Natural Language Engineering **8** (1): 3–23. © 2002 Cambridge University Press
DOI: 10.1017/S1351324901002753 Printed in the United Kingdom

An LVQ connectionist solution to the non-determinacy problem in Arabic morphological analysis: a learning hybrid algorithm

M. A. EL-AFFENDI

Department of Computer Science, College of Computer and Information Sciences, King Saud University, Riyadh 11543, PO Box 51178, Saudi Arabia
email: affendi@ccis.ksu.edu.sa

(*Received 2 March 2000; revised 13 February 2001*)

Abstract

Most of the morphological properties of derivational Arabic words are encapsulated in their corresponding morphological patterns. The morphological pattern is a template that shows how the word should be decomposed into its constituent morphemes (prefix + stem + suffix), and at the same time, marks the positions of the radicals comprising the root of the word. The number of morphological patterns in Arabic is finite and is well below 1000. Due to these properties, most of the current analysis algorithms concentrate on discovering the morphological pattern of the input word as a major step in recognizing the type and category of the word. Unfortunately, this process is non-determinitic in the sense that the underlying search process may sometimes associate more than one morphological pattern with the given word, all of them satisfying the major lexical constraints. One solution to this problem is to use a collection of connectionist pattern associaters that uniquely associate each word with its corresponding morphological pattern. This paper describes an LVQ-based learning pattern association system that uniquely maps a given Arabic word to its corresponding morphological pattern, and therefore deduces its morphological properties. The system consists of a collection of hetroassociative models that are trained using the LVQ algorithm plus a collection of autoassociative models that have been trained using backpropagation. Experimental results have shown that the system is fairly accurate and very easy to train. The LVQ algorithm has been chosen because it is very easy to train and the implied training time is very small compared to that of backpropagation.

1 Introduction

One of the unique properties of Arabic is that most of its words are derived from a finite set of roots using a set of predefined morphological 'patterns' or 'forms'. A morphological pattern is nothing but a template that indicates the positions of the radicals (original or root letters), and shows how the word may be decomposed into its constituent morphemes (prefix, root, infixes, suffix). The morphological pattern also represents the 'class' or 'category' of the word, and in some cases indicates its

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

syntactic and semantic roles. The number of morphological patterns in Arabic is finite, and is well below 1000 (Yagoub 1996). The number of roots does not exceed 10,000 (El-Hannash 1996). Due to these interesting properties, most of the current morphological analysers concentrate on discovering the morphological pattern of the input word either implicitly or explicitly. The problem faced by many of the current algorithms is that the process of recognizing the morphological pattern of a given word is not deterministic, and in some cases, leads to multiple solutions (more than one possible pattern). The solution adopted in current algorithms is to leave the final decision for the upper layer. The upper layer usually relies on the context to choose between the provided solutions.

In this paper, it is shown how connectionist associative memory models may be used to overcome the non-determinacy problem while providing better performance. A hybrid adaptive system consisting of a collection of hetro-associative and auto-associative models is described in some detail. The hetroassociative components are trained using the well known LVQ algorithm (Fauset 1994; Kohonen 1995), while the auto-associative components are trained using the backpropagation algorithm (Fauset 1994). The LVQ algorithm has been chosen because the corresponding training time is very low while its accuracy is very close to that of backpropagation (according to the experiments performed). The memory properties of both algorithms are excellent. Both of them easily accommodated more than 7500 exemplars.

The connectionist approach adopted in this paper is similar to the two-level single-tape finite state machine approach used by Beesley (1996) and the multi-tape finite state machine approach used by Kay (1987), Kiraz (1994, 1998, 2000) and a score of other researchers reported in these papers. The main similarity is that all these approaches represent a mechanism to map between a lexical form that encodes the constituent morphemes and a surface form that represents the actual Arabic words. The main difference, however, is that the two-level FSM models employ some sort of explicit formalism to perform the mapping, while the connectionist models employ only implicit or embedded knowledge to perform the mapping. This implicit knowledge is acquired through training.

Before describing the actual model, the paper starts by giving an idea about the morphological properties of Arabic, and briefly illustrates some of the conventional analysis algorithms and the two-level finite state machine approach.

2 Morphological properties of the Arabic language

The morphological properties of the Arabic language may be summarized as follows:

- Arabic is mainly a derivational language, in the sense that most of its words are derived from a finite set of roots using a finite set of morphological 'forms' or 'patterns'.
- According to El-Hannash (1996), the total number or roots in Arabic is around 9464. This number varies slightly from dictionary to dictionary, but it remains around 10,000 on average. Around 70% of these are tri-literal while the rest are 4-letter roots.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

- In a recent experimental study (El-Affendi 2000), it has been revealed that only 2214 of the above mentioned roots are in common use.
- The total number of morphological forms is around 900, according to Yagoub (1996) (excluding combinations with pronouns and external affixes). The experimental study conducted by El-Affendi (2000) revealed that only 130 of these are in common use.
- An Arabic word may either be a verb, a noun or a particle.
- The total number of particles in Arabic is around 80. Each particle is characterized by a number of syntactic and semantic roles. For a detailed account of these roles, see El-Dahdah (1998).
- Nouns are divided into two main categories: variable and invariable. The variables are further divided into inert and derivable. The inert include pure Masdar (in Arabic, Masdar is a class of nouns that indicates the occurrence of an event or an action that is not bound to time – past, present or future). The pure Masdar is considered to be the origin from which Arabic words are derived by a major school of classic Arabic linguists), noun of genus, proper names, etc. The Derivables include subject noun, object noun, pro-adjective, superlative forms, preference forms, place noun, time noun, tool noun. For more details see El-Dahadah (1988).
- There are three categories of verbs in Arabic: past, present and imperative. In addition, verbs are categorized as pure or extended, regular or defective, etc. For each category, the morphological properties are well encoded in a finite set of morphological forms invented by earlier scholars and specialists. In addition, these forms reflect the behaviour of verbs when combined with pronouns, a common occurrence in Arabic text. Therefore, one of the best approaches to encoding knowledge about verbs is to store all the relevant morphological forms and their attributes. The number of verb forms in Arabic does not exceed 40.
- One of the most interesting properties of Arabic morphology is its non-linear nature (Beesley 1996). Arabic words are not formed by concatenating morphemes, but rather through intersection or interdigitation of morphemes.

3 The non-determinacy problem

As mentioned in the introduction, Arabic morphological analyzers sometimes return multiple answers or solutions, while only one unique solution is expected. This is maily due to the follwing reasons:

1. The combination [Prefix + Stem + Suffix] is not unique for most composite Arabic words. A single word X may correspond to more than one proper combination of the above components. Good examples of this problem are given in Thalouth and Al-Dannan (1990).
2. The stem of a given derivational word X may match more than one morphological form, many of which may lead to proper Arabic roots. The analyser will naturally accept them all, hoping that the overlying application will rely on the context to select one of them.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

Note that non-determinacy is one source of ambiguity in Arabic morphological analysis. Other sources of ambiguity are:

(a) Due to the fact that short vowels are secondary characters that are pronounced but are not necessarily written in most of the cases, the same lexical form may lead to more than one surface word. For example, the lexical form /ktb/ ( ) may be pronounced as /kutib/ ( ) or /kataba/ ( ) or /kutub/ ( ). The only way to choose between these surface forms is to consult the surrounding context or to rely on the statistical properties of the language.
(b) The same morphological pattern may correspond to more than one morphological category. For example, the pattern /fa'al/ ( ) may correspond to a 'Masdar' or 'verb'.

This paper addresses only the non-determinacy problem. The remaining sources of ambiguity may be dealt with using the statistical properties of Arabic (Mryati 1990; El-Affendi 2000) or by relying on the context.

4 Related work

During the past few years, some effort has been exerted to develop models and algorithms for Arabic morphological analysis and generation. These efforts may classified into two main categories:

- The algorithmic approach.
- The two-level finite state machine approach.

In this section, some of the above-mentioned research efforts are briefly reviewed.

4.1 The algorithmic approach

Over the past few years, a number of morphological analysis algorithms have been developed (Thalouth and El-Dannan 1990; El-Affendi 1991; Al-Fedaghi and Al-Anzi 1989; Hilal 1990). A number of these algorithms are now used in public software packages. In this section, we describe typical scenarios for two these algorithms.

4.1.1 A typical conventional algorithm

The conventional algorithm described below is a slightly modified version of the algorithm given in Thalouth and El-Dannan (1990).

The algorithm uses a number of lexicons and flags to perform its work. These may be described as follows:

1. A list of all invariable words in Arabic including particles, pronouns, etc. (less than 500). In the original paper these are referred as 'structured' words.
2. A list of all roots in Arabic, RootsList.
3. A list of all morphological forms (patterns), PatternList.
4. A list of all possible prefixes in Arabic, PrefixList.
5. A list of all possible suffixes in Arabic, SuffixList.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

In addition to the above lists, the algorithm employs the following compatibility matrices:

1. A root-pattern compatibility matrix, from which one can determine if a root is compatible with a particular pattern or not. The dimensions of this matrix may be as large as 10000 × 900, but in fact the size of the matrix is less than that and may be 5000 × 400 on average (some of the roots and patterns are not used in modern Arabic).
2. A (prefix)–(invariable word) compatibility matrix through which one can determine if a prefix is compatible with a particular invariable word or not. The size of the matrix may be around 500 × 50 maximum.
3. A prefix-pattern compatibility matrix from which one can know if a prefix is compatible with the pattern in question or not. The maximum size of the matrix is around 900 × 50.

The above matrices are usually implemented as Boolean flags which may either be set or reset depending on the compatibility of the entries.

In addition, the algorithm employs a 'duality' flag which indicates if a particular word may be interpreted as both variable or invariable word. It is up to the higher layer algorithm to make the choice. In the algorithm below this flag is named DualRole.

The algorithm starts by first checking if the input word is invariable and, if not, proceeds to analyse the word assuming that it is variable (derivational).

The Algorithm

```
begin
Get(word)
prefix:=ExtraclLargestPrefix(word, PrefixList);
word1:=word  prefix;
in invariable(word1) and Compatible(prefix, word1) then
begin
   WriteMorpInfo(word1);
   if not DualRole(word1)  exit;
end
else
begin
   repeat
     suffix:=ExtractLargestSuffix(word1, SuffixList);
     word2:=word2 - suffix;
     for each pattern p in the PatternList do
     begin
       if Compatible(p, word2) and Copatible(prefix, p, suffix) then
       begin
           Root := Intersect(p, word2);
           if ValidRoot(Root)   WriteMorphInfo(word2);
         end;
```

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

```
    end;
    ch := FirstLetter(suffix);
    suffix := suffix - ch;
    word2 := word2 + ch;
  until suffix is empty;
  ch:= LastLetter(Prefix);
  word2:= word2 + ch;
  while not Empty(Prefix) do
  begin
    for each pattern p in the PatternList do
    begin
     if Compatible(p, word2) and Compatible(prefix, p, suffix) then
     begin
         Root := Intersect(p, word2);
         if ValidRoot(Root)   WriteMorphInfo(word2);
     end;
    end;
   Prefix:=Prefix - ch;
   ch:= LastLetter(Prefix);
   word2:= word2 + ch;
 end;
end algorithm;
```

4.2 Sliding Window Approximate Matching algorithm

The Sliding Window Approximate Matching (SWAM) algorithm is one of the algorithms that proved to be very successful and efficient in performing analysis and generation (El-Affendi 1999b). It has been extensively tested using a custom built Arabic Internet search engine (El-Affendi, Al-Sulliman and Al-Doseri 1999). The main attraction of the algorithm is that it is very exhaustive, and combines analysis and generation in one phase. However, due to its exhaustive nature, non-determinacy is much more obvious than with other algorithms. In the following sections, we explain how this problem may be overcome using embedded connectionist models.

As will be apparent, the algorithm deals with derivational Arabic words, as well as combinations of invariable patterns where more than one invariable word are combined to form a compound word (such as combinations of particles and pronouns, e.g. /minkum/ ( ), which is a combination of two words: the particle /min/ (from) and the plural pronoun /kum/ (you). As for derivational words, the algorithm uses a list of stored morphological forms to discover the appropriate decomposition of a given Arabic word. A window containing the stored pattern is slid against the analysed word in an attempt to discover the appropriate decomposition of the word. As for invariable patterns, the algorithm slides a window containing the actual pattern against the input word in an attempt to discover if the pattern matches part of the analysed word. In both cases, the algorithm computes a match score and accepts the resulting decomposition if a number of constraints are satisfied.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

Required Data

SWAM requires the availability of the lists mentioned in section 4.1.1, but does not use compatibility matrices.

Algorithm details

The term 'stored pattern' in the algorithm below is used in a general sense to indicate either an Arabic morphological derivational form, a particle or an invariable noun. As stated above, all these categories are finite in Arabic, and may easily be collected and stored. The term 'word' is used to indicate a given input Arabic word that should be analyzed.

Given a particular input Arabic word, the algorithm tries to find members of the above lists that achieve a reasonable 'match score', as will be described below. A word that achieves an acceptable 'match score' is said to 'partially match the input word'. The significance of this is that the algorithm will be able to discover all derivations for a given Arabic word.

The SWAM Algorithm

```
Get(word);
Window:=1;  //The window begins at position 1
For each stored pattern p do
Begin
  While (window + length(p))  <= length(word)  do
   Begin
     Stem:=ExtractStem(window, p, word);
     Root:=intersect(stem, p);
     Prefix:=ExtractPrefix(p,word,window);
     Suffix:=ExtractSuffix(p,word,window);
     If ValidRoot(Root) and ValidPrefix(Prefix)
       and ValidSuffix(Suffix) then
       MatchVector:=Align(Pattern, stem);
     MatchScore:= ComputeScore(MatchVector);
     If MatchScore>threshold AcceptDecompostion;
     Window:=Window+1;  //move the window one position
   End;
End;
```

The ExtractStem function copies into stem a substring equal to p in length from the input word starting at the window position. The MatchVector is constructed such that MatchVector[i]=1 if either (p[i] represent a radical) or (p[i] is not a radical, but p[i]=stem[i]), otherwise MatchVector[i]=0. The term radical is used to indicate a root character (part of the root). MatchScore is equal to the length of the longest sequence of ones in the MatchVector.

4.3 The two-level finite state machine approach

According to Kiraz (1994), two-level finite state morphology was first introduced by Kosenniemi (1983) to deal with the highly inflectional nature of Finnish morphology.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

Soon after its introduction, the approach was successfully applied to other languages. The two-level model is used to map between two levels of representation: a lexical level (consisting of morphemes) and a surface level representing the actual word. The main problem of the initial two-level model introduced by Kosenniemi is the assumption that simple concatenation may be used to generate surface level strings from lexical level morphemes. This makes it unsuitable for non-concatenative languages such as Arabic.

To compensate for this shortcoming, Kay (1987) introduced the idea of multiple tapes at the lexical level, where a number of lexical tapes are applied in parallel to generate a surface form.

Another important development in the history of two-level morphology is the introduction of a formalism that maps between equal sequences (groups) of characters at the same time (Black, Ritchie, Pulman and Russell 1987) instead of the previous formalisms that deal with one character at a time. Ruessink (1989) added contexts and allowed the mapping between unequal sequences of characters. Pullman and Hepple (1993) modified Black's formalism to accommodate feature-based representations at the lexical level. The major significance of this is that lexical entries may now include variables.

Kiraz (1994) extended the Black formalism further, such that the components of the lexical level expressions are n-tuples of regular expressions, and the left context of lexical expressions may include ellipses (empty slots). The main motivation behind Kiraz extensions is to make the formalism suitable for the representation of multiple lexical tapes suggested earlier by Kay. These extensions led to the multi-tape two-level model that Kiraz applied to Arabic morphology in a number of publications (Kiraz 1994a, 1994b, 1998, 2000) (see the example below). Kiraz (1994b) demonstrated how his two-level model may accommodate the CV analysis suggested by McCarthy (1981) for Arabic, the moraic analysis (McCarthy and Prince 1990) and affixational analysis (McCarthy 1993). For each case, Kiraz presented an illustrative two-level grammar that encodes the mapping between lexical and surface strings.

Another serious attempt to apply the two-level formalism to Arabic morphology has been reported in Beesley (1996). Beesley used single-tape finite state transducers (the term single-tape here refers to the fact that only one tape is used at the lexical level, but of course, both models use an additional surface tape) to build a working Arabic morphological system at Xerox.

In what follows, a brief account of the single-tape transducer model used by Beesley and the multi-tape two level model developed by Kiraz is given.

4.3.1 The single-tape transducer approach

Interestingly, Beesley (1996) has been able to successfully apply the single-tape two-level approach introduced by Kosenniemi (1983) to Arabic morphology, despite the concatenative nature of the model. The main trick is that he used intersection operations at the implementation level, and employed a special algorithm called 'Detour' to handle the special non-concatenative nature of Arabic morphology.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

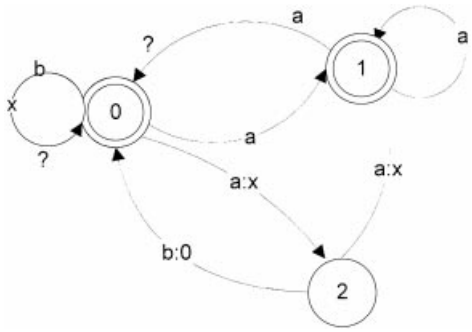


Fig. 1. Simple example transducer that encodes the relation [ab → x].

Beesley used single-tape transducers to encode the peculiarities of Arabic morphology and perform both analysis and generation. A transducer is a special type of finite state machine that encodes a regular relation. Each arc in the transducer is labelled by a pair of symbols (e.g. x:y) rather than a single symbol, as in the case of ordinary finite state machines. Therefore, a transducer may be viewed as a mapping between a pair of strings: the upper string (represented by the first symbol in the label), and the lower string (represented by the second symbol in the label). Traversing an arc of the transducer implies that the first symbol (which belongs to the upper string) in the label is replaced by the second symbol in the lower string.

The '?' symbol is used to indicate the 'unknown symbol', a symbol that does not belong to the alphabet of the transducer. The '0' is used to indicate the null case; for example, the label x:0 indicates that x in the upper string is replaced by nothing in the lower string. A label consisting of a single symbol indicates a symbol that is replaced by itself in the lower string.

As an example, consider the simple transducer shown in figure 1 (from Beesley 1996), that encodes the regular relation [ab → x].

The reader can verify that the above transducer maps the string 'caba' onto the string 'cxa'. For more on transducers, see Beesley (1996).

Clearly, transducers may be employed to encode the relations between Arabic lexical strings and the corresponding surface strings. First, the Arabic lexicon and the corresponding morphological rules are transformed into separate transducers, then composition operations are used to derive a morphological analysis transducer that may be used to perform both analysis and generation. In this transducer, the upper level language describes lexical strings that consist of roots plus forms and vowel tags, while the lower language describes surface strings.

4.3.2 Kiraz multi-tape two-level model

Motivated by the work of Kay (1987), Kiraz (1994a, 1994b) extended the Black-Ruessink-Pullman and Hepple multi-character formalism to build a multi-tape two-level finite state model for Arabic morphology. In its most basic form, Kiraz's model uses three tapes for the lexical level and one tape for the surface level (the Surface Tape (ST)). The three lexical tapes are the Root Tape (RT), the Pattern Tape (PT)

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

	u		i		+			VT (Vowel Tape)
k		t		b	+			RT (Root Tape)
c_1	v_1	c_2	v_2	c_3	+	a	+	PT (Pattern Tape)
2	3	2	3	2	5	1	4	Rule applied
k	u	t	i	b		a		ST (Surface Tape)

Fig. 2. Deriving the surface string /kutiba/.

and the Vocalism Tape (VT). The mapping process is controlled by a set of rules that represent the Kiraz multi-tape formalism. The main attraction of the multi-tape formalism is that it allows mapping between unequal sequences of characters, while taking into consideration the simultaneous transitions in all lexical tapes (the lexical sides of the rules are allowed to be n-tuple regular expressions representing the states of the three lexical tapes).

As an illustration for the multi-tape formalism, Kiraz formulated the default and boundary rules for Arabic as follows:

(1) $$* - X - * \Rightarrow * - X - *$$

(2) $$* - X - * \Rightarrow * - (C, X) - * \; C \in (c1, c2, c3, c4)$$

(3) $$* - X - * \Rightarrow * - (V, , X) - * \; V \in (v1, v2)$$

(4) $$- \; - * \Rightarrow * - + - *$$

(5) $$- \; - * \Rightarrow * - (+, +, +) - *$$

Rule 1 is the default rule which states that a lexical character (on the right-hand side) appears 'as is' on the surface. Note that the right-hand sides of the above expressions represent the lexical level, while the left-hand sides represent the surface level. Rule 2 states that any consonant C on the pattern tape and X on the root tape with no transition on the vocalism tape corresponds to X on the surface tape. Note the use of the tuple (C, X,) which consists of three slots: one for the pattern tape (C in this case), one for the root tape (X in this case) and one for the vocalism tape (empty in this case). Rule 3 states that any V on the pattern tape and X on the vocalism tape with no transition on the root tape maps to X on the surface tape. Rules 4 and 5 are the boundary rules for morphemes and stems, which state that boundary characters at the lexical levels are removed at the surface level. The following example shows how the surface word /kutiba/ (written) may be derived from the three morphemes ktb, ui and $c_1v_1c_2v_2c_3$ using the above rules and the multi-tape model shown in figure 2.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

5 A connectionist solution to the non-determinacy problem

A possible solution to the non-determinacy problem in Arabic morphological analysis is to use a neural network: PatternNet, say, that uniquely associates every Arabic word stem with its morphological pattern. If such a network is properly trained, then whenever a word stem is encountered, the corresponding morphological pattern may be obtained by recalling the pattern associater:

```
pattern = PatternNet(word)
```

The main requirement here is that the neural net should have enough memory to remember all possible patterns and should be easy to train and recall.

5.1 Selecting a neural network model

As is well known in the literature, a wide range of neural network models may be used to implement a pattern associater (Fauset 1994; Schakoff 1997; Principe, Euliano and Lefebvre 2000). In our case, there are three major candidates:

- Multiple perceptrons (MLPs), which are usually trained using the backpropagation rule (Fuaset 1994). These networks are very general, and are used to solve a wide range of problems. Their main attraction is that they have a large memory. However, a major drawback concerning these networks is that they take a long time to train. If the network is to be retrained very often (like the adaptive networks we are considering), then these networks are not good candidates.
- Bidirectional Associative Memories (BAMs) (Fauset 1994), which associate input patterns with output patterns and work in both directions. These models are very easy to train and recall, but their memory is very limited.
- Linear Vector Quantization (LVQ) networks: this type of network is used mainly for clustering purposes, and may be very good classifiers if the number of exemplars is much larger than the number of classes (patterns).

5.2 Experimental results

A number of experiments have been performed to explore the performance and capabilities of MLPs and LVQ networks, and to choose between them. Tables 1–3 summarize the main findings. It is clear that the memory of both models (MLP and LVQ) are very good. However, the training time for MLPs is much more higher than that for LVQ nets. Based on this information, and the adaptive nature of our required pattern associaters, it has been decided to use LVQ networks to associate input word stems with their corresponding morphological patterns.

5.2.1 Performance of the backpropagation algorithm

A number of experiments have been performed in El-Affendi (1999) to explore the memory properties and accuracy of the backpropagation algorithm in building

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

Table 1. *Results of Experiment 1 (Exemplars = 7500; patterns = 20; exemplar/pattern ratio = 375; mean time per epoch = 2.3976 sec.)*

Threshold	Epochs	Time (s)	Stop criterion	Accuracy (%)
0.01	257	616.1832	MSE	95.60
0.001	697	1671.1272	MSE	98.00
0.0001	1000	2397.6000	Epochs	98.40
0.001	2000	4795.2000	Epochs	99.07
0.001	3000	7192.8000	Epochs	99.33

Table 2. *Results of Experiment 2 (Exemplars = 7500; classes = 30; exemplar/class ratio = 250; mean time per epoch = 2.2565 sec.)*

Threshold	Epochs	Time (s)	Stop criterion	Accuracy (%)
0.01	1000	2256.50	Epochs	92.0
0.01	1660	3745.79	MSE	92.9
0.001	2000	4513.00	Epochs	93.5
0.001	3000	6769.50	Epochs	96.8
0.001	4000	9026.00	Epochs	97.2
0.001	4740	10695.81	MSE	97.5
0.0001	5000	11282.50	Epochs	97.6
0.0001	6000	13539.00	Epochs	97.6

morphological hetroassociative models. The main parameters investigated are the exemplar/pattern ratio, the accuracy, memory and training time. Tables 1–3 and figure 3 show the results of these experiments.

Experiment 1: Using an Exemplar/Pattern Ratio of 375

In this experiment, the Exemplar/Pattern(Class) ratio has been set to 375 (7500/20). The model managed to converge twice before the 1000 epoch limit, and its overall accuracy is 99.33%. This accuracy has been attained after two hours of training time. The results are shown in Table 1.

Experiment 2: Decreasing the Exemplar/Pattern ratio to 250

In this experiment, the Exemplar/Pattern ratio has been decreased to 250. It is clear that the performance started to degrade. The model converged once beyond the 1000 epoch level and the maximum degree of accuracy achieved is 97.6%. This accuracy have been achieved after 3.76 hours of training on a normal pentium 3 PC. The results are shown in Table 2.

Experiment 3

This experiment revealed that decreasing the Exemplar/Pattern (Class) ratio to 187.5 greatly degrades performance. The model did not converge at all, and the degree of accuracy dropped below 90%. The results are shown in Table 3.

Clearly, the major drawback of the backpropagation algorithm is the amount of training time required.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

Table 3. *Results of Experiment 3 (Exemplars = 7500; classes = 40; exemplar/class ratio = 187.5; mean time per epoch = 2.35 sec.)*

Threshold	Epochs	Time (s)	Stop criterion	Accuracy (%)
0.01	1000	2350	Epochs	83
0.01	2000	4700	Epochs	88
0.01	3000	7050	Epochs	89
0.01	4000	9400	Epochs	89

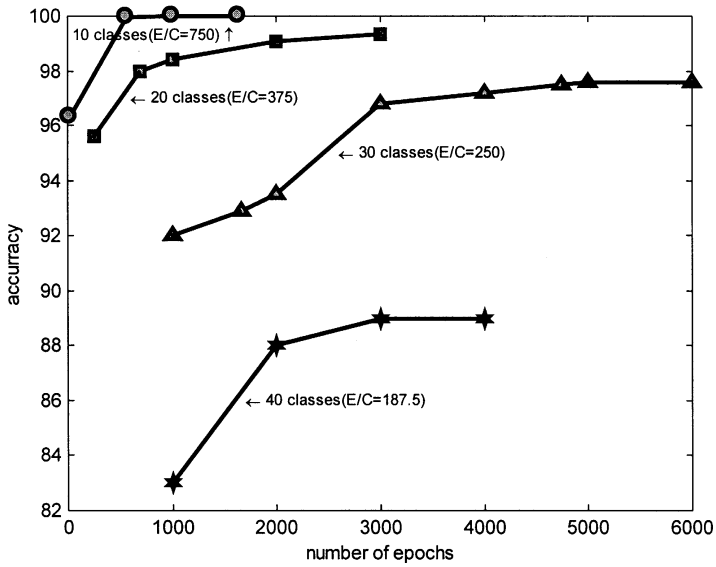


Fig. 3. The effect of exemplar/pattern ratio on accuracy.

5.2.2 Performance of the LVQ algorithm

The experiments performed revealed that the LVQ algorithm can reach an accuracy of 96% in less than 20 minutes. It has been observed that the LVQ2 (Kohonen 1995) algorithm (see the appendix) produces better results than the LVQ1 algorithm. The LVQ3 algorithm is better than the LVQ1 algorithm, but did not improve on the LVQ2 algorithm. The discussion below is based on the results obtained by the LVQ2 algorithm.

It is interesting to note that the LVQ2 algorithm needed only 100 epochs and less than 10 minutes (on a normal pentium 3 PC) to give 95% accuracy when the Exemplar/Pattern ratio is 187.5 (7500 exeplars and 40 classes). Note that for this model backpropagation gave only 89% accuracy. When the epochs were increased to 200 (for the LVQ model), the model accuracy increased to 96%.

Based on these results, it has been decided to use the LVQ in training our hetroassociative models.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

Important note
In performing the above experiments, the following conditions have been observed:

1. All words in the training set (7500 words) have the same length. It has been observed if words of different lengths are mixed then the performance deteriorates.
2. Words are not duplicated in the training set. Each word occurs only once.

5.3 The training process

Both words and patterns are first encoded into binary before presenting them to the neural networks in the above experiments. The encoding procedure is very simple, and replaces each character by its character index (an integer between 0 and 40).

As for backpropagation, the network is trained using the encoded word-pattern list. For the LVQ networks, the training set is prepared as follows:

1. Prepare a list of all the morphological patterns involved in the training set. The number of patterns is usually less or equal to 50.
2. Generate a list of reference vectors from the pattern list. The number of reference vectors is equal to the number of patterns. Each pattern is converted into a reference vector using the above-mentioned binary encoding algorithm.
3. Deduce the class for each word in the training list using its pattern. The class of the word is the index of the reference vector corresponding to its pattern.
4. Use the word-class list generated in step 3 above to train the LVQ net. The training is done using the algorithms described in the appendix.

6 Composite associative memory model for Arabic morphological analysis

As explained earlier, the main task of an Arabic morphological analysis algorithm is to find the morphological pattern for a given input Arabic word. This task may be achieved through well trained hetroassociative neural models. However, for the algorithm to use the hetroassociative model properly, and for the resulting decomposition to be acceptable, a number of additional autoassociative neural models are required. This is mainly because hetroassociative are not capable of giving a clear 'NOT LISTED' answer when presented with a wrong string (a string that was not included in the training list). They always give a matching pattern (which may the correct pattern or an approximation for it). In contrast, autoassociative models (Fauset 1994) may be used to indicate if an input string has been included in the training list or not. If the input of the autoassociative model is not equal to its output, then this may interpreted as a 'NOT LISTED' answer. For a proper member of the list, the following condition is satisfied:

```
Element = AutoNet(Element)
```

where AutoNet is an auto-associative model trained using the proper members of the list.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

The autoassociative models reqired for the algorithm described below include:

- **StemsAutoNet**: an autoassociative neural system that stores all stems for which there is a pattern in the StemsHetroNet. The StemsAutoNet system consists of several autoassociative nets indexed by stem length. Each member net is retrained whenever new stems are added to the corresponding list. This net is needed because our model is adaptive, and is not expected to include all stems from the beginning.
- **RootsAutonet**: an autoassociative system that stores all proper roots in Arabic. This system consists of two major auto nets: one for 3-letter roots and another for 4-letter roots.
- **FullWordsAutonet**: an autoassociative system that stores full Arabic words (not just stems) which are included in the FullWordHetroNet system (see the algorithm below). This system consists of several auto nets indexed by word length.

6.1 The HetroAssociative system

The HetroAssociative system represents the heart of the system and consists of two types of model:

1. A set of **StemsHetroNets** indexed by stem length. As the name implies, these nets are used to associate Arabic word stems with their corresponding morphological patterns. Each net in this group accommodates at most 30 morphological patterns (based on the experimental results), but an unlimited number of stems. The only condition is that the stems belonging to each net should be of the same length.
2. A set of **FullWordsHetronets** indexed by word length. These are used to associate full Arabic words (including prefixes and suffixes) with the corresponding morphological patterns. These nets are created gradually based on interaction with expert users. The advantage of these models is that they freeze the morphological decomposition of a given full word and reduce the analysis process to a single step.

Based on the experimental results of section 5, it has been decided that an LVQ hetro-associative neural model will be used as the backbone model.

Figure 4 illustrates how the hetroassociative and autoassociative systems are combined to perform Arabic morphological analysis.

Given an input word, the algorithm first checks if its pattern may be obtained using the FullWordHetroNet System. The checking is done using the FullWordAutoNet system. If that is not the case, then the word is decomposed using the hybrid version of SWAM (see below), otherwise the obtained pattern is validated by checking the appropriate AutoNets and lists. The pattern is accepted if it passes the validation process or passed to the corrector otherwise. The corrector tries to suggest some alternatives for the input word and its patterns.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

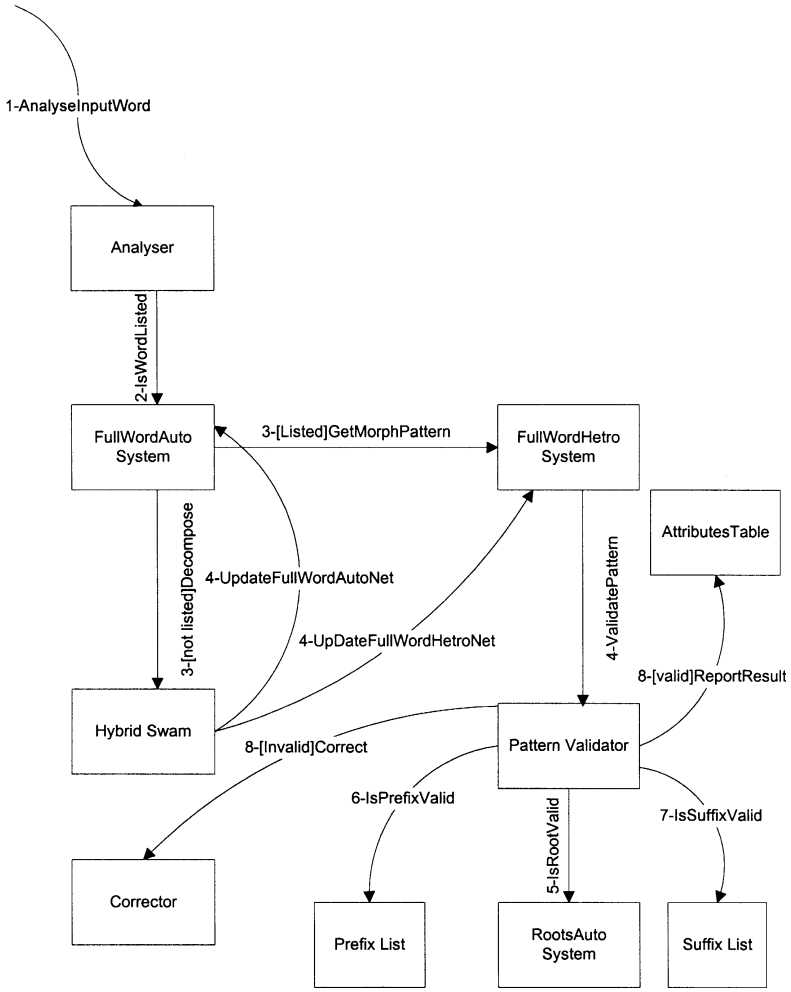


Fig. 4. The Associative Memory Analysis algorithm (note: this is a UML collaboration diagram. The index of each step represents its order in the algorithm. The square brackets [ ] are used to indicate conditions or guards. The step is performed only if the guard evaluates to true).

6.2 HybridSwam

The HybridSwam component in figure 4 uses a slightly modified version of the sliding window algorithm described in section 4.1.2 to detect valid morphological decompositions of the input word, and consequently update the FullWordHetroNet and the FullWordAutoNet systems. To perform the appropriate decomposition, HybridSWAM interacts with the StemsHetroNet system, the StemsAutonet system and the RootsAutoNet system. The following piece of pseudocode describes the behaviour of the algorithm:

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

```
L := length(InputWord);
k:=L-1;  start:=1;
while k>=3 do
begin
   while (start+k) <=L do
   begin
       stem:=substring(InputWord, start, k);
        if stem = StemsAutoNet(k, stem) then
        begin
            StemPattern:=StemsHetroNet(k, stem);
            valid:=Validate(StemPattern);
            if valid then
            begin
                Upadte FullWordAutoNet List;
                Update FullWordHetroNet List;
             end;
           end;
          start:=start+1;
      end;
      k:=k-1;
   end;
```

7 Connectionist vs. two-level FSM morphology: a comparison

The connectionist approach adopted in this paper is similar to the two-level finite state machine approach adopted by Kiraz and Beesley, in the sense that both approaches represent a mechanism to map between lexical level strings and surface level strings. However, one subtle difference is that finite state two-level models are based on some explicit formalism that should be derived and encoded by a human user, while in the connectionist case the formalism is implicit or embedded, and need not be encoded explicitly. A connectionist model learns the mapping rules from the presented exemplars and encodes the acquired knowledge in its numerical weights.

Other points of difference may be summarized as follows:

- Computational efficiency: although connectionist models may take some time to build and train, they are very efficient to use. The mapping between surface and lexical levels is reduced to a single function call and two simple conversions from text to binary, and vice versa. The complexity of the function called depends on the connectionist model used, and may be comparable to a simple hash function sometimes. As for finite state models, the mapping operation requires the traversal of several transducers, which may not be expensive if the number of transducers is small and their size is manageable. However, as reported in Beesley (1996), the Arabic morphological system developed at Xerox is much slower than its counterparts.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

- Accuracy: the two-level finite state models are expected to be more accurate than the connectionist models, since they do not involve any approximations. Connectionist models are based on numerical estimates that are not guaranteed to be accurate. However, the experiments performed have shown that the accuracy of the connectionist models is very high. The resulting inaccuracies are catered for by the autoassociative and corrector components in our model. In the first experiment performed, the LVQ model missed only about 300 of the expected 7500 patterns in the test phase. This may be taken as an upper bound for the expected error. As for the actual experimental connectionist morphological analyser being built, the system gave 100% accuracy when presented with real life data collected from a corpus. For example, the system has been able to recognize all expected patterns for a set of 2889 words presented to it.
- Symmetry: bidirectional connectionist models suffer from severe memory limitations (Fauset 1994). This is why we have been forced to use unidirectional models in our system. This means that our models can only be used for analysis. As for generation, the layers should simply be flipped and the model retrained. Transducer models are bidirectional, and may be used for both analysis and generation.

The storage requirements for the two approaches are comparable, and depend upon the number of components used in the implementation. The connectionist system described in the above section requires around 14 autoassociative nets plus 12 hetroassociative nets on average during the intermediate adaptation phase. Each of these nets requires the storage of small weight matrix (30×30 on average). The number of components required may be greatly reduced as the model converges to maturity.

8 Conclusions

A hybrid connectionist system for Arabic morphological analysis has been described. The system is adaptive and can learn from interaction with users. When the system reaches maturity, the morphological analysis process will be reduced to a single step: a call to the FullWordsHetroNet system. This will greatly improve the performance of overlying applications. Note that, in the limit (after exhaustive training), the only data structures required are the reference vector sets.

The experiments performed revealed the power of the LVQ algorithm in building associative memory models. Its performance is certainly better than that of backpropagation.

On the negative side, the connectionist algorithm introduces an encoding step and a decoding step.

A The LVQ learning algorithms

LVQ1

1. Initialize the reference vectors.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

2. While stop condition is false do
   3. For each training vector x perform the following
      4. Find j such that $|| x - w_j ||$ is minimum.
      5. Update wj as follows:
         If input class = C_j then
           wj(new) = wj(old) + α(x − wj(old))
         else
          wj(new) = wj(old) - α(x − wj(old))
6. Reduce learning rate α.
7. Test stopping condition.

The LVQ2 Algorithm

In the LVQ1 algorithm only the winning reference vector is updated. In the LVQ2 algorithm, two reference vectors (the winner and the runner-up) are updated if certain conditions are satisfied. The conditions under which these two vectors are updated may be summarized as follows:

- The winning vector (the one closest to the input vector) and the runner-up (the second closest) represent different classes.
- The input vector belongs to the class of the runner-up reference vector.
- The distance from the input vector to the winner and runner-up vectors is approximately the same. This condition is usually made more specific using a window. Defining d_c to be the distance from the input vector to the winner and d_r to be the distance to the runner-up, the two reference vectors are updated if:

$$d_c/d_r > 1 - \epsilon \text{ and } d_r/d_c < 1 + \epsilon$$

  where ϵ is a small number.

If all the above conditions are met, then the two vectors w_c (the winner) and w_r (the runner-up) are updated as follows:

$$w_c(t+1) = w_c(t) - \alpha(t)(x(t)w_c(t))$$

$$w_r(t+1) = w_r(t) + \alpha(t)(x(t)w_r(t))$$

References

Al-Fedaghi, S. S. and Al-Anzi, F. S. (1989) A new algorithm to generate root-pattern forms. *Proc. 11th National Computer Conference*, p. 391. Dhahran, Saudi Arabia.

Beesley, K. R. (1996) Arabic finite-state morphological analysis and generation. *Proc. 16th International Conference on Computational Linguistics – COLING'96*, **1**: 89–94.

Beesley, K. R. (1998) Arabic morphological analysis on the Internet. *Proc. 6th International Conference and Exhibition on Multi-lingual Computing*, pp. 17–18. Cambridge, UK.

Black, A. W., Ritchie, G. D., Pulman, S. G. and Russell, G. J. (1987) Formalisms for morphographemic description. *Proc. 3rd European Chapter of The Association for Computational Linguistics ACL*, pp. 11–18. Copenhagen.

El-Affendi, M. A. (1991) An algebraic algorithm for Arabic morphological analysis. *Arabian J. Sci. & Eng.* **16**(4B): 605–611.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

El-Affendi, M. A. (1999a) Pattern association through backprobagation: a study of the performance of the backprobagation algorithm in building hetro-associative neural models. Technical report ccis/backprob/99, Department of Computer Science, CCIS, King Saud University, Saudi Arabia.

El-Affendi, M. A. (1999b) Performing Arabic Morphological Search on the Internet: A Sliding Window Approximate Matching (SWAM) Algorithm and its Performance. Departmental report ccis/swam/99, Department of Computer Science, King Saud University, Saudi Arabia.

El-Affendi, M. A. (2000) On the morphological entropy of Arabic. Technical report ccis/Arabic-entropy, Department of Computer Science, King Saud University, Saudi Arabia.

El-Affendi, M. A., Al-Sulliman, L. and Al-Doseri, A. (1999) Al-Mishkat: An Arabic morphological Internet search engine and its implementation. *Proc. GulfInternet99 Symposium*, Dhahran, Saudi Arabia.

El-Dahdah, A. (1988) *A Dictionary of Terms of Declension and Structure in Universal Arabic Grammar*. Beirut: Lebanon Library Publishers.

El-Hannash, M. (1996) A formal linguistic tool for machine translation programs. *Proc. 1st Workshop on Machine Translation*, KFUPM, Dhahran, Saudi Arabia.

Fauset, L. (1994) *Fundamentals of Neural Networks*. Englewood Cliffs: Prentice Hall.

Hilal, Y. (1990) Morphology and syntax of the Arabic language. In: Mackay, P. (ed.), *Computers and the Arabic Language*, pp. 201–207. New York: Hemisphere.

Karttunen, L. (1994) Constructing lexical transducers. *Proc. 15th International Conference on Computational Linguistics – COLING '94*, pp. 406–411. Kyoto, Japan.

Kay, M. (1987) Noncatenative finite-state morphology. *Proc. 3rd Conference of the European Chapter of The Association for Computational Linguistics*, pp. 2–10. Copenhagen.

Kiraz, G. A. (1994a) Multi-tape two-level morphology: a case study in Semitic non-linear morphology. *Proc. 15th International Conference on Computational Linguistics – COLING '94*, pp. 159–166. Kyoto, Japan.

Kiraz, G. A. (1994b) Computational analysis of Arabic morphology. In: Narayanan, A. and Ditters, E. (editors), *The Linguistic Computation of Arabic*. To appear.

Kiraz, G. A. (1998) Arabic computational morphology in the west. *Proc. 6th International Conference and Exhibition of Multi-Lingual Computing*. Cambridge.

Kiraz, G. A. (2000) Multi-tiered nonlinear morphology using finite automata: a case study on Syriac and Arabic. *J. Computational Linguistics*, **26**(1): 77–105.

Kohonen, T. (1995) *Self Organizing Maps*. New York: Springer-Verlag.

Koskenniemi, K. (1983) *Two Level Morphology*. PhD Thesis, University of Helsinki.

McCarthy, J. (1981). A prosodic theory of nonconcatenative morphology. Linguistic Inquiry 12(3). pp. 373-418.

McCarthy, J. (1993) Template form in prosodic morphology. In: Stevan, L. *et al.* (editors), *Papers from the 3rd Annual Formal Linguistics Society of Midamerica Conference*, pp. 187–218. Indiana University Linguistic Club, Bloomington.

McCarthy, J. and Prince A. (1990) Prosodic morphology and templatic morphology. In: Eid, M. and Mecarthy, J. (editors), *Perspectives on Arabic Linguistics II: Papers from the Second Annual Symposium on Arabic Linguistics*, pp. 1–54. Benjamins, Amsterdam.

Mryati, M. (1990) Statistical studies in Arabic linguistics. In: Mackay, P. (editor), *Computers and the Arabic Language*, pp. 190–200. New York: Hemisphere.

Narayman A. and Hashem, L. (1993) On abstract finite state morphology. *Proceedings 6th Conference of the European Chapter of the Association of Computational Linguistics*, pp. 297–304.

Principe, J. C., Euliano, N. R. and Lefebvre, W. C. (2000) *Neural and Adaptive Systems: Fundamentals Through Simulation*. New York: Wiley.

Pulman, S. and Hepple, M. (1993) A feature-based formalism for two-level phonology: a description and implementation. *Computer Speech & Language*, **7**: 333–358.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press

Ritchie, G. D., Russel, G. J., Black, A. W. and Pulman, S. G. (1991) *Computational Morphology*. Cambridge: MIT Press.

Ruessink, H. (1989) Two-level formalisms. *Working Papers in Natural Language Processing 5*, Rijksuniversiteit Utrecht.

Schalkoff, R. (1997) *Artificial Neural Networks*. New York: McGraw-Hill.

Thalouth, B. and Al-Dannan, A. (1990) A comprehensive Arabic morphological analyser generator. In: Mackay, P. (editor), *Computers and the Arabic Language*, pp. 208–217. New York: Hemisphere.

Yagoub, E. B. (1996) *The Dictionary of Morphological Forms*. Beirut: Dar Aalam El-Kutub.

https://doi.org/10.1017/S1351324901002753 Published online by Cambridge University Press