Simple Reasoning and Knowledge States in a LSTM-Based Agent

Ryan Mukai

June 24, 2020

1 Introduction

This article focuses on the development of a simple form of self-awareness in an LSTM-based agent. We present an agent capable of displaying its knowledge state and of answering questions based on its state of knowledge. If an agent is unable to answer a question, it will indicate this and request assistance from another agent. The other agent, upon receiving such a request, provides data from its knowledge state to aid the requester in its goal of finding an answer. The goals of this work are:

1. Having agents maintain a concept of a propositional sentence as a unit of thought.
2. Having agents possess a concept of their own knowledge in the sense of being able to dump their knowledge state on request.
3. Having agents possess a concept of their own knowledge in the sense of being aware of not being able to answer a question and asking for help.
4. Having agents take advantage of basic self-knowledge to cooperate in order to solve a simple problem.
5. Having agents be aware of contradictions in their knowledge. In propositional logic, a contradiction (i.e. False) can imply anything, and the agent should warn of a contradiction in its knowledge if it recognizes one.

Hence, our definition of self-awareness is a very narrow one that focuses on giving agents the ability to know certain things about their own knowledge state, and it is clear that this is far narrower than anything approaching consciousness. The focus of this article is not on creating a general-purpose reasoning agent since existing literature already covers this in much depth. The focus is instead on having neural network-based agents base their actions on their own understanding of their state of knowledge and use that knowledge to cooperate to solve a problem.

Much work has occurred the field of neural networks and symbolic reasoning. A good survey of the field, with numerous references to recent developments, is [3], which is also a nice introduction to the field. A good but older introduction to neural-symbolic learning is [7], and another overview of the field is in [4]. Reference [8] describes many approaches used in neuro-symbolic learning systems, including the SHRUTI spiking neural network [12], the Core Method for learning first order logic programs [2], topos theory for learning models of predicate logical theories [6], modal and temporal reasoning with neural networks [11], and multi-valued logic and neural networks [9], and many other approaches have been described in the literature. The use of LSTMs for neuro-symbolic tasks also has a precedent, and one example is [10]. In [**?**], the authors introduce PossibleWorldNets capable of achieving very good performance on propositional logic problems. A very interesting example of symbolic mathematical processing is [**?**] in which a neural network was able to beat Mathematica on complex symbolic mathematical tasks.

2 The Problem Agents are Trained to Solve

Agents are trained to reason on sentences involving syllogisms and the logical not(˜), and (&), or (—), exclusive-or (ˆ), implication (==>), and bidirectional implication (<==>) operators. Throughout this article propositions, which can be either True or False, are denoted by the capital letters A through J inclusive. We train agents to handle simple problems of the following forms.

1. Given the status of a propositional variable A as true, false, or unknown, respond to a question about its value.
2. Given a simple sentence such as A =>B and given A is true, answer that B is true if asked about B.
3. Given a simple sentence such as A =>Band given B is false, answer that A is false if asked about A.
4. nGiven the value of a and no other information, if asked about B indicate B is unknown.
5. Given contradictory statements, warn that the knowledge base is contradictory.
6. In addition to sentences of the form A =>B, we include sentences of the form (A — B), (A ˆB), and (A & B). We also use ˜to denote logical not.

3 The Agent

The core of the agent consists of an LSTM neural network created using Keras [5] with Tensorflow [1] as a backend.

The network is a sequence-to-sequence network of the type that has been used for language translation, and the source code is based directly upon the sequence-to-sequence tutorial found at https://keras.io/examples/lstm_seq2seq/ although, instead of translating from English to French, we are using the sequence-to-sequence model to answer logical questions.

The inputs are logic sentences represented with one-hot encoding. These may contain repeated sentences. We use an example to illustrate. Suppose the list of sentences below, one-hot encoded, is our input.

1. A
2. A =>B
3. A =>B
4. A

When presented with the above, the network is expected to be able to recall the sentences in the order they were presented, without any repetitions, as follows:

1. A
2. A =>B

It is possible during the course of operations that some statements presented to the agent may contain one or more repetitions as shown above, and the agent is trained, upon receiving the query HELP, to respond by condensing the knowledge list into just two sentences as shown above. This serves two purposes:

1. Agents presented with lots of potentially repeating information must have a concept of what a logical sentence is, with the ability to condense repeat occurrences and return exactly the sentences that belong to the knowledge base. This implies some knowledge of a sentence as a unit of reason.
2. When an agent is asked for help, it should be able to supply its knowledge base in response to such a query from another agent.

If the word HELP appears, then the environment sets a help flag and send the question HELP to another agent. The other agents response, the dump of its knowledge base, is sent back to the first agent, and the first agent tries to answer the question again. If the word HELP does not appear, then the other agent is not queried for its knowledge since the first agents answer is sufficient.

3.1 Agent Cooperation: Example 1

Suppose we have two agents, Agent 1 and Agent 2. Agent 1 is given the following knowledge:

- A =>B

Agent 2 is given the following knowledge:

- A

If we ask Agent 1 What is B ?, the following will occur.

1. Agent 1 indicates B is unknown and requests help.
2. The simulation environment, in response to seeing Agent 1 request help, sends a help query to Agent 2.
3. Agent 2 dumps its knowledge that A is true.
4. This knowledge is added to the knowledge base of Agent 1.
5. Agent 1 runs again and is able to finally conclude b is true.

In this example, we note the following:

1. Agent 1 lacks adequate information to determine B and indicates a lack of knowledge of B.
2. Agent 1 will, without knowledge of B, make a request for help.
3. Agent 2 responds to a help query by dumping its knowledge state.
4. This allows Agent 1 to find the answer.

Agents only request help if they do not know the answer but will otherwise answer the question. This implies that an agent has some knowledge of its own knowledge state. If one knows the answer, one answers the question. If one does not know an answer, one indicates this is so and requests help. Hence,

1. Agents have a concept of knowing whether or not they know an answer and requesting help when they do not.
2. Agents also have knowledge states that they can dump if they are asked for help.
3. Moreover, in terms of maintaining an internal knowledge state, an agent knows how to purge repeat sentences and understands, to some extent, the idea of a sentence as a unit of knowledge since it can repeat units of knowledge when asked.

3.2 A Second Example

Agent 1 is given the following knowledge:

- A =>B

Agent 2 is given the following knowledge:

- ˜B

If we ask Agent 1 What is A?, the following will occur.

1. Agent 1 indicates a is unknown and requests help.
2. The simulation environment, in response to seeing Agent 1 request help, sends a help query to Agent 2.
3. Agent 2 dumps its knowledge ˜B.
4. This knowledge is added to the knowledge base of Agent 1.
5. Agent 1 runs again and is able to finally conclude A is false.

4 Key Results

The network was found to yield an error rate of slightly less than 1% on the validation data set logic_data_extended.tsv contained in the gzip tar archive at https://beta1-demo.s3.amazonaws.com/beta_demo_data_files.tgz, which we have made publicly available to facilitate peer review. In this tab-separated file, the columns are defined as follows:

1. The first column, or column 0, contains a set of logical statements separated by the logical and (&) operator. This is the input data to the network consisting of logical statements and a question about a variable.
2. The second column, or column 1, contains the same set condensed, without any repetitions. This is used as target data when the query HELP is presented, teaching the network how to create a condensed form of knowledge and teaching the network to recognize logical sentences as units.
3. The fourth column, or column 3, contains the answer to the question in column 0, which could be: True, False, Contradictory, or Unknown HELP!

The network is trained to respond to the question with the answer from column 3 (fourth column). It is also trained to respond to the sentences, but with the question replaced by HELP, by yielding a condensed form of knowledge from column 1 (second column).

5 Some Limitations

In the present implementation, an agent is a Python object, and the agents knowledge base is still maintained as a Python list of sentences presented to a newly initialized LSTM on each call to the agent. Each run of the neural network involves presenting the sentences and the question (where help is one possible question) to the network and subsequently obtaining its output. Maintenance of LSTM state across calls could be used in future editions of the agent to eliminate the need for maintaining an object with a list of sentences. Hence, the memory of an agent is still a Python list, although a sequence-to-sequence network performs operations like distilling the list to the essentials or answering questions about the sentences in the list. Given that our focus is on agents having a (limited) understanding of their own knowledge state, which is a basic form of self-awareness, much less emphasis has been placed on attempting to achieve general propositional reasoning. An error rate of about 1% has been achieved on validation data, but the agent may perform poorly if presented with very long chains of reasoning of more than three or four sentences or with combinations of sentences that differ too much from what was seen in training. Users are invited to view the training and validation sets in the Gzipped tar file at https://beta1-demo.s3.amazonaws.com/beta_demo_nn_and_dictionaries.tgz to view the kinds of problems we trained the agent to solve.

6 Demo and Source Code

There is a demonstration of this system available for peer review. One may visit the Google CoLab notebook at https://colab.research.google.com/drive/1_X9NGH3KzFmJYRYNxjOWGH1Iw in order to run the demo. In the demo, logical operators are represented as follows:

1. Logical NOT: ~
2. Logical AND: &
3. Logical OR: —
4. Logical XOR: ^
5. Implication: ==>
6. Reverse implication: <==
7. If and only if: <=>

Each of the two agents is provided with a list of sentences, and each sentence may only contain one or two propositions. Run each cell to install the code and the data files in Part 1 of the notebook. In Part 2, look for the code cell with this code: run_dual_agent_demo(agent_1_initial_knowledge_list = [~A, B ==>A, A & D],agent_2_initial_knowledge_list = [B ^C],question_for_agent_1 = What is

C ?) Feel free to experiment with different input sentences. Note that the code contains functions to check the syntax of the statements you provide. The source code is available at the above link and at https://github.com/CoderRyan800/redesigned-octo-goggles.git

We would like to give credit to two important sources for a great deal of our source code:

1. The Python repository for the well-known textbook Artificial Intelligence: A Modern Approach is located at https://github.com/aimacode/aima-python and we have borrowed propositional logic code from that repository in order to create training and validation data for our agents. We also use code from this repo to perform syntax checks on user inputs to the agents in our web demo.

2. The Keras documentation at https://keras.io/examples/lstm_seq2seq/ and at https://keras.io/examples/lstm_seq2seq_restore/ provides example code that demonstrates sequence-to-sequence models, and we have also utilized this in a somewhat modified form to create and to train our own networks.

7 Summary

A deep learning neural network using LSTM and Dense layers with the ability to perform a very limited symbolic propositional reasoning task has been created and demonstrated. Agents with the ability to answer a question if the answer is known and with the ability to recognize a lack of knowledge and ask for help have been built, and these agents can also respond to requests for help with a knowledge state dump. The concept of learning a knowledge state has been demonstrated in a simple case.

This work is a starting point. Possible future directions include, but are not limited to:

1. Adding a broader range of problems that can be solved, especially given that we do not solve long chains of propositional sentences.

2. Having agents dump knowledge more selectively rather than giving a full knowledge dump when asked for help.

3. Changing from the existing LSTM-based seq2seq architecture to the much newer Transformer based architecture, introduced in the paper Attention is All You Need [**?**] and described very nicely in the superb article The Illustrated Transformer [**?**].

8 Acknowledgements

The author gratefully acknowledges the valuable feedback and advice received from Zaid Towfic, who pointed out several deficiencies in the first publication of this article.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Sebastian Bader, Pascal Hitzler, and Steffen Hölldobler. *Perspectives of Neural-Symbolic Integration*, chapter The Core Method: Connectionist Model Generation for First-Order Logic Programs, pages 205–232. In Hammer and Hitzler [8], 2007.

[3] Tarek R. Besold, Artur S. d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro M. Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luís C. Lamb, Daniel Lowd, Priscila Machado Vieira Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. *CoRR*, abs/1711.03902, 2017.

[4] Tarek R. Besold and Kai-Uwe Künberger. Towards integrated neural-symbolic systems for human-level ai: Two research programs helping to bridge the gaps. *Biologically Inspired Cognitive Architectures*, 14:97–110, 2015.

[5] François Chollet et al. Keras. https://keras.io, 2015.

[6] Artur S. d'Avila Garcez. *Perspectives of Neural-Symbolic Integration*, chapter Learning Models of Predicate Logical Theories with Neural Networks Based on Topos Theory, pages 233–264. In Hammer and Hitzler [8], 2007.

[7] Artur S. d'Avila Garcez, Krysia B. Broda, and Dov M. Gabbay. *Neuro-Symbolic Learning Systems*. Springer, London, 2002.

[8] Barbara Hammer and Pascal Hitzler, editors. *Perspectives of Neural-Symbolic Integration*. Springer-Verlag, Berlin Heidelberg, 2007.

[9] Barbara Hammer and Pascal Hitzler, editors. *Perspectives of Neural-Symbolic Integration*, chapter Connectionist Representation of Multi-Valued Logic Programs, pages 283–313. In Hammer and Hitzler [8], 2007.

[10] Qiuyuan Huang, Paul Smolensky, Xiaodong He, Li Deng, and Dapeng Wu. A neural-symbolic approach to design of captcha. *arXiv*, 2017. arXiv:1710.11475v2 [cs.CL].

[11] Ekaterina Komendantskaya, Máire Lane, and Anthony Karel Seda. *Perspectives of Neural-Symbolic Integration*, chapter Advances in Neural-Symbolic Learning Systems: Modal and Temporal Reasoning, pages 265–282. In Hammer and Hitzler [8], 2007.

[12] Lokendra Shastri. *Perspectives of Neural-Symbolic Integration*, chapter SHRUTI: A Neurally Motivated Architecture for Rapid, Scalable Inference, pages 183–204. In Hammer and Hitzler [8], 2007.