Document submitted in partial fulfillment of the Qualifying Exam

Beyond pairwise associations: A review of the Simple Recurrent Network and its application to modeling sequential learning

Jennifer P. Provyn

Syracuse University

Abstract

Connectionist models assume the computational properties of vast ensembles of real neuronal elements can be captured with simulations of much smaller networks of more abstract units. The simple recurrent network (SRN), a feedforward network with a context layer that enables the hidden layer to feed back on itself, such that intermediate results at time $t - x$ can influence processing at time t, is a connectionist model widely implemented in simulations of sequential learning. Although the SRN can account for a wide range of adjacent and non-adjacent sequence processing data, it has specific limitations that prevent it from being a completely plausible implementation of human sequence learning.

PDP models employ distributed representations of patterns of activity across many simple neuron-like processing elements called units. Figure illustrates a skeletal representation of such a network. The basis of parallel distributed processing modeling (PDP, Rumelhart, McClelland, & PDP Research Group, 1986a, 1986b), a connectionist methodology, is that the computational properties of vast ensembles of real neuronal elements can be captured with simulations of much smaller networks of more abstract units (Plaut,

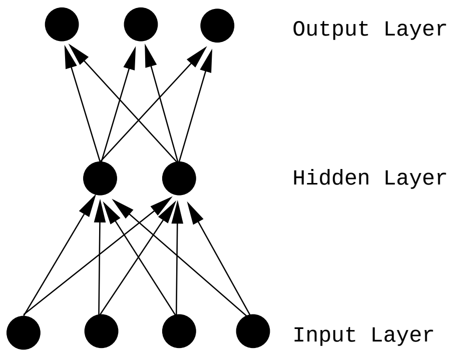

Figure 1. **Generic neural network.** Filled circles represent processing units, which are connected to other units by way of the arrowed connections. Information processing proceeds in the direction of the arrows. The processing units are further divided into different layers.

2000). These types of models can be conceived of as microlevel system analyses, which form the internal structure of the larger, macro-level units. Despite considerable diversity, all PDP models consist of the same basic components: simple processing elements (units) and weighted connections (weights) between those elements. The units correspond roughly to neurons and exhibit nonlinear spatial and temporal summation. The connections between the units approximate synapses and the modifiable weights on these connections are comparable to synaptic plasticity.

The units in a PDP network are subsequently grouped into various layers (Figure) and the interactions among units in different layers (i.e., excitation and inhibition) are governed by the modifiable weights on the connections between the layers. The input and output units approximate the role of stimulus and response. With training, distributed representations of activation from the input layer develop across the hidden units. These internal representations can come to relate task inputs to task outputs, and can be used to model tasks such as simple pattern association.

General Sequence Learning

Learning is often the result of capitalizing upon the fact that causally-linked events proceed sequentially. For example, smell precedes taste when foraging, and the sound of moving prey often precedes its smell. Organisms capable of learning sequential regulari-

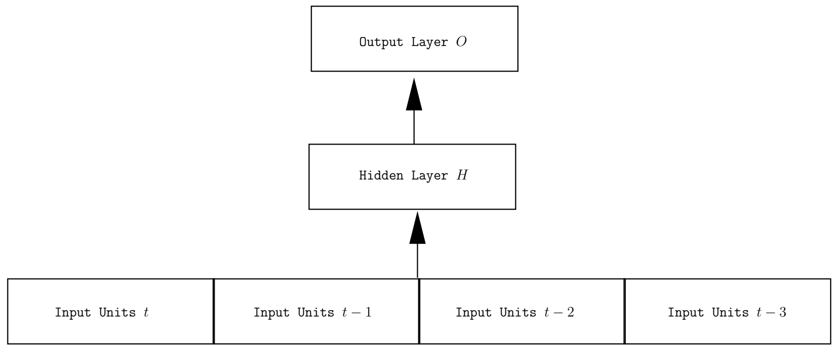

Figure 2. **Memory buffer model.** A connectionist memory buffer network with a temporal window of 4 time steps. The input layer consists of four pools of isomorphic units, with each pool corresponding to a specific time step. Sequence elements are presented, one at a time, to the network starting with the leftmost pool, corresponding to time step t. On each time step the contents of each pool are copied (and typically decayed) to the pool to the right and a new sequence element is presented to pool t. Once the length of the sequence elements exceeds the length of the pools, the contents of the oldest pool, i.e. $t-3$, are lost.

ties are potentially able to predict events and perform anticipatory actions, giving them a decided evolutionary advantage.

In the past few decades sequence learning has become a popular paradigm through which to study the underlying mechanisms of learning in connectionist models (e.g., Berry & Dienes, 1993; Cleeremans, 1993). The rationale underlying sequence learning tasks is that Ss' responses reflect a complex sensitivity to the sequence structure inherent in the stimuli. For example, consider the sequence ABCDABCD in which each letter corresponds to a particular location on a computer screen. In the classic sequence learning task (Nissen & Bullemer, 1987) Ss make a unique response to each individually visually presented stimulus as quickly and accurately as possible. Importantly, Ss are not informed of the underlying stimulus structure, nor are they given explicit intention to learn (e.g., they are not asked to extract patterns from the stimulus presentation). Nonetheless, Ss display sensitivity to the relevant sequential contingencies, or structure, among task stimuli above and beyond proficiency with the stimulus-response mapping.

Buffer models to simulate sequence learning

Connectionist memory buffer models use a pool of inputs for the events present at time t, another pool for events at time $t-1$, to $t-n$, in a "moving window" (Servan-Schreiber, Cleeremans, & McClelland, 1991). Figure illustrates a connectionist memory buffer model, with four pools of input units representing sequence elements up to four time steps, t, $t-1$, $t-2$, and $t-3$. On each time step the contents of the pools are copied (and typically decayed) to the previous pool and a new input is presented on the current time step, t. The contents of the most remote pool, $t-3$, are dropped from the network. For example, consider the sequence A C B D. On time step t, A is presented on the input pool t. At the next time step, the contents of pool t, A, are copied to pool $t-1$, and the next letter in the sequence, C is presented on pool t. Similarly, at time step 3, pool $t-2$ contains the sequence element A, pool $t-1$ contains the sequence element C, and pool t contains the sequence element B.

Connectionist memory buffer models make the effects of events occurring at different relative time points explicit, but this method entails two important limitations. First, the input at time t is duplicated at time $t-n$ onto the corresponding input pools (Elman, 1990). Second, the number of time steps that have been encoded in the input pools determine the length of the remote dependencies that can be learned (Elman, 1990; Cleeremans & Dienes, 2008). As such, elements that fall outside of the buffer window are lost from processing. There are two solutions to the temporal window problem: determine the size of the window based on the longest (possibly infinite) interval between the time when relevant information is presented and the time when it is needed to make a response, or, set an arbitrary buffer size and forfeit the network's ability to process sequences exceeding this size. Both solutions are computationally inefficient and psychologically implausible (Elman, 1990; Cleeremans, 1993).

Part I

Introduction to the Simple Recurrent Network

General introduction. The simple recurrent network (SRN), influenced by the Jordan network (Jordan, 1986), was first proposed by Elman (1990) to model the structural aspects of language as it varies in time. It is an elegant way of instantiating the desirable properties of a connectionist memory buffer while overcoming the limitations of the buffer models. Indeed, the SRN is widely used in cognitive science to approximate contextual manipulations in temporally extended tasks (see Cleeremans, 1993, for a review). In contrast to connectionist memory buffer models, which restrict input processing to a finite window by shifting the representations of elements to different input pools, the SRN uses the *same* units to represent the *same* elements across time steps. This architecture enables items to have effects arbitrarily far into the future. As such, the SRN, coupled with more efficient internal representations and a learning procedure that is completely local in time, has the potential to master an infinite corpus of sequences (Servan-Schreiber et al., 1991).

Architecture

The basic architecture of the SRN is illustrated in Figure 3. Network units are divided into four layers: an input layer with i units, a context layer with l units, a hidden layer with j units, and an output layer with k units. The SRN is subject to the constraint that $l = j$, which is strictly equivalent to a fully recurrent feedback loop on the hidden layer. Every input and context unit is connected to every hidden unit. Similarly, there are massively parallel connections between the hidden and output units.

The SRN's input layer consists of a single pool of units connected to the hidden layer. Recurrent links in the hidden layer serve to copy the activation of the hidden layer to an additional context layer, which in turn feeds the weighted sums of the activation of the previous hidden layer back to the hidden layer at the next time step. For example, when the first input is presented to the network, the context layer has no activation, but the activation of the hidden units is copied onto the context layer for the next time step. On time step $t + 1$ the activation values presented to the network on the input layer, as well as the activation values from the context layer (which are copies of the representations of the hidden layer activations on the previous time step, t) are projected to the hidden layer. This hidden layer recurrence enables the intermediate results of processing at time $t - 1$ to influence the results of processing at time t. Further, at time step $t + 2$ the activation of the hidden units is influenced by both the activation of time step t and time step $t + 1$. As such, the hidden layer encodes not only prior events but also relevant aspects of the representation that were constructed in predicting the prior event from its predecessor. As these representations are fed back into the hidden layer as input, they provide information necessary for the SRN to maintain the prediction relevant features of the entire stimulus sequence (Servan-Schreiber et al., 1991).

The input and output layers' activation is explicitly determined by the user during training. The input layer is clamped, or constrained, to the user-defined input vector and the output layer is modified as a function of the user-defined stimulus-response pairings by a supervised learning algorithm. In contrast, the activation of the internalized recurrent hidden layer is indirectly constrained through learning during training. Backpropagation (first proposed by Werbos in 1974, but see Werbos, 1994), an algorithm used in the SRN and a wide variety of other connectionist models, restricts the internal representations across the hidden layer to correspond to task-relevant features of the input pattern (e.g. Hinton, 1986). The context layer also shapes the hidden layer, providing it with input corresponding to contextual features of previously presented stimuli that shape the original input vectors into contextually-dependent representations. The resultant patterns of activation, or internal representations, that develop across the hidden layer are, as we shall see, of a different dimensionality than the original input patterns. These contextually rich hidden layer representations enable the SRN, with training, to become sensitive to the structural

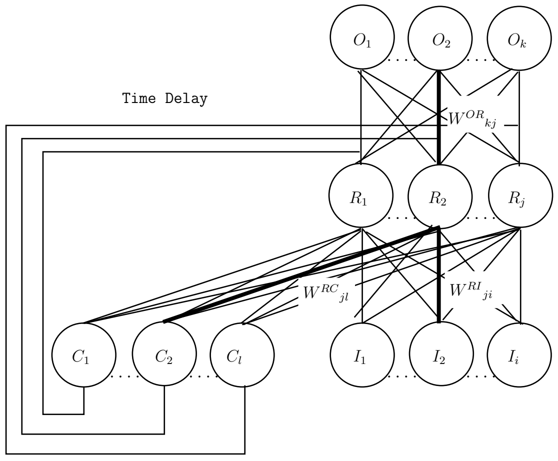

Figure 3. **Simple recurrent network.** Each layer contains a pool of units with each forward arrow representing a complete set of trainable weights between the layers. The backward arrow from the hidden layer to the context layer denotes a copy operation, the recurrence of which provides the network with "memory".

features of the underlying stimulus sequence (e.g. Elman, 1990, 1991, 1993; Cleeremans & McCleeland, 1991; Cleeremans, 1993; Rodriguez, 2001, 2003; Servan-Schreiber et al., 1991; Cleeremans, Servan-Schreiber, & McClelland, 1989).

Processing units

Processing units collect input from a variety of sources. In fully recurrent networks, all units are connected with all other units. In a layered network, of which the SRN is a member, some units are specialized exclusively for input or output, and only interior hidden units maintain the capacity for dimensional reduction and generality. *Input* units receive input from sources that are external to the network. These sources may be sensory inputs or inputs from other parts of the processing system in which the network is embedded. *Output* units may act as effectors, and send activation outward, affecting motor systems or influencing other external systems' responses. *Hidden* units receive and send input only to other units and are not "visible" to outside systems.

The activity of each unit can be represented by a number, and the activity of the whole network can be given by a vector of these numbers. Both the network input, which originates from external sources and is analogous to biological sensory systems, and the network output, which is propagated out of the network and is analogous to biological motor systems, can be treated as vectors. The pattern of activation over the set of processing units captures what the network represents at any given time. The network amounts to a vector processor, and the problem of sequence learning can be understood as a series of operations on vectors (Garson, 2007).

Activation Representations

Localist representations. Although the SRN has distributed activation representations across all hidden layers, it can have *localist* or *distributed* representations across the input and/or output layers. A *localist* network is a *one-to-one-concept* representational system in which single units represent entire concepts, such as letters (e.g. Elman, 1990), words (e.g., Elman's syntax model Elman, 1990, 1991, 1993), or the syntactic roles of the words in a particular sentence. Consider a network architecture with 4 input units, each coding for one of the letters, "A","B","C" or "D". An input vector for the letter "A" would only activate the unit that represents, or codes for, the letter "A": $(1, 0, 0, 0)$.

Distributed representations. Rather than each input and/or output unit corresponding to particular conceptual objects, units in *distributed* networks represent abstract elements over which meaningful patterns can be defined. Input is encoded by means of a set of simultaneously activated units, and each of these units can encode more than one distinct input. Keeping with the previous letter example, the non-orthogonal input vector for the letter "A" across four input units with distributed representations would be graded, for instance: $(0.3, 0.8, 0.1, 0.0)$. As another example of distributed input layer representations, consider Hare, Elman, and Daugherty (1995)'s model of inflection of English past tense, in which input features corresponded to speech segments in specific positions. Fourteen input units corresponded to 14 possible syllable beginnings, six input units corresponded to six possible middles, and 18 input units corresponded to 18 possible syllable endings. The word *bid* was represented by the simultaneous activation of three units: the unit corresponding to b in the first position, the unit corresponding to i in the middle position, and the unit corresponding to d in the ending position. Each of these units also participated in the encoding of other inputs, such as *bad*, wherein the same beginning and ending units were activated.

Processing unit activation. To train the SRN, items are represented as vectors, with each feature or element set to a value (typically) between 0 and 1. At every point in time each of the units in a network has an activation value resulting, either directly or indirectly, from the vector presented on the input units. To present an item to the network the activation values of the input units are set to a desired input pattern, and each input unit activation, o_k, is unidirectionally propagated to the hidden layer units. The corresponding activation value of each hidden unit, a_j, is modulated by the weights on the connections between the units in the input and hidden layers, w_{jk}. The net input over all incoming input units, o_k, to each hidden unit, a_j, is the sum of the products of the sending unit activations and corresponding weights:

$$\text{net}_j = \sum_k w_{jk} o_k. \tag{1}$$

This net value is then passed through a 'squashing' function that maps the input activation to an output signal,

$$o_j = f(\text{net}_j) = \frac{1}{1 + e^{\text{-net}_j}}. \tag{2}$$

$f(\text{net}_j)$ is a logistic sigmoid transformation that transforms net_j values from a range of $-\infty$ to $+\infty$ to the range of 0 to 1.

The activations of output layer units are then calculated by Equation 1, with the hidden layer unit outputs, o_j, and the set of weights between the hidden and output layers, w_{ij}. The net input to the output layer is transformed by Equation 2 to an activation output and produced as the network's output prediction.

Training algorithm

Backpropagation

The values output, or predicted, by the SRN depend upon the network's weight values. For every SRN and corresponding simulation task, there is a space that is defined by all of the possible weight combinations the network can take. The network seeks those weight combinations that minimize the difference between the network's output and the target output. Toward this end, within the space of all possible prediction states, there exists an error surface that is the joint function of the SRN architecture (e.g., the number of hidden units in the network) and the simulation task (e.g., given a sequentially structured input, predict the next stimulus in the sequence). If the error surface was known a priori, the SRN weights could be set to the combination of weights producing the lowest error on the error surface. Typically, however, the error surface is unknown. To determine the lowest point on the unknown error surface the SRN could be simulated to systematically sweep through all possible combinations of weights, testing the network at each point (Elman, 1993). This method is combinatorially exhaustive and inefficient, particularly with networks with a large number of weights. Gradient descent algorithms are methods that allow the network to systematically and efficiently determine the combination of weights that yield the minimum error in the error space.

The SRN employs a gradient descent, weight optimization mechanism to correctly predict input/output mappings. The capacity of PDP networks to generalize response mappings to novel inputs is an emergent property of the networks' modifiable weights. The experience dependent evolution of the SRN's unit representations is a direct function of the network's backpropagation learning algorithm. Backpropagation, a computational constraint built into the network, can be applied to almost any system assembled up from elementary subsystems or calculations. The the one serious constraint with this learning algorithm is that the elementary subsystems must be represented by functions that are both continuous and differentiable (Werbos, 1990). At its core backpropagation is a method for calculating the derivatives of a single target quantity (e.g., prediction error or pattern classification error) with respect to a large set of input quantities (e.g., parameters or weights in a network).

For a given input (or sequence of inputs), the backpropagation training algorithm compares the network output to the target output, and calculates the error between the outputs. Next, the algorithm seeks the spot in parameter space that yields the lowest error signal (i.e., sum squared error) between the target and output values. In practice, the modifiable weights between units are initially set to random values and members of the training set are repeatedly exposed to the network because networks that employ backpropagation typically require extensive training. A vector consisting of the value of the input stimulus is placed on the input units and cycled through the network. The network output is compared to the target output and the sets of weights between the layers of units in the network are modified to minimize the sum of the squared error between each target value and the corresponding values of the output vector (a process called "supervised learning"). If there is no error on the output then the sum squared error is zero and weights are not modified. Reducing the sum squared error, given a non-zero value, amounts to following the anti-gradient of the error in parameter space, as a function of all network parameters.

To update individual weights, each weight is changed proportional to the product of the error signal of a unit receiving input and the output of the sending activation. From Equation 2, the semilinear transfer function maps the total input of a unit to an output value. The derivative of the sigmoid transfer function yields the error signal at an output unit. An output unit's error is defined as:

$$\delta_i = (\mu_i - o_i)o_i(1 - o_i), \tag{3}$$

where μ_i is the target value for unit i (Rumelhart et al., 1986a). The error signals can be recursively computed layer-wise, starting with the weights between the output and hidden layers,

$$\Delta w_{ij} = \eta \delta_i o_j, \tag{4}$$

where η is a learning rate parameter. The learning rate parameter, a constant of proportionality, is necessary to prevent the SRN from oscillating and producing slow convergence to an adequate solution in the parameter space. For example, if the sum of squares is highly variable as a function of parameters and if a high learning rate is used (a large η in Equation 4 leading to large changes in the network's weights), the system will oscillate wildly and possibly not converge to the optimal state of lowest error. To prevent weight oscillation in parameter space, weight modification is based on a linear combination of the current computed weight change and the prior weight change. The component of weight update that reflects previous weight changes is the momentum (see Rumelhart et al., 1986a),

$$\Delta w_{ij}(t+1) = \eta(\delta_i o_j) + \alpha \Delta w_{ij}(t), \tag{5}$$

where α is the momentum parameter over the range $[0, 1]$. The learning rate and momentum are typically inversely related, such that a network with a high momentum (approaching 1) will have a lower learning rate.

Once the hidden-to-output-layer weights are modified by Equation 5, the output error is further propagated to the input-to-hidden layer weights, w_{jk}. However, because the hidden layer does not have an output error, the input-to-hidden-layer weights are updated by an error signal that is computed by propagating the error from the output layer back to the hidden layer,

$$\delta_j = o_i(1 - o_i) \sum_k w_{jk} \delta_i. \tag{6}$$

The weight change between the hidden and input layer is still given by Equation 6, with the δ_i value given by the resultant δ_j from Equation 6 (e.g., $\Delta w_{jk}(n+1) =$

$\eta(\delta_j o_k) + \alpha \Delta w_{jk}(t))$.

Once all of the weights have been updated, the activation of each hidden unit is copied onto its corresponding context unit. Because the context units' activations are direct copies of the hidden unit activations, the weights between the hidden and context layer are immutable and equal to 1. The context units employ a linear, rather than a logistic, activation function. Completion of this process represents one time step, and the next input vector can be presented on the input layer and begin the process anew.

Initializing the network

Initializing network weights. If every weight in the SRN started at 0, the gradient of the error loss with respect to every other weight would be identical. After updating, all of the weights would remain the same because each hidden unit would have received the same error signal from the output layer. Therefore, prior to training, the network weights are initialized to small random values to enable symmetry breaking (Rumelhart et al., 1986a). The random values are typically drawn from a uniform distribution over the range $[-r, r]$. The value of r is important because if the initial weights are too small, both activation and error signals will decay out as they propagate through the network. Conversely, if the initial weights are too large the hidden units will saturate out very close to the asymptotic value of the sigmoid function, essentially blocking any backpropagated error signal from passing through the unit.

There is a nonlinear relationship between the rate of change in the network weights and the prediction that is affected by the weight changes, such that the network output is not necessarily proportional to the network input. A very small difference in the starting point in weight space of two otherwise similar systems can lead to different outcomes (Elman, 1993; Elman et al., 1996). Variation in the size of the initial values is typically examined and reported in simulations, with the convention in the field to chose initial weights with a uniform distribution between $\pm\ \rho$, typically set to $\leq +0.5$ (e.g. Rumelhart et al., 1986a, 1986b). Kolen and Pollack (1990) trained a 2-2-1 network on the XOR (i.e., binary output task) with systematic manipulations in learning rate parameter (i.e., 1.0 or 2.0), momentum parameter (i.e., 0.0, 0.5, or 0.9), and the initial weight range parameters (i.e., $\rho = 0.1$ to 0.9 in 0.1 increments, and $\rho = 1.0$ to 10.0 in 1.0 increments). The authors concluded that values of $\rho \leq 0.5$ were more than sufficient to break the symmetry default.

Learning Phases

Servan-Schreiber et al. (1991) have demonstrated that training the SRN on sequentially structured material results in a learning progression through three qualitatively distinct phases. In phase 1 the network ignores context information and establishes a stable association between each letter and all of its possible successors. The the network ignores the context layer in phase 1 as a consequence of the continuously changing patterns of activation across the hidden layer. For example, consider the sequence S S X X X V P S. The pattern of activation across the hidden layer for each of the S elements, S_1, S_2 and S_3, would be nearly identical.

During phase 2 of learning, patterns copied from the hidden layer to the context layer have generally stabilized. The patterns of activation across the context layer for each letter begin to evolve a unique first-order conditional representation. That is, each context code designates which letter preceded the current letter. The network can then incorporate this first-order contextual information into its output such that it begins to distinguish between different occurrences of the same letter. From the example sequence, S S X X X V P S, the hidden layer representation elicited by S_1 would become progressively different from that elicited by both S_2 and S_3 because each S_n was preceded by a different letter. However, elements that require integration of temporal context larger than one time step into the past, such as X_2 and X_3, will not reliably evoke the target output (i.e., only 50% of the time in this example).

The SRN has transitioned to the third phase of learning when it begins incorporating higher-order conditional information. As more temporally distinct contextual information is encoded into the internal representation of each stimulus element, the network output begins to approximate the conditional probability of each element in the temporal context set by its predecessors. Once fully trained, the SRN should exhibit sensitivity to the transitional probabilities between the elements of the training sequence. By incorporating larger and larger magnitudes of temporal context, the internal representations of identical stimulus elements diverge, and the SRN can begin to differentiate between predictions for the same stimuli (e.g., X_n) along different points in the sequence (e.g., X_2 and X_3).

To summarize the progression of the learning phases of an SRN trained on sequential material, consider again the example sequence S S X X X V P S. In phase 1 of learning the network learns to develop different internal representations for different letters, with repeated letters evoking nearly identical patterns of activation. In phase 2 of learning, the network learns to incorporate temporal information one element back in the sequence such that identical elements no longer have identical internal representations. During phase 2, identical elements can potentially evoke different response outputs. However, elements that require incorporating contextual information > one temporal step into the past will not reliably evoke the target input/output mapping. During phase 3 of learning, the SRN begins to incorporate larger chunks of temporal context, enabling it to differentiate among identical sequence elements based upon each element's unique temporal context (however, see Servan-Schreiber et al., 1991, for limitations to the SRN's temporal context "window" capacity). Ultimately, the SRN's output responses should become the optimal conditional probabilities for each stimulus element. This is equivalent to the minima in the error function located at those points in weight space where the optimal conditional probabilities for each input stimulus element equals the target output activation for that stimulus element (Servan-Schreiber et al., 1991).

Part II

Simulating Sequence Learning with the Simple Recurrent Network

Adjacent Sequence Learning

For a given sequence learning simulation, the task of the SRN is essentially a regression problem: produce a specific output for a specific input. An important issue in modeling sequence learning is to determine the simplest sufficient learning mechanism mediating performance. Central to this issue is whether basic associative processes are sufficient to account for performance, such as associating two sequence elements into a bigram/pair, or if more complicated learning mechanisms are warranted.

Frequency information

Perhaps the simplest possible mechanism to account for sequential learning is sensitivity to frequency information. The effects of manipulating stimulus element frequency on sequence learning can be examined with the serial reaction time (SRT) paradigm. The SRT task measures sensitivity to sequential constraints present in structured material. In the SRT task, human *S*s are instructed to respond with a discriminative response to a finite set of sequentially structured stimulus events. For example, consider the sequence ABCDABCD..., with each letter corresponding to one corner of a computer screen. There is a one-to-one stimulus-response mapping, and on each trial a stimulus appears at one of several locations on a computer screen. *S*s are instructed to respond as quickly and accurately as possible to the stimulus with its predetermined, corresponding keyboard button. Returning to the example, if the letter A in the sequence corresponded to the top right corner of the computer screen, the *S*s would press the key corresponding to that corner of the screen when presented with the stimulus A. The target stimulus is removed upon keypress and the next stimulus is immediately presented. Unbeknownst to the subject, however, the sequence of stimulus events follows an underlying structure. Often an unstructured sequence is presented during an unannounced transfer block to establish that reaction time (RT) savings reflect sequential knowledge and not simply proficiency with the stimulus-response mapping (e.g. Nissen & Bullemer, 1987). With respect to the example sequence above, an example of an unstructured sequence could be DACBAAAC. There are dramatic RT increases to unstructured stimuli presented in the transfer block (i.e., stimuli that are not predictable based upon the temporal context set by previous stimuli in the sequence) as compared to RTs to stimuli presented in the structured trials (e.g., stimuli that are predictable based on the temporal context set by the previous stimuli in the sequence). These RT differences have been demonstrated both prior to, as well as after, the transfer block (e.g. Nissen & Bullemer, 1987; Curran, 1997).

Nissen and Bullemer (1987) first demonstrated that *S*s progressively learn the underlying sequential structure of a repeating series of stimuli, despite little evidence for a correlation in performance with verbalizable knowledge of the underlying structure. However, Nissen and Bullemer's (1987) sequence was wrought with frequency effects that could have accounted for the results. Consider the sequence of presentation positions, D-B-C-A-C-B-D-C-B-A. Some positions, B and C, occur more frequently than others, A and D. As compared to random trials, sequence trials should benefit from the relative frequency of occurrence of the sequence items, and reaction times should be correspondingly faster (Jackson & Jackson, 1995; Reed & Johnson, 1994; Shanks & St. John, 1994). However, even if frequency information is equated across elements, there is still an advantage for sequences; frequency information alone is not a sufficient mechanism to account for sequence learning (Cohen, Ivry, & Keele, 1990; Reed & Johnson, 1994; Stadler, 1993). *S*s must be extracting statistical regularities, beyond just frequency information, in the sequence structure.

First-order conditional sequences

Pairwise associations link adjacent elements together and are efficient for extracting statistical regularities in sequential structures with Markovian dependencies, such as first-order conditional (FOC) sequences. FOC sequences are composed entirely of predictive elements that occur sequentially in time, such that the occurrence of each sequence element is unambiguously predicted by the immediately preceding sequence element. Consider the FOC sequence, A-B-C-A-B-C.... In this sequence, A always predicts B, B always predicts C, and C always predicts A. Although in the example sequence each sequence element predicts another sequence element with 100% probability, sequence elements in principle can predict other elements with probability < 1. For example, a sequence X may predict both sequence elements Y and Z with probability 0.5.

Much of the literature on associative learning (e.g., animal learning and paired-associate learning) has focused on such adjacent dependencies. Adjacent dependencies are also abundant in variants of language acquisition, such as the discovery of word boundaries in fluid speech streams. Delineating the boundaries between words in fluent speech is a challenging problem because speakers do not mark word boundaries with pauses, so listeners must determine where one word ends and another begins without access to obvious acoustic cues (Saffran, 2003). For example, infants do not **innately** know that *pretty* and *baby* are words and that *tyba*, which spans the boundary between *pretty* and *baby*, is not a word. Nonetheless, human adults and infants (e.g. Saffran, Aslin, & Newport, 1996; Saffran, Johnson, Aslin, & Newport, 1999), as well as Cotton-top tamarins–a New World monkey species–(Hauser, Newport, & Aslin, 2001), are able to discover word boundaries through the statistical learning of both frequency information and FOCs.

Non-Adjacent Sequence Learning

Although information processing relies heavily upon adjacent dependencies, there is ample evidence that statistical learning is not limited to just adjacent dependencies. Evidence for non-adjacent dependencies, in which two predictive items are separated by non-dependent embedded items, appears in auditorially presented material from linguistic domains in humans (Gomez, 2002; Perruchet, Tyler, Galland, & Peereman, 2004; Onnis, Monaghan, Christiansen, & Chater, 2004; Newport & Aslin, 2004; Aslin, Saffran, & Newport, 1998; Saffran et al., 1996; Saffran, Newport, Aslin, Tunick, & Barrueco, 1997) and non-linguistic domains (i.e., tone sequences) in both human (Kuhn & Dienes, 2008; Creel, Newport, & Aslin, 2004; Saffran et al., 1999) and non-human primates (Newport, Hauser, Spaepen, & Aslin, 2004). Additionally, non-adjacent contingencies derived from statistically structured material have also been observed in visually presented shape arrays (e.g. Fiser & Aslin, 2001, 2002; Kirkham, Slemmer, & Johnson, 2002). The generality of adjacent and non-adjacent dependency learning across species, human development, and stimulus modality suggests that learning mechanisms not specifically designed for language acquisition may have shaped the structure of human languages.

Second-order sequences

The simplest examples of non-adjacent dependencies are second-order conditional (SOC) sequences: sequences in which each stimulus element is equally likely to be followed by any other stimulus element, such that successive stimulus elements can be predicted only from some combination of multiple preceding items. Consider the SOC sequence A-B-A-D-B-C-D-A-C-B-D-C. Because element B appears in three different prediction contexts, B-A, B-C, and B-D, it does not uniquely predict any given element. However, if the temporal context is extended back two time steps, the combination of element B given, for example, element C, does uniquely predict the next sequence element (Reed & Johnson, 1994; Reber & Squire, 1994; Curran, 1997). Segments with such contingencies can be assembled using artificial grammars. Artificial grammars are generative tools widely employed in sequence learning that allow for the instantiation of SOCs, as well as sequences with higher-order non-adjacent dependencies.

Higher-order sequences and artificial grammars

Overview. Reber (1967) first used the artificial grammar learning (AGL) paradigm as a mechanism to induce complex, structured learning, without the *S*s having either an explicit intention to learn, or verbalizable cognizance of what was learned. A standard AGL architecture is illustrated in Figure 4. Finite grammars consist of nodes connect by arcs.

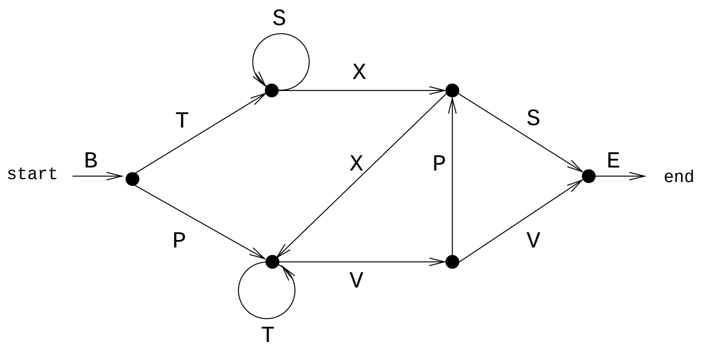

Figure 4. **Artificial grammar.** Finite state grammar used by Reber (1967). Nodes are connected by arcs and each grammatical string is generated by entering the grammar through the "start" node and traversing from node to node, recording each letter corresponding to the label of the arc, until the "end" node is reached.

A string is generated by entering the finite grammar network through the 'start' node and traversing, node to node, until the 'end' node is reached. Each transition across an arc results in the addition of the letter corresponding to the label of the arc to the end of the string.

The AGL task consists of a study phase and a classification phase. At study, *S*s are told of an impending memory test and presented with strings generated from the underlying grammar, though they are not informed of the underlying structure or the rules used to produce the stimuli. At test, *S*s are informed of the existence of a complex set of rules used to generate the stimuli although again are not informed of the specific rules or the AGL architecture itself. The *S*s' task is to classify a new set of exemplars into those that obey the grammar and those that violate the grammar.

In a standard AGL classification task *S*s classify strings with $> 50\%$ accuracy, indicating that they have learned, or can generalize their responses to, something of the underlying grammatical structure (Reber, 1989). *S*s are initially sensitive to statistical redundancy, such as bigrams and trigrams (e.g. Perruchet & Pacteau, 1990; Knowlton & Squire, 1996), and with continued exposure to artificial grammar exemplars, learn higher orders of redundancies (e.g. Cleeremans & McCleeland, 1991). The SRN can account for about 80% of the variance in reaction time data generated from human performance of the AGL task

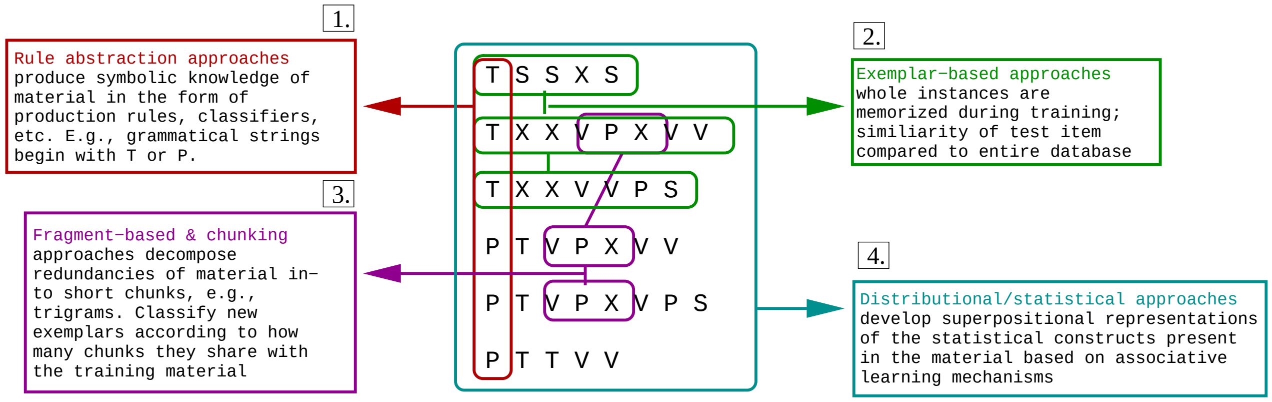
1.
Rule abstraction approaches produce symbolic knowledge of material in the form of production rules, classifiers, etc. E.g., grammatical strings begin with T or P.
2.
Exemplar-based approaches whole instances are memorized during training; similiarity of test item compared to entire database
3.
Fragment-based & chunking approaches decompose redundancies of material in-to short chunks, e.g., trigrams. Classify new exemplars according to how many chunks they share with the training material
4.
Distributional/statistical approaches develop superpositional representations of the statistical constructs present in the material based on associative learning mechanisms
T S S X S
T X X V P X V V
T X X V V P S
P T V P X V V
P T V P X V P S
P T T V V

Figure 5. **AGL approaches.** Representation of different computational approaches to AGL (from Cleeremans & Dienes (2008)).

(Cleeremans & McCleeland, 1991). To analyze network output and compare it to human RT data one typically assumes the normalized activation of each output unit is inversely proportional to RT.

AGL strategies. Before addressing the computational principles characteristic of the mechanisms of sequential learning, consider what both *S*s and the SRN could extract from the artificial grammar in Figure 4. They could learn to memorize fragments of the study strings (the purple boxes/number 3 in Figure 5); they could memorize entire study strings (the green boxes/number 2 in Figure 5); they could learn some of the rules that govern string generation (the red boxes/number 1 in Figure 5); or they could extract certain statistical features of the study strings (the blue boxes/number 4 in Figure 5) (Cleeremans & Dienes, 2008). Models based on retrieving exemplar information stored in memory arrays (i.e., the purple box/number 3 and green box/number 2 in Figure 5) have generally been directed toward accounting for performance at retrieval, while PDP models, such as the SRN, have generally been concerned with the mechanisms of learning (i.e., the red box/number 1 and blue box/number 4 in Figure 5) (Cleeremans & Dienes, 2008). According to Cleeremans and Dienes (2008), models based on retrieving exemplar information stored in memory arrays (i.e., the purple and green boxes in Figure 5) have generally been more concerned with accounting for performance at retrieval rather than on accounting for learning itself, while PDP models, such as the SRN, are centrally concerned with the mechanisms of learning (i.e., the red and blue boxes in Figure 5).

Consider again the examples of grammatical strings generated from Figure 4: 'TSXXVPS', 'PTTTVPS', 'PVV'. Identical letters can occur at different positions in each string, which leads to different predictions about the successor letter's identity (i.e., $t + 1$). Because the underlying structure of the AGL is not expressed completely as pairwise correlations between input and output units, any learning procedure that relies on direct correlations between input and output vectors will generalize very badly on the AGL task (Hinton, 1986). Importantly, sequence strings generated from artificial grammars are insufficiently expressed as hard and fast rules. Although there is not a stable prediction associated with any particular letter in a typical artificial grammar, the SRN is sensitive to subtle statistical patterns, making it especially well-suited to accommodate the graded notions of category membership inherent in an AGL .

Simulating AGL. The SRN does not make any *a priori* distinction between the representation of a letter at the beginning of a string and the same letter at the end of the string. To simulate AGL, the SRN is trained with the same AGL stimuli as human *S*s. Typically in these applications the network's task is to predict the next letter in the sequence: the target output at time t equals the input at time $t + 1$. The network output is compared to the next letter in the sequence and the difference between the target output and network output is used to calculate the error for backpropagation (Rumelhart et al., 1986a).

For each input stimulus, the activation values across all of the SRN's output units are collectively characterized as the network's endorsement rate of a given prediction. These output activation values comprise an output vector for each input stimulus, with the target output comprising a separate, but isomorphic, vector. In other words, the output vectors represent the letters the SRN predicted for each input stimulus, while the target vectors represent the correct prediction for each input stimulus. All of the network's output vectors are then concatenated into a global output vector. The global output vector corresponds to the length of a single output vector multiplied by the length of the test sequence string, plus 1 (because the end of the string is also predicted). A global target vector is likewise concatenated, resulting in one vector for the entire set of network output activations and one vector for the entire set of target activations (e.g. Altmann & Dienes, 1999; Berry & Dienes, 1993; Dienes, Altmann, & Gao, 1999; Kinder & Shanks, 2001; Boucher & Dienes, 2003). Three commonly employed methods for quantifying the prediction accuracy of the SRN are cosine θ, logistic classification, and the Luce ratio/softmax function. All of these methods yield comparable prediction accuracy endorsement rates.

Mechanisms of non-adjacent learning

For a network to correctly learn non-adjacent contingencies it must have a mechanism by which to integrate information over many time steps. The context layer in the SRN specifically serves the role of retaining information from previous time steps. The context layer directly contributes to the internal representation of each input stimulus, expanding the internal representation to contain prediction-relevant contingencies from prior states (O'Reilly & Munakata, 2000).

Average conditional probability (ACP). To begin to evaluate the internal mechanisms driving non-adjacent and adjacent sequence learning in the SRN , consider the string P V P S from Figure 4. At any point in the string, a logical mechanism for predicting the succeeding letter, given the current letter(s), is to calculate the transitional probability of each possible letter. For the set of all possible sequences, at any point in the sequence, there is a set transition probability to each stimulus in the artificial grammar. The transition probability is a function of the preceding grammar state to each stimulus of the set of all possible stimuli.

The same artificial grammar used to generate the training strings is (typically) used to generate a large number of test strings with statistical properties identical to the training strings. The average conditional probability (ACP) can be calculated for each each occurrence of each predicted letter. The ACP is the probability of observing a particular letter at every node of the grammar over all paths of a given length. For each statistical order of a string of a given length the ACP is computed in three steps. The first step is to estimate the conditional probabilities of observing each letter after each possible path through the grammar. For example, calculating the probability of observing each of the five letters from the grammar in Figure 4 given the example sequence P V P. The second step in calculating the ACP is to compute the probability that each path leads to each node of the grammar. For example, the probability that P T V leads to node 1, 2, etc. Finally, the ACP is calculated *for each letter at each node of the grammar* by summing the products of items in steps 1 and 2 over all possible paths. The ACP's corresponding to letters that cannot appear at particular nodes are excluded (e.g., V at node 0). The result is a set of 11 ACPs (one for each occurrence of the five letters and one for E) for each statistical order (path length).

ACPs as predictor variables. If prediction accuracy is based upon encoding of an increasingly large temporal context, ACPs based on short paths should be better predictors of the SRN 's behavior early in training, while ACP's based on longer paths should be better predictors of behavior later in training. In fact, this is precisely what Servan-Schreiber et al. (1991) demonstrated with an SRN (having fifteen hidden units) trained on the Reber grammar depicted in Figure 4. The authors calculated each set of ACPs from the training sequences presented to the network, then ran a regression analysis with the ACPs as a predictor variable of the SRN's actual output.

As is evident from Figure 6, prior to training, the network's performance was best predicted by the 0th order ACPs, corresponding to the frequency of each letter in the training set. However, in tests at epoch 50 and epoch 100, the network's performance was best accounted for by first-order ACPs (i.e., paths based on length 1). This point in training, Servan-Schreiber et al. (1991) suggest, corresponds to the first phase in SRN learning, during which the network does not distinguish between different occurrences of the same letter. By epoch 200 the SRN's performance was best predicted by second- and third-order ACPs,

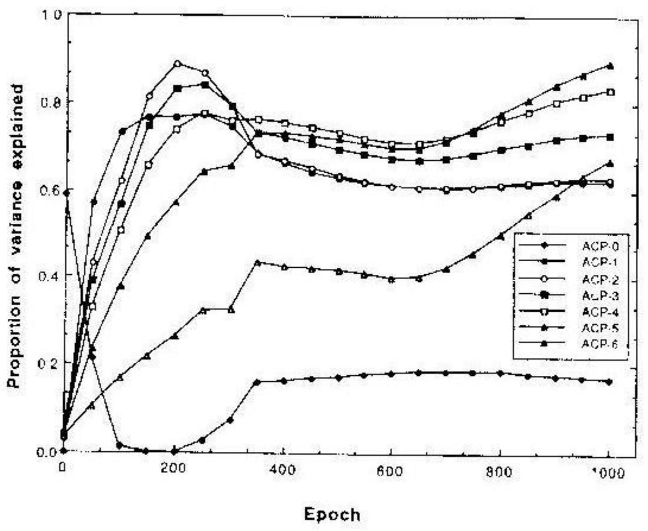

Figure 6. **SRN regression analysis of CPs.** Graphic representation of the percentage of variance in the network's performance explained by average conditional probabilities of increasing statistical order (from 0 to 6). Each point represents the r^2 of a regression analysis using a particular set of average conditional probabilities as the predictor variable, and average activations produced by the network at a particular point in training as the dependent variable. From Servan-Schreiber et al. (1991).

signifying that the network had entered the second phase of learning in which it distinguished between difference occurrences of the same letter. At around epoch 350 the network's behavior was best captured by fourth-order ACPs, where it remained until around epoch 800, when fifth-order ACPs most reliably predicted behavior. The network remained at the fourth-order stage for a much longer period of time than for shorter ACPs, reflecting the difficulty of encoding longer paths. When the network's behavior was reliably predicted by fifth-order ACPs, it had begun to become sensitive to subtler dependencies, such as length constraints, that required encoding the entirety of the traversed path. Although the sixth-order ACP had not dominated in predicting the network's behavior by the 1000th epoch, the curve corresponding to the sixth-order ACP's predictability was steadily rising, and the authors suggested its dominance could only be achieved considerably later in training. Servan-Schreiber et al. (1991) note the large amount of overlap between the percentage of variance explained by the different sets of ACPs, which they suggest is not surprising since most sets of ACPs are partially correlated with each other. Despite the correlation among sets of ACPs, they suggest the results still demonstrate the SRN's successive correspondence to longer temporal contingencies with training.

Long-distance temporal contingencies

Prior work has demonstrated that both the SRN and human *S*s display sensitivity to temporal context and are capable of learning non-adjacent associations. However, in the literature reviewed thus far, the contingencies between sequence elements presented to the SRN were relevant at each processing step. For example, consider the cases where the network needs to predict the last item in the sequence based upon some unknown grammatical structure:

[**1**] GSTU → V

[**2**] HJKL → M

Each of these problems is easily predicted by the SRN because the last letter in each sequence is dependent upon the penultimate letter (i.e., V is contingent on U; M is contingent on L). In each sequence, as long as the context set by the penultimate letters is sufficient to evoke the accurate prediction for the last letter, it is unnecessary for the network to maintain information beyond the immediately prior context. As such, the internal representations associated with the penultimate letter in the example sequences must differ for the network to generate different outputs.

Embedded clauses in finite-state grammars.. In addition to sequences in which the final element is predicted by the penultimate element, natural language processing is also rife with contingencies that contain elements within the body of the sequence that do not predict the final (or later) element(s). Consider, for example, the disjoint temporal contingencies,

[**3**] THE *dog* -THAT ATE THE PILLOW- *is* BORED

[**4**] THE *dogs* <u>-THAT ATE THE PILLOW-</u> *are* BORED

For a simpler example containing embedded elements consider:

[**5**] GSSS → G

[**6**] HSSS → H

The shared letter Ss in each pattern do not supply relevant information for disambiguating the ultimate letter, and the correct predictions for each sequence are identical up to the ultimate letter. In contrast to the sequences in examples [**1**] and [**2**], in which the penultimate letter predicted the ultimate letter, the ultimate letters in sequences [**5**] and [**6**] are predicted by the initial sequence letters, (G and H). To predict the correct successor to the penultimate letter in sequences [**5**] and [**6**], the SRN must develop different internal representations for every letter in each of the sequences. The development of different internal representations of successive presentations of identical items (e.g., each S in sequences [**5**] and [**6**]) is a natural consequence of the recurrent nature of the SRN.

When items are presented to the SRN *both* the input layer and the context layer contribute to the activation across the hidden layer. After the initial time step, t, the context layer shades the input to the network with temporal information from the previous time step(s). With respect to sequences [**5**] and [**6**], the implication is that the initial letters (G and H) can shade subsequent letters in the sequence, inducing different internal representations for each of the letters. Theoretically, contextual shading of each input element's internal representation should enable the SRN to accurately predict the ultimate letter in the sequence.

To evaluate the ability of the SRN to maintain long-distance dependencies on a task more analogous to natural language learning, Servan-Schreiber et al. (1991) tested the network on strings generated by the grammar in Figure 7. The first letter in the string generated by the grammar predicts the final letter. However, Servan-Schreiber et al. (1991) interposed the Reber grammar in Figure 4 (modified to exclude the 'S' and 'T' loops to shorten the average length of the grammatical strings) to function as the embedded clauses.

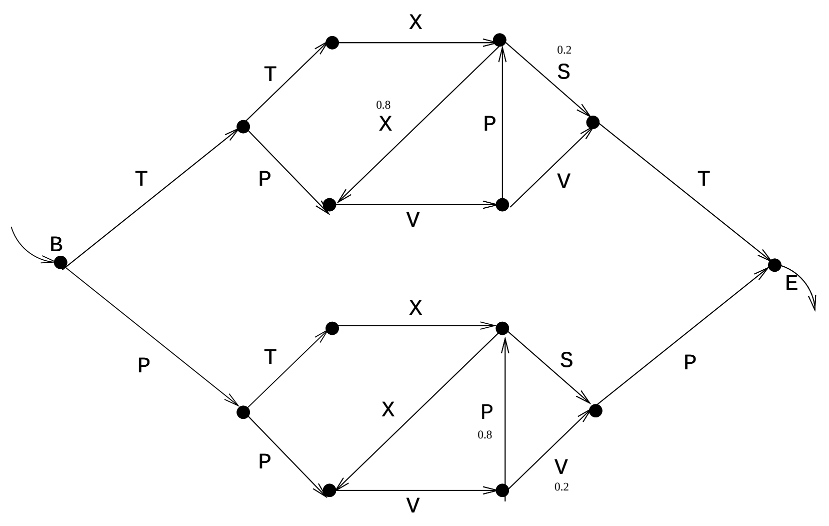

Figure 7. **Finite-state grammar with an embedded clause.** The last letter is contingent on the first letter, while the embedded structure is identical in both branches of the grammar. The grammar can be symmetrical, in which case all arcs are equally likely, or asymmetrical, in which case some of the arcs have different transitional probabilities (i.e., 0.8 and 0.2) in the top and bottom sub-structures.

An SRN with 15 hidden units was first trained on a 'symmetrical grammar', in which all arcs connecting contingencies inside the sub-grammar had the same transitional probability (i.e., 0.5), regardless of the first letter of the string. Network performance was best for sequences with shorter embeddings, such as 3 or 4 letters, and deteriorated as a function of embedding length. Following training on 900,000 grammatical sequences, the SRN only had a 75% correct prediction rate (i.e., the Luce ratio was ≥ 0.6 and the network predicted the correct alternative).

The authors suggested the symmetry of the embedded transitional probabilities not only increased the task difficulty, but did not reflect the nature of natural language. Therefore, they trained an identical network on an 'asymmetrical grammar' (Figure 7). In the asymmetrical version of the grammar the second 'X' arc in the top sub-grammar had a transition probability of 0.8 and the 'S' had a transition probability of 0.2, while the second 'P' in the bottom sub-grammar had a transition probability of 0.8 and the second 'V' had a transition probability of 0.2. All of the other arcs in both the top and bottom sub-grammars had transitional probabilities of 0.5. The SRN was initially trained on the asymmetrical grammar and then tested on the symmetrical version of the grammar. Importantly, while the performance accuracy was 75% with training and testing on the symmetrical grammar, the performance rose to 100% with training on the asymmetrical grammar and testing on the symmetrical grammar. The authors note that performance cannot be attributed to the difference in the transitional probabilities of certain arcs between the top and bottom training sub-grammars because the testing set came from the symmetrical grammar. Instead, they suggest that the SRN demonstrates an advantage for preserving information about the predecessor of an embedded sequence, even across identical embeddings, as long as the potential pathway is differentiated during training.

Cleeremans (1993) corroborated the Servan-Schreiber et al. (1991) results with embedded grammars that network performance suffers if the AGL loops are not differentiated during training. Furthermore, Cleeremans (1993) compared a buffer network with buffer size 4 to an SRN on a grammar with embedded loops and found that the buffer model outperformed the SRN. Unlike the SRN, both the buffer model and human Ss, were better able to learn subtle probabilistic differences over random intervening material. The drop in network prediction performance with increasing embedded string length has led some authors to equate the SRN with a 2nd or 3rd order statistical predictor (Jodouin, 1993).

The network is always in the presence of two times steps of activation, from the the context layer's copy of the representation across the hidden layer at $t-1$, as well as from the projection of the input vector at time step t. Training is expected to surpass this two-step window by developing dimensional compression strategies for retaining information across longer periods. Even with the network that attained 100% performance accuracy with training on the asymmetrical grammar and testing on the symmetrical grammar, accurate network outputs only *suggests* that the network has developed an internal organization of the underlying sequence structure. The SRN's difficulty in retaining information across embedded inputs is attributable to its learning rule. The learning rule imposes pressure to modify connection weights to produce accurate output, and this can compete with the context layer's ability to encode temporal context into the internal representation of each stimulus.

Structure of internal representations.. For each input stimulus, if the network learns to produce the correct output, it is assumed that the internal representations of each input stimulus are the resultant amalgamation of relevant contextual information over multiple time steps. Moreover, satisfactory network generalization to novel inputs is also taken to provide indirect evidence that the structure the network has extracted is consistent with the underlying structure of the sequence material. A plausible conjecture against the contextually rich internal representations hypothesis, it is possible that the context units encode only the prior context (i.e., the preceding stimulus), while continually updating a counter of the total number of letters presented (Servan-Schreiber et al., 1991).

Direct analysis of the internal hidden layer state space trajectories can elucidate the processes underpinning the representations derived from structured material, without appealing to traditional linguistic concepts (Čerňanský, Makula, & Beňušková, 2007; Servan-Schreiber et al., 1991). Although the representations across the input and output layers are readily interpretable, the hidden layer computes dimensional reduction upon the initial input representations, rendering them not obviously interpretable. Thus, addressing whether the hidden layers develop sensible internal representations that express regularities of the sequence structure that are only implicit in the input stimuli, requires more complicated techniques than those used used to evaluate network output performance.

To parse the representations that encode sequential context, a cluster analysis is typically performed on the hidden unit patterns evoked by each sequence. First, each item (e.g., letter) of the stimulus sequence is presented to the SRN and the corresponding pattern of activation across the hidden layer is recorded. Second, the Euclidean distance between each pair of patterns is computed and the matrix of all distances is provided as input to a cluster analysis program. Cluster analysis is a tool that locates the optimal partition of a set of vectors according to some measure of similarity, such as Euclidean distance. Graphical representations of the clusters depict contrast between two groups of clusters by the length of the vertical axis (with the horizontal axis length not meaningful). Servan-Schreiber et al. (1991) applied cluster analysis to the hidden layer representations resulting from an SRN trained on an AGL similar to that of Figure 4. They found that the clusters were predominantly organized according to the predictions associated with the corresponding portion of the sequence. The clusters were further divided according to the path traversed up to that point in the input sequence (see also Cleeremans et al., 1989). Elman (1990) trained an SRN on a similar task with input consisting of sentences of words and found distinct clusters of hidden unit activation space that represented underlying grammatical states (e.g., nouns, verbs). In sum, the SRN implicitly encoded the underlying structural representations of structured input stimuli into the weights of the network.

The internal representations developed by the hidden units reflect the influence of both 'bottom-up' and 'top-down' processes. The context in which each letter was produced yields a characteristic shading, or gradation, across the hidden units that is independent of the input-output mapping. Thus, successively presented letters in an input sequence modify the activation pattern of the hidden units, creating a bottom-up, input-driven, influence (Servan-Schreiber et al., 1991). To preserve structural information, this bottom-up influence is countered by a top-down, input-output mapping, influence. The top-down influence is a function of the pressure to produce the target output by the backpropagation learning algorithm. For example, when different letters yield the same prediction (e.g., $T_1 \rightarrow X_1$; $P_2 \rightarrow X_2$; both T_1 and P_2 predict X) the backpropagation algorithm will exert pressure for the SRN to produce similar patterns of activation across the hidden units.

While 'bottom-up' and 'top-down' pressures influence the evolution of internal representations, the architecture of the SRN, in turn, modulates the 'bottom-up' and 'top-down' pressures. Relative to the number of distinct input stimuli, an SRN with a small number of hidden units will restrict the contextual complexity of its internal representations. With fewer hidden units the representational resources are scarce and less prediction-relevant context can be integrated into the representation for each input stimulus. The 'top-down' pressure from the backpropagation algorithm will force the network to develop internal representations exclusively on the basis of the current context. Networks with large numbers of hidden units can develop different, redundant, representations for particular units, while networks with restricted numbers of hidden units typically preclude the development of redundant representations. Servan-Schreiber et al. (1991) demonstrated an interaction between the size of the network and the size of the problem: adding more hidden units had little impact on small problems, but more hidden units had a much larger impact on problems with many intervening repeated elements.

In summary, the SRN initially acquires a gradual sensitivity to an increasing number of elements of the preceding sequence. With continued training, the SRN is eventually able to encode long-distance dependencies by shading internal representations that are responsible for processing common embeddings in otherwise different sequences (Servan-Schreiber et al., 1991). Because it incorporates prior contextual states into the internal representation of each input stimuli, the SRN learns the *structure* of a sequence it is exposed to rather than simply memorizing the input sequence itself. The network extracts structural properties by allocating distinct areas of its hidden unit space to represent categorical properties of the training sequence.

Representational Overlap

In a series of classic forgetting experiments, Barnes and Underwood (1959) demonstrated that normal human forgetting does not occur catastrophically, but is a gradual cognitive process. In contrast to normal human forgetting, the SRN is prone to severe retroactive interference, also called catastrophic forgetting (McCloskey & Cohen, 1989; Boucher & Dienes, 2003; Ratcliff, 1990; French, 1991). The primary cause of catastrophic interference/forgetting in the SRN is overlap among representations at the hidden unit layer (Hetherington, 1991; Kruschke, 1992; McCrae & Hetherington, 1993; French, 1991). In general, representational overlap is a beneficial architectural attribute that enables the SRN to exhibit robust flexibility in the face of the challenges posed by the real world. For example, despite noisy input or the destruction of units, the SRN remains capable of performing similarity-based processing, exhibiting graceful degradation of function. Additionally, the

SRN does not seize if it encounters undefined state transitions, but is capable of generalizing to sequences that were not part of the training set.

Despite enabling the SRN generalizability to novel inputs, as well as rendering it fairly robust against input noise, representational overlap of the stimulus elements across the hidden layer directly contributes to catastrophic interference. Representational overlap is an emergent property of the way in which the SRN learns. Recall that learning in the SRN is the result of altering the weighted connections between units via backpropagation feedback. Many input patterns can be represented in a distributed manner across the internalized hidden layers. Many units and weights contribute to the encoding of each item, thereby enabling hidden units to capture prediction-relevant relations between input stimuli. Due to the superimposition of input stimuli during training (e.g., even completely non-overlapping, orthogonal input vectors, can overlap at the hidden layer (Hetherington, 1991)), retroactive interference can occur. Again, these overlapping representations inherent in the SRN are responsible for both the networks' ability to generalize, as well as the networks susceptibility to catastrophic interference. Thus, catastrophic interference is a radical manifestation of the 'stability-plasticity' problem: how to design a system that is simultaneously sensitive to, but not radically disrupted by, new input (Grossberg, 1982).

Recast in terms of gradient descent, when a distributed network learns to recognize a set of patterns, it has essentially found the minimum in weight-space for which the network can recognize all of the patterns it has been presented. If the network then learns a new set of patterns, regardless of the size of the new set, it will move to a new solution point in weight-space corresponding to a set of weights that allows the network to recognize the new patterns. Catastrophic interference occurs when the new weight vector is suboptimal, or even completely inappropriate, as a solution for the originally learned patters (French, 1999). The existence of catastrophic interference suggests the presence of a non-smooth weight-space, fraught with ravines that radically disrupt prior learning when moving only a small distance across the weight-landscape.

If most or all hidden units encode initial input stimuli, then later learning will involve changing the very weights used to encode the initial stimuli. Therefore, if an SRN is constrained to allocate a limited subset of hidden units to encode initial items, then the overlap between the initial and latter stimuli should be reduced and retroactive interference should be correspondingly reduced (McCrae & Hetherington, 1993). Most techniques to reduce catastrophic interference are aimed at reducing representational overlap by orthogonalizing hidden layer activations. French (1991) used semi-distributed representations by explicitly decreasing the internal representational overlap with 'sparse vectors' in which only a few hidden layer units were active. An additional step is added into the backpropagation algorithm: the most active hidden unit(s) is increased slightly for each pattern ('sharpened'), while the activations of other units are decreased.

A natural solution for addressing catastrophic interference by constraining the hidden layer to represent limited aspects of the input stimuli is to pretrain the SRN. McCrae and Hetherington (1993) noted that college students, who pose a large body of prior knowledge, typically serve as subjects in sequential learning experiments. However, most networks begin training with no prior training or knowledge built into the network. If a network is pretrained to possess a body of knowledge prior to a sequential learning task, then the sequential learning stimuli must be learned within the constraints from prior knowledge (McCrae & Hetherington, 1993; Elman, 1993). In so doing, the hidden units should theoretically be committed to representing only a limited aspect of the input space by virtue of the constraints imposed on the hidden layer by strong weights from a limited number of input units. McCrae and Hetherington (1993) found that with the benefit of prior knowledge, generalizability remained intact and because the probability of overlap at the hidden layer was reduced, catastrophic interference was all but eliminated from the networks; training on a corpus of items naturally constrained hidden units' receptive fields and sharpened their activations. However, there was less interference in the pretrained networks than is typically found with humans in analogous sequential learning experiments.

Using a variant of a pretraining simulation, Elman (1993) trained an SRN on a language acquisition prediction task. The SRN was trained on a sequence of words, input as vectors of 0's with a single bit randomly set to 1. The prediction task was dependent upon a grammatical structure that involved multiple embeddings. For example, sentences with multiple embeddings could have included "BOYS WHO CHASE DOGS SEE GIRLS", or "DOGS SEE BOYS WHO CATS WHO MARY FEEDS CHASE", while simpler sentences without embeddings were also generated by the grammar, "CATS CHASE DOGS". The grammar was initially unlearnable by an "adult network", a network with full access to the entire training corpus and unconstrained hidden units. In fact, the adult network failed to master the task for the training data, though performance was not uniformly bad. The adult network successfully coordinated the number of the main clause subject mentioned early in the sentence, though would fail to get the agreement correct on some of the relative clause subjects and verbs. For example, the adult SRN may predict *The boys who the girl chase see the dog.* In this example, the SRN would have correctly predicted the number agreement of *boys* and *see*, but incorrectly predicted the more proximal *girl chases*.

To successfully train an SRN to predict sequence elements from the embedded grammatical structure, Elman (1993) organized the training input into corpora of increasing complexity and first trained the network with the simplest input. Training consisted of five phases, with phase one composed of 10,000 simple sentences presented for five epochs. After phase one training data were discarded and new sentences, consisting of 7,500 simple sentence and 2,500 complex sentences were likewise presented. This procedure continued until the fifth trial when the network was trained on 10,000 complex sentence. Importantly, the results from the incremental input training contrasted sharply with the failure of the network to learn when presented with the full corpus. Although the network failed to learn the complex grammar when presented with the full "adult" corpus, with incremental training the SRN mastered both the simple and the complex sentences.

Biological realism.

Elman (1993) suggested that the fact that the SRN performs better when incrementally trained on an adult corpus is at least in some sense similar to natural language learning in children. It can be argued, however, that the SRN is still implausible on both biological and cognitive grounds. The biological argument against the plausibility of the SRN begins with the fact that the network employs the backpropagation gradient descent algorithm. Specifically, the network is given a sequence of inputs and produces an output. First, this output is compared to a target output and a measure of error is calculated between the outputs. Then the error is minimized following the negative gradient of the error as a function of all network parameters, and the network weights are subsequently adjusted. However, this credit assignment problem is a nontrivial task, particularly when weight updates are to be computed from quantities that are locally available at the unit the weight belongs to (i.e., in a biologically plausible manner). Backpropagation bypasses this issue via bidirectionality: activity is propagated forward in the network and each unit's contribution to the total error is propagated backward for weight modification (Grüning, 2007). There is a gap between backpropagation and biological reality, however, with no known evidence of any biological model implementing backpropagation learning.

The cognitive plausibility of the SRN, and its backpropagation gradient descent algorithm, is questionable because it is a *supervised* learning scheme in which a 'teacher' must provide a fully specified answer in the form of the target output to which the network output can be compared. In reality, agents in a natural environment often receive summary feedback about the *degree* of success or failure, rather than a fully specified correct answer.

Despite this discontinuity there are two prominent reasons for tolerating a gap between "the model and the meat" (Elman et al., 1996). First, consider the "sigma-pi" units first proposed by Feldman and Ballard (1982) and Rumelhart et al. (1986a). There was no biological evidence for the existence for these multiplicative synapses, though they were demonstrated to be useful in simplifying circuitry (Durbin & Rumelhart, 1989; Poggio & Girosi, 1990). In principle it was argued there was no evidence that synapses should have the properties of the sort described as sigma-pi units, but such synapses have since been discovered in the brain. Outright rejection of the sigma-pi units would have been premature, especially if modeling is not to remain several paces behind the current state of the sciences

(Elman et al., 1996).

A second reason for tolerating a gap between the model and the meat is the difficulty in ascertaining the functional role specified by specific neural mechanisms. Modeling permits analysis of component interactions and emergent properties. The precise details of implementation may often be approximations towards biological functions. For example, some neural systems may do gradient descent learning in a manner that is functionally similar to backpropagation, even if backpropagation is not the exact mechanism.

The SRN provides a natural formalism for executing many of the operations characteristic of biological systems (e.g., nonlinear responses, the time-dependent nature of learning, the problem of control, and the nature of representation). However, the SRN is not particularly germane to implementing certain functions, such as truly recursive functions, calculus, nor to doing many basic mathematical operations (Elman et al., 1996). Fodor and Pylyshyn (1988) even argue that connectionist models cannot implement strict recursion nor support syntactically composed representations, though Elman et al. (1996) argue that the conclusion that connectionist networks are then insufficient to capture aspects of human cognition is a sweeping judgment.

Direction

Since the backpropagation mechanism is a universal approximator it will not serve as a suitable alternative to a cognitive model. Backpropagation can be useful for non-parametric parameter estimation if it is possible to represent the mathematical/cognitive model in backpropagation terms. However, the parameters necessary for the backpropagation algorithm, weights and activation and bias, are microscopic, whereas the psychologically interesting parameters of associative memory models (e.g., TODAM Murdock, 1982), apply to updates of the whole memory system at a macroscopic level. In general, the SRN offers an elegant account of a large body of sequence learning data, but is subject to interference and human Ss demonstrate greater sensitivity to long distance dependencies than the SRN. Memory buffer models can better fit human long distance dependency data, however, because buffer models are also governed by error correction, they are similarly subject to interference. In short, the SRN can provide an rigorously simulated framework in which to examine sequential learning, toward the end of developing a model that simultaneously accounts for SRN simulated data, as well as models long-distance, embedded, non-adjacent dependencies.

References

Altmann, G. T. M., & Dienes, Z. (1999). Rule learning by seven-month-old infants and neural networks. *Science*, *284*, 875.

Aslin, R. N., Saffran, J. R., & Newport, E. L. (1998). Computation of conditional probability statistics by human infants. *Psychological Science*, *9*, 321-324.

Barnes, J. M., & Underwood, B. J. (1959). Fate of first-list associations in transfer theory. *Journal of Experimental Psychology*, *58*, 97-105.

Berry, D. C., & Dienes, Z. (1993). In C. A. Umilta & M. Moscovitch (Eds.), *Implicit learning: Theoretical and empirical issues.* Hove: England: Lawrence Erlbaum Associates.

Boucher, L., & Dienes, Z. (2003). Two ways of learning associations. *Cognitive Science: A Multidisciplinary Journal*, *27*(6), 807-842.

Čerňanský, M., Makula, M., & Beňušková, L. (2007). Organization of the state space of a simple recurrent network before and after training on recursive linguistic structures. *Neural Networks*, *20*(2), 236-44.

Cleeremans, A. (1993). *Mechanisms of implicit learning: Connectionist models of sequence processing.* Cambridge, MA, USA: The MIT Press.

Cleeremans, A., & Dienes, Z. (2008). Computational models of implicit learning. In R. Sun (Ed.), *Cambridge handbook of computational psychology* (p. Ch. 14). Cambridge University Press.

Cleeremans, A., & McCleeland, J. L. (1991). Learning the structure of event sequences. *Journal of Experimental Psychology: General*, *120*, 235-253.

Cleeremans, A., Servan-Schreiber, D., & McClelland, J. L. (1989). Finite state automata and simple recurrent networks. *Neural Computation*, *1*(3), 372-381.

Cohen, A., Ivry, R. I., & Keele, S. W. (1990). Attention and structure of sequence events. *Journal of Experimental Psychology : Learning, Memory, and Cognition*, *16*, 17-30.

Creel, S. C., Newport, E. L., & Aslin, R. N. (2004). Distant melodies: Statistical learning of non-adjacent dependencies in tone sequences. *Journal of Experimental Psychology : Learning, Memory, and Cognition*, *30*, 1119-1130.

Curran, T. (1997). Higher-order associative learning in amnesia: Evidence from the serial reaction time task. *Journal of Cognitive Neuroscience*, *9*, 522-533.

Dienes, Z., Altmann, G. T. M., & Gao, S.-J. (1999). Mapping across domains without feedback: A neural network model of implicit learning. *Cognitive Science*, *23*, 53-82.

Durbin, R., & Rumelhart, D. E. (1989). Product units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation*, *1*, 133.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*, 179-211.

Elman, J. L. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, *7*, 195-225.

Elman, J. L. (1993). Learning and development in neural networks: the importance of starting small. *Cognition*, *48*(1), 71-99.

Elman, J. L., Bates, E. A., Johnson, M. H., Karmiloff-Smith, A., Parisi, D., & Plunkett, K. (1996). *Rethinking innateness: A connectionist perspective on development.* Cambridge, MA: MIT Press.

Feldman, J., & Ballard, D. (1982). Connectionist models and their properties. *Cognitive Science*, *6*(3), 205-254.

Fiser, J., & Aslin, R. N. (2001). Unsupervised statistical learning of higher-order spatial structures from visual scenes. *Psychological Science*, *12*(6), 499-504.

Fiser, J., & Aslin, R. N. (2002). Statistical learning of higher-order temporal structure from visual shape sequences. *Journal of Experimental Psychology : Learning, Memory, and Cognition*, *28*(3), 458-67.

Fodor, J. A., & Pylyshyn, Z. W. (1988). Connectionisism and cognitive architecture: A critical analysis. *Cognition*, *28*(1), 3-71.

French, R. M. (1991). Using semi-distributed representations to overcome catastrophic interference in connectionist networks. *Proceedings of the 13th Annual Conference of the Cognitive Science Society*, 173-178.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in Cognitive Science*, *3*(4), 128-135.

Garson, J. (2007). Connectionism. In E. N. Zalta (Ed.), *The stanford encyclopedia of philosophy.*

Gomez, R. (2002). Variability and detection of invariant structure. *Psychological Science*, *13*(5), 431-436.

Grossberg, S. (1982). *Studies of Mind and Brain: Neural Principles of Learning, Perception, Development, Cognition, and Motor Control.* Boston, MA: Reidel.

Grüning, A. (2007). Elman backpropagation as reinforcement for simple recurrent networks. *Neural Computation*, *19*(11), 3108-31.

Hare, M., Elman, J. L., & Daugherty, K. G. (1995). Default generalization in connectionist networks. *Language and Cognitive Processes*, *10*(6), 601-630.

Hauser, M. D., Newport, E. L., & Aslin, R. N. (2001). Segmentation of the speech stream in a non-human primate: statistical learning in cotton-top tamarins. *Cognition*, *78*(3), B53-64.

Hetherington, P. A. (1991). *Master's thesis: The sequential learning problem in connectionist networks.* Montreal, Quebec, Canada: McGill Univeristy.

Hinton, G. E. (1986). Learning distributed representations of concepts. *Proceedings of the 8th Annual Conference of the Cognitive Science Society*, 1-12.

Jackson, G. M., & Jackson, S. R. (1995). Do measures of explicit learning actually measure what is being learnt in the serial reaction time task? *Psyche*, *2*(20).

Jodouin, J. F. (1993). Putting the simple recurrent network to the test. *IEEE International Conference on Neural Networks*, *2*, 1141-1146.

Jordan, M. I. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. *Proceedings of the 8th Annual Conference of the Cognitive Science Society*, 531-546.

Kinder, A., & Shanks, D. R. (2001). Amnesia and the declarative/nondeclarative distinction: a recurrent network model of classification, recognition, and repetition priming. *Journal of Cognitive Neuroscience*, *13*(5), 648-69.

Kirkham, N. Z., Slemmer, J. A., & Johnson, S. P. (2002). Visual statistical learning in infancy: evidence for a domain general learning mechanism. *Cognition*, *83*(2), B35-42.

Knowlton, B. J., & Squire, L. R. (1996). Artificial grammar learning depends on implicit acquisition of both abstract and exemplar-specific information. *Journal of Experimental Psychology : Learning, Memory, and Cognition*, *22*(1), 169-81.

Kolen, J. F., & Pollack, J. B. (1990). Back-propagation is sensitive to initial conditions. *Complex Systems*, *4*, 269-280.

Kruschke, J. K. (1992). Alcove: an exemplar-based connectionist model of category learning. *Psychological Review*, *99*(1), 22-44.

Kuhn, G., & Dienes, Z. (2008). Learning non-local dependencies. *Cognition*, *106*(1), 184-206.

McCloskey, M., & Cohen, N. (1989). Catastrophic interference in connectionist networks: The sequential learning problem (vol. 24). In G. H. Bower (Ed.), *The psychology of learning and motivation* (p. 109-164). Academic Press.

McCrae, K., & Hetherington, P. A. (1993). Catastrophic interference is eliminated in pretrained networks. *Proceedings of the 15th Annual Conference of the Cognitive Science Society*, 723-728.

Murdock, B. B. (1982). A theory for the storage and retrieval of item and associative information. *Psychological Review*, *89*, 609-626.

Newport, E. L., & Aslin, R. N. (2004). Learning at a distance i. statistical learning of non-adjacent dependencies. *Cognitive Psychology*, *48*(2), 127-62.

Newport, E. L., Hauser, M. D., Spaepen, G., & Aslin, R. N. (2004). Learning at a distance ii. statistical learning of non-adjacent dependencies in a non-human primate. *Cognitive Psychology*, *49*(2), 85-117.

Nissen, M. J., & Bullemer, P. (1987). Attentional requirements for learning: Evidence from performance measures. *Cognitive Psychology*, *19*, 1-32.

Onnis, L., Monaghan, P., Christiansen, M. H., & Chater, N. (2004). Variability is the spice of learning, and a crucial ingrediant for detecting and generalizing in nonadjacent dependencies. *Proceedings of the 26th Annual Conference of the Cognitive Science Society*, 1047-1052.

O'Reilly, R. C., & Munakata, Y. (2000). *Computational explorations in cognitive neuroscience: Understanding the mind by simulating the brain.* Cambridge, MA: MIT Press.

Perruchet, P., & Pacteau, C. (1990). Synthetic grammar learning: Implicit rule abstraction or explicit fragmentary knowledge? *Journal of Experimental Psychology: General*, *119*, 264-275.

Perruchet, P., Tyler, M. D., Galland, N., & Peereman, R. (2004). Learning nonadjacent dependencies: no need for algebraic-like computations. *Journal of Experimental Psychology: General*, *133*(4), 573-83.

Plaut, D. C. (2000). Connectionist modeling. In A. E. Kazdin (Ed.), *Encyclopedia of psychology.* Washington, DC: American Psychological Association.

Poggio, T., & Girosi, F. (1990). Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, *247*(4945), 978-982.

Ratcliff, R. (1990). Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological Review*, *97*(2), 285-308.

Reber, A. S. (1967). Implicit learning of artificial grammars. *Journal of Verbal Learning and Verbal Behavior*, *6*, 855-863.

Reber, A. S. (1989). Implicit learning and tacit knowledge. *Journal of Experimental Psychology: General*, *118*, 219-235.

Reber, P. J., & Squire, L. R. (1994). Parallel brain systems for learning with and without awareness. *Learning and Memory*, *1*(4), 217-29.

Reed, J., & Johnson, P. (1994). Assessing implicit learning with indirect tests: Determining what is learnt about sequence structure. *Journal of Experimental Psychology : Learning, Memory, and Cognition*, *20*, 585-594.

Rodriguez, P. (2001). Simple recurrent networks learn context-free and context-sensitive languages by counting. *Neural Computation*, *13*(9), 2093-118.

Rodriguez, P. (2003). Comparing simple recurrent networks and n-grams in a large corpus. *Applied Intelligence*, *19*(1-2), 39-50.

Rumelhart, D. E., McClelland, J. L., & PDP Research Group the. (1986a). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Volume 1: Foundations.* Cambridge, MA, USA: The MIT Press.

Rumelhart, D. E., McClelland, J. L., & PDP Research Group the. (1986b). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Volume 2: Psychological and Biological Models.* Cambridge, MA, USA: The MIT Press.

Saffran, J. R. (2003). Satistical language learning: Mechanisms and constraints. *Current Directions in Psychological Science*, *12*, 110-114.

Saffran, J. R., Aslin, R. N., & Newport, E. L. (1996). Statistical learning by 8-month-old infants. *Science*, *274*(5294), 1926-8.

Saffran, J. R., Johnson, E. K., Aslin, R. N., & Newport, E. L. (1999). Statistical learning of tone sequences by human infants and adults. *Cognition*, *70*(1), 27-52.

Saffran, J. R., Newport, E. L., Aslin, R. N., Tunick, R. A., & Barrueco, S. (1997). Incidental language learning: Listening (and learning) out of the corner of your ear. *Psychological Science*, *8*, 101-105.

Servan-Schreiber, D., Cleeremans, A., & McClelland, J. L. (1991). Graded-state machines: The representation of temporal contingencies in simple recurrent networks. *Machine Learning*, *7*, 161-193.

Shanks, D. R., & St. John, M. F. (1994). Characteristics of dissociable human learning systems. *Behavioral and Brain Sciences*, *17*, 367-447.

Stadler, M. A. (1993). Implicit serial learning: Questions inspired by hebb (1961). *Memory & Cognition*, *21*(6), 819-27.

Werbos, P. J. (1990). Backpropagation through time: What it does and how we do it. *Proceedings of the IEEE*, *78*(10), 1550-1560.

Werbos, P. J. (1994). *The roots of backpropagation from ordered derivatives to neural networks and political forecasting.* New York: Wiley Intersciencera.