Appears in the Proceedings of the *CONALD Workshop on Learning from Text and the Web*, June 1998.

Building Intelligent Agents for Web-Based Tasks: A Theory-Refinement Approach

Jude Shavlik and **Tina Eliassi-Rad**
University of Wisconsin-Madison
1210 W. Dayton Street
Madison, Wisconsin 53706
{shavlik, eliassi}@cs.wisc.edu
http://www.cs.wisc.edu/~{shavlik, eliassi}

Abstract

We present and evaluate an infrastructure with which to rapidly and easily build intelligent software agents for Web-based tasks. Our design is centered around two basic functions: ScoreThisLink and ScoreThisPage. If given highly accurate such functions, standard heuristic search would lead to efficient retrieval of useful information. Our approach allows users to tailor our system's behavior by providing approximate advice about the above functions. This advice is mapped into neural network implementations of the two functions. Subsequent reinforcements from the Web (e.g., dead links) and any ratings of retrieved pages that the user wishes to provide are, respectively, used to refine the link- and page-scoring functions. Hence, our agent architecture provides an appealing middle ground between non-adaptive "agent" programming languages and systems that solely learn user preferences from the user's ratings of pages. We present a case study where we provide some simple advice and specialize our general-purpose system into a "home-page finder". An empirical study demonstrates that our approach leads to a more effective home-page finder than that of a leading commercial Web search engine.

Introduction

We describe and evaluate an implemented system which provides a substrate upon which to create intelligent agents for the Web. Our approach is based on ideas from the theory-refinement community (Pazzani & Kibler 1992; Ourston & Mooney 1994; Towell & Shavlik 1994). Users specify their personal interests and preferences using the language we designed for discussing aspects of the contents and structure of Web pages. These instructions are then "compiled" into neural networks, thereby allowing subsequent refinement whenever training examples are available. The Wisconsin Adaptive Web Assistant (Wawa) uses ideas from reinforcement learning to automatically create its own training examples, though Wawa can also use any user-provided training examples. Thus our design has the important advantage of producing self-tuning agents.

We first describe the Wawa system, then show how it can easily be instructed to create a home-page finder. We empirically study this home-page finder and present results that support our claim that our version outperforms the home-page finder that the search engine HotBot provides. Our experiments also demonstrate that Wawa improves its performance by using the training examples it automatically generates.

System Description

At the heart of Wawa are two neural networks, implementing the functions ScoreThisLink and ScoreThisPage. These functions, respectively, guide the system's wandering within the Web and judge the value of the pages encountered. The user mainly programs these two functions by providing what we call *advice*, which is basically, rules-of-thumb for guiding Wawa's wandering and for specifying how it scores pages. Following (Maclin & Shavlik 1996), we call our programming language an advice language, as this name emphasizes that the underlying system does not blindly follow the user-provided instructions, but instead refines this advice based on the system's experiences.

Our networks have very large input vectors (i.e., sets of features), since that allows us to have an expressive advice language; each user's personal advice essentially focuses attention on only a small subset of the features, thereby making learning feasible. For example, a cancer researcher or a stock analyst can express their particular interests in our advice language, then have Wawa regularly monitor relevant Web sites for new articles about their interests.

We envision that there are two types of potential users of our system: (a) developers who build an intelligent agent on top of Wawa and (b) people who use the resulting agent. When we use the phrase *user* in this article, we mean the former case. Both types of users can provide advice to the underlying neural networks, but we imagine that usually the *type B* users will indirectly do this through some specialized interface that the *type A* user creates. A scenario like this will be seen in our experimental section.

Table 1 provides a high-level description of Wawa.

First, its initial neural networks need to be created (or read from disk should this be a resumption of a previous session). One can view the process of converting user-provided advice into neural networks as analogous to compiling a traditional program into machine code, but our system instead compiles instructions into an intermediate language expressed using neural networks. This provides the important advantage that our "machine code" can automatically be refined based on feedback provided by either the user or the Web.

Table 1: The WAWA Algorithm

Unless they have been saved to disk in a previous session, create the *ScoreLink* and *ScorePage* neural networks by reading the user's initial advice (if any).

Either (a) start by adding user-provided URLs to the search queue; or
(b) initialize the search queue with URLs that will query the user's chosen set of Web search engine sites.

Execute the following concurrent processes.

Independent Process #1
While the search queue is not empty nor the maximum number of URLs visited,

Let $URLtoVisit$ = pop(search queue).
Fetch $URLtoVisit$.

Evaluate *URLtoVisit* using *ScorePage* network.
If score is high enough, insert *URLtoVisit* into the sorted list of best pages found.
Use the score of *URLtoVisit* to improve the predictions of the *ScoreLink* network.

Evaluate the hyperlinks in *URLtoVisit* using *ScoreLink* network (however, only score those links that have not yet been followed this session).
Insert these new URLs into the (sorted) search queue if they fit within its max-length bound.

Independent Process #2
Whenever the user provides additional advice, insert it into the appropriate neural network.

Independent Process #3
Whenever the person rates a fetched page, use this rating to create a training example for the *ScorePage* neural network.

The basic operation of WAWA is heuristic search, with our SCORELINK function providing the sorting heuristic. Rather than solely finding one goal node, though, we collect the 100 pages that SCOREPAGE rates highest. The user can choose to seed the queue of pages to fetch in two ways: either by specifying a set of starting URLs or by providing a simple query that WAWA converts into "query" URLs that are sent to a user-chosen subset of selectable search engine sites (currently AltaVista, Excite, InfoSeek, Lycos, and Yahoo).

Although not mentioned in Table 1, the user may also specify a depth limit that puts an upper bound on the distance the system will wander from the initial URLs.

Before fetching a page (other than those initially in the queue), WAWA had predicted the value of fetching the page, based on the contents of the "referring" page that linked to it. After fetching and analyzing the text, the system will have a better estimate of the page's value to the user. Any differences between the "before" and "after" estimates constitute a *temporal difference* (Sutton 1988) error that can be used by backpropagation (BP)(Rumelhart, Hinton, & Williams 1986) to improve the SCORELINK neural network (see (Shavlik & Eliassi-Rad 1998) for further details).

In addition to the above system-internal method of automatically creating training examples, the user can improve the SCOREPAGE and SCORELINK neural networks in two ways. One, the user can provide additional advice. Observing the system's behavior is likely to invoke thoughts of good additional instructions. WAWA can accept new advice and augment its neural networks at any time. It simply adds to a network additional hidden units that represent the compiled advice, a technique whose effectiveness was demonstrated (Maclin & Shavlik 1996) on several tasks. Providing additional hints can rapidly and drastically improve the performance of WAWA, provided the advice is relevant. (In this paper's experiments we do not evaluate incremental provision of advice, though (Maclin & Shavlik 1996) have done so on their testbeds. They also showed that their algorithm is robust when given "bad" advice, quickly learning to ignore it.)

Although more tedious, the user can also rate pages as a mechanism for providing training examples for use by BP. This can be useful when the user is unable to articulate why the system is misscoring pages and links, but is able to provide better scores. This standard learning-from-labeled-examples methodology has been previously investigated by other researchers, e.g., (Pazzani, Muramatsu, & Billsus 1996), and we will not further discuss this aspect of WAWA in this article. We do conjecture, though, that most of the improvement to WAWA's neural networks, especially to SCOREPAGE, will result from users providing advice. In our personal experience, it is easy to think of simple advice that would require a large number of labeled examples in order to learn purely inductively. Empirical support for these claims is a topic of experiments in progress.

WAWA's use of neural networks means we need a mechanism for processing arbitrarily long Web pages with fixed-sized input vectors. We borrow an idea from NETTALK (Sejnowski & Rosenberg 1987), though our basic unit is a word rather than an (alphabetic) letter as in NETtalk. Wawa slides a fixed-sized window across a page, and most of the features we use to represent a page are defined with respect to the current center of this window. We define the score of a page to be the highest score the ScorePage network produces as it is slid across the page. The value of a hyperlink is computed similarly, but Wawa only slides the ScoreLink network over the *hypertext* associated with that hyperlink. However, in this case the window starts by being centered on the first word in the hypertext, which means the nearby words outside of the hypertext will sometimes fill some of the window positions.

An Overview of WAWA's Advice Language

We next turn to how Wawa represents Web pages and the constructs of its advice language. The input features it extracts (from either HTML or plain text) constitute the primitives in our advice language. Following our description of the basic features, we briefly discuss the more complicated language constructs created from the basic ones.

Extracting Features from Web Pages. A standard representation of text used in information retrieval is the *vector-space model* (Salton 1991) (or the *bag-of-words* representation). The left side of Fig. 1 illustrates this representation. Basically, word order is lost and all that is used is a vector that records the words present on the page, usually scaled according to the number of occurrences and other properties (e.g., TFIDF (Salton 1991)).

Typically, information retrieval systems also discard common ("stop") words and "stem" all words to their root form (e.g., "walked" and "walking" both become "walk") (Salton 1991). Doing so greatly reduces the dimensionality of the problem. Wawa performs these two preprocessing steps.

Instead of solely using the bag-of-words model, we use a richer representation that preserves some word-order information. We also take advantage of the structure of HTML documents when a fetched page is so formatted. First, as partially shown by the first two lines of Table 2, we augment the bag-of-words model, by using several localized bags, some of which are illustrated on the right side of Fig. 1. Besides a bag for all the words on the page, we have word bags for: the title, the page's URL, the window, the left and right-sides of the window, the current hyperlink should the window be inside hypertext, and the current section's title. (Our parser of Web pages records the "parent" title of each word; parent's of words are indicated by the standard ⟨H1⟩ through ⟨H6⟩ constructs of HTML, as well as other indicators such as table captions and table-column headings. Actually, we also have bags for the words in the grandparent and great-grandparent sections, should the current window be nested that deeply.)

(Before continuing, a word of clarification is in order. A Web page has its own URL, while there are also URLs within the page's contents. We refer to the former as *url* and the later cases as *hyperlinks*, in an attempt to

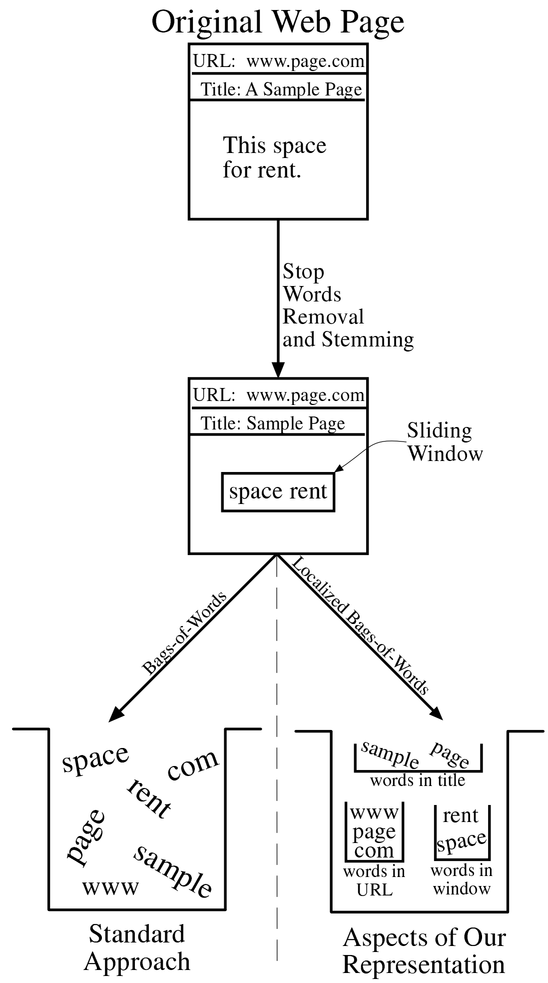

Figure 1: Internally Representing Web Pages

Table 2: Sample Extracted Input Features

`anywhereOnPage(`$\langle word \rangle$`)`
`anywhereInTitle(`$\langle word \rangle$`)`
...
`isNthWordInTitle(`$\langle N \rangle, \langle word \rangle$`)`
...
`isNthWordFromENDofTitle(`$\langle N \rangle, \langle word \rangle$`)`
...
`NthFromENDofURLhostname(`$\langle N \rangle, \langle word \rangle$`)`
...
`leftNwordInWindow(`$\langle N \rangle, \langle word \rangle$`)`
`centerWordInWindow(`$\langle word \rangle$`)`
...
`numberOfWordsInTitle()`
`numberOfAdviceWordsInTitle()`
...
`insideEmphasizedText()`
`timePageWasLastModified()`

reduce confusion.)

In addition to these word bags, we also represent several fixed positions. Besides the obvious case of the positions in the sliding window, we represent the first and last N words (for some fixed N) in the title, the URL, the section titles, etc. Due to its important role in the Web, we also specially represent the last N fields (i.e., delimited by dots) in the *server* portion of URLs and hyperlinks, e.g. `www aaai org` in `http://www.aaai.org/aaai98/papers.html`.

Thus, we use many Boolean-valued features to represent a Web page, ranging from *anywhereOnPage(aardvark)* to *anywhereOnPage(zebra)* to *rightNwordInWindow(3, AAAI)* to *NthFromENDofURLhostname(1, edu)*. (The current version of WAWA does not use any TFIDF methods, due to the manner we compile advice into neural networks.)

Our design leads to a larger number of input features, assuming a typical vocabulary of tens of thousands of words, on the order of a million! However, we sparsely represent these input vectors by only recording those features whose value is "true," taking advantage of an important aspect of neural networks. Specifically, if we represent absent words by zero (and we do), then these zero-valued input features play no role in the forward-propagation phase of neural networks, since weighted sums are used, nor on the BP step, due to the partial derivatives involved.

Beside the input features related to words and their positions on the page, WAWA's input vector also includes various other features, such as the length of the page, the date the page was created (should the page's server provide that info), whether the window is inside emphasized HTML text, the sizes of the various word bags, how many words mentioned in advice are present in the various bags, etc.

One might ask how a learning system can hope to do well in such a large space of input features. Dealing with this many input features would indeed be infeasible if WAWA solely learned from labeled examples. Fortunately, as we shall see, our use of advice means that users indirectly select a subset feature space from this huge implicit input vector. Namely, they indirectly select those features that involve the words appearing in the their advice. (The full input space is still there, but the weights out of input features used in advice have high values, while all other weights have values near zero. Thus, there is the potential for words not mentioned in advice to impact the networks' output, following much BP training.)

WAWA's Complex Predicates. Table 3 contains some of the more complicated predicates that WAWA defines in terms of the basic input features. Some of the advice used in our home-page finder experiment appears in this table. (The *anyof()* construct used in the table is satisfied when any of the listed words is present.)

Our basic instruction says that under some conditions, either increase or decrease the output of one or

Table 3: Sample Advice

(1) `WHEN consecutiveInTitle(`
`anyOf(`*Joseph Joe J.*`)`
Smith's home page`)`
`STRONGLY SUGGEST SHOWING PAGE`

(2) `WHEN hyperlinkEndsWith(`
`anyOf(`*Joseph Joe Smith jsmith*`)` /
`anyOf(`*Joseph Joe Smith jsmith index home homepage my me*`)`
`anyOf(`*htm html /*`))`
`STRONGLY SUGGEST FOLLOWING LINK`

(3) `WHEN NOT(anywhereOnPage(`*Smith*`))` `STRONGLY SUGGEST AVOID SHOWING PAGE`

both of WAWA's neural networks. We will use the first entry in Table 3 to also illustrate how advice is mapped into a neural network. Assume we are seeking Joseph Smith's home page. The intent of rule 1 is as follows. When the system is sliding the window across the page's title, it should look for any of the plausible variants of this person's first name, followed by his last name, followed by apostrophe s, and then the phrase "home page." When these conditions are met, then a large weighted sum should be sent to the output unit of the SCOREPAGE network.

This is accomplished using a variant of the KBANN algorithm (Towell & Shavlik 1994). Rule 1 in Table 3 compiles to five positions (*apostrophe-s* is treated as a separate word) in the sliding window, along with the constraint that the *insideTitle* predicate be true (i.e., have an activation value of 1). WAWA then connects the referenced input units to a newly created hidden unit, using weights of value 5. Next, WAWA sets the bias (i.e., threshold) of the new hidden unit, which has a sigmoidal activation function, such that all the required predicates must be true in order for the weighted sum of its inputs to exceed the bias (27.5) and produce an activation of the hidden unit near 1. (Some additional zero-weighted links are also added to this new hidden unit, to further allow subsequent learning, as is standard in KBANN.)

Finally, WAWA links the hidden unit into the output unit with a weight determined by the strength given in the rule's consequent. WAWA interprets the phrase "suggest showing page" as "increase the page's score."

Unshown variants of rule 1 that we use in our case study allow for the possibility of Smith having a middle name or initial on his home page, by using WAWA's (single-word) "wildcard" symbol, and the possibility his home-page's title is of the form "home page of ... ." Rule 2 shows another useful piece of advice for home-page finding. This one gets compiled into the *NthFromENDofHyperlink()* input features, which are true when the specified word is the *Nth* one from the *end* of the current hyperlink. When there is a match, the weighted sum into the SCORELINK is increased substantially. (Note that WAWA treats the '/' in URLs as a separate word.) Rule 3 shows that advice can also specify when *not* to follow a link or show a page; negations and AVOID instructions become negative weights in the neural networks.

Experiments

This section presents a case study that illustrates the effectiveness and ease of creating a specialized agent on top of the general-purpose WAWA system for a Web-based task. We chose a task already in the literature: creating a home-page finder (Shakes, Langheinrich, & Etzioni 1997). Their AHOY! system uses a technique called Dynamic Reference Sifting, which filters the output of several Web indices and generates new guesses for URLS' when no promising candidates are found.

We wrote a simple interface layered on top of WAWA that asks for whatever relevant information is known about the person whose home page is being sought: first name, possible nicknames, middle name or initial, last name, miscellaneous phrases, and a partial URL (e.g., `edu` or `ibm.com`). We then wrote a short program that reads these fields and creates advice that is sent to WAWA. We also wrote 76 general advice rules related to home-page finding, many of which are slight variants of others (e.g., with and without middle names or initials). Specializing WAWA for this task and creating the initial general advice took only one day, plus we spent parts of another 2-3 days tinkering with the advice using the "training set" we describe below.

Some technical comments are needed to fully understand the details of the following experiments. First, users can retract advice from WAWA's neural networks. Thus, new advice is added and the old erased for each request to find a home page. However, one would also like to learn something in general about home-page finding. This is accomplished via a crude *variable binding* mechanism. WAWA accepts instructions that certain words should be bound to $SpecialWord_N$, and its input vectors contain the Boolean-valued fields $specialWordNisInWindowPositionM$. We thus assign the query person's first name to $SpecialWord_1$, alternate first names (if any) to $SpecialWord_2$ and $SpecialWord_3$, etc. Then we can write general-purpose advice about home-page finding that uses these new Boolean-valued features (hence, rule 1 in Table 3 is actually written using the $SpecialWord_N$ markers and not the names of specific people).

Motivation and Methodology

We randomly selected 100 people from Aha's list of machine learning and case-based reasoning researchers (`www.aic.nrl.navy.mil/~aha/people.html`) to run experiments that evaluate WAWA; to reduce the computational load of our experiments, we limited this to people in the United States. Out of the 100 people selected, we randomly picked 50 of them to train WAWA and used the remaining 50 as our test set. By "training" we mean here that we manually ran the system on these train-set people, manually tinkering our advice before "freezing" the advice and evaluating on the testset. We did not do any BP-based training with the training set.

To judge WAWA's performance in the task of finding home-pages, we provide it with the advice discussed above. It is important to note that we intentionally did *not* provide any advice that is specific to ML, CBR, AI research, etc. WAWA has several options which effect its performance-both in the amount of execution time and the accuracy of its results. We chose small numbers for our parameters, using 105 for the maximum number of pages fetched (which includes the five queries initially sent off to search engines), and 3 as the maximum distance to travel away from the pages returned by the search engines.

We start WAWA by providing it the person's name as given on Aha's Web page, though we partially standardized our examples by using all common variants of first names. (e.g., "Joseph" and "Joe"). WAWA then converts the name into an initial query (see the next paragraph) which is sent to the five search engines mentioned earlier.

We compare the performance of WAWA with the performances of AHOY! and HOTBOT, a search engine not used by WAWA and the one that performed best in the home-page experiments of (Shakes, Langheinrich, & Etzioni 1997). We provided the names in our testset to AHOY! via its Web interface. We ran HOTBOT under two different conditions. The first setting performs a specialized HOTBOT search for people; we use the name given on Aha's page for these queries. In the second variant we provide HOTBOT with a general-purpose disjunctive query, which contains the person's last name as a *required* word, all the likely variants of the person's first name, and the words *"home page"*, *homepage*, and *home-page*. The latter is the same query that WAWA initially sends to its five search engines. For our experiments, we only look at the first 100 pages HOTBOT returns, under the assumption that few people would look further into the results returned by a search engine.

Since people often have different links to their home pages, rather than comparing URLS to those provided on Aha's page, we instead do an exact comparison on the contents of fetched pages to the contents of the page linked to Aha's page. Also, when running WAWA we never fetched any URLS whose server matched that of Aha's page, thereby preventing WAWA from using Aha's page.

The only BP learning WAWA performs in these experiments is that of refining the SCORELINKS function, by automatically creating training examples via temporal-difference learning, as discussed above. We also ran an experimental control where we did no BP'ing.

Results and Discussion

Table 4 lists our results. Besides reporting the percentage of the 50 testset home-pages found, we report the average ordinal position (rank) *given a page is found*, since WAWA, AHOY!, and HOTBOT all return sorted lists. These results provide strong evidence that the version of WAWA, specialized into a home-page finder by adding simple advice, produces a better home-page finder than does the proprietary people-finder created by HOTBOT; with 95% probability, we can say that WAWA's accuracy on this testset is between 69% and 91%. The differences between the first and third rows also suggests that temporal-differencing, BP-refinement of SCORELINKS is effective. Our results also suggest that WAWA performs better than AHOY!, but this difference is not significant at the 95% confidence level.

One cost of using our approach is that we fetch and analyze many Web pages, which takes longer. We have not focused on speed in this study, ignoring such questions as how well we can do fetching only the first N characters of Web pages, only using the capsule summaries search engines return, etc. One relevant statistic we do have is that, given WAWA finds a home page, on average it is the ninth page fetched.

Table 4: Empirical Results

System	**% Found**	**Mean Rank**
WAWA with BP	80%	1.2
AHOY!	74%	1.5
WAWA without BP	70%	1.3
H'BOT person search	66%	12.0
HOTBOT general	44%	15.4

Related Work

Like WAWA, *Syskill and Webert* (Pazzani, Muramatsu, & Billsus 1996), and *WebWatcher* (Joachims, Freitag, & Mitchell 1997) are Web agents that use machine learning techniques. They, respectively, use a Bayesian classifier and an RL-TFIDF hybrid to learn. Unlike WAWA, these systems are unable to accept (and refine) advice, which usually is simple to provide and can lead to better learning that rating or manually visiting many Web pages.

Current and Future Work

We plan to further validate our claim of having appealing Web-based middleware by creating additional testbeds, such as a personalized (and adaptive) electronic newspaper or email filter. Finally, we plan to continue to expand our advice language and to build into WAWA the ability to use information about synonyms (e.g., WORDNET (Miller 1995)) and other knowledge about text. We would also like to add the capability of automatically creating plausible training examples by observing the actions made by users during their ordinary use of WAWA.

Conclusion

We present and evaluate the WAWA system, which provides an appealing approach for creating intelligent agents for the Web. A central aspect of our design is that a theory-refinement system is at the core. This means that the agents built on top of WAWA will be (self) adaptive. It also means that users (both types A and B defined above) may tailor the resulting agents to match their personal preferences by rating the information retrieved. We argue that a promising way to create useful software agents is to involve both the ability to do direct programming (e.g., provide a set of approximate rules) as well as the ability to accept (or automatically create) training examples. Due to the largely unstructured nature and the size of the Web, such a hybrid approach is more appealing than ones solely based on non-adaptive programming languages or relying on users to rate a large number of Web pages. The case study we presented supports our claim.

References

Joachims, T.; Freitag, D.; and Mitchell, T. 1997. Webwatcher: A tour guide for the World Wide Web. In *Proc. IJCAI-97.*

Maclin, R., and Shavlik, J. 1996. Creating advice-taking reinforcement learners. *Machine Learning* 22:251–281.

Miller, G. 1995. WordNet: A lexical database for English. *Communications of the ACM* 38:39–41.

Ourston, D., and Mooney, R. 1994. Theory refinement: Combining analytical and empirical methods. *Artif. Intel.* 66:273–309.

Pazzani, M., and Kibler, D. 1992. The utility of knowledge in inductive learning. *Machine Learning* 9:57–94.

Pazzani, M.; Muramatsu, J.; and Billsus, D. 1996. Identifying interesting web sites. In *Proc. AAAI-96.*

Rumelhart, D.; Hinton, G.; and Williams, R. 1986. Learning representations by back-propagating errors. *Nature* 323:533–536.

Salton, G. 1991. Developments in automatic text retrieval. *Science* 253:974–979.

Sejnowski, T., and Rosenberg, C. 1987. Parallel networks that learn to pronounce English text. *Complex Systems* 1:145–168.

Shakes, J.; Langheinrich, M.; and Etzioni, O. 1997. Dynamic reference sifting: A case study in the homepage domain. In *Proc. of the Sixth International World Wide Web Conference*, 189–200.

Shavlik, J., and Eliassi-Rad, T. 1998. Intelligent agents for Web-based tasks: An advice-taking approach. In *AAAI/ICML Workshop on Learning for Text Categorization.*

Sutton, R. 1988. Learning to predict by the methods of temporal differences. *Machine Learning* 3:9–44.

Towell, G., and Shavlik, J. 1994. Knowledge-based artificial neural networks. *Artif. Intel.* 70:119–165.