Degraded Printed Word Recognition with a Hopfield-style Network *

Arun Jagota (jagota@cs.buffalo.edu)
Dept Of Computer Science
State University Of New York Buffalo

Abstract

We have previously developed and analysed a new Hopfield-style network model. This model has the advantage of higher storage capacity and less interference between stored memories than the standard discrete Hopfield network. In this paper, this model is applied towards machine printed word recognition. Words to be recognised are stored as content-addressable memories. Word images are first processed by an OCR. The network is then used to postprocess the OCR decisions. It is shown that when the number of stored words is small, the network can reliably recall the correct word and when the number of stored words is large, exact recall performance deteriorates but the network can still be used to postprocess the OCR output with very good performance. Lastly, very large dictionaries are handled by partitioning them by word length and storing them in multiple networks and this scheme gives very good performance.

1 Introduction

The Hopfield-style Network [Jag90b], is a one-layer recurrent network, a variant of the Hopfield Network [Hop82], developed specifically for content-addressable memories. Compared with the standard discrete Hopfield network, it has better properties for content-addressable memories. Specifically, it has very high storage capacity, less interference between stored memories, rapid (linear-time) retrieval of stored memories, with the dynamics of such a retrieval being noise-removal (from the input) followed by completion (filling in missing information in the input). The last property is not possible in the standard discrete Hopfield network with its auto-correlation learning rule. To go with this last property, the Hopfield-style network has another nice property. When the network is overloaded (large number of stored memories), noise-removal is still reliable, even though noise-removal followed by completion is not. For word recognition, when the network is overloaded, we have exploited this property and used the network in noise-removal-only mode. This only removes noise in the input, but with excellent performance. With the standard Hopfield network this would not be possible. The Hopfield-style network can also take advantage of the structure of the memories to be stored in ways that the standard discrete Hopfield network can not. This is mainly because the essential characteristics of the weight matrix can be described by a graph underlying the network, a graph that captures the structure of the stored memories and whose maximal cliques are the stable states of the network. The structures of the stored memories can be related to the properties of the graph. This has allowed us to show for example that if the stored memories are sparse, the network retrieval time is provably better and the network capacity is expectedly better. For word recognition, the memories *are* sparse. The test input to the network can be a weighted set, allowing the network to support very general kinds of information in test input. This includes confidence values on features of the test input (elements of the set), multiple hypotheses for the same feature (multiple elements in the set) and so on. Due to the network's dynamics, noise-removal followed by completion, multiple hypotheses in a test input are always resolved to at most one hypothesis before any completion is done. Test input to the standard discrete Hopfield network is a binary vector which can not support confidence values. Multiple hypotheses can be supported during testing but the standard Hopfield network cannot exploit them in its dynamics in the way the Hopfield-style network can. For word recognition, such flexible test representations have been very beneficial. To find the best recognition performance, many different forms of input to the network have had to be explored. The network test input representation (weighted set) has allowed us to explore all such forms. For example, we have been able to use confidence values on features in the input. Secondly, we have been able to include

*Submitted to *International Journal of Patter Recognition and AI - Special Issue on Neural Networks in Vision and Pattern Recognition*

multiple competing hypotheses (values) for a single feature in the input and due to its dynamics the network has always removed all but atmost one such hypothesis before doing any completion. This has allowed us to use the network in noise-removal-only mode to start with a set containing collections of many competing hypotheses and remove all or all but one hypothesis from each competing set.

This paper is organised as follows. Section 2 formulates our specific problem of word recognition as that of content-addressable memories on the Hopfield-style network and reviews our previous storage capacity results. Section 3 describes the Hopfield-style network, Section 4 describes how words are stored in the Hopfield-style network and reviews the theoretically proven properties that the network has for their storage and retrieval. It also shows how spurious memories develop. Section 5 goes back to our specific application and describes the format and information content of the OCR output, on a word image. Section 6 presents experimental results on storing a small dictionary and postprocessing the output of a real OCR on images whose words are contained in the stored dictionary. Section 7 presents the mechanism for storing very large dictionaries by partitioning and storing the partitions in multiple networks. It also presents experimental results on storing a very large dictionary this way and postprocessing the output of a real OCR on images whose words are contained in the stored dictionary. Since even the partitioned dictionaries are too large for reliable recall, the network is used in noise-removal-only mode. The results indicate excellent noise removal performance. Section 8 presents a search rule for taking the (noise-removed) network output of the experiments in Section 7 and finding the word in the dictionary that matches it. Experimental results are also reported in this section showing that the search rule has far better performance on (noise-removed) network output than on the original OCR output. Section 9 presents a special case of the network and suggests that it should work nearly as well if only the noise-removal-only mode is to be used. This has implications towards analog/optical implementations. Section 10 is the conclusion. The main advantage when the dictionary size is small is that the network can reliably retrieve (recognise) the correct word very rapidly, specifically in about 20-30 network steps, translating to about 200 milliseconds on a SUN-4. When the dictionary size is large, the network can reliably remove the noise from the input very rapidly, in about 10-20 cycles in the worst case, which also translates to about 100 milliseconds on a SUN-4.

2 Problem Formulation

In this paper, we restrict ourselves to the problem of recognising words from an image when the dictionary is given.

Author's note: The next two paragraphs are new - not in the manuscript submitted to IJPRAI

Specifically an OCR will generate a set of letter hypotheses (zero, one or more per position) and the task is to find the word, from the dictionary, in the image. This problem can be treated as one of *supervised learning*, that is to use pairs of (OCR generated hypothesis set, correct word) to train a learning device that when given an OCR generated hypothesis set, perhaps not seen before, the device reliably produces the correct word. One supervised learning method that might work is the Bayesian Classifier (statistical pattern recognition), but in practice, this could lead to space and performance difficulties because the number of classes (dictionary words) could get too large (10,500 in our experiments). The number of classes could be reduced by Compound Bayesian sequential decision methods. However these are more difficult to analyse theoretically , and it is not clear how well they would perform on this problem, in practice. Another supervised learning method that might work is the multi-layer feedforward neural network with error back-propagation. One possible advantage of this, over Bayesian classification, could be that the network might be able to form efficient *distributed representations* of the dictionary words, while learning. This would be a big space advantage for large dictionaries. It is unclear, however, if in practice, the network can indeed successfully form compact distributed representations of words, as the dictionary size gets large (say 10,500 words). There are many representation issues to be explored here, specifically how to represent words in the network input and output layers to try getting a compact distributed representation of words in all layers. Even if the network can form efficient distributed representations of the dictionary words, it will very likely need huge amounts of training to learn and to do so. Another reason for (possibly) large training times is that the training set of pairs of (OCR generated hypothesis set, correct word) needed for the network to perform well is likely to be very large. This is because the number of different OCR hypothesis sets generated from images of the same word can be quite large. In other words, the task is to learn a many-to-1 mapping (OCR sets to words) where the many can indeed be *many* and the cardinality of the co-domain (range) of the many-to-1 mapping (that is, number of words) could also be very large.

An alternative way to formulate our problem is as a *search* problem. Specifically, given an OCR hypothesis

set, a search can be conducted on a stored dictionary to find the word that best matches it. One major advantage of this scheme, if it works, is that no long period of supervised training, as for back-propagation, is required. The search criterion could be based on some distance measure. One such measure is the symmetric set difference (Hamming distance) between two strings. When the OCR hypothesis set can be interpreted as a string (for example OCR final-decisions, see Section 5), this such a distance based criterion could be used. However, in many cases, the OCR hypothesis set cannot be interpreted as a string, since it contains multiple letter hypotheses at one position (OCR final-with-confusion, OCR TOP-3, see Section 5). It is unclear how to define a distance measure between sets of (multiple) hypotheses and word strings and even if it is definable, how well it would work. One might consider the possibility of taking an OCR (multiple) hypothesis set and generating strings from all their legal combinations and searching the dictionary with each such string as the key, using the defined distance measure. The main problem with this is that the number of strings generated in this way, from a typical (multiple) hypothesis set (see Section 5, example on OCR final-with-confusion) could be (combinatorially) very large. Hence this involves much longer testing times, since searches have to be conducted on the entire dictionary with each string as the key. Secondly, it is not clear how to select a unique word from the list of words that each search with a specific key retrieves. This is because, while the rules of selection, when searching with a specific key, are defined via the distance measure, they do not apply to selecting between the results of different keys.

Authors Note: End of new -

One key issue is how the dictionary should be organised and how it should be searched so as to maximize performance at minimal cost (search time). This problem has been studied previously and in the context of word recognition many methods have been proposed and explored [SHC83, BT80, ST79, SB82]. Several ways of organising a dictionary are described in [Pet80].

In this paper, we formulate the problem of organising the dictionary and searching it as that of content-addressable memories in a one-layer neural network. To our knowledge, we are the first to have formulated the problem this way [Jag90a]. Specifically, words in the dictionary are stored in the Hopfield-style network in a representation first shown in [Jag90a].

Authors Note: This is new - not in IJPRAI manuscript We note, as like the conventional search formulations discussed earlier, training the network involves *only* storing the words. In particular, the network never sees any OCR hypothesis sets during training. In addition to making the training process extremely rapid, another advantage is that the network trained in this way could be used for a variety of applications, since what the network is trained on is completely independent of what the network will be tested on.

Authors Note: End of new -

Given a word image, it is assumed that individual characters can be recognised by an OCR. The set of OCR decisions on a word image constitutes the test input to the network, from which the network retrieves the desired stored word. One important advantage of this scheme is that the network can naturally tolerate errors and missing information in the input. A second important advantage is that its retrieval is very rapid and independent of the number of stored words. This contrasts with many conventional retrieval schemes in which every word in the dictionary needs to be searched and so the retrieval time scales with dictionary size. The main problem with our scheme is that spurious memories can develop when storing words in the network, obstructing retrieval performance of the network. The rate of spurious memory development increases dramatically with the number of stored words. In [Jag90a], we have experimentally evaluated the storage capacity (number of spurious memories) of the network using different dictionaries (sets of words) of various types and sizes. We have also formulated and experimentally evaluated an algorithm for unlearning spurious memories. The main result of [Jag90a] was that the network can be used for reliable recall if the number of stored words is between 100-600 for the representation [1] used in [Jag90a]. Near the upper bound, the spurious memories are a bit high, but the unlearning algorithm of [Jag90a] learns to reliably avoid them during recall. We also report in [Jag90a] that when the number of stored words is very large, for example 5000 words, there are too many spurious memories, and the unlearning algorithm is not effective either, in learning to avoid them during recall.

In the current paper we apply the knowledge of our results in [Jag90a] to decide how to handle word recognition. We first investigate word recognition issues when small dictionaries (upto 500 words) are stored. Even in this limited case, our explorations with the network have led to many discoveries. Firstly, it is seen that the OCR output is complex and the network retrieval performance depends on how and what part of it is used. As mentioned earlier, the first important point in this regard is that the network allows and exploits

[1] Explicitly representing higher-order features can improve the network performance [Jag90b], but then many more units are required for the same set of memories to be stored

a variety of information in the test input, which has enabled us to use the OCR output in many ways that we have wanted to. This contrasts with the standard Hopfield network which does not allow or exploit certain information that we would like it to. Exploiting this, one of our discoveries has been that, especially for words of small length , the network exhibits better performance when presented with hypotheses derived from a confusion matrix describing the OCR's confusion characteristics. This requires taking the OCR output, one letter decision per position in word, and replacing each letter by many letters derived from a confusion matrix. This produces a representation containing multiple competing letter hypotheses at each position. As we have seen earlier, this entire set of hypotheses can be directly given to the network as input. The network naturally relaxes the hypotheses to settle to one letter at each position.

Experiments [Jag90b] had shown that when the dictionary size was large and the Hopfield-style network was still used in retrieval mode (noise-removal followed by completion) , even though the number of spurious memories retrieved was very high, indicating low performance for exact recall, the number of letters wrong in the spurious memories was still very low. In other words, if the measure was the percentage of letters correct in the network output, the network was still doing quite well. We should note that this phenomenon arises only because the network forms a *distributed* representation of the stored memories. Conventional methods that don't form distributed representations can clearly not exhibit such characteristics. This suggests that the network still retains good performance on large dictionaries if viewed as a filter, that is a preprocessing component in a more complex recognition system. The variation of this that we've discussed earlier is to use the network in noise-removal-only mode. Experimental results [Jag90b] justify use of the network as a filter in both the retrieval and noise-removal-only modes.

3 The Hopfield-style Network - Brief description

In this section, we briefly review the Hopfield-style network, described in detail in [Jag90b]. The Hopfield-style network is a fully connected one layer network with $n+k$ units ($0 \leq k \leq n$), the i^{th} unit denoted by p_i and the connection (weight) between p_i, p_j denoted by w_{ij}. Connections are symmetric ($w_{ij} = w_{ji}$) and there are self-connections (w_{ii}). $p_i(t)$ also denotes the state of the i^{th} unit. During testing, initially, $p_i(t)$ can be continuous (in $[0,1]$) but in the stable (final) state, $p_i(t)$ are always binary (in $\{0,1\}$). With the first n units, we associate a symbol s_i with unit p_i and let D denote the set of these n symbols. With the remaining k units ($p_{n+1}, \ldots p_{n+k}$) we associate the symbols $1, \ldots k$ to represent the sizes of the sets on D we wish to store. Such units are called *size* units. We will see that they are used during learning to prevent situations in which a memory to be stored collapses , after learning, into another memory if the latter is a (strict) superset of the former. Sometimes we will refer to a unit by its symbol name and vice versa. The initial *long-term* state of the network is

$$\begin{aligned} w_{ij}(0) &= \rho \quad \forall i,j \;\; i \neq j \\ w_{ii}(0) &= 0 \quad \forall i \end{aligned} \tag{1}$$

where ρ is a global network parameter < 0.

Learning Rule

Given a set $M \subseteq D$ to be stored at time t, we add, if necessary, the size unit to it as follows

$$M_s = \begin{cases} M \cup \{|M|\} & \text{if } k \geq |M| \\ M & \text{otherwise} \end{cases}$$

Then for all $i, j \neq i$

$$w_{ij}(t+1) := \begin{cases} \frac{\lambda}{2} & \text{if } s_i,\ s_j \in M_s \text{ and } w_{ij}(t) = \rho \\ w_{ij}(t) + \lambda * f(w_{ij}(t)) & \text{if } s_i,\ s_j \in M_s \text{ and } w_{ij}(t) \neq \rho \\ w_{ij}(t) & \text{otherwise} \end{cases} \tag{2}$$

$$w_{ii}(t+1) := \begin{cases} w_{ii}(t) + \lambda * f(w_{ii}(t)) & \text{if } s_i \in M_s \\ w_{ii}(t) & \text{otherwise} \end{cases} \tag{3}$$

where $\lambda \in (0,2]$ is the learning rate. We note that $\lambda > 0$. And,

$$f(x) = \frac{1-x^2}{2} \tag{4}$$

Energy descent rule

During testing, the input (initial network state) can also be represented by a set $S_m \subseteq D \cup \{1, \ldots k\}$, but some elements in S_m can also be *weighted* or *clamped* to 1. *weighting* can be used to provide an (initial) confidence value (in $[0, 1]$) to a unit and *clamping* to 1 to force the network to keep the symbol in the set (the unit active) throughout energy-descent. Also, as shown above, size symbols can be used in S_m. The notation we will use for S_m is,

$$S_m = \{s_{i_1}^{*1}, s_{i_2}, .., v_{i_j}.s_{i_j}\}, \qquad S'_m \subseteq D$$

where S'_m is the set corresponding to S_m with all the $v'_{i_k}s$ and $*1$ removed. v_{i_k} (in $[0, 1]$) denotes the weight of an element in S'_m and $*1$ denotes that it is clamped to 1 (not allowed to switch OFF). Given S_m, the initial states of the units p_i are assigned as follows.

$$p_{i_k}(0) = \begin{cases} v_{i_k} & \text{if } v_{i_k}.s_{i_k} \in S_m \\ 1 & \text{if } s_{i_k} \text{ or } s_{i_k}^{*1} \in S_m \\ 0 & \text{otherwise} \end{cases}$$

Given the state of the network at time t, the state at time $t + 1$ is computed as follows. Clamped (*1) units are not allowed to switch.

$$n_i(t+1) = \sum_j w_{ij} * p_j(t) \tag{5}$$

$$V_i(t+1) = \begin{cases} 1 & \text{if } n_i(t+1) > 0 \text{ and } p_i(t) < 1 \\ -1 & \text{if } n_i(t+1) < 0 \text{ and } p_i(t) > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta E_i(t+1) = -V_i(t+1) * n_i(t+1) \tag{6}$$

where $n_i(t+1)$ is the net input into p_i at time $t+1$, V_i is the direction p_i is allowed to switch in and ΔE_i is the change in energy this switch would cause. $n_i(t+1), V_i(t+1), \Delta E_i(t+1)$ above are computed in parallel for all units p_i. Next, the unit $p_{i_{min}}$ with the smallest energy $\Delta E_{i_{min}}(t+1) = min(\Delta \bar{E}(t+1))$ is picked. At $t+1$, if $\Delta E_{i_{min}}(t+1) \geq 0$, the state of no unit is changed else only the state of $p_{i_{min}}$ is changed as follows.

$$p_{i_{min}}(t+1) = \begin{cases} 1 & \text{if } V_{i_{min}}(t+1) = 1 \\ 0 & \text{if } V_{i_{min}}(t+1) = -1 \end{cases}$$

From the energy-descent rule, we see that the initial network state can be continuous (in $[0, 1]^n$) but the final network state is always discrete (in $\{0, 1\}^n$).

4 Storing words in the network

The symbols that make up words are letters at particular positions. Let $s_i = (l_i, p_i)$ denote that l_i is the letter at position p_i. Let $s_1, ..., s_n$ be all the symbols used in the collection of words to be stored. In other words for every $s_i, 1 \leq i \leq n$ there is at least one word to be stored that contains the letter l_i at position p_i in the word. Let D denote this set of symbols. The network has n units, one for each symbol s_i in D. Let m denote the length of the largest word to be stored. Then the network has m more *size* units, representing the sizes 1 through m respectively. A word is then stored as a set, containing all the symbols from D that appear in the word plus the size symbol representing the word length. Storage of this set is done according to the learning rule equations.

Example: Store words *cat, car , dot* in network.

```
D = { c1 a2 t3 r3 d1 o2 } U { 1 2 3 }
  = { c1 d1 } U { a2 o2 } U { r3 t3 } U { 1 2 3 }
```

The words are represented as the following sets, with size symbols included.

```
{ c1 a2 t3 3 }   { c1 a2 r3 3 }   { d1 o2 t3 3 }
```

Fig 1 shows the graph underlying the network after these three sets are stored. The test input can be more general weighted sets. The following are examples of the kind of sets that can be presented to the network during testing.

```
{ c1 a2 }      { 0.4-c1 0.6-d1 a2 o2 t3*1 3 }
```

The first test input contains only two symbols from which the network will retrieve { c1 a2 r3 3 } or { c1 a2 t3 3 } denoting *car* and *cat* respectively. The second test input contains confidence values on the symbols, like 0.4 on c1 and 0.6 on d1, the unit t3 clamped to 1 meaning that t3 must be present in the network final state and multiple letter hypotheses c1 and d1, a2 and o2, for the position 1 and 2 respectively. It also contains the size unit 3. In the final state, the network guarantees that there will be only one letter at one position (at most one if it is used in noise-removal-only mode) and that it will represent a word of length 3 since the input contains the size unit 3.

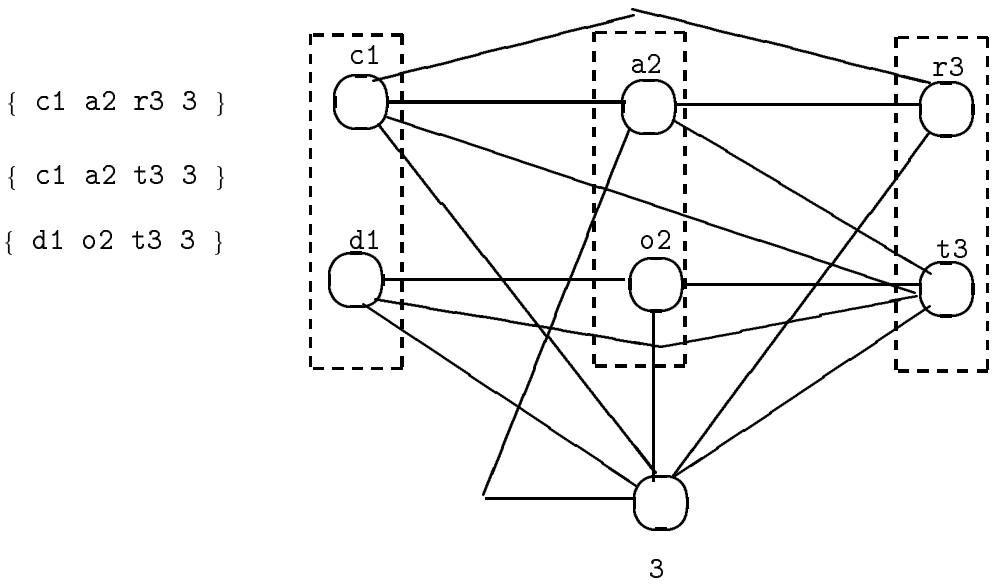

Fig 1 - Graph after storing car, cat, dot

Properties

Let us review the key network properties for the storage and retrieval of words stored in this way. These properties will form the basis for the performance of the network on the problem of machine printed word recognition. We assume that ρ, the network initial weights parameter, is $< -2n$. Lemma 1 was stated and proved in [Jag90b].

Lemma 1 *The stable states of the network correspond 1-1 with the maximal cliques in the graph underlying the network.*

From Fig 1, {c1 a2 r3 3} is a maximal clique of this graph and hence by the above Lemma it is a network stable state. In this instance we are fortunate because this network stable state is one of the stored words, exactly what we want. We will shortly see that we can guarantee a distinct network stable state for every stored word. In other words, we can guarantee that the network will not forget any stored word, no matter how many (exponential number) of words are stored. We will also see, however, that there can be spurious network stable states that don't correspond to stored words.

We briefly summarise the other important properties, which have all been formally proven in [Jag90b].

1. The network does not forget any stored word, that is for every stored word there is a distinct network stable state. A direct consequence of this is that the network has *exponential* stable storage capacity. [Thm 5, [Jag90b]] To see this, assume that only lower case words are to be stored and maximum word length is m. Then the network uses at most $26m + m$ units (26 units for all letters at one position and there are m positions), plus m size units. But the number of strings of length $\leq m$ is of the order of 26^m and the network can store any collection of such strings stably.

2. The network has the following dynamics, during testing.

(a) The network switches OFF units successively in the first $k \geq 0$ steps until the first time all active units are pair-wise mutually excitatory.

(b) The network then switches ON units successively in the next $j \geq 0$ steps until such a set is *maximal*.

[Thm 4, [Jag90b]] Also in item (a) above if a unit switches OFF then it will be a one inhibited by the largest number of active units (Lemma 7, [Jag90b]]). This is the network dynamics that we earlier refered to as noise-removal followed by completion. The network first removes noise, if any, that is first switches OFF units, if any. The network next completes what remains, if any completion is required, that is switches ON units, if any. We call this the retrieval mode of the network. We have indicated earlier that it is sometimes also advantageous to stop the network after it has removed the noise. This can be done by stopping the network just before the first time it tries to switch a unit ON which can be done by checking the value of the V_i flag of the unit p_i that will switch in the next cycle. We call this mode of the network the **noise-removal-only** mode.
Examples: Consider the network of Fig 1 storing the words *car,cat,dot*. With the left most set as input, the dynamics of the network in the two modes is shown below.

```
Mode                              Lemma 7 [Jag90b]           Thm 4 [Jag90b]
Retrieval          : { c1 d1 t3 } ----------------> { c1 t3 } -------------> {c1 a2 t3 }
Noise-Removal-Only: { c1 d1 t3 } ----------------> { c1 t3 }
```

3. Recall that m denotes the length of the largest stored word. Let I_m be the size of the largest set (input) we can present to the network during testing. We also assume, for this application, that I_m is of the order of m, a reasonable assumption. The network will converge within $2I_m$ iterations of the energy-descent rule. [Lemma 8, [Jag90b]] Furthermore, there is an efficient sequential algorithm with which the network will converge within $O(I_m n)$ steps, even on a sequential machine [Jag90b]. Since $I_m << n$, this means that even on a sequential computer the network converges in near linear time, in the number of units.

4. The network does not, in general, use a *minimum Hamming distance* criterion. [Corollary 3, [Jag90b]] Eg, stored words *plagiarize, plug*. Input {p1 g4 z9 }. Output *plagiarize*.

Spurious Memories

The main problem of storing words in the network is that spurious memories can develop and their number increases dramatically as more words are stored. A simple example illustrates a spurious memory. Consider storing three words *car,con* and *for*. As shown in Fig 2 below, the spurious memory *cor* develops. This example is from [Jag90a].

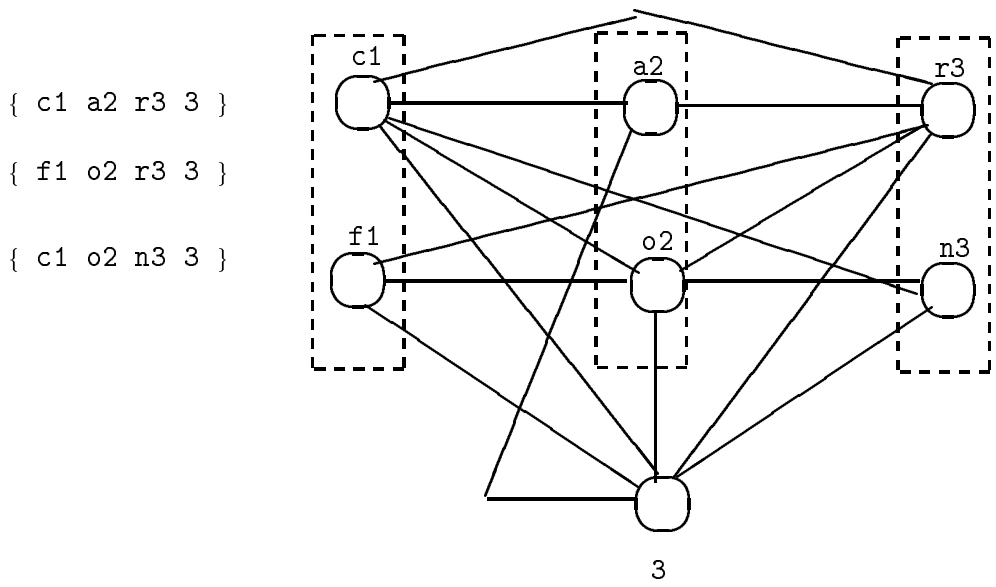

Fig 2 - Spurious memory cor after storing car,for,con

5 OCR output content and format

In this application, the OCR scanned images and produced certain kinds of output. The words in the images were machine printed, but of low quality. Fig 3 shows the kinds of images that the OCR scanned.

Fig 3 - Degraded Images of two printed words

The choice of what information to collect from word images was not under our control. Our goal was to evaluate how well the network could utilize all the information that the OCR produced, if and how the network performance varied with selective use of this information and what (selective) use of the information gave best results and what they were, in absolute terms. From our experiments, we have concluded that the network performs very well given the OCR output available. A particular advantage is the flexibility of input representation that it *naturally* allows. One of the best examples of this is the ease with which it can handle multiple hypotheses. Additionally, it can make use of confidence information the OCR provides (via numeric weights on elements in test sets), and provides a convenient mechanism to allow thresholding of decisions, by allowing units to be *clamped*.

We describe the various kinds of information produced by the OCR with an example of a *mixed* case word. In reality, the OCR produces even more information, we are describing only information that we have been able to utilise to advantage. Table 1 shows OCR output for the image of the word, *CaT*. The following information is generated. The numbers preceeding the final decisions (in interval $[0, 100]$) are the confidences of those decisions. The number on the first row, TOP-3 lower case and TOP-3 upper case are the confidences that the particular position has a lower case character or an upper case character respectively. The numbers preceeding the TOP-3 (lower or upper case) decisions are the confidences of those decisions *within* their columns. For example, assuming that the first position is lower case, the OCR gives confidence of 60 to c, 22 to o and 18 to e.

<word-length>	FINAL decision	TOP-3 lower case			TOP-3 upper case		
		40	60	40	60	40	60
3	50-C 45-a 50-I	60-c	60-a	60-h	60-C	60-O	50-I

```
                    22-o  25-e  23-t      20-O  30-D 30-T
                    18-e  15-o  17-l      20-E  10-E 20-L

Table 1 - OCR O/P for image of word 'CaT'
----------------------------------------
```

We have explored and experimentally evaluated the following selective uses of the information exemplified above. All but the use which includes the confusion matrix gave very similar results. Augmenting the OCR final decision with a confusion matrix gave the best results. Details are in the sections on experiments. One important point to note is that word length information produced by the OCR is *crucial* and always used in test input to the network (via size units which are *clamped to 1* in the input), regardless of which other OCR produced information is used. This is important because it constrains the network to retrieve only words of the same length as the (unknown) one in the input. Without such information, the network can easily retrieve words of length different than the one desired, for example if the words *catch* and *cat* were stored in the network (among others) and the input during testing was {c1 a2 t3 }, it is possible that the network will retrieve {c1 a2 t3 c4 h5 5 }, since the former is a subset of the latter. But if the input were {c1 a2 t3 3*1 }, that is the size unit was included and clamped, then provably the network will retrieve {c1 a2 t3 3 }. One consequence of this is that we can only process images for which the OCR word length estimate is accurate, in other words there are no character segmentation errors.

final-decision OCR final decisions, without confidence values. For the above example, the input would be the set { **C1,a2,I3,3*1** }. Note that the size information '*3*' generated by the OCR is explicitly included and clamped in the input via the size unit.

final-with-confidence Add confidence values to final-decision. The input would be { **0.5-C1,0.45-a2,0.5-I3,3*1** }.

TOP-3, TOP-6, TOP-3B If the stored words are all in one case (lower or upper), the TOP-3 choices of the OCR are used, for that case, for all positions. Consider the word *cat*. For each position (1-3), select the TOP-3 letters in the lower case column. Also use the confidence values that prefix them. The input would be { **0.6-c1,0.2-o1,0.2-e1,0.6-a2,0.25e2,0.15o2,0.5-i3,0.3-t3,0.2l3,3*1** }. If the stored words are mixed case then one possibility is to use TOP-6, that is both **TOP-3 Upper and TOP-3 lower case**. This means giving six hypotheses at each position in the input to the network. An alternative is to use TOP-3B standing for **TOP-3 best**. This time, at each position, the case which has the higher confidence is picked (see example from first row, Table 1). The TOP-3 choices for that case are included in the input to the network for that position.

thresholded-final-decision Threshold OCR final decisions on their confidence values. If confidence is greater than threshold, the corresponding symbol in the input to the network is clamped. Assuming a threshold of 50, in the above example the thresholded-final-decision would be { **C1*1,a2,I3,3*1** }.

final-with-confusion-matrix Augment the OCR final decisions by hypotheses derived from a confusion matrix of OCR decisions. In our case, such a matrix is obtained by statistically analysing about 15,000 OCR letter decisions. For each OCR decision (digit, lower or upper case letter), a probability of being the correct letter is assigned to each of 52 letters (lower, upper case). These probabilities are stored in a matrix. An element (x,y) of the matrix means that if the OCR decides 'x', then (x,y) is the probability that the character is really 'y'. Consider the following example confusion matrix.

```
\  a      b       c    ....
a  0.5    0.2     0.3
b  0.6    0.4     0.0
c  0.5    0.0     0.5
..

Confusion Matrix
----------------
```

If the OCR final decision for some position is 'a', the probability that it is really an 'a' is 0.5, that it is really a 'b' is 0.2 and so on. A variant of this is a boolean confusion matrix in which every entry is 0 or 1. We have also used boolean confusion matrices in our experiments. The confusion matrix is used as follows. Each letter in the OCR final-decision is replaced by the column letters of those entries in the row of the confusion matrix for that letter that exceed a threshold. The entry value is included as the confidence value of the replacement letter. The above scheme is used both for boolean and non-boolean confusion matrices. Consider an example.

```
OCR final-decisions Confusion Matrix        New Set
                    of Above Example

   { c1 a2 t3 3*1 } ---------------> { 0.5-a1 0.5-c1 0.5-a2 0.2-b2 0.3-c2 0.4-b3 0.6-a3 3*1 }
```

6 Experiments with small dictionaries

Two sets of experiments, with the network trained on two different small dictionaries, were conducted. Single-shot training sufficed, to get near peak performance [2]. To get the best performance, unlearning of spurious memories was required. This was a bit more time consuming. We used the unlearning algorithm discussed in [Jag90a, Jag90b]. For each experiment, the test set contained OCR output for images of only the words stored in the network. These images were extracted from postal addresses that were scanned by a working postal OCR at roughly 200 pixels per inch. Test input was derived from the OCR output for the appropriate set of images. Different information was used in different experiments, as described in the section on OCR content and format. Only those images were used for which the OCR estimated the word length correctly, since the OCR word length estimate is *crucial* information used in test input to the network and without which good performance is not guaranteed.

The first experiment set involved training the network on 132 upper case words. The test set was 224 images of words in the training set. Of these, OCR output for 33 images (15%) had segmentation errors. These were removed, so that the test set was reduced to OCR output for 191 images. For the test input to the network, the TOP-3 upper case choices of the OCR were used. With this input, the network retrieved the correct word for 76% of the test cases.

The second experiment set involved training the network on 441 mixed case words. The test set was 741 images of words in the training set. Of these, OCR output for 147 images (20%) had segmentation errors. These were removed, so that the test set was reduced to OCR output for 594 images. For the test input to the network, a variety of selective uses of OCR output were tried. Table 2.1 summarises the performance of the network on these different cases. It shows that for all kinds of input without a confusion matrix the results are very similar, roughly the network retrieves the correct word in 63% of the test cases. Of the 37% test cases leading to errors it shows that only about 10% of them lead to spurious memories, suggesting further analysis of the errors. To do this, we separated the errors out by word length and Table 2.2 shows the results. It shows that most errors occur in words of length 2 or 3. In retrospect this makes sense since if some part of the OCR output of a small word is in error, there is not enough information in the OCR output to suppress its effect. Table 2.3 shows a number of experiments which were then conducted to try to reduce the number of errors in words of length 2 and 3. It was found that augmenting the OCR final decisions with a confusion matrix gave the best results. The last row of Table 2.3 shows the results when the OCR final decisions were augmented with a confusion matrix for all the 594 test cases. The network retrieved the correct word for 71% of the test cases, all of the 8.5% improvement coming from words of length 2 and 3. From Table 2.1 we also see that if the OCR final decisions are interpreted as the words, only 21% of them are correct. As we've seen, after augmenting them with a confusion matrix and processing them with the network, this improves to 71% correct. Table 2.1 also contains similar statistics at the letter level. In the OCR final decisions, only 47.5% of the letter decisions are correct. After processing OCR final decisions with the network, this improves to 80.5% of the letters correct (62.5% of the words correct). These results support our earlier claims that due to the distributed representation formed by the network, its performance on correcting individual letters is significantly better than on correcting words. This ability can be exploited to use the network as a filter. We will see our exploitation of it in the experiments on large dictionaries.

[2] Even though a word was presented only once, a large frequency of presentation parameter was used to reinforce the memory. See [Jag90a] for the reason

OCR info used	% words correct	% spurious	% letters correct
Direct OCR final-decisions stats (no net)	21.0%	-	47.5%
TOP-3	62.5%	8.4%	-
thresholded-final (threshold $\geq$ 60)	63%	10.1%	-
final-decisions	62.5%	10.9%	80.5%

Table 2.1 - Tests on training set of 441 mixed case words

Word size	1	2	3	4	5	6
# words	56	93	131	80	85	54
% errors	34%	56%	56%	37.5%	25%	28%

Table 2.2 - Error distribution by Word Length

Test set size	OCR info used	% words correct	% spurious memories
233 word length 2,3	final-decisions	50%	9%
233 word length 2,3	TOP-6	53%	14.5%
233 word length 2,3	TOP-3B (best)	54%	13.3%
233 word length 2,3	final-decisions-with-boolean-confusion-matrix	67%	12.5%
233 word length 2,3	final-decisions-with-confusion-matrix (non-boolean)	70%	14%
594 (all test cases)	Same as above	71%	12.5%

Table 2.3 - More tests on words of length 2,3

7 Experiments with large dictionaries

The retrieval performance of this network on small dictionaries is quite good , immediately suggesting its application to problems for which small dictionaries suffice. But there are other problems that require large dictionaries and we know that, due to spurious memories, the network retrieval performance degrades dramatically with dictionary size. Our solution to this is to partition large dictionaries by word length and store each partition in a separate network. The number of networks can be kept small by combining two or more appropriate partitions into a single partition and storing them in a single network. The central idea behind this partitioning scheme is that it allows each individual network to store much smaller dictionaries. Moreover, only one network is involved in processing one test input, avoiding the problem of combining the output of different networks. All words of the same length are stored in only one network and this network can be identified during testing from the word length estimate that the OCR produces, for the particular test input. Given OCR output for an image, the word length information is used to find the network to send test input to. The output of this network is the output of the entire system of networks. This scheme is based on the crucial assumption that the word length information provided by the OCR is correct. But we needed to make this assumption even in the single network case and so in partitioning the dictionaries this way, we have not really made any new assumption.

The following set of experiments was conducted on a dictionary of 10,548 words of length 1-14. The dictionary was partitioned into sub-dictionaries of word length

```
1-3, 4, 5, 6, 7, 8, 9, 10, 11-14
```

respectively. Networks were trained (single-shot plus unlearning) on their respective sub-dictionaries independently and in parallel. Table 3.1 presents the training statistics. The "percentage of spurious memories" entries were estimated by testing the network on large and varied test cases and counting the number of times that the network settled at a spurious memory. The statistics show how the percentage of spurious memories increases with the ratio $\frac{\#\text{ words}}{\#\text{ units}}$. The statistics also suggest that even after partitioning, the retrieval performance is unlikely to be good, especially for words of lengths between 4 and 9. We see below (Table 3.2) that this is indeed the case.

Word Length	# words	# units	% spurious (estimated)
1-3	321	84	15%
4	795	100	50%
5	1432	126	59%
6	1919	152	60%
7	1894	175	61%
8	1699	205	57%
9	1338	224	43%
10	765	227	24%
11-14	384	274	2.6%

Table 3.1 - Training Statistics

Word Length	# test cases	Words wrong			Letters wrong		
		% bfr ('F')	% aft ('F')	% aft ('FC')	% bfr ('F')	% aft 'F'	% aft 'FC'
2-3	23	87%	78%	48%	66%	63%	34%
4	22	73%	59%	50%	48%	42%	28%
5	33	79%	67%	45%	33%	30%	16%
6	18	78%	67%	33%	32%	27%	14%
7	18	89%	56%	44%	28%	17%	7%
8	6	83%	67%	67%	37.5%	31%	21%
9	9	100%	67%	44%	41%	25%	14%
10	3	100%	100%	100%	63%	60%	27%
11	1	100%	100%	100%	82%	73%	9%
ALL	133	82.5%	67%	47%	40%	33%	18%

Table 3.2 - Network in Retrieval Mode

Word Length	Network letter Statistics			
	OCR FD		OCR FC	
	% err	% uncommitted	% err	% uncommitted
2-3	5%	62%	20%	14%
4	10%	37%	19%	10%
5	4%	30%	9.5%	8.5%
6	1%	31%	11%	4%
7	1%	27%	5%	4%
8	2%	35%	15%	6%
9	1%	40%	7%	18%
10	0%	63%	20%	13.3%
11	0%	82%	9%	9%
ALL	3%	37%	12%	8%

Table 3.3 - Network in noise-removal-only mode

The performance of this collection of networks was tested on OCR output for 133 images of words in the training set. The results on using the system of networks in *retrieval* mode are summarised in Table 3.2 and on using the system of networks in *noise-removal-only* mode in Table 3.3. 'F' and 'FC' in Table 3.2 stand for test input being OCR final-decisions and OCR final-with-confusion matrix (non-boolean) respectively. The fields called '(bef)' represent statistics on the OCR final-decisions interpreted as letter strings. The fields called '(aft)' represent statistics on the output of the network.

The last row of Table 3.2 summarises the OCR final-decision statistics and main results of retrieval mode performance. 82.5% of the OCR final-decision strings are wrong (not the correct word). 40% of the letters in the OCR final-decisions are wrong. After the system of networks is used in retrieval mode with OCR final-decisions as input, 67% of the network output strings are still wrong (not the correct word) and 33% of the letters in them are still wrong. With the network system in retrieval mode but with OCR final-with-confusion-matrix (non boolean) as input, a better 47% of the network output strings are still wrong (not the correct word) and only 18% of the letters in them are still wrong.

The last row of Table 3.3 summarises the main results of network noise-removal-only mode performance. After the system of networks is used in noise-removal-only mode with OCR final-decisions as input (40% of whose letters are in error), *only* 3% of the letters in the network output strings are wrong and 37% of the

letters are uncommitted. With the network system in noise-removal-only mode but with OCR final-with-confusion-matrix (non boolean) as input, 12% of the letters in the network output strings are wrong but only 8% of the letters are uncommitted.

Let us now analyse these results. The best performance is achieved when the network is used as a filter in the **noise-removal-only** mode. The interesting **noise-removal-only** mode result that for input in the 'FC' form more letters are wrong but much less uncomitted compared with input in the 'F' form in which much less letters are wrong but many more uncommitted offers interesting options. The decision of which input representation ('F' or 'FC') to use can be made on the basis of which gives better performance in specific situations to further postprocess the network output. Performance could depend on the capabilities of the next level processing algorithm or the specifics of the dictionary or both. If the next level processing algorithm can handle letter errors better than uncommitted letters, then 'FC' should be used. If it can handle only minimal letter errors but can handle many more uncommitted letters then the 'F' representation should be used.

Another interesting result is that in OCR final-decisions, 40% of the letters are in error and after using the network in the noise-removal-only mode on OCR final-decisions as input, the network output has 3% of letters in error and 37% of letters uncommitted. This strongly suggests that these statistics are related, in fact that of the 40% letter errors in input, the network, in noise-removal-only mode, has detected and removed 92.5% of them, leaving 3% still as errors and 37% (the ones it detected) as uncommitted. This observation is further strengthened by comparing , row by row, the '% bfr ('F')' column of Table 3.2 with the OCR FD columns of Table 3.3. In all rows the sum of the two OCR FD columns is nearly equal to the entry in the '% bfr ('F')' column. Our account of the behavior of the network in the noise-removal-only mode with OCR final-decisions as input, is stated as follows. "The network removes as many of the actual errors as it can from the input but never (or rarely) makes an error, that is removing something that is not an error". Our understanding of the network dynamics and intuition about the statistics of the stored memories also support this hypothesis. The theory to support this is beyond the scope of this paper and will be developed elsewhere.

Finally, given the excellent noise-removal performance of the noise-removal-only mode network with OCR final-decisions as input, it is surprising that on the same input the retrieval mode network performance even for letter statistics (column 7, last row Table 3.2) is so dismal. The explanation is that the noise-removal part of the retrieval mode does correct most of the errors, but due to spurious memories, the completion part of the retrieval mode introduces many new errors while switching units ON. The new errors take the letter statistics back to near the same dismal level that they were in the OCR final-decisions. By comparison, the reason why the letter statistics of retrieval mode network output on OCR final-with-confusion-matrix as input are so good is that the OCR final-decisions contain digits, a fact that we had not highlighted in our description of OCR format and content (Section 5). Our words, however, contain no digits. In other words the OCR sometimes confuses letters with digits, for example recognising an 'S' as a '5'. Our confusion matrix is not square, it has rows for digits also, whereas there are columns only for upper and lower case alphabets. Given a digit in OCR final-decisions it is replaced by the appropriate set of alphabetic characters according to the row in the confusion matrix for that digit. This replacement models how OCR confuses alphabets with digits, allowing the network to function well. When OCR final-decisions is input such digits remain in the OCR final-decisions and the network ignores them because it has no units for them. But this means that fewer letter positions (in the input set) are occupied, making the probability of reaching a spurious memory with many letter errors much higher.

8 Further postprocessing of the network output

We see that the network performs very well at filtering (removing letter errors) in **noise-removal-only** mode. However, this behavior has practical applications mainly if it can be exploited to find the correct word with better performance than without such filtering. In this section we describe a search rule that can take any string and search a specified dictionary to find matching word[s] in it. We show that applying this search rule to the **noise-removal-only** mode network output on OCR final-with-confusion-matrix input gives very good performance and much better performance than applying the search rule directly to the OCR final-decisions.

The search rule emulates the following algorithm.

```
HammingDistance = 0
```

```
MatchList = Match(string,dictionary,HammingDistance)
WHILE |MatchList| = 0 DO
 INC(HammingDistance);
 MatchList = Match(string,dictionary,HammingDistance)
END;
```

where 'Match' takes a parameter 'string' which is a string of characters including '.' and returns all the words in the dictionary that are within the Hamming Distance of it specified by the 'HammingDistance' parameter. In our case, Hamming Distance between the string parameter s and a dictionary word w is defined as the number of positions in s, excluding those that contain the character '.' that don't match the character at the same position in w. In other words, the '.' in s matches any character at the same position in w. The above algorithm basically says that we start by looking for matches with Hamming distance 0 and we keep incrementing the Hamming distance by 1 until the first time that the list of Matching words from the dictionary is non-empty.

PostProcessing Level	Word Length	Avg MatchList Size	% of time correct word in MatchList	Avg # of Tests (= # Search Rule Iterations * Sub-dictionary Size)
OCR	2-3	66.5	65%	2.6*321
final-decisions	4	17.0	73%	2.5*795
	5	51.0	94%	2.6*1432
	6	20.7	94%	2.8*1919
	7	2.15	100%	2.95*1894
	8	2.3	100%	4.0*1699
	9	1.3	89%	4.3*1338
	10	-	0%	-
	11	-	0%	-
	ALL	30	83.5%	3687
noise-removal-only	2-3	86	87%	1*321
network mode	4	17.4	73%	1*795
output on OCR	5	50.5	91%	1*1432
final-decisions	6	20.7	94%	1*1919
	7	2.16	100%	1*1894
	8	2.3	100%	1*1699
	9	1.3	89%	1*1338
	10	3.6	100%	1*765
	11	3.0	100%	1*384
	ALL	33.8	89.5%	1245
noise-removal-only	2-3	3.5	61%	1.1*321
network mode	4	3.1	77%	1.3*795
output on OCR	5	2.0	91%	1.3*1432
final-with-	6	2.9	89%	1.5*1919
confusion-matrix	7	1.2	100%	1.3*1894
	8	1.0	62.5%	2.0*1699
	9	1.0	89%	1.5*1338
	10	1.0	100%	3*765
	11	1.0	100%	2*384
	ALL	2.6	84%	1763

Table 4.1 - Search Rule Statistics

Word Length	2-3	4	5	6	7	8	9	10	ALL	
% of test cases $\|MatchList\| = 1$	61%	59%	82%	89%	89%	100%	100%	100%	100%	79%
% of test cases $\|MatchList\| = 1$ and word is correct	43.5%	50%	79%	78%	89%	83%	89%	100%	100%	70.5%

Table 4.2 - Exact Decision Statistics Of Search Rule

On noise-removal-only mode network output on OCR final-with-confusion-matrix input of the same 133 test cases Table 4.1 presents the results on using the Search Rule on strings at different levels (OCR final-decisions, Network output) for the 133 test cases corresponding to the experiments reported in Tables 3.2,3.3. Given the string to match, the Search Rule only searches the sub-dictionary of words of the same length as the string, making the search process quite efficient. The statistics of the OCR final-decisions are in Table 3.2. From Table 4.1 we see that when the search rule is applied to the OCR final-decisions or to the noise-removal-only mode network output on OCR final-decisions as input, an average of about 30 matching words are retrieved for every string as the search key. This statistic alone makes these results relatively impractical. The best performance is obtained by applying the search rule to the noise-removal-only mode network output on OCR final-with-confusion-matrix as input, where an average of only 2.6 matching words are retrieved for every string as the search key and 84% of the time the correct word is in the list of matching words retrieved. The more detailed analysis of this case in Table 4.2 shows even better results, that for 79% of the search keys the network retrieves exactly one matching word and for 70.5% of the search keys the network retrieves exactly one matching word which is the correct word. So for exact match, this scheme still gives a very good performance of 70.5%. Relative to our test set, the Prob(word is correct| $|MatchList| = 1$) is 0.89. This suggests that the output of the search rule on noise-removal-only network output on OCR final-with-confusion-matrix input can be further processed by the following decision rule.

```
If |MatchList| = 1 THEN Accept word as correct
ELSE Reject
```

which on our test set would give 70.5% correct, 8.5% error and 21% reject performance on the requirement of finding the correct word exactly.

We have seen that the functional performance on applying the search rule to noise-removal-only network output with OCR final-with-confusion-matrix as input is excellent. Equally important is the time efficiency of this scheme. We think that no other scheme can match the functional performance of our scheme and have comparable time efficiency at the same time. Our reason is that in order to achieve the best functional performance, augmenting the OCR output with the confusion matrix seems necessary. Augmenting the OCR output with the confusion matrix produces a set of multiple letter hypotheses at each position. Search keys for distance-based search rules are usually only strings, that is a set of zero or one letter hypotheses at each position. Our network seems the most efficient scheme to convert the set of multiple hypotheses to a string, retaining the most relevant information correctly. It is efficient because the network takes only as many network cycles as the number of extraneous hypotheses it removes from one set. This will typically be between 5 to 20 cycles.

9 Network special-case for noise-removal-only mode

There is another important implication of the noise-removal-only mode. If only this mode is to be used, then a special case of the network called the *two-state weights* form is expected to perform almost as well, specifically give very similar results to the noise-removal-only results we've seen here, for word recognition. This form is a simpler variant of the Hopfield-style network and has been studied earlier [Jag90d, Jag90c] for a number of reasons. The simplest such reason is that the weights in the network in this form are only two-valued ($\{\rho, 1\}$), suggesting that analog/optical implementations could benefit from these extremely low precision requirements. Even with such low precision weights, the network surprisingly, can solve many problems. We would like to add "using it in noise-removal-only mode for word recognition" to this list and for now we will do it at least in an "expected" sense. The learning rule for the two-state weights form can be obtained from the general learning rule equations (2,3) by setting $\lambda = 2$. More explicitly, the learning

rule for the two-state weights form replaces equations (2,3) by

$$w_{ij}(t+1) \;:=\; \begin{cases} 1 & \text{if } s_i,\ s_j \in M_s \text{ and } w_{ij}(t) = \rho \\ w_{ij}(t) & \text{otherwise} \end{cases} \tag{7}$$

$$w_{ii}(t+1) \;:=\; \begin{cases} 1 & \text{if } s_i \in M_s \\ w_{ii}(t) & \text{otherwise} \end{cases} \tag{8}$$

Everything else remains the same.

Let us now see why, in the noise-removal-only mode, the two-state weights form is expected to perform as well as the general form. As discussed in Section 3 (properties) and by Lemma 7 [Jag90b] if a unit switches OFF then it will be a one inhibited by the largest number of active units. If the unit meeting this condition is unique, then only the negative (of value ρ) weights in the network affect this decision, the actual values of the positive weights (in $(0, 1]$) are unimportant and they might as well be equal to 1, giving us the two-state weights form. The only time that the positive weights affect a switch OFF decision is when the unit inhibited by the largest number of active units is not unique. We think that in word recognition, the case where the letter hypothesis inhibited by the maximum number of active letter hypotheses *is unique*, is the important case to recognise. The two-state weights form network does that, by switching OFF this letter hypothesis. We think that when such a hypothesis is not unique the exact choice of which of these should be switched OFF is relatively unimportant, that is, contributes significantly less to overall noise-removal-only mode performance. We suggest empirical validation of our hypothesis as further work.

A direct consequence of the arguments in this section is that in the noise-removal-only mode, if the general network form is used (as we have used), no unlearning need be done. The network is expected to do almost as well if the set of words is trained only once (single-shot). This could save considerable time spent otherwise on unlearning, especially when the number of words to be stored is large.

Experiments with the two-state weights network form

This entire section is new, not in the manuscript submitted to IJPRAI for consideration. It tests the hypothesis that in the noise-removal-only mode, the *two-state weights* form performs as well as the general network form. The experimental conditions are exactly the same as for the experiments reported in Table 3.3. Table 5.1 shows the results when using the *two-state weights* network form. The reader can compare it with Table 3.3 (general network form) and see that the results are almost identical. Any number in Table 5.1 that is different by more than 1% from the number in the same position in Table 3.3 is marked with a '*'. The ALL results for both kinds of input are almost the same in Table 3.3 and Table 5.1 (none are marked with a '*').

Word Length	Network letter Statistics			
	OCR FD		OCR FC	
	% err	% uncommitted	% err	% uncommitted
2-3	5%	62%	18%	14%
4	10%	37%	19%	10%
5	5%	30%	12%*	10%*
6	1%	31%	9%*	3%
7	1%	27%	5%	4%
8	2%	35%	12.5%*	6%
9	2%	39.5%	7%	17%
10	0%	63%	13%*	13%
11	0%	82%	9%	9%
ALL	3%	37%	11%	8%

Table 5.1 - Two-state weights form networks in noise-removal-only mode

In order to provide a visual feel for the information processing behavior of the network, the following example shows strings related to experiments on test images on words of length 9. Each column corresponds to a specific 9 letter word image used during testing (Table 5.1, row 7). In each column, the string in the 1st row is the correct word, the string in the 2nd row is the OCR final-decisions, the string in the 3rd row is the *two-state weights* network form (noise-removal-only mode) output with OCR final-decisions as input (statistics in Table 5.1, row 7), the string in the 4th row the same but with OCR final-with-confusion-matrix as network input (statistics in Table 5.1, row 7). In rows 3 and 4, a '_' indicates that the network output had no letter at the corresponding position (uncommitted).

```
Shoreview Millstone Windomere Westgrove Lafayette Flagstone Montclair Brookdale Westgrove
shorevieW Mi11St0ne W1ndo0eCe WestgroVe r7f7v7tE. 8l9.9Cone Montcla1r 8t00hd3f2 We5tgroVe
_horevie_ Mi___t_ne W_ndo_e_e Westgro_e __f___t__ _l____one Montcla_r _t__h____ We_tgro_e
Shoreview Millstone Windo_eee Westgrove __f___t__ Ble_econe Montclair Bro_hda_e Westgrove
```

Storing a large dictionary in a single network

This subsection is also new, not in the manuscript submitted to IJPRAI. Since the noise-removal-only mode gives the best performance and also works very well in the two-state weights form, we decided to see how well this mode, in the two-state weights form, would work when the entire dictionary was stored in the same network. An important consideration was also that using only the noise-removal-only mode was expected to make unlearning unnecessary during training, simplifying this particular experiment considerably and showing that large dictionaries can be efficiently trained. We stored 10,548 words in a single network. The network used 321 units. We then tested the same 133 images, with the two-state weights form network in noise-removal-only mode, with network input being OCR final-decisions and OCR final-with-confusion-matrix respectively (as before). The total real time (watch measured) for training (storing 10,548 words) and testing on both test sets (for 133 images) was only 2 minutes, on a SUN-4. Table 6.1 summarises the results, showing that the noise-removal-only performance of a single two-state weights form network is almost as good as the performance of the partitioned general form networks. Again, deviations (from Table 3.3) of more than 1% are marked with a '*'.

Word Length	Network letter Statistics			
	OCR FD		OCR FC	
	% err	% uncommitted	% err	% uncommitted
ALL	4%	36%	16%*	6%*

Table 6.1 - Single Two-state weights form network in noise-removal-only mode

10 Conclusions

We have shown with empirical tests that our network works very well (71% correct) for recognition of printed but degraded words on small dictionaries, of upto 500 words. In such situations its main practical advantage over conventional methods is its speed. It can process sets that include multiple hypotheses within 5-30 network cycles, a performance we think is unlikely to be matched by other means. We have also shown that even though the network performance degrades drastically for exact retrieval as larger dictionaries are stored, it can still be used to rapidly postprocess sets of multiple hypotheses, relaxing competing hypotheses, with very good performance. Indeed the best recognition results (2.6 candidate words and 84% of the time the correct word is in this list) are obtained when the OCR final-decisions are first augmented with a confusion matrix to produce multiple competing hypotheses, the set of these hypotheses is then postprocessed with the network, and the network output is then used as a key to search the dictionary.

Author's Note : This is new - Although partitioning a large dictionary and storing in multiple networks gives the best performance, storing entire dictionary in a single two-state weights form network gives almost as good performance. The latter is clearly much simpler and cost-effective than the former.

End of new

Since the use of the confusion matrix seems so crucial to obtaining good performance, the only scope for improvement would be to replace the network by some other method for postprocessing the set of multiple hypotheses or directly recognising the word from it. We think that it is extremely unlikely that even if some other method can demonstrably achieve the functional performance we have achieved (we are not optimistic of this even), that it can also match the processing speed of the network, 5-30 network cycles to postprocess one multiple hypothesis set. This speed could become a practical reality if the network were realised in parallel (digital) hardware to exploit its inherent parallelism. In [Jag90b], we present an algorithm for the network dynamics which can be realised on vector machines and exploits this parallelism. Even on a sequential machine the network postprocesses its input in real-time.

The above discussion suggests that the network is really suited to applications requiring recognition in real-time.

Finally, the results and techniques of this paper also have more general implications.

1. All the techniques and empirical results of this paper are likely to carry over to any application concerned with storing (small or large) dictionaries of strings of length 5-15 on an alphabet of 20-30 symbols and operating the network in retrieval or noise-removal-only mode. Two applications that fit these requirements are "spelling correction of stored words" and "content-addressable memories on phoneme sequences" (speech recognition).

2. More generally, all the techniques and methods of this paper are likely to carry over to any application concerned with storing and retrieving finite length sequences over any alphabet. Absolute results may differ depending on the lengths of the sequences and the size of the alphabet. The techniques of this paper can also be viewed as a new algorithm for string matching, exact or approximate.

11 Acknowledgements

We would like to thank Professor Srihari and his Laboratory for the excellent environment in which this application was born and where the experiments were conducted, especially Tin-Kam Ho for providing and setting up the dictionaries and test databases, and for suggesting the use of the confusion matrix and providing the matrix, generated earlier at considerable effort. We would also like to thank Jonathan Hull for his support of the experiments and for granting us permission to use the dictionaries, real image data and real OCR output.

References

[BT80] Bouchard D.C., and Toussaint, G.T. (1980). Heuristic search methods for efficient use of dictionary information in text recognition. *TR SOCS 80.5, School of Computer Science, McGill Univ.*

[HHS89] Ho Tin-Kam, Hull Jonathan J., and Srihari Sargur N. (1989). Recognition of Degraded Text Images. *Pattern Recognition Systems and Applications Conference of the* 10^{th} *International Conference on Pattern Recognition.*

[Hop82] Hopfield, J.J. (1982). Neural networks and physical systems with emergent computational properties. *Proceedings of the National Academy of Sciences, USA, 79, 2554-2558.*

[Jag90a] Arun Jagota (1990). A Neural Lexicon in a Hopfield-style network. *Proceedings of the International Joint Conference On Neural Networks, January 1990, Washington-DC*

[Jag90b] Arun Jagota (1990). A new Hopfield-style network for content-addressable memories. *TR 90-02, Dept Of Computer Science, State University Of New York At Buffalo, NY 14260*

[Jag90c] Arun Jagota (1990). What can be done with about 1/4 million words stored in a Hopfield-style network? *In Review, Copies available from author.*

[Jag90d] Arun Jagota (1990). The Hopfield-style Network as a Maximal Clique Graph Machine. *TR 90-25, Dept Of Computer Science, State University Of New York At Buffalo, NY 14260*

[Pet80] Peterson J.L. (1980). Computer programs for detecting and correcting spelling errors. *Communications of the ACM 23,12 (Dec 1980), 676-687.*

[ST79] Shingal R., and Toussaint G.T. (1979). A bottom-up and top-down approach to using context in text recognition. *Int. J. Man-Mach. Studies. 11 (1979), 201-212.*

[SB82] Srihari Sargur N., and Bozinovic R. (1982) A string correction algorithm for cursive script recognition. *Proc. 6th Int. Conf. on Pattern Recognition (Munich, W. Germany, Oct 19-22), IEEE, New York.*

[SHC83] Srihari Sargur N., Hull Jonathan J., and Choudhari Ramesh (1983). Integrating Diverse Knowledge Sources in Text Recognition. *ACM Transactions on Office Information Systems, Vol. 1, No. 1, January 1983, Pages 68-87.*