A Note on the Relationships between Logic Programs and Neural Networks

Pascal Hitzler*
Department of Mathematics, University College, Cork, Ireland
Email: phitzler@ucc.ie
Web: http://maths.ucc.ie/~pascal/

Anthony Karel Seda
Department of Mathematics, University College, Cork, Ireland
Email: aks@ucc.ie
Web: http://maths.ucc.ie/~seda/

Abstract

Several recent publications have exhibited relationships between the theories of logic programming and of neural networks. We consider a general approach to representing normal logic programs via feedforward neural networks. We show that the immediate consequence operator associated with each logic program, which can be understood as implicitly determining its declarative semantics, can be approximated by 3-layer feedforward neural networks arbitrarily well in a certain measure-theoretic sense. If this operator is continuous in a topology known as the atomic topology, then the approximation is uniform in all points.

1 Introduction

Logic Programs and Neural Networks are two important paradigms in Artificial Intelligence. Their abilities, and our theoretical understanding of them, however, seem to be rather complementary. Logic Programs are highly recursive and well understood from the point of view of declarative semantics. Neural Networks can be trained but yet lack a declarative reading. Recent publications, for example [4, 13, 14], suggest studying the relationships between the two paradigms with the long-term aim of merging them in such a way that the advantages of both can be combined.

The results we wish to discuss draw heavily on the work of Hölldobler, Kalinke and Störr [13, 14], which we will in part generalize. It will be convenient to briefly review their approach and their results. For undefined notions, see Section 2.

In [13], a strong relationship between propositional logic programs and 3-layer feedforward and recurrent networks was established. For each such program P, a 3-layer feedforward network can be constructed which computes the single-step or immediate consequence operator T_P associated with P. To this end, each atom in P is represented by one or more units in the network. If the program is such that iterates of T_P, for any initial value, converge[1] to a unique fixed point of T_P, (which can be understood to be the declarative semantics[2] of P), then the network can be cast into a recurrent network which settles down into a unique

*The first named author acknowledges financial support under grant SC/98/621 from Enterprise Ireland.

[1]This convergence is essentially convergence in the atomic topology (introduced in Definition 2.1), but this was not noted in [13] however.

[2]In the sense of the supported model semantics or Clark completion semantics [6].

stable state corresponding to the fixed point. On the other hand, for each 3-layer network a propositional logic program P can be constructed such that the corresponding operator T_P is computed by the network.

In [14], an attempt was made to obtain similar results for logic programs which are not propositional, that is, for programs which allow variables. The main obstacle which has to be overcome in this case is that the Herbrand base, which is the domain on which the logic programs operate, is in general infinite; it is therefore not possible to represent an atom by one or more units in the network. The solution suggested in [14] uses a general result due to Funahashi [8], see Theorem 3.1, which states that every continuous function on a compact subset of the real numbers can be uniformly approximated by certain types of 3-layer neural networks. By casting the T_P-operator into such a function, approximating the single-step operator is shown to be possible.

In order to obtain a continuous real-valued function from T_P, metrics were employed in [14]. For acyclic[3] logic programs, a complete metric[4] can be obtained which renders the single-step operator a contraction, see also [19]. By identifying the single-step operator with a mapping on the reals, a contractive, and therefore continuous, real-valued function is obtained which represents the single-step operator. This function can in turn be approximated by neural networks due to the result of Funahashi mentioned above. For certain kinds of acyclic programs[5], the resulting network can then again be cast into a recurrent network which settles down into a unique stable state corresponding to the unique fixed point of the operator.

In this paper, we will investigate a more general approach to representing the single-step operator for (non-propositional) normal logic programs by neural networks.

After some preliminaries in Section 2, we will, in Section 3, use a result from [18] which characterizes continuity of the single-step operator in the atomic topology, and apply the approximation theorem of Funahashi in order to approximate single-step operators by neural networks.

In Section 4, we will show that for any given normal logic program, its associated single-step operator can be realized as a Borel measurable real-valued function. An approximation theorem due to Hornik, Stinchcombe and White [15], see Theorem 4.1, can then be applied to show that each single-step operator for any normal logic program can be approximated arbitrarily well by neural networks in a metric ϱ_μ defined in measure-theoretic terms in Section 4, see Theorem 4.1.

Finally, in Section 5, we briefly discuss our results and indicate the extensions of them which we are currently pursuing.

2 Preliminaries

We assume that the reader is familiar with basic topological and measure-theoretic notions as well as basic notions from logic programming and the theory of neural networks. However, we will review some of the concepts which are important in the sequel.

Logic Programs

A normal logic program is a finite set of clauses (in the sense of first order logic) of the form

$$\forall(A \leftarrow B_1 \wedge \cdots \wedge B_k \wedge \neg C_1 \wedge \cdots \wedge \neg C_l),$$

usually written as

$$A \leftarrow B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_l,$$

where A, all the B_i and all the C_j are atoms over some underlying first order language. As usual, we refer to a negated atom, for example $\neg C$, as a literal. We call A the head of the clause, and $B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_l$

[3]These programs were called recurrent in [14], see also [5, 1].

[4]This metric also generates the atomic topology if the level mapping has the additional property that the preimage of each $n \in \mathbb{N}$ is a finite set, see [18], which is a condition fulfilled by all the level mappings which arise in practice.

[5]Acyclic programs which admit an injective level mapping.

the body of the clause. Our notation and terminology generally follow that of [16], which is also our reference for background in logic programming.

We will work over Herbrand interpretations only. Thus, B_P denotes the Herbrand base for a given program P, that is, the set of all ground atoms over the underlying language of P. As usual, Herbrand interpretations for P can be identified with subsets of B_P. In turn, this means that the set I_P of all Herbrand interpretations for P can be identified with the power set of B_P, and this identification will normally be made. The set of all ground instances of clauses in P will be denoted by ground(P).

Note that B_P is countably infinite if there is at least one function symbol that occurs in P. In the sequel, we will impose the mild condition that this is indeed the case. In fact, if no function symbols occur in P, then B_P is finite. Thus, P can be thought of as a propositional program, and this case was handled in [13] as already noted in the introduction.

A major tool for analysing logic programs (and for assigning a denotational semantics to a program) is the single-step or immediate consequence operator $T_P : I_P \to I_P$ associated with a given program P. This operator is defined as follows: for $I \in I_P$, we set $T_P(I)$ to be the set of all $A \in B_P$ for which there exists a clause $A \leftarrow B_1, \ldots, B_k, \neg C_1, \ldots, \neg C_l$ in ground(P) such that $I \models B_1 \wedge \cdots \wedge B_k \wedge \neg C_1 \wedge \cdots \wedge \neg C_l$ (so, in the classical two-valued logic we are using here we have, for all i, j, that $B_i \in I$ and $C_j \notin I$). It is well-known that an interpretation $I \in I_P$ is a model of P if and only if $T_P(I) \subseteq I$, that is, if and only if I is a prefixed point of T_P. In particular, the fixed points of T_P are called the supported models of P and they play an especially important rôle in the theory. In many ways a program P can essentially be represented by its associated single-step operator, and we will take up this point again in the discussion in Section 5.

We now proceed to define the atomic topology Q on I_P, which will play a major rôle in the development. The atomic topology was introduced in [18] as a generalization of the query topology [3], but coincides with it in the context of Herbrand preinterpretations.

2.1 Definition Let P be a normal logic program. The set $\mathcal{G} = \{\mathcal{G}(A) : A \in B_P\} \cup \{\mathcal{G}(\neg A) : A \in B_P\}$, where $\mathcal{G}(L) = \{I \in I_P : I \models L\}$ for each literal L, is the subbase of a topology called the atomic topology Q on I_P.

The basic open sets of Q are of the form $\mathcal{G}(A_1) \cap \cdots \cap \mathcal{G}(A_k) \cap \mathcal{G}(\neg B_1) \cap \cdots \cap \mathcal{G}(\neg B_l)$, of course, being derived from the subbasic open sets in the usual way, and we will denote such an open set by $\mathcal{G}(A_1, \ldots, A_k, \neg B_1, \ldots, \neg B_l)$. Note also that the collection of basic open sets is countable.

The Cantor Topology

As mentioned above, I_P can be identified with the powerset of B_P. It can therefore also be identified with the set 2^{B_P} of all functions from B_P to $\{0, 1\}$ (or to any other two-point space). Using this latter identification, the topology Q becomes a topology on the function space 2^{B_P}, and is exactly the product topology (of point-wise convergence) on 2^{B_P} if the two-point space is endowed with the discrete topology, see [18] for details.

If we interpret I_P as the set of all functions from B_P to $\{0, 2\}$, so that we now take the two-point space as $\{0, 2\}$, we can identify I_P with the set of all those real numbers in the unit interval $[0, 1]$ which can be written in ternary form without using the digit 1; in other words we can identify I_P with the Cantor set. The product topology mentioned above then coincides with the subspace topology inherited from the natural topology on the real numbers, and the resulting space is called the Cantor space $\mathcal{C}$. Thus, the Cantor space $\mathcal{C}$ is homeomorphic to the topological space (I_P, Q), and in the following $\iota : I_P \to \mathcal{C}$ will denote a homeomorphism between I_P and $\mathcal{C}$, see [18] for more details. It is well-known that the Cantor space is a compact subset of $\mathbb{R}$, see [21], and we can define $l(x) = \max\{y \in \mathcal{C} : y \leq x\}$ and $u(x) = \min\{y \in \mathcal{C} : y \geq x\}$ for each $x \in [0, 1] \setminus \mathcal{C}$.

We refer the reader to [21] for background concerning elementary topology.

Neural Networks

A 3-layer feedforward network (or single hidden layer feedforward network) consists of an input layer, a hidden layer, and an output layer. Each layer consists of finitely many computational units. There are connections from units in the input layer to units in the hidden layer, and from units in the hidden layer to units in the output layer. The input-output relationship of each unit is represented by inputs x_i, output y, connection weights w_i, threshold θ, and a function ϕ as follows:

$$y = \phi\left(\sum_i w_i x_i - \theta\right).$$

The function ϕ, which we will call the squashing function of the network, is usually nonconstant, bounded and monotone increasing, and sometimes also assumed to be continuous. We will specify the requirements on ϕ that we assume in each case.

We assume throughout that the input-output relationships of the units in the input and output layer are linear. The output function of a network as described above is then obtained as a mapping $f : \mathbb{R}^r \to \mathbb{R}$ with

$$f(x_1, \ldots, x_r) = \sum_j c_j \phi\left(\sum_i w_{ji} x_i - \theta_j\right),$$

where r is the number of units in the input layer and the constants c_j correspond to weights from hidden to output layers.

Measurable Functions

A collection M of subsets of a set X is called a σ-algebra if (i) $\emptyset \in M$; (ii) if $A \in M$ then its complement ${}^cA \in M$; (iii) if (A_n) is a sequence of sets in M, then the union $\bigcup A_n \in M$. The pair (X, M) is called a measurable space. A function $f : X \to X$ is said to be measurable with respect to M if $f^{-1}(A) \in M$ for each $A \in M$.

If M is a collection of subsets of a set X, then the smallest σ-algebra $\sigma(M)$ containing M is called the σ-algebra generated by M. In this case, a function $f : X \to X$ is measurable with respect to $\sigma(M)$ if $f^{-1}(A) \in \sigma(M)$ for each $A \in M$. If $\mathcal{B}$ is the subbase of a topology $\mathcal{T}$, and $\mathcal{B}$ is countable, then $\sigma(\mathcal{B}) = \sigma(\mathcal{T})$. If $\mathcal{B}$ is a subbase of the natural topology on $\mathbb{R}$, then $\sigma(\mathcal{B})$ is called the Borel σ-algebra on $\mathbb{R}$, and a function which is measurable with respect to this σ-algebra is called Borel-measurable. A measure on $(\mathbb{R}, \sigma(\mathcal{B}))$ is called a Borel-measure.

We refer the reader to [2] for background concerning elementary measure theory.

3 Approximating Continuous Single-Step Operators by Neural Networks

Under certain conditions, given in Theorem 3.2, the single-step operator associated with a logic program is continuous in the atomic topology. By identifying the space of all interpretations with the Cantor space, a continuous function on the reals is obtained which can be approximated by 3-layer feedforward networks. We investigate this next.

The following Theorem can be found in [8, Theorem 2].

3.1 Theorem Suppose that ϕ is non-constant, bounded, monotone increasing and continuous. Let $K \subseteq \mathbb{R}^n$ be compact, let $f : K \to \mathbb{R}$ be a continuous mapping and let $\varepsilon > 0$. Then there exists a 3-layer feedforward network with squashing function ϕ whose input-output mapping $\bar{f} : K \to \mathbb{R}$ satisfies $\max_{x \in K} d(f(x), \bar{f}(x)) < \varepsilon$, where d is a metric which induces the natural topology on $\mathbb{R}$.

In other words, each continous function $f : K \to \mathbb{R}$ can be uniformly approximated by input-output functions of 3-layer networks.

We already know that the Cantor space $\mathcal{C}$ is a compact subset of the real line and that the topology $\mathcal{C}$ inherits as a subspace of $\mathbb{R}$ coincides with the Cantor topology on $\mathcal{C}$. Also, the Cantor space $\mathcal{C}$ is homoeomorphic to I_P endowed with the atomic topology Q. Hence, if the T_P-operator is continuous in Q, we can identify it with a mapping $\iota(T_P) : \mathcal{C} \to \mathcal{C} : x \mapsto \iota(T_P(\iota^{-1}(x)))$ which is continous in the subspace topology of $\mathcal{C}$ in $\mathbb{R}$.

The following characterization of continuity in Q was given in [18, Theorem 9].

3.2 Theorem Let P be a normal logic program. Then T_P is continous in Q if, for each $I \in I_P$ and for each $A \in B_P$ with $A \notin T_P(I)$, either there is no clause in P with head A or there is a finite set $S(I,A) = \{A_1,\dots,A_k,B_1,\dots,B_{k'}\}$ of elements of B_P with the following properties:

(i) $A_1,\dots,A_k \in I$ and $B_1,\dots,B_{k'} \notin I$.

(ii) Given any clause C with head A, at least one $\neg A_i$ or at least one B_j occurs in the body of C.

As a corollary, one obtains that programs without local variables[6] have continuous single-step operators. It can also be shown that acyclic[7] programs have continuous single-step operators. For the slightly larger classes of acceptable [1] and locally hierarchical [5] programs[8], the single-step operator is in general not continuous, see [11, 19].

We can now present our first main theorem.

3.3 Theorem Let P be a normal logic program. If, for each $I \in I_P$ and for each $A \in B_P$ with $A \notin T_P(I)$, either there is no clause in P with head A or there is a finite set $S(I,A) = \{A_1,\dots,A_k,B_1,\dots,B_{k'}\}$ of elements of B_P satisfying the properties (i) and (ii) of Theorem 3.2, then T_P (more precisely $\iota(T_P)$) can be uniformly approximated by input-output mappings of 3-layer feedforward networks.

In particular, this holds for the operator T_P if P is acyclic or does not contain any local variables.

Proof: Under the conditions stated in the Theorem, the single-step operator T_P is continuous in the atomic topology. Using a homeomorphism $\iota : I_P \to \mathcal{C}$, the resulting function $\iota(T_P)$ is continuous on the Cantor space $\mathcal{C}$, which is a compact subset of $\mathbb{R}$. Applying Theorem 3.1, $\iota(T_P)$ can be uniformly approximated by input-output functions of 3-layer feedforward networks. ∎

4 Approximating the Single-Step Operator by Neural Networks

By Theorem 3.1, continuous functions can be uniformly approximated by input-output functions of 3-layer feedforward networks. It is also possible to approximate each measurable function on $\mathbb{R}$, but in a much weaker sense. We will investigate this in the present section.

The following was given in [15, Theorem 2.4]

4.1 Theorem Suppose that ϕ is a monotone increasing function from $\mathbb{R}$ onto $(0,1)$. Let $f : \mathbb{R}^r \to \mathbb{R}$ be a Borel-measurable function and let μ be a probability Borel-measure on $\mathbb{R}^r$. Then, given any $\varepsilon > 0$, there exists a 3-layer feedforward network with squashing function ϕ whose input-output function $\bar{f} : \mathbb{R}^r \to \mathbb{R}$ satisfies

$$\varrho_\mu(f,\bar{f}) = \inf\{\delta > 0 : \mu\{x : |f(x) - \bar{f}(x)| > \delta\} < \delta\} < \varepsilon.$$

In other words, the class of functions computed by 3-layer feedforward neural nets is dense in the set of all Borel measurable functions $f : \mathbb{R}^r \to \mathbb{R}$ relative to the metric ϱ_μ defined in Theorem 4.1.

We have already noted that the operator T_P is not continuous in the topology Q in general, nor is it continuous in the Scott topology on I_P in general (it satisfies this latter property when P is definite, that

[6] A variable is called local if it occurs in the body of a clause, but not in the corresponding head.

[7] See [19, 14].

[8] A unifying approach to these classes can be found in [10].

is, when P contains no negative literals). We proceed to show next that the single step operator has the pleasing property that it is Borel measurable for arbitrary programs, and therefore that it can always be extended to a measurable function on $\mathbb{R}$.

4.2 Proposition Let P be a normal logic program and let T_P be its associated single-step operator. Then T_P is measurable on $(I_P, \sigma(\mathcal{G})) = (I_P, \sigma(Q))$.

Proof: We need to show that for each subbasic set $\mathcal{G}(L)$, we have $T_P^{-1}(\mathcal{G}(L)) \in \sigma(\mathcal{G})$.

First, let $L = A$ be an atom. If A is not the head of any clause in ground(P), then $T_P^{-1}(\mathcal{G}(A)) = \emptyset \in \sigma(\mathcal{G})$. If A is the head of a clause in ground(P), then there are at most countably many clauses

$$A \leftarrow A_{i1}, \ldots, A_{ik_i}, \neg B_{i_1}, \ldots, \neg B_{il_i}$$

in ground(P) with head A, and we obtain

$$T_P^{-1}(\mathcal{G}(A)) = \bigcup_i \mathcal{G}(A_{i1}, \ldots, A_{ik_i}, \neg B_{i_1}, \ldots, \neg B_{il_i})$$

which is indeed in $\sigma(\mathcal{G})$.

Now suppose that $L = \neg A$ is a negative literal. If A is not the head of any clause in ground(P), then $T_P^{-1}(\mathcal{G}(\neg A)) = I_P \in \sigma(\mathcal{G})$. So assume that A is the head of some clause in ground(P). If there is a unit clause with head A, then $T_P^{-1}(\mathcal{G}(\neg A)) = \emptyset \in \sigma(\mathcal{G})$. So assume that none of the clauses in ground(P) with head A is a unit clause. Then there are at most countably many clauses

$$A \leftarrow A_{i1}, \ldots, A_{ik_i}, \neg B_{i_1}, \ldots, \neg B_{il_i}$$

in ground(P) with head A. We then obtain

$$T_P^{-1}(\mathcal{G}(\neg A)) = \bigcap_i \mathcal{G}(\neg A_{i1}) \cup \cdots \cup \mathcal{G}(\neg A_{ik_i}) \cup \mathcal{G}(B_{i_1}) \cup \cdots \cup \mathcal{G}(B_{il_i})$$

which is indeed in $\sigma(\mathcal{G})$. ■

By means of Proposition 4.2, we can now view the operator T_P as a measurable function $\iota(T_P)$ on $\mathcal{C}$ by identifying I_P with $\mathcal{C}$ via the homeomorphism ι. Since $\mathcal{C}$ is measurable as a subset of the real line, this operator can be extended[9] to a measurable function on $\mathbb{R}$ and we can now state our second main theorem.

4.3 Theorem Given any normal logic program P, the associated operator T_P (more precisely $\iota(T_P)$) can be approximated in the manner of Theorem 4.1 by input-output mappings of 3-layer feedforward networks.

In fact, we are able to strengthen this result a bit by giving an explicit extension of T_P to the real line. We define a sequence (T_n) of measurable functions on $\mathbb{R}$ as follows (where $l(x)$ and $u(x)$ are as defined earlier):

$$T_0(x) = \begin{cases} \iota(T_P)(x) & \text{if } x \in \mathcal{C} \\ \iota(T_P)(0) & \text{if } x < 0 \\ \iota(T_P)(1) & \text{if } x > 1 \\ 0 & \text{otherwise} \end{cases}$$

$$T_1(x) = \begin{cases} \iota(T_P)(l(x)) + \frac{\iota(T_P)(u(x)) - \iota(T_P)(l(x))}{u(x) - l(x)} & \text{if } x \in [3^{-1}, 2 \cdot 3^{-1}] \\ 0 & \text{otherwise} \end{cases}$$

$$T_i(x) = \begin{cases} \iota(T_P)(l(x)) + \frac{\iota(T_P)(u(x)) - \iota(T_P)(l(x))}{u(x) - l(x)}(x - l(x)) & \text{if } x \in \bigcup_{k=1}^{2 \cdot 3^{i-2}} [(2k-1)3^{-i}, 2k \cdot 3^{-i}] \\ 0 & \text{otherwise} \end{cases}$$

[9]E.g. as a function $T : \mathbb{R} \to \mathbb{R}$ with $T(x) = \iota(T_P(\iota^{-1}(x)))$ if $x \in \mathcal{C}$ and $T(x) = 0$ otherwise.

for $i \geq 2$. We define the function $T : \mathbb{R} \to \mathbb{R}$ by $T(x) = \sup_i T_i(x)$ and obtain $T(x) = \iota(T_P(x))$ for all $x \in \mathcal{C}$ and $T(\iota(I)) = \iota(T_P(I))$ for all $I \in I_P$. Since all the functions T_i, for $i \geq 1$, are piecewise linear and therefore measurable, the function T is also measurable. Intuitively, T is obtained by a kind of linear interpolation.

If $i : B_P \to \mathbb{N}$ is a bijective mapping, then we can obtain a homeomorphism $\iota : I_P \to \mathcal{C}$ from i as follows: we identify $I \in I_P$ with $x \in \mathcal{C}$ where x written in ternary form has 2 as its $i(A)$th digit (after the decimal point) if $A \in I$, and 0 as its $i(A)$th digit if $A \notin I$. If $I \in I_P$ is finite or cofinite[10], then the sequence of digits of $\iota(I)$ in ternary form is eventually constant 0 (if I finite) or eventually constant 2 (if I cofinite). Thus, each such interpretation is the endpoint of a linear piece of one of the functions T_i, and therefore of T.

4.4 Corollary Given any normal logic program P, its single-step operator T_P (more precisely $\iota(T_P)$) can be approximated by input-output mappings of 3-layer feedforward networks in the following sense: for every $\varepsilon > 0$ and for every $I \in I_P$ which is either finite or cofinite, there exist a 3-layer feedforward network with input-output function f and $x \in [0, 1]$ with $|x - \iota(I)| < \varepsilon$ such that $|\iota(T_P(I)) - f(x)| < \varepsilon$.

Proof: We use a homeomorphism ι which is obtained from a bijective mapping $i : B_P \to \mathbb{N}$ as in the paragraph preceeding the Corollary. We can assume that the measure μ from Theorem 4.1 has the property that $\mu\{[x, x + \varepsilon]\} \leq \varepsilon$ for each $x \in \mathbb{R}$. Let $\varepsilon > 0$ and $I \in I_P$ be finite or cofinite. Then by construction of T there exists an interval $[\iota(I), \iota(I) + \delta]$ with $\delta < \frac{\varepsilon}{2}$ (or analogously $[\iota(I) - \delta, \iota(I)]$) such that T is linear on $[\iota(I), \iota(I)+\delta]$ and $|T(\iota(I))-T(x)| < \frac{\varepsilon}{2}$ for all $x \in [\iota(I), \iota(I)+\delta]$. By Theorem 4.1 and the previous paragraph, there exists a 3-layer feedforward network with input-output function f such that $\varrho_\mu(T, f) < \delta$, that is, $\mu\{x : |T(x) - f(x)| > \delta\} < \delta$. By our condition on μ, there is $x \in [\iota(I), \iota(I) + \delta]$ with $|T(x) - f(x)| \leq \delta < \frac{\varepsilon}{2}$. We can conclude that $|\iota(T_P(I)) - f(x)| = |T(\iota(I)) - f(x)| \leq |T(\iota(I)) - T(x)| + |T(x) - f(x)| < \varepsilon$ as required. ■

It would be of interest to strengthen this approximation for sets other than the finite and cofinite elements of I_P, although it is interesting to note that the finite interpretations correspond to compact elements in the sense of domain theory, see [20].

5 Conclusion

There are two aspects to this work. On the one hand, one can consider the problem of approximating the T_P operator, associated with logic programs P, by means of input-output functions of multi-layer neural networks, as we have done here. This, in detail, involves relating properties of the network to classes of programs for which the approximation is possible. It also involves the consideration of what mathematical notions of approximation are useful and appropriate. Here we have discussed two well-known ones: uniform approximation on compacta, and a notion of approximation closely related to convergence in measure. Both these strands need further investigation, and this paper is an account of our work to date which is at an early stage of development. In the other direction, and we have not discussed this at all here except in passing, is to view logic programs as fundamental and to view the approximation process as a means of giving semantics to neural networks based on the declarative semantics of logic programs. There is considerable point in doing this in that the semantics of logic programming is well understood whilst that of neural networks is not, but is something to be taken up elsewhere.

At the detailed mathematical level, the mapping $P \mapsto T_P$ is not injective. So, although the single-step operator can basically be used to represent a program semantically, different programs may have the same single-step operator. This fine tuning is lost by our representation of logic programs by neural networks. However, passing to classes of programs with the same single-step operator is something that is often done in the literature on semantics and in fact is exactly the notion of subsumption equivalence due to Maher, see [17]. Moreover, there exist uncountably many homeomorphisms $\iota : I_P \to \mathcal{C}$; for example, every bijective mapping from B_P to $\mathbb{N}$ gives rise to such a homeomorphism as observed in the paragraph preceeding Corollary 4.4. So

[10] $I \in I_P$ is cofinite if $B_P \setminus I$ is finite.

there is a lot of flexibility in the choice of ι and therefore in how one embeds I_P in $\mathbb{R}$. The homeomorphism used in [14] employed the quaternary number system.

In [14], as mentioned in the introduction, the neural network obtained by applying the approximation Theorem of Funahashi was cast into a recurrent network which settled down in a unique stable state corresponding to the unique fixed point of the single-step operator of the underlying program P. Strong assumptions had to be placed on P to make this possible: P was required to be acyclic with an injective level mapping. Acyclicity of the program yields the existence of a complete metric on I_P with respect to which its single-step operator is a contraction. For larger classes of programs such metrics are yet unknown, and there are indications that they do not exist. However generalized metric structures on I_P can render T_P a contraction, and these matters are currently under investigation by the authors, see e.g. [9, 11, 12, 19], using various methods including those of many-valued logic.

Acknowledgement
One of the referees of this paper has drawn our attention to two lines of investigation implicit in the work reported in this paper, as follows. First, it would be of interest to make more explicit the process of associating a logic program with a neural network. The second is to examine this process in terms of semantics from both the point of view of logic programming and the point of view of neural networks. These are both ongoing projects of the authors, but we thank the referee for highlighting them and their importance.

References

[1] Apt KR, Pedreschi D. Reasoning about Termination of Pure Prolog Programs. Information and Computation 1993; 106:109-157

[2] Bartle RG. The Elements of Integration. John Wiley & Sons, New York, 1966

[3] Batarekh A, Subrahmanian VS. Topological Model Set Deformations in Logic Programming. Fundamenta Informaticae 1989;12(3):357-400

[4] Blair HA, Dushin F, Jakel DW et al: Continuous Models of Computation for Logic Programs. In: Apt KR, Marek VW, Truszczynski M et al (ed) The Logic Programming Paradigm: A 25 Year Perspective. Springer, Berlin, 1999, pp 231-255

[5] Cavedon L: Continuity, Consistency, and Completeness Properties for Logic Programs. In: Proceedings of the 6th International Conference on Logic Programming, Lisbon, Portugal, 1989, pp 571-581

[6] Clark KL: Negation as Failure. In: Gallaire H, Minker J (ed) Logic and Data Bases. Plenum Press, New York, 1978, pp 293-322

[7] Fitting M. Metric Methods: Three Examples and a Theorem. Journal of Logic Programming 1994;21(3):113-127

[8] Funahashi K. On the Approximate Realization of Continuous Mappings by Neural Networks. Neural Networks 1989;2:183-192

[9] Hitzler P, Seda AK: Acceptable Programs Revisited. In: Etalle S, Smaus J (ed) Proceedings of the Workshop on Verification in Logic Programming, 16th International Conference on Logic Programming (ICLP'99), Las Cruces, New Mexico, November 1999. Elsevier, 1999, pp 1-15 (Electronic Notes in Theoretical Computer Science, Volume 30, No 1)

[10] Hitzler P, Seda AK: Characterizations of Classes of Programs by Three-valued Operators. In: Gelfond M, Leone N, Pfeifer G (eds) Logic Programming and Nonmonotonic Reasoning, Proceedings of the 5th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'99), El Paso, Texas, USA, December 1999. Springer, Berlin, 1999, pp 357-371 (Lecture Notes in Artificial Intelligence No. 1730)

[11] Hitzler P, Seda AK. A Topological View of Acceptability. Preprint, Department of Mathematics, University College, Cork, Ireland, January 2000, pp 1-12

[12] Hitzler P, Seda AK: The Fixed-Point Theorems of Priess-Crampe and Ribenboim in Logic Programming. In: Valuation Theory and its Applications, Proceedings of the 1999 Valuation Theory Conference, University of Saskatchewan in Saskatoon, Canada. Fields Institute Communications Series, American Mathematical Society, pp. 1-17. To appear

[13] Hölldobler S, Kalinke Y: Towards a Massively Parallel Computational Model for Logic Programming. In: Proc. ECAI94 Workshop on Combining Symbolic and Connectionist Processing, ECCAI (1994), pp 68-77

[14] Hölldobler S, Störr H, Kalinke Y. Approximating the Semantics of Logic Programs by Recurrent Neural Networks. Applied Intelligence 1999;11:45-58

[15] Hornik K, Stinchcombe M, White H. Multilayer Feedforward Networks are Universal Approximators. Neural Networks 1989;2:359-366

[16] Lloyd JW. Foundations of Logic Programming. Springer, Berlin, 1988

[17] Maher M: Equivalences of Logic Programs. In: Minker J (ed) Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann Publ. Inc., Los Altos, 1988

[18] Seda AK. Topology and the Semantics of Logic Programs. Fundamenta Informaticae 1995;24:359-386

[19] Seda AK, Hitzler P: Strictly Level-Decreasing Logic Programs. In: Butterfield A, Flynn S (eds) Proceedings of the Second Irish Workshop on Formal Methods (IWFM'98), Cork, 1998. British Computer Society, 1999, pp 1-18 (Electronic Workshops in Computing)

[20] Stoltenberg-Hansen V, Lindström I, Griffor R. Mathematical Theory of Domains. Cambridge University Press, 1994

[21] Willard W. General Topology. Addison-Wesley, Reading MA, 1970