A Recurrent Network that performs a Context-Sensitive Prediction Task

Mark Steijvers
Department of Psychology
Indiana University
Bloomington IN 47405
msteyver@indiana.edu

Peter Grünwald
Department of Algorithmics
CWI, P.O. Box 4079
1009 AB Amsterdam, The Netherlands
pdg@cwi.nl

Abstract

We address the problem of processing a context-sensitive language with a recurrent neural network (RN). So far, the language processing capabilities of RNs have only been investigated for regular and context-free languages. We present an extremely simple RN with only one parameter z for its two hidden nodes that can perform a prediction task on sequences of symbols from the language $\{(ba^k)^n \mid k \geq 0, n > 0\}$, a language that is context-sensitive but not context-free. The input to the RN consists of any string of the language, one symbol at a time. The network should then, at all times, predict the symbol that should follow. This means that the network must be able to count the number of a's in the first subsequence and to retain this number for future use. We present a value for the parameter z for which our RN can solve the task for $k = 1$ up to $k = 120$. As we do not give any method to find a good value for z, this does not say anything about the *learning* capabilities of our network. It does, however, show that context-sensitive information (the count of a's) can be *represented* by the network; we analyse in detail how this is done. Hence our work shows that, at least from a representational point of view, connectionist architectures can handle more complex formal languages than was previously known.

Introduction

An important issue when modeling grammars and grammatical inference with recurrent neural networks (RNs) is to determine what kind of formal languages a recurrent neural network can process and generate. In this paper we show that a very simple recurrent network of a kind that has often been studied before is able to process a fairly complex language: a language that is neither regular nor context-free.

Regular languages represent the simplest class of formal languages in the Chomsky hierarchy (Hopcroft & Ullman, 1979). Regular languages are generated by regular grammars. Each regular language L has an associated deterministic finite state automaton (DFA) M and vice versa: M accepts all correct sentences of L and rejects all incorrect sentences. A more complex class in the Chomsky hierarchy is that of the context-free languages; the regular languages are a proper subset of this class. For each context-free language there is an associated push-down automaton (and vice versa). An even more complex language class is that of the context sensitive languages with the associated linear bounded automata. This class properly includes all context-free languages.

The theory of these languages and automata from a symbol processing perspective is well established (Hopcroft & Ullman, 1979). It is not clear however, what kind of automata RNs can implement. So far, only performance on regular and context-free languages has been reported (Cleeremans, Servan-Schreiber & McClelland, 1989; Giles *et al.*, 1992; Sun *et al.*, 1993; Wiles & Elman, 1995). This situation led us to investigate whether it is possible to go beyond these language classes; in this paper, we come up with a RN that performs a prediction task on the symbols of a context-sensitive language that is not context-free. To the best of our knowledge, this has never been attempted before – as RNs already have great difficulties in handling simple context-free languages, we take our task to be quite challenging for RNs. Our work also has repercussions for the role of connectionism in psychology. In our view, in order for connectionism to be a serious paradigm for psychology, it should be clear what its capabilities are when dealing with formal languages which lie at the heart of 'symbol-oriented' models (Wiles & Elman, 1995).

We will consider a type of recurrent neural network that has initially been explored by Jordan (1986) and more recently by Elman (1990) and Pollack (1991). More specifically, we use a second order recurrent network (Giles *et al.*, 1992; Omlin & Giles, 1992) simplified to having only one parameter, and we show by simulation that its nodes can represent the input to the network in a way that captures the essential structure of our context sensitive language.

When RNs are applied to processing languages, the solutions provided by the network are often best understood from a dynamical systems perspective (Omlin & Giles, 1994). This perspective can sometimes offer new insights and provide new mechanisms for solving tasks that are usually dealt with from a more traditional symbolic framework. From this point of view, the problem we will have to face here is to control the non-linear dynamics of the network in such a way that the regions in state space that correspond to the various symbols to be predicted are linearly separable.

The Task

Consider all sequences of the form:

$$b(a)^k b(a)^k \ldots \quad \text{for integers } k \geq 0 \tag{1}$$

In any such sequence symbol b is followed by k symbols a after which this subsequence repeats itself. Each value of k defines a unique sequence which we will call the k-sequence. For example, here are the initial segments of the 1-,2- and 3-sequences:

$$\begin{array}{ll} bababababa\ldots & (k=1) \\ baabaabaab\ldots & (k=2) \\ baaabaaaba\ldots & (k=3) \end{array}$$

The task we want our network to perform is the following: for any sequence of the form (1), after having been presented the first n symbols of the sequence, the network should correctly predict the $n+1$-st symbol of the sequence. The symbols are presented in order: at time $t = 1$ the first symbol is presented, at $t = 2$ the second and so on. Note that before the second b has been presented, the next symbol may turn out to be either a or b as it is not clear yet which sequence the network is dealing with. But after the second b, there is only one possible sequence left, and all future symbols are unambiguously determined. Therefore, if the actual sequence turns out to be $b(a)^k b \ldots$ for some particular k, then we want our network to correctly predict all future symbols at all times $t > k + 2$.

Complexity of the Task

Consider the language $L_{\text{CS}} = \{(ba^k)^n \mid k \geq 0, n > 0\}$. Thus for example, *bababababa* and *baaaabaaaa* are correct sentences of the language while *babaaa*, *babab* and *baaba* are not. It is clear that our task can be reinterpreted as follows: after having been presented a few initial symbols (the first 'run' of a's) one must correctly predict what the next symbols in the sequence must be, such that at some point in the future, the part of the sequence seen until then will be a correct sentence of the language L_{CS}.

Now L_{CS} is a context-sensitive language which, moreover, is *not* context-free. This can be proven using standard techniques of formal language theory; an exact proof can be found in the appendix. Intuitively, one can understand why L_{CS} is not context-free if one tries to recognize L_{CS} using a push-down automaton. A push-down automaton is roughly just a non-deterministic finite state automaton with a stack; it is clear that the only way to count the number of a's that have been seen already is to use this stack. After the first b symbol, one can fill the stack with the first k a symbols. Then, after the second b, one can empty the stack again, to produce the next k a symbols, but since the contents of the stack are then empty, the information about k is lost, and further processing of the sequence is impossible. Therefore, a single stack is not sufficient to solve this task. What we really need to do is to implement a counter that counts the number of consecutive a symbols after which it *retains* this value to process the next (more than one!) subsequences of a-symbols.

Just as one cannot use a push-down automaton to *recognize* a language that is not context-free, one cannot use it to *predict* the consecutive symbols of the correct strings of such a language either (Hopcroft & Ullman, 1979). It is in this sense that the power of a recurrent network that would perform well on our prediction task goes beyond the power of context-free grammars or, equivalently, push-down automata.

The Network

We use a second-order recurrent network with two input nodes, I_1 and I_2, two hidden nodes H_1 and H_2 and an output unit O. x_i^t denotes the activation of node X_i at time t. We employ a unary encoding of our symbols: a is encoded as $i_1 = 1$ and $i_2 = 0$, b as $i_1 = 0$ and $i_2 = 1$. The hidden node activations h_1 and h_2 at time t are mapped into those at time $t+1$ according to:

$$\begin{aligned} h_1^{t+1} &= f(\; i_1^t h_1^t w_1 + i_1^t h_2^t w_2 + \\ &\qquad i_2^t h_1^t w_3 + i_2^t h_2^t w_4 + w_5) \end{aligned} \tag{2}$$

$$\begin{aligned} h_2^{t+1} &= f(\; i_1^t h_1^t w_6 + i_1^t h_2^t w_7 + \\ &\qquad i_2^t h_1^t w_8 + i_2^t h_2^t w_9 + w_{10}) \end{aligned} \tag{3}$$

where f is the sigmoid discriminant function: $f(x) = 1/(1+e^{-x})$. We simplified this network by removing some of the recurrent connections: $w_2 = w_3 = w_4 = w_6 = w_9 = 0$. The substitutions $w_1 = w_7 = w_8 = z$ and $w_5 = w_{10} = -z/2$, simplify the network to:

$$h_1^{t+1} = f(i_1^t h_1^t z - z/2) \tag{4}$$

$$h_2^{t+1} = f(i_2^t h_1^t z + i_1^t h_2^t z - z/2) \tag{5}$$

The output O is a linear threshold unit that outputs prediction symbols a and b:

$$O_t = \begin{cases} a & \text{if } w_{O1} h_1^t + w_{O2} h_2^t + \theta_O < 0 \\ b & \text{if } w_{O2} h_1^t + w_{O2} h_2^t + \theta_O \geq 0 \end{cases} \tag{6}$$

A single parameter, the weight z determines the representation of the input symbols in hidden state space. The weights w_{O1}, w_{O2} and θ_O determine the linear equation[1] that divides the hidden state space in regions where the predicted symbol is a or b.

With computer studies, we wanted to find a z such that there is a combination of w_{O1}, w_{O2} and θ_O such that the predictions of the symbols of the sequences for $k = 1$ up to a maximum of m are correct. Therefore, for these sequences, the hidden node activations corresponding to a processed input sequence where the next symbol is a a or b should be linearly separable.

For $z = 3.9924$, we found that the predictions for the first $m = 120$ sequences of this task are linearly separable. In figure 1, the trajectories in hidden node space are shown for the first 6 sequences. These trajectories go from the left to the right; the successive points of the maps are connected by lines (the return trajectories to the left are omitted). The points indicated by filled circles are the points where k symbols a are received, so at these points, the output symbol should be b. The points indicated by open circles are the points where a symbol a should be predicted. The dashed line shows the linear equation that separates the a and b prediction symbols. Figure 2 shows the trajectories for all 120 sequences.

How the Network Does It

The behavior of the network can best be understood from a dynamical systems viewpoint. As all sequences consist of perpetually self-repeating subsequences, it is clear that for each value of k, the network eventually goes into an associated limit cycle. The real difficulty of our task is to combine

[1]Notice that we could just as well have taken the sigmoid discriminant function f again to determine the output activation as $O = f(w_{O1} h_1 + w_{O2} h_2 + \theta_O)$. If we then interpreted $O < 0.5$ as a and $O \geq 0.5$ as b, this would always yield the same predictions as our threshold unit. For recurrent neural networks, usually this latter approach is taken (Omlin & Giles, 1994). We have opted for the equivalent threshold approach in order to clarify the analysis of the hidden node activation space.

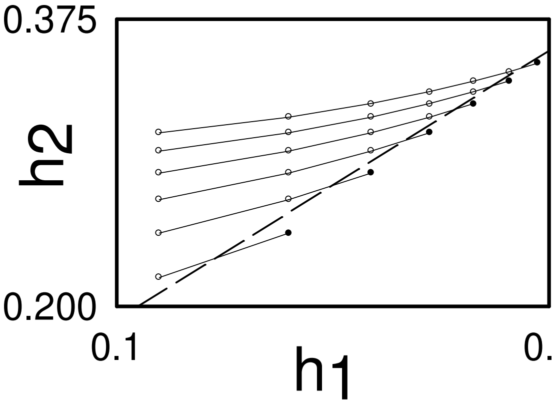

Figure 1: The trajectories of h_1 and h_2 in the network with $z = 3.9924$ are shown for $k = 1$ to 6.

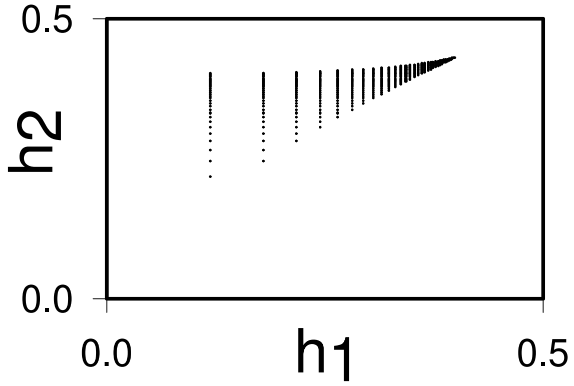

Figure 2: All points of the trajectories in hidden state space for the sequences $k = 1$ to 120.

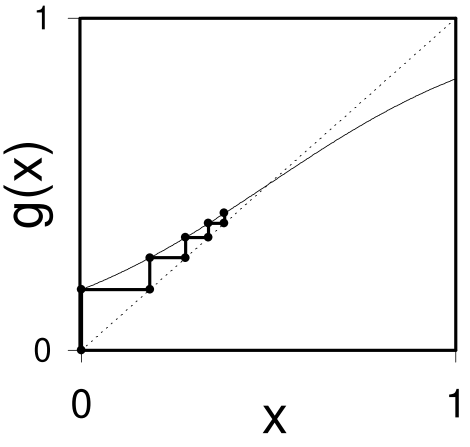

Figure 3: The map $g : x \mapsto 1/(1 + e^{-(z\cdot(x-1/2))})$ is shown for $z = 3$. This map has only one attractor $x = 1/2$. Five iterations on this map are shown starting on $x = 0$.

the dynamics of every limit cycle in such a way that it is possible to extract useful information from the cycle trajectories in hidden node state space. The non-linear nature of limit cycle trajectories makes it difficult to be combined with the linear nature of the separation functions provided by threshold output units. Our main idea is to simplify the network by introducing a *reset mechanism* for every limit cycle. The endpoints of the trajectories (before reset again) are then linearly separable from all other points of the different trajectories; as will be seen below, this is exactly what is needed for our prediction task. For a more detailed analysis of the network's performance, we need to look at the following function:

$$g(x) = f(z \cdot (x - 1/2)) \tag{7}$$

Here f is the sigmoid function again. We will write $g^{(0)}(x) = x, g^{(1)}(x) = g(x), g^{(2)}(x) = g(g(x))$ etc. For $0 \le z < 4$, g has only one fixed point: $g(1/2) = 1/2$ (figure 3). This point is an attractor[2]; as n increases, $g^{(n)}(x)$ converges to $1/2$ for all starting values x. Also shown in figure 3 are the iterations from $g^{(0)}(0)$ to $g^{(5)}(0)$.

Now let us suppose that we feed our network a sequence consisting only of a's (i.e. $i_1^t = 1, i_2^t = 0$). Then, if we start at time t and set x equal to h_1^t, we can see from (4) that updating h_1 becomes identical to iterating g:

$$h_1^{t+i} = g^{(i)}(x) \quad \text{for all } i \ge 0 \tag{8}$$

Exactly the same applies to H_2: if we had set $x = h_2^t$, then h_2 would have evolved according to g, as can be seen from (5).

But what happens to h_1^t when a symbol b arrives at time t? (i.e. $i_1^t = 0, i_2^t = 1$). We see that

$$\begin{aligned} h_1^{t+1} &= f(0 \cdot h_1^t \cdot z - z/2) \\ &= g^{(1)}(0) \end{aligned} \tag{9}$$

Thus each time a b arrives (in particular, at time $t = 1$ when the first one arrives), h_1^t will be 'reset' to $g(0)$. This, together with (8) implies that if, for any t, the previous $i + 1$ symbols were of the form ba^i, then h_1^t will *always* be equal to $g^{(i+1)}(0)$. In what follows, we will write $g^{(i)}(0)$ simply as $g^{(i)}$.

H_2 is influenced in a different manner when a b arrives. If a b arrives in a k-sequence at a time t with $t > 1$ (we will not consider the first b here), then h_2^{t+1} is changed as follows:

$$\begin{aligned} h_2^{t+1} &= f(1 \cdot h_1^t \cdot z + 0 \cdot h_2^t \cdot z - z/2) \\ &= g(h_1^t) \\ &= g(g^{(k+1)}(0)) \\ &= g^{(k+2)} \end{aligned} \tag{10}$$

Here h_1^t is equal to $g^{(k+1)}$ because we are in a k-sequence, and thus the b that arrives at time t has been preceded by ba^k.

In other words, whenever a symbol b is processed at time t in a k-sequence, the activation of H_1 is reset to $g^{(1)}$, while H_2 'takes over' from H_1: h_2^{t+1} is set to $g^{(k+2)}$. Returning to figure 2, we can see that in between two b's, node H_1 iterates g starting from $g^{(1)}$ to $g^{(k+1)}$, while H_2 iterates g from $g^{(k+2)}$ to $g^{(2k+2)}$. Thus the points (h_1^t, h_2^t) in hidden node

[2]One can actually prove this; see the work of Omlin & Giles (1994) where the same function is used for a different purpose.

activation space will always be of the form $(g^{(1+i)}, g^{(2+k+i)})$ for some $0 \leq i \leq k$. Also, the points $(g^{(1+k)}, g^{(2+2k)})$ coincide exactly with the hidden node activations when all k a's of the k-sequence have been presented, i.e. when a b should be predicted. On the other hand, all points at which an a should be predicted must be of the form $(g^{(1+i)}, g^{(2+k+i)})$ with $i < k$. As $g^{(t)}$ increases with t, this means that it is enough to make sure for all $i > 0$ that $(g^{(i)}, g^{(2i)})$ lies beneath the separating line while $(g^{(i)}, g^{(2i+1)})$ lies above it; see figure 1 and 2 again. As can be seen there, for increasing i, $g^{(i)}$ goes to $1/2$, but (for our choice of z) the points $(g^{(i)}, g^{(2i)})$ are connected through an almost linear function. This partially explains why the points $(g^{(i)}, g^{(2i)})$ and $(g^{(i)}, g^{(2i+1)})$ are linearly separable for such a large range of i.

Related Work

Concerning *regular* languages, Omlin & Giles (1994) provide an algorithm that, given any DFA M as input, outputs (a description of) an equivalent second-order recurrent network R. Here 'equivalent' means that R outputs a 1 if and only if its input is a string of the regular language corresponding to M.

For context-free languages, things get more complicated. Wiles & Elman (1995) studied the behavior of a RN on the language $a^n b^n$ which is context-free but not regular. They trained a small RN to predict the symbols from strings from this language with n ranging from 1 to 12. In one of several training sessions, they found a RN that exhibited generalization to $n = 18$. The similarity to our work consists in the fact that both $a^n b^n$ and L_{CS} can be processed using only a counter rather than a complete stack or tape; however, in Elman & Wiles' work the trained network turned out to count in a completely different manner from ours, namely by combining the dynamics of attractors and repellors that implement counting up and down respectively; like a stack, this mechanism 'forgets' the number of a's after processing the b's.

Sun *et al.* (1993) also studied RNs when trained on context-free languages but their RNs were augmented with a stack. Here, the task for the RN was to accept or reject a string as belonging to the trained language. They achieved very good generalization performance when training their network with short example strings of some context-free languages including $a^n b^n$. The advantage of providing the RN with a real stack is that *different* symbols can be written on the stack and read off again, while it remains to be seen whether that can be achieved with techniques like those used by Wiles & Elman (1995) and us. On the other hand, the stack extension cannot be of any help in representing languages like L_{CS} that are not context-sensitive, and does not show whether RNs by *themselves* are more powerful than DFA's.

Discussion and Conclusion

It is important to realize that we did not use learning algorithms for the network itself to come up with solutions; instead, we 'hard-wired' the weights to solve the task. We felt the need to separate the possibilities of *learning* a language as difficult as a context-sensitive one from the possibility of *representing* it with a RN. It could turn out that there are weights for the RN which make for a good representation of the language, while none of the known learning algorithms for RNs will ever find these weights. On the other hand, it could be that the representational capability of the RN was not powerful enough to start with. In the words of Minsky & Papert (1988): 'no machine can learn to recognize X unless it possesses, at least potentially, some scheme for representing X'. In this work, we showed that at least some RNs *do* have the representational capabilities to deal with at least some languages that are not context-free. We certainly do not think that RNs will turn out to be capable of handling *any* context-sensitive language; we *do* think however that they can provide a new and interesting manner to process some languages that are rather complex from the point of view of formal language theory.

References

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

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

Giles, C., Miller, C., Chen, D., Chen, H., Sun, G. & Lee, Y. (1992). Learning and extracting finite state automata with second-order recurrent networks. *Neural Computation*, 2, 331–349.

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

Jordan, M. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Ninth Annual conference of the Cognitive Science Society* (pp. 531–546). Hillsdale, NJ: Lawrence Erlbaum Associates.

Minsky, M. & Papert, S. (1988). *Perceptrons*. Cambridge, MA: MIT Press.

Omlin, C. & Giles, C. (1994). Constructing Deterministic Finite-State Automata in recurrent neural networks. Technical report 94-3. Troy, NY: Rensselaer Polytechnic Institute, Department of Computer Science.

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

Sun, G., Giles, C., Chen, H., & Lee, Y. (1993). The neural network push-down automaton: model, stack and learning simulations. Technical Report UMIACS-TR-93-77 & CS-TR-3118. College Park, MD: UMIACS.

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

Proof that L_{CS} Is Not Context-Free

We want to prove the following:

Theorem 1 *The language L_{CS} defined by $L_{CS} = \{(ba^k)^n \mid k \geq 0, n > 0\}$ is not context-free.*

We will prove this theorem using the pumping lemma for context-free languages, a standard tool for proving certain languages not to be context-free (Hopcroft & Ullman, 1979). Before giving the actual proof, we will first state the pumping lemma and explain the proof technique used. In the following, i, j, k, m and n will be variables taking on non-negative integer values. t, u, v, w, x, y and z will be variables taking on string values. A string is a concatenation of zero or more symbols taken from the alphabet $\Sigma = \{a, b\}$. For any string z, $|z|$ denotes the length of (number of symbols appearing in) string z. xy stands for the concatenation of strings x and y. We are now ready to state the pumping lemma (for a proof of the pumping lemma itself and details about the notation, see for example Hopcroft & Ullman (1979)):

Lemma 1 (Pumping Lemma for Context-Free Languages) *Let L be any context-free language. Then there is a constant n, depending only on L, such that if z is in L and $|z| \geq n$, then we may write $z = uvwxy$ such that*

1. $|vx| \geq 1$,
2. $|vwx| \leq n$, and
3. for all $i \geq 0$, uv^iwx^iy is in L.

The general idea behind the proof of theorem 1 is as follows: we first suppose that L_{CS} were context-free. Then the pumping lemma holds, so the constant n mentioned in the pumping lemma exists. The trick is to cleverly pick a string z in L_{CS} with length $|z| \geq n$ such that any choice of u, v, w, x and y with $z = uvwxy$ will violate at least one of the three conditions in the pumping lemma. As the pumping lemma states that the three conditions hold for *any* z in L_{CS} with length $|z| \geq n$, this shows that the pumping lemma does not hold after all. Thus assuming L_{CS} is context-free leads to a contradiction; L_{CS} is therefore not-context free. We now proceed to the actual proof:

Proof of Theorem 1: Suppose that L_{CS} were context-free. Let n be the constant of the pumping lemma. Consider the string $z = ba^nba^nba^n$. It is clear that z is in L_{CS} and that $|z| \geq n$. Write $z = uvwxy$ such that it satisfies the conditions of the pumping lemma. We must now find out where v and x, the strings that can get 'pumped', lie in $ba^nba^nba^n$. Since $|vx| \leq |vwx| \leq n$, vx contains at most one b. We can distinguish two cases: 1) vx contains no b's at all ; 2) vx contains exactly one b.

In case 1), as $|vwx| \leq n$, we must have that $|u| + |v| \geq 2n + 3$. This means that either u can be written as $u = ba^nt$ for some t or y can be written as $y = tba^n$ for some t. Now consider the string uwy (the string uv^iwx^iy with $i = 0$). As $|vx| \geq 1$ and vx contains no b's, the string uwy contains less than $3n$ a's but still three b's. However, either u starts with ba^n or y ends with ba^n. So uwy is of the form $ba^iba^jba^k$, where either i or k must be equal to n and at least one of i, j and k is less than n. This means that uwy is not of the form $(ba^{k'})^{n'}$, and thus uwy is not in L_{CS}. But by the pumping lemma $uwy = uv^0wx^0y$ must be in L_{CS}; a contradiction.

In case 2), the string uwy contains only 2 $b's$. Suppose first that the leftmost b is missing in uwy (I.e. u is the empty string and $vx = bt$ for some string t). Then uwy is of the form $a^mba^nba^n$. This string can only be in L_{CS} if $m = 0$. But then $|vx| = |uvwxy| - |uwy| = n + 1$, while $|vx| \leq n$; a contradiction. So suppose that the middle b is missing; then uwy is of the form ba^mba^n. This string can only be in L_{CS} if $m = n$. Again, it follows that $|vx| = n + 1$ so once more, we have arrived at a contradiction. Finally, suppose that the rightmost b is missing. Again, this would lead to $|vx| = n+1$; another contradiction.

We thus see that supposing that L_{CS} is a context-free language inevitably leads to a contradiction. Therefore, L_{CS} is not a context-free language. $\square$