In their seminal paper Fodor and Pylyshyn (1988) argue that Classical machines but not Connectionist machines support productive capacities. The argument has been frequently repeated, even in journals of connectionist theory (Schwarz 1992, Van der Velde 1995). The argument can be summarized as follows:

*In a system such as a Turing machine, where the length of the tape is not fixed in advance, changes in the amount of available memory can be affected without changing the computational structure of the machine; viz. by making more tape available. By contrast, in a finite state automaton or a Connectionist machine, adding to the memory (e.g. by adding units to the network) alters the connectivity relations among nodes and thus does affect the machine’s computational structure. Connectionist cognitive architectures, by their very nature, ... cannot support productive cognitive capacities (Fodor and Pylyshyn, 1988, pp. 34-35).*

I argue here that this argument is fallacious. Classical machines do no better than connectionist machines in respect to productivity.
“Productivity refers to the ability to produce and entertain an unbounded set of novel propositions with finite means” (Richard L. Lewis, The MIT Encyclopedia of Cognitive Science, 1999). The most familiar example of productivity in cognitive science is found in the Chomskian conception of universal grammar. There, productivity (or ‘generativity’, as it is often called in that context) refers to the capacity of an adequate grammatical theory to characterize the infinite number of grammatical sentence types in a finite number of (finite) grammatical rules. Chomsky (1965, p. 5) borrows the conception from recursion theory (he explicitly refers to Post, 1936, who introduced the notion of a productive machine) and linguistic theory (Humboldt). Chomsky also declares that “linguistic theory is mentalistic” (p. 4), which implies that an ideal speaker-hearer has a productive linguistic capacity, in the sense that it is possible in principle to understand and generate any of these infinite number of sentence types. The expression “productivity of thought” refers, in general, to the human ability to entertain an infinite number of representation types such as “a (potentially) infinite set of - for example - belief-state types” (Fodor 1987, p. 147). Claims have also been frequently made about productivity – of both thought and machines – in the context of arithmetical operations such as addition (Kripke 1982, Fodor 1990, Schwarz 1992). The claims here are that humans and machines do (or do not) have the capacity to produce m+n for any m and n. ‘Productivity’ here refers both to the capacity to entertain infinitely many types of (codes of) numbers m and n, and to capacity to generate or produce infinitely many types of (codes of) numbers m+n.

Claims about productive capacities are always qualified with terms such as ‘potential’, ‘ideal’ and ‘in principle’, and ‘infinite’ often turns into ‘unbounded’. All emphasize that humans and physical machines do not actually entertain infinite number of types of expressions, thoughts, and representations. Ideal humans just have potential capacities to entertain an unbounded number of types of expressions. All in all, then, we can view the claim that there are productive cognitive capacities as the statement that cognitive and other physical systems are capable of entertaining an unbounded number of types of expressions (thoughts, etc) in finite means. This ability need not be actual, however – it can be potential or ideal.

The term ‘finite means’ is more ambiguous. One difficulty is that it seems that the ability to entertain or produce an unbounded number of types surely requires some unbounded means. Thus ‘finite means’ really means that only some of the means are finite. It usually means that the “rules” followed by humans and machines are finite. The other difficulty is that the term ‘finite rule’ is itself ambiguous. It may turn out that a rule is finite in the sense that it can be formulated in finitely many symbols, but its execution
requires other unbounded means. I will claim that this is the case with Turing machines. It is also important to note that the requirement for finite means is motivated by the assumption that only finite means are physically realizable. After all, we would surely prefer a system that supports productive capacities via means that are infinite but physically realizable to a system that supports productive capacities via finite means that are not physically realizable. We will therefore pay special attention to the possibility of means that are physically realizable.

There are two immediate objections to F&P’s argument. One is that it is unsound because humans have no ability to entertain and produce unbounded expressions. This may be so, but I am less concerned here with whether there are productive cognitive capacities. My main interest is in the relations between Classical and Connectionist machines and productive capacities in general. My aim is to show that, contrary to F&P, Connectionist machines do no worse than Classical machines in supporting productive capacities.

The other objection is that, contrary to F&P, finite-state automata and Connectionist machines support some productive capacities. It is well known, for example, that finite-state automata can compute infinite functions (i.e., functions defined over infinitely many arguments), including addition, that do not require storage memory. I therefore assume that what F&P mean by ‘productivity’ is the ability to support more complex functions, and perhaps even a universal ability to support any “computable” function. The thesis of this paper is that Connectionist architectures do no worse than Classical architectures in supporting any of these functions.

CLASSICAL MACHINES

The term ‘Classical’ was coined by F&P (1988) to refer to systems with the following architectural properties: (a) there is a functional distinction between the program-part where the rules of operations are implemented and the memory part(s) where items are stored; (b) the rules of operations are defined over symbols with combinatorial syntax and semantics; and (c) the rules of operations are sensitive to the syntactic structure of the representations (symbols) over which they are defined.

A paradigm example of a combinatorial representational system is the decimal system whose members represent numbers. A decimal is a string of one or more primitive symbols that are composed according to a finite number of production rules, where the interpretation of each string is a function of the interpretation of its constituents and of its
syntactic structure. Thus the decimal system is defined by a finite number of finite construction and interpretation rules, but it encodes an infinite number of types\(^1\).

A paradigm example of a program/memory system is a Turing-machine (TM). In TM, the memory tape is divided into squares. Each square may contain a symbol. The tape is finite but can be extended indefinitely from both ends. The program-part consists of a finite number of states and a read/write mechanism that reads symbols from and writes symbols on the tape. At every stage, the program part is located “above” one of the squares. Its operation at each stage is determined by its current state and the symbol just read. The options are: to replace the symbol by another, to move one square to the left or to the right, or to change its current state. It is well known that every effectively computable function can be computed by some TM (if the Church-Turing thesis is true), and that there is a universal TM that can compute any of these functions. A Classical machine, then, is a physical system that implements TM or some similar abstract structure.

Let me highlight the salient features of TM that pertain to F&P’s argument. First, the machine reduces computation to a stepwise process, where each step requires only finitely many resources. Moreover, there are only finitely many types of steps a TM can perform. In each step of computing addition, for example, the finite program-part identifies one token of a digit – of which there are only ten types – and then performs any of the other above-mentioned operations, such as replacing one digit by another. Each operation obviously requires only finite resources. This achievement is a result of the fine matching between the combinatorial definition of the decimals – that each decimal is a composition of simple types of decimals (digits) – and the rules of operations that employ this combinatorial structure by defining each simple operation over the digits. But since there are only finitely many digits, each step in the computation requires only finitely many resources.

Second, the process of computing the (defined) values of a computable function requires only finitely many steps. In particular, since each decimal is, by definition, a finite string of tokens of digits, the process of computing the sum of \(m\) and \(n\) for any two concrete input-strings \(m\) and \(n\) requires finitely many steps. But each step, as we have just seen, is finite, and hence the process of stepwise computation as a whole is also

\(^1\) One way to define the decimals is with the following rules of syntax:

1. ‘0’, ‘1’, … ‘9’ are primitive strings (digits).
2. If \(T\) is a string, and \(x\) is a digit, then \(Tx\) is a string.

The rules of semantics are then:

1. \(I(‘0’) = 0; I(‘1’) = 1; \ldots; I(‘9’) = 9.\)
2. \(I(Tx) = 10 \times I(T) + I(x).\)
finite. In other words, producing a decimal m+n for input decimals m and n is a *finite* process for *any* m and n. The process consists of finitely many steps, and each step requires only finite means. We have to be a bit careful here. It *does not follow* from the last observation that TM employs only finite resources to produce m+n for any m and n. Producing m+n for one pair of m and n is a finite process for any pair (m,n). Still, there are infinitely many decimals – some of which are very long strings. It is therefore possible that TM actually needs unbounded resources to produce m+n to *any* of these infinitely many types of strings. It is indeed agreed that TM for addition requires unbounded memory.

Another salient feature of TM, one that is central to F&P’s argument, is that memory can be indefinitely extended without altering the finite program-part. This feature is just another description of the functional distinction between program and memory. It means that we can always extend the memory tape in order to entertain longer decimals m and n, even though it is the very same program part that is used to produce m+n. A central question of this paper is what can we conclude from this feature. F&P infer that unbounded memory is the only unbounded means a TM employs to support productive capacities. But I will contend that the inference and its conclusion are invalid.

**CONNECTIONIST MACHINES**

A Connectionist architecture consists of processors called “neurons” with each neuron connected to other neurons. A neuron is an entity whose output (activation value aᵢ) is a function of the values it receives from other neurons, whereas these values are affine combinations of the activation values of the other neurons (aⱼ) and the weights of the connections (wᵢⱼ). Such a combination is given by the term \(\sum aⱼwᵢⱼ\). The neurons and the connections between them can be seen as a directed graph, usually called “a neural network”. A Connectionist machine is a physical system that implements a neural network.

In their argument, F&P clearly have in mind neural networks whose activation values are binary (or at least bounded). These nets, as F&P argue, have to be expanded in order to entertain and produce an unbounded number of representation-types\(^2\). But not all nets are like this. There are also nets whose activation values are continuous or rational. Most interestingly, Siegelmann and Sontag (1995) proved that the computational powers of

---

\(^2\) A nice example of such a net is a self-inhibiting attractor neural net for the N-queens problem (Shagrir 1992). That is, for any given n, there is a n×n net that stabilizes on a solution for any arbitrary initial conditions. The weights of the net are determined by the same set of equations for any n.
TM are equivalent to the computational powers of finite nets with fixed rational values. The main results are that for every Turing-computable function there is a rational value net that computes the values of this function, and that there is a finite net, with less than 900 neurons, which is as powerful as a universal TM. The main features of such a net are:

(a) The net consists of finitely many neurons and weights. The activation value of the neuron is given by a linear function. The weights of the net are fixed in advance to rational values in [0,1]. The values are simple in the sense that they are not “too close” to each other, and so can be easily implemented.

(b) There are designated input unit(s) and output unit(s). The net can entertain and produce inputs and outputs in two ways. It can entertain and produce binary (or other symbolic) streams, meaning that at each time the input unit entertains or the output unit produces at most one symbol. The other way is to entertain and to produce rational values. In this case, we feed the input unit(s), at one time, with rational value(s), and the net produces, at the output unit(s), another rational value(s). This net is called a net without I/O.

(c) There is no need to expand the net. The very same net can entertain and produce an unbounded number of inputs and outputs. The net that computes addition, for example, produces m+n for any inputs m and n. No changes are made for large m and n. The relevant memory can simply be stored in the rational activation values, of which there are infinitely many.

(d) A net without I/O computes addition – or any other computable function – in bounded real time. That is, it takes the same time to compute 6+3 and 3456+6789, assuming that it takes one real time step to produce the activation value of a neuron, \( f(\sum_{a_i(t)}w_{ij}) \), where \( f \) is the above mentioned linear function.

(e) Though the net has finitely many neurons and connections, it requires the unbounded ability to support and discriminate between very close rationals. If, for example, we encode each natural number n by an activation value 1/n, the net must be able to produce precise activation values for any n, as well as to distinguish between 1/n and 1/(n+1).

In what follows I refer to nets without I/O (henceforth, NN). NN is different from TM in that its operations are not necessarily sensitive to the constituents of the pertinent representations, if there are any. In contrast to TM, NN may token a representation of ‘11’ without tokening ‘1’. The advantage is that NN need not store and process arbitrarily long strings (features c and d). The disadvantage is that NN still has to encode and discriminate between an unbounded number of representation types (feature e). How crucial are these differences between TM and NN in respect to productivity? F&P argue that they are very crucial. Let us return to their argument.
F&amp;P’s argument can be spelled out as follows: It is agreed that both (abstract) TM and NN can entertain and produce an unbounded number of representation types. It is also agreed that both machines employ unbounded means to support this capacity. TM deploys unbounded memory, whereas NN requires unbounded discriminative powers. It may also be true that these means are not physically realizable. But there is an important difference. The computational structure of TM is identified with its rule of operation, namely its program part, whereas the computational structure of NN is constituted in the neurons and connection weights. Thus, when we add more memory to the TM we do not alter its computational structure, as the program part remains the same. We can therefore say that we are talking about the same computational machine. By contrast, when we add more neurons and connections to NN we change its computational structure. We therefore cannot even say that we are talking about the same computational machine.

This difference in abstract systems is reflected in the implementing physical machines via the distinction between competence and performance. The distinction has been forcefully employed by Chomsky (1965, pp. 3-5) to emphasize that hearers/speakers have the competence to hear/understand arbitrarily long expressions even if they do not actually perform well when the expressions are too long. In our context, the claim is that even though the actual performance of both physical Classical and Connectionist machines is limited to a finite number of types of representations, the Classical machine, but not the Connectionist machine, has the competence to entertain and produce unbounded number of representation-types. We can summarize the argument as follows:

(a) The competence of a Classical machine is associated with its program part, where the rule (program) for addition is implemented. Performance also depends on the size of memory (which is external to the program): “memory is external to the program, so that the memory capacity can be increased without changing the program, that is without changing the computed function itself” (Van der Velde, p. 250).

(b) The competence of a Classical machine is therefore grounded in the operation of the program part under ideal conditions: “Competence simply is performance under ideal processing conditions; i.e. without memory constraints. Competence will differ from performance when the system does not realize the required amount of memory” (Schwarz, p. 215).

(c) If so, the potential performance of a Classical machine (i.e., its competence) is revealed when the working memory is unbounded. Under these conditions, the machine will produce m+n for any m and n. “Thus, the infinity of these machines is not actual but a potential infinity” (Van der Velde, p. 250).
(d) In Connectionist machines, however, there is no distinction between competence and performance because there is no distinction between program and memory or “between the available knowledge base and the available computational resources.” (Fodor and Pylyshyn, p. 34). In connectionist systems, it is not possible to extend the memory of the machine without changing its computational structure. You cannot change the performance of a net without changing its competence. Thus, the competence of any Connectionist machine is just its actual performance. However, the actual performance of any Connectionist machine falls short of entertaining and supporting an unbounded number of representation types. Connectionist machines are therefore not competent to entertain and produce an unbounded number of representation types. Thus these machines cannot support productive capacities.

In the next section I argue: (1) that F&P’s argument overestimates the ability of Classical machines to support unbounded capacities in finite means, (2) that it underestimates the abilities of Connectionist machines to entertain and produce unbounded number of representation types, and (3) that, all in all, it fails to show that Connectionist machines do worse than Classical machines in supporting productive capacities.

ASSESSING FODOR AND PYLYSHYN’S ARGUMENT

My first thesis is that F&P are wrong to think that TM employs only finite means in supporting unbounded capacities. The problem is not just with its memory. I grant F&P that memory can be indefinitely extended without altering the computational structure of TM. The main difficulty is rather with the program part. The difficulty is that the finite program part requires *unbounded iterative powers*. Since the number of steps required to produce m+n for all decimals m and n is unbounded, the program part must have the ability to iterate its finite number of states an unbounded number of times. Assume, for example, that the program part possesses the state q₂, which is tokened whenever ‘1’ occurs in the string. This requires the tokening of q₂ not only in the cases where the machine encounters the single string ‘1’, but also in the cases where the machines encounters any string that contains ‘1’ (e.g., ‘10’). But this requires the tokening of q₂ over and over again in the cases where the machine encounters strings that consist of many ‘1’s. It requires the tokening of q₂ an unbounded number of times. It thus seems that TMs require two kinds of “unbounded means”. In addition to unbounded memory, they require the capacity to token each state of the program part an unbounded number of times. The second capacity, however, is directly associated with the program part and so with the computational structure of TM. A program part without unbounded iterative powers is unable to entertain and support an unbounded number of expression-types,
even if memory were unbounded. We must therefore conclude that the competence of TM to handle unbounded expression-types requires infinite means.

Worse still, it seems that Classical machines, as physical machines, do not have unbounded iterative powers. This is because a physical system, unlike an abstract system, is constrained by physical laws. And according to these laws, the physical program part will break down at some point. If the two input-strings are too long, the program part will simply disintegrate before it completes its mission. In an abstract TM the program part works ad infinitum, and its memory tape can be extended indefinitely. Thus there is no dispute that an abstract TM can entertain arbitrarily long strings, and thus can compute addition. The trouble, however, is that no physical machine complies with these conditions. When the two input-strings are too long, the physical system is unable to produce the sum, not only because there is not enough memory to encode the data, but also because the program part will break down after a while. Even if we leave the program part fixed, and just add more memory when needed, the program part is bound to break down at some point. If the strings are too long, the physical program part would simply not produce m+n even if it operates with an unbounded memory. Thus, even if we accept F&P’s assumption that the competence of the Classical machine is identical with its performance using unbounded memory, we still cannot conclude that the system’s competence is computing addition. For it is simply not true that the system would produce m+n for any m and n. Any program part will break down before it completes the addition of two very long strings.

Let us now turn to my second thesis that F&P’s argument does not target NNs. F&P’s argument against Connectionist machine is that the expansion of memory alters the computational structure of the machine. But, as we saw, NN needs no memory expansion. NN can entertain and produce the unbounded number of representation types without expansion. NNs are therefore not threatened by F&P’s argument.

One may argue that a Connectionist machine that implements NN is not physically possible. Georg Schwarz (1992), for example, argues that Connectionist machines cannot compute infinite functions because there are physical constraints on both the actual processing elements of the net and the weights connecting them. These constraints limit the range of the arguments that the system can entertain. The machine performs the task only “if the processing elements can assume infinitely precise activation values. Moreover, infinite-precision activation values can be produced or exploited only if the weights are of infinite precision as well... [But] the primary problem with infinities of any kind is that they preclude physical realization. No real network can... employ infinitely precise activation values (Schwarz, 1992, p. 211).”

I agree that NN requires unbounded discriminative powers. I’m not sure, though, that they are not physically realizable. Of course, it will be a great surprise if it turns out that
there is some hardware that could reliably discriminate rationals in [0,1]. Nonetheless, it is not obvious that physics rules out this possibility. We may one day find physical structures which can spin into an infinite number of directions, or that can get into an infinite number of energy levels. The truism that we currently cannot build such Connectionist systems is no guarantee that these machines are not physically possible.

My main point, however, is that even if NNs are not physically realizable, Connectionist machines do no worse than Classical machines in supporting productive capacities. It is true that an abstract NN requires unbounded discriminative powers. And it may be true that NN is not physically realizable. But it is also true that an abstract TM requires not only unbounded memory, but also unbounded computational means, that is, unbounded iterative powers. And it is also true that these computational means are not physically realizable, as their realization requires a perpetuum mobile program part. My third thesis, then, is that F&P’s argument fails. They fail to show that the architectural differences between Classical and Connectionist machines entail that the first but not the second type of machine supports productive capacities.

**BEYOND FODOR AND PYLYSHYN’S ARGUMENT**

Abstract TMs and NNs require unbounded computational means to entertain and produce an unbounded number of representation types. As a result, Classical and Connectionist physical systems may not have productive capacities. But this is not a result we like. We’d like to think that, in some sense, physical systems do have the ability to handle an unbounded number of representation types. Some even think that there is a sense in which Classical, but not Connectionist, machines have the desired ability. In the last section I argued that F&P’s argument fails to demonstrate this. In this section I briefly consider and reject other possible arguments to this effect. My aim here is not to deny that Classical machines support productive capacities in some sense. My claim, rather, is that Connectionist machines do no worse that Classical ones in respect to productivity.

**Argument #1 – competence and performance again**: The simple fact is that by extending the memory, a Classical machine will have more powers. The same machine will entertain and produce m+n for novel m and n, just by adding more memory. But in a Connectionist machine adding more memory alters computational structure. Thus the competence of Classical, but not of Connectionist machines, goes beyond actual performance. It thus makes more sense to idealize to unbounded capacities in the Classical, but not in the Connectionist, case.
Reply: The claim about Connectionist machines here is just false. In Connectionist systems, much as in Classical systems, we can sometime extend performance \textit{without} altering computational structure (i.e., without adding more nodes). The weight-matrix of the Connectionist machines might be able to 10 different values, even if the input-unit can encode at most 2 different numbers. We could therefore say that the competence of this Connectionist machine is different from its actual performance. We could expand the performance of the the machine, by refining the input-unit without altering its computational structure.

Argument #2 – maintaining the program part: A referee suggested that a preemptive maintenance could be added to the Classical machine, or even perhaps a self-maintaining mechanism that replaces one of components of the program part at each step. This program part could then work ad infinitum.

Reply: First, the maintaining mechanism will also break down at some point, at least when the universe comes to an end. Second, the “new” machine is quite different from current Classical machines. Thus current machines are still not truly productive. Third, a Connectionist machine might be able to discriminate between any two rationals if we could only get rid of all interfering noise. This is not less likely than the possibility of a self-maintaining Classical machine.

Argument #3 – ideal laws (Fodor): Cognitive productive capacities are rooted in Classical architectures whose competence is just a matter of psychological-computational ideal law:

\textit{If there are psychological laws that idealize to unbounded working memory, it is not required in order for them to be in scientific good repute that we know all of what would happen if working memory really were unbounded. All we need to know is that, if we did have unbounded memory, then, \textit{ceteris paribus}, we would be able to compute the value of \textit{m+n} for arbitrary \textit{m} and \textit{n}. And that counterfactual the theory itself tells us is true.} (Fodor, 1990, p. 95).

Reply: We could not compute the value of \textit{m+n} for arbitrary \textit{m} and \textit{n} even if the memory were unbounded. Perhaps we could do so if we lived in an ideal environment with no friction, noise, etc. But then the route to ideal laws is open to the Connectionists as well. Connectionists can say, much like Fodor, that our productive capacities are rooted in a Connectionist architecture of NN, whose competence is revealed, as a matter of ideal law, in a noiseless environment where activation and weight values are infinitely precise.
**Argument #4 – implementation:** A Classical machine computes addition and not quaddition by virtue of implementing a program for addition. After all, the program for computing addition is obviously quite different from a program for computing quaddition. Therefore, the physical program part that implements the first program must be very different from one that implements the second program. The first program part really implements a program for addition.

**Reply:** A physical Connectionist system computes addition and not quaddition by virtue of implementing NN for addition. After all, NN for computing addition is very different from NN for computing quaddition. So if the argument works for Classical machines, it also works for Connectionist machines.

**Argument #5: theoretical considerations (F&P):** In a note in their paper, Fodor and Pylyshyn suggest that productivity can be also viewed as no more than a theoretical maxim. They say that there is no need to assume that we possess infinite generative capacities. “Infinite generative capacity can be viewed, instead, as a consequence or a corollary of theories formulated so as to capture the greatest number of generalizations with the fewest independent principles” (1988, p. 33, note 22). Theories of cognition have to generalize over individuals whose actual performance to produce m+n varies quite dramatically. In addition, it is known that individuals can improve their performance by “relaxing time constraints, increasing motivation, or supplying pencil and paper” (p. 34). Thus healthy theoretical principles such as generality, simplicity, and economy motivate us to attribute to individuals the capacity to produce m+n for any m and n.

**Reply:** I have no objection to this way of viewing productivity. My point is that Classical architectures have no monopoly over theoretical principles. There are no theoretical resources available to Classicist which are not also available to the Connectionist in explaining productivity. In particular, simplicity, economy, and generality are not principles associated just with Classical systems. It is obvious they play the same role in Connectionist theories.

In sum, then, there are currently no reasons to think that Classical, but not Connectionist, machines support productive capacities.

**ACKNOWLEDGEMENT**

I am grateful to Daniel Amit, Patricia Churchland, Yuval Dolev, Itamar Pitowsky, Carl Posy, Georg Schwarz, Hava Siegelmann and Tony Travis for corrections, explanations, and insightful suggestions. An early version of the paper was presented in the NTCS 99
conference on computationalism in Vienna. I thank the participants for the long and stimulating discussion.

REFERENCES


Oron Shagrir
shagrir@cc.huji.ac.il

Department of Philosophy
The Hebrew University
Jerusalem, ISRAEL