Many-Layered versus Few-Layered Learning

Paul E. Utgoff
David J. Stracuzzi
Richard P. Cochran
Department of Computer Science
140 Governor's Drive University of Massachusetts
Amherst, MA 01003 U.S.A.

Technical Report 01-14
January 9, 2001

Abstract

We discuss a variety of issues related to learning and layered organizations of knowledge. A many-layered approach is contrasted with a few-layered approach. A specific case study in the realm of card stackability illustrates many of the advantages and disadvantages. The general conclusion is that many-layered organizations have more benefits than few-layered organizations.

1 Introduction **1**

2 Case Study: Card Stackability **1**

2.1 Two Target Concepts from Solitaire 2

2.2 A Many-Layered Organization 2

2.3 Few-Layered Organizations 6

2.4 Transfer 7

3 Discussion **8**

1 Introduction

Learning is an ever-present element of intelligent behavior that transpires on a multitude of fronts during the lifetime of an agent. Not all knowledge is immediately accessible to an agent. A five-year old will be receptive to some new ideas, but not to others. The same can be said of a fifty-year old. Knowledge accumulates over a very long period of time. That which is currently known to the agent provides a basis for understanding new knowledge in a succinct enough manner to be grasped and held. We shall refer to this idea of accumulating many layers of knowledge as *many-layered learning*.

In contrast, consider what we shall call *few-layered learning*, in which the agent is constrained to represent its knowledge in at most a handful of layers. How can knowledge accumulate over time? Under the constraint of few layers, the only possibilities are that one or more layers grow laterally, or that there be many disjoint pieces of knowledge. Do either of these few-layered organizations lend themselves to a lifetime of knowledge accumulation? We suspect not, and present a simple case study in which issues of knowledge topology can be illustrated and considered more concretely.

Before embarking on the case study, consider a simple illustration of the organizational tradeoff. Suppose we were to wish to build a Boolean logic circuit patterned by the expression:

```
or( and( or(A,B), or(C,D)),
    and( or(E,F), or(G,H))),
```

where the letters indicate Boolean input values. As written, the corresponding circuit would have three (3) layers of computation, using seven (7) two-input logic gates and fifteen (15) wires, counting inputs and output. In contrast, by distributing the two `and`s, a functionally equivalent logic circuit can be pattered by the expression:

```
or( and(A,C), and(A,D),
    and(B,C), and(B,D),
    and(E,G), and(E,H),
    and(F,G), and(F,H)).
```

The corresponding circuit would have two (2) layers of computation, using eight (8) two-input logic gates, one (1) eight-input gate, and twenty-five (25) wires. The three-layered circuit requires less hardware than the two-layered circuit.

More generally, as one reduces linearly the number of computational layers by elaborating combinations, the width of the computational layers grows exponentially. The shallower circuit will execute somewhat more quickly, but it will consume exponentially more space, and it will take more time to learn as there are more functions (gates) to specify. One must see a greater variety of cases to be able to learn each specialized function, increasing learning-time significantly. These arguments apply equally well to threshold logic. In terms of cognitive economy, we would expect that learning more quickly and consuming exponentially less memory would enhance agent survivability more than a slight improvement in computational speed. Now consider a somewhat richer case study, which will allow us to explore in greater detail these and other issues.

2 Case Study: Card Stackability

We compare several approaches to learning two concepts from the domain of card solitaire. These comparisons illustrate issues related to learning-time and space consumption. Although it is common to compare classification accuracies, we are less concerned with accuracy because we assume that the concepts will be learned ultimately. So, instead of holding the training effort at a single shared level and then measuring accuracy, we shall hold the accuracy at a single shared level and then measure training effort. For our discussion, we are concerned primarily with issues of learning-time and space consumption. After describing the two

target concepts to be learned, we discuss different learned representations of these two concepts. Two are many-layered, and several are few-layered. Issues of task decomposition and transfer are also considered.

2.1 Two Target Concepts from Solitaire

In many forms of card solitaire played with a standard 52-card deck, a card can be placed onto another in two different contexts. The first, called *column_stackable*, pertains to cards that are still in play. A card c_1 can be stacked onto a card c_2 already at the bottom of a column if two conditions hold. First, the color of the suit of card c_1 and the color of the suit of card c_2 must differ, and second, the rank of card c_1 must be exactly one less than that of card c_2. We shall ignore the rules for which cards may be at the head of a column, as they vary in different forms of solitaire and are immaterial here.

The second context, called *bank_stackable*, applies to cards that become out of play upon being stacked onto a bank. A card c_2 that is still in play may be placed onto a card c_1 that is out of play in a bank if two conditions hold. First, the suit of card c_2 and the suit of card c_1 must be identical, and second, the rank of card c_2 must be exactly one more than that of card c_1. Again we shall ignore the rules for which cards may start a bank (typically the aces) as they are unimportant for our study.

Define the target concept *column_stackable* to be the relation that includes those ordered pairs of cards (c_1, c_2) for which it is legal to stack card c_1 onto a card c_2 that is already at the bottom of a column. Also define the target concept *bank_stackable* to be the relation that includes those ordered pairs of cards (c_2, c_1) for which it is legal to stack card c_2 onto a card c_1 that is already in a bank. We shall examine several approaches to learning these target concepts.

Notice that these concepts depend on the properties of each card individually and collectively. The notions of suit, suit_color, rank, suit_colors_differ, rank_successor, and suits_identical are mentioned, and can be considered concepts themselves. For humans, these concepts are not difficult to compute, primarily because the rank, suit, and suit_color are indicated plainly on each card. To make the problem slightly richer, yet nevertheless understandable, suppose that the deck of cards to be used does not have these standard indications. Imagine instead that each of the fifty-two (52) cards has solely one of the integers in the interval $[0, 51]$ indicated, without rank, suit, or color.

Clearly for card value c the rank can be computed as $c \bmod 13$, yielding an integer in the interval $[0, 12]$. The suit can be computed as $c/13$, with the integer division producing an integer value in the interval $[0, 3]$. For the purpose of suit_color, it would be convenient to group suits 0 and 1 into one color, and suits 2 and 3 into the other. Once again however, we shall contrive one last minor difficulty by grouping suits 0 and 2 into one color, and suits 1 and 3 into the other. Our objective has been to fashion a problem that is simple enough to understand in its entirety, yet rich enough to lend itself to many layers of knowledge.

Figure 1 depicts the two target concepts as a matrix. Card c_1 indexes the row, and card c_2 indexes the column. In each cell of the matrix is one of three characters. A letter 'c' indicates that the ordered pair is column_stackable, a letter 'b' shows that the ordered pair is bank_stackable, and a dot means that the ordered pair is either impossible or that it is not a member of either target concept. No ordered pair can be both column_stackable and bank_stackable. One can see that in these two input dimensions, each of the concepts requires some care to specify exactly.

2.2 A Many-Layered Organization

The definitions for column_stackable and bank_stackable make use of auxiliary concepts, as described above. Consider the definitions more formally:

suit(x) = ((x)/13)
rank(x) = ((x) mod 13)

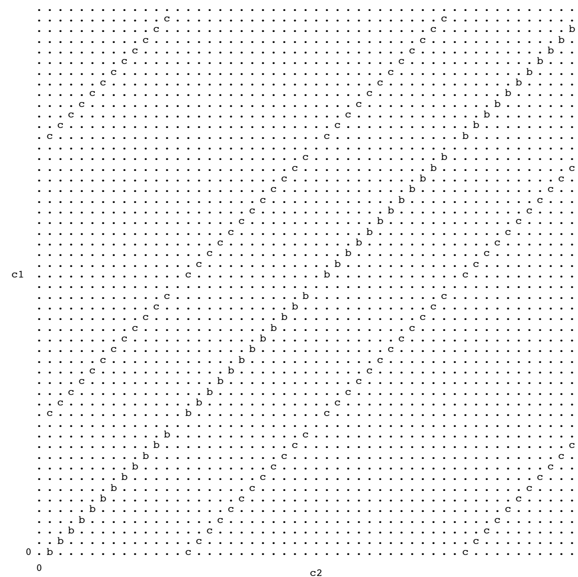

Figure 1. Target Concepts

suit_color(x) = (suit(x) mod 2)
column_stackable(c1,c2) ↔
  (suit_color(c1) ≠ suit_color(c2)) ∧
  (1+rank(c1) = rank(c2))
bank_stackable(c2,c1) ↔
  (suit(c1) = suit(c2)) ∧ (1+rank(c1) = rank(c2))

Rather than explore the intricacies of integer division and modular arithmetic, let us examine instead whether a layered network of simple computing elements can represent the same target concepts. Figure 2 shows such a network, and we shall explain it first, before discussing how it was learned. Notice that there are six layers of computation.

The propositional concepts for each card individually are symmetric. Looking at those for card 'c1', the concept of 'suit' corresponds to fixed sub-intervals of the domain interval $[0,51]$. The 'c1<13', 'c1<26', and 'c1<39' concepts enunciate the critical sub-interval boundaries. With knowledge of these sub-intervals, it is straightforward to compute the suit of c1 by testing whether it falls into a particular sub-interval, but not the next one larger.

From 'suit' and input card value 'c1', one can compute 'rank' from the linear combination 'c1 - 13*suit(c1)'. The weight from each 'suit' unit to the 'rank' unit subtracts the corresponding multiple of 13. The suit colors 'red' and 'black' are simple disjunctions of the relevant suits. The 'suits_black_red', 'suits_red_black', and 'suit_colors_different' units compute an exclusive-or of the suit_color.

The 'rank_successor' concept is somewhat fussy. To test for a difference of exactly one using only inequalities, it is necessary to test simultaneously for whether the difference in rank is at least one (1) and for whether the difference is at most one (1). If each is true, then of course the difference is exactly one (1).

The concept of 'suits_identical' is a disjunction of the four suit-equivalence tests. Finally, 'column_stackable' is the conjunction of the 'suit_colors_differ' and 'rank_successor' concepts, and 'bank_stackable' is the conjunction of the 'suits_identical' and 'rank_successor' concepts.

All of these concepts represent what is known about the ordered pair of cards individually and as a pair. The network computes the sub-interval, suit, rank, and suit_color for each card. It also computes whether

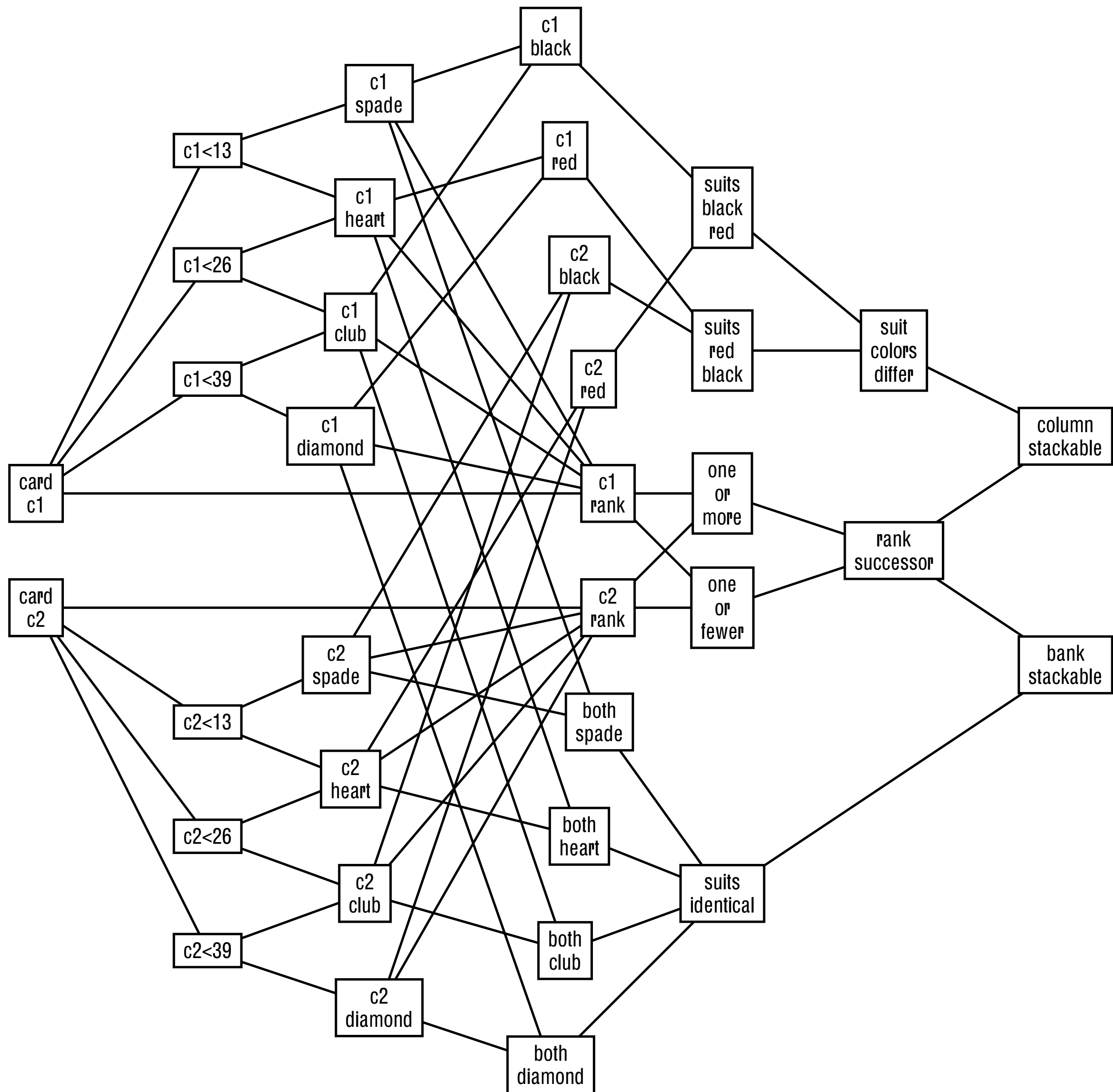

Figure 2. Many-Layered Network

the cards are of the same suit, different suit color, whether c2 is rank successor of c1, whether c2 is bank_stackable onto c1, and whether c1 is column_stackable onto c2.

The network was learned sequentially, one layer at a time. This supposes a good teacher, or a good text, a controlled environment, or an internal task decomposition mechanism (discussed below). All units were implemented as linear hard-threshold units (Nilsson, 1965; Duda & Hart, 1973) except for the 'rank' units, which were unthresholded linear combinations. Although the main goal for the agent is to learn the two concepts regarding stackability, these may be too difficult if earlier learning has not built a satisfactory basis. In this case, it is important to learn first that certain sub-intervals of integer values are important to recognize. From that basis, it becomes much easier for the agent to learn the suit concepts. So it goes, each layer of knowledge making the agent ready to acquire the next.

When the exact dependencies are known, as shown in Figure 2, the layers are learned successively in 3, 2, 300, 3, 2, and 1 epochs respectively. The real-valued rank units required more training than the others. Stepsize was 0.1 for all computational layers except the third, for which is was 0.001. Each epoch consists of training on every possible pair once, sampled in a fixed order. Total cpu time was under 17 seconds on a 733-megahertz Pentium III. When the dependencies are not known, and all inputs or existing units serve as inputs to the new layer of units, the layers are learned successively in 33, 7, 2914, 5, 11, and 18 epochs respectively, in just under 378 seconds. It would be useful to examine each of these problems with respect to sample complexity and learnability. It would be informative to ascertain what accuracy levels short of perfection would be sufficient in order to start learning the next layer. However, these latter issues are peripheral to the discussion here, where our main concerns relate to memory organization.

The approach of allowing all inputs or previously learned concepts to serve as an input basis for subsequent learning has the desirable property of allowing the agent to draw on whatever it already knows in order to understand what is new, to find regularity where it would otherwise be obfuscated. However, an undesirable property of this massive connectivity is an ever-growing dimensionality for subsequent learning, which is difficult to embrace. Methods for learning in the presence of many irrelevant subconcepts (or variables or features) would be especially helpful (Littlestone, 1988), but other mechanisms such as independent tests for correlation with the present targets (for current output layer) may be needed. Though important, this point is beyond the scope of the present discussion.

If we were to know the network topology perfectly ahead of time, as shown in Figure 2, but not the connection weights, could we expect a learning algorithm such as back propagation (Rumelhart & McClelland, 1986; Freeman & Skapura, 1991) to fill in the weights correctly? If not, then the more difficult task of considering full-forward-from-all-previous-layers connectivity (discussed above) would not likely work either. In preparation for two experiments, we replaced all linear-hard-threshold units with linear-soft-threshold (sigmoid) units, and left the two rank units as unthresholded linear combinations.

For the first experiment, we attempted simply to train the network one layer at a time, as before. This presented some unexpected difficulties. The main problem is that with many layers, the output values afforded by soft-threshold units are not as clean as those provided by a hard-threshold unit. Values that may be sufficiently different for immediate discrimination may not be sufficiently different when propagated forward through many layers of computation. The first layer of computation, which is not difficult to learn in isolation, did not achieve sufficient differentiation to function properly in the six layers of computation. It was necessary to initialize the weights cleverly, effectively bypassing learning at that layer. Having done so, the layers were learned successively in 9, 11595, 207016, 16035, 5480, and 3785 epochs respectively. We tried the same scenario with full connectivity (described above) and the training bogged down hopelessly.

A second well-known problem with using soft-threshold units is that the learning algorithm finds a least-mean-squared-error fit of a sigmoid surface to the data. This can serve as a discriminator, but it is quite different from a hard-threshold discriminator, which does not use a continuous error measure. There are linear separable problems that cannot be learned with a soft-threshold unit. This means that one cannot generally expect to be able to trade in hard-threshold units for soft-threshold units and still obtain a satisfactory network. For our particular problem, the one-for-one set of substitutions is acceptable.

For the second experiment, we had planned to test whether the backpropagation algorithm would be able to tune all the weights across the six computational layers, using just the essential connections as shown in Figure 2. We backed off slightly, attempting to learn just column_stackable. This did not work well. After 475,000 epochs (17 cpu hours), the network made 96 errors. This was the same number of errors as after just 1,000 epochs.

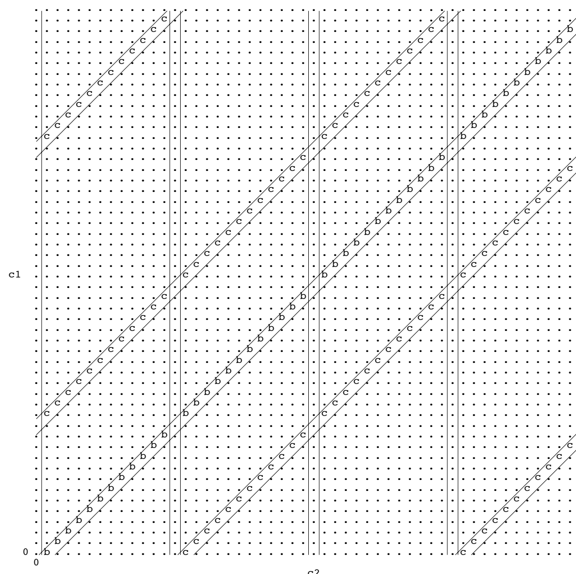

Figure 3. First Layer Decision Boundaries

This is a strawman anyway, because one cannot expect to know the correct connectivity in advance. Given the large training times needed in the forward direction (over five and a half cpu hours) with much richer training signals, and the failure for concept_stackable, we judged that the more difficult training scenarios were doomed. For example, we did not try backpropagation on the fully connected version of the network of six computational layers. In addition to the large times involved, the error surfaces in weight space are likely to contain a plethora of local minima.

2.3 Few-Layered Organizations

What is learned when using just three layers of computation? For these two stackable concepts, we have the luxury of being able to examine the input space, as displayed in Figure 1. Decision boundaries are needed that can be grouped into a conjunction as necessary to describe the twelve (12) regions of interest, which are those that contain positive examples of the targets (letters in the figure), but not negatives (dots in the figure). The first layer of computational units can define the boundaries of interest, and the second layer of computational units can define the regions of interest. The third and final computational layer of target concepts groups the regions of interest into a disjunction as appropriate for each target. Figure 3 shows a set of boundaries that collectively delineate the regions of interest.

We can see that a few-layered representation exists. We implemented a 2-17-12-2 layered network, and applied the backpropagation gradient-descent algorithm. The best network made several hundred classification errors. Note that two target concepts provides two opportunities for error per example. We tried again with a 2-22-16-2 layered network. The best network made 32 errors over all 2704 possible inputs, after approximately two cpu days. Although we were unable to obtain a perfect solution, suppose that we had, and suppose that it corresponded to our expectation as indicated in Figure 3.

With regard to what was learned, the hidden units (neither the inputs nor the output units) describe concepts very much different from those of the many-layered version. It is difficult to conceive their meanings. For example, consider the rising diagonal closest to the lower right of the figure. What set of card pairs does it describe (to its upper left)? There is no explicit notion of suit, or rank, or suit_color, or any of the hidden units of the many-layered network. These are two very different representations of the two target concepts.

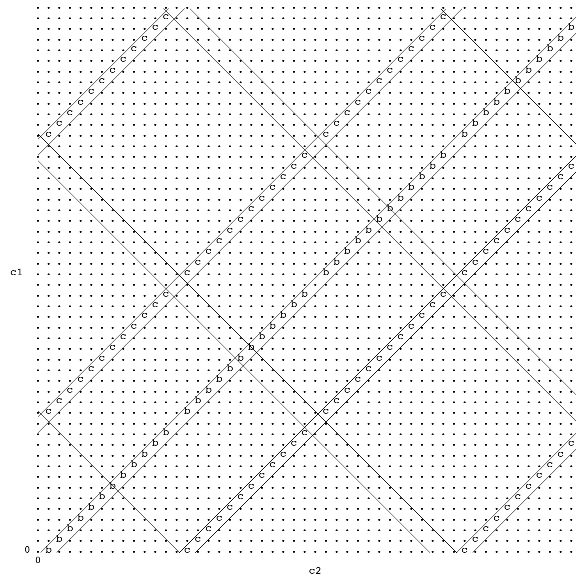

Figure 4. First Layer for Column_Stackable

It can be exciting that backpropagation finds useful weights to specify it computational units. It is intriguing that the subconcepts seem to take form by themselves, and that they do not necessarily match subconcepts familiar to us. The sober view is that the first computational layer finds useful boundaries, the second layer groups them by conjunction into useful individual regions, and that the third layer groups them by disjunction into sets of regions as needed for the targets. The mapping of one representation to the next is limited to these purposes.

2.4 Transfer

To understand better what capabilities the many-layered and few-layered networks offer beyond learnability, consider the important issue of transfer. Instead of learning these two stackable concepts simultaneously, suppose that we were to learn column_stackable first, followed by bank_stackable. In the many-layered network shown in Figure 2, imagine lopping off the bank_stackable unit and recursively any unit so rendered as an output unit. That would excise bank_stackable, suits_identical, both_spades, both_hearts, both_clubs, and both_diamonds. Now, to learn bank_stackable, it would be necessary to learn these six concepts. The modularity of the subconcepts is apparent, making the learning of bank_stackable relatively easy, given the existing knowledge of column_stackable.

For the few-layered case, learning the column_stackable concept first could result in the decision boundaries shown in Figure 4, modeled as a 2-14-8-1 network. These boundaries are not at all useful for the bank_stackable target. To learn bank_stackable would require nine (9) additional units at the first layer of computational units, assuming that the previously learned units for column_stackable were held fixed. It would also require four (4) additional units at the second computational layer, and one (1) additional unit at the output layer for the new target, for a final 2-23-12-2 network. Notice that the smaller 2-17-12-2 network discussed above would be representationally sufficient were the tasks learned simultaneously.

Benefits of simultaneous multitask learning have been explored (Suddarth & Holden, 1991; Caruana, 1997), with the main observation being that hidden units can serve multiple purposes when it is advantageous to do so, improving learning efficiency. Sequential multitask learning is a different matter. For it to work well, the decomposition of the targets into useful subconcepts must take place without knowledge of those learning

tasks yet to be encountered in the future. This suggests strongly that learning simple useful concepts in a many-layered organization can lay the foundation for whatever else may come.

One could propose that a few-layered network could be equipped with a vast store of computational units that are available for any purpose. Sequential learning would simply continue the process of adapting connected computational units as needed to improve the correctness of all target concepts. Although a system could be engineered this way, it would not account for the immediate reuse of previous knowledge on which we all depend. For example, rank_successor is needed for each of the target concepts. When it has been learned for one, it is immediately available for the other.

3 Discussion

There are significant benefits to many-layered learning, including reduced learning-time, exponentially reduced space requirements, and the synergy of transfer. This is so even when using soft-threshold units. The disadvantage is that more than the minimal number of layers of computation may be needed in order to compute a result. One can ponder whether agent survivability depends more on fast learning and strong generalization than on slightly faster mental execution. From a pragmatic engineering view, artificial agents will do better to learn quickly and accurately than to shave microseconds of computation from concept evaluation.

Learning simple concepts can be accomplished quickly. An agent is receptive to those concepts in which regularity is not difficult to identify. However, when the regularity is unfathomable, the agent does not understand what it observes, and the information in the observation is fundamentally lost. Learning of simple concepts can provide a new basis for understanding, in which regularity that was formerly beclouded emerges from the mist. Clark & Thornton (1997) discuss this in terms of Type I and Type II learning. Elman (1993) has suggested that the less developed mental capacity of infants helps learning by admitting only small chunks of knowledge. Simulations with language learning in artificial neural networks indicated that starting with a small network capable of processing short sentence helps to accelerate learning. When development continues, modeled by a larger network, the ability to learn to process longer sentences is considerably enhanced by having already learned to handle short sentences. Starting with the larger network impairs learning.

To learn many layers of knowledge over time may suggest that an agent needs to decompose tasks skillfully on its own. There are two important reasons to believe that task decomposition need not be a sophisticated process left entirely to the agent. First, decomposition occurs naturally, by virtue of receptivity. We will learn next what we are capable of absorbing. Layering of knowledge is a very beneficial artifact of our limited ability to learn new concepts. Our apparent shortcomings may well be the bedrock of our intelligence. Second, humans thrive intellectually when guided well by teachers, well-chosen books, and other stimuli that are 'at the right level'. We should embrace the idea of an agent learning over an extended period of time from curricula designed to match the receptivity of that agent. Long term education of an agent is an essential element of growing that agent's intelligence.

In the case study on card stackability, we found that each of the layers could be learned quickly with linear combination units, hard-thresholded or not as needed. Learning was much more difficult when using soft-thresholds in place of hard-thresholds. We have good theoretical reasons to believe, and growing empirical evidence demonstrating, that backpropagation will not be effective across many layers of computational units. By analogy, one can ask whether a teacher would do well to start the semester with the very last chapter of a text, and to hammer away at it week after week, waiting for all the subconcepts (hidden in the earlier chapters) to form themselves. Educators recommend starting at chapter one.

Acknowledgments

This work was supported by National Science Foundation Grant IRI-9711239.

References

Caruana, R. (1997). Multitask learning. *Machine Learning, 28*, 41-75.

Clark, A., & Thornton, C. (1997). Trading spaces: Computation, representation, and the limits of uninformed learning. *Behavioral and Brain Sciences, 20*, 57-90.

Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis*. New York: Wiley & Sons.

Elman, J. L. (1993). Learning and development in neural networks: The importance of starting small. *Cognition, 48*, 71-99.

Freeman, J. A., & Skapura, D. M. (1991). *Neural networks: Algorithms, applications, and programming techniques*. Addison-Wesley.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm . *Machine Learning, 2*, 285-318.

Nilsson, N. J. (1965). *Learning machines*. New York: McGraw-Hill.

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing*. Cambridge, MA: MIT Press.

Suddarth, S. C., & Holden, A. D. C. (1991). Symbolic-neural systems and the use of hints for developing complex systems. *International Journal of Man-Machine Studies, 35*, 291-311.