The Open A.I. Kit$^{\text{TM}}$: General Machine Learning Modules from Statistical Machine Translation

Daniel J. Walker
The University of Tennessee
Knoxville, TN
USA

Abstract

The Open A.I. Kit implements the major components of Statistical Machine Translation as an accessible, extendable Software Development Kit with broad applicability beyond the field of Machine Translation. The high-level system design policies of the kit embrace the Open Source development model to provide a modular architecture and interface, which may serve as a basis for collaborative research and development for endeavors in Artificial Intelligence.

1 Introduction

In the mid-twentieth century, the emerging field of Artificial Intelligence (AI) embraced the challenge of automatic Machine Translation (MT) of human languages as one of its original goals. Throughout the subsequent decades, MT researchers have adopted, fostered and often originated many techniques used across the numerous sub-disciplines of AI. Examples include algorithms fundamental to the so called "expert system" paradigm for embedding knowledge in intelligent systems as well as the "machine learning" paradigm for constructing systems that automatically acquire problem-domain specific knowledge. Though many of these techniques are frequently taught in universities, well documented in original papers and textbooks, and widely understood by the AI community at large, there remain barriers to empirical research and teaching due to the expense of implementing the necessary software infrastructure. Similarly, development costs often hinder commercial deployment of these algorithms, thereby depriving the general public of potential benefits in the form of consumer goods and services.

As an example, consider the Hidden Markov Model (HMM), which has been applied successfully to several machine learning tasks. Automatic speech recognition research pioneered the use of HMMs for machine learning (Jelinek et al., 1975). Later, (Brown et al., 1993) adapted the HMM language model of speech recognition systems for use in Statistical Machine Translation (SMT). They can be used for other Natural Language Processing (NLP) tasks such as part-of-speech tagging and have broad applicability beyond NLP for general problems related to modeling sequential events. Several textbooks including (Charniak, 1993) and (Russell and Norvig, 1995) provide instructional material to aid in teaching HMMs. However, HMM implementations for a particular problem domain are not readily available for examination, modification, and unrestricted collaborative research. The mathematics of HMMs are easily taught on the blackboard but not in the lab, due largely to the development costs of software infrastructure. Though HMMs are common in commercial speech recognition applications in the telecommunication industry, their lack of use in other markets is attributable, at least in part, to development costs.

Over the past years, the software development model commonly referred to as Open Source has proven its potential to lower the development costs of complicated engineering tasks. However, as famously noted in (Raymond, 2001) the success of Open Source projects is predicated on certain characteristics of the problem addressed by the project, the project's target user group, and the access granted that group to the project's source code. Among these criterion are a specific, tenable need for the project, commonly available tools and informational resources that can be leveraged by the project, and a wide user base with the competence to manipulate the project's implementation. Above all, however, the project must be implemented in terms of open, accessible source code with few technical or legal restrictions to its use, modification, or distribution.

The field of MT provides fertile ground for

Open Source development. The software infrastructure needs of MT practitioners are shared by an even wider audience due to the applicability of machine learning techniques to the NLP and AI communities. Open Source development may draw on abundant resources including standard development tools, libraries, corpora and the rich literature describing machine learning in MT. This paper outlines the system level goals, desiderata, constraints and priorities of a Software Development Kit (SDK) providing general purpose machine learning modules. Together these modules may be used to construct full SMT systems but also constitute an open infrastructure for research and development in Artificial Intelligence at large. The primary goal of the system is to live up to its name: the Open A.I. Kit (OAK).

The following sections provide rationales and descriptions of the high-level system design policies of the kit. Existing systems with comparable functionality are examined, followed by descriptions of the architecture, interface, data, documentation and licensing policies employed by the project. These policies are intended to increase the likelihood of success of the nascent project in terms of providing useful functionality and minimizing impediments to its adoption. The concluding section briefly outlines the status of the project's initial release.

2 Comparable Preexisting Systems

Several comparable software packages inform the high-level design of the Open A.I. Kit.[1] A popular system for language modeling in the NLP community is the CMU-Cambridge toolkit[2] (Clarkson and Rosenfeld, 1997). Also of note are HTK, the Hidden Markov Model toolkit[3] (Odell, 1995), and SRILM[4], the SRI Language Modeling toolkit (Stolcke, 2002). Several SMT toolkits have become available recently including GIZA++[5] (Och and Ney, 2003), the Rewrite Decoder[6] (Germann et al., 2001) and Pharaoh[7] (Koehn, 2004). Though all of these systems are representative of major advancements and have served researchers well over the years, their suitability as an adaptable infrastructure for further research and development is limited.

For example, non-commercial licensing clauses restrict the use of many of the systems to educational and research purposes. Some flatly deny users redistribution rights for any purpose. Restrictions such as these limit the number of users who can work with the system, both directly and via derivative works. The licenses effectively isolate the systems from the broader Open Source community by impeding the projects' ability to incorporate other Open Source projects or to be incorporated into other Open Source projects, which typically do not limit their audience to academia. Additionally, the licenses negatively impact the systems' potential to serve as a common framework for collaboration, thereby seriously decreasing their likelihood of reaping the full benefits of the Open Source development model.[8]

As importantly, the architectures of many of the above systems impose technical barriers to use, modification, and extension. Most of the systems do not include programming interfaces[9] as they are intended to be used as a set of command line utilities. The utilities interact with each other via fixed output formats recorded in files or transmitted through pipes. They may be combined in novel ways via shell scripts. However, this is a much less flexible means of adaptation compared to an accessible Application Programming Interface (API). The SMT systems are particularly specialized for specific tasks: training translation models or decoding. Though they include sub-systems which may have use beyond their specialization, they are geared towards accomplishing an explicit task rather than generality. At the source code level, many of the systems are implemented in well thought out modules. However, as these source level modules are not the intended user interface, they tend to be less documented than the command line utilities. Extracting a source code module for a custom purpose or replac-

[1]It should be noted that the open source project Weka (Witten and Frank, 2005) offers machine learning modules. Currently, Weka primarily consists of decision tree related algorithms for data mining purposes. Though Weka has been used in a variety of applications, it does not include MT capabilities at the present time and therefore does not directly relate to the immediate goals of the Open A.I. Kit.

[2]http://www.speech.cs.cmu.edu/SLM_info.html

[3]http://htk.eng.cam.ac.uk

[4]http://www.speech.sri.com/projects/srilm/

[5]http://www.fjoch.com/GIZA++.html

[6]http://www.isi.edu/licensed-sw/rewrite-decoder/

[7]http://www.isi.edu/licensed-sw/pharaoh/

[8]An exception is GIZA++, which is licensed under the standard GNU General Public License.

[9]An exception is SRILM, which explicitly aims to provide good APIs as its primary interface and is largely successful. Also, HTK provides a programming interface.

ing a module with a user defined substitute are fragile enterprises. These technical hurdles may frustrate user efforts to adapt the systems for original research and development.

3 Architecture and Interface

In the interest of the greatest possible adaptability and extendability, the Open A.I. Kit is architecturally a development library with a programming interface rather than a set of executables or some other more elaborate interactive system. Limiting the set of deliverable functionality to an SDK allows time and energy to be focused on the quality of the algorithm implementations without consideration of specific application details. This comes at the cost of shrinking the pool of immediate users as some people are unable to commit the initial investment of time and energy in programming. Many prefer a ready-made, runnable application. However, as an SDK, OAK equips developers with tools to create such applications. OAK's ultimate aim is to supply an infrastructure that allows the development of a variety of client programs that address the demands of a variety of user groups.

Application independence allows OAK to limit its interaction with the operating system since few machine learning modules require operating system services beyond basic scheduling, heap allocation and file I/O. Issues such as signal handling and error reporting to log files, STDERR, or pop-up windows are reserved as application implementation details with few constraints placed on the client program's options. Independence from the operating system not only increase portability but also eases integration into preexisting programs.

Adaptability and portability need not be sacrificed for efficiency. OAK addresses all three using the C++ programming language for implementation and API. C++ is already commonly used for implementing SMT, NLP, and other AI related systems. Several compilers on many operating systems support the ISO 14882 standard definition of the C++ language and standard library. C++ provides strong interoperability with the C programing language, which in turn can be embedded in numerous other languages as well as common middleware. Optimizing compilers generate efficient native machine code in terms of both CPU and memory consumption. By adopting C++, OAK is able to harness a wide array of available Open Source development tools including third-party utility and scientific libraries, an automatic testing framework, a test coverage tool, profiling and allocation consistency tools, and an inline documentation system. Perhaps, the most compelling reason to adopt C++ is the language's numerous features supporting abstraction and modularity such as facilities for metaprogramming, generic, functional, object-oriented, and procedural programming.

Of these methodologies, the most prevalent employed in OAK is generic programming. Generic programming allows algorithms to operate on arbitrary objects regardless of their type so long as they meet certain abstract syntactic, semantic and complexity constraints commonly referred to as "concepts". This is accomplished via compile-time, parametric polymorphism using the C++ template construct. See (Alexandrescu, 2001) for a complete introduction. Parametric polymorphism is similar to the traditional object-oriented, sub-type polymorphism but does not require compatible types to be related via inheritance, which consequentially avoids the associated runtime overhead and type-safety issues. Generic programming greatly eases the task of adapting OAK to new problem domains, since the abstract generic concepts needed for customizing many machine learning algorithms (such as Iterators, Functors, and Graphs) are already defined in the generic programming literature. Thus, the need to invent and acquaint users with novel abstract interfaces is eliminated to a large degree.

4 Module and Data Interaction

Following generic programming methodologies, the Open A.I. Kit avoids prescribing specific types for data required by algorithms. Instead, algorithms are implemented in terms of generic concepts wherever possible. This allows client programs to reuse specific algorithms on customized types for purposes such as applying the algorithm to a new problem domain or increasing efficiency. This also allows different algorithms to be applied to the same data if the type of the data meets the minimum conceptual requirements of each algorithm.

An obvious example of a set of algorithms suitable for this approach are the translation counts, $c(f|e;\boldsymbol{f},\boldsymbol{e})$, of the IBM translation models 1 through 5 (Brown et al., 1993) for estimating the counts of connections between words in a translated sentence pair. The definition of c

changes from one model to the next. It is defined only in terms of translation probabilities in model 1. In model 2, it is defined in terms of translation and alignment probabilities. Fertility and distortion probabilities are added in model 3, etc. However, the parameters of c remain the same; i.e. the translation counts accumulated during training are indexed by word and sentence pairs regardless of the translation model employed. In OAK, these counts are collected in a trellis and the functions which populate the trellis for each model access it via the Graph concepts defined in the Boost Graph Library (BGL) (Siek et al., 2002). The BGL is an Open Source, high-performance generic library geared toward scientific computing with graphs and sparse matrices. The user need not select the BGL as the trellis implementation and instead may provide an alternate implementation with conforming API. Likewise, new translation count algorithms implemented in terms of the BGL can be interchanged to extend and modify the translation models as desired.

A more interesting example is the HMM, which has applicability beyond SMT. Algorithms related to HMMs such as the Baum-Welch training algorithm, the Forward algorithm for assigning emission probabilities, and the Viterbi search algorithm for identifying the most likely state sequence for a given emission sequence all require the same data elements: an emission alphabet, sets of states and edges denoting the topology of the HMM, and the associated transition and emission probabilities. Again, the BGL Graph concepts provide a vocabulary for algorithms to query, manipulate, and modify the various properties of HMMs. By specifying the emissions and topology of the HMM the user can apply the algorithms to different problem domains. The generality of this framework does not preclude practical efficiency concerns. For example, as illustrated in (Jelinek, 1997, 62-69), assigning emission and transition probabilities need not hinge on the Baum-Welch algorithm when a large enough sample set is available to estimate the probabilities directly from empirical counts. In such a scenario, one may employ a linear interpolation of probability estimates, which are parameterized on successively smaller histories of the sequence of events being modeled. This method, called deleted interpolation, implies a specific HMM topology, which OAK provides via the same Graph concepts used by the other HMM algorithms. In this way the Forward and Viterbi algorithms may be applied to HMMs based on either Baum-Welch or deleted interpolation. The framework also allows for further optimizations such as parameter tieing and the structural elimination of zero probability transitions.

The sequence of events modeled by the HMM, be they words or otherwise, are represented as ranges delimited by objects implementing the Iterator concept. Other concepts used by the ISO C++ standard library, such as Containers, are leveraged wherever appropriate. Data abstraction is extended to persistent data in the form of file formats. OAK does not specify persistent formats for any of the data it uses or generates but instead draws on the serialization methodologies of C++ to allow the user to serialize individual data elements and choose the format by which their organization is recorded on disk. This allows OAK to interoperate with emerging standard file formats, proprietary formats, or any custom file type of the user's choosing.

5 Documentation and Support

The utility of the Open A.I. Kit largely depends on the quality of the API documentation. As the API constitutes the primary intended user interface, all information pertinent to developing with OAK is presented for the user's convenience including function parameters, return values, type requirements, pre- and post-conditions, exception specifications, and run-time complexity. An effort is also made to explain algorithms, their origins and potential applications, trade-offs between similar algorithms along with bibliographical references to original works and page references in prominent text-books.

Perhaps no explanation is more clarifying than a concrete example. As such, OAK offers basic, complete sample programs for each major module. In order to minimize the number of prerequisites needed to understand the modules, the sample programs are implemented in simple terms with no overt dependencies beyond OAK and the standard C++ library whenever possible. A sample program can also serve as a template or starting point for the user to begin the development process. Thus, users may start development from a small, functioning, runnable illustration and incrementally expand and customize it for their own purposes.

In addition to user documentation, a public mailing list will be maintained to answer questions from OAK users and to coordinate OAK development. Also, the list may be used to report problems, post patches for bugs and portability issues, as well as discuss enhancements and new features. The mailing list can provide a forum for OAK users and developers to learn from each other's experiences, provide feedback and advice, and generally collaborate to improve OAK and related projects.

6 Licensing and Distribution

The Open A.I. Kit is intended to provide accessible, modifiable, redistributable, open source code as a reliable infrastructure for Artificial Intelligence related endeavors by engineers, researchers, teachers and students alike.

As such, OAK is licensed under very simple terms that place few restrictions as to its use, modification and distribution for academic, non-profit or commercial purposes. In synopses, the only restrictions are as follows: redistribution of OAK in any form must include the original copyright notice and disclaimer and must not use the names of contributers or the OAK trademarks to endorse products without prior permission. This is to ensure both that future recipients are aware of the original contributers' copyrights and that contributers to OAK are not subjected to unfair treatment.

Specifically, OAK adopts the terms of the University of California, Berkeley Software Distribution (BSD) license as certified by the Open Source Initiative[10]. The BSD license was chosen primarily because, at just over 200 words, it is simple enough that no legal expertise is required to understand its terms and conditions. Also, the BSD license is compatible with the GNU General Public License[11] but has less restrictions pertaining to derivative works. Thus, few legal hurdles circumscribe OAK's interoperability with third-party Open Source projects. For more information regarding the BSD license or Open Source licensing in general see (St. Laurent, 2004).

All future official OAK releases will be distributed on the Internet as compressed archive files via File Transfer Protocol (FTP). The official OAK source code tree will be available continually via the GNU Concurrent Versions System (CVS). Information about obtaining OAK will be available on the World Wide Web at http://www.openaikit.org.

[10]http://www.opensource.org

[11]http://www.gnu.org/licenses

7 Current Status and Conclusion

The public introduction of the Open A.I. Kit corresponds with the publication of this paper. As such, the success or failure of the policies described above cannot be corroborated by experience as of yet. However, explicitly documenting and publicly publishing the system-level design decisions establishes an initial foundation that can serve the project well. Also, by announcing the project in a forum such as the Machine Translation Summit X workshop on Open Source MT, the project benefits early on from the commentary and criticism of an expert audience.

However, above all else, the functionality of the first release of a new open source project is paramount in determining the project's future use and development. Though the project need not fulfill all of its goals initially, it must at least provided a minimum amount of functionality to meet some of the needs of users. Accordingly, the first release of OAK does not include a plethora of AI related algorithms, nor even all of the desirable techniques for MT, but only the minimal algorithms needed to construct an SMT system: namely, an n-gram language model based on HMMs, IBM translation models 1 and 2, and a simple decoder.

Through specific high-level system design policies, the Open A.I. Kit encapsulates these SMT machine learning techniques in a general framework from which researchers and developers in the wider field of AI may benefit. By addressing the specific needs of SMT in generic modules, leveraging available development tools and literature, and furnishing the implementation in well documented, open source code with minimal technical or legal limitations, OAK may be employed as an infrastructure to lower the costs of research, development and education in MT, NLP and AI.

References

Andrei Alexandrescu. 2001. *Modern C++ Design: Generic Programming and Design Patterns Applied.* Addison-Wesley, Boston, Massachusetts.

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation. *Computational Linguistics*, 19(2):263–313.

Eugene Charniak. 1993. *Statistical Language Learning*. The MIT Press, Cambridge, Massachusetts.

Philip Clarkson and Roni Rosenfeld. 1997. Statistical language modeling using the cmu-cambridge toolkit. *Proceedings of ESCA Eurospeech 97*, 1:2707–2710.

Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. 2001. Fast decoding and optimal decoding for machine translation. *Proceedings of the 39th meeting of the Association of Computational Linguistics*, pages 228–235.

Fredrick Jelinek, Lalit R. Bahl, and Robert L. Mercer. 1975. Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions on Information Theory*, 21:250–256.

Fredrick Jelinek. 1997. *Statistical Methods for Speech Recognition*. The MIT Press, Cambridge, Massachusetts.

Philipp Koehn. 2004. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. *Proceedings of the 6th Conference of the Association for Machine Translation in the Americas*, pages 115–124.

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Julian J. Odell. 1995. *Lattice and Language Model Toolkit Reference Manual*. Entropic Cambridge Research Laboratories, Inc., Cambridge, England.

Eric S. Raymond. 2001. *The Cathedral and the Bazaar*. O'Reilly, Sebastopol, California.

Stuart J. Russell and Peter Norvig. 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, New Jersey.

Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. 2002. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley, Boston, Massachusetts.

Andrew M. St. Laurent. 2004. *Understanding Open Source and Free Software Licensing*. O'Reilly, Sebastopol, California.

Andreas Stolcke. 2002. Srilm - an extensible language modeling toolkit. *Proceedings of the International Conference on Spoken Language Processing*, 2:901–904.

Ian H. Witten and Eibe Frank. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, California.