Simple Reasoning and Knowledge States in a LSTM-Based Agent

Ryan Mukai

14 December 2018

1 Introduction

This paper focuses on solving a simple propositional logic problem using LSTM neural network based agents. 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 requestor in its goal of finding an answer.

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 [13], 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].

The focus of this paper differs in that we have LSTM agents base their actions on their state of knowledge and cooperate to solve a problem. We do not, however, pursue a direct relationship between neural connections, weights, and symbolic representations as in most of the literature of the field [3]. Also, we are not aiming for logical completeness and are instead focusing on the following topics:

1. Use of LSTMs for simple logical reasoning, motivated by LSTM memory capabilities and by desire to possibly apply this to chatbots which maintain state and interact with users.

2. Having LSTMs maintain a limited concept of a logical sentence as a unit to be stored in memory for later recall.

3. Having LSTMs have a limited concept of their own knowledge in the sense of being able to dump their knowledge state on request.

4. Having LSTMs have a limited concept of their own knowledge in the sense of being aware of not being able to answer a question and asking for help.

5. Having a simple scenario of LSTM agents that can cooperate and share knowledge to solve a simple problem.

We create two simple LSTM-based agents to solve propositional logic problems involving simple syllogisms. Each agent is given a simple knowledge base and is trained to respond to questions based on its available knowledge. If it lacks an answer, the agent is trained to ask for help. Each agent is also trained to respond to a request for help by simply dumping its knowledge base, possibly yielding clues to help a requestor in finding an answer. An agent that is able to answer a question does not request help.

In this paper, all reasoning involves propositional logic.

2 The Problem Agents are Trained to Solve

Syllogisms are a simple and well known form in logical reasoning, and 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 $a \rightarrow b$ and given a is true, answer that b is true if asked about b.

3. Given $a \rightarrow b$ and given b is false, answer that a is false if asked about a.

4. Given the value of a and no other information, if asked about b indicate b is unknown.

5. Given the statement $a \rightarrow b$ and nothing else, if asked about either a or b, conclude the requested value is unknown.

6. Given no statements at all or given only statements that say nothing about a, if asked about a, conclude a is unknown.

The above involves simple propositional reasoning about simple syllogisms. In addition, we also want an agent to exhibit these behaviors.

1. If the answer to a question is unknown, then ask another agent for help.

2. If the answer to a question is known, given the correct answer and do not ask for any help, since help is not needed.

3. If a request for help is received, dump the statements in one's knowledge base. This may aid the requestor in answering the original question.

The second list of items above constitutes a very simple form of knowledge of internal state. Here, an agent should be aware of whether it knows the right answer, which is either true or false, or whether it needs help because the answer is unknown. An agent should also be aware of the statements in its knowledge base and be capable of supplying them when asked for help.

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 was created using the Keras functional API as follows.

- `x1 = LSTM(256, return_sequences=True)(inputs)`
- `x2 = LSTM(256, return_sequences=True)(x1)`
- `x3 = (Dense(Y1[0,0].size))(x1)`
- `y1 = Activation('softmax')(x3)`
- `x4 = (Dense(Y1[0,0].size))(x2)`
- `y2 = Activation('softmax')(x4)`
- `model = Model(inputs=inputs, outputs=[y1, y2])`

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

1. a is unknown
2. b is unknown
3. a is unknown
4. $a \rightarrow b$
5. a is true
6. b is unknown

The output y1 consists of the following sentences, represented using one-hot encoding.

1. b is unknown
2. $a \rightarrow b$
3. a is true

Here we note that the sentence "a is unknown" represents an obsolete knowledge state that is replaced by "a is true". Note also that the agent still has "b is unknown" stored as it does not, at this stage, conclude that b is true. For the purposes of $y1$ it is merely reciting the sentences in the order in which they were learned with obsolete unknowns removed and in the same order of learning. Note that repetitions are purged. Hence, the purpose of training the first of two LSTM layers and a corresponding Dense output layer on y1 is to teach the system to develop both a concept of replacing old unknowns with new knowledge and, importantly, to teach the system the concept of sentences as units of information or knowledge.

No matter what question is asked, the output $y1$ is always a simple recital of knowledge sentences, without drawing any logical conclusions.

The output $y2$ is, however, dependent on the question asked.

- Question "what is a ?" results in "a is true"
- Question "what is b ?" results in "b is true"
- Question "what is c ?" results in "c is unknown . help"
- Question "help" results in "b is unknown . $a \rightarrow b$. a is true"

Note that the response to "help" makes $y2$ the same as $y1$. Here, $y1$ can be thought of as an "internal" or "private" output. By contrast, $y2$ can be thought of as the "external" or "public" output.

The simulation environment is designed to only have an output if the network does not have the keyword "help" in its output. But if the word "help" appears, then the environment sets a help flag and send the question "help" to another agent. The other agent's response, the dump of its knowledge base, is sent back to the first agent, and the first agent tries to answer the question again.

The present system assumes that all knowledge presented to it is logically consistent. Although the statement "a is unknown" can be replaced with the statement "a is true" or with the statement "a is false", the present networks are not trained to handle situations in which a previous belief is reversed. In particular, they are not trained for the case in which a previous belief such as "a is true" is replaced by a belief that "a is false" or vice versa. There exist systems that are able to process such updates in knowledge but that is outside of the scope of the present work, which focuses on teaching LSTMs to update their knowledge assuming no inconsistencies and is focused on awareness of knowledge state and on cooperation between two agents.

3.1 Agent Cooperation: Example 1

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

- $a \rightarrow b$

Agent 2 is given the following knowledge:

- a is true

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, send 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 a limited 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 limited 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 obsolete known sentences of the form "a is unknown" with "a is true" or with "a is false", implying the ability to understand that once one knows something is true or false then it is not unknown any more.

3.2 A Second Example

Agent 1 is given the following knowledge:

- $a \rightarrow b$

Agent 2 is given the following knowledge:

- b is false

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, send a help query to Agent 2.
3. Agent 2 dumps its knowledge that b is false.
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.

In the next Section, we describe the training templates used in this simple agent.

4 Training the Agent

At the heart of the agent we have a deep learning neural network with LSTM and Dense layers as described in Section 3. A network must be trained to respond to questions that are presented to it given a certain knowledge base.

Each training template contains a list of statements, a question, and an answer. Here is an example:

```
training_template = {
  'statement_list' : ['if pa1 then pa2', 'pa2 is false'],
  'question' : 'what is pa1 ?',
  'answer' : 'pa1 is false'
}
```

In the above example, "pa1" and "pa2" are placeholders. When the training examples are generated, they will be replaced by actual variables from the set $a0, a1, \ldots, a9$ in order to create a training example, so there are 90 different variable combinations possible for the template above. This is necessary in order to teach the network to generalize well. Not only are different variables used in the placeholders when creating problems from a template, but the input sentences in the "statement_list" value will be presented in random order, with possible repetitions and with each statement being presented one or more times.

- The resulting input "X" for a problem instances will be a one-hot encoded set of statements based on the statement list as described above. It will include the statements from the statement list presented in random order with repetitions and end with the "question".
- The first output "Y1" will be a version of the "X" in which any each statement is presented only once and in the order in which it appeared in "X". Hence, "Y1" is the knowledge state, and we are teaching the network to output its knowledge state in "Y1" by repeating sentences in the order they were learned. This constitutes a knowledge memory.
- The second output "Y2" is a one-hot encoded version of the "answer" to the "question".

Since we can generate multiple problem instances for each template, each template also contains a "max_instances" member as well, which tells the system the maximum number of randomly generated problems to create for the given template. Different placeholders (i.e. "pa1" and "pa2") also correspond to distinct variables but that the same variable is always used for a given placeholder during the instantiation process.

A full list of the training templates is shown the Github repository [12].

5 Key Results

The network was found to yield no errors on the validation data set, consisting of 5000 sample problems, after being trained through over 1700 epochs of training data. It was subsequently also tested as in examples 1 and 2 in Sections 3.1 and 3.2 as well.

6 Some Limitations

It must be noted that in the present implementation, an Agent is a Python object, and the knowledge base is still maintain as a Python list of sentences that is 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.

At present, agents are not trained to handle more complex reasoning chains such as a, $a \rightarrow b$, $b \rightarrow c$, and then conclude c is true. This would imply there is still an incompleteness in agent knowledge in the sense that a more complete agent would have the power to first conclude the truth of b and combine that knowledge via modus ponens with $b \rightarrow c$ to conclude c. Also, the agents are not trained to solve general propositional reasoning problems and can only handle

the limited task scope described above. This is considered a direction for future work.

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.

8 Appendix

Source code is available at https://github.com/CoderRyan800/alpha.git and can be installed as follows.

- Create a Python 3.6 or Python 3.7 virtual environment.
- Activate the new virtual environment.
- "git clone https://github.com/CoderRyan800/alpha.git"
- "cd alpha"
- "pip install ."
- "cd neural"

The main program is nn_entity_v1p̀y. On line 544 of the code you can set the flag new_network_flag to False if you wish to use the pre-built neural network located in the data subdirectory. You can set it to True if you want to train the neural network yourself, but unless you have a GPU it may take many days to train.

The Github repository was called "alpha" because the author regards this as alphaware. The author does not claim suitability for any purpose and only provides the software "as is".

IMPORTANT: If you do not choose to train a new network you have to download a pre-existing data set. You can go to:

`http://alpha-demo.s3-website-us-east-1.amazonaws.com/`

This is needed not only for the data set but also to obtain the neural network trained after over 1700 iterations if you wish to avoid retraining. Put these downloaded files into the "data" subdirectory in the alpha repository on your local system prior to trying to run with the flag on line 544 set to False.

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] Ryan Mukai. Github repository alpha by coderryan800. https://github.com/CoderRyan800/alpha.git, NOTE = "ryan20083437@gmail.com".

[13] 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.