Representation Beyond Finite States: Alternatives to Push-Down Automata

Janet Wiles, Alan D. Blair and Mikael Bodén

1 Introduction

It has been well established that Dynamical Recurrent Networks (DRNs) can act as deterministic finite-state automata (DFAs — see Chapters 6 and 7). A DRN can reliably represent the states of a DFA as regions in its state space, and the DFA transitions as transitions between these regions. However, as we shall see in this chapter, DRNs can learn to process languages which are non-regular (and therefore cannot be processed by any DFA). Moreover, DRNs are capable of *generalizing* in ways which go beyond the DFA framework.

We will show how DRNs can learn to predict context-free and context-sensitive languages, making use of the transient dynamics as the network activations move towards an attractor or away from a repeller. The resulting trajectory can be thought of as analogous to winding up a spring in one dimension and unwinding it in another. In contrast to push-down automata, which rely on unbounded external memory, DRNs must instead rely on arbitrary precision in order to process strings of arbitrary length from non-regular languages. The issue of robustness and precision will be discussed in relation to learning and generalization as well as Chomsky's competence/performance dichotomy.

1.1 Finding structure in continuous state space

There are many situations in which a DRN can properly be analyzed as implementing a DFA (see Chapter 7). However, in processing a sequence, the state space of a DRN does not always partition neatly into functionally discrete regions. This point is critical to understanding the source of the power of DRNs to go beyond the representational ability of DFAs, so is worth elaborating here.

A sequence of symbols processed by a DRN corresponds to a trajectory through its state space. The DRN state space is typically defined over the real numbers — which, given unbounded precision, theoretically gives rise to an unbounded number of possible states (which we will refer to as points in the state space). In emulating a DFA, points in the DRN state space cluster into functionally equivalent regions, such that all points within one region generate the same sequence of outputs for all possible future inputs. This definition of functional equivalence — based on the response of the system to future inputs — derives from the definition of an information processing state which underlies the theoretical foundations of automata theory (Hopcroft and Ullman, 1979). Conversely, if a sequence of inputs causes trajectories from one region of the DRN state space to diverge so that different outputs are produced at some point in the future, then this region cannot be considered as a single functional state. Functional equivalence also lies at the heart of extracting DFAs from DRNs (see Chapters 7 and 12).

In this chapter we analyze the behavior of DRNs in which trajectories through the DRN state space do not partition neatly into a finite number of discrete, functionally homogeneous regions. Many DRNs show sensitivity to initial conditions

Language	Machine	Example
recursively enumerable	Turing machine	true QBFs
context-sensitive	linear bounded automaton	$a^n b^n c^n$
context-free	push down automaton	$a^n b^n$
regular	finite state automaton	a^n (n odd)

Table 1: Chomsky hierarchy: correspondence between classes of languages and classes of machines. QBF stands for "Quantified Boolean Formula" (see (Hopcroft and Ullman, 1979), Section 13.4).

such that trajectories passing through arbitrarily close points in the DRN state space later diverge to produce different outputs (Kolen, 1994). No finite discretization of the state space can capture the behavior of such DRNs. We will demonstrate such trajectories, explain their computational power, and show how that power allows them to go beyond DFAs. Note that in this chapter we are concerned with practical demonstrations of DRN computing abilities. Theoretical considerations of their computational limits are considered in Chapter 9.

2 Hierarchies of languages and machines

In this section we briefly diverge from DRNs to review computational hierarchies of machines and the tasks they can compute, as a basis for understanding the relative difficulty of the tasks discussed in later sections.

2.1 The Chomsky hierarchy

The simplest kind of languages are the *regular* languages (as described in earlier chapters). Chomsky pointed out that new, more powerful language classes were needed in order to model the essential properties of human languages [Chomsky, 1956, 1959]. The hierarchy of language classes which he introduced has now become the cornerstone of formal language theory.

DEFINITION: A (formal) *language* $\mathcal{L}$ is a (possibly infinite) set of strings of symbols drawn from a finite alphabet $\mathcal{A}$.

There are two standard ways to characterize a formal language. One way is to specify a grammar (set of re-write rules) which *generates* it (i.e. produces all the strings of the language, and no others); the other way is to specify an abstract machine which *recognizes* it (i.e. tells whether any given string is in the language or not). In the present context, it will be convenient to define the four classes of languages that make up the Chomsky hierarchy — regular, context-free, context-sensitive and recursively enumerable – in terms of the classes of machines recognizing them — which are, respectively, finite state automata (DFA), push-down automata (PDA), linear bounded automata (LBA) and Turing machines (TM). Note however that the traditional approach has been to define each class of language in terms of the class of grammars generating it, and then to prove the equivalence of the corresponding machine-based characterizations (see (Hopcroft and Ullman, 1979), for further details).

2.2 Regular and context-free languages

The definition of a DFA was given in Chapter 7; a PDA is essentially a DFA which is additionally able to push and pop symbols from an external stack. The power of a PDA to go beyond a DFA derives from the unbounded stack which, unlike the states of a DFA, is not constrained to a finite size. Our notation is taken from (Hopcroft and Ullman, 1979).

DEFINITION: A *push-down automaton* (PDA) is a system $\mathrm{M} = \{Q, \Sigma, \Gamma, q_0, z_0, F, \delta\}$, where:

Q is a finite set of states;
Σ is an alphabet, called the *input alphabet*;
Γ is an alphabet, called the *stack alphabet*;
q_0 in Q is the *initial state*;
z_0 in Γ is a particular stack symbol, called the *start symbol*;
$F \subseteq Q$ is the set of *final* states;
δ is a mapping from $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$.

The interpretation of δ is as follows: if (q, γ) is an element of $\delta(p, a, z)$ then the PDA can, on reading input a, move from state p to state q, popping z from the stack and pushing the symbols of the string γ onto the stack. The empty string is denoted ε. If $(q, \gamma) \in \delta(p, \varepsilon, z)$ then the PDA can move from state p to state q, pop z and push γ, without reading any input. M is called a *deterministic* PDA if two additional conditions are satisfied:

(i) $\delta(q, a, z)$ contains at most one element, for all $a \in \Sigma \cup \{\varepsilon\}$,
(ii) if $\delta(q, \varepsilon, z)$ is not empty, then $\delta(q, a, z)$ is empty for all $a \in \Sigma$.

A string α is *recognized* by M if M, starting in state q_0 with the single symbol z_0 on the stack, can read the symbols of α one at a time by some series of moves, and finish in one of the designated final states, F.

DEFINITION: A language which can be recognized by a PDA is called a *context-free language* (CFL). A language which can be recognized by a deterministic PDA is called a *deterministic* context-free language.

As an example, consider the language $\mathcal{L} = a^n b^n$, which consists of the strings ab, $aabb$, $aaabbb$, ... for all $n \geq 0$. In this language, every string has exactly the same number of b's as a's. We cannot build a DFA to recognize $\mathcal{L}$, because it would be necessary to have a different state for each value of n (which is impossible by definition, since the DFA only has a finite number of states). However, a PDA can be constructed which recognizes $a^n b^n$ by pushing symbols onto the stack as it "counts" the number of a's, and then popping those symbols from the stack one at a time as it checks to see that there are the same number of b's. Since the stack depth is unbounded, the PDA is not limited to a finite number of a's and b's.

Virtually all computer languages are context-free. Natural languages have much in common with CFLs (but are not quite the same). Examples of structures which the stack of a PDA can be used to process include embedded clauses in natural language, nested loops in computer languages and balanced parentheses in formal languages.

2.3 Context-sensitive and recursively enumerable

In order to define the remaining two classes of the hierarchy, it is necessary to introduce the Turing machine, which is essentially a DFA that is additionally able to read and write to an external "tape".

DEFINITION: A *Turing machine* is a system $M = \{Q, \Sigma, \Gamma, q_0, B, F, \delta\}$, where:

Q is a finite set of states;
Γ is the finite set of allowable *tape symbols*,
B is a special symbol in Γ, called the *blank*,
Σ, a subset of Γ not including B, is the set of *input symbols*,
q_0 in Q is the *initial state*;
$F \subseteq Q$ is the set of *final* states;
δ is a mapping from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, r\}$
(δ may, however, be undefined for some arguments).

The following procedure is used to determine whether a given string α is accepted by machine M. Initially, the string α is written on the left end of the tape one symbol at a time. The rest of the tape (which extends infinitely to the right) is filled with blanks. The machine then performs a series of moves, governed by the mapping δ. At each step, if M is in state p and reads x from the tape, M will change to state q, write y to the tape and move the tape to the left [resp. right] if $\delta(p, x) = (q, y, L)$ [resp. (q, y, R)]. If the computation eventually halts in one of the designated final states, the string is accepted.

DEFINITION: A *linear bounded automaton* (LBA) is a Turing machine which is restricted to only writing on a portion of the tape whose length is bounded by some linear function of the length of the input string.

DEFINITION: A language which can be recognized by an LBA is called a *context-sensitive language* (CSL); a language which can be recognized by a Turing machine is called *recursively enumerable*.

Turing machines are *universal* in the sense that any symbolic computation system can be emulated by a Turing machine, and vice-versa. The set of recursively enumerable languages therefore includes any language which can be described by a computable function.

It can be shown that each class in the Chomsky hierarchy is properly contained in the next class. For example, the language of true Quantified Boolean Formulas is recursively enumerable but not context-sensitive (see (Hopcroft and Ullman, 1979), section 13.4). An example of a language which is context-sensitive but not context-free is the language $a^n b^n c^n$, consisting of the strings abc, $aabbcc$, $aaabbbccc$, etc. PDAs are not able to process this language because they cannot "re-use" information once it has been popped off the stack. Interestingly, if a second stack is added to a PDA, the resulting system acquires the full computing power of a Turing machine (because each stack can be used to simulate one semi-infinite half of the tape). Natural languages do occasionally make use of constructs requiring more than a simple stack, but their use is rare; for example, we can say "John, Paul and Ringo played guitar, bass and drums, respectively".

2.4 Learning and induction

The remarkable thing about human language processing is our ability, once we have "learned" a language, to produce new utterances which are accepted and understood by others, even though those exact utterances have never been written or spoken before. This process of language *induction* can be modeled with formal languages using formal prediction or recognition tasks.

Prediction task:

A learning system is presented with a number of *training* strings, all of which belong to the language. The system is then presented with a number of *test* strings (which also belong to the language) one symbol at a time, and must predict, at each step, what the next symbol in the sequence will be.

Recognition task:

A learning system is presented with a number of training strings — each string labeled to indicate whether or not it is in the language. The system is then presented with a number of test strings and must declare, for each of these strings, whether or not it is in the language.

Language induction depends on certain assumptions about the framework within which the language is likely to be found. Suppose, for example, that a learning system is presented with the training strings $a^n b^n$ for $0 \leq n \leq 10$, and asked to induce a language from these data. It is of course possible to build a DFA (with 22 states) which would accept exactly these 11 strings and no others. However, we might be more impressed with a learning system which can generalize the pattern of the input, and learn to accept new strings of the form $a^n b^n$ for values of n greater than 10. Note that this kind of generalization necessarily steps outside the DFA paradigm, since it involves adding additional states to the DFA. We will see in Sections 3.2 and 3.4 that DRNs are indeed able to generalize in this way, for both recognition and prediction tasks. Therefore, with respect to generalization, DRNs have properties which go beyond the DFA framework.

2.5 Alternative hierarchies

Chomsky's hierarchy had substantial impact, particularly in the area of theoretical computer science. The establishment of a correspondence between classes of languages that can be expressed with a given syntax, and classes of machines expressed in terms of memory requirements, allows a deep understanding of these classes from the two different perspectives.

However, Chomsky's language hierarchy is not the only one that has been proposed, and linguists have argued that natural languages do not neatly fit into any particular class in the Chomsky hierarchy. Note in particular that the four classes of machine used to define the Chomsky hierarchy have in common that they are essentially discrete or *symbolic* in nature, and are distinguished by the type of memory resources available to them (no external memory, an external stack, a linear bounded tape, or an infinite tape). Researchers have more recently begun to study language classes which correspond to dynamical systems (Moore, 1998). Dynamical systems are not symbolic in nature, but continuous, and the limitation on their power is not naturally defined in terms of external memory, but rather precision, as we shall see in the next section.

2.6 Performance limitations

The remainder of this chapter will be devoted to a description of how DRNs can learn to process context-free and context-sensitive languages. There is one caveat we must mention first, namely the issue of *robustness*.

Physically implemented computation devices do not completely live up to the capabilities of idealized constructs such as PDAs or Turing machines. Strictly speaking, a digital computer with finite memory is really a finite state machine, and therefore resides at the lowest rung of the Chomsky hierarchy (for example, a four-gigabyte computer has $2^{2^{35}}$ possible states). However, because of the exponentially large number of states, it is more useful to think of a computer as if it were implementing a Turing machine, but keeping in mind that it is constrained by *performance limitations*. This typically means that it can be relied upon to perform any computation perfectly, unless it is provided with input greater than a certain critical size, in which case it will either produce a wrong answer, or halt with an "out of memory" or "stack overflow" error message. This is known as *catastrophic failure*.

An analog computer such as a DRN is also subject to performance limitations — not because of finite memory space but because of limited *precision*; a physically implemented DRN is subject to *noise*, which causes tiny random fluctuations in the activation levels of the units. Rather than catastrophic failure, these fluctuations will typically produce a pattern of *graceful degradation*. This means that the DRN will sometimes make mistakes, and that the probability of making a mistake gradually increases as the input gets longer and more complex, representing a soft limit rather than a hard one. Since DRNs rely on an infinitesimally fine subdivision of their state space in order to process non-regular languages, this process is susceptible to errors when noise is introduced into the system. In other words, although DRNs are capable of implementing non-regular languages, it has been shown that they are not able to do so *robustly* (Casey, 1995; Maass and Orponen, 1998). We will return to this point in section 3.2.

3 DRNs and non-regular languages

3.1 Augmenting DRNs with an external stack

Inspired by the formal description of a PDA, LBA or Turing machine as a DFA augmented by a stack or tape, the early work in training DRNs on context-free languages involved similarly augmenting the DRN with an external stack or tape. DRNs have been shown to learn to operate the read, write and shift transitions for a Turing machine. (**?**).

Further integration of the stack into the DRN is achieved in the Neural Network Pushdown Automaton (NNPDA) which employs a continuous stack with differentiable dynamics. The network in the NNPDA can be trained to induce the state transition rules and to use and manipulate the stack (Giles et al., 1990; Das et al., 1993; Sun et al., 1998). The NNPDA implements a state-to-state mapping

$$S_{t+1} = G_s(S_t, R_t, I_t; W_s), \quad (1)$$

where S is the continuous state, R is what is available on the continuous stack, I is the input symbol (represented as a bit vector) and W_s is the weight matrix. Moreover, the NNPDA determines its next stack action by

$$A_{t+1} = G_a(S_t, R_t, I_t; W_a), \quad (2)$$

where A is the stack operation on the current input symbol: push, pop or none, together with its *length* (utilizing the continuum of A) which corresponds to the size of the space for storing the symbol on the continuous stack. W_a is the weight matrix for the action mapping. The infinite memory required by the model can be seen from the stack-reading map

$$R_t = G_r(A_1, A_2, ..., A_t, I_1, I_2, ..., I_t). \tag{3}$$

Hence, the R value depends on the entire history of stack operations and past inputs. NNPDAs have been shown to learn to recognize strings in a number of small CFLs (Sun et al., 1998).

Zeng et al. [1994] present a DRN that uses discretization in its feedback links. The discretization automatically performs clustering (as understood in a completely analog network) to enforce stability. Furthermore the external stack is completely discrete. Using pseudo-gradient learning the network acquires stable representations for small CFLs from example sequences.

The use of an infinite stack obviates the need for unbounded precision to encode recursive structure. It is even sometimes possible to identify a finite (and stable) number of states (in the network) and combine these with the stack to extract a symbolic PDA from a trained NNPDA (Sun et al., 1998; Zeng et al., 1994).

3.2 DRNs without an external stack

In the abovementioned studies, the DRN itself is only required to play the same role within the overall system that is played by the DFA component of a PDA, LBA or Turing machine. It seems natural to ask: could a DRN learn to perform the task of the *whole* system, rather than just the DFA part of it? Because it has been proven that a DRN cannot robustly process non-regular languages, is it also the case that the *learning* and *inductive bias* of a DRN must be characteristic of regular languages?

To put it another way, any attempt by a DRN to process a non-regular language must ultimately be *non-robust* in the sense that noise injected into the system will cause it to make mistakes when it tries to process deeply embedded structures. But how important is robustness? Linguists have argued persuasively that the languages invented or induced by humans are inherently non-regular, notwithstanding the fact that humans also make mistakes while processing these languages, particularly when deeply nested structures are involved. That is, humans have a *competence* for non-regular languages, even though their *performance* on those languages is less than perfect (**?**)[1]. Can the same argument be made for DRNs? What evidence might we consider to test the claim that DRNs are learning non-regular languages? First, we might ask whether a DRN trained on sequences from a CFL is able to generalize to other sequences from the same CFL (e.g., more deeply nested structures). Second, it would be telling if a DRN trained on sequences from a regular language showed some bias towards inducing a non-regular language.

Over the past ten years, evidence has been accumulating for both of these aspects of DRN learning. There are now several studies showing that Simple Recurrent Networks (SRNs; (Elman, 1990)) trained to predict the next word in sentence processing tasks, are able to generalize levels of center embedding [Elman, 1991 — see also Christiansen and Chater, 1999 and Elman, 1993]. The Recursive Auto-Associative Memory model (RAAM) can also learn tree structures for simple context-free languages, with some systematic generalization to unseen cases (Pollack, 1987; Pollack, 1990; Blank et al., 1992; Kwasny and Kalman, 1995).

In addressing the second point, it is important to consider the type of evidence that would be accepted as showing that a DRN had induced either a regular or a non-regular language. As mentioned in section 8.1, once a DRN has been trained,

[1] This idea really goes back to the *lange/parole* distinction of (Saussure, 1915).

it is possible to extract from it a DFA which approximates the behavior of the DRN (see Chapters 7 and 12). The approximation is based on functional equivalence of regions in state space. In some cases, the DRN induces a regular language which is exactly modeled by the DFA. In others, the DRN induces a non-regular language, which cannot be completely modeled by any DFA. An explanation for this seeming contradiction is found by analyzing the networks at a higher resolution, allowing them to be viewed as infinite state systems (Kolen, 1994; Pollack, 1991; Crutchfield and Young, 1991). Extraction methods then produce a series of non-deterministic finite-state machines, modeling the DRN's behavior at successively more refined levels of detail. Indeed, studies have shown that neural networks sometimes prefer to induce a non-regular language, even when presented with data which could be described by a simple DFA (Blair and Pollack, 1997).

The approximation of a DFA by a DRN can be visualized in terms of discrete regions of the DRN state space corresponding to the states of the DFA. When a DRN computes a non-regular language, such visualization into regions of equivalent functionality is no longer appropriate. In the next subsection, we show how the behavior of a DRN trained on a simple CFL can be understood in terms of dynamical systems theory, and show how such analysis allows visualization of its performance.

3.3 Representing counters in recurrent hidden units

In this section, we show how simple counting tasks, such as predicting the next symbol from the language $a^n b^n$, can be performed by a DRN without an external stack. The simulations reviewed in this section can be seen as a possible mechanism by which the studies referred to above achieve their generalization results.

The network is presented with a sequence of strings from the language $a^n b^n$ and must predict, at each time step, the next symbol in the sequence. For example, suppose part of the input sequence is $aaaaabbbbbaabbaaabbb\ldots$ $(a^5b^5a^2b^2a^3b^3)$. We do not expect the network to correctly predict the first b in the sub-sequence a^5b^5 (since the number of a's is arbitrary) but we do expect it to successfully predict the next four b's, and in particular to predict the following a (which is the initial symbol of the next sub-sequence). Wiles and Elman [1995] demonstrated that SRNs with two inputs, two fully recurrent hidden units and two outputs (using the logistic transfer function) can be trained to perform this prediction task by backpropagation through time (BPTT; (Williams and Zipser, 1988)).

The studies reviewed here use one-shot encodings for their inputs and outputs (where one unit is "on" and all the others are "off"). For $a^n b^n$ the a input is represented by $(1, 0)$, the b input by $(0, 1)$. The most frequently found solution uses the natural property of recurrent hidden units with self weights, to converge to an attractor or diverge to a limit cycle. By matching the rate of convergence in one recurrent hidden unit with the rate of divergence in another, a DRN can "count" up and down (Rodriguez et al., 1999).

To elaborate on how the network accomplishes this task we introduce some concepts from dynamical systems theory. If we ignore (for the moment) some complications introduced by non-linear output functions, a DRN can be seen as a discrete-time *dynamical system* F. The activation vector $X \in \mathcal{R}^n$ of the n *state variables* (activation of n recurrent hidden units) changes over time as follows:

$$X_{t+1} = F(X_t + I_t), \tag{4}$$

where I_t is the externally provided input pattern at time t. In our case there are two (non-linear) automorphisms F_a and F_b (corresponding to inputs a and b). Since these automorphisms can be applied in any order (determined by the input string) it is useful to think of them as comprising an Iterated Function System [Barnsley, 1993 – see also Chapter 5].

A *fixed point* of F is a state $\tilde{X} \in \mathcal{R}^n$ for which $\tilde{X} = F\tilde{X}$. The standard technique of linearization provides a convenient way of determining the characteristic properties of a non-linear system in the neighborhood of a fixed point — as either attracting or repelling, with differing periodicity, in different directions, etc. The Jacobian (partial derivative matrix) of the (non-linear) state transformation allows us to analyze the system around fixed points as if it were linear. As an example, the Jacobian matrix for two state variables (x, y) is

$$J(x_t, y_t) = \begin{bmatrix} \frac{\partial x_{t+1}}{\partial x_t} & \frac{\partial x_{t+1}}{\partial y_t} \\ \frac{\partial y_{t+1}}{\partial x_t} & \frac{\partial y_{t+1}}{\partial y_t} \end{bmatrix}. \tag{5}$$

An *eigenvalue* of a linear map F is a scalar λ and an *eigenvector* of F is a vector $\mathbf{v}$ such that $F\mathbf{v} = \lambda\mathbf{v}$. The eigenvalue expresses the rate of contraction or expansion of F along the *principal dimension* given by the corresponding eigenvector. When studied close to the fixed points of the system, the Jacobian has eigenvalues and eigenvectors which approximate the way the non-linear system changes over time (Rodriguez et al., 1999).

The non-linearities also pose a problem for determining the location of the fixed points. By first estimating the locations of fixed points (by iteration) it is then possible to use standard techniques for estimating the roots of the system of equations (Rodriguez et al., 1999). Consider, for example, a 2-dimensional fully recurrent state space using the logistic transfer function $f(x) = 1/(1 + exp^{-x})$:

$$x_{t+1} = f(x_t w_{xx} + y_t w_{xy} + w_{x\theta}) \tag{6}$$
$$y_{t+1} = f(x_t w_{yx} + y_t w_{yy} + w_{y\theta}) \tag{7}$$

where w_{xx} is the self weight of x, w_{yy} is the self weight of y, w_{xy} is the weight on the connection from y to x, w_{yx} is the weight on the connection from x to y, and $w_{x\theta}$ and $w_{y\theta}$ are the biases for x and y, respectively. The fixed point $(\tilde{x}, \tilde{y})$ (which is not changing over time) can thus be determined by combining

$$\tilde{y} = -\frac{\tilde{x} w_{xx} + w_{x\theta} + \ln(-\frac{\tilde{x}-1}{\tilde{x}})}{w_{xy}} \tag{8}$$

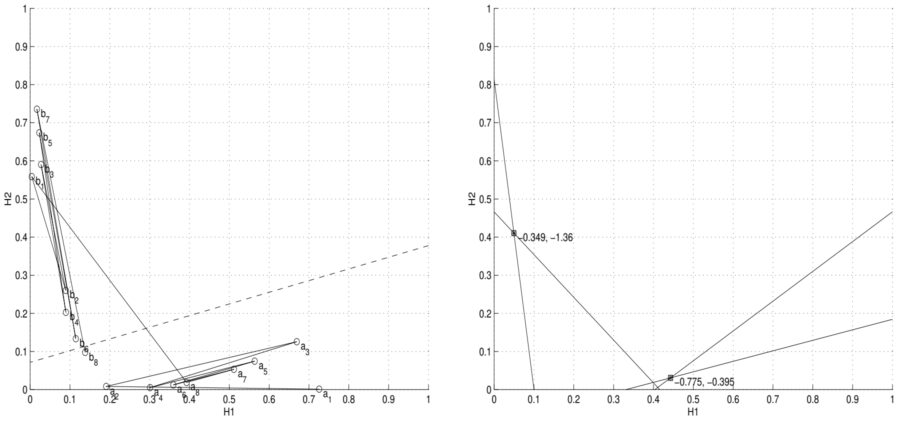
and

$$\tilde{x} = -\frac{\tilde{y} w_{yy} + w_{y\theta} + \ln(-\frac{\tilde{y}-1}{\tilde{y}})}{w_{yx}}. \tag{9}$$

The combination can be fine-tuned, for example, by Newton's method [cf. Tino et al., 1995].

The trajectory of activations in the recurrent hidden unit space of a trained network, as it processes the sequence a^8b^8, is shown in Figure 1(a) . As the a's are presented, the state converges towards an attracting fixed point at $(0.44, 0.03)$. On presentation of the first b, the state shifts to the neighborhood of a repelling fixed point at $(0.05, 0.41)$. Successive b's then take the state steadily away from the repeller, in such a way that the final b pushes it over the boundary from the "predict b" region to the "predict a" region (implemented by the network's output weights). Figure 1(b) shows the linearizations (around their fixed points) of the transformations defined by the F_a map and the F_b map. Note that in Figure 1(b) the largest (absolute) eigenvalues of these linearizations are approximately reciprocal $(-0.775 \times -1.36 \approx 1.0)$. Basically this means that the rate at which F_a converges towards its (attracting) fixed point corresponds to the rate at which F_b diverges from its (repelling) fixed point.

The negative sign of the eigenvalue indicates that the activations oscillate around each of the fixed points. Positive eigenvalues correspond to a monotonic solution, for



(a) State trajectory. (b) Eigenvectors.

Figure 1: A trajectory in the state space of a first order SRN with two recurrent hidden units, processing a^8b^8. (a) Each successive a in the sequence a_1 to a_8 causes an oscillation towards the F_a fixed point at (.44,.03). On presentation of the first b, the state jumps to a point governed by F_b, and then with each successive b diverges from the F_b fixed point at (.05,.41), oscillating outwards. The rate of divergence of the state under F_b matches the rate of convergence under F_a, so that the last b in the sequence (in this case b_8) crosses the hyperplane defined by the output unit (shown as a dotted line). The DRN then predicts correctly that the next input will be an a. (b) The eigenvectors corresponding to F_a and F_b in (a).

which the activations always remain on the same side of the fixed point. Networks sometimes converge on a monotonic solution, but these networks typically do not generalize as well as the alternating, oscillatory solutions. If a fixed point has an absolute eigenvalue below 1 for one of its eigenvectors, it is attracting along the principal dimension given by the associated eigenvector. Similarly, if the absolute eigenvalue is above 1, the fixed point repels.

The potential of the architecture and oscillatory dynamics (but using a linear state transformation) has been investigated using hand-crafted (analytically derived) networks (Holldobler et al., 1997). Using such networks, it has been shown that finely tuned SRNs can correctly predict sequences of arbitrary length from the a^nb^n language, given arbitrary precision.

We take such generalization results, combined with the dynamical systems analyses that show matching converging and diverging trajectories, as evidence that the network's competence should be properly characterized as learning a non-regular language.

3.4 Learning, stability and generalization for a^nb^n

The previous section outlined a typical solution for representation of a CFL by a DRN. In this section, we continue our analysis of DRN computation by reviewing the learning and stability of DRNs trained on sequences from CFLs.

SRNs have been successfully trained on the a^nb^n prediction task by BPTT using an incremental learning approach (Wiles and Elman, 1995). Even though they were only trained on strings of length $n = 1$ to $n = 11$, many of them successfully generalized to correctly predict strings of length $n = 12$ and $n = 13$. One network was observed to generalize to $n = 18$. Had the network been constructing specific states in a DFA for each string in the training data of length n, no states would have been constructed for strings longer than those in the training set, and hence, the generalization observed would not have been possible.

However, the task of the network in learning to march the converging and diverging oscillating trajectories is not an easy one for BPTT. It has been found that SRNs encounter instabilities as they learn this task by BPTT, repeatedly losing the solution, then finding it again (Tonkes et al., 1998; Bodén et al., 1999). The reason for the problem lies in the shape of the search space.

For the most frequently found solution to $a^n b^n$ in an SRN with two recurrent units, the presentation of a particular input largely results in oscillation along a single dimension. For the other input condition the other dimension is similarly used. Basically this means that at any time one recurrent unit is oscillating, while the other is passive (Bodén et al., 1999).

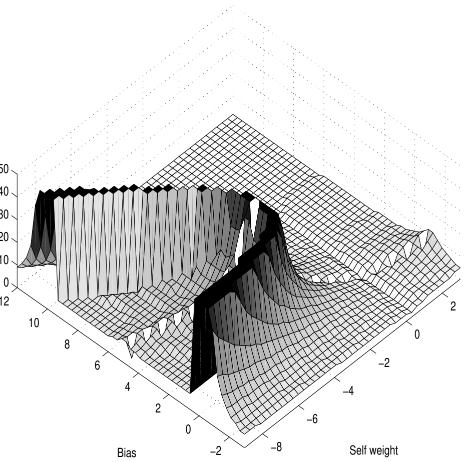
If we consider a single recurrent hidden unit (with a bias) it is possible to identify combinations of the self weight and the bias which lead to oscillating behavior. As can be seen in Figure 2(a) — which illustrates the number of oscillations that can be fitted in before a fixed point or limit cycle is reached — the self weight and bias need to be precisely coordinated. If the weights (self weight and bias) are positioned on the outside of the ridge in Figure 2(a), convergent oscillations (as found for F_a) appear. The weight configurations on the inside give rise to divergent behavior (as found for F_b). Thus the ridge corresponds to a bifurcation boundary between weights that allow a fixed point to be attracting and those that make the same fixed point repelling. The bifurcation occurs when the eigenvalue (of the Jacobian) is exactly -1 (for which the system is non hyperbolic). An additional complication is that to achieve a sufficient number of oscillations, weights must be located close to the boundary (one recurrent unit on the inside, one on the outside). When weights cross this boundary, the system's behavior around the fixed point changes radically, from attracting to repelling or vice versa (Bodén et al., 1999). If the weights are initialized to small values (close to the origin in Figure 2(a)) learning must take the system (weights of one recurrent hidden unit) through at least this particular bifurcation.

The instability of gradient learning can thus be explained by the proximity between the optimal weights and the bifurcation boundary. Figure 2(b) illustrates typical weight changes during a training session. Three phases can be discerned: (1) weights smoothly move closer to solution space, (2) the system's behavior varies radically when the weights are close to the bifurcation boundary, indirectly affecting weight changes which start fluctuating heavily, (3) a large weight change moves the system out of solution space. The error gradient tends to be complex in the vicinity of the bifurcation border; the magnitudes of the resulting gradients make gradient based learning inappropriate and unreliable (Bodén et al., 1999).

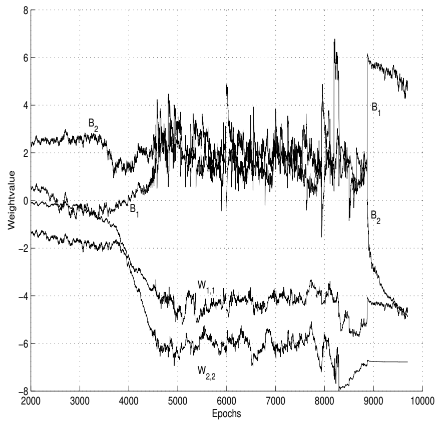
One way to combat these instabilities is to use an evolutionary hillclimbing algorithm, where the weights are only updated if a network with new weights is found to predict a series of test strings better than the previous network (Tonkes et al., 1998). This algorithm was able to learn the task stably, and also provided a stronger demonstration of generalization. During training, the network sometimes jumped several levels, generalizing to strings up to 14 symbols longer than it had previously encountered (n=42 to n=49).

3.5 Learning the context-sensitive language $a^n b^n c^n$

The previous subsection described how a first order network with two recurrent hidden units could learn to predict the CFL $a^n b^n$. We might expect that a second order network, or a network with more recurrent hidden units, might be able to predict more complex languages such as the context-sensitive language $a^n b^n c^n$. It has been shown that a first order SRN with three recurrent hidden units can learn



(a) Oscillation performance.



(b) Weight changes.

Figure 2: (a) The oscillation performance (number of oscillations fitted in before converging into a fixed point or diverging into a limit cycle) for a self recurrent unit with a bias. A maximum of 50 oscillations were plotted. (b) The instability of learning can be observed by looking at the weight changes of two connected self recurrent units during a typical run (the influence of inputs have been included in the biases). The error gradients when weights are in the vicinity of the bifurcation boundary are complex, making gradient based weight adaptation unstable and unreliable.

the $a^n b^n c^n$ prediction task (Chalup and Blair, 1999) using the hillclimbing algorithm referred to above. Interestingly, SRNs seem unable to learn the same task by BPTT — presumably due to the instabilities described in Section 3.4. However, sequential cascaded networks (SCNs; (Pollack, 1991)) with either two or three recurrent hidden units and with shortcut connections, have been able to learn this context-sensitive task by BPTT [Bodén and Wiles, in press]. The state trajectories of these networks, shown in Figures 3 and 4, display the same basic characteristics as the original SRN of (Chalup and Blair, 1999).

The way the task is accomplished by an SCN can be understood by comparison with the solutions for the $a^n b^n$ prediction task described in Section 3.3. The same basic description applies to the 3D case (Figure 3) and the 2D case (Figure 4). The network begins by counting up the number of a's as it converges to an attractor. Upon presentation of the first b, the activation shifts to a more central part of the hidden unit space, where it employs a two-pronged strategy of counting *down* by divergence from a repeller along one principal dimension (given by one eigenvector, as can be seen in Figure 4(b)) while simultaneously counting *up* by convergence to an attractor along another dimension (given by another eigenvector). The former ensures that the first c is predicted correctly, while the latter prepares for the c's to be counted down by divergence from a new repeller, ready to predict the a at the beginning of the next string.

The linearizations of the 2D case (around the fixed points) are shown in Figure 4(b). Interestingly the eigenvalues, collected over a large number of runs, demonstrate that the dynamics employed for $a^n b^n$ extend naturally to $a^n b^n c^n$ [Bodén and Wiles, in press]. One eigenvalue of the F_a fixed point is approximately reciprocal to one of the eigenvalues of the F_b fixed point. The second eigenvalue of the F_b fixed point is approximately reciprocal to one of the eigenvalues of the fixed point of F_c. Moreover, the eigenvectors were found to be aligned so as to enable appropriate

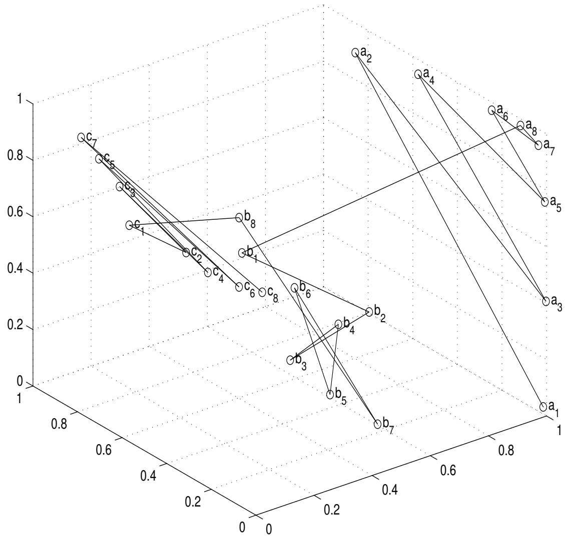
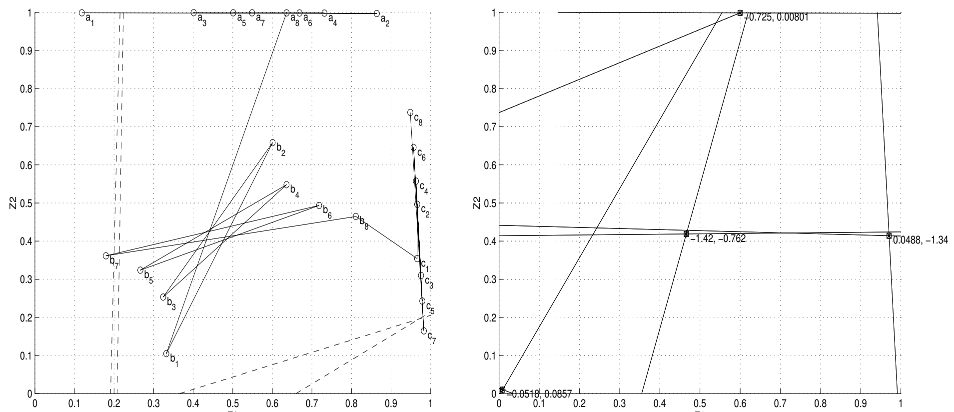


Figure 3: The trajectory in the state space of an SCN with three recurrent hidden units, processing $a^8 b^8 c^8$. The state converges (counts up) while processing the a's, then on processing the b's, it diverges (counts down) in one dimension and converges in another. Finally, in processing the c's, it diverges, oscillating outwards to a point where the final c predicts the first a of the next sequence (output hyperplane is not shown here).



(a) State trajectory.

(b) Linearization.

Figure 4: (a) The trajectory in the state space of an SCN with two recurrent hidden units processing $a^8 b^8 c^8$ and (b) its linearizations around the fixed points.

communication between the three autonomous systems. Hence, the inductive bias of DRNs, revealed through *training* of the networks, explores dynamics for predicting a CFL which extend naturally to at least one CSL — in sharp contrast with what is implied by the Chomsky hierarchy [Bodén and Wiles, in press].

Other context-sensitive language tasks have been implemented in hand-crafted DRNs, without addressing learning issues (Steijvers and Grünwald, 1996).

4 Generalization and inductive bias

Savitch (Savitch, 1989) argues that an *essentially infinite* language can be understood intuitively as a set of strings which have a simpler description as a subset of an infinite language, compared to their description as a finite set. For example, consider the set of strings containing an odd number of a's, with at most 10^{10} symbols. The DFA that would recognize just these strings and no others would be finite, but extremely large. By contrast, the (infinite) set of all strings with an odd number of a's can be described by a much simpler DFA with just two states.

Extending Savitch's idea, we introduce the term *essentially context-free* to describe data which have a simpler description as part of a context-free language, rather than as part of a regular language. Of course the notion of "simplicity" is not absolute, but depends on the framework within which the description is to be framed. Gold's theorem (Gold, 1967) says that CFLs cannot be learned from positive examples categorically; however, given structural assumptions, learning a PDA can be reduced to learning a DFA.

Studies of learning enable us to expose the *inductive bias* of a particular class of machines. We have considered such evidence in this chapter. Two frameworks may be equivalent with respect to what they can represent, and yet differ with respect to learning and generalization. With finite precision, any trained DRN may be analyzed as a DFA, albeit with a huge number of states. The network that learned $a^n b^n$ to $n = 49$ could be described using 100 states in a DFA description. Alternatively, it could be represented as a dynamical system emulation of a CFL, using just four eigenvectors in which the convergence rate into an attractor is matched by the divergence rate from a corresponding repeller.

The DFA analysis does not account for the fact that the language induced by the DRN may be described much more simply as a finite approximation to a CFL and generalizes in a way which is best described in terms of context-free languages rather than regular ones.

Symbolic and dynamical systems may naturally induce different languages from the same set of training strings. The "natural" class of languages induced by DRNs may even be some completely new class which does not correspond to any particular rung of the Chomsky hierarchy (see Chapter 15).

5 Conclusion

This chapter examined the abilities of DRNs to go beyond the representational capacities of finite-state automata. In particular, it described how networks with real valued transfer functions in their hidden units can represent and learn context-free languages by using the transient dynamics around the fixed points in the recurrent hidden unit activation space.

Analyzing DRNs as dynamical systems in terms of attractors gives a very different perspective than analyzing them in terms of finite regions. The simulations reviewed in this chapter show that one cannot deduce generalization properties in learning by considering only the classification of the architecture in terms of the

Chomsky hierarchy. We have explained the dynamical mechanisms whereby DRNs generalize CFLs and CSLs, even though they can only be robustly equivalent to DFAs. Along with many other researchers, we consider that precision provides a better resource constraint in classifying DRNs in a computational hierarchy than the memory constraints which are conventionally used for discrete symbolic systems.

References

Barnsley, M. (1993). *Fractals Everywhere.* Academic Press, Boston, 2nd edition.

Blair, A. and Pollack, J. B. (1997). Analysis of dynamical recognizers. *Neural Computation*, 9(5):1127–1142.

Blank, D. S., Meeden, L. A., and Marshall, J. B. (1992). Exploring the symbolic/subsymbolic continuum: A case study of RAAM. In Dinsmore, J., editor, *The Symbolic and Connectionist Paradigms: Closing the Gap*, pages 113–148. Lawrence Erlbaum, Hillsdale, NJ, USA.

Bodén, M. and Wiles, J. Context-free and context-sensitive dynamics in recurrent neural networks. *Connection Science.* Submitted.

Bodén, M., Wiles, J., Tonkes, B., and Blair, A. (1999). Learning to predict a context-free language: Analysis of dynamics in recurrent hidden units. In *Proceedings of ICANN'99.*

Casey, M. (1995). *Computation in discrete-time dynamical systems.* PhD thesis, Department of Mathematics, University of California at San Diego, La Jolla, CA.

Chalup, S. and Blair, A. (1999). Hill climbing in recurrent neural networks for learning the $a^n b^n c^n$ language. In *Proceedings of the Sixth International Conference on Neural Information Processing.*

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124.

Chomsky, N. (1959). On certain formal properties of grammars. 2(2):137–167.

Christiansen, M. and Chater, N. (1999). Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, 23:157–205.

Crutchfield, J. and Young, K. (1991). Computation at the onset of chaos. In Zurek, W., editor, *Proceedings of the 1988 Workshop on Complexity, Entropy and the Physics of Information*, pages 223–269, Redwood City, CA. Addison-Wesley.

Das, S., Giles, C. L., and Sun, G.-Z. (1993). Using prior knowledge in a NNPDA to learn context-free languages. In *Advances in Neural Information Processing Systems 5*, pages 65–72. Morgan Kaufmann.

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*, 48:71–99.

Giles, C., Sun, G., Chen, H., Lee, Y., and Chen, D. (1990). Higher order recurrent networks & grammatical inference. In Touretzky, D., editor, *Advances in Neural Information Processing Systems 2*, pages 380–387, San Mateo, CA. Morgan Kaufmann Publishers.

Gold, E. (1967). Language identification in the limit. *Information and Control*, 16:447–475.

Holldobler, S., Kalinke, Y., and Lehmann, H. (1997). Designing a counter: Another case study of dynamics and activation landscapes in recurrent networks. In *Proceedings of KI-97: Advances in Artificial Intelligence*, pages 313–324. Springer Verlag.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation.* Addison–Wesley, Reading, MA.

Kolen, J. (1994). Fool's gold: Extracting finite state machines from recurrent network dynamics. In Cowan, J., Tesauro, G., and Alspector, J., editors, *Advanves in Neural Information Processing Systems 6*, pages 501–508, San Francisco, CA. Morgan Kaufmann.

Kwasny, S. C. and Kalman, B. L. (1995). Tail-recursive distributed representations and simple recurrent networks. *Connection Science*, 7(1):61–80.

Maass, W. and Orponen, P. (1998). On the effect of analog noise in discrete-time analog computations. *Neural Computation*, 10(5):1071–1095.

Moore, C. (1998). Dynamical recognizers: Real-time language recognition by analog computers. *Theoretical Computer Science*, 201:99–136.

Pollack, J. (1987). *On Connectionist Models of Natural Language Processing.* PhD thesis, Computer Science Department, University of Illinois, Urbana, IL.

Pollack, J. (1991). The induction of dynamical recognizers. *Machine Learning*, 7:227–252.

Pollack, J. B. (1990). Recursive autoassociative memories. *Artificial Intelligence*, 46(1):77–105. raam.

Rodriguez, P., Wiles, J., and Elman, J. (1999). A recurrent neural network that learns to count. *Connection Science*, 11(1):5–40.

Saussure, F. d. (1974/1915). *Course in General Linguistics.* Fontana/Collins, London.

Savitch, W. J. (1989). Infinity is in the eye of the beholder.

Steijvers, M. and Grünwald, P. D. G. (1996). A recurrent network that performs a context–sensitive prediction task.

Sun, G., Giles, C., Chen, H., and Lee, Y. (1998). The neural network pushdown automaton: Architecture, dynamics and training. In Giles, C. and Gori, M., editors, *Adaptive Processing of Sequences and Data Structures: Lecture Notes in Artificial Intelligence*, pages 296–345. Springer Verlag, New York, NY.

Tino, P., Horne, B., and Giles, C. (1995). Fixed points in two–neuron discrete time recurrent networks: Stability and bifurcation considerations. Technical Report UMIACS-TR-95-51 and CS-TR-3461, Institute for Advance Computer Studies, University of Maryland, College Park, MD 20742.

Tonkes, B., Blair, A., and Wiles, J. (1998). Inductive bias in context-free language learning. In *Proceedings of the Ninth Australian Conference on Neural Networks (ACNN98)*, pages 52–56.

Wiles, J. and Elman, J. (1995). Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent networks. In *Proceedings of the 17th Annual Conference of the Cognitive Science Society*. MIT Press.

Williams, R. and Zipser, D. (1988). A learning algorithm for continually running fully recurrent neural networks. Technical Report ICS Report 8805, Institute for Cognitive Science, University of California at San Diego, La Jolla, CA.

Zeng, Z., Goodman, R. M., and Smyth, P. (1994). Discrete recurrent neural networks for grammatical inference. *IEEE Transactions on Neural Networks*, 5(2):320–330.