Experience-Induced Neural Circuits That Achieve High Capacity

Experience-Induced Neural Circuits That Achieve High Capacity

Vitaly Feldman* Leslie G. Valiant†

January 23, 2009

Abstract

Over a lifetime cortex performs a vast number of different cognitive actions, mostly dependent on past experience. Previously it has not been known how such capabilities can be reconciled, even in principle, with the known resource constraints on cortex, such as low connectivity and low average synaptic strength. Here we describe neural circuits and associated algorithms that respect the brain's most basic resource constraints and support the execution of high numbers of cognitive actions when presented with natural inputs. Our circuits simultaneously support a suite of four basic kinds of task that each require some circuit modification: hierarchical memory formation, pairwise association, supervised memorization, and inductive learning of threshold functions. The capacity of our circuits is established via experiments in which sequences of several thousands of such actions are simulated by computer and the circuits created tested for subsequent efficacy. Our underlying theory is apparently the only biologically plausible systems-level theory of learning and memory in cortex for which such a demonstration has been performed, and we argue that no general theory of information processing in the brain can be considered viable without such a demonstration.

1 Introduction

Cortical computations face severe resource constraints. Each neuron is connected to only a very small fraction of the other neurons (Braitenberg & Schuz, 1998). Also, the individual influence of any one neuron on another to which it is connected is often weak, the synaptic strength being weak on the average (Abeles, 1991). Further, cortex can perform complex tasks very fast, even tasks involving learned information such as the recognition of manufactured objects (VanRullen & Thorpe, 2001a, 2001b), and it is consequently believed that it must be doing these using a modest number, say 10-20, of temporal steps of neural processing, perhaps in feedforward fashion. Yet, it can handle large numbers of concepts, perhaps in the hundreds of thousands, without undue interference amongst them. The severity of these constraints taken together makes it challenging to identify any computational mechanisms that might overcome them together, even in principle.

In this paper we formulate this problem as one of establishing circuits by processes that are prompted by natural inputs to the circuits. These processes will be of four different kinds, each realizing a separate *task*: hierarchical memory formation, pairwise association, supervised memorization, and inductive learning of threshold functions. The circuit will be exposed to a sequence of inputs. Each input will *prompt* an instance of one of these four tasks of circuit modification to be executed. Each act of circuit modification will aim to add the new functionality demanded by that input, without causing previously added functionalities to be seriously degraded. After a long enough sequence of such task executions, any further modifications will

*IBM Almaden Research Center, San Jose, CA 95120, USA

†Harvard University, School of Engineering and Applied Sciences, Cambridge, MA 02138, USA

inevitably cause more serious degradation. The upper limit attainable without such degradation we shall call the *capacity*. We investigate this capacity by performing computer simulations on a network that supports algorithms for the four tasks and is subject to long sequences of such prompts.

The main finding is that the circuits we describe do attain impressive capacities using realistic resource and computation-time constraints. The phenomena we investigate dichotomize into two algorithmic regimes. In regime α weak synapses are sufficient and highly distributed representations result. In regime β large synaptic strengths are needed and sparse representations are supported. We find that regime α can support sequences of tens of thousands of actions, and regime β hundreds of thousands. Such capacities are necessary to reconcile with the various estimates of human capacity, such as the 30,000 estimate of the number of visual concepts (Biederman, 1987).

Our approach can be distinguished from previous work by the simultaneous emphasis on building circuits for *multiple tasks*, and awareness of biologically realistic *quantitative constraints* on the parameters of neuron numbers, neuron connections, synaptic strengths, computation time, capacities, and the numbers of neurons that correspond to a single real-world item. Previous theoretical approaches to analyzing networks of neuron-like structures have tended to analyze one task at a time. For example, there is a large literature on associative memories (Willshaw, Buneman, & Longuest-Higgins, 1968; Marr, 1969; Hopfield, 1982; Graham & Willshaw, 1997), structures that enable fixed patterns to be memorized and retrieved. There is another literature on inductive learning algorithms (Rosenblatt, 1958; Minsky & Papert, 1969; Rumelhart, Hinton, & Williams, 1986; Littlestone, 1987). Such previous approaches have not yielded general schemes that achieve the large capacities for multiple tasks and realistic constraints such as we consider here.

We emphasize the quantitative aspect of our approach. Our algorithms rely on particular relationships among the following four parameters: n the number of neurons, d the number of neurons from which a neuron receives presynaptic inputs, k the minimum number of neurons that need to be active to cause a common neighbor to become active, and r the number of neurons that are active when a typical real world item, such as an odor for an olfactory system, is represented. All these parameters can be measured in principle. All four have been measured and analyzed for at least one neural system (Valiant, 2006; Jortner, Farivar, & Laurent, 2007), and some can be estimated for others (Quiroga, Reddy, Kreiman, Koch, & Fried, 2005; Valiant, 2006). These parameter values and the quantitative relationships among them that our algorithms support can offer telltale signs of whether our style of algorithms may be at work, where direct verification is beyond current experimental methods.

The algorithms described and tested are based on simple synaptic modifications triggered by the inputs. They take one or two basic steps, which guarantees fast computation times. Such algorithms are called *vicinal* (Valiant, 1994) because they emphasize neighborly communication among neurons and the storage of information in the neighborhood of where related information is stored. The biological grounding of this type of algorithm has been investigated by Shastri (2001). The relationship intended between our model and biology is that the model be faithful in the four numerical parameters n, k, d and r but otherwise be clearly supportable by neurons and hence underestimate their capabilities.

In this work we consider vicinal algorithms for the four tasks described above. Algorithms similar to the ones we investigate here have been previously analyzed one execution at a time (Valiant, 1994, 2005, 2006). Analysis of long sequences of such executions appears to be beyond current analytic methods. Here we demonstrate through computer simulations of long sequences of such actions that high capacities are achievable. In order to make these multiple functions attain substantial capacities we needed to make several innovations over the previous work. First, we developed a more refined semantics of the network, one that distinguishes appropriately among the following three situations: a given real-world item A is being recognized, item A is not being recognized, and the intermediate third case, which in normal operation is guaranteed to occur with only negligible probability. We believe that the proposed semantics is appropriate for exploring phenomena more complex still than the ones tested in this paper. Second, compared to previous work (Valiant, 1994, 2005) we have made regime α algorithms particularly simple, and they all now

require just one circuit level. Third, our algorithms incorporate new methods for dealing with interference in the network. Previously suggested algorithms do not handle several types of interference that were observed in the course of simulation. Finally, our suite of algorithms have a modest number of adjustable numerical parameters overall, and we needed to choose these carefully to achieve the reported capacities. The computational experiments described involve substantial computations, and were performed on a large memory (256GB) 16-processor machine.

The relationship of our approach to previous work is discussed more fully by Valiant (1994, 2005, 2006). We note here that the allocation of unused memory to a new item, in the manner of our hierarchical memorization, has been called recruitment learning (Feldman & Ballard, 1982; Feldman, 1982). With respect to earlier general discussions of localist versus distributed representations, and sparsity versus density, we observe that our two regimes are both precisely defined in terms of numerical parameters. Both regimes α and β have a highly distributed nature, but the latter also has an additional strongly localist aspect.

In conclusion, we observe that the rationale for the two regimes α and β that we investigate in this capacity study, is simply that these are the only two for which we have previous evidence (Valiant (1994, 2005, 2006)) of a capability for computing instances of all of the tasks that we target. At the simplest level regime α should be thought of as having essentially minimal assumptions - a web of randomly connected neurons with all synapses weak and all algorithms conceptually simple and working between pairs of directly connected neurons. One would have to conjecture that this regime would be easy to evolve once neurons are available because the algorithms are the simplest imaginable. However, as we show the capacity of this regime while considerable is nevertheless limited and does not scale well with the total number of neurons. In order to go to higher capacities the only proposal we know is our regime beta, in which some extra complexity is needed, namely the existence of maximally strong synapses and algorithms act conceptually via intermediate so-called relay neurons. Note that the algorithms are still strictly distributed, acting on each neuron independently using only information local to it and the firing activity of its presynaptic neurons. However, the mechanisms are more complex and one would have to conjecture, for example, that if they exist in biology then they would have been discovered at a later stage of evolution

2 Overview

We shall now describe how items are represented in the network and give a brief overview of how the network is created and tested.

2.1 Item Representation

We represent a real world *item*, such as an event, word, concept or odor, by a set of model neurons. More precisely, let A be such a real world item. A basic assumption of our model is that there exists a set of neurons S_A such that the activity of neurons in S_A correlates strongly with A being recognized. The model does not require perfect correlation between each neuron in S_A and the system's recognition of A. Such perfect correlation is impossible, for example, if the neuron sets of different items overlap, as is allowed in our regime α. Instead the recognition of A will be associated with a probability distribution over the fraction of neurons in S_A that are firing. The nonrecognition of A will be associated with a different probability distribution. While we do not know the exact distribution for every item we assume that such distributions assign high weight to fractions close to 1. More formally, we introduce the notion of the ON fraction bound function, $\mathcal{C}_{ON}$. For every fraction p, $\mathcal{C}_{ON}(p)$ bounds from below the probability that at least p-fraction of neurons representing an item A will be active when A is recognized. The probability is taken over all the times that A is recognized. Similarly, we define $\mathcal{C}_{OFF}$ to be the function that bounds from above the probability that at most p-fraction of neurons representing an item A will be active when A is *not* recognized.

Specifying these bounds is essential for making specific what the simulations are to achieve, and we shall use the following functions for this purpose. For real $a, b \in [0,1]$ and τ such that $\texttt{sign}(\tau) = \texttt{sign}(b-a)$ we define a fraction bounding functions $\mathcal{C}[a,b,\tau]$ to be

$$\mathcal{C}[a,b,\tau](p) = \begin{cases} 0 & \texttt{sign}(\tau) = \texttt{sign}(a-p) \\ 1 & \texttt{sign}(\tau) = \texttt{sign}(p-b) \\ 1 - \frac{2^{-p/\tau} - 2^{-b/\tau}}{2^{-a/\tau} - 2^{-b/\tau}} & \text{otherwise} \end{cases}$$

For our regime α when an item is recognized, we shall choose an ON distribution and characterize it by the parameters [0.98,0.88,-0.01]. This will mean that when the item is recognized (i) with probability 1 at least 88% of the neurons in S_A fire, (ii) for a fraction $p \in [0.88, 0.98]$ the probability that less than p-fraction of the neurons in S_A fires is exponentially decreasing as p decreases (with 100=1/0.01 as an exponent multiplier), and (iii) nothing is guaranteed for fractions above 98%.

For the same regime when an item is not recognized we choose an OFF distribution with parameters [0.05, 0.3, 0.025]. This will mean that (i) with probability 1 at most 30% of the neurons in S_A fire, (ii) for a fraction $p \in [0.05, 0.30]$ the probability that more than p-fraction of the neurons in S_A fires is exponentially decreasing as p increases, and (iii) nothing is guaranteed for fractions below 0.05%. We define parameters for regime β similarly. The specific parameters therefore impose a precise semantics on the network of neurons by which the success of any algorithm can be evaluated. We note that this semantics comes in two varieties depending on whether the neuron sets S_A for distinct items are disjoint or not. These two versions are refinements of what have been called *positive* and *positive shared* representations (Valiant, 1994, 2005), which are employed in regimes β and α, respectively.

2.2 Model of Computation

The underlying model of computation used for the networks is the neuroidal model (Valiant, 1994), which was designed to capture the communication capabilities and limitations of cortex as simply as possible. Below we provide a brief summary of the relevant assumptions. A neuroidal net consists of a weighted directed graph G with a model neuron or *neuroid* at each node. A neuroid is a linear threshold element that can be in one of a fixed set of states. The weight of an edge or connection from node v to node u is $w_{v,u}$ and models the strength of the synapses for which v is presynaptic and u is postsynaptic. The only way a neuroid u can be influenced by other neuroids is through the quantity w_u which equals the sum of the weights $w_{v,u}$ over all neurons v presynaptic to u that are in firing states. The firing of a neuroid can be caused either by an external input or when $w_u \geq \Theta$ for threshold Θ. In this paper the values of Θ are fixed throughout the computation. Each neuroid executes an algorithm that is local to itself and can be formally defined in terms the transitions that update the neuroid's state and the weights of its incoming connections. The changes made by a transition can depend only on the neuroid's current state and w_u.

The model assumes a simple timing mechanism. There is a global synchronization mechanism such that if the representations of a set of items, such as items A and B, are prompted to fire simultaneously, by an external input for example, then the neurons in these representations will fire synchronously enough that the succession of transitions that will be caused to execute by the algorithms running locally on the relevant neurons will keep in lockstep for the few steps of duration of these algorithms. In this paper all state and threshold transitions are assumed to take one unit of time.

2.3 Design Overview

The networks that we investigate consist of two main parts or *layers*. The first layer, which we refer to as *primitive*, will contain the neurons used for the so-called primitive items to which neurons are preassigned. These can be thought of as representing sensory input primitives preprogrammed at birth. The second or

main layer will contain the neurons for all the other items. The basic process of adding an item is *hierarchical memory formation,* which creates an item in the main layer that is the conjunction of two previously existing items. The other cognitive tasks involve changing the weights and states of neurons in the main layer. We shall, in particular simulate three such tasks: *association*, *supervised memorization*, and *inductive learning.*

We examine two regimes which have been identified as efficacious for neural processing by earlier theoretical work (Valiant, 1994, 2005). In *regime* α the influence of each neuron on its neighbors is limited. Specifically, to cause a neuron to fire at least k neurons directly synapsing on to it have to fire simultaneously, where $k > 1$. In general large item size r is required to overcome the weakness of the influences. Representing a significant number of items in this regime is only possible if the sets of neurons representing items may overlap. In Figure 1 we provide a schematic view of a regime α network.

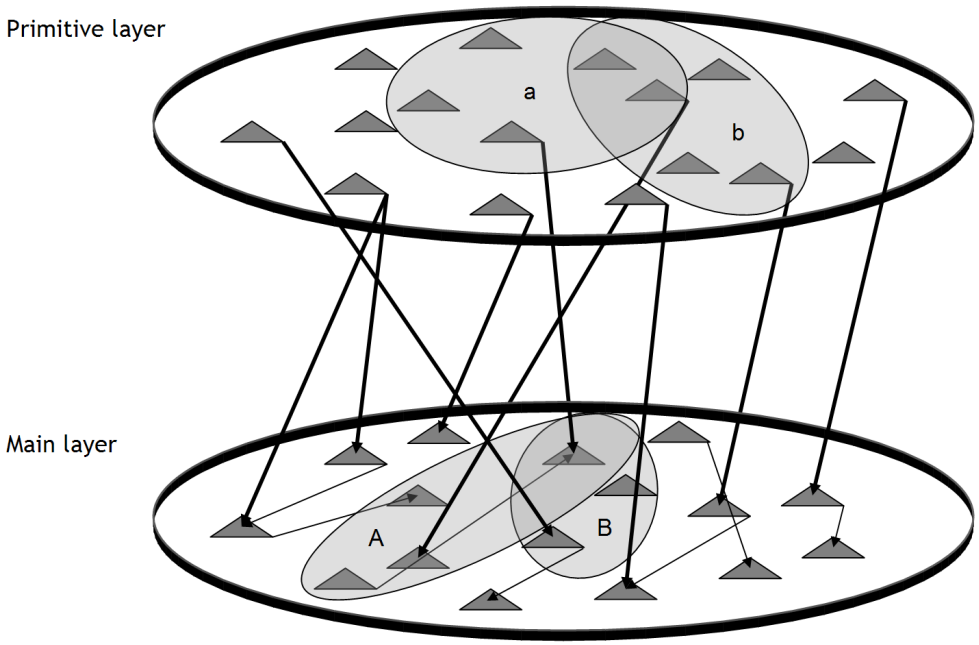

Figure 1: Regime α network: a and b are primitive items; A and B are main layer items.

In *regime* β, the firing of a single neuron is sufficient to cause its neighbor to fire. In other words $k = 1$. In addition, in this regime neurons in the main layer are connected to each other via a layer of *relay* neurons. Strong synapses and use of the relay layer allow significantly smaller item sizes and hence items may be represented by disjoint sets of neurons and higher capacities attained. In Figure 2 we provide a schematic view of a regime β network.

We note that *regime* α will require only minimally simple network structure and algorithms, and only weak synapses. *Regime* β is a little more demanding in these respects, but in return will be shown to achieve extremely high capacities.

The life-cycle of the network that we simulate can be split into the following stages:

- Create the neurons and connections of both the primitive and the main layers. In regime β the relay layer and its connections are also created. Assign neurons to items in the primitive layer.
- Neurons are assigned to items in the main layer via hierarchical memory formation. For a pair of items

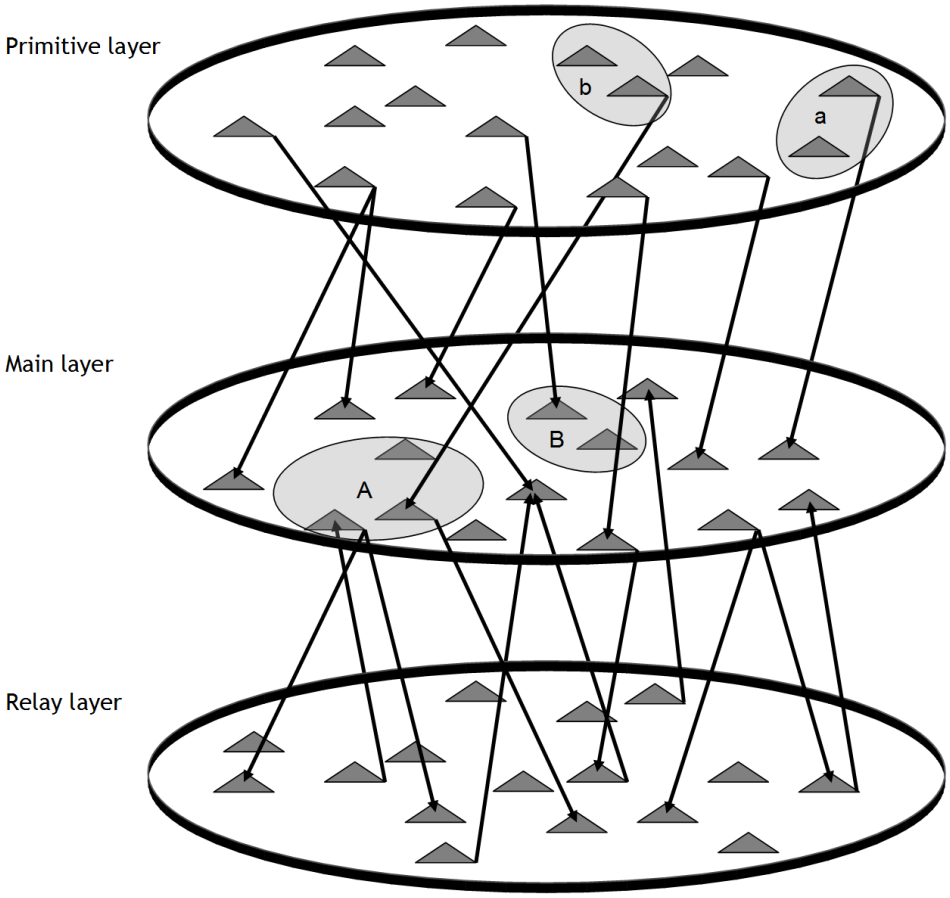

Figure 2: Regime β network: a and b are primitive items; A and B are main layer items.

A and B in the primitive layer, neurons in the main layer that are the most strongly connected to the neurons representing A and B will represent the conjunction $A \wedge B$.

- Perform the sequence of association, supervised memorization, and inductive learning requests on the items in the main layer.
- Measure the effectiveness and robustness of the circuits created by these requests and the network as a whole.

Hierarchical memory formation is the process in which a higher-level item corresponding to a combination of specific lower-level items is created. Here we only consider one level of hierarchical memory formation, namely, for a pair of primitive items A and B, an item that represents $A \wedge B$ is formed in the main layer. The neurons that are assigned to represent $A \wedge B$ are the neurons that fire whenever all neurons in both A and B are firing. While it is easy to implement this process using a vicinal algorithm (we describe such algorithms for both regimes in Section 4), the circuits we obtained are not robust to interference from later executions of other tasks. Therefore we use this stage only to produce an assignment of neurons to items in the main layer that is based on connections to combinations of items in the primitive layer. We note that in regime α items created via this method often share a significant fraction of neurons making interference more likely.

The goal of each task execution in the main layer is to modify the weights so that on future inputs some chosen *target* item becomes recognized for new combinations of certain other *source* items being recognized. The network supports the following types of task.

- **Association**. An item A is associated with item B if activation of B activates A.
- **Supervised memorization**. An item A is a memorized conjunction of items B_1 and B_2 if A is activated whenever both B_1 and B_2 are activated. (Each supervised memorization task in our simulations is executed in two steps.)
- **Inductive learning**. Inductive learning is used when the desired functionality is not known in advance but correct examples of this functionality are observed. It permits creation of a variety of functionalities via a single learning mechanism. Here we will use a simple mechanism based on the Winnow algorithm (Littlestone, 1987). This mechanism enables learning of monotone linear threshold functions. We tested this mechanism on threshold functions of up to eight items and weights from the set $\{0, 1, 2\}$. We also assumed that the examples which are very close to the separating plane do not occur, in other words there is a margin between the separating plane and the observed examples. Learning of linear thresholds with margins by neural networks was previously studied by Arriaga and Vempala (1999). They also point to a number of cognitive experiments that support the use in human learning of linear threshold functions with substantial margins.

  Unlike the other tasks learning involves a step for each one of the many examples that are presented. These steps are not necessarily performed consecutively.

In Figure 3 we illustrate the relations between items that can be created by executing tasks on the network.

A task execution is initiated by a prompt, i.e. by setting a subset of the neurons representing the relevant source items to firing states. The subset depends also on the semantics of representation defined above (see Section 5 for details). In addition it is assumed that the type of the task is "known" to each of the neurons representing the target item. This is necessary to ensure that the local algorithm for the given task type is performed at each of the neurons of the target item. We take no position on how the type of task to be executed is communicated to the target neurons. Possible options include additional vicinal algorithms, fine grain timing mechanisms in the prompt, or global mechanisms akin to neuromodulators in the brain.

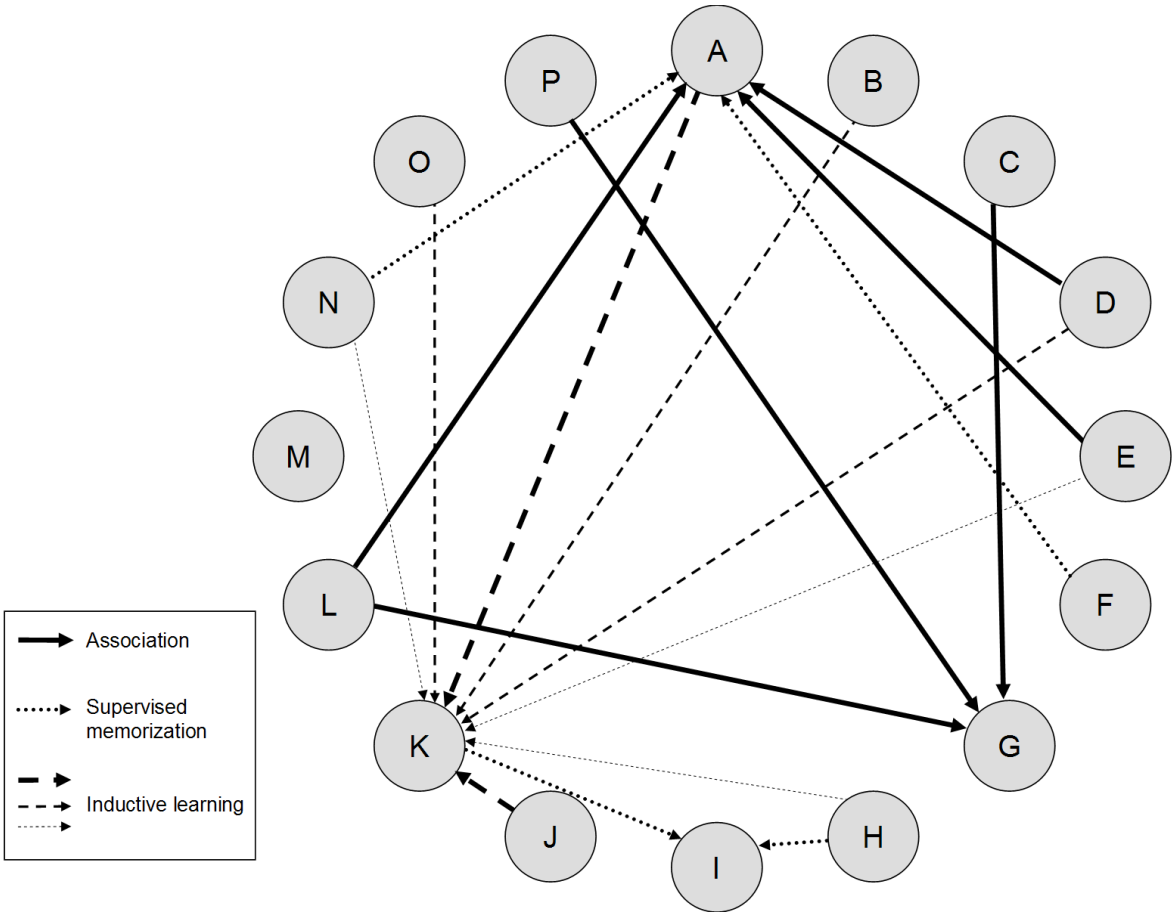

Figure 3: Relations between main layer items after the following sequence of tasks: (1) A is associated with D; (2) G is associated with C; (3) I is memorized as conjunction of K and H; (4) G is associated with P; (5) A is associated with L; (6) A is associated with E; (7) A is memorized as conjunction of N and F; (8) K is learned as a linear threshold of N, O, A, B, D, E, H and J (the thickness of an edge represents the weight that is contributed by a source item); (9) G is associated with L. Note that (8) would be executed as a result of the presentation of a series of examples, which need not be contiguous but may be intermingled with other task executions.

The list of all task instances that are executed in a simulation corresponds to complex relations among the items in the brain. In our simulations these relations are chosen randomly and uniformly within a number of constraints governed by explicit parameters. The capacity of a network with naturally created relations and examples may differ from the capacity for random relations.

In the rest of this document we will describe in detail each of these stages, list the parameters that govern their execution, and describe the numerical results obtained. The description of the tasks and testing is given from the perspective of the simulator. This makes the algorithms for the tasks and the assumptions on the external inputs explicit.

3 Network Creation

In this stage first the neurons that represent neurons of the primitive and the main layers are created. The number of neurons in the primitive layer is controlled by parameter `primitiveNetSize` and the number of neurons in the main layer is controlled by the parameter `netSize`. In addition, in regime β `relaySize` neurons are created that represent the relay neurons. For simplicity, we will use primarily networks with

$$\texttt{primitiveNetSize} = \texttt{netSize} = \texttt{relaySize}$$

and refer to this size by n.

In the next step random edges are created from the primitive layer to the main layer and inside the main layer. Each edge in this network models a synaptic connection between two neurons. The strength of a synapse is modeled by an integer number in the range from 0 to `maxSynapseStrength`. This implies that all weights in the network are non-negative, that is correspond to excitatory synapses. This will be sufficient for implementing the tasks we consider. We note that the underlying model also supports negative weights to model inhibitory synapses.

For each of the neurons of the primitive layer, `primitiveDegree` outgoing connections are created to distinct randomly and uniformly chosen neurons of the main layer. The initial strength of these connections is set to the maximum value (`maxSynapseStrength`). In regime α, for each of the neurons of the main layer, `degree` incoming connections are created from distinct randomly and uniformly chosen neurons of the main layer. The initial strength of these connections is set to 0. In regime β, for each of the neurons of the main layer, `degree` incoming connections are created from distinct randomly and uniformly chosen neurons of the relay layer and `relayDegree` outgoing connections are created to distinct randomly and uniformly chosen neurons of the relay layer. The initial strength of the incoming connections is set to 0 and the initial strength of the outgoing connections is set to the maximum value. For simplicity we will use

$$\texttt{primitiveDegree} = \texttt{degree} = \texttt{relayDegree}$$

and denote this degree by d.

We note that expected effective incoming degree of each neuron in the main layer is `degree`+`primitiveDegree`$\cdot$ `primitiveNetSize`/`netSize`. Expected outgoing degree for every main-layer neuron in regime α is `degree`. In regime β each relay neuron has expected incoming degree of `relayDegree` $\cdot$ `netSize`/`relaySize` and outgoing degree of `degree` $\cdot$ `netSize`/`relaySize`. The action potential of each neuron is equal to $k \cdot$ `maxSynapseStrength` (which equals `maxSynapseStrength` in regime β). We refer to this value as the *threshold* of a neuron and denote it by Θ.

Then, in the primitive layer we create `primitiveItemN` items each of size `primitiveItemSize`, which is the r value for the primitive layer. In regime α the neurons representing each item are chosen randomly and uniformly and therefore the items might overlap. In regime β the neurons are chosen randomly and uniformly from the neurons that have not been already assigned to another item. This might appear to be a

strong assumption on primitive items since in an actual biological system the neurons representing primitive items might not be distributed randomly. However this choice has little impact on the performance of the network as long as pairs of primitive items share few neurons. In particular, in regime β the choice of neurons for the primitive items does not affect the performance of the network when the primitive items are disjoint.

4 Hierarchical Memory Formation

In the hierarchical memory formation stage neurons are assigned to all items in the main layer. Each item in the main layer corresponds to a conjunction of two primitive items. The simulator creates `workingItemN` items using distinct pairs of primitive items chosen randomly and uniformly from the set of all pairs of primitive items.

For a pair of primitive items A and B, the neurons that are assigned to represent $A \wedge B$ are the neurons that have the largest number of connections to both A and B. Specifically, the mechanism for assigning the neurons is as follows. In regime β a neuron in the main layer is assigned to represent the item $A \wedge B$ if it has incoming connections from both A and B and was not previously assigned to represent another item. In regime α we explore two alternative mechanisms. In the first each of the neurons that represent either A or B is activated. Neurons in the main layer whose input potential is higher than their threshold are assigned to represent $A \wedge B$. This is equivalent to saying that neurons that have at least k incoming connections altogether from the neurons of A and B will represent $A \wedge B$.

In the second mechanism, the neurons that represent A are activated and in the next step neurons that represent B. Neurons in the main layer whose input potential is higher than their threshold in both steps are assigned to represent $A \wedge B$. This process assigns all neurons that have at least k incoming connections from the neurons of A and also at least k incoming connections from the neurons of B to represent $A \wedge B$. We refer to these two mechanisms as *one-step* and *two-step*, respectively, because they correspond to the two sources being activated concurrently versus sequentially. In both mechanisms items that share a common primitive item (i.e. $A \wedge B$ and $A \wedge C$) also share a significant fraction of neurons. This fraction is lower for two-step memory creation.

We choose the value of the parameter `primitiveItemSize` to ensure that the average size of the items created in the main layer equals the value of the parameter r. In both regimes the size of an item created in the main layer depends on several random choices (such as the choice of the neurons in A and B, and connections from these neurons). The expectation of this size is a monotone function of `primitiveItemSize`. Therefore, the desired value of `primitiveItemSize` can be found via a binary search.

5 Association, Supervised Memorization and Inductive Learning

Next we describe the three basic task types implemented within the main layer. The goal of each task execution is to modify the weights so that on future inputs some chosen *target* item becomes active for new combinations of certain other *source* items being active. Each of the four task types we have described is realized by an explicit algorithm detailed later. The simulator starts by creating a list of all the operations that need to be simulated. The total of `taskN` cognitive tasks are performed by the simulator. They are as follows:

- **Association**. An item A is associated with item B if activation of B activates A. Randomly chosen `taskN`/5 of all the `workingItemN` items are targets of association. Each of the targets is associated with 3 random other items (therefore total $3 \cdot$ `taskN`/5 associations are performed).
- **Supervised memorization**. An item A is a memorized conjunction of items $B_1, B_2, \ldots, B_\ell$ if A is activated whenever all of the B_i's are activated. Randomly chosen `taskN`/5 of all the `workingItemN`

items are targets of supervised memorization for $\ell = 2$. This might include items that were previously chosen as the target or the source of an association task. If the target of a supervised memorization task is not also the target of an association then two source items are chosen randomly from the set of all the items excluding the target itself. If this target is also the target of some association then two source items are chosen randomly from the set of all the items excluding the three sources of associations with that target item. This is necessary to avoid conflicting requirements on the activation function of the target item. Note that this does not prevent conflicts at the level of single neurons and synapses as they might be used for different functions.

- **Inductive learning**. Inductive learning is used when the desired functionality is not known in advance but correct examples of this functionality are observed. Here we will use a simple mechanism based on the Winnow algorithm (Littlestone, 1987) that allows learning of monotone linear threshold functions with significant margins. Further details of the mechanism are given in Section 5.3. Randomly chosen `taskN`/5 of all the `workingItemN` items are targets of inductive learning. Each of these target items is learned as some function of eight source items. The source items are chosen randomly from all the items that are not already used as sources for that same target item (in association or supervised memorization). Unlike the other tasks that are performed in a single operation, learning involves many steps that are not necessarily performed consecutively.

We shall now describe how each of these tasks is performed. Each of the algorithms can be easily implemented within the model of computation described in Section 2.2. We shall require the following notation in this discussion. For neurons u and v, we denote by $w_{u,v}$ the current weight on the connection from u to v; by F_v the set of all the neurons that have incoming connections to v and are currently active (that is firing), and by w_v the current total incoming potential of v, namely the sum of the connection weights of connections incoming to v from neurons in F_v.

5.1 Association

Our association algorithm is based on the LINK algorithm given by Valiant (2005). Let A be the target item and B be the source item of an association task. First, all the neurons in S_B are set to the firing state. In regime β this causes firing of all the neurons in the relay layer that have incoming connections from S_B. Now for every neuron $v \in S_A$, if $w_v \geq \alpha_1 \cdot \Theta$ then nothing is changed. Here α_1 is a constant chosen in advance. Otherwise, for every $u \in F_v$ the new weight of the connection u, v is set to be

$$w'_{u,v} = \min \left\{ w_{u,v} + \frac{\alpha_1 \cdot \Theta - w_v}{|F_v|},\ \texttt{maxSynapseStrengh} \right\}$$

rounded to the closest integer. In other words, the weight that is required to reach $\alpha_1 \cdot \Theta$ is distributed evenly among all the incoming connections from firing neurons (but is not allowed to exceed `maxSynapseStrengh`). Note that in regime α these incoming connections are from other neurons in the main layer, whereas in regime β they are from neurons in the relay layer.

The constant factor α_1 is used to compensate for situations where the source item is recognized but not all of the neurons in S_B are firing. It also compensates for some degree of interference with other task instances performed on the network. Values that were used in our simulations are $\frac{5}{4}$ for regime α and $\frac{4}{3}$ for regime β.

We note that in regime α this algorithm can be performed in a single step of neural processing at each of the neurons that represent the target item. In regime β an additional step of processing is required to activate the neurons of the relay layer.

5.2 Supervised Memorization

This algorithm is similar to the association algorithm above. Let item A be the target of a supervised memorization task and items B and C be its sources. First, all the neurons in S_B are set to the firing state (in the case of regime β causing the firing of neurons connected to them in the relay layer). Now for every neuron $v \in S_A$, if $w_v \geq \alpha_2 \cdot \Theta/2$ then nothing is changed. As before, α_2 is a constant chosen in advance. Otherwise, for every $u \in F_v$ the new weight of the connection u, v is defined to be

$$w'_{u,v} = \min \left\{ w_{u,v} + \frac{\alpha_2 \cdot \Theta/2 - w_v}{|F_v|},\ \texttt{maxSynapseStrengh} \right\}$$

rounded to the closest integer. In other words the weight that is required to reach $\alpha_2 \cdot \Theta/2$ is distributed evenly among all the incoming connections from firing neurons. Then the same operation is performed with all the neurons in S_C set to the firing state. In other words, half of the weight that is required to reach $\alpha_2 \cdot \Theta$ is distributed evenly among all the incoming connections from firing neurons when B is activated and the other half of this weight is distributed evenly among all the incoming connections from firing neurons when C is activated. We used $\alpha_2 = \frac{6}{5}$ in our simulations.

As can be easily seen from this description, in regime α supervised memorization can be performed in two steps of neural processing at each of the neurons that represent the target item. In regime β an additional step of processing is required to activate the neurons of the relay layer. We also note that a one-step overall algorithm in regime α is also possible, but would give lower capacities.

5.3 Inductive Learning

The learning process that we simulate is based on the use of examples of the correct function to attain or approximate the same function. Here an example specifies the states (recognized or not) of all the source items and the desired state of the target item. We assume that when receiving an example all the source items are in the state that is specified by the example and the correct label is given to every neuron of the target item. Upon receiving an example each neuron of the target item can update the weights on its incoming connections and thereby update the function it computes. In our model the activation function of a neuron is a linear threshold function over the states (firing or not) of the neurons from which there are incoming connections. A linear threshold function (LTF) over m variables is defined by a vector of m weights and a threshold value. We denote an LTF by $[\bar{w}, \theta]$, where $\bar{w}$ is the weight vector and θ is the threshold. For a point $\bar{x} \in \mathbb{R}^m$, $[\bar{w}, \theta](\bar{x})$ equals 1 if $\langle \bar{w}, \bar{x} \rangle = \sum_{i \leq m} w_i x_i \geq \theta$ and equals 0, otherwise.

5.3.1 Learning Algorithm

In our setting, that is, when examples are given one-by-one (also referred to as online learning) a number of algorithms for learning LTFs are known, most notably, the Perceptron algorithm and the Winnow algorithm (Rosenblatt, 1958; Minsky & Papert, 1969; Littlestone, 1987). We chose to use a variant of Littlestone's Winnow algorithm for learning LTFs with non-negative weights in our simulations. The Winnow algorithm is known to be robust to moderate amounts of noise and its performance is not affected significantly by the presence of irrelevant input variables. This makes it a natural choice for our purposes.

The Winnow algorithm starts with a certain fixed LTF as its initial hypothesis and updates it if the prediction of the hypothesis is not correct on a given example. We denote by $h^t = [\bar{w}^t, \theta]$ the hypothesis generated by the algorithm after t mistakes. When the algorithm receives an example $\bar{x}$ labeled by $b \in \{0, 1\}$. Then for every i such that $x_i = 1$, the new weight w_i^{t+1} is set to be $w_i^t \cdot \alpha^{b - h^t(\bar{x})}$ where $\alpha > 1$ is a constant chosen in advance. Notice that no changes are made if the prediction is correct.

If the examples come from a linearly separable set of examples X then the Winnow algorithm is guaranteed to converge to an LTF that correctly classifies all the examples in X. The maximum number of mistakes

that the Winnow algorithm can make until it converges depends on the maximum margin with which the data points can be separated; the larger the separation margin the fewer mistakes the Winnow algorithm will make. Formally, for a set of examples X and a linear threshold function $(\bar{w}, \theta)$ consistent with the examples, the margin of $[\bar{w}, \theta]$ on X is equal to the shortest distance from a data point in X to the hyperplane defined by $\langle \bar{w}, \bar{x} \rangle = \theta$, or

$$\frac{\min_{x \in X}\{|\langle \bar{w}, \bar{x} \rangle - \theta|\}}{||\bar{w}||} .$$

In order to use a learning algorithm in our model we need the learning algorithm to produce a hypothesis that is not only consistent with the examples but also robust to interference from other tasks and imperfect representation of items. This robustness can be improved by using an LTF that separates the examples with a "large" margin. The original Winnow algorithm does not necessarily produce an LTF with a large margin even when such an LTF exists. We therefore modify it as follows. Instead of updating the weights only on mistakes we also update them when an example is "too" close to the current separating hyperplane. Formally, for example $\bar{x}$ labeled by b, if $b = 0$ and $\langle \bar{w}^t, \bar{x} \rangle \geq \beta_1 \cdot \theta$ then for every i such that $x_i = 1$, $w_i^{t+1} = w_i^t / \alpha$. If $b = 1$ and $\langle \bar{w}^t, \bar{x} \rangle < \beta_2 \cdot \theta$ then for every i such that $x_i = 1$, $w_i^{t+1} = w_i^t \cdot \alpha$. Here $\beta_1 < 1$ and $\beta_2 > 1$ are fixed constants that depend on the maximum margin of the example set. A similar modification was previously applied to the Perceptron algorithm by Krauth and Mezard (1987).

In addition, in order to reduce the number of examples needed for learning we allow more than one update on every example. That is, if after the update on example $(\bar{x}, b)$ it is still classified incorrectly or too close to the separating hyperplane, then the current hypothesis is updated on this example again. This is done up to `reuseBound` times for a small constant `reuseBound`. Some such bound is useful here since algorithms that reuse examples many times are less robust to outliers.

Finally, the weights of the connections are integers in the range from 0 to `maxSynapseStrengh`. Therefore we limit the weights that are produced by the learning algorithm to this range and round the weights to the closest integer after each update. Rounding after an update of a low weight connection can potentially cancel the effect of the update. In such cases we decrease or increase the weight by 1 according to the direction of the original update. We remark that this is exactly the update rule in the Perceptron algorithm.

5.3.2 Target Functions and Distributions

The LTFs on which we test the learning algorithms are chosen randomly from the set of all balanced thresholds with weights $0, 1$ and 2, where by balanced we mean that the threshold is equal to the sum of all the weights divided by 2. More formally, for each $i \leq 8$ we choose a random weight $w_i \in \{0, 1, 2\}$ (equiprobably). The threshold is then set to be $\theta = \frac{1}{2} \sum_{i \leq 8} w_i$. With this LTF we use all the examples that are sufficiently far from the hyperplane $\langle \bar{x}, \bar{w} \rangle = \theta$. That is we only allow examples from the set

$$X_{\bar{w},\gamma} = \{\bar{x} \in \{0,1\}^8 \mid |\langle \bar{x}, \bar{w} \rangle - \theta| > \gamma \cdot \theta\} ,$$

for a fixed constant γ. This effectively makes the example set separable with a large margin. The learning algorithm is supplied with examples randomly and uniformly chosen from $X_{\bar{w},\gamma}$. Note that this makes the positive and the negative examples equiprobable.

5.3.3 Training

Examples are created for each instance of an inductive learning task until that task exceeds the bound on the total number of mistakes denoted by `mistakeBound` or does not encounter a mistake in the last `correctRunLength` examples. The second condition indicates that the learning process has likely achieved sufficiently high accuracy.

To feed an example $(\bar{x}, b)$ we need to set the source items into the states prescribed by the point $\bar{x}$. This implies that we need to simulate an item in the state of being recognized and in the state of being not recognized. For brevity we refer to these states as being ON and OFF states. To simulate these states we define worst case distributions on fractions that do not violate the bounds $\mathcal{C}_{OFF}$ and $\mathcal{C}_{ON}$. Specifically, for an item of size r we denote by $\mathcal{D}^r_{ON}$ the "worst" distribution on the number of neurons that are active when the item is recognized. Here by the worst we mean that smaller numbers are generated with the highest possible probability that does not violate $\mathcal{C}_{ON}$. It is easy to see that for every $r' \leq r$, the probability that exactly r' neurons are active $\mathcal{D}^r_{ON}(r') = \mathcal{C}_{ON}(\frac{r'-1}{r}) - \mathcal{C}_{ON}(\frac{r'}{r})$. Similarly, we define $\mathcal{D}^r_{OFF}$ and note that $\mathcal{D}^r_{OFF}(r') = \mathcal{C}_{OFF}(\frac{r'}{r}) - \mathcal{C}_{OFF}(\frac{r'+1}{r})$.

For an item A let r_A be the size of S_A. In order to simulate A in the ON state we choose r' randomly according to the distribution $\mathcal{D}^{r_A}_{ON}$. We then randomly choose r' neurons from S_A and set their states to firing. Similarly, in order to simulate A in the OFF state we choose a random r' according to distribution $\mathcal{D}^{r_A}_{OFF}$ and then fire random r' neurons from S_A.

After the activation of the source items according to $\bar{x}$ (and the activation of corresponding relay neurons in regime β), each of the neurons that represent the target item updates its weights according to the algorithm described above (the correct label b is available to every target neuron). The simulator also needs to determine whether an example should be considered a mistake. We consider a positive (negative) example a mistake if more than `trainingONBound` (`trainingOFFBound`) fraction of the neurons in the target item required updating when this example was fed. These constants are chosen in advance and depend on the bounding functions $\mathcal{C}_{OFF}$ and $\mathcal{C}_{ON}$.

In regime α weight updates for each example can be performed in a single step of neural processing at each of the neurons that represent the target item. In regime β an additional step of processing is required to activate the neurons of the relay layer.

5.4 Order of Execution

It is easy to see that the order in which the task instances are performed on the network influences the final state of the network. The simulator permutes randomly all the operations that it needs to perform and then performs them sequentially. Here a single operation is either an association, a supervised memorization or processing a sequence of examples for inductive learning until either a mistake is encountered or no mistakes have been made for `correctRunLength` examples. (For considerations of efficiency in regime α, the learning operation is performed until 4 mistakes - rather than just 1 - are encountered. This did not affect the results of our simulations substantially.)

Thus, for example, if `taskN` $= 250,000$ then the operations that are performed in random order are the following: 150,000 associations, 50,000 supervised memorizations, and an appropriate number of presentations of examples for 50,000 targets of inductive learning that give them a chance to converge. Note that the various presentations for each inductive learning target are also randomly permuted.

6 Testing

The main phenomenon we wish to demonstrate is that sequences of thousands of tasks can be done without undue interference with each other. Thus while the later task instances modify synaptic weights these later modifications will not interfere unduly with the performance of task instances executed earlier. This is a most basic requirement for claims of capacity. It can be verified by testing each task instance in isolation from each other, with inputs that manipulate and test only the source and target items of that task. The tests we perform are of this general nature. More advanced tests of chaining are considered in Section 6.4 which investigates whether the chaining of more than one task in the main layer introduces undue biases.

Following the completion of all the `taskN` circuit modifications, we test in worst case fashion whether they behave correctly. For each of the task instances we first test whether the target item evaluates the desired function of the source items. We then test how the evaluation is affected by the simultaneous activation of a number of irrelevant items. We also test the whole network: we activate random sets of items and test whether this causes any unintended items to be recognized. Finally, we test in worst case fashion the behavior of all chained tasks. We find all task instances for which at least one of the source items is the target item of another task instance. For each instance of such chaining we test whether the target item evaluates the function obtained by chaining the corresponding functions of individual task instances.

All our testing is done in a worst case manner in the following sense. If, for example, we are testing a condition in which an item is recognized, for some conditions it may be more onerous on the algorithms if all the neurons are active, and for other conditions it may be more onerous if the neurons are active with the minimal probabilities $\mathcal{D}^r_{ON}$ (as defined in Section 5.3). In each case we perform the testing using the more onerous distribution.

6.1 Testing Functionality

To test the functionality of a certain task the simulator tests the activity of the target item on all the permissible combinations of Boolean values, or "states" of its source items. These tests are separated into testing of the ON state and testing of the OFF state. Testing of the ON state is done by simulating all the states of the source items that should cause the activation of the target item. Each of these simulations activates a certain subset of the neurons that represent the target item. The results of all such simulations for a single task are aggregated and compared to the bounds imposed by $\mathcal{C}_{ON}$. The fraction of tests that do not satisfy $\mathcal{C}_{ON}$ is considered the ON error of this task. More formally, let A be the target of a task T and let r' be the number of neurons in S_A that fired as a result of a test of A. We add the fraction $\frac{r_1}{r_A}$ to the collection of the fractions obtained while testing the ON state for T. We denote this collection by $\Phi_{T,ON}$ and define $\mathcal{P}_{T,ON}(p)$ to be the probability that a fraction randomly and uniformly chosen from $\Phi_{T,ON}$ is greater than or equal to p. Then the ON error of the task T is simply $\max_{p\in[0,1]}\{\mathcal{C}_{ON}(p) - \mathcal{P}_{T,ON}(p)\}$.

Analogous tests are also performed for the OFF state, that is the states that should not cause the recognition of a target item. In this case we define $\mathcal{P}_{T,OFF}(p)$ to be the probability that a fraction randomly and uniformly chosen from $\Phi_{T,OFF}$ is less or equal than p. Accordingly, the OFF error is defined to be $\max_{p\in[0,1]}\{\mathcal{C}_{OFF}(p) - \mathcal{P}_{T,OFF}(p)\}$.

For each type of task average ON and OFF errors are calculated.

6.1.1 Association

The following tests are performed for each association task. Let A be the target item and B be the source item of a certain association task. Item A has to be recognized when item B is recognized. To simulate the activation of B we use $\mathcal{C}_{ON}$ – the lower bound on the number of firing neurons when an item is recognized. That is, as in Section 5.3.3, the simulator chooses a random r' according to the distribution $\mathcal{D}^{r_B}_{ON}$. Then it activates r' neurons randomly chosen from S_B. We refer to such an activation as a *random ON state*. The fraction of the neurons in S_A that fired as a result of this test is added to the collection of ON fractions for this association task. This test is repeated `testRepeat` times to test the function for different random ON state activations.

The testing of the OFF state is analogous. The item A should not be activated when B is not activated. To simulate the OFF state of B the simulator chooses a random r' according to the distribution $\mathcal{D}^{r_B}_{OFF}$. Then it activates r' neurons randomly chosen from S_B. We refer to such an activation as a *random OFF state*. The fraction of the neurons in S_A that fired as a result of this test is added to the collection of OFF

fractions for this association task. This test is repeated `testRepeat` times (for different random OFF state activations).

Given the results of these tests, ON and OFF errors are estimated for each of the tasks.

6.1.2 Supervised Memorization

The testing of supervised memorization task instances is done similarly to the testing of association task instances. Let A denote the target item of a supervised memorization task and let B and C denote its source items. Item A has to be recognized when items B and C are recognized. Therefore to test the ON state of A, the simulator activates random ON states of B and C. It then adds the fraction of the neurons in S_A that fired as a result of this activation to the collection of ON fractions for this supervised memorization task. This test is repeated `testRepeat` times.

The item A should not be activated if at least one of the source items is not activated. Therefore to test the OFF state of A, the simulator activates a random OFF state of item B and fires all the neurons in S_C. All the neurons in S_C are fired (and not a random ON state of C) since we need to use an upper bound on the number of firing neurons, whereas a random ON state gives a lower bound. The same test is then performed with the roles of B and C reversed. These tests are repeated `testRepeat` times. The tests with both B and C being in the OFF state are not performed since correct performance on these tests is implied by the correct performance in the above OFF state tests.

6.1.3 Inductive Learning

Let A be the target item of an inductive learning task, let $B_1, B_2, \ldots, B_8$ be the source items, and let $[\bar{w}, \theta]$ be the LTF on which A was trained. To test the ON state of A, the simulator tests it on each example $\bar{x} \in X_{\bar{w},\gamma}$ such that $[\bar{w}, \theta](\bar{x}) = 1$. To perform this test the simulator activates a random ON state of each B_i such that $x_i = 1$ and adds the fraction of the neurons in S_A that fired as a result to the collection of ON fractions for this learning task. Note that B_i's for which $x_i = 0$ are not activated at all since we need to use a lower bound on the number firing source neurons in ON tests.

To test the OFF state the simulator tests A on each example $\bar{x} \in X_{\bar{w},\gamma}$ such that $[\bar{w}, \theta](\bar{x}) = 0$. To perform this test, the simulator activates a random OFF state for each B_i such that $x_i = 0$ and fires all the neurons in S_{B_i} for each B_i such that $x_i = 1$ (thereby producing an upper bound on the number of firing source neurons).

6.2 Testing Robustness to Irrelevant items

In addition to testing functionality when only the source items of a task are activated, the simulator also tests functionality when other irrelevant items are activated at the same time. We consider an item B irrelevant for an item A if B is not a source item for any of the task instances for which A is the target item. We first observe that since all the weights in the network are positive, irrelevant items can only increase the number of neurons that fire during an execution of a task. This implies that only the OFF error can increase and therefore only the OFF state tests are performed in the presence of irrelevant items.

For each of the task instances the simulator performs OFF state tests again while activating a number of irrelevant items. Specifically, for an association task a random OFF state of its source item is activated. For a supervised memorization task one of the source items is activated entirely and the other is activated in a random OFF state (and then vice versa). For a learning task a random OFF state point $\bar{x} \in X_{\bar{w},\gamma}$ is chosen (that is $[\bar{w}, \theta](\bar{x}) = 0$) and the source items are activated according to $\bar{x}$ as in the OFF state tests. Then the simulator activates all the neurons in a randomly chosen irrelevant item. The fraction of neurons in the target item that fired as a result is recorded. Now the neurons of another randomly chosen irrelevant item are set to the firing state in addition to the neurons that were already activated. The new fraction of

neurons in the target item that fired as a result is recorded. This is continued until ℓ irrelevant items are added for some number ℓ that depends on the type of the task tested. For each task this test is repeated `irrelevantRepeat` times. For each task and each number of added irrelevant items, the resulting fractions are aggregated into a separate collection. The OFF error is then calculated for each of these collections. This error measures the robustness of the task to a specific number of active irrelevant items.

6.3 Robustness of the Whole Network

We would also like to ensure that activity of items in the main will not cause activity in any item that is totally unrelated to them through any of the task instances executed. (The tests above all fired at least one item that was related to one of the task instances.) To test this property, for a number ℓ, the simulator performs the following test `wholeNetworkTests` times. It chooses randomly ℓ items $B_1, B_2, \ldots, B_\ell$ and activates all the neurons that represent them. Then for every item A such that none of the B_i's is a source in any of the performed task instances, it records the fraction of neurons in S_A that fired as a result. All these fractions are aggregated in a separate collection for each item A and number ℓ. The OFF error is then calculated for each of these collections. The *sum of the OFF errors for all the items equals the expected number of items that violate the OFF bound*. This statistic is referred to as the *total OFF error with ℓ irrelevant items*.

6.4 Chaining of Tasks

The main goal of our experiments is to estimate the capacity of networks by examining the performance of and interference among the individual task instances realized by the network. However the network model, the semantics of item representations, and the task algorithms are all designed to also support the chaining of task executions in a hierarchical manner. To provide support for the possibility of such chaining we also tested the performance of all chained main layer tasks within our simulation design. We note that all the items in the main layer tested here are created by hierarchical memory formation from pairs of items in the primitive layer, and in that sense the simulations are consistent with chaining starting from activation of the primitive layer alone, which is the mode of operation ultimately intended.

The chaining of tasks can introduce two new sources of error. The first one comes from the fact that if a task instance T_2 is chained with a task instance T_1 then the inputs to T_1 are no longer truly random ON or OFF states (as in our tests), since biases may be introduced among the neurons that fire in the target item of T_2. The other source of error comes from the fact that the activation of the source items of T_2 might cause firing of neurons that do not represent the target of T_2. These *irrelevant* firing neurons might interfere indirectly with the functionality of T_1. In our tests we examine both of these sources of error.

6.4.1 Test Design

Let A be the target item of task instance T_1 and let $B_1, B_2, \ldots, B_\ell$ be the source items of T_1. Further we assume that one or more of the source items are themselves target items of some task instances. For concreteness let us assume that B_1 is the target item of task instance T_2 with source items $C_{1,1}, C_{1,2}, \ldots, C_{1,\ell}$, B_2 is the target item of task instance T_3 with source items $C_{2,1}, C_{2,2}, \ldots, C_{2,\ell}$ and $B_3, B_4, \ldots, B_\ell$ are not targets of any task instances. We refer to T_1 as *root* task instance and to T_2 and T_3 as *level-two* task instances.

As a result of such chaining, item A realizes a function of items $C_{1,1}, C_{1,2}, \ldots, C_{1,\ell}$; $C_{2,1}, C_{2,2}, \ldots, C_{2,\ell}$ and $B_3, B_4, \ldots, B_\ell$. In order to test if A behaves correctly when the input items are activated we examine two possible ways to time the activation of the input items.

- *Parallel.* All source items of level-two task instances are activated simultaneously, that is the neurons representing items $C_{1,1}, C_{1,2}, \ldots, C_{1,\ell}$ and $C_{2,1}, C_{2,2}, \ldots, C_{2,\ell}$ are activated at the same time according

to the input that is tested. This causes firing of some of the neurons representing B_1 and B_2 as well as possibly some other neurons in the next step. We denote the set of neurons that fired as a result by S. Then, in addition to the neurons in the set S, the neurons in $B_3, B_4, \ldots, B_\ell$ are activated according to the tested input. In the following time step this causes firing of some of the neurons of A. The fraction of A's neurons that are firing is used to calculate the error of the function realized by A in the same way as for one-level functions.

- *Sequential*. Source items of level-two task instances are activated one after the other, that is, first the neurons in items $C_{1,1}, C_{1,2}, \ldots, C_{1,\ell}$ are activated and then the neurons in items $C_{2,1}, C_{2,2}, \ldots, C_{2,\ell}$ are activated (according to the tested input). Let S_1 and S_2 denote the sets of neurons that fire as a result of activating the sources of T_2 and T_3 respectively. In the next time step, in addition to the neurons in the sets S_1 and S_2, the neurons in $B_3, B_4, \ldots, B_\ell$ are activated according to the tested input. In the following time step this causes firing of some of the neurons representing A. As in the previous case, the fraction of A's neurons that are firing can be used to derive the error rates of this instance of chaining.

The difference between these two methods is that in the parallel one task instances T_2 and T_3 may interfere causing the firing of some neurons that would not fire if sources of just one of the two task instances were activated. Note that because of the monotonicity of all the functions realized such interference only increases the OFF error (and might decrease the ON error). On the other hand, the sequential method assumes a more delicate timing of the input item activations.

In order to estimate the ON and the OFF errors of chained task instances we test the behavior of the target item on random inputs to the function realized by the chained task instances in a fashion analogous to our one-level tests. To create a random setting of all the inputs in the ON error test we first choose a random Boolean setting of the ℓ inputs to T_1 such that the Boolean function realized by T_1 equals to 1 on this setting. We denote this setting by $v_1, v_2, \ldots, v_\ell$. For example, if T_1 is a supervised memorization task then $\ell = 2$ and the only setting of inputs for which supervised memorization equals 1 is $v_1 = 1, v_2 = 1$. For each input of T_1 that itself is the target item of a task instance, say T_2, we choose a random Boolean setting of inputs to T_2 that would give the desired value of the target item of T_2. For example if $v_1 = 1$ and the first input to T_1 is the target of a learning task instance T_2 then a random Boolean setting of inputs to T_2 that causes the threshold to be equal to 1 is chosen. If a certain input of T_1 is the target of more than one task instance then one of the chained task instances is chosen randomly. Given a setting of all the inputs, we activate the neurons of the corresponding items in the same conservative way as in the one-level tests: if the input is supposed to be 1 then the random ON state of the corresponding source item is activated; otherwise none of the neurons in the corresponding item are activated. These activations are fed to the network as defined by either parallel of sequential testing method and the fraction of A's neurons that are firing is recorded. For each chained task instance this test is repeated 25 times (as in the tests of robustness to irrelevant item activations) and ON error is calculated as described in Section 6. We refer to the error obtained in this way as the *chained ON error*.

We estimate the OFF error of chained tasks in an analogous way. In addition, we estimate the influence of the irrelevant firing neurons on the error. To do this we count the number of neurons that were caused to fire by the source items of the level-two task instances (T_2 and T_3 in the example above) but are not representing the target items of the level-two task instances (B_1 and B_2 in the example above). The magnitude of such irrelevant firing is measured in item equivalents, that is we divide the number of neurons by r and round to the closest integer. Using this value we aggregate the firing fractions of the target item by the magnitude of irrelevant firing (in items). For magnitude m we refer to the error obtain in this way as the *chained OFF error with m irrelevant items*. We use the same term as in the one-level error tests since in both cases we measure the effect that activity outside of the source items has on the OFF error of a task instance. However, unlike in the one-level tests where active irrelevant items are added randomly, in chaining tests the irrelevant

activity is inherent in the test since it is caused by the source items of the level-two task instances. For each chained task instance this test is repeated 75 times.

We test ON and OFF errors of all the two-level chained task instances that were formed by the task instances executed in the main layer. Given all the errors obtained in this way we calculate the average errors for each type of the root task instance.

7 Results

We now summarize the results of simulations for each of the two regimes. We emphasize that our main goal was to demonstrate that high capacities, namely high values of `taskN`, could be successfully simulated with low error rates for each of the various measures of error. For each of regimes alpha and beta we achieved the reported capacities by carefully choosing the values of the controllable parameters. Each such choice of parameters is reported as a column in a table. The values in the top block describe the values of the 11 parameters chosen, and the entries below state the error rates measured for each of the four types of error detailed in Section 6. The effect of the 11 parameters on the various error measures is highly complex and we do not attempt here to generalize about it beyond informal statements. The goal is to demonstrate that high capacities are achievable with combinations of reasonable parameter values.

7.1 Regime α

In our base setting of regime α there are 250,000 neurons in both the primitive and the main layers, and $d = 8,000$. We found that the average r that supports the highest capacity while maintaining relatively low errors is around 898. A total of 3,200 items of such average size were created using (the simpler) one-step memory formation on random pairs of 1,600 primitive items. Note that each neuron in the main layer is shared by more than 11 (i.e. 3,200*898/250,000) items on average. In Table 1 we describe all the parameters of this base setting along with brief descriptions. Our main tests show that sequences of 2,000 task executions can be supported with low average error in the functionality that those executions aim to realize, even in the presence of some irrelevant items firing. Further more than 99.9% of the time, simultaneous firing of up to 8 items will cause no spurious unrelated activity anywhere in the network.

In Table 2 we describe the results of regime α simulations along with the parameters that were modified from the base regime α setting.

- **AVG, STDEV:** This column contains the averages of each of the tested error measurements over 10 simulations of networks with the base parameters described in Table 1. Specifically, 10 random networks with the base setting of parameters are created, `taskN` = `2000` task instances are executed and then the network is tested (as detailed in Section 6). For each error measurement, the average value for these 10 simulations is reported. In the second column ("STDEV") we give the standard deviations for these 10 simulations to illustrate the general level of variation.

- **MLN:** In this setting the number of neurons in each layer is $1,000,000$ while d and k are the same as in the base setting. In order to support high capacity with low interference average r is chosen to be about 3,581. Testing of this network with the same number of task instances as in the base setting produced slightly lower general level of errors than in the base setting. This suggests that the base setting becomes more stable when it is scaled up. However, as we will elaborate in the next section, in regime α capacity is largely determined by the value of d/k and cannot be increased significantly by merely increasing the number of neurons n.

- **T1000:** In this setting we construct the same base network but execute fewer task instances on it. Specifically, `taskN` = $1,000$. Predictably, the errors resulting from interference between the tasks are

Name	Value	Description
`netSize`	250,000	number of neurons in the main layer
`primitiveNetSize`	250,000	number of neurons in the primitive layer
d	8,000	number of connections to and from neurons in the main and relay layers and from the primitive layer to the main layer
k	16	inverse of the maximum influence of a neuron on its neuron
r	898	average size of items in the main layer
`primitiveItemSize`	116	size of items in the primitive layer
`maxSynapseStrength`	200	maximum weight of a connection (corresponding to synaptic strength)
`primitiveItemN`	1,600	number of primitive items
`workingItemN`	3,200	number of items in the main layer
`taskN`	2,000	total number of task instances performed
$\mathcal{C}_{ON}(p)$	$B[0.98, 0.88, -0.01](p)$	lower bound on the distribution of fractions when an item is active
$\mathcal{C}_{OFF}(p)$	$B[0.05, 0.3, 0.025](p)$	upper bound on the distribution of fractions when an item is inactive
α_1	5/4	association compensating factor
α_2	6/5	supervised memorization compensating factor
α	4/3	Winnow multiplicative factor
β_1	4/5	Winnow negative margin
β_2	5/4	Winnow positive margin
γ	2/5	target LTF margin
`mistakeBound`	20	bound on the number of mistakes done while learning an LTF
`reuseBound`	3	maximum number of updates on an example
`correctRunLength`	50	bound on the number of examples classified without a single mistake
`trainingONBound`	0.98	a positive example is considered a mistake if more than this fraction of target's neurons required updating
`trainingOFFBound`	0.05	same as above for negative examples
`testRepeat`	200	number of times association and supervised memorization functionality tests are repeated for each task
`irrelevantRepeat`	25	number of times each task is tested with irrelevant items
`wholeNetworkTests`	200	number of times the whole network is tested for undesired activations

Table 1: Parameters of the base regime α simulation

Param/Test name	AVG	STDEV	MLN	T1000	T3000	D1000	2sD1000	2-step	10mst	$k = 16/5$	PRIM
`netSize`	250,000	250,000	1,000,000	250,000	250,000	250,000	250,000	250,000	250,000	250,000	250,000
`primitiveNetSize`	250,000	250,000	1,000,000	250,000	250,000	250,000	250,000	250,000	250,000	250,000	50,000
d	8,000	8,000	8,000	8,000	8,000	8,000	8,000	8,000	8,000	8,000	8,000
k	16	16	16	16	16	16	16	16	16	16/5	16
`primitiveItemSize`	116	116	458	116	116	116	324	324	116	40	116
`primitiveItemN`	1,600	1,600	1,600	1,600	1,600	1,000	1,000	1,600	1,600	10,000	1,600
memory formation	1-step	1-step	1-step	1-step	1-step	1-step	2-step	2-step	1-step	2-step	1-step
average r	898	898	3581	898	898	898	893	893	898	369	898
`workingItemN`	3,200	3,200	3,200	3,200	3,200	3,200	3,200	3,200	3,200	20,000	3,200
`taskN`	2,000	2,000	2,000	1,000	3,000	2,000	2,000	2,000	2,000	12,500	2,000
`mistakeBound`	20	20	20	20	20	20	20	20	10	20	20
Assoc ON	0.0103	0.0014	0.0076	0.0031	0.0297	0.0317	0.0275	0.0104	0.0036	0.0201	0.0126
Assoc OFF	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Assoc OFF, 4 irrel.	4.6E-6	9.2E-6	0.0	0.0	2.4E-5	2.6E-5	0.0	0.0	0.0	0.0	0.0
Assoc OFF, 5 irrel.	7.6E-6	1.1E-5	0.0	0.0	2.2E-4	1.6E-4	0.0	0.0	0.0	0.0	0.0
Assoc OFF, 6 irrel.	4.3E-5	3.5E-5	1.3E-5	0.0	0.0024	0.0011	0.0	0.0	5.2E-5	0.0	1.5E-4
Assoc OFF, 7 irrel.	3.7E-4	1.2E-4	5.4E-4	0.0	0.1595	0.0104	1.7E-4	0.0	4.2E-4	2.0E-6	0.0019
Assoc OFF, 8 irrel.	0.0042	8.6E-4	0.0060	0.0	0.6589	0.1427	0.0193	2.1E-4	0.0045	4.2E-5	0.041
Sup.mem ON	0.0057	0.0039	0.0049	5.3E-5	0.0062	0.0078	0.0068	0.004	5.4E-4	7.4E-4	0.0025
Sup.mem OFF	5.4E-4	5.9E-4	8.3E-4	0.0	0.0	0.0013	0.0	0.0	0.0	0.0	0.0
Sup.mem OFF, 1 irrel	6.6E-4	6.2E-4	8.9E-4	0.0	0.0014	0.0018	0.0	0.0	7.1E-4	0.0044	1.3E-4
Sup.mem OFF, 2 irrel	0.0046	0.0014	0.0048	0.0	0.1271	0.0186	0.0070	0.002	0.0224	0.0503	0.0079
Sup.mem OFF, 3 irrel	0.1248	0.0105	0.1405	2.4E-4	0.7143	0.2773	0.1961	0.0722	0.2472	0.1933	0.2414
Sup.mem OFF, 4 irrel	0.6294	0.0088	0.6498	0.0068	0.9768	0.8031	0.7149	0.5073	0.6980	0.6007	0.7856
Learn ON	0.0274	0.0034	0.0219	0.0170	0.0249	0.0293	0.0194	0.0259	0.0947	0.0566	0.0258
Learn OFF	0.0148	0.0017	0.0118	0.0070	0.0210	0.0198	0.0150	0.0144	0.0558	0.0274	0.0173
Learn OFF, 1 irrel.	0.0317	0.0026	0.0313	0.0149	0.0561	0.0482	0.0359	0.0281	0.0968	0.0525	0.0369
Learn OFF, 2 irrel.	0.0620	0.0043	0.0563	0.0219	0.1285	0.0963	0.0662	0.0534	0.1550	0.0896	0.073
Learn OFF, 3 irrel.	0.1212	0.0073	0.1067	0.0315	0.2827	0.1911	0.1295	0.1037	0.2385	0.1491	0.1576
Learn OFF, 4 irrel.	0.2275	0.0102	0.2139	0.0476	0.4949	0.3415	0.2508	0.1928	0.3550	0.2365	0.2934
Total OFF, 4 irrel.	0.0014	0.003	0.0	0.0	0.0017	0.0017	0.0	0.0	0.0	0.0	0.0
Total OFF, 5 irrel.	0.0013	0.002	0.0	0.0	0.0021	0.0044	0.0	0.0	0.0	0.0	5.3E-4
Total OFF, 6 irrel.	3.9E-4	6.8E-4	9.0E-4	0.0	0.0056	0.0069	0.0	0.0	0.0014	0.0	0.0053
Total OFF, 7 irrel.	0.0027	0.0024	7.0E-4	0.0	0.5711	0.1028	0.0	0.0	0.0	0.0	0.0051
Total OFF, 8 irrel.	0.0098	0.0077	0.0041	0.0018	148.56	1.0914	0.0	0.0	0.0	0.0	0.0149
Total OFF, 9 irrel.	0.1656	0.2038	0.853	0.0	1110.7	65.817	4.4094	0.0	0.2069	0.0	12.46
Total OFF, 10 irrel.	28.056	5.3671	41.292	0.0	2248.7	350.81	120.20	1.836	30.348	0.0	167.85

Table 2: Results of regime α simulations .

significantly lower than in the base setting.

- **T3000:** In this setting we construct the same base network and execute $\texttt{taskN} = 3,000$ task instances on it. As can be seen from the results this increases the interference, especially errors resulting from the activation of irrelevant items. In particular, this demonstrates that $\texttt{taskN} = 2,000$ is close to the capacity if one wants to ensure correct performance of the network when 8 or more irrelevant items are activated.

- **D1000:** In this setting we examine the influence of the number of primitive items on the performance of a circuit. When fewer primitive items are used to create items in the main layer, more items in the main layer share a primitive item. Such items also share more neurons causing higher interference in functionality. In this setting we use 1,000 primitive items (versus 1,600 in the base setting) and as a result errors are higher.

- **2sD1000:** In this setting, as in the previous one, only 1,000 primitive items are used in the memory formation but with the two-step memory formation mechanism used in place of the one-step mechanism (see Section 4 for details). In the two step mechanism, items in the main layer that share a primitive item share a smaller fraction of neurons than in the one-step memory creation mechanism. Therefore the errors in this test are lower than in D1000 and are similar to those of the base setting.

- **2-step:** This is the same setting as the base one but with the two-step memory formation. According to the results of the simulation, the errors in this setting are observably lower than in the base setting.

- **10mst:** In this setting we test learning from fewer examples. Specifically, at most 10 mistakes per every learning task instance are allowed (versus 20 in the base setting). This increases the average error of inductive learning from about 2% to about 7.5%. Errors of other tasks are lower because each learning task instance now causes less interference with other task instances.

- **k=16/5:** In this setting we test a network with stronger synapses and the more robust two-step memory formation. Specifically, we use $k = 16/5$ in place of $k = 16$. The results of this simulation demonstrate, that for an appropriately chosen average r, 12,500 task executions on 20,000 main layer items can be supported without significant degradation in performance of association and learning tasks and with significantly improved robustness to unrelated activity anywhere in the network. This shows that the maximum strength of synapses has a significant impact on the capacity of a network.

- **PRIM:** In this setting the primitive layer contains 50,000 neurons (versus 250,000 in the base setting). The errors are similar to the base setting. This suggests that our simplifying assumption that the sizes of primitive layer and the main layer are equal is not necessary to achieve the claimed capacities. Note that the smaller size of the primitive layer implies that fewer connections are incoming to neurons of the main layer. Specifically, the expected incoming degree of each neuron in this setting is $8,000 + 8,000 * 50,000/250,000 = 9,600$ whereas in the base setting it equals $16,000$.

7.1.1 Chaining

In Table 3 we present the results of testing the functionality of chained tasks. The first column describes the type of error that is measured. Then for each task type of the root task instance, we give three versions of that error. The first is the error of the task without any chaining. The second (Ch S) is the error of the task when chained with other tasks and the sequential method of testing the chained tasks is used (as detailed in Section 6.4.1). Similarly, the third one (Ch P) gives the error of the task when chained with other tasks and the parallel method of testing the chained tasks is used. For association these two methods are identical (since it has one input) and therefore only one column is given. Some entries are left blank since

Error/Test type	Assoc	Assoc Ch S/P	Assoc Freq	SMem	SMem Ch S	SMem Ch P	SMem Freq	Learn	Learn Ch S	Learn Ch P	Learn Freq
ON	0.0111	0.006		0.0098	0.007	0.0063		0.0208	0.0124	0.0104	
OFF with 0 irrel.	0.0	0.0	0.921	0.0	0.0	6.9E-4	0.814	0.0169	0.0165	0.0168	0.601
OFF with 1 irrel.	0.0	3.0E-4	0.442	6.1E-5	0.0089	0.0136	0.647	0.0354	0.0309	0.0397	0.691
OFF with 2 irrel.	0.0	0.0	0.184	0.0018	0.0201	0.0272	0.409	0.0641	0.0518	0.086	0.542
OFF with 3 irrel.	0.0	–	0.078	0.1256	0.096	0.1468	0.211	0.1256	0.1294	0.1906	0.369
OFF with 4 irrel.	0.0	–	0.005	0.6407	0.3348	0.5454	0.101	0.2365	0.1817	0.4077	0.22

Table 3: Results for tests of chained tasks in regime α base setting.

very few tests with that number of active irrelevant items were available. Finally, in column "Freq" for each magnitude of irrelevant activity in OFF tests, the table provides the fraction of task chaining instances in which that magnitude of irrelevant activity occurred. Note that different magnitudes of irrelevant activity can occur when testing a single instance of task chaining since numerous tests are performed for each instance (75 in this case). Therefore the fractions do not sum up to 1.

Given the results we first observe that for all tasks the ON error of chained tasks is lower than for one-level tests. The likely explanation for this is that active irrelevant items compensate for any deterioration that results from imperfect firing of the source items. Our second observation is that for the sequential method of testing the errors of chained tasks the OFF errors obtained are (on average) within the errors of individual task instances. This suggests that no significant deterioration is caused by the bias in the source items. Finally, we observe that for the parallel method of testing the OFF errors obtained are observably higher. This implies that the interference caused by the simultaneous activation of all source items of level-two task instances is quite significant.

7.2 Regime β

In our base setting for regime β the primitive layer neurons, the relay neurons and non-relay main layer neurons each number 20,000,000, and $d = 2,400$. The average r was chosen to be 50. 100,000 main layer items are created from 20,000 primitive items each of size 17. Note that effectively only 340,000 (=17*20,000) neurons in the primitive layer are used and approximately 5,000,000 (=50*100,000) in the main layer. 100,000 task instances are executed on the main layer items. Note that in this regime we allow higher density of memory formation (that is the number of main layer items per primitive item) and relations in the main layer (that is `taskN/workingItemN`) than in regime α. The general level of errors achievable in regime β tests is also significantly lower than in regime α. In Table 4 we describe all the parameters of this base setting along with brief descriptions.

In Table 5 we describe the results of regime β simulations along with the parameters that were modified from the base regime β setting.

- **BASE:** These are the results of a single simulation performed with the base setting of the parameters.
- **1/10, STDEV:** To obtain an estimate of the error variation in the base setting we performed 10 tests of this setting with 10,000 task instances performed on 10,000 items chosen randomly from the 100,000 created by memory formation. For each error measurement the average value for these 10 simulations and the standard deviation are reported.

  Here and in the other tests in regime β extensive testing of all 100,000 concepts was beyond our computational resources. We note that while fewer task instances are performed, the density of the relations created between the items is preserved. This informal explanation and the simulations show

Name	Value	Description
n	20,000,000	number of neurons in the primitive, main, and relay layers
d	2,400	number of connections to and from neurons in the main and relay layers and from the primitive layer to the main layer
r	49	average size of items in the main layer
`primitiveItemSize`	14	size of items in the primitive layer
`maxSynapseStrength`	500	maximum weight of a connection (corresponding to synaptic strength)
`primitiveItemN`	20,000	number of primitive items
`workingItemN`	100,000	number of items in the main layer
`taskN`	100,000	total number of task instances performed
$\mathcal{C}_{ON}(p)$	$B[.99, 0.89, -0.01](p)$	lower bound on the distribution of fractions when an item is active
$\mathcal{C}_{OFF}(p)$	$B[0, 0.25, 0.025](p)$	upper bound on the distribution of fractions when an item is inactive
α_1	4/3	association compensating factor
α_2	6/5	supervised memorization compensating factor
α	4/3	Winnow multiplicative factor
β_1	4/5	Winnow negative margin
β_2	6/5	Winnow positive margin
γ	1/3	target LTF margin
`mistakeBound`	20	bound on the number of mistakes done while learning an LTF
`reuseBound`	3	maximum number of updates on an example
`correctRunLength`	50	bound on the number of examples classified without a single mistake
`trainingONBound`	0.98	a positive example is considered a mistake if more than this fraction of target's neurons required updating
`trainingOFFBound`	0.05	same as above for negative examples
`testRepeat`	200	number of times association and supervised memorization functionality tests are repeated for each task
`irrelevantRepeat`	25	number of times each task is tested with irrelevant items
`wholeNetworkTests`	200	number of times the whole network is tested for undesired activations

Table 4: Parameters of the base regime β simulation

Params/Test	BASE	1/10	STDEV	NetX5	d=4000	T5000	T15000	10mst	K=4
n	$2 \cdot 10^7$	$2 \cdot 10^7$	$2 \cdot 10^7$	10^8	10^8	$2 \cdot 10^7$	$2 \cdot 10^7$	$2 \cdot 10^7$	$2 \cdot 10^7$
d	2,400	2,400	2,400	2,400	4,000	2,400	2,400	2,400	2,400
`primitiveItemSize`	14	14	14	72	26	14	14	14	17
`primitiveItemN`	20,000	20,000	20,000	20,000	50,000	20,000	20,000	20,000	14,000
average r	49	49	49	252	100	49	49	49	74
`workingItemN`	100,000	10,000	10,000	100,000	250,000	10,000	10,000	10,000	10,000
`taskN`	100,000	10,000	10,000	100,000	250,000	5,000	15,000	10,000	10,000
`mistakeBound`	20	20	20	20	20	20	20	10	20
Assoc. ON	0.0014	0.0012	2.9E-4	3.8E-5	6.3E-5	7.1E-4	0.0015	5.4E-4	1.9E-4
Assoc. OFF	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Assoc. OFF with 4 irrel.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Assoc. OFF with 5 irrel.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Assoc. OFF with 6 irrel.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Assoc. OFF with 7 irrel.	5.0E-7	2.0E-7	6.3E-7	0.0	0.0	0.0	2.0E-6	2.0E-6	0.0
Assoc. OFF with 8 irrel.	2.5E-6	9.0E-7	1.7E-6	0.0	0.0	0.0	4.0E-6	5.0E-6	1.1E-5
Assoc. OFF with 9 irrel.	7.4E-6	1.1E-5	8.3E-6	5.3E-7	0.0	0.0	2.0E-5	3.2E-5	9.9E-5
Assoc. OFF with 10 irrel.	2.7E-5	3.5E-5	1.6E-5	5.4E-5	0.0	0.0	5.1E-5	1.5E-4	0.0016
Assoc. OFF with 11 irrel.	1.5E-4	1.7E-4	5.5E-5	6.5E-4	0.0	1.8E-5	3.7E-4	6.7E-4	0.0089
Assoc. OFF with 12 irrel.	8.1E-4	8.4E-4	1.6E-4	0.0029	0.0	1.7E-4	0.0017	0.0023	0.0271
Sup. mem. ON	0.0016	0.0022	7.7E-4	5.9E-5	7.3E-5	0.0031	0.0021	0.0029	1.3E-5
Sup. mem. OFF	0.0044	0.0042	7.5E-4	0.0048	0.0011	0.0023	0.0056	0.0092	7.5E-4
Sup. mem. OFF, 1 irrel.	0.0051	0.0048	9.1E-4	0.0056	0.0013	0.0027	0.0074	0.0104	0.0018
Sup. mem. OFF, 2 irrel.	0.0067	0.0062	0.0011	0.0077	0.0018	0.0033	0.0109	0.0125	0.0037
Sup. mem. OFF, 3 irrel.	0.0133	0.0129	0.0012	0.0214	0.0025	0.0057	0.0227	0.0207	0.0201
Sup. mem. OFF, 4 irrel.	0.0391	0.0391	0.0018	0.0578	0.0039	0.0157	0.0647	0.0476	0.0752
Sup. mem. OFF, 5 irrel.	0.0788	0.0788	0.0025	0.1028	0.0097	0.0313	0.1255	0.0867	0.1341
Sup. mem. OFF, 6 irrel.	0.1167	0.1165	0.0031	0.1434	0.027	0.0494	0.1806	0.1237	0.1738
Learning ON	0.0142	0.0146	0.0012	0.0176	0.0125	0.0134	0.0145	0.0732	0.0099
Learning OFF	0.0093	0.0092	7.7E-4	0.0084	0.0062	0.0076	0.0093	0.0488	0.0063
Learning OFF, 1 irrel.	0.0147	0.0146	8.0E-4	0.0133	0.0094	0.0108	0.0163	0.064	0.0127
Learning OFF, 2 irrel.	0.02	0.0199	9.4E-4	0.0183	0.0115	0.0148	0.024	0.0758	0.0195
Learning OFF, 3 irrel.	0.0276	0.0277	0.0013	0.0257	0.0146	0.0193	0.0356	0.0895	0.0311
Learning OFF, 4 irrel.	0.0385	0.039	0.0015	0.0365	0.0185	0.0257	0.0525	0.1051	0.0506
Learning OFF, 5 irrel.	0.0529	0.0537	0.0018	0.0512	0.0236	0.0337	0.0749	0.1224	0.0772
Learning OFF, 6 irrel.	0.0698	0.0709	0.0021	0.0694	0.0302	0.0443	0.101	0.1405	0.106
Total OFF, 8 irrel.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Total OFF, 9 irrel.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Total OFF, 10 irrel.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Total OFF, 11 irrel.	0.0662	4.6E-4	0.0015	0.1715	0.0	0.0	0.0407	0.0129	5.9248
Total OFF, 12 irrel.	0.4558	0.062	0.0778	7.0786	0.0	0.0	0.1447	0.4218	23.428
Total OFF, 13 irrel.	7.7367	0.7421	0.1977	49.791	0.0	0.0	1.9505	2.5636	71.412
Total OFF, 14 irrel.	51.43	5.3508	0.6382	158.29	0.0	0.4395	17.442	9.2672	171.3
Total OFF, 15 irrel.	166.49	16.92	1.0568	401.26	0.0	1.4686	50.501	26.385	294.72
Total OFF, 16 irrel.	348.02	35.849	1.5714	843.4	0.0	5.4011	101.6	51.77	437.78

Table 5: Results of regime β simulations .

that the results obtained in this way closely approximate the results of testing 100,000 task instances on 100,000 items. It should be noted that "Total OFF error" measures the expected number of items that violate the OFF error bound and therefore scales linearly with the number of items. In particular, "Total OFF error" of the base setting is approximately 10 times larger than "Total OFF error" when only 10,000 items are examined.

- **NetX5:** In this setting the number of neurons n and sizes of items are increased fivefold, that is $n = 100,000,000$ and average r is about 252. This allows us to examine how the errors are influenced by scaling up the network. According to the results of the simulation, the overall level of errors is similar to the base setting. The scaling has decreased ON errors of association and supervised memorization but increased total OFF errors.
- **d=4000:** In this setting we show that even higher capacities are achievable in a network with higher degree of connectivity. Specifically, for $n = 100,000,000$, $d = 4,000$ and average $r = 100$ we obtain capacity as high as 250,000 task executions on 250,000 main layer items. In addition, all errors are significantly lower than in the base and NetX5 settings.
- **T5000:** In this setting the base network is the same as in "1/10" but only 5,000 task instances are executed (versus 10,000 in the "1/10" setting). The errors resulting from interference between the tasks are observably lower than in the base setting.
- **T15000:** This setting is the base as "1/10" setting but with 15,000 task instances executed on 10,000 items. According to the simulations, errors of all tasks are similar to the base setting but the whole network is significantly less robust to activations of irrelevant items.
- **10mst:** In this setting we test learning from fewer examples. Specifically, at most 10 mistakes per every learning task instance are allowed (versus 20 in the base setting). This increases the average error of learning from about 1.2% to about 6.1%.
- **k=4:** The goal of this setting is to show that the strongest "$k = 1$" synapses are only required for connections incoming into the relay neurons. We restricted the strength of the connections incoming to all target neurons to be at most 1/4 of the neurons' threshold value. In order to achieve this the average r needs to be increased from 49 to 74 and, correspondingly, `workingItemN` decreased from 100,000 to 70,000. We present the results of testing 10,000 task instances on 10,000 items (of the 70,00 created). According to the results, the errors of all tasks are lower than in the base setting but the network is less robust to activations of irrelevant items.

7.2.1 Chaining

In Table 6 we present the results of testing the functionality of chained tasks in the "1/10" regime β setting. The format of the table is the same as the one used in Table 3. According to the results, for both the sequential method of testing and the parallel one, the errors of chained tasks are (on average) within the errors of individual task instances. This implies that no significant deterioration is caused by the bias in the source items. In addition, the error caused by the simultaneous activation of all source items of level-two task instances is relatively insignificant. This gives further evidence that computations in regime β are more robust to activity of irrelevant items.

8 Discussion

We have shown that long sequences of association, supervised memorization and inductive learning tasks performed on items created via hierarchical memory formation, as we have defined them, can be robustly

Error/Test type	Assoc	Assoc Ch S/P	Assoc Freq	SMem	SMem Ch S	SMem Ch P	SMem Freq	Learn	Learn Ch S	Learn Ch P	Learn Freq
ON	0.0016	0.0041		0.0019	0.0044	0.0036		0.0138	0.0102	0.0083	
OFF with 0 irrel.	0.0	2.1E-4	0.873	0.003	0.002	0.0024	0.66	0.0091	0.0071	0.0094	0.229
OFF with 1 irrel.	0.0	9.3E-4	0.433	0.0034	0.0056	0.0053	0.633	0.0137	0.0129	0.0125	0.358
OFF with 2 irrel.	0.0	4.6E-4	0.296	0.0044	0.0125	0.012	0.468	0.019	0.0195	0.0195	0.444
OFF with 3 irrel.	0.0	0.0033	0.149	0.0109	0.0268	0.0256	0.323	0.0269	0.0249	0.0302	0.457
OFF with 4 irrel.	0.0	0.0046	0.062	0.0365	0.0542	0.0573	0.204	0.0379	0.0365	0.0445	0.414
OFF with 5 irrel.	0.0	0.0	0.021	0.0742	0.0961	0.0954	0.135	0.0526	0.0503	0.0561	0.327
OFF with 6 irrel.	0.0	–	0.001	0.1105	0.1234	0.1386	0.067	0.07	0.0778	0.0767	0.242

Table 6: Results for tests of chained tasks in regime β base setting.

supported by a suite of algorithms working on networks with realistic neuron and synapse numbers. We believe that cognition in humans is realized over a lifetime by a sequence of discrete interactions with the world, which encompass various kinds of action including those modeled by our four task types. The challenge for information processing in the brain is then to execute these long sequences of actions, each with the desired long-term results, without undue interference by later actions on the long-term behavior desired from earlier ones. In this paper we have exhibited, apparently for the first time, an information processing scheme that supports this desirable behavior over tens of thousands of such actions, while using only realistic resource parameters. We believe that no general theory of information processing in cortex can be considered viable without such a demonstration.

The simulations show that the desired results can be realized in two regimes with distinguishable characteristics. In the first, regime α, algorithms can be supported by weak synapses. They need a large replication factor r but two items may be represented by overlapping sets of neurons. They correspond to systems in which neurons that represent an identified real world item abound (e.g. place cells in hippocampus (OKeefe & Dostrovsky, 1971)) and are, in that sense, easily found by single neuron recordings. The simulations show, for example, that starting from 3,200 items created by hierarchical memory formation in 250,000 neurons with $k = 16$, and with average $r = 898$, sequences of 2,000 task instances altogether, consisting of associations, supervised memorization and inductive learning tasks in random order, can be supported without the functionality degrading significantly. With stronger synapses $k = 16/5$ and two step memory formation, the higher capacity of 12,500 can be attained. The following informal argument shows where the capacity limitation comes from in regime α. If there are nd synapses in the system and each item uses at least rk of them, then if each synapse is used for at most one item on the average, then at most $nd/(rk)$ items can be represented. But it was previously observed (Valiant, 2005) that association in this regime requires $d/k > n/r$, which means that the bound $nd/(rk)$ is at most $\sim (d/k)^2$. Thus, for example, if the ratio d/k is regarded as fixed, then having more neurons does not increase this upper bound on the capacity at all, though it may increase the capacity by making the operations more stable. The similarity between columns AVG and MLN corroborates this analysis. The reader should also note that by the same argument, if we had increased n to higher numbers such as the 10^8 used in Table 5, while keeping d and k unchanged, we would not have obtained significantly higher capacities.

The algorithms in the second regime, regime β, require some strong $k = 1$ synapses and are slightly more complex, requiring an intermediate level of relay neurons. We have shown here that the $k = 1$ synapses are only needed at the static relay neurons, and the dynamic synapses at the target neurons may be weaker, such as $k = 4$. The advantage regime β offers is that the replication factor r can be small, and hence even if items are represented by disjoint neuron sets, the capacity is large, namely n/r. Association in this regime requires that $r > n/d^2$ (Valiant, 1994). This gives the number of relay neurons activated by an item to be

$\sim r \cdot d > n/d$ if r and d are small enough. The capacity upper bound n/r is therefore $\sim d^2$. Our simulations show that now with average $r = 100$, sequences of as many as 250,000 task instances can be supported on 100,000,000 neurons without substantial degradation. Representing items by such disjoint sets of neurons may also make it easier to realize higher level functions, but we know of no rigorous argument that shows that regime β is fundamentally more powerful than regime α.

Verifying directly whether a certain algorithm is being executed in a biological system may be beyond current experimental techniques. However, our various algorithms do have various telltale signatures. If all four parameters n, d, k and r can be measured, as done by Jortner et al. (2007), then the relationship quoted above can be verified (Valiant, 2006). In the case of regime β where r is small, it may be difficult to measure it by finding cells that represent any one chosen item. In that case the existence of strong synapses ($k = 1$) and silent cells would be telltale signs, as would also the relay neurons which would resemble the shared denser representations of regime α. Note that the expressions given above for regime α, r is at least nk/d while for regime β the number of relay neurons activated by an item is at least n/d. In general for all our vicinal algorithms there is no requirement that the set of neurons representing any one item should synapse on each other with more than average frequency or strength. On the other hand they may synapse more frequently and strongly on a set of postsynaptic neurons that represents a semantically related item.

We also make the following remarks on issues that are not covered in this work. As has been observed elsewhere the memory formation mechanisms described are not robust if implemented to many levels, but there are several ideas for approaching this (Valiant, 1994; Gerbessiotis, 1998; Gunay & Maida, 2006; Beal & Knight, 2008). These mechanisms are very sensitive to the size of the items and also to interference from other active items. A thorough investigation of how greater robustness can be achieved remains to be done. We note, however, that in the brain there are mechanisms that go beyond the excitatory local mechanisms that are sufficient for everything described in this paper. For example there are inhibitory synapses, and also global action realized by neuromodulators. It remains for future work to explore how these extra mechanisms might be harnessed for the benefit of implementing the sets of functionalities we have described more effectively.

Finally we would like to draw attention to the extreme simplicity of the mechanisms that are sufficient in regime α to realize each of our tasks. The neurons are identical and random connections among them suffice. The updates operate across just one circuit level, and, with the exception of supervised memorization, in just one time step. Regime β differs in that it enables items to be represented sparsely and disjointly, in that sense akin to a digital computer, and our experiments suggest that it supports significantly higher capacities than does regime α.

Acknowledgments

This work was supported by grants NSF-CCF-04-32037 and NSF-CCF-04-27129.

References

Abeles, M. (1991). *Corticonics: Neural circuits of the cerebral cortex.* Cambridge: Cambridge University Press.

Arriaga, R., & Vempala, S. (1999). An algorithmic theory of learning: Robust concepts and random projection. In *proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS)* (pp. 616–623).

Beal, J., & Knight, T. F. (2008). Analyzing composability in a sparse encoding model of memorization and association. In *proceeding of the 7th IEEE International Conference on Development and Learning (ICDL 2008)* (p. 180-185).

Biederman, I. (1987). Recognition-by-components: A theory of human image understanding. *Psychological Review*, *94*, 115-147.

Braitenberg, V., & Schuz, A. (1998). *Cortex: Statistics and geometry of neuronal connectivity.* Berlin: Springer-Verlag.

Feldman, J. A. (1982). Dynamic connections in neural networks. *Biological Cybernetics*, *46*, 27–39.

Feldman, J. A., & Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive Science*, *6*, 205–254.

Gerbessiotis, A. (1998). A graph-theoretic result for a model of neural computation. *Discrete Applied Mathematics*, *82*, 257–262.

Graham, B., & Willshaw, D. (1997). Capacity and information efficiency of the associative net. *Network: Comput. Neural Syst.*, *8*, 35.

Gunay, C., & Maida, A. S. (2006). A stochastic population approach to the problem of stable recruitment hierarchies in spiking neural networks. *Biological Cybernetics*, *94*, 33–45.

Hopfield, J. (1982). Neural networks and physical systems with emergent collective computational abilities. In *Proc. natl. acad. sci.* (Vol. 79, p. 2554-2558).

Jortner, R., Farivar, S., & Laurent, G. (2007, February). A simple connectivity scheme for sparse coding in an olfactory systems. *Journal of Neuroscience*, *27*(7), 1659-1669.

Krauth, W., & Mezard, M. (1987). Learning algorithms with optimal stability in neural networks. *Journal of Physics A*, *20(11)*, 745-752.

Littlestone, N. (1987). Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning*, *2*, 285-318.

Marr, D. (1969). A theory of cerebellar cortex. *J. Physiol*, *202*, 437.

Minsky, M., & Papert, S. (1969). *Perceptrons: an introduction to computational geometry.* Cambridge, MA: MIT Press.

OKeefe, J., & Dostrovsky, J. (1971). *The hippocampus as a spatial map.* Preliminary evidence from unit activity in the freely moving rat. J. Brain Res. 34, 171.

Quiroga, R., Reddy, L., Kreiman, G., Koch, C., & Fried, I. (2005). Invariant visual representation by single neurons in the human brain. *Nature*, *435*, 1102-1107.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*, 386-407.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533-536.

Shastri, L. (2001). Biological grounding of recruitment learning and vicinal algorithms in long-term potentiation. In *Emergent neural computational architectures based on neuroscience* (p. 348-367).

Valiant, L. G. (1994). *Circuits of the mind.* Oxford University Press.

Valiant, L. G. (2005). Memorization and association on a realistic neural model. *Neural Computation*, *17*(3), 527-555.

Valiant, L. G. (2006). A quantitative theory of neural computation. *Biological Cybernetics*, *95*(3), 205–211.

VanRullen, R., & Thorpe, S. (2001a). Is it a bird? is it a plane? ultra-rapid visual categorisation of natural and artifactual objects. *Perception*, *30*(6), 655-668.

VanRullen, R., & Thorpe, S. (2001b). The time course of visual processing: From early perception to decision-making. *Journal of Cognitive Neuroscience*, *13*, 454-461.

Willshaw, D., Buneman, O., & Longuest-Higgins, H. (1968). Non-holographic associative memory. *Nature*, *222*, 960-962.