Goal Management in a Recurrent Neural Network

Patrick Simen
psimen@eecs.umich.edu
EECS Dept.
University of Michigan

Thad Polk and **Rick Lewis**
tpolk@umich.edu, rickl@umich.edu
Psychology Dept.
University of Michigan

Eric Freedman
freedman@umich.edu
Psychology Dept.
University of Michigan, Flint

Abstract

The management of goals and subgoals is widely acknowledged to be important in complex cognitive tasks involving planning or problem solving, and goal management plays a central role in a number of symbolic cognitive architectures (e.g., Soar, ACT). Nevertheless, few neural network models have attempted to model goal management explicitly. We present a recurrent neural network goal stack mechanism. It uses activation decay over time to distinguish more recent goals from older goals and uses attractor dynamics to retrieve the most active goal after the current goal has been achieved. We have incorporated this mechanism in a model of a well-known neuropsychological task, the Tower of London (Shallice, 1982).

Introduction

A neural model of goal management is the centerpiece of a larger effort by our group to model prefrontal cortex function in humans. Our model is based on the following basic assumptions.

- A single goal controls behavior at any given time. We will refer to this goal as the controlling goal.
- Multiple non-controlling goals can also be active at the same time as the controlling goal.
- More recent goals tend to be more active than less recent goals.
- When a controlling goal becomes inactive (e.g., because it was achieved), the most active remaining goal takes control.

We implement these assumptions using continuous-valued recurrent networks. Each goal is represented in two networks: once in its own network (called a *memory pool*), and also in a separate network (called the *goal* module) when it is the currently controlling goal. Those goal module units that are highly active when a goal is controlling are collectively called the goal's *on-set*. A goal's memory pool activity decays over time as long as the goal is non-controlling. The goal module is bidirectionally connected to each of the memory pools via excitatory connections. We also assume that the goal module receives external input (both excitatory and inhibitory) associated with specific goals from other sources that are task-specific and that it sends modulatory output to task-specific targets in order to influence behavior. In a subsequent section, we will provide a specific example for the Tower of London task.

The typical operation of the model is as follows. External task-specific input to the goal module excites potential goals. If more than one goal is excited, those goals compete via attractor dynamics to select a single controlling goal (Goal 1). The controlling goal in turn excites its associated goal memory pool via the excitatory connections from the goal module to the goal memory pools. If subsequent task-specific input excites a different goal (Goal 2, e.g., a subgoal that needs to be achieved before Goal 1 can be achieved), then the new goal will become the controlling goal and its memory pool will become activated. The memory pool for Goal 1 will therefore lose its excitatory input from the goal module and its activity will begin to decay. If another goal (Goal 3) subsequently replaces Goal 2 in the goal module, then its memory pool will be activated, Goal 2's memory pool will begin to decay, and Goal 1's memory pool will continue to decay. Hence, as new goals are added, a gradient in activation levels develops across previous goals: older goals (which have had more time to decay) will have lower activity levels than will more recent goals (which have had less time to decay). If the current controlling goal is actively inhibited (perhaps because it has been achieved) and no other goals are receiving task-specific excitation, then the active memory pools excite previous goals and those goals compete for control. In general, the most active memory pool (which typically corresponds to the most recent goal) will win the competition and the most recent goal will become the controlling goal. In this way, the model exhibits 'last-in-first-out' (LIFO) stack behavior. Task-specific input can 'push' new goals on top of the stack so that they control behavior, and when those goals are accomplished (or inhibited for some other reason), they are 'popped', and then the most recent previous goal tends to be selected.

Decay in Memory Pools

The activation function of all units in the model is a continuous differential equation determined by a function of summed input,

$$f(input) = \frac{1}{1 + e^{-\lambda * (input\ -\ bias)}}.$$

The decay process in a memory pool is governed by the relative values of λ, the uniform excitatory connection strength between units in the pool, and the number of units in the pool. Fig. 1 illustrates the process as a cobweb diagram. The vertical axis represents current activation of one of the pool's units. The activation function is overlaid on a reference line through the origin. The slope of this line is determined by connection strength and number of pool units. If there are ten units, units are self-exciting, and connection strengths are uniformly 1, then the slope of the reference line will be 1/10: if unit k currently has activation y, then the other 9 units will also be active at approximately y, since all units become active or inactive together in the absence of noise. Thus, given connection strengths of 1, the input to unit k on the next cycle will be $10 * y$ (in other words, move horizontally from the point on the activation curve with height y until reaching the reference line in order to find the next cycle's input value). The course of activity over time in one unit will therefore follow the trajectory shown by the stair steps, as long as external input to the pool is held constant. (More accurately, since activation in the model is determined by an approximately continuous differential equation, net input to a unit on the next cycle will increase or decrease only slightly relative to its previous value in the direction indicated by the cobweb diagram. The effect is the same however: decay at a rate controlled by λ and connection strength.)

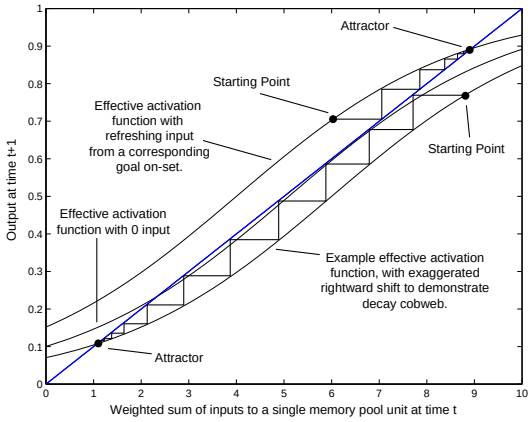

Figure 1: Cobweb diagram illustrating decay and refresh of single unit activation in a memory pool. Here, λ is 0.43 and *bias* is 5.1.

External input to a memory pool comes from the corresponding pattern on-set in the goal module. Since goal module on-sets spend most of their time either maximally or minimally active, intermediate, transient levels of activity can be ignored. The effect of an on-set being fully active is that the activation curve for memory pool units is shifted to the left. This is because input to a unit is now based on the sum of inputs from other memory units plus a relatively constant level of goal module input, so the leftward shift is equal to the input from the goal module. In that case, the cobweb diagram indicates that the memory pool will become refreshed to a high level of activity. When the goal module on-set is minimally active, the memory units' activation curves are based on internal pool activity alone and are shifted back to the right. In this case, the cobweb diagram indicates a slow, steady rate of decay.

We chose λ so that f's derivative at $input = bias$ is nearly equal to the slope of the reference line. This results in a steady rate of decay over a large portion of a unit's activation range, rather than a rate that varies over time. This choice was motivated by the need to keep memory activity of successively active goals as uniformly spread out as possible. Decay that is too fast will result in the forgetting of goals. Decay that is too slow will result in goal memories which are too close to each other to ensure LIFO retrieval.

Tower of London model

We have incorporated this goal stack mechanism in a specific model of the Tower of London task (Shallice, 1982). This task has been used extensively to assess planning impairments and is thought to depend crucially on goal management (Ward & Allport, 1997). The task is a variant of the Tower of Hanoi problem and involves moving colored balls on pegs from an initial configuration until they match a goal configuration (Fig. 4). There are no constraints on which balls can go on which others (unlike the Tower of Hanoi problem), but the pegs differ in how many balls they can hold at one time (the first peg can hold one ball, the second peg can hold two, and the third peg can hold three). Participants are often asked to try to figure out how to achieve the goal in the minimum number of moves and are sometimes asked to plan out the entire sequence of moves before they begin.

The structure of the model is illustrated in Fig. 2. The goals correspond to getting specific balls in specific locations (e.g., getting the blue ball onto the bottom of the third peg). In addition to the goal module and goal memory pools, the model explicitly represents the current configuration of the balls (in the *visual* modules), which balls are blocked from moving (the *ball status* modules), the goal configuration (in the *goal visual* modules), and the currently selected move (in the *move* module). Finally, the model also encodes information that is relevant to the current goal, specifically: what is above the ball that the goal refers to (*abovesource*), what is in the target position (*intarget*), and the lowest free position on the peg that is not the source or target of the current goal (*freeposition*) This information is crucial for getting balls out of the way without disrupting progress toward the current goal.

Excluding the influence of the goal module on behavior, the operation of the model is fairly simple. The representation of the configuration (*visual* modules) excites all legal moves in the *move* module and moves that involve blocked balls are strongly inhib-

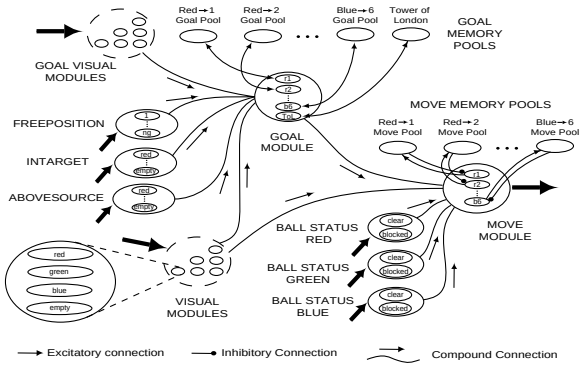

Figure 2: Model schematic: ovals represent modules, and ovals within ovals represent on-sets. Rightward bold arrows indicate clamping by Matlab code.

Tower of London algorithm:

1. Goal is initialized to 'Tower of London'.
2. **IF** all goals are achieved (i.e., the current board configuration represented in Visual matches the goal configuration), simulation ends.
3. **ELSE** GoalVisual is clamped to represent the most important, unachieved base-level goal to work on, and excites that goal's on-set in Goal.
4. **IF** the move that would achieve the current goal is legal, Goal causes Move to make that move. Then Visual and BallStatus are clamped to values based on the updated environment, and the goal-push modules are clamped to 0. The new Visual configuration then wipes out the current goal representation and memory pool with strong inhibition (it begins to pop the stack).
   - 4.1. **IF** all goals are achieved, simulation ends.
   - 4.2. **ELSE IF** it was active recently enough, the most recent unachieved goal is retrieved (completing the pop and bringing the retrieved element to the top of the stack). (4) is repeated.
   - 4.3. **ELSE IF** multiple goals emerge, Goal and Goal memory pools are clamped to zero, and (1) is repeated (this situation occurs very infrequently).
   - 4.4. **ELSE** the Tower of London goal is retrieved. (2) is repeated.
5. **ELSE** Freeposition, Abovesource and Intarget are set appropriately in order to push a new subgoal.
   - 5.1. **IF** Freeposition is set to 'no goal (ng)', which can happen sometimes, no information is available (according to this algorithm) to guide subgoal selection. The 'ng' pattern in Freeposition inhibits the Goal net strongly in that case, and the Move module is now free to select any legal move, in order to do something essentially random.
     - 5.1.1. **IF** a unique move is selected, Visual and BallStatus are updated, and the goal-push modules are zeroed. (2) is repeated.
     - 5.1.2. **ELSE** the Move module is zeroed, and (5.1.1) is repeated (this situation is also rare).
   - 5.2. **ELSE** a new goal is activated according to the perceptual attributes, and that goal's memory pool becomes refreshed. (4) is repeated.

Figure 3: The model algorithm.

ited by the *ball status* modules. The possible moves then compete with each other in the *move* module via attractor dynamics until one is selected (i.e., convergence to an attractor pattern is detected using a threshold on Euclidean distance between previous *move* module activity and current activity). Without other sources of input, the move that is finally selected is random and simply depends on noise. Once a move is selected, the representation of the configuration is changed accordingly, and the new configuration once again votes for any legal moves. In short, in the absence of the goal module, the model simply performs random search using any moves that are legal in the current configuration.

The goal module modulates processing in the *move* module by exciting moves that will achieve the current goal and inhibiting moves that won't. This modulation biases the competition in the *move* module so that moves that will achieve the current goal will tend to be selected. If no legal move will achieve the current goal, then no move is selected (because the current goal will inhibit all legal moves in that case).

The selection of a new goal can occur in one of three ways. First, if the current goal has been achieved, then it is inhibited and the most recent goal on the stack will tend to be reinstantiated. Second, if the current goal cannot be directly achieved because of some obstruction (either a ball in the target position or a ball above the ball that we want to move), then a new goal to remove the obstruction will be proposed (via input from *abovesource*, *intarget*, and *freeposition*). Third, if there is no specific current goal (aside from just solving the Tower of London problem), then the goal configuration will vote for getting the balls into the final positions. These goals compete in the goal module until one wins and assumes control.

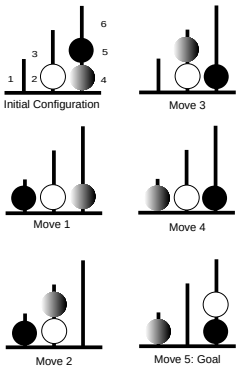

Figure 4: A five-move Tower of London configuration sequence (optimal solutions of four moves exist). Black stands for blue, shaded stands for green, and white stands for red.

Figs. 5 and 6 illustrate the behavior of the model on the problem from Fig. 4. The top panel of Fig. 5 plots the activation of different goal module on-sets over time during problem solving. The plotting symbols correspond to goals as indicated in Fig. 6. Activation of various on-sets is largely all-or-none, demonstrating that only one goal can influence behavior at a time. The bottom panel in Fig. 5 shows the activation of the corresponding memory pools. As the figure illustrates, activation of previous goals decays away when they are not directing behavior. When a current goal is achieved, the memory pools compete to choose a new goal and the most recent goal wins. When a previous goal is reinstated as the controlling goal, its memory pool is also refreshed.

The Goal Stack as a Tool for Cognitive Modeling

An appealing feature of this goal stack mechanism for cognitive modeling is that stack depth is inherently limited. This is encouraging, since people

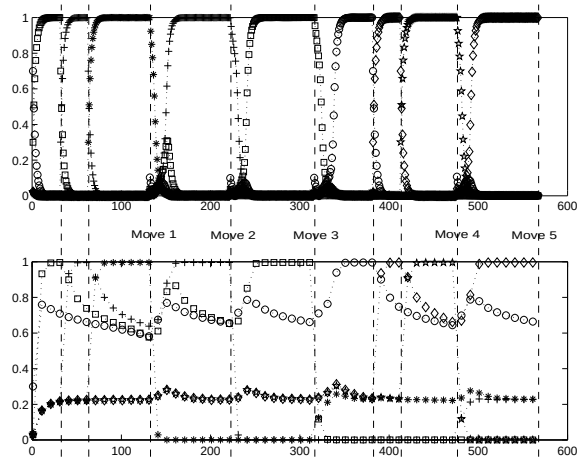

Figure 5: Memory pool and goal module pattern activation. Dashed lines show cycles at which convergence was detected. Marker symbols correspond to goals as in Fig. 6.

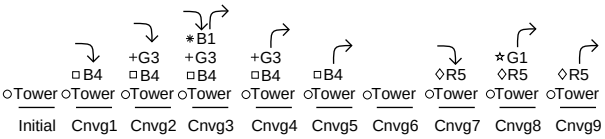

Figure 6: The stack structure implicit in the activation plot of Fig. 5. Pushes and pops are indicated by downward and upward arrows respectively.

clearly exhibit limited working memory for goals. In order to get stack functionality, memory pools must be constructed so that successive goals can be distinguished by the goal module in terms of relative activation levels: if differences are not large enough, the goal module may activate a pattern out of order, or simultaneously activate multiple patterns. If decay is too slow, activation levels for successive goals will not be distinguishable by the time the next convergence occurs. The consequence of decay, however, is that without external refreshing input, memory activity becomes minimal in finite time, so goals pushed too deeply into the stack have a tendency to 'drop out the bottom' and be permanently forgotten. We do not currently know the value of the limit, or how maximum stack depth relates to the various factors influencing recall. But it appears that increasing the effective stack depth requires an increasingly precise relationship between activation function parameters and self-excitation and number of units in memory pools – this in turn produces an increased sensitivity to noise. Thus, arbitrarily deep stacks are implausible.

Goal memory activation dynamics show further interesting properties: although the depth of the stack is limited, the persistence of a goal below the top of the stack need not be. That is, even though a goal may have failed to become active in the goal module for an arbitrarily long period, its memory pool activation may remain high indefinitely. This can happen when competition between goals in the goal module is such that some losers of the competition come close to becoming active before ultimately losing (after the third convergence in Fig. 4, for example, 'g→3' is retrieved, but memory pools for 'b→4' and 'Tower of London' both show reversal of decay and maintenance of relative order). In doing so, a goal module on-set must become transiently active at a level between minimal and maximal. This activity also provides refreshing input to the corresponding memory pool, so decay is slowed or slightly reversed. The effect appears almost 'homeostatic', in that memories in the goal stack retain their relative activation level order, and the topmost goals (those which are active enough to recruit their own refreshing input after the next goal pop) persist indefinitely in the stack.

Means-ends analysis, a ubiquitous human problem-solving strategy, is defined in terms of a goal stack (Newell & Simon, 1972). Goal stacks are also commonplace in symbolic cognitive models, so achieving a neural implementation is a step toward unifying symbolic and subsymbolic computation. However, there has recently been criticism from within the ACT-R symbolic modeling community of ACT-R's reliance on a perfect goal stack (Altmann & Trafton, in press). Altmann and Trafton argue that perceptual cues and goal activation decay should take the place of the perfect ACT-R stack, and that non-LIFO goal retrievals will therefore be a natural occurrence in many domains. In the same spirit, the neural stack-mechanism presented here should be seen as a special case of a more general use of memory pools and goal modules which could easily be adapted to allow for perceptual cues (and any obvious implementation of perceptual goal-retrieval cues seems guaranteed to disrupt strict LIFO behavior). Nevertheless, it is important to show that a neural model is capable of managing goals in the absence of such cues, so we have restricted our attention to that task here.

References

Altmann, E., & Trafton, J.G. (in press). Memory for goals: an activation-based model. *Cognitive Science.*

Hopfield, J.J. (1984). Neurons with graded response have collective computational properties like those of 2-state neurons. *Proceedings of the National Academy of Sciences, 81,* 3088–3092.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Shallice, T. (1982). Specific impairments of planning. *Philosophical Transactions of the Royal Society of London, 298,* 199–209.

Ward, G., & Allport, A. (1997). Planning and problem-solving using the five-disc Tower of London task. *The Quarterly Journal of Experimental Psychology, 50A (1),* 49–78.