A Compositional Neural-network Solution to Prime-number Testing

László Egri (laszlo.egri@mail.mcgill.ca)
School of Computer Science, McGill University, 3480 University Street
Montreal, QC H3A 2B4 Canada

Thomas R. Shultz (thomas.shultz@mcgill.ca)
Department of Psychology and School of Computer Science, McGill University, 1205 Penfield Avenue
Montreal, QC H3A 1B1 Canada

Abstract

A long-standing difficulty for connectionism has been to implement compositionality, the idea of building a knowledge representation out of components such that the meaning arises from the meanings of the individual components and how they are combined. Here we show how a neural-learning algorithm, knowledge-based cascade-correlation (KBCC), creates a compositional representation of the prime-number concept and uses this representation to decide whether its input *n* is a prime number or not. KBCC conformed to a basic prime-number testing algorithm by recruiting source networks representing division by prime numbers in order from smallest to largest prime divisor up to √*n*. KBCC learned how to test prime numbers faster and generalized better to untrained numbers than did similar knowledge-free neural learners. The results demonstrate that neural networks can learn to perform in a compositional manner and underscore the importance of basing learning on existing knowledge.

Keywords: knowledge-based learning; compositionality; KBCC; prime-number testing.

Introduction

Compositionality

Compositionality is the idea of building a problem representation out of components such that the meaning comes from the meanings of the components and the way they are combined (Fodor & Pylyshyn, 1988). In a compositional representation, there is a distinction between structurally atomic and molecular representations. Structurally molecular representations have syntactic constituents that are themselves either structurally molecular or are structurally atomic, and the semantic content of a representation is a function of the semantic contents of its syntactic parts, together with its constituent structure. For example, consider the sentence *The princess kissed the frog.* The meaning of this sentence can be computed from the meanings (semantic content) of the units *the*, *princess*, *kissed* and *frog*, and their positions in the sentence (constituent structure). Changing the word order changes the meaning, so that the sentence *The frog kissed the princess* conveys a different idea.

Compositionality exists in a variety of psychological functions, not only in language (Bregman, 1977). For example, related arguments were made about similarity comparisons sometimes requiring a focus on structural relations between elements rather than mere attributes of the elements being compared (Gentner & Markman, 1993).

A symbolic expression exhibits what is called *concatenative* compositionality, which means that the expression incorporates its constituents without changing them, something that is supposed to be impossible for neural networks (Fodor & Pylyshyn, 1988). In response to this challenge, it has been argued that current neural networks exhibit a unique *functional* form of compositionality that may be able to model the compositional character of cognition even if the constituents are altered when composed into a complex expression (van Gelder, 1990). Perhaps the original constituents could be retrieved from a complex expression by natural connectionist means. For example, an *encoder* network can learn to encode simple syntactic trees on distributed hidden unit representations and then decode them back into the same syntactic trees at the outputs (Pollack, 1990). We argue later in this paper that a newer connectionist algorithm implementing knowledge-based learning can exhibit *concatenative* compositionality.

Prime numbers

One area in which to study the issue of compositionality is that of testing for prime numbers. Is an integer *n* a prime number or not? A prime number is a natural number that has exactly two different divisors, 1 and itself. A number that has more than two divisors is *composite*. The number 1 is neither composite nor prime.

At first glance, the most intuitive way to test whether *n* is a prime number is by checking whether *n* is divisible by any numbers between 2 and $n - 1$. However, this test can be much more efficient (Nagell, 1951). First, because *n* is divisible by a number in the interval $[2, n - 1]$ if and only if it is divisible by a number in the interval $[2, \sqrt{n}]$[1], it is sufficient to check whether *n* is divisible by any number in that interval $[2, \sqrt{n}]$. Second, if *n* is divisible by a composite number in the interval $[2, \sqrt{n}]$, then *n* is also divisible by all prime factors of that composite number. Therefore, it is sufficient to check whether *n* is divisible by any primes in the interval $[2, \sqrt{n}]$. For example, 105 is a composite number because it is divisible by 3, a prime number in the interval [2, 10].

In short, to test whether a number *n* is a prime number, one could divide *n* by all the primes in increasing order, up

[1] Throughout this paper, √n denotes the integer part of a real number, e.g., 2.3 is interpreted as 2.

to the integer part of the square root of *n*, stopping when a successful divisor is found. Thus, the statement that *n is prime* can be represented in a compositional fashion by the Boolean expression: ¬(*n* is divisible by 2) ∧ ¬(*n* is divisible by 3) ∧ … ∧ ¬(*n* is divisible by √*n*). In this representation, propositions of the form *n is divisible by x* correspond to syntactic units. These syntactic units have their own semantic content (i.e., a number *n* is divisible by number *x*). The semantic content of the overall expression is a function of the syntactic units, together with the constituent structure. The constituent structure is expressed by the Boolean operators *not* and *and*.

To demonstrate that constituent structure plays an essential role in the semantic content of the overall expression, consider a different Boolean expression: (*n* is divisible by 2) ∧ ¬(*n* is divisible by 3) ∧ (*n* is divisible by 5) ∧ ¬(*n* is divisible by 7) …. The syntactic parts are exactly the same as before but the constituent structure is different. Accordingly, the meaning of the overall expression has changed. Here the expression means that *n is divisible by prime numbers in odd positions, but is not divisible by prime numbers in even positions in the list of prime numbers*.

So far the only psychological evidence on how ordinary people test prime numbers comes from continued educational use of the ancient *sieve of Eratosthenes* (circa 200 BC). In number theory, a sieve is a process of successively eliminating members of a list according to a set of rules such that only some members remain. With Eratosthenes' method, all integers from 1 to *n* are inscribed on paper. Because 1 is not a prime, it is crossed out. The smallest remaining number, 2 is a prime number. All other multiples of 2 are crossed out because they are composites. The next smallest remaining number 3 is the next prime number. All other multiples of 3 are similarly crossed out because they are composite numbers. This process is repeated until the next divisor is greater than √*n*, by which time all the composites have been crossed out. Numbers not crossed out comprise all the primes from 2 to *n*. An interesting feature of this method is that the smaller the prime number, the more composite numbers it eliminates.

Cascade-correlation

Cascade-correlation (CC) is a neural-network learning algorithm that constructs a feed-forward network as it learns (Fahlman & Lebiere, 1990). A CC network initially has no hidden units; there is only a randomly-determined direct connection from each input to each output unit. Throughout this paper, whenever a connection weight is randomized, it is set to a random value in the range [-0.5, 0.5] in a uniform distribution.

All the networks in this paper are built out of sigmoid units with an offset of -0.5 to center activation around 0:

$$y_j = \frac{1}{1+e^{-x_j}} - 0.5 \qquad \text{Equation 1}$$

where x_j is net input to unit *j*, *e* is the base of the natural logarithm, and y_j is resulting activation value of unit *j*.

Net input *x* to a unit *j* is computed as a sum of products of sending-unit activations y_i and connection weights w_{ij}:

$$x_j = \sum_i w_{ij} y_i \qquad \text{Equation 2}$$

CC learning begins with reducing network error by training weights feeding the output layer. Training these output weights is the so-called *output* phase of CC. Any single-layer network learning algorithm could be used to train output weights but the usual choice is the *Quickprop* algorithm (Fahlman, 1988). Quickprop works like ordinary back-propagation learning except that it employs the second (curvature), as well as the first (slope), derivative of the error surface to adjust connection weights, thus enabling it to reduce error more aggressively (Shultz, 2003). In Quickprop, weight change is proportional to the negative of slope (the rate at which error changes with a change in weight) divided by curvature (the rate at which slope changes with a change in weight).

When output training ceases to reduce error or if error is still too high after a specified number of training cycles (epochs), training shifts to *input* phase. In input phase, a single new hidden unit is trained and added to the network. After the new unit is added, its input weights are frozen and a new output phase begins. This two-phase cycle continues until error is reduced to a satisfactory level, meaning that all output activations are within a specified range of their target values, known as the *score-threshold*.

In input phase, candidate hidden units receiving incoming connections from input units and from pre-existing hidden units are trained. In this phase, candidate output is not connected to any other unit. The incoming weights of the candidate units are initialized with small random weights. Next, the set of training examples is cycled through (a single epoch) and the weights feeding the candidates are adjusted with Quickprop in order to increase *S*, the magnitude of a modified correlation between a candidate unit's activation and network error observed at the output units. Input phase ends when *S* values cease to improve. At this point, the candidate unit with the highest absolute correlation is connected to all the output units with small random weights and a new output phase begins. Usually eight candidates are trained in parallel, the best one is installed, and the rest of the candidates are discarded.

The original version of CC is the one just described and it is sometimes called *deep* CC because each new hidden unit is installed on its own separate layer, creating a fairly deep network topology. However, the version used here is sibling-descendant CC (SDCC). In SDCC, two types of candidate units are trained. Four of the eight candidate units are trained as *descendant* units like in deep CC (Baluja & Fahlman, 1994). The other four candidate units are trained as *sibling* units without receiving any connections from the previous layer. Thus, when a sibling unit is installed, network depth does not increase.

Sibling and descendant candidate hidden units compete with each other for being recruited. The *S* values of descendant candidates are typically multiplied by 0.8 to avoid overly-deep networks resulting from a natural bias to recruit the more complex descendant candidates, having extra weights from the current highest layer of hidden units.

This 0.8 multiplier has been observed to reduce network depth without harming network generalization (Baluja & Fahlman, 1994).

A resulting SDCC network then has multiple layers with one or more units in each hidden layer. SDCC networks have been found to perform similarly to CC networks in psychology simulations but with fewer layers and more interesting varieties of network topology (Shultz, 2006).

Knowledge-based cascade-correlation

Knowledge-based cascade-correlation (KBCC) is a natural extension of CC in which the target network can recruit not only single hidden units, but also previously-learned networks with possibly multiple inputs and outputs (Shultz & Rivest, 2001). We refer to previously-learned networks in the candidate pool as *source* networks and to networks created by KBCC as *target* networks.

Source networks can be installed indirectly or directly. When they are installed indirectly, each input of the source network has an incoming connection from all inputs of the target network, from all outputs of previously installed unit(s) and network(s), and from the bias unit. If a hidden unit or network was installed as a sibling, there is no connection from the current highest layer of hidden units.

When source networks are installed directly, weights connecting the corresponding inputs of the target network and the source network are initialized with weights of 1.0 and the other connection weights are initialized with weights of 0.0. This direct connection scheme implies that the number of inputs of the target and source networks must be the same. Direct connection enables the use of relevant knowledge without much additional training.

In general, all networks have a bias unit sending a trainable connection to every other unit and recruited network except for the input units. The bias unit has a constant activation value of 1.0. All connection weights are initialized with random values.

An example KBCC network from the present project is portrayed in Figure 1. This network has a bias unit, nine input units, and one output unit. It recruited six source networks, installing all of them as siblings on the same layer. Further details about KBCC can be found elsewhere (Shultz & Rivest, 2001).

A major advantage of KBCC over previous knowledge-based learning algorithms is that, with indirect connections, KBCC can recruit any function that predicts network error, without regard to matching the number and function of input units between the source and target networks.

As noted, a long-standing difficulty for connectionism has been to implement compositionality. A goal of this paper is to demonstrate that KBCC creates a compositional representation of the prime-number concept and uses this representation to decide whether its input is a prime number or not. We hypothesized that learning a compositional representation is faster, results in better generalization to input numbers not used in training, and requires less recruitment relative to knowledge-free SDCC learning.

Some evidence already exists for compositionality in KBCC solutions. In a task requiring a network to distinguish points inside a two-dimensional shape from points outside the shape, KBCC recruited source networks representing a vertical rectangle and a horizontal rectangle to learn a target cross shape (Shultz & Rivest, 2001). Also KBCC recruited source knowledge of small checkerboard shapes to learn target patterns of larger checkerboard shapes and source knowledge of small parity problems to learn target problems of larger parity problems (Rivest & Shultz, 2004). Knowledge-based learning was considerably faster than learning without knowledge, and faster with relevant than with irrelevant knowledge.

Method

Primality testing

Given an input number in the range [21, 360], a target network had to indicate whether the input was prime or composite. There were nine input units because the largest number 360 requires nine digits in binary format. There was one binary output unit. A target output value of 0.5 indicated that the input was prime, and -0.5 indicated that the input was composite. The score-threshold was 0.4. Thirty-four randomly-chosen problem instances not used in training (10% of the total) were used to test generalization ability of the networks.

Source networks

Nineteen SDCC source networks were trained in the following way. The input to each source network was a number encoded in binary in the range [2, 360]. The binary digit 1 was encoded as 0.5, and the binary digit 0 as -0.5. Nine input units were needed because the largest number 360 has nine digits in binary format. Each source network was trained to divide its input by a specific number. There was a network to divide by 2, a network to divide by 3, and so on up to 20. Each source network had one output unit. The target output was 0.5 if the input was divisible by the number the given network was responsible for and otherwise the target output was -0.5. These source networks were trained with a relatively low score-threshold of 0.01 to ensure that they provided a clear output. We refer to a source network that was trained to divide by number x as a *division-by-x* network.

Conditions

There was an experimental and two control conditions. In the experimental condition, 20 sibling-descendant KBCC (SD-KBCC) target networks were trained on primality testing. The set of candidate units and networks SD-KBCC could recruit from is called the candidate pool. The candidate pool here contained the 19 divisor (source) networks described earlier. Divisor networks could be connected only directly, either as a sibling or descendant of the current highest layer. The candidate pool also contained four sigmoid units that likewise could be installed as a sibling and four that could be installed as a descendant.

There were also 32 randomized networks in the candidate pool that were created in the following way. Four trained divisor networks were chosen randomly. The weights in the

networks were reset to random values, keeping only the structures of the divisor networks. In effect, we erased the memory in the source networks, but left all the structure intact. For each randomized network, four copies were created and put in the candidate pool. The first copy was directly connected and the remaining copies were indirectly connected. Each copy could be installed as a sibling or a descendant. The idea of including these randomized networks in the candidate pool was to control for complexity of at least some of the recruits. It might be that KBCC would recruit the most complex source, regardless of whether it contained relevant knowledge. Recruiting trained sources over randomized sources would rule out a mere preference for complexity and demonstrate a preference for relevant knowledge.

In one control condition, 20 SDCC networks were trained on the primality-testing problem. The candidate pool contained four sigmoid units that could be installed as a sibling and four sigmoid units that could be installed as a descendant. Comparison with KBCC networks provided an assessment of the impact of prior knowledge, in this case knowledge of divisibility with divisors between 2 and 20.

In another control condition, randomized KBCC, 20 KBCC networks were trained on the primality-testing problem. Their candidate pool was exactly the same as that in the KBCC condition except that connection weights in the 19 divisor networks were reset to random values. Comparison with ordinary KBCC provided another assessment of the impact of prior knowledge but with an additional control for complexity of the recruits. In this condition, there were networks in the candidate pool that were just as complex as the candidates in the KBCC condition, but they contained no knowledge of divisibility because their connection weights were randomized.

Results

Performance comparisons

Numbers of epochs to learn, recruits, and layers, as well as percent success on generalization were each subjected to a factorial ANOVA with condition as the single factor. Results are presented in Table 1 in terms of F ratios and comparison of means using Tukey's *HSD* test at $p < .05$. KBCC networks learned in fewer epochs than did randomized KBCC networks, which in turn learned faster than knowledge-free SDCC networks. The same pattern held for number of recruits during learning. KBCC and randomized KBCC created shallower networks than did SDCC. Generalization to untrained numbers was better for KBCC than for the other two network types.

Table 1: Comparison of performance means.

	Condition means			$F(2, 57)$
Variable	KBCC	Random	SDCC	
Epochs to learn	335	< 1015	< 1122	347
Recruits	6.00	< 8.60	< 9.95	43.5
Network layers	1.00	≈ 1.25	< 2.35	26.4
Generalization	98.7	> 75.1	≈ 73.1	112

Knowledge-representation analysis

KBCC networks were subjected to further analysis to discover how they managed their successful performance. Almost all (17 / 20) KBCC networks had the same structure and recruited their sources in the same order. Divisor source networks were recruited in the following order: division-by-3, division-by-5, … , division-by-17 (all primes from 3 to 17). There were three exceptions and in those networks, only the last two recruitments differed from this pattern.

Weights feeding the recruited source networks were extracted from the 17 typical KBCC networks. For each source network, weights connecting the input of the target network to the corresponding input of the source network had values close to 1, and all other weights feeding the inputs of the source network had values close to 0.

Weights going to the output units (output weights) of the target network were also extracted from the 17 typical KBCC networks. There was one weight from the bias unit, nine weights from the input units, and six weights from the recruited sources. Because weight patterns of the networks were very similar, output weights were averaged over the 17 typical networks. The mean output weights from the recruited source networks, bias unit, and last input unit are shown in Figure 1.

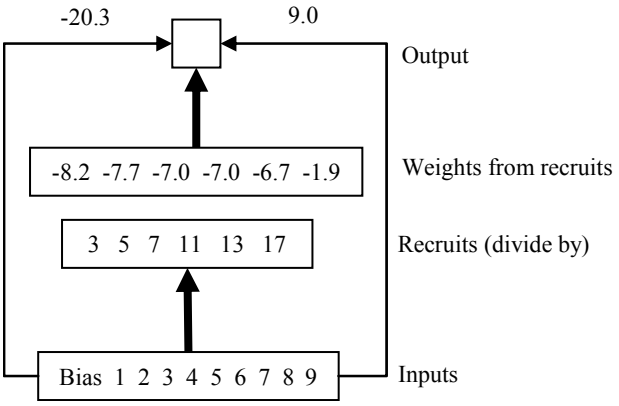

Figure 1: KBCC network with mean output weights from recruited source networks, bias unit, and input 9.

We found that KBCC networks could also be trained and tested on the primality of the same input numbers used in their source networks. If the KBCC networks were trained on numbers in the interval [2, 360], they took longer to learn (M = 437 epochs) and recruited more divisor networks (M = 8.05), but they learned their training patterns perfectly and generalized correctly to 98.2% of their test patterns. The order of their divisor-network recruits was a bit more variable, but still correlated highly with size of divisor (M = .86). All but 3 of these 20 correlations were significantly positive at $p < .05$.

Discussion

Results showed that KBCC networks learned faster and with fewer recruits than did either randomized KBCC networks or knowledge-free SDCC networks. This confirms previous

research showing the superior learning speed of knowledge-based networks (Shultz & Rivest, 2001), and shows that the key advantage of KBCC lies with its knowledge and not the mere complexity of its topology. Randomized KBCC networks of equal complexity as KBCC networks learned faster and with fewer recruits than did SDCC networks, showing that complexity alone has some beneficial effect.

Generalization, often considered to be an essential characteristic of successful learning, was nearly perfect in KBCC networks and considerably better than in either randomized KBCC or SDCC networks. This indicates that knowledge-based learning in KBCC differs not only in speed and efficiency but also in quality of learning.

At first glance, it may seem that randomized-KBCC and SDCC networks generalized rather well on their test problems at 75% and 73% success, respectively. However, because only 19% of the integers in the range [21, 360] are prime numbers, a system can do well (81% success) by guessing that every integer in that range is composite. Thus, learning to test for prime numbers appears to be sufficiently challenging that knowledge-based learning is required for correct generalization. This is the first task we have studied in which CC or SDCC networks could not eventually catch up to the learning quality of KBCC networks. In that sense, the present results underscore the importance of knowledge-based learning. It may be that some tasks can only be learned by building on existing knowledge.

It is interesting that KBCC networks recruited only those six source networks that were trained to divide by the prime numbers from 3 to 17 (3, 5, 7, 11, 13, and 17) and recruited them in that precise order from smallest to largest. This is in accord with the basic primality-testing method discussed in the introduction. KBCC target networks avoided recruiting division-by-composites networks, any networks with divisors greater than $\sqrt{n}$, even if the divisor was a prime number, i.e., 19, and randomized source networks.

The last network recruited was the division-by-17 network. The reason for stopping is that composite numbers less than or equal to 360 always have a prime factor less than or equal to 17. The lowest number that has no factor less than or equal to 17 is 361 = 19 x 19. In other words, the primality-testing problem could be solved without considering any divisors larger than 17.

The knowledge-representation analysis shows how KBCC combined the six syntactic components (i.e., the source networks). First recall that the input to the target network was directly delivered to the source networks with weights of about 1 between corresponding units. With that in mind, let's attempt to interpret the output weights, i.e., weights entering the output unit.

The bias unit has a large negative influence on the output. Assume that the input is a prime number. In that case, all recruited source networks output a value close to -0.5 because the input number is not divisible by any prime numbers between 3 and 17. Given these negative activation values on the outputs of the six recruited networks, because weights from these outputs to the output of the target network have large negative values, the overall effect of the recruited networks on the target network's output is a large positive value (because the product of two negatives is positive). In addition, if the input is a prime number then its last digit is 1, coded here as 0.5. Because the weight from this last input to the output averaged 9.0, this further increases the output value. More precisely, the mean net input to the output unit (excluding the smallish direct input-output connections from inputs 1-8) for a prime-number input can be calculated according to Equation 2 as (1 x -20.3) + (0.5 x 9.0) + (-0.5 x -8.2) + (-0.5 x -7.7) + (-0.5 x -7.0) + (-0.5 x -7.0) + (-0.5 x -6.7) + (-0.5 x -1.9) = 3.45. This net-input value is fed into the sigmoid output unit, according to Equation 1, yielding a value close to 0.5, thus, indicating that the input is a prime number.

Notice that if any of the recruited networks indicate that the input is divisible by some number (meaning that the input number is composite), then at least one of the -0.5 activation values in this computation becomes 0.5. Because all the weights from the source networks to the output of the target network are close to -7, the overall net input decreases by about 3.5 and becomes negative, making the output of the network negative, indicating that the input is a composite number.

Precise activation and weight values do not matter much for this exposition because the weights were averaged over all networks and the small, direct input-output weights from inputs 1-8 were ignored. For present purposes, we are interested only in the average trend of network performance.

There is an exception in the size of the recruit output weights: the weight connecting the output of the division-by-17 network to the output of the target network is -1.9, which is not a value close to -7. This affects only one composite input number, 17 x 17 = 289. It is likely that KBCC networks handled this particular input of 289 by using the direct input-output weights that we excluded from the foregoing computation.

This interpretation of the network structure is in accord with the idea that the internal representation of prime numbers in KBCC networks is computationally equivalent to the Boolean expression: $\neg$(n is divisible by 2) $\wedge$ $\neg$(n is divisible by 3) $\wedge$ … $\wedge$ $\neg$(n is divisible by $\sqrt{n}$). Therefore KBCC represents the prime-number concept in a compositional way.

Astute readers may have noticed one glaring omission from this analysis: KBCC did not ever recruit a division-by-2 source network, even though 2 is a prime number that rules out more composites than any of the larger prime numbers. This does not imply that KBCC networks ignored divisibility of the input number by 2. Instead, KBCC networks noticed that if the last digit of a binary number is 0 (coded here as -0.5), the number is an even number divisible by 2 and thus a composite number. Notice that unlike the small weights from inputs 1-8 to the output unit, this last input 9 has a relatively large weight of about 9 to the output unit. The computation of 9 x -0.5 = -4.5 would drive the net input to the output unit to be negative, thus allowing a conclusion that the input number is a composite.

This provides a simple and ingenious solution of the sort often noted in humans who don't need to explicitly divide by 2 to determine whether a decimal-coded number is odd

or even – they only need to check whether the last digit is odd or even. As another example, a number in decimal form is divisible by 5 if and only if the last decimal digit is either 0 or 5. In this way, humans can easily tell whether a number is divisible by 5 without actually dividing that number by 5. The results show that KBCC networks achieved an effective compositional solution to prime-number testing.

Even more convincing for compositionality would be a demonstration that KBCC networks could learn to do any of several tasks with the same component source networks. This would show that KBCC can build different compositions with the same components to solve different target tasks. We are currently investigating this possibility.

It is important to consider whether the compositional solution achieved by the present KBCC networks is concatenative in the sense that the components of the composition are preserved. Or, is this compositional solution merely functional as in van Gelder's (1990) characterization of Pollack's (1990) recursive networks for encoding and decoding syntactic trees? Based on how the KBCC algorithm works, we would argue that KBCC's solution does indeed implement a genuine concatenative compositionality because, when source networks are recruited, their internal structure and content are preserved intact. Only weights from the inputs of the target-network to the inputs of the recruited network and weights from the outputs of recruited networks are trained.

Besides demonstrating that KBCC networks can learn a compositional procedure for prime-number testing, we wonder whether KBCC could serve to model the psychology of ordinary human performance on this problem. Apart from pedagogical recommendations to use the ancient *sieve of Eratosthenes* in teaching about prime numbers, there is currently little or no psychological analysis to rely on. However, it is interesting that Eratosthenes' method does bear some interesting similarities to the solution learned by KBCC. Both methods order divisors from small to large and use only prime divisors below $\sqrt{n}$. We are currently studying whether people test prime numbers with these constraints.

In conclusion, the KBCC learning algorithm creates a compositional representation of the prime number concept and the resulting network uses this representation to decide whether the input is a prime or composite number. Further, KBCC's compositional representation results in faster learning and better generalization, using less hidden units than control networks without knowledge. The claim that neural networks cannot handle compositionality appears to be incorrect, at least when such networks are allowed to recruit previously-learned knowledge. Have we shown that neural networks can handle all of the compositional tasks that humans are capable of? Absolutely not, but success on even one compositional task shows the claim of impossibility to be incorrect and suggests that neural networks may be able to achieve other kinds of compositionality.

Acknowledgments

This research was supported by a grant from the Natural Sciences and Engineering Research Council of Canada to the second author. We are grateful to Yoshio Takane, J-P. Thivierge, and Frédéric Dandurand for helpful comments.

References

Baluja, S., & Fahlman, S. (1994). *Reducing network depth in the cascade-correlation learning architecture* (Tech. Rep. No. CMU-CS-94-209). Pittsburgh, PA: Carnegie Mellon University, School of Computer Science.

Bregman, A. S. (1977). Perception and behavior as compositions of ideals. *Cognitive Psychology, 9*, 250-292.

Fahlman, S. E. (1988). Faster-learning variations on back-propagation. *Proceedings of the 1988 Connectionist Models Summer School.* Morgan Kaufmann.

Fahlman, S. E., & Lebiere, C. (1990). The cascade-correlation learning architecture. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 2.* Los Altos, CA: Morgan Kaufmann.

Fodor, J. A., & Pylyshyn, Z. W. (1988). Connectionsism and cognitive architecture: A critical analysis. *Cognition 28,* 3-71.

Gentner, D., & Markman, A. B. (1993). Analogy – Watershed or Waterloo? Structural alignment and the development of connectionist models of cognition. In S. J. Hanson, J. D. Cowan, & C. L. Giles (Eds.), *Advances in neural information processing systems, 5*. San Mateo, CA: Kaufmann.

Nagell, T. (1951). *Introduction to number theory*. New York: Wiley.

Pinker, S. (1997). *How the mind works*. New York: Norton.

Pollack, J. (1990). Recursive distributed representations. *Artificial Intelligence, 46*, 77-105.

Rivest, F., & Shultz, T. R. (2004). Compositionality in a knowledge-based constructive learner. *Papers from the 2004 AAAI Fall Symposium, Technical Report FS-04-03*, pp. 54-58. AAAI Press: Menlo Park, CA.

Shultz, T. R. (2003). *Computational developmental psychology*. Cambridge, MA: MIT Press.

Shultz, T. R. (2006). Constructive learning in the modeling of psychological development. In Y. Munakata & M. H. Johnson (Eds.), *Processes of change in brain and cognitive development: Attention and performance XXI*. Oxford: Oxford University Press.

Shultz, T. R., & Rivest, F. (2001). Knowledge-based cascade-correlation: Using knowledge to speed learning. *Connection Science, 13,* 43-72.

van Gelder, T. (1990). Compositionality: A connectionist variation on a classical theme. *Cognitive Science, 14*, 355-364.