Word Boundary Learning in Simple Recurrent Networks

Andrew B. Kope

University of Western Ontario

Abstract

In the present study I analyzed the effects of quantity of training (number of training epochs) and of network architecture (number of hidden units) on the ability of a simple recurrent network to distinguish word boundaries in a letter string. I hypothesized that there would be a main effect of both quantity of training and network architecture. The results indicate that although each model is able to detect word boundaries and is sensitive to the effects of context, there is no effect of either quantity of training or the number of hidden units on these properties. Furthermore, the results demonstrate that four models varying on both dimensions fail to generalize to novel words, and that there is no effect of either quantity of training or network architecture on this failure. The implications of these results for the use of simple recurrent network models to detect word boundaries are discussed.

Keywords: simple recurrent network, back-propagation, word boundary learning

Word Boundary Learning in Simple Recurrent Networks

Artificial neural networks are used both in psychological research and in industry to model the behaviour of neural systems and associated psychological processes. Neural networks attempt to abstract away the complexity of a real biological system, and thereby focus on what is theoretically most important from an information processing point of view for a given task or process. In industry, artificial neural networks are used to read handwriting, find icebergs in nautical radar signals, and to predict tides. In psychological research, neural network models that can accurately mimic human performance (and human error patterns) are used to support hypotheses that a network abstraction has captured something important about the modeled process (Arbib, 1998). There are several different types of neural networks, for example back propagation (e.g. Elman, 1990), constraint satisfaction (e.g. Baolong, Lei, & Guanzhong, 2000), and self-organizing networks (e.g. Fukushima, 1980), with each type of network suited to different types of tasks.

Elman et al. (1996) provide a general overview of connectionist networks. Connectionist network models use an interconnected network of simple units, or nodes, to produce complex emergent processes. These nodes are akin to simple artificial neurons, in that they collect input from a variety of sources and provide output to others. Connectionist network nodes can be divided into three main categories: input nodes, which receive input from the 'outside' (in the form of patterns passed in to the network by a control program); hidden nodes, which send and receive input to and from other nodes; and output nodes, which send activation 'outside' of the network (in the form of output passed out to a control program). Figure 1 illustrates several possible connectionist network architectures.

Activation in these networks spreads along connections between nodes (Elman et al.,

1996). Depending on the type of network, connections can be unidirectional (as in Figure 1b) such that activation spreads in only one direction through the network, or bidirectional (as in Figure 1a) such that activation spreads in both directions through the network. The connections between nodes are weighted, usually with small continuous real values (e.g. 1.2354, -0.6458) such that the activation transferred from the output of one node to the input of another is scaled according to the weight of the connection between the two. In most networks, one node receives input from many other nodes; the input activation for that one node then becomes the sum of the weighted activations coming from all of its input connections. For most models this input activation does not translate directly to an output response, but is instead scaled according to a response function (e.g. a sigmoid curve, see Figure 2). The 'knowledge' that a network can be said to possess, therefore, is stored as the weights between nodes.

Despite this relatively simple architecture, Elman et al. (1996) explain several abilities and applications of connectionist models. Relatively simple feed-forward models, where activation travels unidirectionally forward through a model employ Hebbian learning rules to learn pair-wise correlations (correlations between pairs of stimuli). Hebb (1949) suggests that when an axon of a cell A is near enough to excite a cell B repeatedly or persistently, some growth process will occur in one or both cells, such that A's efficiency as one of the cells firing B increases; this behaviour is replicated by feed-forward models. Back-propagation network models allow for an error (or teacher) signal at the level of the output to influence changes in weights between nodes of the network. To do this, the amount of error present at the output units (i.e. the difference between the observed activation and the target activation) is used to adjust the weights leading into those output units in whatever way necessary to decrease this error. In this way, a back-propagation network does not simply learn connections between input patterns, but

learns to generate a desired output based on feedback from an error signal.

Although these conceptually simple networks are able to learn relatively complex patterns between input and output stimuli, they lack a means of representing patterns that change systematically over time. In order to capture the influence that prior context has on the present processing of a given input pattern, Elman (1990) created the simple recurrent network model (Figure 3). In this model architecture, the activation of the hidden units at a given time step is copied back to a context layer on a one-to-one basis. During training at the next time step, the hidden units must then map both the activation from the input layer resulting from the presentation of a new pattern, and the activation from the context layer representing its previous internal state. Because of the addition of activation from the context layer, to achieve the desired output the hidden units must develop representations that are encoded with the temporal properties of a sequential input. By including the context surrounding one input vector within a sequential input, a simple recurrent network can solve problems and detect patterns that simple feed-forward and simple back-propagation networks cannot.

For example, Elman (1990) explains the use of a simple recurrent network to 'discover' the notion of a word. Citing debate among linguists and psycholinguists regarding the precise definition of the concept of a 'word,' Elman explains that the precise form of the concept remains an open question, and suggests that the notion of a word could emerge as a consequence of learning the sequential structure of the letter sequences constituent of words and sentences. To demonstrate the feasibility of this suggestion, Elman trained a simple recurrent network on a lexicon of fifteen words, with the aim of having the network predict the next letter in a sequential input string of words from the lexicon. He found that at the onset of a new word in the pattern, the error in predicting that letter is high. As more of the word is received however, the error in

predicting the next letter declines, because the sequence is increasingly more predictable. This change in error predicting the next letter of a sequence indicates what the recurring sequences in the input are, and these recurring sequences correlate highly with words. Elman was careful to indicate that this information is not categorical, however, and reflects only graded statistical co-occurrence of letters. Therefore, based on these letter-to-letter changes in error predicting the next letter in the sequence, it is possible to determine (approximately) what subsections of the sequence constitute words. This determination of word boundaries is liable to error, however, due to common letter sequences that incorporate more than one word (e.g. *ey* in th*ey* and th*e* *y*ear).

In addition to word boundaries, Elman (1995) describes the ability of an attractor network to capture the semantic or categorical features of words from distributional facts given by their position within a sentence. To demonstrate this ability, Elman trained a network on a lexicon of 29 words, arranged into 10 000 sentences, with each word presented to the network as a localist vector. The task of the model was to predict the upcoming word in the input sequence. At the completion of training, Elman tested the network by comparing its prediction of the next word against the corpus. Because the corpus was nondeterministic (in that any number of words could follow one given word), Elman suggested that in short of memorizing the entire sequence, the network would be unable to make exact predictions; instead, the network predicted a cohort of potential word successors in each context. The activation of each of these cohorts, however, was highly correlated with the conditional probability of each word in that context. Elman explains, "this behaviour suggests that in order to maximize performance at prediction, the network identifies inputs as belonging to classes of words based on distributional properties and co-occurrence information" (Elman, 1995, p. 205). To test this possibility, Elman averaged the

hidden unit activation for each word across all instances in all contexts. Taking these mean activation vectors as prototypes, when clustered hierarchically they revealed that the network had acquired categorical knowledge of several word types (e.g. animate nouns, intransitive direct-object verbs; see Figure 4). Elman (1995) was careful to point out, however, that the network did not 'know' anything about the actual semantic content of these categories. Instead, this structure was created in the network because it was the best method for accounting for the distributional properties of the words in the training set.

With substantial evidence demonstrating the capacity of simple recurrent networks to learn the distributional and categorical patterns present within a set of data, I aimed to investigate the architectural and methodological factors affecting the ability of a model to capture these patterns. In the present study I analyzed the effects of quantity of training (number of training epochs) and of number of hidden units on the ability of a simple recurrent network to distinguish word boundaries in a letter string. I hypothesized that there would be a main effects of both quantity of training and the number of hidden units, as demonstrated in a model's ability to detect boundaries between words, in the influence of context on the prediction of the next letter in a sequence, and in a model's ability to generalize to novel words not part of the training lexicon.

Network Architectures under Investigation

In his original investigation of the use of simple recurrent networks to determine word boundaries, Elman (1990) used a network with 5 input units, 5 output units, 20 hidden units, and 20 context units. With a lexicon of 15 different words, Elman used a sentence generation program to create 200 sentences between 4 and 9 words in length. He then concatenated these sentences to form a stream of 1270 words, and 4963 characters. Each letter in this string of 4963

letters was converted into a five-bit input vector, resulting in a serial input stream of 4963 five-bit input vectors. Elman then presented this stream of vectors to his model one at a time, with the model tasked to predict the next input letter at each time step.

To investigate the effects of quantity of training and number of hidden units on the ability of a simple recurrent network to distinguish word boundaries in a letter string, I created two network models based on Elman's 1990 model described above.

My first model consisted of 26 input units, 26 output units, 26 hidden units, and 26 context units, and one bias node. Interactions between layers was as described in Elman et al. (1996), with the addition of a bias node which always had an activation of 1.0 and provided output to each node of both the hidden layer and the output layer (the weights of these connections changed as usual during training). More specifically, the 26 input units were fully interconnected with the 26 hidden layer units, the 26 hidden layer units were fully interconnected with the 26 output layer units, and the hidden layer was connected to the context layer to allow for a copy-back of the hidden layer activation to the context layer.

My second model was identical to the first, except that in place of 26 hidden units and 26 context units, it instead had 52 hidden units and 52 context units.

Training

Software

To train the models used in this study, I used version 2.05 of the PDPTool extension for MatLab. It is hosted by the University of Stanford, and was designed for the implementation of the models described in Rumelhart and McClelland (1986).

Pattern

To create the serial training pattern I began with a lexicon of 13 different words (see Appendix A), from which I created 200 different randomized word strings, each of which used the entire lexicon. I then concatenated those word strings to create one master string of 2600 words, 10 600 characters long. PDPTool accepts input in the form of a string of characters separated by spaces, converting each character into a localized 26-bit input vector, so I therefore did not convert the master string into smaller pattern vectors before training.

The distribution of words in the training pattern was directly proportional to the thirteen word lexicon. The distribution of letters in the training pattern is presented in Table 1. The statistical probabilities of one letter appearing given another letter are presented in Table 2. To assess the effect of quantity of training, I exposed both model architectures (26 hidden units and 52 hidden units) to both 10 and 50 epochs of training, which resulted in four distinct models.

Research Design

Models for Analysis

In fully crossing model architecture with quantity of training, I created a two-by-two array of network models for analysis: small hidden layer size and small quantity of training, with 26 hidden units and 10 training epochs; small hidden layer size and large quantity of training, with 26 hidden units and 50 training epochs; large hidden layer size and small quantity of training, with 52 hidden units and 10 training epochs; and large hidden layer size and large quantity of training, with 52 hidden units and 50 training epochs.

Testing

To test each of the four models under analysis, I presented each model with a testing string composed of five randomized word strings (each of the five strings contained the 13 words from the lexicon used to create the testing pattern), recording the sum squared error and the

output activation vector for the presentation of each letter of the testing string.

Word Boundary Detection

To assess the ability of a given model to detect word boundaries, I compared the sum squared error (SSE) of the model in predicting the first letter of a word and the SSE in predicting the fourth letter of a word. At the onset of a new word in the pattern the error in predicting that letter will be high, however as more of the word is received the error in predicting the next letter should decline until the first letter of another word is reached; therefore, a greater difference in SSE between the fist and fourth letter of a word is indicative of greater ability of a model to detect word boundaries. To make this comparison within one model, I took the average SSE of the model in predicting the first letter of three words (*pail, mountain, fetch*) across all five randomized word strings in the test pattern, and compared it to the average SSE of the model in predicting the fourth letter of those words (again averaged across the five word strings in the test pattern) using a *t*-test.

To compare word boundary detection performance between models, for each model I calculated the average difference in SSE in predicting the first and the fourth letter of three words (*pail, mountain, fetch*), and then compared between-models the average of those differences in SSE across the five word strings with an ANOVA.

Effect of Context on Letter Prediction

To assess the effect of context on one model's ability to predict the next letter following a given letter, I compared the output vectors following the presentation of the letter E in *fetch* and *went*. To make this comparison, I summed the absolute value of the difference in activation following the presentation of the letter E in *fetch* and *went* for each node of the output layer in both contexts. This sum reflects the magnitude of the effect of context on a model's letter

prediction, in that if a model does not use context in its predictions, the difference in node-by-node activation following the presentation of the same letter in two different contexts should be zero; the more a model uses context in its predictions, the larger this sum will be however, as there will be a systematic difference in the pattern of activation in the output layer due to the context surrounding the letter presented. Therefore, a measure of a given model's sensitivity to context in predicting the next letter of a sequence is the magnitude of the sum of these differences in activation (MSDA), with a greater magnitude indicating a greater sensitivity to context.

To judge each of the four models under investigation on its sensitivity to the effects of context, I compared the MSDA averaged across the five randomized word strings to zero using a *t*-test. To compare the four models on their sensitivity to the effects of context, I compared the averaged MSDA for the five randomized word strings used to create the testing pattern between-models using an ANOVA.

Word Boundary Detection with Novel Words

To assess the ability of a model to generalize to a novel word, I first created a new random word string that also included the novel words *watch* and *male*. I selected *watch* as a novel word because though novel, it still follows the same patterns in letter order as other words in the training lexicon. I selected *male* as a novel word because though wholly composed of letters from the training lexicon, it does not follow the same patterns in letter order as the other words of the training lexicon. I presented each model with a testing string composed of five randomized word strings (each of these five strings contained the 13 words from the lexicon used to create the testing pattern and the two novel words; see Appendix B), again recording the sum squared error and the output activation vector for the presentation of each letter of this new

testing string.

To assess the novel word boundary detection ability of one model I took the average SSE of the model in predicting the first letter of the two novel words (*watch, male*) across all five randomized word strings in the test pattern, and compared it to the average SSE of the model in predicting the fourth letter of those words (again averaged across the five word strings in the test pattern) using a *t*-test.

To compare novel word boundary detection performance between models, for each model I calculated the average difference in SSE in predicting the first and the fourth letter of the two novel words (*male, watch*), and then compared between-models the average of those differences in SSE across the five word strings with an ANOVA.

Results

Word Boundary Detection

Each of the four models under investigation demonstrated a significant difference in average SSE in predicting the first letter as compared to the fourth letter of *pail, mountain,* and *fetch* (averaged across all three words, and averaged across the five randomized word strings used to create the testing pattern). These results are presented in Table 3.

The ANOVA comparing between-models the difference in SSE for the prediction of the first and fourth letters of *pail, mountain, and fetch* (averaged across all three words, and averaged across the five randomized word strings used to create the testing pattern) was not significant. There was no main effect of network architecture, $F(1) = .15$, $p = .70$, no main effect of quantity of training $F(1) = .11$, $p = .74$, and no significant interaction between these two factors, $F(1) = .30$, $p = .59$. These mean differences in SSE of prediction for the four models are presented in Table 4.

In summary, the results indicate that each model can detect word boundaries, as reflected by differences in error between the first and fourth letters of a word. There is no significant effect, however, of either network architecture or quantity of training on the ability of a model to detect word boundaries.

Effect of Context on Letter Prediction

Each of the four models under investigation demonstrated a significant effect of context on their ability to predict the next letter following a given letter, as given by an MSDA value that is significantly different than zero. The MSDA for each model (averaged across the five randomized words strings used to create the testing pattern) and the results of these comparisons are presented in Table 5.

The between-models ANOVA comparing these mean of the MSDA values for each model (averaged across the five randomized words strings used to create the testing pattern, see Table 5) produced no significant results. There was no main effect of network architecture, $F(1) = .02$, $p = .91$, no main effect of quantity of training $F(1) = .81$, $p = .38$, and no significant interaction between the two factors, $F(1) = .00$, $p = .98$.

In summary, these results indicate that each model is sensitive to the effect of context on its ability to predict the next letter following a given letter, as reflected by MSDA values that are significantly different than zero. There is no significant effect, however, of either network architecture or quantity of training on a model's sensitivity to context effects.

Word Boundary Detection with Novel Words

Although each of the four models under investigation did demonstrate a significant difference in average SSE in predicting the first letter as compared to the fourth letter of *male* and *watch* (averaged across both words, and averaged across the five randomized word strings

used to create the testing pattern), the average SSE in prediction of the fourth letter was higher than the first letter for every model. These results are presented in Table 6.

The ANOVA comparing between-models the difference in SSE for the prediction of the first and fourth letters of *watch* and *male* (averaged across all three words, and averaged across the five randomized word strings used to create the novel testing pattern) was not significant. There was no main effect of network architecture, $F(1) = .085$, $p = .77$, no main effect of quantity of training $F(1) = 1.20$, $p = .28$, and no significant interaction between the two factors, $F(1) = 4.04$, $p = .05$. These mean differences in SSE of prediction for the four models are presented in Table 7.

In summary, the results indicate a significant difference between SSE in the first and fourth letters of novel words for each model, however the SSE is greater for the fourth letter of those words. This increase in SSE in from the first to the fourth letter of novel words suggests that the differences in activation reflect an inability of the models to predict the letters of a novel word. Furthermore, there is no significant effect of either network architecture or quantity of training on the ability of the models to detect word boundaries of novel words.

Discussion

I hypothesized that there would be a significant effect of both network architecture (i.e. 26 versus 52 hidden/context units) and quantity of training (i.e. 10 versus 50 epochs) on a simple recurrent network's ability to detect word boundaries, its sensitivity to a letter's context in making a prediction of the next letter in a pattern, and its ability to detect word boundaries in novel words.

The results indicated that each of the four models I investigated had a strong ability to detect the boundaries between words present in the lexicon used to create the training pattern;

contrary to my hypothesis however, this ability did not differ between different network architectures or between different quantities of training. Figures 5, 6, 7, and 8 display graphs of SSE over a sample of the first testing pattern (which included only words from the original thirteen word lexicon) that illustrate the jumps in SSE bounding words for each model under investigation.

Similarly, the results indicated that each of the four models I investigated were sensitive to the context in which a letter appears (that is, the letters in the pattern preceding that given letter) as reflected by their predictions for the next letter following that given letter. Contrary to my hypothesis however, this sensitivity did not differ between different network architectures or between different quantities of training.

The results also suggest none of the four models in my investigation were able to generalize to novel words not used in the training pattern. Additionally, contrary to my hypothesis, there was no effect of network architecture or quantity of training on the ability of a network to generalize to novel words. Figures 9, 10, 11, and 12 display graphs of SSE over a sample of the second testing pattern (which also included the novel words *watch* and *male*) for each model under investigation. If the models had been able to generalize to the two novel words, instead of increasing in SSE as more letters of the novel words are presented to the models, the SSE would have decreased (Elman, 1990). At present, the models demonstrate a jump in SSE characteristic of a word boundary at the beginning of a novel word; however, the SSE fails to decrease over the course of presentation of the novel words, indicating that the models cannot predict the next letter of the pattern for novel words. That is, the four models are insensitive to the word boundaries at the termination of novel words.

I would explain the inability of any model to generalize to words outside of the lexicon

used to compile the training pattern by citing the relatively small nature of that lexicon. Although the models were able to capture and demonstrate the distributional structure of the words in the lexicon quite capably, general trends in English words such as *–le* and *–tch* were not captured by the models. If a much larger lexicon were used to create the training pattern, that is, one that possesses such general trends in English words and therefore does not emphasize solely the sequences of letters of the words of the training lexicon, I would expect to see much better novel word boundary detection performance.

The results of the present study, therefore, have two major implications for the use of simple recurrent networks to find word boundaries. The results indicate that within the ranges used in my study, the number of hidden units and the depth of training do not have a significant impact on word boundary detection or on the sensitivity of a model to a letter's context; in essence, it appears all four of the present models are highly adept at word boundary detection and highly sensitive to a letter's context, and therefore there are no differences between models on these factors. As such, in creating models for the purpose of word boundary detection, it is advisable to use a smaller number of hidden units, and a smaller quantity of training, as there are no performance benefits to be accrued from employing more complicated network architectures or greater quantities of training.

Secondly, as explicated in my discussion of the models' ability to generalize to novel words, when the training lexicon is small and words are randomized multiple times to create the training pattern, the only systematic characteristic of the training pattern that will be reflected in the weights of the resultant model will be the sequence of letters corresponding to each of the words in the training lexicon. The models will not capture trends in short sequences of letters within words; only the wholesale structure words. As such, I posit that in order to create a

generalizable model capable of detecting word boundaries in novel words, a substantial lexicon of hundreds or perhaps thousands of words would need to be used in the creation of the training pattern. Otherwise, as Elman (1990) suggests, with a small lexicon and a large number of randomly-ordered presentations of that lexicon in the training pattern, a model will only reflect the graded statistical co-occurrence of letters in the sequences of the words in the lexicon, and not the general trends in English necessary to accommodate novel words.

References

Arbib, M. A. (Ed.). (1998). *The handbook of brain theory and neural networks*. MIT Press.

Baolong, G., Lei, G., & Guanzhong, D. (2000). Constraint satisfaction neural network. *Tien Tzu Hsueh Pao, 28*(1), 81-84.

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

Elman, J. L. (1995). Language as a dynamical system. In R. Port, & T. nan Gelder (Eds.), *Mind as motion* (pp. 195-225). MIT Press.

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

Hebb, D. O. (1949). *The organization of behavior*. New York: Wiley & Sons.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics, 36*(4), 193-202.

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition*. Cambridge, MA: MIT Press.

Appendix A: List of Words in Training Set

jackie
and
gilligan
went
up
the
mountain
to
fetch
a
pail
of
water

Appendix B: List of Words in Novel Testing Set

up
went
watch
pail
jackie
a
mountain
fetch
gilligan
of
male
and
water
to
the

Note. Novel words are in bold.

Table 1.

Distribution of Letters in the Training Pattern

Letter	Frequency	Letter	Frequency
A	1400	N	1000
B	0	O	600
C	400	P	400
D	200	Q	0
E	1000	R	200
F	400	S	0
G	400	T	1200
H	400	U	400
I	1000	V	0
J	200	Q	400
K	200	X	0
L	600	Y	0
M	200	Z	0

Table 2.

Probability of One Letter Given Another in the Training Pattern

Given Letter	Probability of Each Letter Following								
A	a: 0.008	b: 0.000	c: 0.143	d: 0.000	e: 0.000	f: 0.017	g: 0.012	h: 0.000	i: 0.286
	j: 0.011	k: 0.000	l: 0.000	m: 0.015	n: 0.286	o: 0.014	p: 0.011	q: 0.000	r: 0.000
	s: 0.000	t: 0.159	u: 0.014	v: 0.000	w: 0.024	x: 0.000	y: 0.000	z: 0.000	
C	a: 0.000	b: 0.000	c: 0.000	d: 0.000	e: 0.000	f: 0.000	g: 0.000	h: 0.500	i: 0.000
	j: 0.000	k: 0.500	l: 0.000	m: 0.000	n: 0.000	o: 0.000	p: 0.000	q: 0.000	r: 0.000
	s: 0.000	t: 0.000	u: 0.000	v: 0.000	w: 0.000	x: 0.000	y: 0.000	z: 0.000	
D	a: 0.090	b: 0.000	c: 0.000	d: 0.000	e: 0.000	f: 0.085	g: 0.055	h: 0.000	i: 0.000
	j: 0.095	k: 0.000	l: 0.000	m: 0.080	n: 0.000	o: 0.090	p: 0.090	q: 0.000	r: 0.000
	s: 0.000	t: 0.175	u: 0.080	v: 0.000	w: 0.160	x: 0.000	y: 0.000	z: 0.000	
E	a: 0.058	b: 0.000	c: 0.000	d: 0.000	e: 0.000	f: 0.044	g: 0.030	h: 0.000	i: 0.000
	j: 0.018	k: 0.000	l: 0.000	m: 0.048	n: 0.200	o: 0.024	p: 0.038	q: 0.000	r: 0.200
	s: 0.000	t: 0.263	u: 0.032	v: 0.000	w: 0.045	x: 0.000	y: 0.000	z: 0.000	
F	a: 0.100	b: 0.000	c: 0.000	d: 0.000	e: 0.500	f: 0.032	g: 0.032	h: 0.000	i: 0.000
	j: 0.055	k: 0.000	l: 0.000	m: 0.038	n: 0.000	o: 0.005	p: 0.045	q: 0.000	r: 0.000
	s: 0.000	t: 0.068	u: 0.048	v: 0.000	w: 0.078	x: 0.000	y: 0.000	z: 0.000	
G	a: 0.500	b: 0.000	c: 0.000	d: 0.000	e: 0.000	f: 0.000	g: 0.000	h: 0.000	i: 0.500
	j: 0.000	k: 0.000	l: 0.000	m: 0.000	n: 0.000	o: 0.000	p: 0.000	q: 0.000	r: 0.000
	s: 0.000	t: 0.000	u: 0.000	v: 0.000	w: 0.000	x: 0.000	y: 0.000	z: 0.000	
H	a: 0.090	b: 0.000	c: 0.000	d: 0.000	e: 0.500	f: 0.000	g: 0.040	h: 0.000	i: 0.000
	j: 0.042	k: 0.000	l: 0.000	m: 0.030	n: 0.000	o: 0.060	p: 0.028	q: 0.000	r: 0.000
	s: 0.000	t: 0.070	u: 0.058	v: 0.000	w: 0.082	x: 0.000	y: 0.000	z: 0.000	
I	a: 0.000	b: 0.000	c: 0.000	d: 0.000	e: 0.200	f: 0.000	g: 0.200	h: 0.000	i: 0.000
	j: 0.000	k: 0.000	l: 0.400	m: 0.000	n: 0.200	o: 0.000	p: 0.000	q: 0.000	r: 0.000
	s: 0.000	t: 0.000	u: 0.000	v: 0.000	w: 0.000	x: 0.000	y: 0.000	z: 0.000	
J	a: 1.000	b: 0.000	c: 0.000	d: 0.000	e: 0.000	f: 0.000	g: 0.000	h: 0.000	i: 0.000
	j: 0.000	k: 0.000	l: 0.000	m: 0.000	n: 0.000	o: 0.000	p: 0.000	q: 0.000	r: 0.000
	s: 0.000	t: 0.000	u: 0.000	v: 0.000	w: 0.000	x: 0.000	y: 0.000	z: 0.000	
K	a: 0.000	b: 0.000	c: 0.000	d: 0.000	e: 0.000	f: 0.000	g: 0.000	h: 0.000	i: 1.000
	j: 0.000	k: 0.000	l: 0.000	m: 0.000	n: 0.000	o: 0.000	p: 0.000	q: 0.000	r: 0.000
	s: 0.000	t: 0.000	u: 0.000	v: 0.000	w: 0.000	x: 0.000	y: 0.000	z: 0.000	
L	a: 0.058	b: 0.000	c: 0.000	d: 0.000	e: 0.000	f: 0.037	g: 0.027	h: 0.000	i: 0.333
	j: 0.022	k: 0.000	l: 0.333	m: 0.022	n: 0.000	o: 0.027	p: 0.005	q: 0.000	r: 0.000
	s: 0.000	t: 0.050	u: 0.037	v: 0.000	w: 0.050	x: 0.000	y: 0.000	z: 0.000	
M	a: 0.000	b: 0.000	c: 0.000	d: 0.000	e: 0.000	f: 0.000	g: 0.000	h: 0.000	i: 0.000
	j: 0.000	k: 0.000	l: 0.000	m: 0.000	n: 0.000	o: 1.000	p: 0.000	q: 0.000	r: 0.000
	s: 0.000	t: 0.000	u: 0.000	v: 0.000	w: 0.000	x: 0.000	y: 0.000	z: 0.000	
N	a: 0.062	b: 0.000	c: 0.000	d: 0.200	e: 0.000	f: 0.028	g: 0.020	h: 0.000	i: 0.000
	j: 0.025	k: 0.000	l: 0.000	m: 0.018	n: 0.000	o: 0.036	p: 0.035	q: 0.000	r: 0.000
	s: 0.000	t: 0.468	u: 0.023	v: 0.000	w: 0.085	x: 0.000	y: 0.000	z: 0.000	
O	a: 0.058	b: 0.000	c: 0.000	d: 0.000	e: 0.000	f: 0.365	g: 0.032	h: 0.000	i: 0.000

	j: 0.035	k: 0.000	l: 0.000	m: 0.012	n: 0.000	o: 0.035	p: 0.020	q: 0.000	r: 0.000
	s: 0.000	t: 0.027	u: 0.355	v: 0.000	w: 0.062	x: 0.000	y: 0.000	z: 0.000	
P	a: 0.600	b: 0.000	c: 0.000	d: 0.000	e: 0.000	f: 0.025	g: 0.050	h: 0.000	i: 0.000
	j: 0.030	k: 0.000	l: 0.000	m: 0.055	n: 0.000	o: 0.028	p: 0.028	q: 0.000	r: 0.000
	s: 0.000	t: 0.095	u: 0.002	v: 0.000	w: 0.088	x: 0.000	y: 0.000	z: 0.000	
R	a: 0.180	b: 0.000	c: 0.000	d: 0.000	e: 0.000	f: 0.055	g: 0.075	h: 0.000	i: 0.000
	j: 0.065	k: 0.000	l: 0.000	m: 0.065	n: 0.000	o: 0.055	p: 0.105	q: 0.000	r: 0.000
	s: 0.000	t: 0.200	u: 0.090	v: 0.000	w: 0.110	x: 0.000	y: 0.000	z: 0.000	
T	a: 0.191	b: 0.000	c: 0.167	d: 0.000	e: 0.167	f: 0.010	g: 0.019	h: 0.167	i: 0.000
	j: 0.020	k: 0.000	l: 0.000	m: 0.013	n: 0.000	o: 0.181	p: 0.014	q: 0.000	r: 0.000
	s: 0.000	t: 0.027	u: 0.012	v: 0.000	w: 0.013	x: 0.000	y: 0.000	z: 0.000	
U	a: 0.000	b: 0.000	c: 0.000	d: 0.000	e: 0.000	f: 0.000	g: 0.000	h: 0.000	i: 0.000
	j: 0.000	k: 0.000	l: 0.000	m: 0.000	n: 0.500	o: 0.000	p: 0.500	q: 0.000	r: 0.000
	s: 0.000	t: 0.000	u: 0.000	v: 0.000	w: 0.000	x: 0.000	y: 0.000	z: 0.000	
W	a: 0.500	b: 0.000	c: 0.000	d: 0.000	e: 0.500	f: 0.000	g: 0.000	h: 0.000	i: 0.000
	j: 0.000	k: 0.000	l: 0.000	m: 0.000	n: 0.000	o: 0.000	p: 0.000	q: 0.000	r: 0.000
	s: 0.000	t: 0.000	u: 0.000	v: 0.000	w: 0.000	x: 0.000	y: 0.000	z: 0.000	

Note. Given letters that do not appear in the training pattern are excluded from the table.

Table 3.

Sum Squared Error (SSE) in Predicting the First and Fourth Letters of Pail, Mountain, Fetch

Number of Hidden Units	Training Epochs	First Letter SSE	Fourth Letter SSE	*t*-test for Equality of Means
26	10	M = 1.0034 SD = .0719	M = .0004 SD = .0005	$t(28) = 54.00$, $p < .000$
	50	M = .9869 SD = .0513	M = .0001 SD = .0001	$t(28) = 74.44$, $p < .000$
52	10	M = .9863 SD = .0654	M = .0005 SD = .0011	$t(28) = 58.36$, $p < .000$
	50	M = .9898 SD = .0903	M = .0001 SD = .0002	$t(28) = 42.47$, $p < .000$

Note. The sum squared error in prediction was averaged across five randomized word strings and across the three words used for comparison. For all above *t*-tests Levene's test for equality of variances was significant, so equal variances were assumed.

Table 4.

Mean Differences in SSE of Prediction between the First and Fourth Letters of Pail, Mountain, and Fetch for Each Model

Number of Hidden Units	Training Epochs	*M*	*SD*
26	10	1.003	.072
	50	.989	.051
52	10	.956	.065
	50	.990	.090

Note. $n = 15$ for all above means. Averages are across all three words, and across the five randomized word strings used to create the testing pattern.

Table 5.

Mean Magnitude of the Sum of Differences in Activation for Each Model, As Compared to Zero

Number of Hidden Units	Training Epochs	*M* of MSDA	*SD* of MSDA	*t*-test for Difference from Zero
26	10	2.003	.085	$t(4) = 52.92$, $p < .000$
	50	2.024	.006	$t(4) = 718.57$, $p < .000$
52	10	2.007	.048	$t(4) = 93.00$, $p < .000$
	50	2.026	.022	$t(4) = 203.62$, $p < .000$

Note. $n = 5$ for all above means. The Mean Magnitude of the Sum of Differences in Activation (MSDA) was calculated as the sum of the absolute value of all differences in activation following the presentation of the letter E (in the word *fetch* and in the word *went*) for each node of the output layer in both contexts. The mean MSDA above is the average across the five randomized word strings that together composed the testing pattern.

Table 6.

Sum Squared Error (SSE) in Predicting the First and Fourth Letters of Male and Watch

Number of Hidden Units	Training Epochs	First Letter SSE	Fourth Letter SSE	*t*-test for Equality of Means
26	10	M = .9039 SD = .2596	M = 1.8387 SD = .2667	$t(18) = -7.94$, $p < .000$
	50	M = .9593 SD = .1811	M = 1.9984 SD = .0017	$t(18) = -18.14$, $p < .000$
52	10	M = .9542 SD = .2260	M = 2.0853 SD = .1756	$t(18) = -12.50$, $p < .000$
	50	M = .9510 SD = .2603	M = 1.7272 SD = .3774	$t(18) = -5.35$, $p < .000$

Note. The sum squared error in prediction was averaged across five randomized word strings and across the two novel words used for comparison. For all above *t*-tests Levene's test for equality of variances was significant, so equal variances were assumed.

Table 7.

Mean Differences in SSE of Prediction for Each Model between the First and Fourth Letters of Male and Watch for Each Model

Number of Hidden Units	Training Epochs	*M*	*SD*
26	10	-.935	.41
	50	-1.039	.18
52	10	-1.131	.29
	50	-.776	.48

Note. $n = 10$ for all above means. Averages are across both novel words, and across the five randomized word strings used to create the testing pattern.

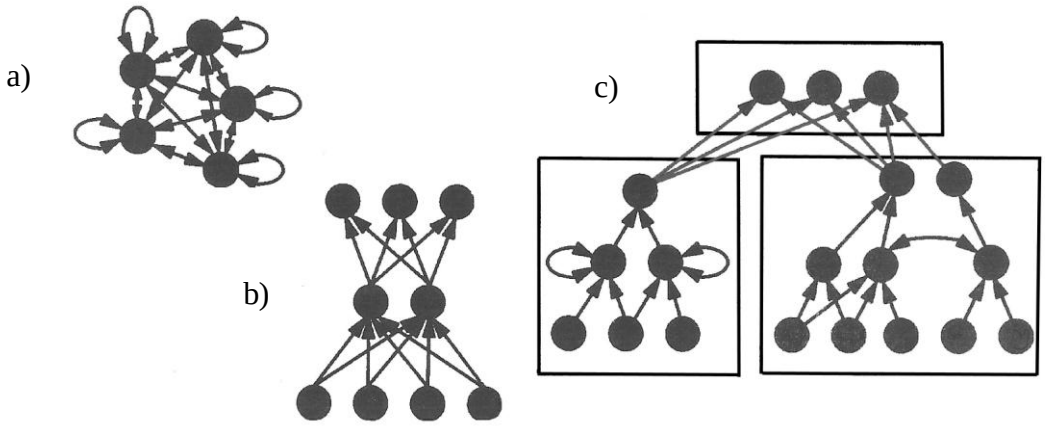

Figure 1. Various types of connectionist network architecture: a) A fully recurrent network; b) a three-layer feed-forward network; c) a complex network consisting of several modules. Arrows indicate the direction of flow of excitation/inhibition. (Elman et al., 1996)

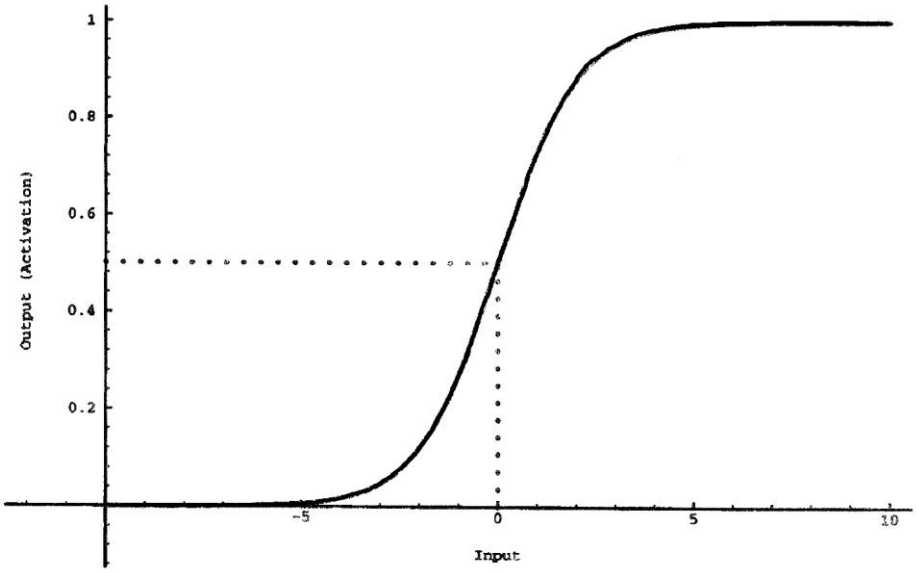

Figure 2. A sigmoid activation function often used for units in neural networks. Outputs are shown for a range of possible input activations. Network units with this sort of activation function exhibit an all or nothing response given a highly positive or highly negative input, but are sensitive to small differences around the zero point. In the absence of input, the units output 0.5, the middle of their response range. (Elman et al., 1996)

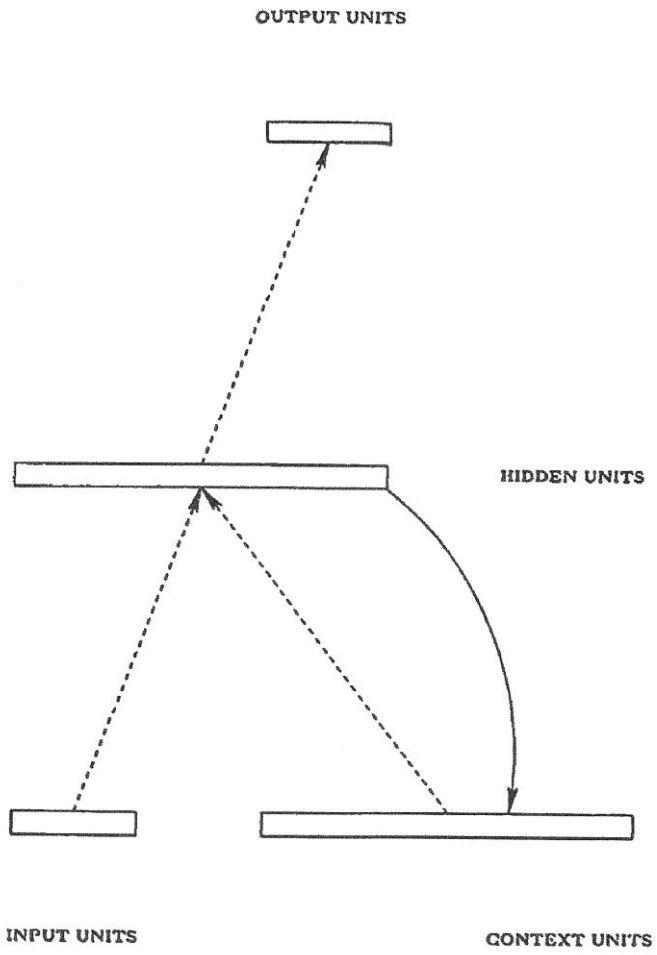

Figure 3. A simple recurrent network in which activations are copied from hidden layer to context layer on a one-to-one basis, with fixed weight of 1.0. Dotted lines represent trainable connections. (Elman, 1990)

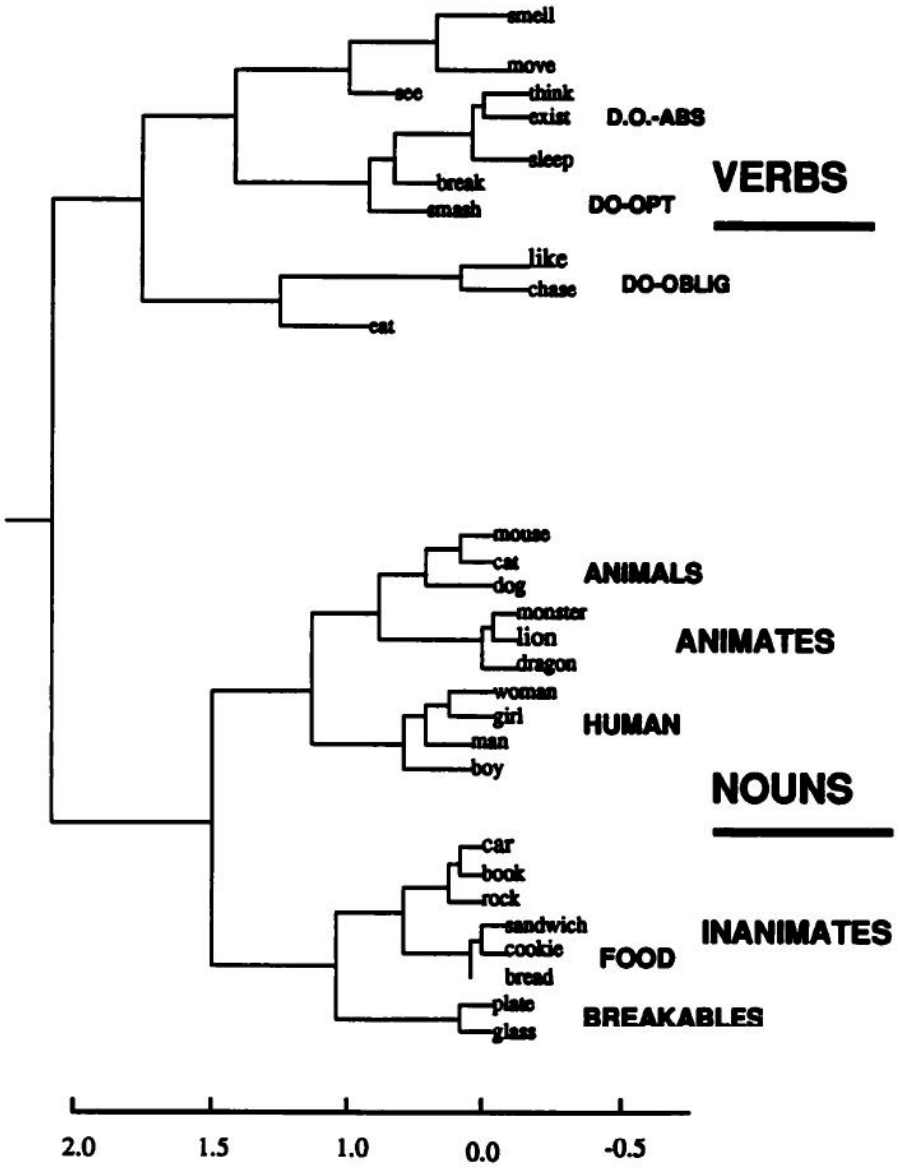

Figure 4. Hierarchical clustering diagram of hidden unit activations in the simulation with simple sentences. After training, sentences are passed through the network, and the hidden unit activation pattern for each word is recorded. The clustering diagram indicates the similarity structure among these patterns. This structure, which reflects the grammatical factors that influence word position, is inferred by the network; the patterns which represent the actual inputs are orthogonal and carry none of this information. (Elman, 1995)

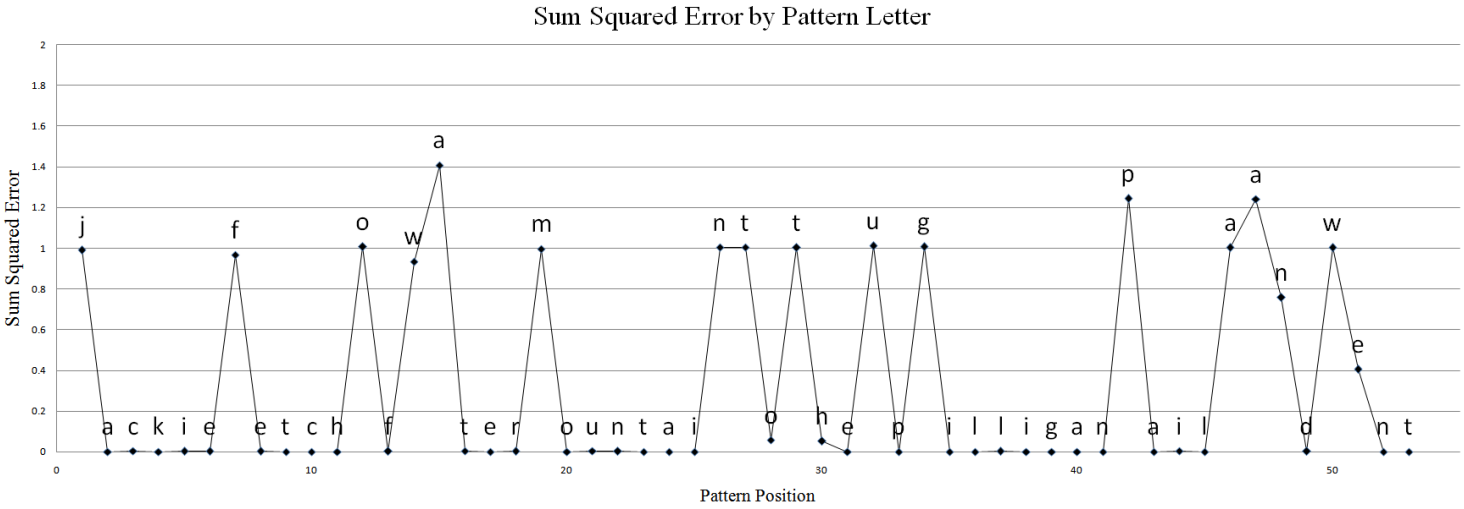

Figure 5. Sum squared error (SSE) by position in testing pattern for the model with 26 hidden units, trained for 10 epochs. Spikes in SSE for a letter indicate the model was unable to predict the occurrence of that letter in the pattern, and suggest the boundary for a word.

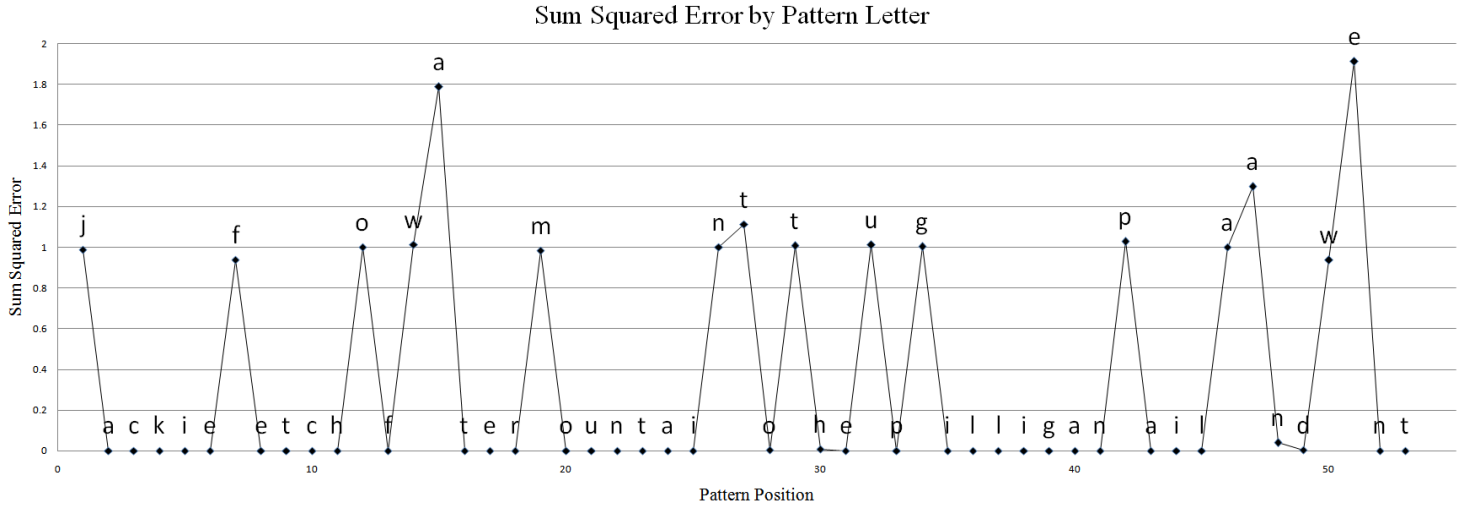

Figure 6. Sum squared error (SSE) by position in testing pattern for the model with 26 hidden units, trained for 50 epochs. Spikes in SSE for a letter indicate the model was unable to predict the occurrence of that letter in the pattern, and suggest the boundary for a word.

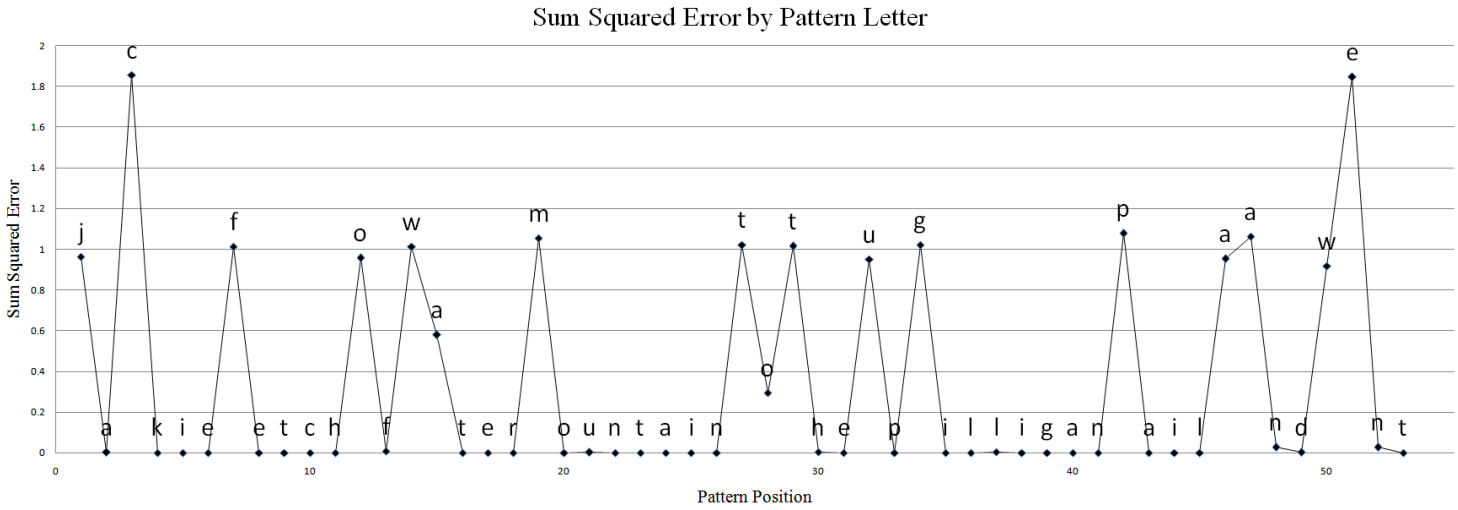

Figure 7. Sum squared error (SSE) by position in testing pattern for the model with 52 hidden units, trained for 10 epochs. Spikes in SSE for a letter indicate the model was unable to predict the occurrence of that letter in the pattern, and suggest the boundary for a word.

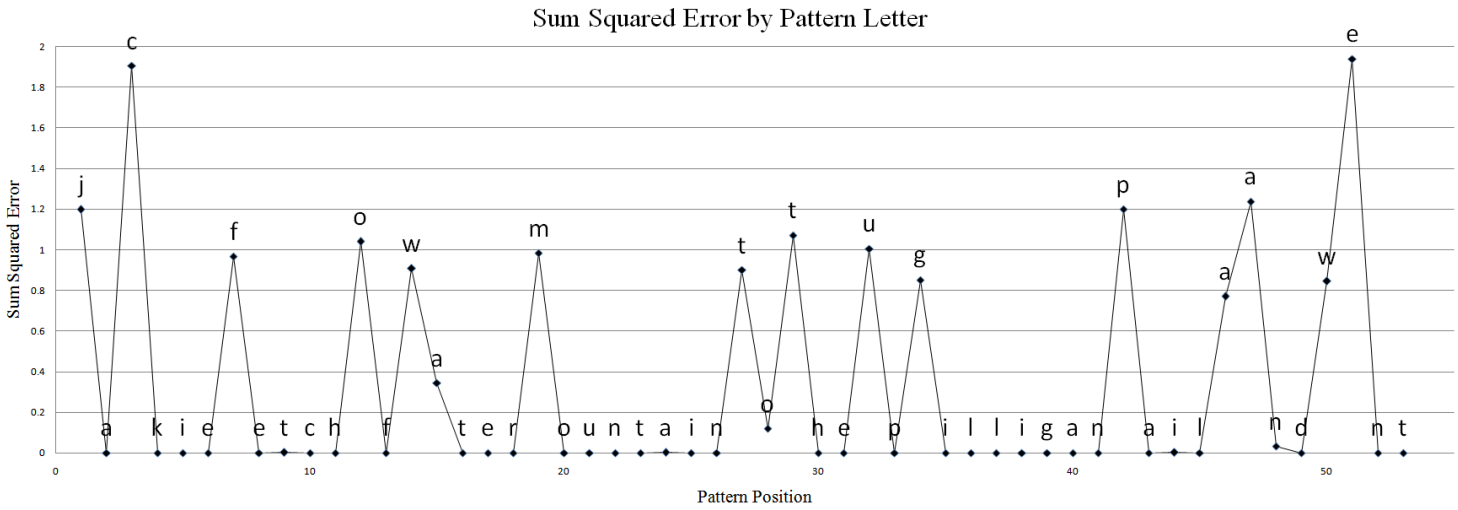

Figure 8. Sum squared error (SSE) by position in testing pattern for the model with 52 hidden units, trained for 50 epochs. Spikes in SSE for a letter indicate the model was unable to predict the occurrence of that letter in the pattern, and suggest the boundary for a word.

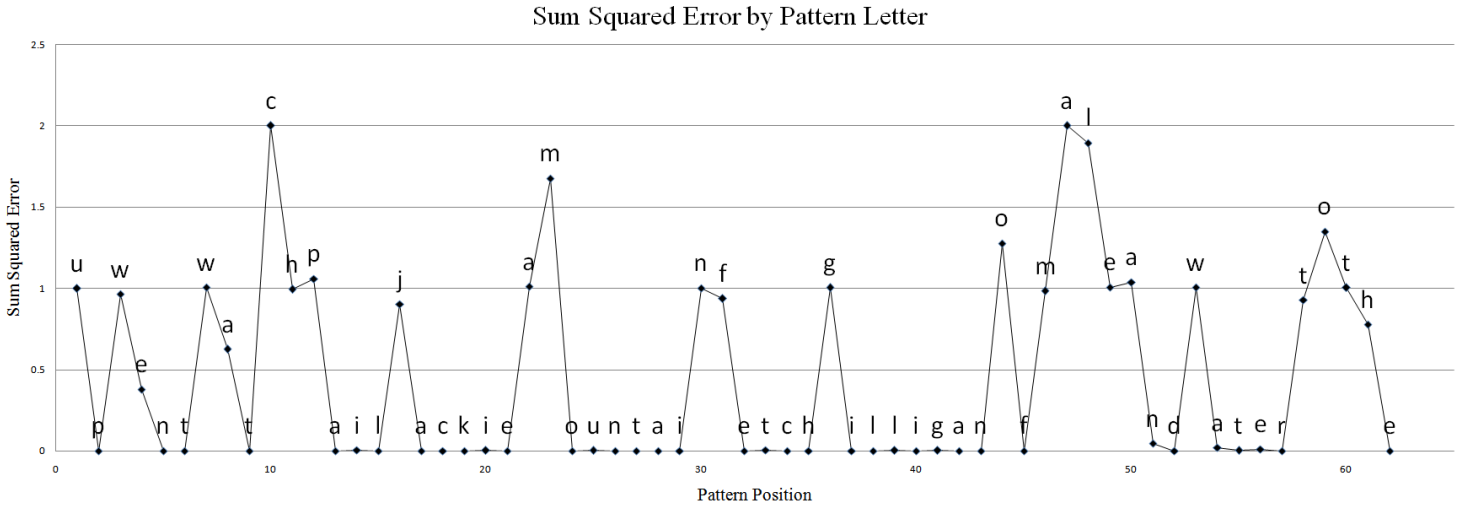

Figure 9. Sum squared error (SSE) by position in novel testing pattern for the model with 26 hidden units, trained for 10 epochs. Spikes in SSE for a letter indicate the model was unable to predict the occurrence of that letter in the pattern, and suggest the boundary for a word.

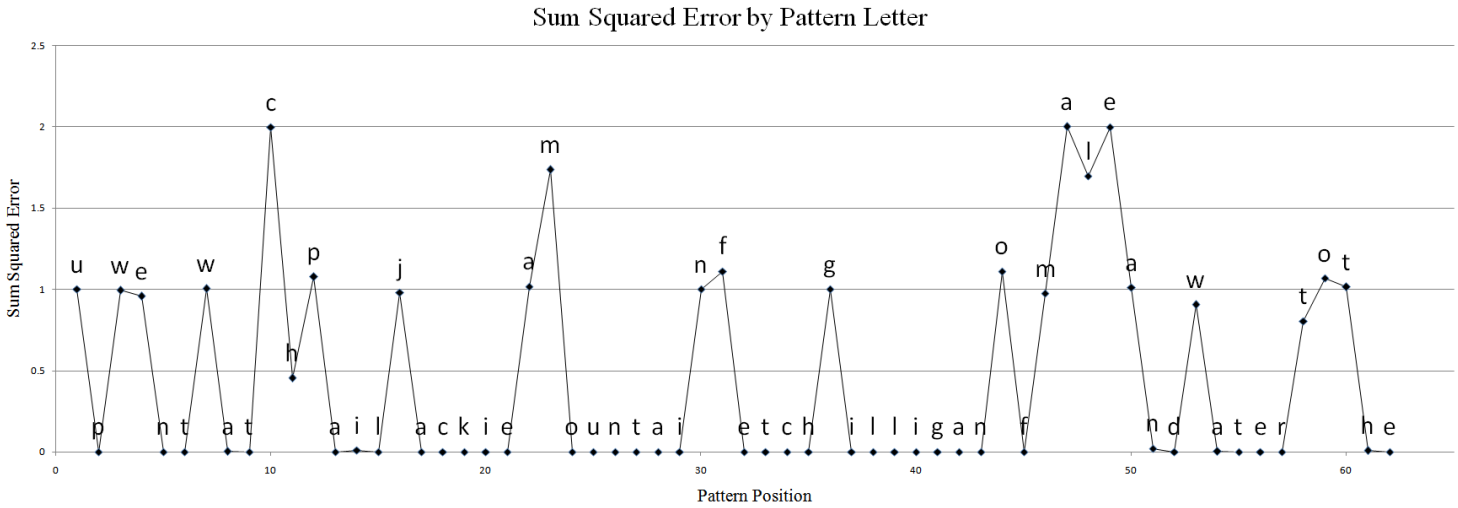

Figure 10. Sum squared error (SSE) by position in novel testing pattern for the model with 26 hidden units, trained for 50 epochs. Spikes in SSE for a letter indicate the model was unable to predict the occurrence of that letter in the pattern, and suggest the boundary for a word.

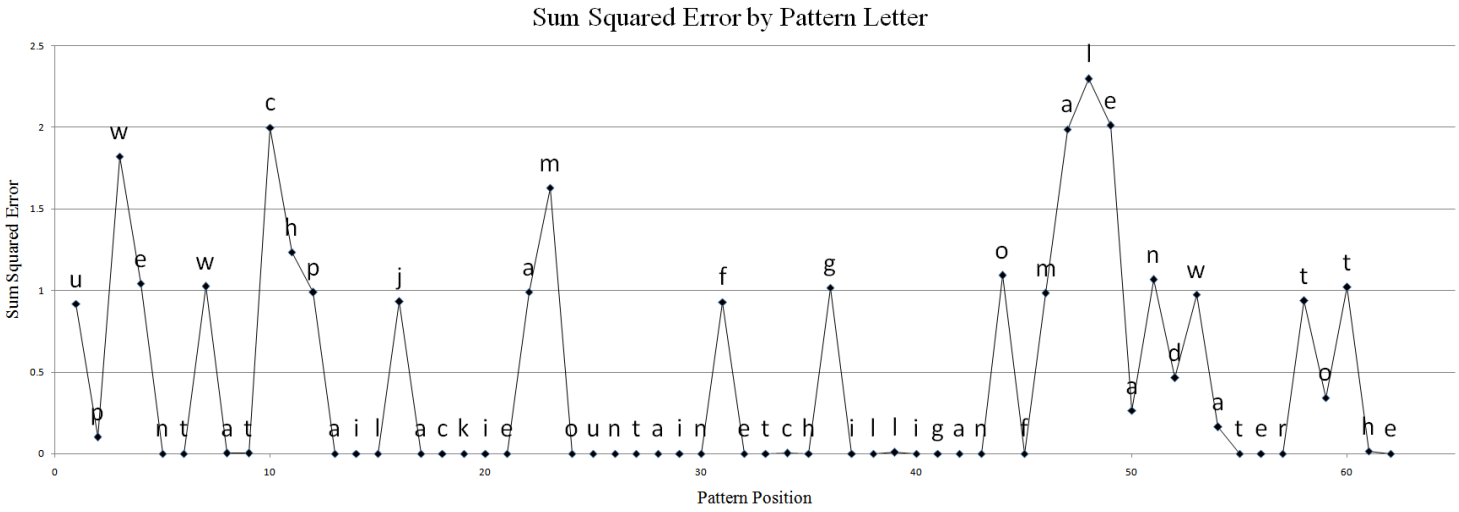

Figure 11. Sum squared error (SSE) by position in novel testing pattern for the model with 52 hidden units, trained for 10 epochs. Spikes in SSE for a letter indicate the model was unable to predict the occurrence of that letter in the pattern, and suggest the boundary for a word.

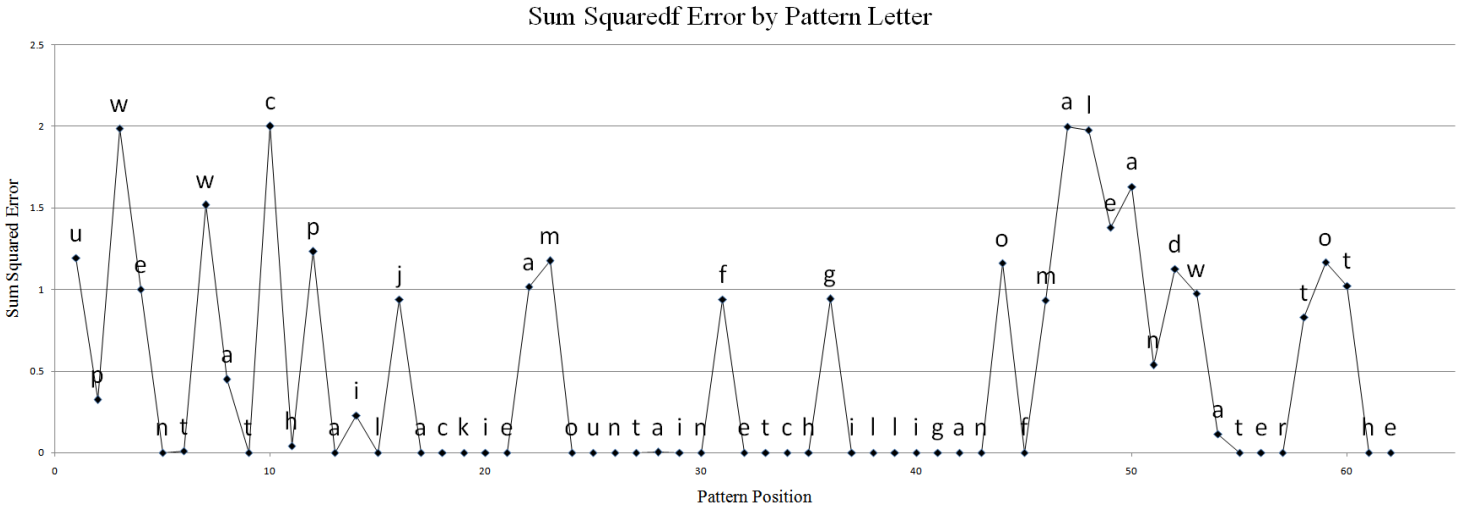

Figure 12. Sum squared error (SSE) by position in novel testing pattern for the model with 52 hidden units, trained for 50 epochs. Spikes in SSE for a letter indicate the model was unable to predict the occurrence of that letter in the pattern, and suggest the boundary for a word.