Universal computation by networks of model cortical columns

Patrick Simen
EECS Dept.
University of Michigan
Ann Arbor, Michigan
Email: psimen@eecs.umich.edu

Thad Polk
and Rick Lewis
Psychology Dept.
University of Michigan
Ann Arbor, Michigan

Eric Freedman
Psychology Dept.
University of Michigan
Flint, Michigan

***Abstract*— We present a model cortical column consisting of recurrently connected, continuous-time sigmoid activation units that provides a building block for neural models of complex cognition. Recent progress with a hybrid neural/symbolic cognitive model of problem-solving [9] prompted us to investigate the adequacy of these columns for the construction of purely neural cognitive models. Here we examine the computational power of networks of columns and show that every Turing machine maps in a straightforward fashion onto such a network. Furthermore, several hierarchical structures composed of columns that are critical in this mapping promise to provide biologically plausible models of timing circuits, gating mechanisms, activation-based short-term memory, and simple if-then rules that will likely be necessary in neural models of higher cognition.**

I. Introduction

Researchers in cognitive neuroscience are increasingly interested in neural models of problem-solving and planning [2], [5], [9]. Characterizing problem-solving and planning as the manipulation of symbols has proven very useful in artificial intelligence and cognitive psychology [8], but it is by no means clear how such symbolic processing is implemented in the brain. Yet this is just the kind of question that interests cognitive neuroscientists.

In [9], we examined the hypothesis that dorsolateral prefrontal cortex in humans subserves activation-based short-term memory for symbolic goals during problem-solving in the Tower of London task, using a hybrid neural/symbolic model. The model uses a system of locally recurrent neural networks to encode symbolic information as stable patterns of reverberating neural activity. The activation V_i of a unit i is a value in $(0, 1)$ governed by a standard sigmoid activation function:

$$\frac{dV_i}{dt} = -V_i + \frac{1}{1 + exp(-\lambda(NetIn_i - \theta_i))} \tag{1}$$

where $NetIn_i = \sum_{j=1}^{n} w_{ij} V_j$, and w_{ij} is the synaptic weight on the connection from unit j to unit i. V_i is taken to model the recent average firing rate of a neuron or a population.

Local networks, considered in isolation from external input, are Hopfield networks with orthogonal memories [4]. They correspond to symbolic variables, and the basins of attraction around the possible stable patterns of activity within a network correspond to the values of that variable. In this respect, the model accords well with the recurrent connectivity seen in cortex [10] and with evidence for short-term memory maintenance through persistent neural firing [3]. We simulated 'productions', – the if-then rules that constitute the basic symbolic processing operations of existing symbolic cognitive architectures – by feedforward connections between these modular networks. Production systems such as ACT [1], EPIC [6] and Soar [7] have been used extensively to model higher cognition due to their ease of programming and psychological plausibility.

In [9], when feedforward connections from the units highly active in pattern u of upstream network U excite the units in downstream network D which are highly active under the symbolic representation d, we say that the system is implementing the rule: *if* $U = u$, *then* $D = d$. This model represents goals as patterns of activity in a network that 'votes' for actions in an action network by exciting those that would help achieve the current goal. However, it relies on non-neural control code that detects approximate convergence to an attractor in the action network as the trigger for initiating a new step of neural processing, which it accomplishes by 'clamping' certain networks to particular values.

In order to show that the basic neural processing mechanisms of [9] can be incorporated into a purely neural system for general symbolic processing, we sought a biologically plausible set of replacements for the non-neural mechanisms it employs. We obtain systems compatible with the principles of [9] that also have the required convergence detection and sequencing functions by employing neural mechanisms at five levels of hierarchical composition. At the lowest level is the firing rate model of a unit or population given in eq. 1. At the next level is a mechanism inspired by a characteristic feature of cortical organization: cortex is organized into vertical columns of interconnected neurons that extend throughout the six or so horizontal layers of cortex [10]. We model columns as structured arrangements of sigmoid units, as shown in fig 1. Columns are themselves composed through lateral, inhibitory connections into modules, which are in many respects identical to the attractor networks used in [9]. Columns turn out to be essential for controlling the rate of signal propagation in order to support sequencing and convergence detection. Modules can themselves be composed with feedforward connections into complete circuits. The highest layer of organizational abstraction allows construction of circuits from characteristic combinations of storage and gating modules. All of these mechanisms are consistent with known cortical organization, since they require only structured vertical arrangements of densely connected neurons with lateral connections to other columns, along with long-distance, vertical projection axons [10].

Here we highlight the key points of a proof that such models allow for general symbolic processing by showing that any Turing machine can be mapped in a straightforward way onto a system of these mechanisms. (We point out that while universal computation is certainly a capability of systems of simpler neurons – e.g., binary threshold units – it is not clear that systems that more closely approximate real neurons have this property.) We can therefore conclude that systems of sufficiently many cortical columns are capable of carrying out any algorithm. Under the Church-Turing thesis, this is all that any physically realizable computational device can be expected to do. In section II we discuss the neural building blocks that handle the shortcomings of [9] and also make the Turing machine mapping possible, and in section III we discuss the mapping itself through the use of a simple example.

II. Additional Mechanisms

The extension to [9] that we propose still relies entirely on the feedforward composition of locally recurrent, modular networks. In the extended architecture, however, a few different classes of modules exist which differ from each other in respect of their strength of internal lateral inhibition, their threshold terms θ_i, and the degree to which they delay propagation of input signals and act as a low-pass filter for them. It is the last two of these properties that rely on the cortical column mechanism.

A. Controllable propagation delay for convergence detection and self-terminating productions

A significant problem with the production system analogue in [9] is that compositions of modules with feedforward connections run into timing difficulties. In [9], the symbolic controller can be removed if the action network has the property that it only sends output to the rest of the system when it has entered some ε-neighborhood of an attractor corresponding to an action, and if the rest of the system can then inactivate the action representation. This is a special case of the following situation.

Consider a module A in which the symbolic value a is represented by high activity in a subset of A's units and low activity in the rest. Call the set of high-activity a units $High_a$. We say that $A = Null$ when none of A's units are highly active.

It is often useful to implement a *self-terminating* production of the form

$$\mathrm{IF}(A = a)\mathrm{THEN}((B = b) \wedge (A = Null)) \qquad (2)$$

using excitatory connections from A to B and inhibitory connections from B to A. Such an arrangement allows A to activate B when some event occurs. A therefore acts as an indicator that some process should take place, and B acts as the process that handles A's event. B eliminates A's event indication but can still maintain the value b indefinitely.

Things do not work as intended with straightforward composition, however, because as $High_a$ in A begins to activate $High_b$ in B, $High_b$ begins to inhibit $High_a$. The result is that the two modules wind up approaching an equilibrium in which $High_b$ never quite becomes active, and $High_a$ never quite shuts off. It can be shown that it is not possible to manipulate the weights between the *if* module A and the *then* module B and the λ and θ_i parameters of the activation function of any of the units involved so that $High_a$ can drive $High_b$ to a value arbitrarily close to 1, and $High_b$ then drives $High_a$ arbitrarily close to 0. This can, however, be accomplished if large changes in $High_a$ take sufficiently long to produce large changes in $High_b$. Two potential sources of propagation delay are: 1) connections between modules with sufficiently slow propagation rates, and 2) chains of units in between A and B that multiply the effect of a single unit's time constants. We chose instead a more economical mechanism with greater flexibility in controlling propagation rate, and we hypothesize that this is a functional role of cortical columns in the brain.

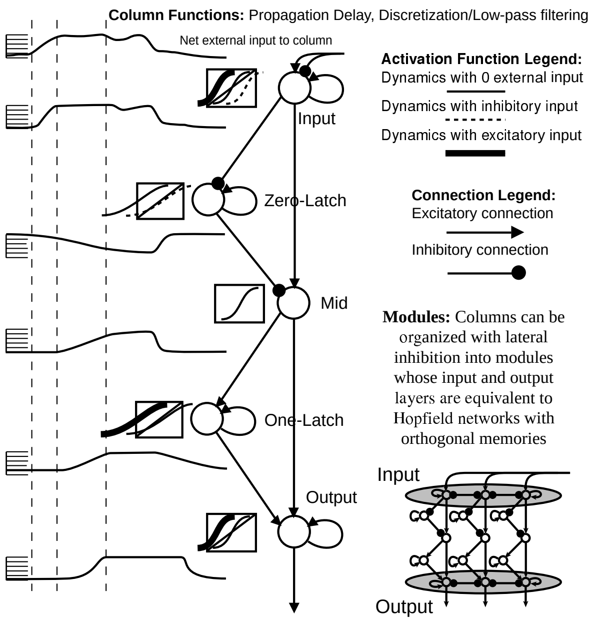
B. Cortical columns

The cortical column-inspired version of a module is schematically depicted in fig 1. Instead of a single, fully connected recurrent network, a module now consists of two identical copies of such a network. One copy functions as the input interface to the module, and the other functions as the output interface. Each input layer unit $Input_i$ sends a feedforward connection to its counterpart $Output_i$ in the output layer via an intermediate unit, Mid_i. The Mid_i unit is inhibited by a self- exciting unit $ZeroLatch_i$ that also receives input from $Input_i$. The $ZeroLatch_i$ unit serves to prevent rapid transmission through Mid_i of large jumps in activation of $Input_i$ when a gain signal to $ZeroLatch_i$ is high. When the gain signal is low, $ZeroLatch_i$ never becomes significantly active, and transmission through Mid_i is as fast as the time constants in the activation function will allow (these constants are all set to 1 in our simulations).

Propagation delay is thus a controllable parameter of a module which is tunable by external gain signals that can themselves be generated by units in other modules. In the simulation discussed here, all gain signals are fixed at either 0 or a single larger value. (In future models, these gain signals can perhaps best be modeled as the diffuse effect of a neurotransmitter such as dopamine.) Fig 1 shows the effect of a large gain value in all the units of a column i. Notice that delay is accompanied by low-pass filtering which tends to discretize the input to the column. This discretizing function itself may play a useful role in symbolic processing, but for our purposes here, propagation delay is all that is important. Nevertheless, this behavior motivates the choice of the term $ZeroLatch_i$: the $ZeroLatch_i$ resists changes in input layer representation of a value near 0. Similarly, the $OneLatch_i$ tends to hang on to values near 1 in the face of a drop in input layer activation.

Hebbian learning can be used to modulate the delay characteristics of a column so that delays of arbitrary precision (up to the limit imposed by noise in neural transmission) can be learned. With columns, the gain signal can simply be tuned up or down through Hebbian or anti-Hebbian synaptic modification of the connection from the gain signal generator to $ZeroLatch_i$ and/or $OneLatch_i$.

The delay and filtering properties of columns derive from weak positive feedback in the $ZeroLatch_i$ and $OneLatch_i$ units. When the gain signal to the $ZeroLatch_i$ unit is high and $Input_i$ is low, the $ZeroLatch_i$ unit becomes highly active. At this point, a jump upward in $Input_i$ causes $ZeroLatch_i$ to receive stronger inhibition, but positive feedback in $ZeroLatch_i$ resists this inhibition, with the effect that $ZeroLatch_i$ winds down slowly. Only when it has reached a level near 0 is the Mid_i unit disinhibited enough to transmit signals to $Output_i$.



Column Functions: Propagation Delay, Discretization/Low-pass filtering
Net external input to column
Input
Zero-Latch
Mid
One-Latch
Output
Activation Function Legend:
Dynamics with 0 external input
Dynamics with inhibitory input
Dynamics with excitatory input
Connection Legend:
Excitatory connection
Inhibitory connection
Modules: Columns can be organized with lateral inhibition into modules whose input and output layers are equivalent to Hopfield networks with orthogonal memories
Input
Output

Fig. 1. The cortical column mechanism provides a means for an excitatory gain signal to delay signal propagation and filter high-frequency inputs to a column by a variable amount. Here the effects of a maximal gain signal are shown – the activation plots are generated with $Input$ and $Output$ units only weakly self-exciting, in order to show effects clearly.

We now define a few terms to enable a more formal description of the dynamics. A *self-exciting unit* is a unit i whose output value V_i is weighted by a nonzero synaptic strength w_{ii} and added to the $NetIn_i$ term of its own activation function. Non-self-exciting units have $w_{ii} = 0$. A *weakly self-exciting* unit i has $1/w_{ii} \geq \lambda/4$. A *strongly self-exciting* unit has $1/w_{ii} < \lambda/4$.

The dynamics of self-exciting units can be qualitatively characterized by the cobweb diagrams shown in fig 2. In these diagrams, the horizontal axis represents current input from a unit to itself, $Int_i = w_{ii} \cdot V_i$, and the vertical axis is output activation. External input is held constant. Self-excitation means that at time t a unit i will be adding a proportion of its output to its own net input term at any given moment determined by the weight w_{ii}. Thus, a unit's input to itself at time t is determined by tracing from the current output value on the sigmoid curve horizontally to the line $1/w_{ii} \cdot Int_i$,

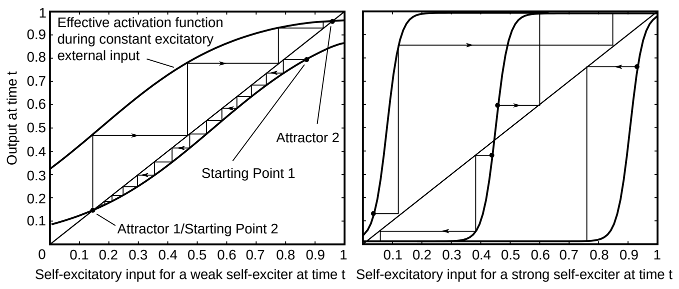


Fig. 2. *Left:* The behavior of a weakly self-exciting unit with external input held constant at two different levels. The leftmost curve might correspond to a $ZeroLatch$ unit with high gain and no inibition from $Input_i$, while the rightmost curve corresponds strong inhibition from $Input_i$. *Right:* A strongly self-exciting unit with three different levels of external input. The leftmost curve corresponds to a value sufficient to make an inactive unit latch on to a high value, and the rightmost reflects a value sufficient to wipe out a latched high value. The middle curve reflects activation-based maintenance of a value in the absence of strong external input.

and from there vertically to the horizontal axis. The rate of change of V_i for a self-exciting unit with constant net external input $Ext_i = NetIn_i - Int_i$ from other units is equal to the size of the 'stair step' so formed between the sigmoid activation curve and the line $1/w_{ii} \cdot Int_i$ (hereafter called the *reference line*). The system approaches the intersection of the activation curve and reference line whenever the slope of the reference line is greater than the slope of the activation curve at the point of their intersection.

The behavior of the system therefore depends critically on the ratio of λ to w_{ii} and on θ_i. Weakly self-exciting units have activation curves that intersect the reference line at only one point, because the activation curve has slope $\lambda/4$ at the point of inflection, and for a weak self-exciter, this slope is shallower than that of the reference line. Strong self-exciters may have one, two, or three intersections, depending on the value of θ_i.

Importantly, external inputs to a self-exciting unit i effectively cause the activation sigmoid to shift left or right along the Int_i axis for excitatory or inhibitory input respectively. Variable delay characteristics thus derive from the fact that a high gain signal shifts the $ZeroLatch_i$ activation curve leftward. With low gain, $\theta_{ZeroLatch_i}$ causes the activation curve to sit far to the right, so that it intersects the reference line very near 0. With high gain and $Input_i \approx 0$, it intersects the reference line at a value near 1. In the high gain regime, a value of nearly 1 at $Input_i$ sends inhibition to $ZeroLatch_i$ that shifts it back rightward so that the curve is close to the reference line but intersects it near 0 (see fig 2, left). In this case, a large number of small, roughly equal size stair steps indicates slow, steady decay of $ZeroLatch_i$, and therefore slow transmission of the $Input_i$ value to $Output_i$.

C. Latches and Gates

In production systems, a central clock pulse synchronizes the discrete beginnings and endings of production firing, but it seems unlikely that there is anything equivalent to a central clock pulse sent to all neurons in the brain. It is more likely

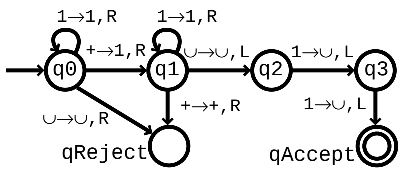


Fig. 3. A simple unary adder Turing machine. It moves right on the tape until it sees a +, which it replaces by a 1. Then it erases two 1s from the end of the second argument.

that asynchronous timing of processing events occurs. *Latches* and *gates* are the mechanisms we propose to allow for such distributed, asynchronous control.

In [9] and the neural model shown in fig 4, modules collect and integrate the outputs of other modules in order to compute their own outputs. But the components of a circuit may require the synchronous arrival of signals from multiple upstream modules in order to compute correctly. Timing problems like these are handled in digital logic design by buffering values in registers whose values are updated at each clock pulse. Here, the analogous structure is a latch, which uses strong self-excitation to buffer its values until inhibited strongly by an external signal (its effective activation function must be shifted roughly to the position of the leftmost curve in the righthand diagram of fig 2 to shut it off once active).

A gate is a copy of a module that can transmit the symbolic value of that module when activated, but can also be inhibited by external control signals while the original module maintains its original value.

III. Turing machine emulation

We now have all the pieces necessary to construct circuits that emulate Turing machines. A Turing machine consists of a finite state machine that controls the operation of a 'tape head' that moves left and right over an infinitely long memory tape, reading and writing one of a finite set of alphabet symbols at each transition of the machine. The behavior of a machine at any step of processing is determined by the current state of the control mechanism and the alphabet symbol in the current tape cell.

The circuit shown in fig 4 emulates the Turing machine shown in fig 3. This circuit is a unary adder: it computes the sum of two natural numbers m and n represented as a sequence of $(m+1)$ 1's and $(n+1)$ 1's respectively. Thus $2+1$ is represented as $111+11=1111$, which is the unary encoding for 3.

A. Finite State Controller

The finite state control circuit depicted in the top half of fig 4 stores current states and computes the transition function by computing the next state, emitting a symbol to be written to the current tape cell, and emitting a tape head movement direction. The last two signals are sent to the tape emulation circuit shown in the bottom half of fig 4. Some modules are shown with an activation plot that covers a few important windows of time during a single transition of the emulated Turing machine. All windows in these diagrams begin and end at the same cycles of the Matlab simulation.

The current state is stored in the module Q at the top of the diagram. The boxed number 1 indicates that the flow of activation during a single Turing machine transition is entirely determined by the contents of the modules in the gray box labeled 1. Q is a winner-take-all, latch module in which the column representing the current state is highly active (its $Input$ and $Output$ are near 1), and the other columns, inhibited by the active column through lateral inhibition in the $Input$ and $Output$ layers, have $Input$ and $Output$ near 0.

The current tape symbol is stored in the module Σ. At the appropriate times, the values of Q and Σ flow to $\Sigma gate$ and $Qgate$, and these values in turn excite a set of *Conjunction* nodes. Each *Conjunction* responds strongly only to a single conjunction of current state and current symbol. Conjunction nodes excite the columns in $\Sigma write$, *TapeHead* and *Qnext* that represent the symbolic values specified by the transition function of the emulated Turing machine. It is important that only the $\Sigma write$ symbol be transmitted to the tape mechanism at this point. If the *TapeHead* signal were to leak out prematurely, the tape mechanism in fig 4 would move to a different tape cell prematurely (note the distance between time steps 7 and time 8 in the activation plots of fig 4 – these time slices show that *TapeHeadGate* does not become active until well after $\Sigma writeGate$). Similarly, the next state symbol in *Qnext* should not overwrite the current symbol in Q, because the current transition is not complete at this point, and an overwrite will prevent the full transition from being carried out. Thus a set of gates are called for: $\Sigma writeGate$, *TapeHeadGate* and *QnextGate*. Both $\Sigma writeGate$ and *TapeHeadGate* make use of strong self-excitation at the $Input$ layers and weak connections from $\Sigma write$ and *TapeHead* to keep their values arbitrarily near 0 until release from inhibition by the timer, at which point, strong self-excitation allows these gates to emit $Output_i$ values arbitrarily near 1 (see fig 2, righthand diagram). This proves to be necessary in order for these representations to have their effects at the proper times.

The clock circuit shown at the top of the finite state control in fix X times transitions of the simulated Turing machine. While the node *Transition* is active, the current tape symbol and state supporting the current transition are protected from overwriting by *Transition*'s inhibition of $\Sigma nextGate$ and $QnextGate$. After a sufficient amount of time has passed for the tape head mechanism to complete its processing, the last module in the timer sequence inhibits *Transition*, allowing the new symbol and new state to flow into Σ and Q. At the same time, the gates that activate to allow this flow inhibit $\Sigma gate$ and $Qgate$, and the gates that allow control signals to flow to the tape mechanism are inhibited, ending the previous transition and preparing the next one by loading the new symbol and state (time steps 11 and 12 in fig 4).

B. Tape

The tape mechanism has to carry out a few major functions. First, when the finite state control in fig 4 is loading new

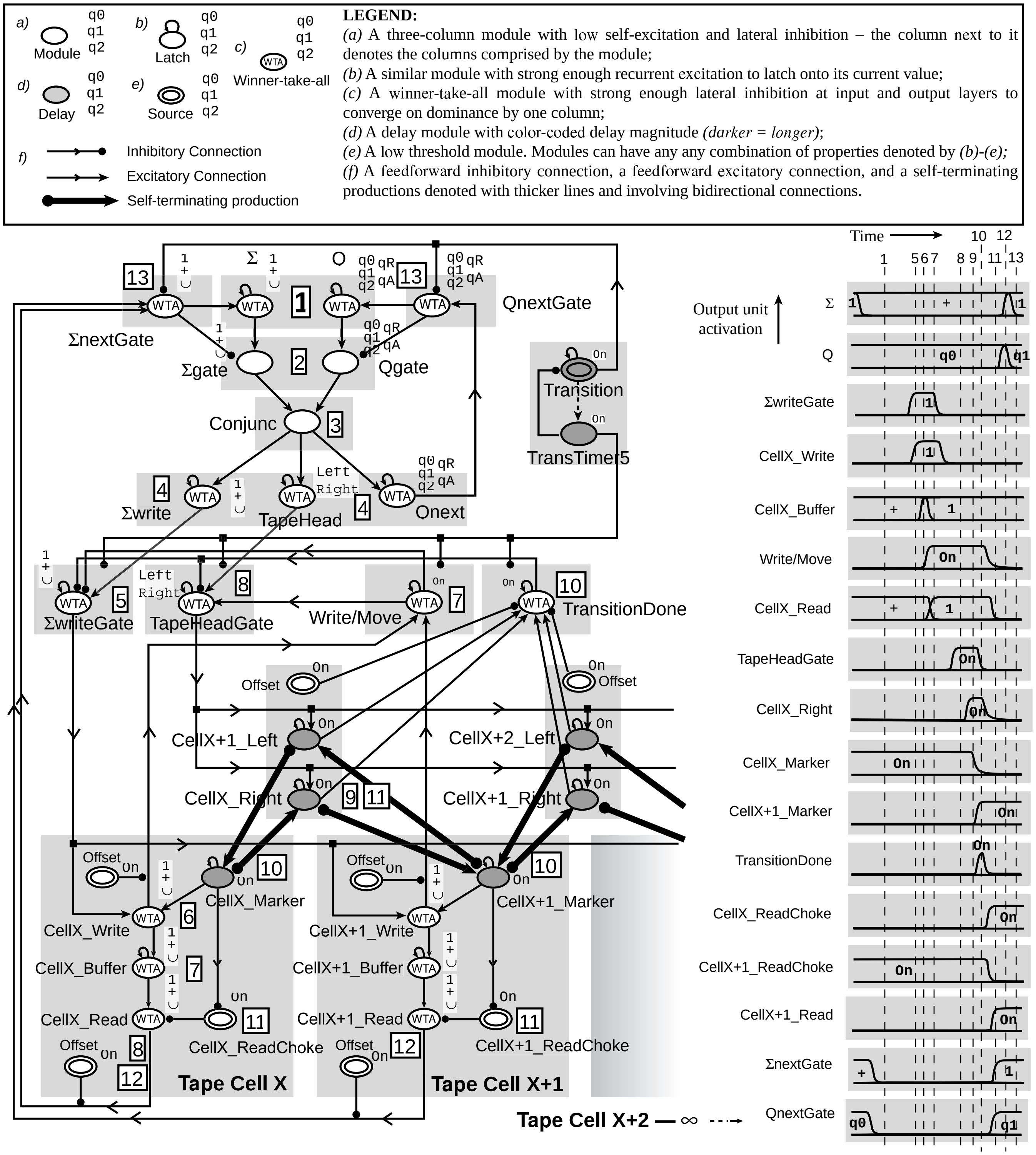

LEGEND:
(a) A three-column module with low self-excitation and lateral inhibition – the column next to it denotes the columns comprised by the module;
(b) A similar module with strong enough recurrent excitation to latch onto its current value;
(c) A winner-take-all module with strong enough lateral inhibition at input and output layers to converge on dominance by one column;
(d) A delay module with color-coded delay magnitude *(darker = longer)*;
(e) A low threshold module. Modules can have any any combination of properties denoted by *(b)-(e);*
(f) A feedforward inhibitory connection, a feedforward excitatory connection, and a self-terminating productions denoted with thicker lines and involving bidirectional connections.

Fig. 4. This circuit implements the finite state controller and tape of the unary adder Turing machine in fig 3. Numbered boxes indicate the flow of control as the machine starts in state $q0$, reads a $+$ in tape cell X, begins to write a 1 to cell X, moves the tape head right to cell $X+1$ and transitions into state $q1$. For some modules, the time course of activation of the output layer is plotted along the right side of the diagram. Numbered boxes also correspond to time slices through these plots and denote that a time step is significant because of critical changes in the modules so numbered in the connection diagram. The plots also show the symbolic value represented by the most active $Output_i$ unit in a module as time progresses.

symbol and state values, the tape must provide the tape symbol stored at the current tape cell. When the control circuit issues a write command, the current cell must overwrite its current value with the new one. When a tape head movement command is received, the current cell must activate the correct next cell. A significant amount of the complexity of the tape circuit derives from the need to execute precisely one write and one head movement per simulated transition. This precision requires the use of self-terminating productions, labeled with bold arrows in fig 4, and therefore requires the use of a propagation delay device.

The unique currently active cell X is denoted by high activity in the *CellX_Marker* node in that cell and low activity in the other marker nodes on the tape (note the lack of activation overlap in the plots for *CellX_Marker* and *CellX+1_Marker* in fig 4 – these activations are separated by time steps 9 and 10). In all tape cells, the symbol stored at that cell is represented by the activity in the *CellX_Buffer* module that persists throughout all transitions unless overwritten. Reading can only occur when the *CellX_Marker* node is active and the *CellX_ReadChoke* node is inactive. Writing can only occur when both the *CellX_Marker* node is active and a strong write signal is received by $\Sigma writeGate$.

A write operation is asynchronously timed: only when the *CellX_Write* node is very active can a write occur, and as soon as this node becomes active, it activates the *Write/Move* node. This node in turn terminates the write operation and initiates the tape head movement operation. The *Offset* node at the top of each tape cell in fig 4 is necessary to counter the cumulative effect of an infinite number of *CellX_Write* nodes on the single *Write/Move* node. Only one *CellX_Write* node is significantly active at any given moment, but all such nodes are active at some small value greater than 0, and the offset nodes send a constant inhibitory output that cancels this baseline activity and makes the net effect on downstream modules approximately 0.

Once the *Write/Move* node is active, the *TapeHeadGate* node receives sufficient excitation for the tape head movement control signal to be allowed through. At this point (time step 9), that signal combines with the effect of the unique *CellX_Marker* node to activate a single transition node, *CellX_Left* or *CellX_Right*. This activation is part of a self-terminating production that allows the transition node to latch onto a high value while at the same time inhibiting the *CellX_Marker* node. In turn, a second self-terminating production activates *Cell(X+1)_Marker* and inactivates the transition node. Prior to that node's inactivation, it also activates the *TransitionDone* node. This is necessary to inactivate the *TapeHeadGate* module so that multiple transitions do not occur. All timing in these operations is asynchronous and control is distributed among the modules involved in the operations.

At this point, the *TransitionDone* signal could be used to issue a 'next transition' command to the finite state control mechanism. This is not how the example demonstrated in fig 4 was constructed though, so instead, the system waits for the timer to expire.

IV. Conclusion

We currently have a simulation of the unary adder in fig 3 which appears to handle arbitrarily large inputs. A formalization of the mapping used for that simulation allows us to prove that any Turing machine can be translated into a neural network of the type described here, although the details of that proof have not been given. In particular, we can translate any universal Turing machine into such a network.

The significance of this result is twofold: 1) that biologically plausible neural cognitive models such as [9] are capable in principle of arbitrarily complex algorithmic computation, with complexity limited only by number of neurons, and 2) that the particular mechanisms employed in the mapping – activation-based short-term memory, productions, self-terminating productions, gates and clocks – together provide critical functionality for neural models of higher cognition.

It is interesting to note that, while we do not address synaptic plasticity in this model, synaptic weights are the only parameters that vary between modules with different behavior in terms of latching, winner-take-all dynamics, and propagation delay. It is useful to vary the activation thresholds θ_i of different neurons, but the λ_i parameters can be uniformly equal to λ throughout the model.

We emphasize that we our claim of biological plausibility extends only to the components of these circuits up to the hierarchical level of latches and gates. We do not advocate modeling cognition simply by translating arbitrary computer code into neural algorithms of the kind used in the proof we have sketched. In particular, we do not think that the tape mechanism described here ought to serve as the fundamental mechanism for short-term memory storage in cognitive models (such a system would not be conducive to associative recall). Our purpose has been simply to show that columnar attractor networks, in principle, have what it takes to compute. Nevertheless, mechanisms similar to the tape could be used as a fundamental building block for hierarchical or associative chaining methods of sequential motor programming.

References

[1] J. Anderson and C. Lebiere. *The atomic components of thought.* Lawrence-Erlbaum Associates, 1998.

[2] S. Dehaene and J. Changeux. A hierarchical neuronal network for planning behavior. *Proceedings of the National Academy of Science, USA*, 1997.

[3] J. M. Fuster and G. E. Alexander. Neuron activity related to short-term memory. *Science*, 1971.

[4] J.J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Science, USA*, 1984.

[5] J.C. Houk and S.P. Wise. Distributed modular architectures linking basal ganglia, cerebellum and cerebral cortex: their role in planning and controlling action. *Cerebral Cortex*, 1995.

[6] D.E. Kieras and D.E. Meyer. An overview of the epic architecture for cognition and performance with application to human-computer interaction. *Human Computer Interaction*, 1997.

[7] J. Laird, A. Newell, and P. Rosenbloom. Soar: an architecture for general intelligence. *Artificial Intelligence*, 1987.

[8] A. Newell and H.A. Simon. *Human problem-solving*. Prentice Hall, 1972.

[9] T. A. Polk, P. A. Simen, R. L. Lewis, and E. G. Freedman. A computational approach to control in complex cognition. *Cognitive Brain Research*, 2002.

[10] E.L. White. *Cortical circuits: Synaptic organization of the cerebral cortex, structure, function, and theory*. Birkhauser, 1989.