(\cdot)$ is affine function defined respectively below. So the output function family of neural nets is a linear space of

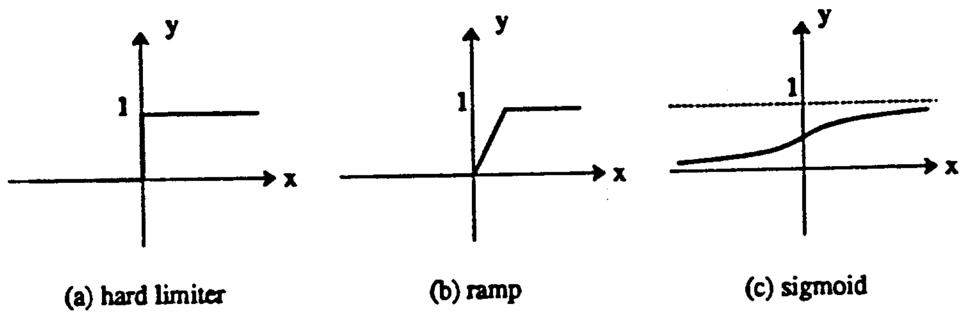


Figure 4: Examples of Squashing Functions

affinely shifted squashing functions. We denote the above class of affinely shifted squashing functions as $\sum^r(G)$ in the sequel.

Definition 3 (squashing function) A non-decreasing function $g : R \rightarrow [1,0]$ is a squashing function if $\lim_{x\rightarrow\infty} g(x) = 1$, and $\lim_{x\rightarrow-\infty} g(x) = 0$. □

Some of the examples of squashing functions are hard limiter, ramp function, etc. as given in Figure 4.

Definition 4 (affine function) For any $r \in N$ where N is an integer, A^r is the set of all affine functions from R^r to R, that is, the set of all functions of the form $A(x) = w \cdot x + b$ where $w, x \in R^r$ and $b \in R$. □

Theorem 2 given below proves that neural net output function class, $\sum^r(G)$, *can* approximate any arbitrary real valued continuous function to any degree of accuracy. Hornik ae al.[1] originally provided a long proof of theorem 2 with minor mistakes. A succint proof is given in the appendix. The proof given in this paper follows totally different approach than the approach of Hornik and others.

Theorem 2 [1] For every squashing function G, every r, $\sum^r(G)$ is a dense subset of set of all continuous real valued functions from R^r to R on arbitrary compact set X. □

To recap the mathematical findings so far, we show that the neural net output can approximate any given continuous function provided that the neural net has enough number of hidden nodes and appropriate learning algorithms. However, we do not know how to build such a neural net, that is, we have no ideas how many hidden nodes are needed and how well a neural net approximate, etc. Next section investigates about the construction of approximators.

3 Construction of Approximators

As we have found in the previous section, neural nets are universal approximators *if* built properly. In this section we further look into the infra structures of neural nets and the function classes of neural net outputs in orer to understand more about the characteristics of neural nets.

We may have interesting observation if we look at the various types of squashing functions of the hidden nodes. Although theorem 2 proves that we can use any squashing function in the hidden nodes for approximation, we see some of the function classes we use for the hidden nodes are *algebra*. Hard limiter is a simple but important example of such kinds. The family of affinely shifted hard limiters is algebra because multiplication of two elements in the linear space of affinely shifted hard limiters becomes a member of that space again:

$$g_1 \cdot g_2 \in \sum^r G(\cdot) \text{ if } g_1, g_2 \in \sum^r G(\cdot)$$
$$\text{where} \sum^r G(\cdot) = \left\{ g : g = \sum_j \beta_j h_j(ax+b), a, b \in R, h(x) = \begin{pmatrix} 0 & x < 0 \\ 1 & x \geq 0 \end{pmatrix} \right\}$$

Another example of such classes is the class of arctan(·).

Besides the fact that it is guaranteed the existence of universal approximators if hard limiters are used for the hidden nodes, the hard limiter case gives us an important insight about the characteristics of neural nets. Even though we do not know how to build approximating neural net using hard limiters, we have already long used universal approximators made of hard limiters, that is, analog to digital converters by way of pulse sequences.

It is interesting to notice that the rectangular pulse sequences can be actually generated by the neural net that uses hard limiter in its hidden nodes. That is, the unit pulse with the width of δ is made by taking the difference of two hard limiters shifted by δ:

$$p(x) = \begin{cases} 1 & \text{for } 0 \leq x < \delta \\ 0 & \text{otherwise} \end{cases} = h(x) - h(x - \delta)$$

Figure 5 shows an example of neural net structure with single input that constitutes the sequence of pulses with equal width of δ at the hidden nodes. In Figure 5, all the weights, w's, from input node to hidden nodes are 1 identically. The i-th hidden node has the bias b_i such that $b_i = -i \cdot \delta$ for $-N \leq i \leq N$ where N is an integer. The output of the i-th hidden node is accordingly a hard limiter shifted by $i \cdot \delta$. So any two contiguous hidden nodes constitute a unit pulse of width δ. This is actually how the final neural net output y is constructed. The i-th output node receives signals only from outputs of i-th an (i+1)-th hidden nodes. That is, i-th output node has all zero weights identically except for the two edges (b_i, y_i) and (b_{i+1}, y_i). In Figure 6, we eliminated all the edges with ientically zero weights for clarity. The weight between y_i and b_i is assigned β_i such that β_i is the actually measured output, i.e., learned value. The weight

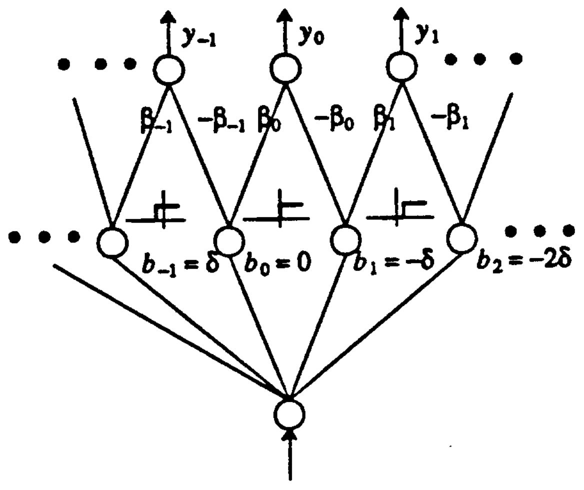


Figure 5: Neural Net Producing Pulse Sequence

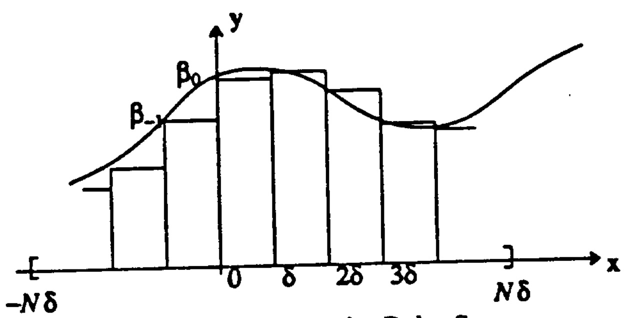


Figure 6: Approximation by Pulse Sequence

between y_i and b_{i+1} is set $-\beta_i$. It is easy to see that y_i is nothing but a pulse spanning from $x = i \cdot \delta$ to $x = (i+1) \cdot \delta$ of height β_i:

$$\ldots, y_0 = \begin{cases} \beta_0 & 0 \le x < \delta \\ 0 & \text{otherwise} \end{cases}, y_1 = \begin{cases} \beta_1 & \delta \le x < 2\delta \\ 0 & \text{otherwise} \end{cases}, \ldots$$

The neural net in Figure 5, therefore, emulates the pulse sequence that approximates actual function on $[-N\delta, N\delta]$. See Figure 6. We know that at least theoretically any degree of accuracy can be achieved by selecting appropriate pulse width δ.

Note further from theorem 2 that not only the squashing functions satisfying the conditions of the S-W theorem such as the hard limiter but any squashing function serves well to construct pulse sequence for approximation purpose. Two ramp functions, for instance, can be used to build spike pulse as in Figure 7, and the spike pulse can be easily realized using neural net structure similarly.

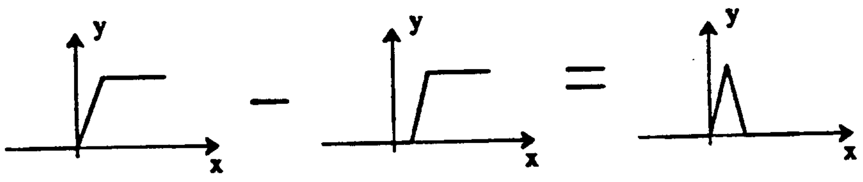


Figure 7: Spike Pulse Made of Two Ramp Functions

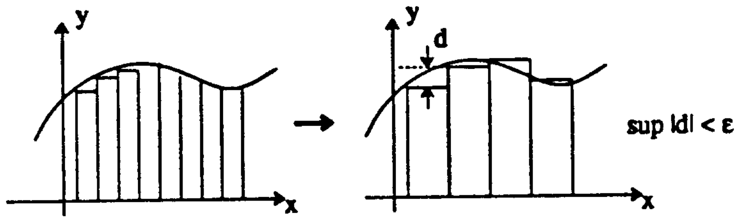


Figure 8: Histogram Approximation

The above trivial but important example leads us to a question: Doesn't a neural net actually converge to a structure such as given in the above example? In other words, Isn't the approximation by pulse sequence all what a neural net is actually doing?

The observation we made so far is disappointing because we do expect neural nets to do more than just brute force approximation by pulse sequences. The approximation by pulse sequence is a total waste in terms of amount of memory it requires. We do not need dense pulses for the portions that do not vary rapidly. Maybe neural nets approximate to the same degree of accuracy spending less amount of memory.

Actually, we know obvious and effective improvement over brute force pulse sequence approximation. One simple improvement is to replace the unnecessarily dense parts of pulse sequence with single thick rectangular column as long as the replacement by the column preserves the accuracy requirement, $\sup|d| < \epsilon$, where d is the difference between the desired value and approximation. See Figure 8.

Probably, a good neural net using hard limiters in its hidden nodes would come up with the approximating columns similar to the ones in Figure 8. Note that such a neural net can easily be constructed simply by changing biases in Figure 5 and such a neural net will save a lot of nodes to achieve the same

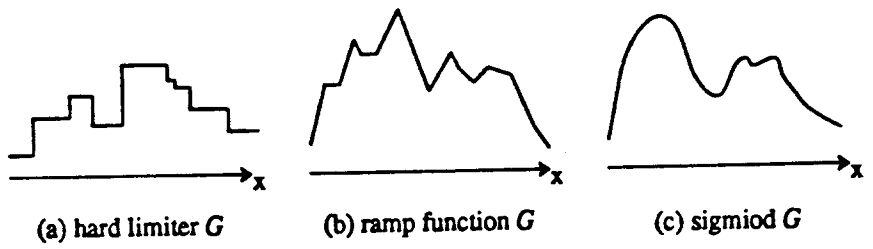


Figure 9: Various Neural Net Outputs

degree of approximation accuracy compared to the pulse sequence neural net approximator.

Note, however, further that such an approximation can easily done having nothing to do with neural nets. We know that we can devise reasonable algorithms that produce histograms that approximate any continuous function on any compact input domain. Anyway, it seems feasible to devise the optimal algorithm or pseudo-optimal algorithms that find a histogram which approximate a function on a compact input domain using near minimal number of columns. Even a simple minded algorithm will serve our purpose to greatly reduce the number of columns for approximation. Moreover, any of such algorithms will consume much less learning samples compared to the enormous amount of learning samples usually required for neural nets. We do not dwell on the invention of such algorithms here since it is of minor interest and difficulty. What is important here to pay attention is to find the real characteristics of neural nets.

The observations in this section give a belief that neural nets are approximators. This belief is more evident if we closely look at the neural net outputs. As is shown in the previous section, the output of a neural net is a linear space of affinely shifted squashing functions, $\sum_j \beta_j G(A_j(x))$, at all. Figure 9 shows the plots of neural net outputs for various squashing function they can use. If we use hard limiters in the hidden nodes, the output of the neural net forms a ragged edge as shown in Figure 9(a). If ramp functions are used, the output becomes a graph made of straight lines as in Figure 9(b). Squashing functions whose first derivatives are continuous will produce curves in general as in Figure 9(c). Anyway, it is clear that any of those outputs will be meaningful only when the output approximates a certain relationship of concern.

4 Discussion and Conclusions

Mathematical findings in this paper include:

1. The S-W theorem and theorem 2 show that the neural net output can approximate any real functions on any compact domain. This tells us that neural nets do approximation.

2. Roles of neural nets in real world applications are approximators or input classifiers.

3. We find that neural net output can actually produce pulses or columns that can successfully approximate any continuous function.

4. Inspection of the function class of neural net outputs shows that neural net outputs are meaningful only when they are approximators of certain relationship of interest.

5. Above all, we also know that we do not need neural net structure in order to build such approximators since easy-to-build non-neural net type approximators do the jobs efficiently.

Realizing that the output of a neural net is a linear space of affinely shifted squashing functions at all and that non-neural net type approximators can be constructed more efficiently using such functions, we have to investigate more about the genuine characteristics of neural nets other than just approximation property if any exists before we push all the unfair burdens of AI problems to the obscure neural net structures.

It will be valuable to perform a simulation study to see if the infra structures of neural nets converge to form approximators or something else that do more than just approximation.

As we have observed in the previous sections it would raise some doubts about the utility of neural nets if the sole purpose of using neural nets is to approximate. Neural net approximators are disadvantageous compared to non-neural type approximators, e.g., histogram approximator, since we do not have a priori knowledge about the structure of neural net required to solve a specific approximation problem. Moreover, neural net approximator, even though we hope it works, requires ample multitude of learning samples which non-neural approximators do not need.

It is suggested to incorporate the idea of non-neural net type approximators into the structure of neural nets. We may be able to invent neural nets that can approximate more efficiently than the currently used neural nets.

We have seen many experimental works of neural nets, but we have to realize that most of the roles of neural nets in various applications have been approximators or input classifiers. In this sense, we suggest that more researches should be performed how we can effectively utilize approximators to solve AI problems and we will have to put our endeavors towards the understanding of why and how approximators can solve various AI problems instead of pursuing myths of neural nets blindingly.

5 Appendix

Proof of Thm.2

All we have to show is that $\sum^r(G)$ is ρ_X-dense for every fixed squashing function $G(\cdot)$.

We know that the set of all polynomials form an algebra. So the set of all polynomials is ρ_X dense on any arbitrary compacta X. For every G, for arbitrary compacta X, and for arbitrary $\epsilon > 0$, there exists $g \in \sum^r(G)$ such that $\sup_{x \in X} \mid p(x) - g(x) \mid < \epsilon$ where $p(x)$ is any polynomial. We can always find such $g(x)$ because we can build a pulse shaped function of unit height with two affinely shifted squashing functions such that the width of the pulse shaped function can be set arbitrarily narrow. It is clear that any polunomial can be approximated on any compacta by the linear combination of affinely shifted pulse shaped function created as above. That is, we can approximate any polynomial on any compacta using the sequence of arbitrarily narrow pulses. This shows that $\sup_{x \in X} \mid p(x) - g(x) \mid < \epsilon$ where $g(x) = \sum_j G(A_j(\cdot))$ for an appropriate j. This proves, in turn, that $\sum^r(G)$ is ρ_X-dense. $\square$

References

[1] K. Hornik et al. Multilayer Feedforward Networks are Universal Approximators *Neural Networks* Vol.2 pp. 359-366,1989

[2] H. L. Royden *Real Analysis* 2nd ed. MacMillan Pub. Co. 1968