Software Libraries for PGMs

Kevin Rothi
Prepared for Dr. Rina Dechter's Spring 2018 UCI ICS 276 Course

When we look at the software libraries available for implementing deep learning and neural networks, the clear market winner is Google's TensorFlow[1]. TensorFlow has become ubiquitous in this space, with a few competitors finding niches in which they thrive. For a comparison of these libraries, see this footnote[2].

I bring this up because the thesis of this paper is that the situation for libraries for implementing PGMs is quite different. There is no clear market leader. There is no TensorFlow analog for graphical models. Instead, what we see in this space is a fracas of small libraries, each seeming to vie for the attention of researchers, students, and interested observers.

I propose this question: Are graphical models as mature in theory as neural networks? Is the application space for graphical models as rich as the application space for neural networks? If the answer to these questions is yes, then what's holding PGMs back from being as widely accepted, adopted, and implemented in commercial applications as neural networks?

I suspect that it's the deficit in software solutions for PGMs that has held it back from achieving the kind of success we see with neural networks. Another contributing factor for neural network's popularity may be that the applications where neural networks have achieved the greatest degree of success tend to be spectacular in the sense that the results are quite striking. Images require little interpretation. This may have contributed to the high degree of interest in neural networks, but this is a flawed argument because graphical models can also be applied to the many of these problems. Therefore, it's reasonable to conclude that the lack of high quality software for PGMs is a significant factor in their adoption rate lagging behind other AI techniques.

When we consider the fragmentation of the software libraries that can be used for PGMs, several observations become apparent. The first issue is that there has historically been a problem with data ingestion. The data that is used to populate a PGM has lacked a consistent data format. This isn't an insurmountable issue. The data that neural networks operate on, for example, isn't always consistent in its format. What is true, however, is that the lack of a universal format has encouraged researchers to develop their own toolsets on an individual basis. This has resulted in a seemingly perpetual reinvention of the wheel as library after library has implemented the same algorithms to tackle different problems based solely on the need to handle disparate datasets. This reflects a lack of prowess in engineering, however. What is needed is a modularization of code. The data ingestion code should be loosely coupled with the actual algorithms. We see this kind of architecture in libraries like TensorFlow.

The second observation we can make about software libraries for graphical

---

[1] https://www.tensorflow.org/

[2] https://www.microway.com/hpc-tech-tips/deep-learning-frameworks-survey-tensorflow-torch-theano-caffe-neon-ibm-machine-learning-stack/

models is that there is a spectrum of usability to generality. The more general a library is, the harder it is to use. Conversely, when a library is easy to use, it typically sacrifices generality in the sense that the variety of models and algorithms is reduced.

The Python library pgmpy[3] lies on the usable side of this spectrum. It offers high level abstractions that are easy to interact with. The code is often quite concise and easy to read. What is sacrificed is efficiency since Python is an interpreted language, and the library offers fewer algorithms and models than some of its more heavyweight counterparts.

The C++ library OpenGM2[4] lies on the general side of this spectrum. It boasts a seemingly endless supply of algorithms which can be leveraged against a given problem, and since C++ is a compiled language, it executes much faster than a language like Python. What you pay for this power is that the library is harder to use, requiring much more from the user to get up and running.

What can be done about this apparent catch 22? I make the following suggestion: Embrace a hierarchical approach. We see this with TensorFlow and Keras, for example. TensorFlow is complicated, leaving many details up the the user. Luckily, Keras is an abstraction over TensorFlow that makes simple tasks easy and hard tasks possible. What PGMs need to become more mainstream is for something like TensorFlow to emerge, and for something like Keras to be built on top of it. This would provide the user with options. If the application demands a complex solution, opt for the lower level of the hierarchy. If the application lends itself to a simpler solution, use the high level tools. This would facilitate rapid prototyping in the initial stages of implementation and a detailed, tailored final solution.

TensorFlow's predecessor, a closed source proprietary machine learning system called DistBelief, evolved into TensorFlow at Google Brain. This process took 6 years.[5] It wasn't an overnight success. It's unreasonable to think that a software library for graphical models will rise to prominence overnight, but what is reasonable is to think that somewhere there might be a library in an incubation stage which will rise to such prominence, and, if that is the case, then it's likely to happen in one of two places; either at a university or at a major software company.

For this to happen, however, I believe the software needs to meet certain criteria. First, the software needs to be open source and available under a license which would permit it to be used for commercial applications. Second, the software would need to be implemented in one of the more popular programming languages, such as Python. Finally, the institution which has developed the software needs to both use the software internally and promote its use externally. This includes providing extensive documentation, tutorials, and marketing so that the relevant users know of the software, its use cases, and how to use it.

Pgmpy and OpenGM2 seem to be the most likely candidates at this time, but neither seem to be reaching the critical mass necessary for widespread adoption. Pgmpy is limited in terms of its applicability due to its lightweight nature, which is paradoxically its most valuable asset. OpenGM2 suffers from the opposite issue; it

---

[3] http://pgmpy.org/

[4] http://hciweb2.iwr.uni-heidelberg.de/opengm/

[5] https://en.wikipedia.org/wiki/TensorFlow

appears to be collapsing under its own weight. Please see the accompanying slides for more information about these libraries, such as their authors.

Time will tell what will happen in this space. Will PGMs ever get its TensorFlow? Or will PGMs be fated to see the same algorithms implemented over and over again? The answer to these questions is unclear at this time, but I'm optimistic that PGMs will eventually become as mainstream as deep learning.

The 2011 best selling "Thinking Fast and Slow", by Nobel Prize Laureate Daniel Kahneman, presents a fascinating glimpse into how the mind works. The core claim of the book is that the mind actually has two distinct systems, what the author calls "System 1" and "System 2". The book's Wikipedia entry provides a clear description of both, "The central thesis is a dichotomy between two modes of thought: 'System 1' is fast, instinctive and emotional; 'System 2' is slower, more deliberative, and more logical."[6]

Some examples of the different types of problems the respective systems perform well at are that System 1 handles tasks like reading text on a billboard or driving a car on an empty road. System 2 is responsible for tasks such as calculating the price to quality ratio of two computer systems or determining the validity of a complex logical reasoning task. This is *not* pure conjecture. There is significant evidence to support the idea that there are, factually, different systems in the mind that handle these tasks.

Perhaps you can already see where I'm going with this. System 1 solves the types of problems that deep learning and neural networks have historically been proficient at, while System 2 is needed to solve the kinds of problems we hope to be able to solve with graphical models. If we ever want to replicate human like intelligence, we will need both.

This is a bold claim, but I believe there is sufficient reason to believe it to be true, and I don't think I'm alone in this thinking. The probabilistic programming language Edward[7] is an abstraction over TensorFlow, and it combines neural networks with probabilistic concepts. This is a promising new direction, and I'm very interested to see how this language evolves in the future. It could represent a paradigm shift in how we think about artificial intelligence.

As a species, we've devoted an enormous amount of time and energy into engineering systems that can solve System 1 problems well. Speech recognition and image recognition have come a long way due to advances in this space, but the time is on the horizon when we will need to turn our attention to solving the System 2 problems that obfuscate the path to true artificial intelligence. Graphical models and probabilistic reasoning are the best tools we have at this time to make progress on these problems, and I believe that once the AI community realizes and accepts this, we will see graphical models become just as popular as deep learning and neural networks, and the software tools we have will reflect this.

---

[6] https://en.wikipedia.org/wiki/Thinking,_Fast_and_Slow

[7] http://edwardlib.org/