A Comparison of a Novel Neural Spell Checker and Standard Spell Checking Algorithms

Victoria J. Hodge
Dept. of Computer Science,
University of York, UK
vicky@cs.york.ac.uk

Jim Austin
Dept. of Computer Science,
University of York, UK
austin@cs.york.ac.uk

Abstract

In this paper we propose a simple and flexible spell checker using efficient associative matching in the AURA modular neural system. Our approach aims to provide a pre-processor for an Information Retrieval (IR) system allowing the user's query to be checked against a lexicon and any spelling errors corrected, to prevent wasted searching. IR searching is computationally intensive so if we can prevent futile searches we can minimise computational cost. We evaluate our approach against several commonly used spell checking techniques for memory-use, retrieval speed and recall accuracy. The proposed methodology has low memory use, high speed for word presence checking, reasonable speed for spell checking and a high recall rate.

Keywords: Binary Neural Spell Checker, Associative Matching, Supervised Learning, Accuracy, Memory Usage.

1 Introduction

Spelling errors are abundant in the queries generated by the users and can easily defeat search and retrieval operations in IR systems if not detected. An approximate string matching algorithm is required to detect errors in users' queries and using some measure of similarity, recommend words to the user which are most similar to each mis-spelt query word. This error checking would prevent futile searching for mis-spelt words which wastes both computational processing and user time and would make the system more robust to user errors.

We describe a spell checking method that allows a presence check of a query word against the stored lexicon, identifies any spelling errors in queries and suggests alternative spellings. The system is built in the AURA modular neural network architecture [1]. We are producing an Information Retrieval application on the architecture and have constructed a spell checker as a front-end processor to the developing IR system, designed to integrate with the architecture of the system. The spell checker uses an integrated hybrid approach to overcome the four main forms of spelling errors: insertion, deletion, substitution and transposition (double substitution). We use an n-gram approach to overcome the first two errors and integrate a Hamming Distance approach to overcome substitution and transposition errors. N-gram approaches match small character subsets. The words are divided into character subsets and the subsets matched. N-gram approaches are able to accommodate letter omissions or insertions. Hamming Distance matches query words against lexicon words by left aligning the words and matching letter position by corresponding letter position. Hamming Distance does not work well for insertion and deletion errors as the letters of the query word do not align with the letters in the correct spelling. However, Hamming Distance matching works well for transposition and substitutions where most characters are still aligned. Our hybrid system

can exploit the best match from either Hamming Distance or n-gram and thus overcome all four error types.

In our IR system, the user supplies a simple query, a list of the required search words. Each word is matched against a list of all terms present in the text corpus (the lexicon) by the spell checker. If the word is present the user can either elect to just query on that term or search for similar terms using the spell checker where the terms have the same word stem as the query term but different suffixes, e.g. {engine, engines} or {search, searches, searched}. If the word is not present the system assumes a spelling error and the spell checker suggests a list of alternative spellings from which the user can select any number to use in the query.

Our string matching approach is simple and flexible. We assume the query words and lexicon words comprise sequences of characters from a finite alphabet of 30 characters (a to z and four punctuation characters). The approach translates words to binary bit vectors by mapping characters onto specific bit positions. The lexicon is represented by a storage-efficient binary matrix that stores all bit vectors. The bit vector approach is not language-specific so may be used on other languages or for example on DNA sequences. Our spell checker aims to be memory efficient and produce swift retrieval - although memory efficiency and the ability to dovetail with the IR system being produced are more important than pure speed. It aims to have high recall[1] possibly at the expense of precision[2]. In this paper, we just return sets of matching words with no ordering. Where the retrieved word sets are large we can in future implement a scoring system to rank and filter the retrieved sets to improve precision.

[1] The percentage of correct spellings retrieved.

[2] The percentage of words other than the correct spelling retrieved.

Some alternative approaches include the Levenshtein Edit distance [2], agrep [3] [4], pure Hamming Distance (see section 1.4.2) and pure n-gram (see section 1.4.3). We evaluate our approach against these alternatives. The reader is referred to Kukich [2] for a thorough exposition of spell checking techniques. We compare our method with the other methods for speed of retrieval. We compare our system for memory use with pure n-gram and Levenshtein Edit Distance where the words are stored in an array of strings (agrep does not store the words in memory but rather reads them in each query). Finally, we compare with all alternatives and the MS Word 97 spell checker for quality of retrieval - the percentage of correct words retrieved from 40 mis-spelt words giving a figure for the recall accuracy with noisy inputs (mis-spellings).

1.1 Levenshtein edit distance

Levenshtein edit distance [2] generates a score for the similarity between the query word and each lexicon word in turn. The score is the number of single character insertions, deletions or substitutions required to transform the query word to the lexicon word, for example, to transform *him* to *ham* requires one substitution or to transform *ham* to *harm* requires one insertion. The lexicon word with the lowest score is deemed the best match. A function $f(0,0)$ is set to 0 as in equation 1 for all comparisons and the function $f(i,j)$ is calculated, iteratively counting the string difference between the query $q_1 q_2 ... q_i$ and the lexicon word $l_1 l_2 ... l_j$. Each insertion, deletion or substitution is awarded a score of one see equation 2.

$$f(0,0) = 0 \tag{1}$$

$$f(i,j) = min[(f(i-1,j)+1, f(i,j-1)+1, f(i-1,j-1)+d(q_i,l_j)] \\ \text{where } d(q_i,l_j) = 0 \text{ if } q_i = l_j \text{ else } d(q_i,l_j) = 1 \tag{2}$$

Edit distance is $O(mn)$ for retrieval as it performs a brute force comparison with all n character of all m words in the lexicon and therefore can be slow for

a large lexicon.

1.2 Agrep

Agrep [3] [4] is based upon Edit Distance and finds the best match, the word with the minimum single character insertions, deletions and substitutions. Agrep uses several different algorithms for optimal performance with different search criteria. For simple patterns with errors, agrep uses the Bayes-Moore algorithm with a partition scheme (see [3] for details). Agrep essentially uses arrays of binary vectors and pattern matching, comparing each character of the query word in order, to determine the best matching lexicon word. The binary vector asks as a mask so only characters where the mask bit is set are compared, minimising the computation required. There is one array for each error number for each word, so for k errors there are $k+1$ arrays ($R^0 ... R^k$) for each word. R_j denotes step j in the matching process of each word and R_{j+1} the next step. RShift is a logical right shift, AND and OR denote logical AND and OR respectively and S_c is a bit vector representing the character being compared c. The following two equations describe the matching process for up to k errors $0 < d \leq k$.

$$R_0^d = 11...100...000 \text{ with d bits set} \quad (3)$$

$$R_{j+1}^d = \text{Rshift}[R_j^d] \text{ AND } S_c \text{ OR Rshift}[R_j^{d-1}] \text{ OR Rshift}[R_{j+1}^{d-1}] \text{OR} R_j^{d-1} \quad (4)$$

For a search with up to k errors permitted there are $k+1$ arrays and there are 2 shifts, 1 AND and 3 OR operations for each character comparison so the running time is $O((k+1)n)$ for an n word lexicon (see [3]).

1.3 AURA

AURA [1], [5] is a modular binary neural network architecture that uses Correlation Matrix Memories (CMMs) [1]. AURA uses a supervised learning rule,

similar to a hash function, to map inputs to outputs (see figure 1). AURA does not suffer from the lengthy training problem of other neural networks; training and retrieval are one-pass processes producing the network's high speed [6]. Storage is efficient in the CMMs as the matrix size is defined when the CMM is instantiated so new inputs do not require additional memory allocation as they are overlaid with existing trained patterns. AURA is able to partially match inputs which is essential for a spell checking system as the query word will partially match the lexicon words.

1.4 Our methodology

The word spellings form the inputs and the matching words from the lexicon form the outputs of the system. The spellings and words are translated to their respective binary bit vectors by the data-to-binary lexical token converter (see figure 1). For the inputs, we divide a binary bit vector of length 960 into a series of 30-bit chunks. Each chunk represents a letter of the alphabet or one of four punctuation characters so words of up to 30 characters may be represented. (We need two additional character chunks for the shifting n-gram described later).

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	-	'	&	/
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

(5)

Each word is divided into its constituent characters. The appropriate bit is set in the chunk to represent each character in order of occurrence (see equation 5 for the index of characters against bit set). The chunks are concatenated to produce a binary bit vector to represent the spelling of the word and form the input to the CMM. Any unused chunks in the bit vector are set to all zero bits.

Each word in the lexicon has a unique orthogonal binary bit vector to represent it and form the output from the CMM. A single bit is set corresponding

to the position of the word in the alphabetical list of all words (see equation 6).

$$bitVector^p = p^{th}\text{bit set } \forall p \text{ for p = position\{words\}} \tag{6}$$

The CMM represents the lexicon of all words in the corpus, the inputs are the spellings and the outputs the matching words.

1.4.1 Training the network

The binary patterns representing the word spellings are input to the CMM and the binary patterns for the matching words form the outputs for the CMM. Essentially, we associate the spelling of the word to an orthogonal output vector to uniquely identify it. Figure 2 shows a CMM after 1, 2 and 3 patterns have been trained. The binary input vectors (word spellings) are 01001000, 00100100 and 00101000 and their respective output vectors (unique identifiers to retrieve the matching words) are 01000000, 00010000 and 00100000. The CMM is set to one where an input row (spelling bit) and an output column (word bit) are both set (see figure 2). After storing all spelling-word associations, the CMM weight w_{kj} for row j column k (where $\vee$ and $\wedge$ are logic 'or' and 'and' respectively and $inputSpelling$ and $outputWord$ are the binary input and output vectors respectively) is given by equation 7:

$$w_{kj} = \bigvee^{all\ \mathrm{i}} inputSpelling_j^i \wedge outputWord_k^i = \left[\sum^{all\ \mathrm{i}} inputSpelling_j^i \wedge outputWord_k^i\right] \tag{7}$$

1.4.2 Recalling from the network - binary Hamming Distance

For binary Hamming Distance, we use the CMM to retrieve the lexicon words that match the letters of the input spelling where 'match' implies the same letter in the same position in the word. We use the CMM to count the aligned letters. CMMs produce perfect recall - they retrieve ALL expected matches [5]. Only the spelling pattern is applied to the network during recall and the matching

words are retrieved in an output vector. The columns are summed to produce an output activation vector, see equation 8.

$$output_j = \sum^{alli} input_i \wedge w_{ji} \tag{8}$$

The output activation vector is thresholded generating a binary output vector (see figure 3). The binary output vector represents the words trained into the network matching the input spelling presented to the network. We use the Willshaw threshold set to the number of bits in the input vector to threshold the output activation vector (see figure 3). Willshaw threshold sets to 1 all the positions in the binary output vector where the corresponding position of the activation vector is greater than or equal to a predefined threshold value. The remaining bits are set to 0. For an exact match, we wish to retrieve all outputs that match all characters in the input. If the input has three bits set we retrieve all columns that sum to three (see figure 4).

If only a partial match of the input spelling is required, i.e., only M of the N letters ($M < N$) in the input must match exactly then this combinatorial problem is easily resolved. The input is sent to the network and the Willshaw threshold is set at M. This partial match provides a very efficient, single-step mechanism for selecting those words that best match. We threshold the output vector at the highest match value to retrieve ALL best matching lexicon words. For example, in figure 4 to match only two letters of the input word, we threshold at two and retrieve {'therefore', 'the', 'she', 'three'}.

We are able to use the '?' convention from UNIX for unknown characters by setting all bits in the chunk representing a universal OR, i.e., the chunk represents a 'don't care' during matching and will match any letter or punctuation character for the particular letter slot. For example, if the user is unsure whether the correct spelling is 'separate' or 'seperate' they may input 'sep?rate'

to the system and the correct match will be retrieved as the chunk with all bits set will match 'a' in the lexicon entry 'separate'.

1.4.3 Recalling from the network - Shifting n-grams

We utilise three n-gram approaches [7] and select the most appropriate for the query word, unigram for spellings with less than four characters, bigrams for four to six characters and trigrams for spellings with more than six characters. Mis-spelt words with less than four characters are unlikely to have any bigrams or trigrams found in the correct spelling, for example 'teh' for 'the' have neither common. Spellings with four to six characters may have no common trigrams but should have common bigrams and words with more than six characters should match trigrams. We describe a recall for a seven-letter word ('theatre') using trigrams below and in figure 5. All n-gram techniques used operate on the same principal, we just adjust the size of the comparison window.

For the shifting n-grams, we use the CMM to count the matching n-grams. We take the first three characters of 'theatre', i.e. 'the' and input these left-aligned to the CMM, see figure 5. We threshold the output activation vector at three to find any words matching all three characters of the trigram. We then shift the trigram vector one 30-bit chunk to the right, input to the CMM and threshold at three to find any words matching all three characters of the trigram but shifted one character right as in the second CMM of figure 5. We continue sliding the trigram to the right until the first letter of the trigram is in the position of the last character of the spelling and the third letter aligned with the input plus two characters (as in figure 5) as nearly all spelling mistakes are within two characters of the correct spelling [2]. We logically OR the output vector from each trigram position to produce an output vector denoting any word that has matched any of the trigram positions for this trigram see figure 5. We then move onto the second trigram 'hea', left align, input to the CMM,

threshold and slide to the right producing a second trigram vector of words that match this particular trigram in any place relative to the length of the query word. When we have matched all m trigrams {'the', 'hea', 'eat', 'atr', 'tre'} from the spelling, we will have m output vectors representing the words that have matched each trigram respectively. We sum these output vectors to produce an integer vector representing a count of the number of trigrams matched for each word. We can then threshold at the maximum value of the integer vector to find the best matching words. The thresholded output vector is passed to the lexical token converter binary-to-data to retrieve the matching words as for Hamming Distance.

1.4.4 Superimposed outputs

Partial matching with Hamming Distance and shifting n-grams generates multiple word vector matches superimposed in a single output vector after thresholding (see figure 4). These outputs must be identified. A list of all valid output vectors is held in a content-addressable memory and matched in the lexical token converter - binary to data see figure 1. The thresholded output vector is passed to the lexical token converter which separates the bit vector into separate orthogonal bit vectors and retrieves the word associated with each separate orthogonal vector. The time for this process is proportional to the number of bits set in the output vector Θ(bits set), there will be one matching word in the lexical token converter per bit set for orthogonal output vectors.

1.5 Hybrid

For exact matching (checking whether a word is present in a lexicon) we use the Hamming Distance and a length match. If the word is not present in the lexicon we can then spell check using the word and produce a list of alternative spellings preventing a futile search for a mis-spelt word. For exact match, we perform the

Hamming Distance match described in section 1.4.2, thresholding at the length of the input spelling (i.e., number of bits set in the input vector) to retrieve all words beginning with the input spelling. The input spelling forms a word stem to the words retrieved from the Hamming Distance match. In figure 4, the input would be thresholded at three (length of 'the') to retrieve {'the', 'therefore'}. We then input a bit vector with all bits set to one and threshold the output at the exact length of the input spelling (number of bits set) to count the number of characters in each stored word. There is one bit per character so if all bits are activated and summed we effectively count the length of the word. From figure 4, all bits are set in the input and the output thresholded at exactly three to retrieve the three letter words {'the', 'are', 'she'}. This negates the additional storage requirement of storing the word length with each lexicon word and then searching the word length array for appropriate length words. Our exact match is rapid as we demonstrate later so we are not penalised, the approach is fast and imposes no storage overheads. We can then logically AND the Hamming Distance output vector with the length vector to retrieve the exact match if one exists, i.e., matching all input characters AND the exact length of the input. The ANDed bit vector is passed to the lexical token converter to retrieve the matching word {'the'}.

For the best match, we produce a union of the Hamming Distance best match and the shifting n-gram best match, any word that is the best match for either. We threshold the output of the Hamming Distance at the maximum value to retrieve the set of lexicon words matching the maximum number of input spelling letters. We threshold the output of the shifting n-gram at the highest value to retrieve the lexicon words matching the maximum number of input n-grams. We logically OR the two thresholded outputs to produce the union and pass this vector to the lexical token converter to retrieve the matching words.

1.6 Recalling from the network - N-grams

We also evaluate an alternative n-gram approach using only trigrams and constructed in AURA which builds upon Cherkassky [8] and Ullman [7]. A 3-dimensional representation of all possible trigrams is produced with the x-dimension representing the first letter, y the second and z the third. The characters are represented by 30 bit chunks as with our hybrid approach. The 3-D representation is then mapped onto a single dimension vector of length 30x30x30 with one bit position for each possible triple (see figure 6). Each word to be trained is subdivided into its constituent trigrams and the appropriate bits are set in the 1-D vector. The output vector is again a single-bit set (orthogonal) binary vector. The approach is fundamentally similar to the n-gram approach described in section 1.4.3, but there is no length limit imposed while matching so we could match characters 28, 29, and 30 for a three letter input word. Our hybrid approach limits matching to within two characters of the length of the input. The trigram approach does not work well for less than 5 character words as there may be no common triples between the query and the correct spelling.

2 Evaluation

For all evaluations we use the lexicon from the UNIX spell program comprising 23791 words and 194784 characters in total.

2.1 Memory use

We compare the memory usage of the trained CMM and word decoder (lexical token converter) for our hybrid methodology where each 30-bit chunk uniquely identifies each character in the spelling and for the trigram approach where all possible trigrams are mapped onto a 1-D vector and the appropriate bits are set for the trigrams present in each spelling.

From table 9, our hybrid approach uses approximately one third of the memory used for the triple mapping approach. Triple mapping requires vectors of length 2700. Our approach requires vectors of length (30 bits per character * maximum word length to be stored). Here the maximum length word in the lexicon is twenty-nine characters so we elected to use 960 length vectors (30 characters + 2 extra for the shifting n-gram). The matrix for the triple mapping is very sparse as many of the triple possibilities never occur in the English language, for example 'xyz', 'aaa', 'zzz' and storage space is wasted. In a standard matrix structure many empty rows would be stored, although in the CMM approach the empty rows are represented by null pointers so there is no wasted space. The memory usage of the Edit distance word array and the hybrid CMM are approximately equivalent. If we integrated Edit distance into the IR system we are building, a word decoder would be required to map the best matching word identified by the Edit Distance to the bit vector which forms the input vector for the next stage of the IR system. Therefore, the memory use for Edit distance and our hybrid approach would approximately be equivalent.

2.2 Training Time

The training time was the time to produce the bit vectors for each word spelling, associate each word with an orthogonal output vector and train the spelling-word association into the memory matrix. Our hybrid technique processes the 194784 characters of the UNIX 'spell' dictionary in 3.3 seconds. Therefore the approach may process 59025 characters per second for training. Training is only performed once and is a single epoch process.

2.3 Retrieval time

We compare the retrieval times for an exact match search and a best match search for each of the approaches described in the paper. For comparison we use a short word 'the' comprising one trigram and a very long word that we added to the lexicon for evaluation purposes 'floccinaucinihilipilification' with 27 trigrams. The length variation enables us to compare the running time for each algorithm and see whether they are closer to $O(n)$ or $O(1)$, i.e., dependent on the length of the input or independent and approaching constant time for retrieval.

2.3.1 Exact match

Our methodology implements exact match through a Hamming Distance and length comparison as this is faster than the shifting n-gram approach from empirical comparison ($O(1)$ compared to $O(n)$). The triple mapping approach retrieves the words matching all triples in a thresholded output vector and retrieves the words of equivalent length to the input in a second thresholded output vector. Similar to our approach, it forms a logical AND of the two output vectors to retrieve the word that have all triples in common AND are equivalent length to the input, i.e., the exact match.

See table 9 for the exact match retrieval times. Our hybrid approach, the triple mapping and agrep are all $\Theta(1)$ as the retrieval is independent of the length of the input, only Levenshtein is $O(inputLength)$. For our hybrid, retrieval time depends only on the number of matches for Hamming Distance match and length match which must be retrieved from the lexical token converter. Our hybrid matches against the 194784 characters of the UNIX 'spell' lexicon in 0.03 seconds so extrapolating would be able to process approximately 6492800 characters per second.

2.3.2 Best match

For best match (see table 9), our hybrid approach is dependent on the shifting n-gram which is $O(n)$. Our hybrid approach took 1.08 seconds to retrieve the best match for 'floccinaucinihilipilification'. The shifting n-gram requires 1.06 seconds so the shifting n-gram occupies 98% of the total retrieval time of the hybrid. For a three letter word input, our approach took 0.03 seconds for 194784 characters so could process 6492800 characters per second. For the 29 letter word the retrieval took 1.08 seconds so we can process 180356 characters per second. We could speed this retrieval by having three CMMs, one with unigrams stored, one bigrams and the third trigrams. This would be faster, equivalent to the n-gram triple mapping, i.e., 0.03 seconds for the 29 character word and $O(1)$ but the memory storage would be approximate six times higher. Three times higher for the trigram CMM (as seen in the the memory use evaluation in section 2.1), twice for the bigram and equivalent for the unigram. We feel the slightly slower speed of our shifting n-gram is preferable to the higher memory requirement of the alternative representation. Most searches are likely to be approximate 7 or 8 characters. N.B. for the Hamming Distance and triple mapping the search for 'the' took longer than the twenty-nine letter word. This is because there are many possible matches for 'the' which must be retrieved from the lexical token converter but there is only one to be retrieved for the longer word. The retrieval speed of the lexical token converter is proportional to the number of words retrieved.

2.3.3 Quality of retrieval

We extracted 40 spelling errors from news groups and unformatted electronic text and input the spelling errors to each algorithm in turn. Each mis-spelt word had either one or two errors and there was a roughly equal mixture of insertion, deletion, substitution and transposition (double substitution) errors.

We counted the number of times each algorithm suggested the correct spelling among the set of possibilities retrieved. We include the score for MS Word 97 spell checker for a benchmark comparison. The results are given in table 9.

Edit distance and agrep tend to produce a small number of possibilities (edit distance ≤ 6 returns and agrep ≤ 58 returns) as they score the matched and produce ranked word retrieval. On average edit distance produces a single best match and agrep roughly three best matches. We have lower precision as we return more false positives. We just produce an unordered set of the best matches. However, our hybrid approach has the best recall rate, only failing to find 'dealt' from the misspelling 'delt' - it retrieved 'delta' and 'deltoid' as the best matches. Our hybrid and MS Word 97 were the only techniques that identified 'the' as the best match for 'teh' and 'him' for 'hmi'. Hamming distance is the weakest but often succeeds when the shifting n-gram fails and vice versa hence we use the union of both techniques in our hybrid system. Our shifting n-gram, restricted to the length of the input word outperformed the free match of the standard n-gram so even though the shifting n-gram is slower we feel the additional quality mitigates the lower speed.

3 Conclusion

Spell checkers are somewhat dependent on the words in the lexicon. Some words have very few words spelt similarly, so even multiple mistakes will retrieve the correct word. Other words will have many similarly spelled words so one mistake may make correction difficult or impossible. Of the techniques evaluated, our hybrid approach had the highest recall rate at 97.5%, only failing to find 'dealt' from 'delt' as both 'delta' and 'deltoid' were present in the lexicon and matched higher. Humans averaged 74% for isolated word-spelling correction [2] (where no word context is included and the subject is just presented with

a list of errors and must suggest a best guess correction for each word) so our approach outperforms a human. Kukich [2] posits that ideal isolated word-error correctors should exceed 90% when multiple matches may be returned which we have comfortably exceeded.

Our exact match procedure is very rapid and independent of the length of the input spelling. The best match process is slower as the shifting triple is slower. However, we feel the superior quality of the shifting triple as compared to the regular trigram offsets the lower speed. The approach does not rank the retrieved best matching words. It just returns an unranked alphabetically sorted list. We may include scoring later, for example, when matching 'wprd' for 'word', 710 best matches were returned, the average number of returns is approximately eight. To prevent the user being overwhelmed we could score when 10 or more best matches are found and return them in blocks of 10 in score order. Our approach is only intended as a front-end to an IR system so we require an implementation that amalgamates with the existing IR system as the orthogonal word vectors will form the inputs to the next stage of the system. It is efficient with respect to memory storage and speed of retrieval and identifies the intended words from incorrect spellings but is still reasonably simple.

4 Acknowledgement

This work was supported by an EPSRC studentship.

References

[1] J. Austin. Distributed associative memories for high speed symbolic reasoning. In R. Sun and F. Alexandre, editors, *IJCAI '95 Working Notes of*

Workshop on Connectionist-Symbolic Integration: From Unified to Hybrid Approaches, pages 87–93, Montreal, Quebec, August 1995.

[2] K. Kukich. Techniques for Automatically Correcting Words in Text. *ACM Computing Surveys*, 24(4):377–439, 1992.

[3] S. Wu and U. Manber. Fast Text Searching With Errors. *Communications of the ACM*, 35, October 1992.

[4] S. Wu and U. Manber. AGREP - A Fast Approximate Pattern Matching Tool. In *Usenix Winter 1992 Technical Conference*, pages 153–162, San Francisco, CA, January 1992.

[5] M. Turner and J. Austin. Matching Performance of Binary Correlation Matrix Memories. *Neural Networks*, 10(9):1637–1648, 1997.

[6] V.J. Hodge and J. Austin. An Evaluation of Standard Retrieval Algorithms and a Binary Neural Approach. *Neural Networks*, 14(3), 2001.

[7] J. R. Ullman. A Binary n-Gram Technique for Automatic Correction of Substitution, Deletion, Insertion and Reversal Errors in Words. *Computer Journal*, 20(2):141–147, May 1977.

[8] V. Cherkassky, N. Vassilas, G. Brodt, and H. Wechsler. Conventional and Associative Memory Approaches to Automatic Spelling Correction. *Engineering Applications of Artificial Intelligence*, 5(3):223–237, 1992.

5 Biographies

Professor Jim Austin has the Chair of Neural Computation in the Department of Computer Science, University of York, where he is the leader of the Advanced Computer Architecture Group. He has extensive expertise in neural networks as well as computer architecture and vision. Jim Austin has published extensively in this field, including a book on RAM based neural networks.

Victoria Hodge is a PostGraduate Research Student in the Department of Computer Science, University of York. She is a member of the Advanced Computer Architecture Group investigating the integration of neural networks and information retrieval.

6 Author Details:

Contact Author:

Victoria J. Hodge

Dept. of Computer Science,

University of York,

UK

YO10 5DD

Tel: +44 1904 432729

Fax: +44 1904 432767

E-Mail: vicky@cs.york.ac.uk

Co-Author:

Prof. Jim Austin

Dept. of Computer Science,

University of York,

UK

YO10 5DD

Tel: +44 1904 432734

E-Mail: austin@cs.york.ac.uk

7 Figure Captions

Fig 1: Diagram of the AURA modular system.

Fig 2: Diagram showing three stages of network training.

Fig 3: Diagram showing system recall. The input pattern has 1 bit set so the CMM is thresholded at 1.

Fig 4: Diagram showing Hamming Distance matching.

Fig 5: Diagram showing a trigram shifting right.

Fig 6: Diagram showing the triple mapping process.

8 Figures

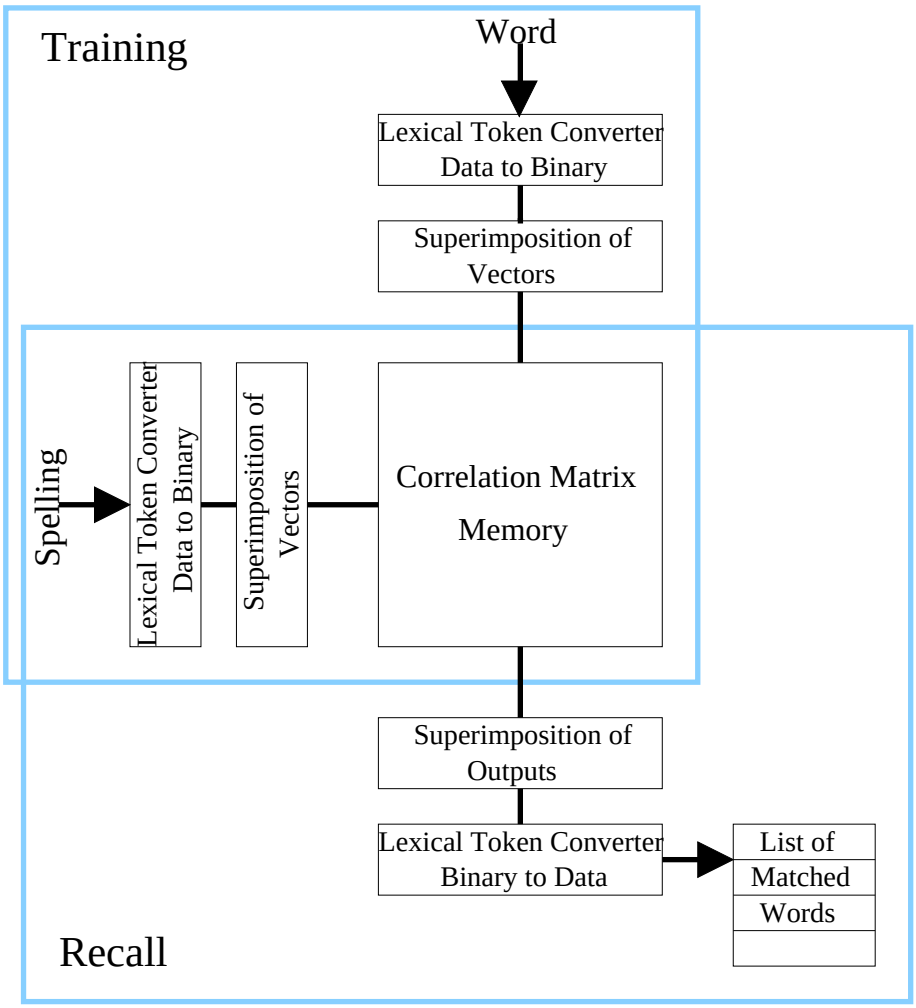
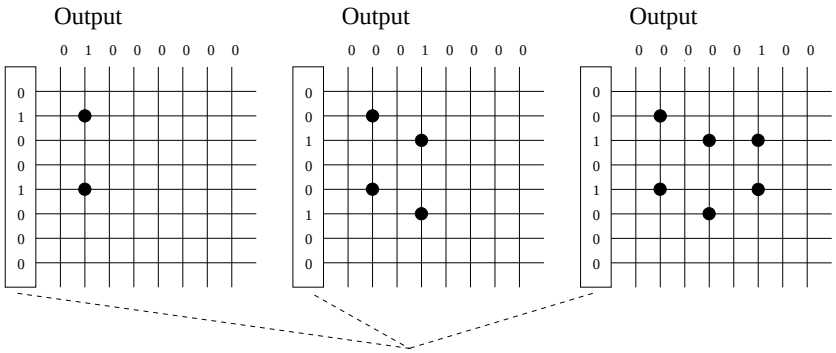
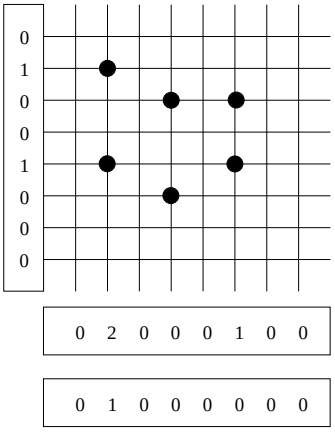

Training
Word
Lexical Token Converter
Data to Binary
Superimposition of
Vectors
Spelling
Lexical Token Converter
Data to Binary
Superimposition of
Vectors
Correlation Matrix
Memory
Superimposition of
Outputs
Lexical Token Converter
Binary to Data
List of
Matched
Words
Recall

Figure 1:



Inputs to be trained into the network. The input spelling is shown with 4-bit chunks for simplicity. The input spellings are 2 letter words with 1 bit set in each chunk. The output vector serves as an identifier for the input spelling to uniquely identify each spelling in the lexical token converter.

Figure 2:

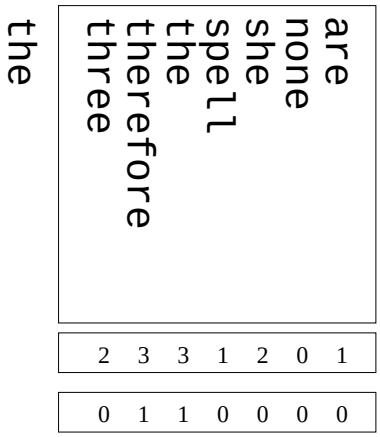


Here the input spelling is shown with 4-bit chunks for simplicity. The word is a 2 letter word with 1 bit set in each chunk. To match exactly, we set the Willshaw threshold to 2 to identify any lexicon words that match both characters in the relevant positions.

Activation - 2 input bits set: threshold at 2

Output pattern after thresholding

Figure 3:



The input word ‘the’ is compared to 7 lexicon words trained in to the CMM. Any word matching the three letters in the first three letters of the word will be retrieved.
Both ‘the’ and ‘therefore’ match the first 3 letters correctly.

3 characters input so threshold at 3.

Superimposed output vector

Figure 4:

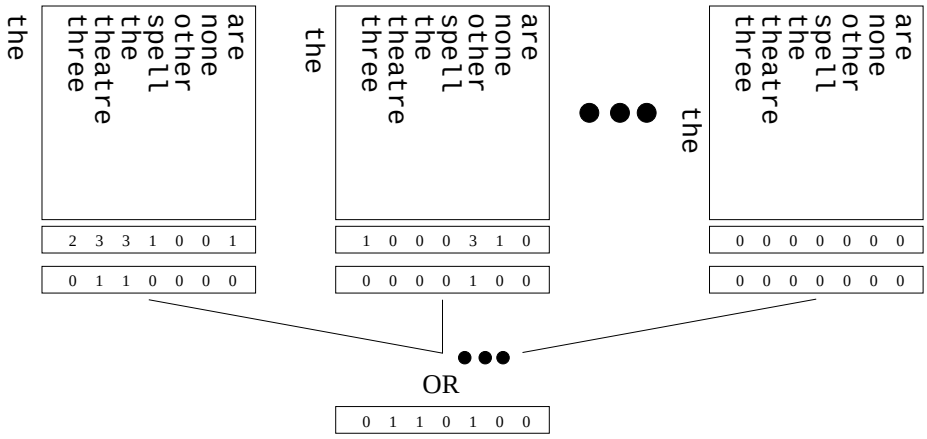


Figure 5:

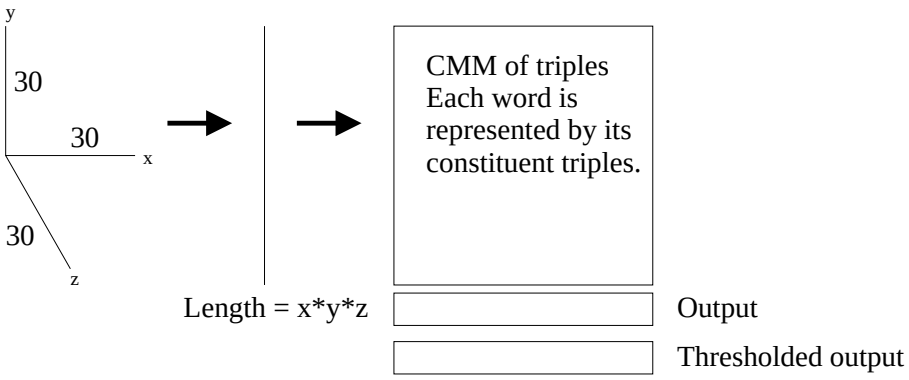

y
30
30
x
30
z
CMM of triples
Each word is
represented by its
constituent triples.
Length = x*y*z
Output
Thresholded output

Figure 6:

9 Tables

Method	CMM size bytes	Word Decoder size bytes
Hybrid method	1211416	1432620
Triple mapping method	3184784	1432620
Edit Distance word array	1189550	

Table 1: Table showing the memory usage for three comparable spell checking approaches.

Method	Time (secs) for 'flocci...'	Time (secs) for 'the'
Hybrid - Hamming $\wedge$ length	0.03	0.03
Triple mapping	0.02	0.02
Agrep	0.02	0.02
Levenshtein	1.53	0.17

Table 2: Table detailing the retrieval time in seconds for a 29-letter word exact match retrieval and a 3-letter word exact match retrieval.

Method	Time (seconds) for 'flocci...'	Time (seconds) for 'the'
Hybrid	1.08	0.03
Hamming Distance	<0.01	0.01
Shifting n-gram	1.06	0.03
Triple mapping	<0.01	0.03
Agrep	0.02	0.05
Levenshtein	1.53	0.17

Table 3: Table detailing the retrieval time in seconds for a 29-letter word best match retrieval and a 3-letter word best match retrieval.

Method	Recall	Method	Recall
Hybrid	39/40	Agrep	35/40
Levenshtein	38/40	Triple mapping	29/40
MS Word 97	37/40	Hamming Distance	21/40
Shifting n-gram	35/40		

Table 4: Table detailing the recall accuracy for 40 spelling errors.