Discovering Faithful 'Wickelfeature' Representations in a Connectionist Network

Michael C. Mozer

CU-CS-463-90 March 1990

Michael C. Mozer
Department of Computer Science
and Institute of Cognitive Science
University of Colorado
Boulder, CO 80309-0430

e-mail: mozer@boulder.colorado.edu

Thanks to Jeff Elman and Yoshiro Miyata for their insightful comments and assistance. The graphical displays of network states are due to Miyata's SunNet simulator. Dave Rumelhart and Jay McClelland were kind enough to provide me with the phonological encoding and classification of verbs from their simulation work. This research was supported by Contracts N00014-85-K-0450 NR 667-548 and N00014-85-K-0076 with the Office of Naval Research, a grant from the System Development Foundation, and a Junior Faculty Development Award from the University of Colorado.

Abstract

A challenging problem for connectionist models is the representation of varying-length sequences, e.g., the sequence of phonemes that compose a word. One representation that has been proposed involves encoding each sequence element with respect to its local context; this is known as a *Wickelfeature* representation. Handcrafted Wickelfeature representations suffer from a number of limitations, as pointed out by Pinker and Prince (1988). However, these limitations can be avoided if the representation is constructed with a priori knowledge of the set of possible sequences. This paper proposes a specialized connectionist network architecture and learning algorithm for the discovery of *faithful* Wickelfeature representations — ones that do not lose critical information about the sequence to be encoded. The architecture is applied to a simplified version of Rumelhart and McClelland's (1986) verb past-tense model.

A challenging problem for connectionist models is the manipulation and representation of varying-length sequences. Consider the problem of representing a sequence of symbols, say letters. Using symbolic, LISP-like structures, this is straighforward: A short string like **ARM** can be represented as **(A R M)**, and a longer string like **FIREARM** can be represented by concatenating extra symbols onto the list — **(F I R E A R M)**. However, using connectionist activity patterns to represent these strings is a more complex matter. The activity pattern must indicate not only what the symbols are, but their positions in the string. This suggests the straightforward idea of reserving a processing unit for for every possible symbol in every position, but this scheme requires knowing the maximum length of the sequence in advance. It also suffers from the serious difficulty that two sequences containing common subsequences may appear quite different. For example, using the notation *x/n* to refer to a unit that is activated by the symbol *x* in position *n*, activity patterns corresponding to **{ A/1, R/2, M/3 }** and **{ F/1, I/2, R/3, E/4, A/5, R/6, M/7 }** have no overlap. The common subsequence **ARM** is not represented by an overlap in the activity patterns because **ARM** appears in a different position in each string. Overlap between similar activity patterns is critical in connectionist representations because it determines how a connectionist network will generalize to novel instances: if a network responds a certain way to **ARM**, one might like it to respond similarly to **FIREARM**, yet the position-specific letter encoding will not facilitate this.

Any representation of sequences should satisfy four criteria.

- The representation must be *faithful* (Smolensky, 1987), meaning that a one-to-one mapping exists between sequences and activity patterns. This requirement may be relaxed somewhat in the context of a particular task: The representation need only be sufficient to perform the desired input-output mapping. If two sequences have exactly the same consequences in all situations, there is no need to encode them distinctly. Task-irrelevant features do not have to be captured in the representation.
- The representation must be capable of encoding sequences of varying lengths with a fixed number of units.
- The representation must be capable of encoding relationships between elements of a sequence.
- The representation should provide a natural basis for generalization. It is on this ground that the position-specific encoding fails.

Wickelgren (1969) has suggested a representational scheme that seems to satisfy these criteria and has been applied successfully in several connectionist models (Mozer, 1990; Rumelhart & McClelland, 1986; Seidenberg, 1990). The basic idea is to encode each element of a sequence with respect to its local context. For example, consider the phonetic encoding of a word. Wickelgren proposed context-sensitive phoneme units, each responding to a particular phoneme in the context of a particular predecessor and successor. I will call these units *Wickelphones*, after the terminology of Rumelhart and McClelland. If the word *explain* had the phonetic spelling /eksplAn/, it would be composed of the Wickelphones $_{-}e_{k}$, $_{e}k_{s}$, $_{k}s_{p}$, $_{s}p_{l}$, $_{p}l_{A}$, $_{l}A_{n}$, and $_{A}n_{-}$ (where the dash indicates a word boundary). Assuming one Wickelphone unit for every such triple, activation of a word would correspond to a distributed pattern of activity over the Wickelphone units. With a fixed number of Wickelphone units, it is possible to represent arbitrary strings of varying length. Generally, this representation is faithful. In such cases, the unordered set of Wickelphones is sufficient to allow for the unambiguous reconstruction of the ordered string.

Rumelhart and McClelland devised a more compact and distributed encoding of phoneme sequences that depended on *features* of the phonemes rather than the phonemes themselves. Units in this *Wickelfeature* representation encode triples of phonemic features (such a "voiced" or "dental").

Smolensky (1987) provides a formalism that allows the Wickelphone and Wickelfeature encodings to be viewed in a uniform representational framework, as tensor products of feature vectors. In the remainder of this paper, I use the term "Wickelfeature" to denote a context-sensitive encoding of features of sequence elements or of the elements themselves, thereby subsuming the term "Wickelphone" and allowing the representation to be applied to arbitrary sequences.

Pinker and Prince (1988; Prince & Pinker, 1988) point to several serious limitations of handcrafted Wickelfeature representations, in particular the representation used by Rumelhart and McClelland in their model of learning past tenses of English verbs. One critical limitation is that if the class of sequences to be represented contain repeated subsequences of length two or more, the resulting representation is ambiguous. For example, the set of Wickelfeatures { $_{-}\mathrm{A}_{\mathrm{B}}$, $_{\mathrm{A}}\mathrm{B}_{\mathrm{A}}$, $_{\mathrm{B}}\mathrm{A}_{\mathrm{B}}$, $_{\mathrm{A}}\mathrm{B}_{-}$ } could correspond to the sequence **ABAB** or to **ABABAB** or an infinite number of other such strings. Similarly, the set { $_{-}\mathrm{A}_{\mathrm{B}}$, $_{\mathrm{A}}\mathrm{B}_{\mathrm{X}}$, $_{\mathrm{B}}\mathrm{X}_{\mathrm{A}}$, $_{\mathrm{X}}\mathrm{A}_{\mathrm{B}}$, $_{\mathrm{A}}\mathrm{B}_{\mathrm{Y}}$, $_{\mathrm{B}}\mathrm{Y}_{\mathrm{A}}$, $_{\mathrm{Y}}\mathrm{A}_{\mathrm{B}}$, $_{\mathrm{A}}\mathrm{B}_{-}$ } could correspond either to sequence **ABXABYAB** or **ABYABXAB**. Thus, the Wickelfeature representation can lose order information.

There are potential ways around these problems. One quick solution is to represent more contextual information in the Wickelfeatures. If a Wickelfeature consists of v sequence elements rather than just three, confusions arise only if the strings contain repeated subsequences of length $v-1$ or greater. As v grows, however, the representation becomes more and more localist and loses the advantages that we set out to attain. Another solution is to have the Wickelfeature units be activated in a graded fashion, not all-or-nothing. This would allow a unit to signal the number of instances of that Wickelfeature in a sequence, which handles the **ABAB** problem. Alternatively, the amount of activity could correspond to the position in a sequence; in the **ABXABYAB** example, the $_{\mathrm{A}}\mathrm{B}_{\mathrm{X}}$ unit could be less active than the $_{\mathrm{A}}\mathrm{B}_{\mathrm{Y}}$ unit, indicating its primacy in the sequence.

Hand coding Wickelfeature representations of this sort gets quite tricky. In this paper, I report on an alternative approach using connectionist learning algorithms to discover Wickelfeature-like representations. The advantage of leaving the job to learning is that whatever representations the system develops, they are assured of being sufficient for the domain at hand, i.e., they will satisfy the faithfulness criterion mentioned above. A further advantage of using learning is that by discovering only domain-relevant Wickelfeatures, the overall representation can be more compact. For instance, a system whose task is to encode English letter strings as Wickelfeatures will not develop a $_{\mathrm{P}}\mathrm{K}_{\mathrm{T}}$ unit.

A network architecture to learn Wickelfeatures

The approach I have taken involves training a network to map input sequences to target output patterns through a layer of units that learn to respond as Wickelfeatures. It does not much matter what the output patterns are; they could be localist representations of the sequences, responses to the sequence, or perhaps sequences themselves. Figure 1 shows a schematic drawing of an architecture that performs this mapping. The input layer represents a small window on the sequence. At any time, the input layer views several consecutive elements of the sequence — three elements in the Figure. Presenting a complete sequence to the network involves sliding the sequence through the window. More concretely, time is quantized into discrete *steps*, and at each time step, the sequence is advanced by one position in the input window. Once the entire sequence has been presented, the output units should respond appropriately. The output layer is activated by the context layer, the purpose of which is to remember those elements of the input sequence that are critical for performing the input-output mapping. At each time step, units in the context layer integrate their current values with the new input to form a new context representation.

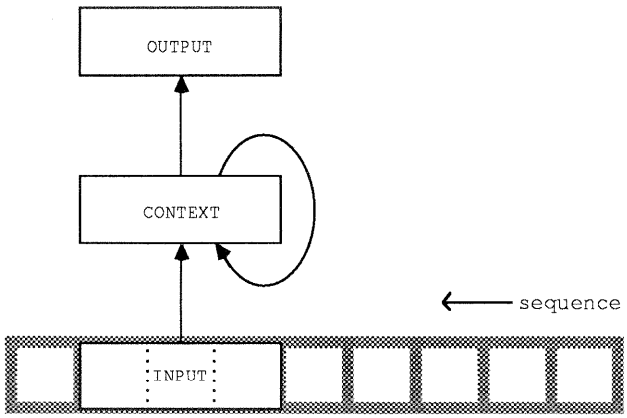

Figure 1. A three-layered recurrent network consisting of input, context, and output units. Each labeled box indicates a set of processing units. The arrows indicate complete connectivity from one layer to another.

The context layer thus forms a *static* internal representation of the dynamic input sequence. The goal of learning, of course, is for this to become a Wickelfeature representation. The use of a sliding input window does part of the job: At each time, the context units can only see a local "chunk" of the sequence. This allows the context units to detect local conjunctions of sequence elements, or conjunctions of features of sequence elements. Once activated by a pattern in the input, the context units should remain on. Thus, it seems sensible to have self-connected context units, but not to connect each context unit to each other, using an activation function like:

$$c_i(t+1) = d_i c_i(t) + s[net_i(t)] ,$$

where $c_i(t)$ is the activity level of context unit i at time t, d_i is a decay weight associated with the unit, s is a sigmoid squashing function, and $net_i(t)$ is the net input to the unit:

$$net_i(t) = \sum_j w_{ji} x_j(t) ,$$

$x_j(t)$ being the activity of input unit j at time t, w_{ji} the connection strength from input unit j to context unit i. Thus, a context unit adds its current activity, weighted by the decay factor, to the new input at each time. The decay factor allows old information to fade over time if d_i is less than one.

To summarize, the Wickelfeature-learning architecture differs from a generic recurrent architecture for temporal sequence recognition in three respects: (1) the input layer consists of a small temporal window holding several elements of the input sequence; (2) connectivity in the context layer is restricted to one-to-one recurrent connections; and (3) integration over time in the context layer is linear.

A training algorithm for the Wickelfeature architecture

The standard procedure for training a recurrent network with temporally-varying inputs using the back-propagation algorithm is to "unfold" the network in time (Rumelhart, Hinton, & Williams, 1986), transforming the recurrent network into a feedforward network. The unfolding procedure requires that each unit remember a temporal history of its activation values and is computation intensive. For the Wickelfeature architecture, however, the unfolding procedure can be avoided, as described by Mozer (1989). To summarize the result, consider a sequence with s elements and an architecture with a v-element window. The number of time steps, t, required to slide the sequence through the window is simply $s-v+1$. Consequently, at time t the network receives a target vector over the output units and an

error E can be computed. Using the ordinary back propagation procedure, the error derivative with respect to each context unit i,

$$\delta_i(t) \equiv \frac{\partial E}{\partial c_i(t)}$$

can be computed. The weight update rule for the recurrent connections, d_i, is then:

$$\Delta d_i = -\varepsilon \delta_i(t) \alpha_i(t) ,$$

where $\alpha_i(t)$ is defined by the recurrence relation

$$\alpha_i(\tau) = c_i(\tau-1) + d_i \alpha_i(\tau-1)$$

with boundary value $\alpha_i(0) = 0$. Similarly, the weight update rule for the input-context connections, w_{ji}, is:

$$\Delta w_{ji} = -\varepsilon \delta_i(t) \beta_{ji}(t) ,$$

where $\beta_{ji}(0) = 0$ and

$$\beta_{ji}(\tau) = s'[net_i(\tau)] x_j(\tau) + d_i \beta_{ji}(\tau-1) .$$

Thus, explicit back propagation in time is not necessary. Bachrach (1988), Gori, Bengio, and De Mori (1989), and Williams and Zipser (1989) have independently discovered the idea of computing an activity trace during the forward pass as an alternative to back propagation in time. However, this is the first use of the architecture for the purpose of learning Wickelfeature-like representations.

Simulation results

Implementation details

In the simulations to be reported, an additional parameter z_i — called the *zero point*, was added to the context-unit activation function, for reasons described by Mozer (1989). The complete activation function is:

$$c_i(t+1) = d_i c_i(t) + s[net_i(t)] + z_i ,$$

where the value of z_i is determined by gradient descent as for the other parameters.

The initial input-context and context-output connection strengths were randomly picked from a zero-mean gaussian distribution and normalized such that the L1 norm of the fan-in (incoming) weight vector was 2.0. The z_i were initially set to -0.5, and the d_i picked from a uniform distribution over the interval .99-1.01. The weights were updated only after a complete presentation of the training set (an *epoch*). Momentum was not used. Learning rates were adjusted dynamically for each set of connections according to a heuristic described by Mozer (1989).

Learning Wickelfeatures

Starting with a simple example, the network was trained to identify four sequences: **_DEAR_**, **_DEAN_**, **_BEAR_**, and **_BEAN_**. Each symbol corresponds to a single sequence element and was represented by a random binary activity pattern over three units. The input layer was a two-element buffer through which the sequence was passed. For **_DEAR_**, the input on successive time steps consisted of **_D**, **DE**, **EA**, **AR**, **R_**. The input layer had six units, the context layer two, and the output layer four. The network's task was to associate each sequence with a corresponding output unit. To perform this

task, the network must learn to discriminate **D** from **B** in the first letter position and **N** from **R** in the fourth letter position. This can be achieved if the context units learn to behave as Wickelfeature detectors. For example, a context unit that responds to the Wickelfeatures **_D** or **DE** serves as a **B-D** discriminator; a unit that responds to **R_** or **AR** serves as an **N-R** discriminator. Thus, a solution can be obtained with two context units.

Fifty replications of the simulation were run with different initial weights. The task was learned in a median of 488 training epochs, the criterion for a correct response being that the output unit with the largest value was the appropriate one. Figure 2 shows the result of one run. The weights are grouped by connection type, with the input-context connections in the upper-left array, followed by the decay connections (d_i), zero points (z_i), and context-output connections. Each connection is depicted as a square whose *area* indicates the relative weight magnitude, and shading the weight sign — black is positive, white is negative. The sizes of the squares are normalized within each array such that the largest square has sides whose length is equal to that of the vertical bars on the right edge of the array. The absolute magnitude of the largest weight is indicated by the number in the upper-right corner. Because normalization is performed within each array, weight magnitudes of different connection types must be compared with reference to the normalization factors. The units within each layer are numbered. The weights feeding into and out of context unit 1 have been arranged along a single row, and the weights of context unit 2 in the row above. Bias terms (i.e., weight lines with a fixed input of 1.0) are also shown for the context and output units.

For the activity levels in the lower half of the figure, there are four columns of values, one for each sequence. The input pattern itself is shown in the lowest array. Time is represented along the vertical dimension, with the first time step at the bottom and each succeeding one above the previous. The input at each time reflects the buffer contents. Because the buffer holds two sequence elements, note that the second element in the buffer at one time step (the activity pattern in input units 4-6) is the same as the first element of the buffer at the next (input units 1-3). Above the input pattern are, respectively, the context unit activity levels after presentation of the final sequence element, the output unit activity levels at this time, and the target output values. The activity level of a unit is proportional to the area of its corresponding square. If a unit has an activity level of 0, its square has no area — an empty space. The squares are normalized such that a "unit square" — a square whose edge is the length of one of the vertical bars — corresponds to an activity level of 1. While the input, output, and target activity levels range from 0 to 1, the context activity levels can lie outside these bounds, and are, in fact, occasionally greater than 1.

With these preliminaries out of the way, consider what the network has learned. At the completion of each sequence, the context unit activity pattern is essentially binary. Context unit 1 is off for **_BEAN_** and **_BEAR_**, and on for **_DEAN_** and **_DEAR_**; thus, it discriminates **B** and **D**. Context unit 2 is off for **_BEAN_** and **_DEAN_**, and on for **_BEAR_** and **_DEAR_**; thus it discriminates **N** and **R**. However, the context units do not behave in a straightforward way as Wickelfeatures. If context unit 1 were sharply tuned to, say, **_D**, the input-context weights should serve as a matched filter to the input pattern **_D**. This is not the case: the weights have signs -+--+- but the **_D** input pattern is 110011. Nor is context unit 1 tuned to the **DE**, whose input pattern is 011010. Instead, the unit appears to be tuned equally to both patterns. By examining the activity of the unit over time, it can be determined that the unit is activated partly by **_D** and partly by **DE** but by no other input pattern. This makes sense: **_D** and **DE** are equally valid cues to the sequence identity, and as such, evidence from each should contribute to the response. To get a feel for why the detector responds as it does, note that **_D** (110011) is distinguished from **_B** (110001) by activity in unit 5; **DE** (011010) from **BE** (001010) by activity in unit 2. The weights from inputs 2 and 5 to context unit 1 are positive, allowing the unit to detect **D** in either context. The other

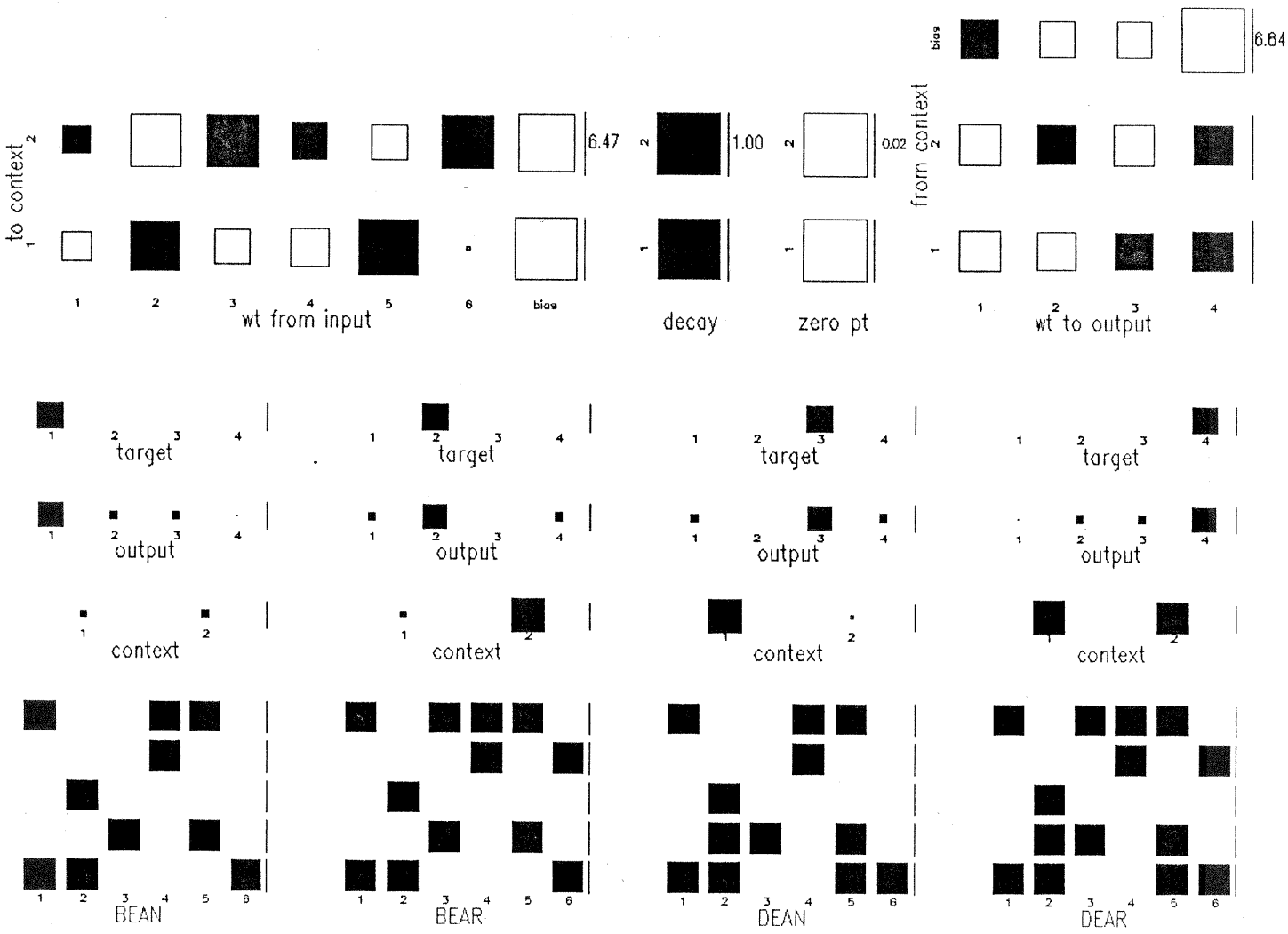

Figure 2. The **DEAR/DEAN/BEAR/BEAN** problem. The upper half of the figure shows learned weights in the network, the lower half activity levels in response to each of the four input sequences.

weights are set so as to prevent the unit from responding to other possible inputs. Thus, the unit selects out key features of the Wickelfeatures **_D** and **DE** that are not found in other Wickelfeatures. As such, it behaves as a **_DE** Wickelfeature detector, and context unit 2 similarly as a **AR_** detector.

Generalization testing supports the notion that the context units have become sensitive to these Wickelfeatures. If the input elements are permuted to produce sequences like **AR_BE**, which preserves the Wickelfeatures **AR_** and **_BE**, context unit responses are similar to those of the original sequences. However, with permutations like **_RB_**, **_DAER_**, and **DEAR** (without the end delimiters), which destroy the Wickelfeatures **AR_** and **_BE**, context unit responses are not contingent upon the **D**, **B**, **N**, and **R**. Thus, the context units are responding to these key letters, but in a context-dependent manner.

Learning the regularities of verb past tense

In English, the past tense of many verbs is formed according to a simple rule. Regular verbs can be divided into three classes, depending on whether the past tense is formed by adding /ˆd/ (an "ud" sound), /t/, or /d/. Examples of the classes are /dEpend/ (*depend*), /fAs/ (*face*), and /dEscrIb/ (*describe*), respectively. Each string denotes the phonetic encoding of the verb in italics, and each symbol a single phoneme. The phoneme notation and the examples have been borrowed from Rumelhart and McClelland (1986). The rule for determining the class of a regular verb is as follows.

> If the final phoneme is dental (/d/ or /t/), add /ˆd/;
> else if the final phoneme is an unvoiced consonant, add /t/;
> else (the final phoneme is voiced), add /d/.

A network was trained to classify the twenty examples of each class. Each phoneme was encoded by a set of four trinary acoustic features (see Rumelhart & McClelland, 1986, Table 5). The input layer of the network was a two-element buffer, so a verb like /kamp/ appeared in the buffer over time as **_k**, **ka**, **am**, **mp**, **p_**. The underscore is a delimiter symbol placed at the beginning and end of each string. The network had eight input units (two time slices each consisting of four features), two context units, and three output units — one for each verb class.

In fifteen replications of the simulation, the network performed at 90% within 100 epochs, learned the training set perfectly in under 1000 epochs. A verb was considered to have been categorized correctly if the most active output unit specified the verb's class. The network has learned the underlying rule, as evidenced by perfect generalization to novel verbs. Typical weights learned by the network are presented in Figure 3, along with the output levels of the two context units in response to twenty verbs. These verbs, though not part of the training set, were all classified correctly.

The response of the context units is straightforward. Context unit 1 has a positive activity level if the final phoneme is a dental (/d/ or /t/), negative otherwise. Context unit 2 has positive activity if the final phoneme is unvoiced, near zero otherwise. These are precisely the features required to discriminate among the three regular verb classes. In fact, the classification rule for regular verbs can be observed in the context-output weights (the rightmost weight matrix in Figure 3). Connections are such that output unit 1, which represents the "add /ˆd/" class, is activated by a final dental phoneme; output unit 2, which represents the "add /t/" class, is activated by a final non-dental unvoiced phoneme; and output unit 3, which represents "add /d/" class, is activated by a final non-dental voiced phoneme.

Note that the decay weights in this simulation are small in magnitude; the largest is .02. Consequently, context units retain no history of past events, which is quite sensible because only the final phoneme determines the verb class. This fact makes verb classification a simple task: it is not necessary for the context units to hold on to information over time. Simulations were also conducted giving the

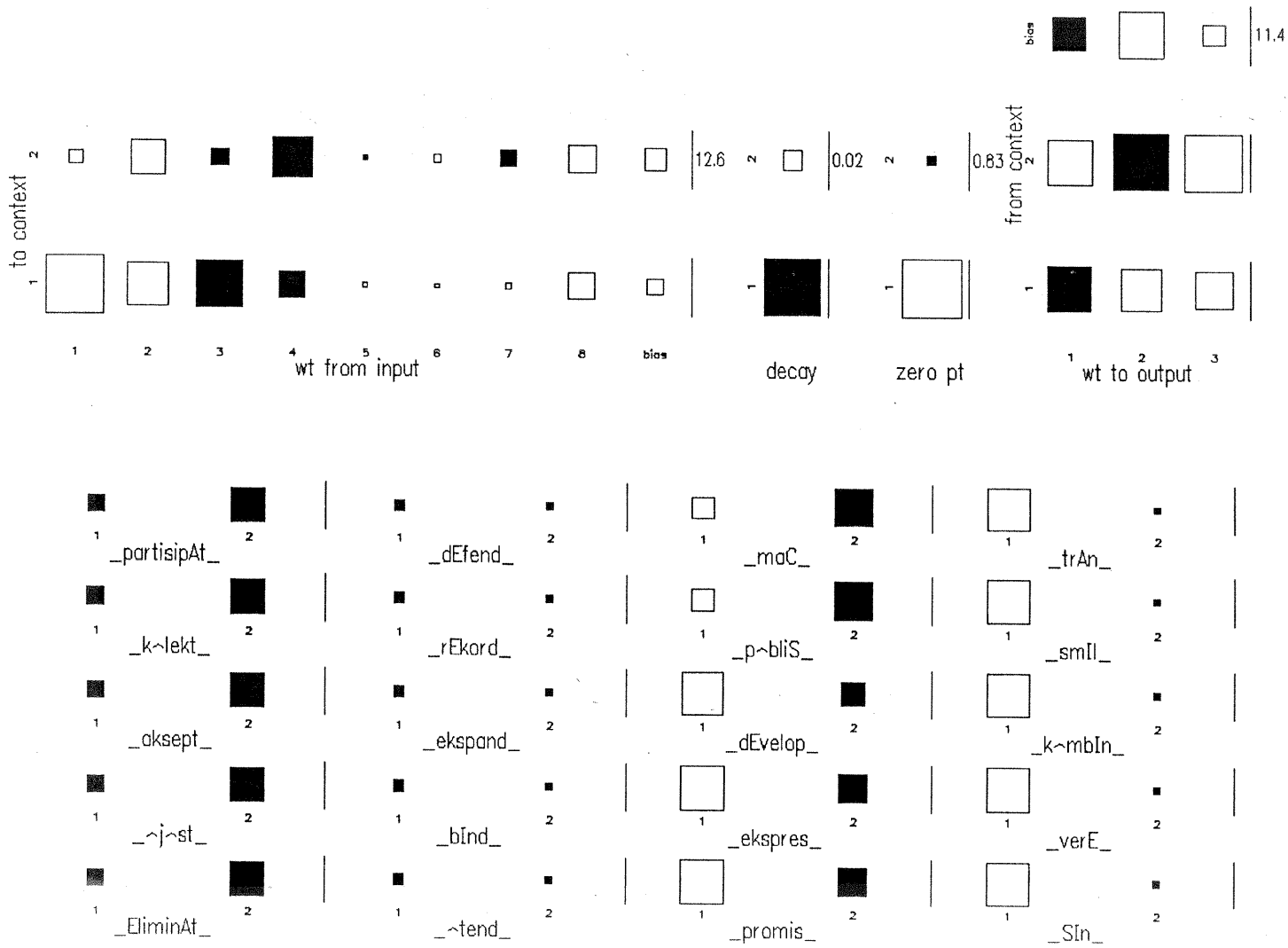

Figure 3. The regular verb problem. The upper half shows learned weights in the network, the lower half shows the final activity levels of the context units in response to a variety of verbs. Verbs in the first column all end with /t/, in the second column with /d/, in the third column with an unvoiced consonant, and the fourth column with a voiced consonant or vowel.

network the same verb classification task, but reversing the order of the phonemes; instead of /dEpend/, /dnepEd/ was presented. In this problem, the relevant information comes at the *start* of the sequence and must be retained until the sequence is completed. Nonetheless, the network is able to learn the task. Interestingly, a more standard network architecture was unsuccessful at learning the task.

Large verb simulation

To study a more difficult task, the regular-verb categorization problem was extended to a larger corpus of verbs. As before, the task was to classify each verb according to the manner in which its past tense is formed. The complexity of the task was increased by including both regular and irregular verbs, 136 training instances altogether, and a total of thirteen response categories — three for regular forms and ten for irregular (see Mozer, 1989, for examples of these categories). The categories are based loosely on a set suggested by Bybee and Slobin (1982).

The corpus of verbs was borrowed from the Rumelhart and McClelland (1986) model. The model, designed to account for children's acquisition of verb past tenses, produces the past tense of a verb given its infinitive form as input. The representation used at both input and output ends is a handcrafted Wickelfeature encoding of the verb, built into the model. The purpose of this simulation is to demonstrate that a network, given a sequence of phonemes, can *learn* a representation like that presupposed by Rumelhart and McClelland's model.

The task is difficult. The verb classes contain some internal regularities, but these regularities are too weak to be used to uniquely classify a verb. For instance, all verbs in category 3 end in a /d/ or /t/, but so do verbs in categories 4, 5, and 11. Whether a verb ending in /d/ or /t/ belongs in category 3 or one of the other categories depends on whether it is regular, but there are no simple features signaling this fact. Further, fine discriminations are necessary because two outwardly similar verbs can be classified into different categories. *Swim* and *sing* belong to category 10, but *swing* to category 12; *ring* belongs to category 10, but *bring* to category 8; *set* belongs to category 4, but *get* to category 11. Because the category to which a verb belongs is somewhat arbitrary, the network must memorize a large number of special cases.

The network architecture was similar to that used in the regular verb example. The input layer was a two-phoneme buffer, and the encoding of phonemes was the same as before. The output layer consisted of thirteen units, one for each verb class, and the context layer contained 25 units.

In ten replications of the simulation, the network learned to select the correct category in about 500 epochs. At intermediate stages of learning, verbs are sometimes "overregularized", as when the past tense of *eat* was considered to be *eated*. Overgeneralization occurs in other respects, as when *sit* was misclassified in the category of verbs whose past tense is the same as the root — presumably by analogy to *hit* and *fit* and *set*. Interpretation of the behavior of individual context units is difficult, but by examining similar input sequences that are classified differently, e.g., /riN/ and /briN/, one can pinpoint context units responsible for certain behaviors.

These simulations demonstrate the feasibility of constructing faithful Wickelfeature-like representations using connectionist learning procedures, instead of having to craft the representations by hand. Further, the simulations show that intrinsically temporal or sequential input can be dealt with as such, instead of as static patterns. This is a necessary first step in the modeling of language and speech processes.

References

Bachrach, J. (1988). *Learning to represent state.* Unpublished master's thesis, University of Massachusetts, Amherst.

Bybee, J. L., & Slobin, D. I. (1982). Rules and schemas in the development and use of the English past tense. *Language, 58,* 265-289.

Gori, M., Bengio, Y., & Mori, R. de (1989). BPS: A learning algorithm for capturing the dynamic nature of speech. In *Proceedings of the First International Joint Conference on Neural Networks, Volume 2* (pp. 417-423).

Mozer, M. C. (1989). A focused back-propagation algorithm for temporal pattern recognition. *Complex Systems, 3.*

Mozer, M. C. (1990). In *The perception of multiple objects: A connectionist approach.* Cambridge, MA: MIT Press/Bradford Books.

Pinker, S., & Prince, A. (1988). On language and connectionism. *Cognition, 28,* 73-193.

Prince, A., & Pinker, S. (1988). Wickelphone ambiguity. *Cognition, 30,* 189-190.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Volume I: Foundations* (pp. 318-362). Cambridge, MA: MIT Press/Bradford Books.

Rumelhart, D. E., & McClelland, J. L. (1986). On learning the past tenses of English verbs. In J. L. McClelland & D. E. Rumelhart (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Volume II: Psychological and biological models* (pp. 216-271). Cambridge, MA: MIT Press/Bradford Books.

Seidenberg, M. S. (1990). Word recognition and naming: A computational model and its implications. In W. D. Marslen-Wilson (Ed.), *Lexical representation and process.* Cambridge, MA: MIT Press.

Wickelgren, W. (1969). Context-sensitive coding, associative memory, and serial order in (speech) behavior. *Psychological Review, 76,* 1-15.

Williams, R. J., & Zipser, D. (1989). Experimental analysis of the real-time recurrent learning algorithm. *Connection Science, 1,* 87-111.