Cognitive Module Networks for Grounded Reasoning

Alexey Potapov[1,2], Anatoly Belikov[1], Vitaly Bogdanov[1], Alexander Scherbatiy[1]

[1]SingularityNET Foundation, The Netherlands
[2]ITMO University, St. Petersburg, Russia
{alexey, vitally, abelikov, alexander.scherbatiy}@singularitynet.io

Abstract. The necessity for neural-symbolic integration becomes evident as more complex problems like visual question answering are beginning to be addressed, which go beyond such limited-domain tasks as classification. Many existing state-of-the-art models are designed for a particular task or even benchmark, while general-purpose approaches are rarely applied to a wide variety of tasks demonstrating high performance. We propose a hybrid neural-symbolic framework, which tightly integrates the knowledge representation and symbolic reasoning mechanisms of the OpenCog cognitive architecture and one of the contemporary deep learning libraries, PyTorch, and show how to implement some existing particular models in our general framework.

Keywords: grounded reasoning, cognitive architectures, neural module networks, visual question answering

1 Introduction

Most contemporary cognitive architectures (CAs) are considered as hybrid [1]. However, it is difficult to find an architecture that tightly integrates a powerful symbolic reasoning with modern deep neural networks (DNNs). At the same time, such neural-symbolic integration of learning and reasoning constitutes a separate important field of research[1]. Unfortunately, there are just a few attempts to create a general framework, within which neural-symbolic models for solving different tasks can be developed. Moreover, conceptually sound approaches usually don't rely on the contemporary frameworks and practical models of deep learning and efficient engines of symbolic reasoning, but implement a particular type of models with specific inference procedures, for which mapping between neural networks and logical expressions is established (e.g. [2]).

Some general-purpose neural-symbolic frameworks, which combine contemporary DNN and symbolic reasoning tools, do exist. DeepProbLog [3] is one such frameworks. Unfortunately, examples of its applications are mostly limited to such toy problems as recognizing a pair of MNIST digits conditioned on their known sum. Other works on a hybrid neural-symbolic approach based on deep probabilistic pro-

[1] http://www.neural-symbolic.org/

gramming (e.g. [4]) also don't show how state-of-the-art models for various benchmarks can be created within them. At the same time, one can encounter a variety of modern individual solutions to specific problems based on ad hoc hybrid models, which are quite efficient, but narrowly applicable (e.g. the Transparency by Design, TbD, model [5]).

One of the prominent examples of this situation can be found in the field of visual reasoning, in particular, Visual Question Answering (VQA) that requires explicit reasoning capabilities. In particular, VQA implies variable binding, handling which is considered as a classical problem for connectionist models [6]. Although contemporary attention models incorporated into DNNs (in particular, in VQA [7]) partially address this problem, but without compositionality featured by symbolic approach.

On the one hand, DNNs achieve state-of-the-art results on some VQA datasets containing real-world images, and the use of contemporary DNN models and frameworks in visual processing seems essential. However, it is convincingly argued [8] that pure neural models tend to learn statistical biases in datasets (in particular, strong language priors, e.g. [9]) and to map inputs to outputs directly instead of explicitly modeling the underlying reasoning processes that results in a considerable decrease of performance on specially designed datasets (such as CLEVR [10] or GQA [11]). On the other hand, application of pure symbolic reasoning systems, which supposes that the input images are preliminarily processed by a vision subsystem and converted into symbolic form, is not robust and has low performance.

Apparently, hybrid solutions are desirable in order to account for all aspects of VQA. However, state-of-the-art VQA models frequently use narrow imperative program executors instead of general declarative reasoning systems (see, e.g. [5][8]).

In this work, we propose a framework of hybridization of the integrative cognitive architecture OpenCog with symbolic inference engine operating on declarative knowledge bases with modern deep learning libraries supporting gradient descent optimization of differentiable functions over real-valued (subsymbolic) parameters.

Attempts to bridge the symbolic/subsymbolic gap via such hybridization of symbolic reasoning and deep neural networks in OpenCog has been done before [12][13]. However, they were aimed at specific DNN architectures (a version of DeSTIN system and a hierarchical attractor neural network), and didn't support end-to-end training of the DNN model as a component of a pipeline that includes symbolic reasoning.

The proposed framework enables integration of OpenCog with arbitrary DNN models providing means to backpropagate errors from conclusions to DNNs through symbolic inference trees. This allowed us not only to reproduce the example used to illustrate DeepProbLog [3], but also to re-implement the TbD model [5] with the use of the general symbolic reasoning engine operating over declarative knowledge instead of imperative program executor specifically design for CLEVR VQA dataset.

2 Motivation: Grounded Reasoning

As it is shown in [14], the OpenCog's language Atomese suits well to express queries about image content, for example, in the task of semantic image retrieval. These queries are executed by OpenCog's reasoning subsystems such as the Unified Rule Engine (URE), in particular, with the Probabilistic Logic Networks (PLN) rule set, and the Pattern Matcher over the labels assigned by DNNs to the detected objects.

For example, the following query in Atomese can retrieve a video frame that contains a bounding box recognized as a helicopter (and easily can be extended to more complex queries):

```
BindLink
  VariableList
    VariableNode “$Frame”
    VariableNode “$BB”
  AndLink
    InheritanceLink
      VariableNode “$Frame”
      ConceptNode “Frame”
    InheritanceLink
      VariableNode “$BB”
      ConceptNode “Helicopter”
    MemberLink
      VariableNode “$BB”
      VariableNode “$Frame”
  ListLink
    VariableNode “$Frame”
    VariableNode “$BB”
```

Here, `BindLink` specifies the rule with three parts: a variable declaration, a pattern to be found in Atomspace, a graph to be formed for each matched subgraph (for different variable groundings). `InheritanceLink` is used to indicate that some bounding box (which is distinguished by its name, e.g. `ConceptNode “BB-03-11”`) is recognized as an object of some specific class, and `MemberLink` is used to indicate that the bounding box belongs to a certain frame. In order to successfully retrieve information, OpenCog will just need inheritance and member links for frames and bounding boxes (each of which is represented as an atom, e.g. `ConceptNode`) to be stored in Atomspace.

However, the simplest way to perform visual reasoning, which consists in preliminary processing images with DNNs and inserting the descriptions of the images into Atomspace with consequent pure symbolic reasoning, is far from enough even in the case of image retrieval. One may want to find images with either a happy child or a jumping boy, which can be the same. This means that assigning one label per object or bounding box in image is not enough.

The problem is even more obvious if we consider the task of VQA, in which more complex questions are frequent, e.g. “are the people looking in the same direction?” or “are the chairs similar?” Apparently, to answer these questions, one should not simply reason over symbolic labels, but should go down to the level of image features that implies a deeper neural-symbolic integration. Although complete disentanglement of reasoning from vision and language understanding can work for such datasets as CLEVR [15], we consider such disentanglement not as an achievement, but as oversimplification, which is not scalable to real-world reasoning.

Thus, what we want to make our system to reason about is not mere symbols, but symbols with their groundings (e.g. grounded predicates), which are calculated by demand in the course of reasoning.

Let us assume for example that we have a VQA system, which detects a number of bounding boxes (BBs) in the image and describes them with some high-level features (that is quite typical for models developed for some benchmarks [7]). A naïve neural-symbolic system will apply a multinomial classifier to these features to produce most probable labels for bounding boxes (maybe a few such classifiers to recognize objects and their attributes). Each output neuron of such classifier can be considered as a grounded predicate corresponding to a certain concept (e.g. “boy”, “happy”, etc.).

Instead of precomputing truth values of all these predicates, the system can compute only those predicates, which are necessary. For example, the question “Is the boy happy?” requires to check predicates “boy” and “happy”, while the question “What color is the car?” can use a symbolic knowledge base to select predicates corresponding to concepts inherited from the concept “color”.

Of course, this requires using a one-class classifier for each concept instead of a multinomial classifier, which can only calculate truth values of all predicates simultaneously. However, this is not really a drawback, because no two concepts are precisely mutually exclusive. A boy is also a child (and interestingly, we can frequently recognize children without recognizing them as boys or girls, so it is more likely that we use different grounded predicates to recognize classes and subclasses instead of recognizing subclasses only and inferring classes symbolically). Even more, an object can be simultaneously black and white, and even a boat can have a shape of a banana.

One can argue that all these predicates can still be pre-calculated before reasoning without too large overhead. However, it is not really the case, when we are talking about relations between objects in images, especially those, which require descending on the level of image features or even pixels.

Moreover, the reasoning system can influence the sequence of operations performed by the vision system or influence the output of different levels of the vision system by imposing priors dependent on the current state of the cognitive system (e.g. in neural-symbolic generative models). For example, in the TbD model, a sequence of applications of DNN modules is constructed in a symbolic (although not declarative) way (see fig. 1).

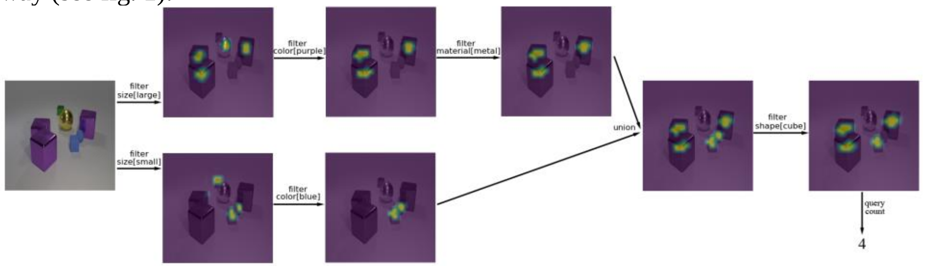

Fig. 1. Module network answering, “How many blocks are tiny blue objects or big purple metallic objects”

Here, grounded predicates or functions are applied to the whole image instead of bounding boxes, and they produce attention maps and features that are fed to the next DNN modules, although one can imagine that modules are selected by the reasoning system dynamically depending on the already obtained results and background knowledge.

Here, we aim not at discussing, what models better fit to visual (or, more generally, grounded) reasoning, but at designing a framework, which allows combining dynamically arbitrary DNN models with knowledge-based symbolic reasoning. Since one of the main motivations for this is to replace an ad hoc hand-coded imperative "reasoner" (program executor) in Neural Module Networks with an entire cognitive architecture, we call this approach Cognitive Module Networks.

3 Cognitive Module Networks

The best way to achieve the stated above goal would be to keep the possibility of using the DNN modules in the existing Neural Module Networks while replacing only hard-coded program executors with a general reasoning engine. Thus, what we need from OpenCog is to chain forward applications of DNN modules in a similar way as the program executors do. Technically important issue is the necessity to construct this chain of applications as an uninterrupted computation graph that supports error backpropagation by the corresponding DNN library.

On the side of OpenCog, such application can be carried out by executing `GroundedSchemaNodes` and `GroundedPredicateNodes` (which differ in that the former returns Atoms while the latter returns `TruthValues`). Some restrictions of the existing API for `GroundedSchemaNode` were to be overcome to achieve the necessary functionality. In particular, execution of methods of dynamically created objects rather than static objects, and passing tensors (data structures specific to a certain DNN library) between calls to `GroundedSchemaNodes` without conversion are desirable. Different solutions to these problems are possible. However, we will not go into technical detail here and focus more on a conceptual level.

Consider the following code in Atomese that corresponds to the question "Is the zebra fat?" (or more precisely, "Is there a fat zebra in the image?").

```
SatisfactionLink
  VariableNode "$X"
  AndLink
    InheritanceLink
      VariableNode "$X"
      ConceptNode "BoundingBox"
    EvaluationLink
      GroundedPredicateNode "py:runNN"
      ListLink(VariableNode("$X"), ConceptNode("zebra"))
    EvaluationLink
      GroundedPredicateNode "py:runNN"
      ListLink(VariableNode("$X"), ConceptNode("fat"))
```

Its execution by Pattern Matcher will cause the enumeration of all `ConceptNodes` that inherit from “BoundingBox” and pass them to the wrapper function `runNN`, which will take visual features for the given bounding box (e.g. attached as `Values` to `ConceptNodes`) and pass them to the DNN that corresponds to the provided class to be recognized (e.g. “zebra” or “fat”). Depending on implementation, it can be one DNN that accepts word embeddings as input, or there can be many small classifiers over high-level visual features (e.g. taken from ResNet or such) – each classifier for each concept. Then, `runNN` should convert the DNN output to OpenCog’s `Truth-Value`, over which `AndLink` acts. Thus, all bounding boxes will be retrieved that classified simultaneously as “zebra” and “fat”.

This simple code already does a sort of variable grounding for neural networks and use of declarative knowledge, which neural networks lack otherwise. However, this solution didn’t allow for training DNNs based on conclusions made by the reasoner, and it has an ad hoc interface to run particular networks.

In the companion paper [16], we describe how differentiable rules for URE can be constructed that enables both learning tensor truth values and learning formulas for rules themselves by gradient descent. In this paper, we extend this approach by using predicates and schemas grounded in DNN models. More precisely, we focus more on a DNN-centered framework, which can be adopted by the deep learning community. Current implementation supports PyTorch backend, although Tensorflow and other backends can be added in the future.

As described in [16], if formulas attached to URE rules are implemented as operations on PyTorch tensors, application of a sequence of formulas corresponding to the chain of reasoning steps found by URE will yield a PyTorch computation graph, over which errors from final conclusions to PyTorch variables can be backpropagated. For example, PLN rule set for URE can help us to infer the truth value of the conclusion

```
EvaluationLink
  PredicateNode “green”
  ConceptNode “apple-001”
```

using modus ponens from the truth values of premises:

```
ImplicationLink
  PredicateNode “apple”
  PredicateNode “green”
EvaluationLink
  PredicateNode “apple”
  ConceptNode “apple-001”
```

With the use of tensor truth values and PyTorch implementation of the formula for modus ponens, the error can be propagated from the truth value of the conclusion to the truth values of the premises.

If we replace `PredicateNode` in the above example with `GroundedPredicat-eNode`, which can in particular execute a DNN that outputs the probability that some object in an image can be recognized as an apple, then the PyTorch computation graph will include this DNN as a subgraph, and error will be propagated through the truth value (probability) produced by it to its weights. Instead of just adjusting truth values, we will train neural networks to output such values that lead to correct conclusions inferred by the reasoning system. Since the OpenCog reasoning subsystems perform the process of rewriting subgraphs of a (hyper)graph composed of Atoms, they can compose and execute an arbitrary graph (architecture) of neural modules.

In order to make this possible, DNN modules should be attached to atoms, to which variables in queries can be bound. CogNets library (its experimental implementation can be found here [2]) provides class `CogModule` that inherits from `torch.nn.Module`. On the one hand, `CogModule` objects can behave as ordinary `torch.nn.Module` objects implying that if we take some module network and change the inheritance of its modules to `CogModule`, it will continue working correctly. On the other hand, each `CogModule` object also attaches itself (through Values) to the specified Atom in Atomspace. Execution of neural modules attached to Atoms is done through a special `GroundedSchemaNode` that extracts `CogModule` objects from Atoms and passes arguments to them.

The basic application of CogNets will be to just inherit all modules in the TbD model from `CogModule`. Then, we will be able to use OpenCog to execute queries represented in the form of `BindLinks`. One question, which we don't consider in detail here, is how to obtain such queries from question in natural language. OpenCog contains natural language processing components, in particular RelEx, that can be used to parse questions and then convert them to Atomese queries. However, these components have some limitations. Another possibility is to reuse the pre-trained LSTM-based program generator from [8] (used in the TbD model also), which produces programs from questions, which they can be easily translated into Atomese. This approach works well for CLEVR, although cannot be applied to the COCO VQA benchmark in contrast to RelEx.

"Reasoning" in the TbD model is performed by executing an imperative program, composed of a sequence of applications of DNN modules. For example, the question "What color is the cylinder?" will be transformed to the consequent application of `filter_shape[cylinder]` module and `query_color` module. The `query_color` module will take as input the image features masked by the attention map produced by `filter_shape[cylinder]` module and outputs new feature map, which is then passed to the final classifier. The multinomial classifier will calculate the probabilities of all answers.

Thus, if we directly apply the TbD model just replacing its program executor with OpenCog, we will gain not too much, because these Atomese queries will be nested applications of `GroundedSchemaNodes`. Although these `GroundedSchemaNodes` will be presented in Atomspace knowledge base, OpenCog's reasoning capabilities will not be involved.

However, we can explicitly introduce the query variable $X, replace the query module with the corresponding filter module `filter_color[$X]`, and ask the reasoning engine to find such value of $X which will produce a non-empty final attention map. Therefore, the question "What color is the cylinder?" can be represented declaratively in Atomese:

```
AndLink
```

[2] https://github.com/singnet/semantic-vision/tree/master/experiments/opencog/cog_module

```
  InheritanceLink
    VariableNode “$X”
    ConceptNode “color”
  EvaluationLink
    GroundedPredicateNode “py:filter”
    ConceptNode “cylinder”
  EvaluationLink
    GroundedPredicateNode “py:filter”
    VariableNode “$X”
```

if we use a specially coded static Python function filter, which executes a corresponding DNN module depending on the name of the given argument. Here, we will need to add to Atomspace such facts as

```
InheritanceLink(ConceptNode(“red”),ConceptNode(“color”))
```

Pattern Matcher will be able to enumerate different colors and call different `filter_color` modules, for which PLN will infer the truth value of the given `AndLink`.

CogNets provide a general wrapper function `CogModule.callMethod` to extract Python objects attached to Atoms and call their method with automatic unwrapping of their arguments from Atoms and wrapping their results back into Atoms in such a way that, in particular, forward methods of `torch.nn.Module` objects can be used as is. Thus, the above code with the use of CogNets will look like

```
AndLink
  InheritanceLink
    VariableNode “$X”
    ConceptNode “color”
  EvaluationLink
    GroundedPredicateNode “py:CogModule.callMethod”
    ListLink
      ConceptNode “cylinder”
      ConceptNode “call_forward_tv”
      ConceptNode “image”
  EvaluationLink
    GroundedPredicateNode “py:CogModule.callMethod”
    ListLink
      VariableNode “$X”
      ConceptNode “call_forward_tv”
      ConceptNode “image”
```

`CogModule.callMethod` will extract the Python object (which will be a DNN module as `CogModule` object here) attached to `ConceptNode “cylinder”` and execute its `call_forward_tv` method (which is the method of `CogModule` inherited from `torch.nn.Module`), which will extract Python object (PyTorch tensor here) from `ConceptNode “image”` and execute forward method of the DNN module reducing its output to the tensor truth value. CogNets library provides a syntactic sugar in Python to form the necessary Atomese expressions concisely.

One can see that this allows for assembling modules in neural module networks using symbolic knowledge and reasoning over (probabilistic) logic expressions with variable grounding. This opens the path to real visual reasoning. For example, Atomspace can contain the fact that left(X,Y) :– right(Y,X). Applying this fact during the chain of reasoning performed by URE will result in transforming the module network and using `relation[right]` module instead of `relation[left]` module of TbD. In particular, given the question "To the left of what object is the green pyramid?" humans will most likely find the green pyramid first and then look to the right of it. Direct conversion of the question into the imperative program cannot represent such visual reasoning, while it can appear naturally within our approach.

In contrast to the TbD model with its hard-coded program executor, our approach can naturally be applied not to CLEVR, but also to COCO VQA and even to the example given in the paper on ProbLog [3], namely to recognize digits on a pair of MNIST digits conditioned on their sum. The premise for the query (requiring the sum to be 7) with two `VariableNodes` of `NumberNode` type will look like

```
AndLink
  EqualLink
    PlusLink
      VariableNode("$X")
      VariableNode("$Y")
    NumberNode "7"
  EvaluationLink
    GroundedPredicateNode "py:CogModule.callMethod"
    ListLink
      VariableNode "$X"
      ConceptNode "call_forward_tv"
      ConceptNode "image1"
  EvaluationLink
    GroundedPredicateNode "py:CogModule.callMethod"
    ListLink
      VariableNode "$Y"
      ConceptNode "call_forward_tv"
      ConceptNode "image2"
```

4 Conclusion

We have considered an approach to neural-symbolic integration, within which knowledge-based reasoning is carried out over symbols grounded in perception through deep neural networks, that, in particular, allows the symbolic reasoner to interoperate with execution of neural modules and to assemble a neural module network on fly depending on the current input and background knowledge.

We have implemented a framework, which embodies this approach with the use of the contemporary cognitive architecture and deep learning library, namely OpenCog and PyTorch. This implementation enables such integration of OpenCog with arbitrary DNN models that allows for error backpropagation from conclusions to DNNs through symbolic inference trees.

On example of neural-symbolic models widely used for the CLEVR benchmark, we have shown how a domain-specific program executor, which assembles neural module networks using given linear sequences of imperative commands, can be replaced with a general-purpose reasoning engine operating over a declarative knowledge base that can equally be used to reproduce models implemented within other frameworks, in particular, ProbLog.

References

1. Duch, W., Oentaryo, R.J., Pasquier, M.: Cognitive Architectures: Where Do We Go from Here. Frontiers in Artificial Intelligence and Applications (Proc. 1st AGI Conference), vol. 171, pp. 122–136 (2008)
2. Besold, T.R., et al: Neural-Symbolic Learning and Reasoning: A Survey and Interpretation. arXiv preprint, arXiv: 1711.03902 (2017)
3. Manhaeve, R.M., et al.: DeepProbLog: Neural Probabilistic Logic Programming. arXiv preprint, arXiv: 1805.10872 (2018)
4. Overlan, M.C., Jacobs, R.A., Piantadosi, S.T.: Learning abstract visual concepts via probabilistic program induction in a Language of Thought. Cognition, vol. 168, pp. 320-334 (2017)
5. Mascharka, D., Tran, Ph., Soklaski, R., Majumdar, A.: Transparency by Design: Closing the Gap Between Performance and Interpretability in Visual Reasoning. arXiv preprint, arXiv: 1803.05268 (2018)
6. Fodor, J. A., Pylyshyn, Z. W.: Connectionism and cognitive architecture: A critical analysis. Cognition, 28(1–2), 3–71 (1988)
7. Singh, J., Ying, V., Nutkiewicz, A.: Attention on Attention: Architectures for Visual Question Answering (VQA). arXiv preprint, arXiv: 1803.07724 (2018)
8. Johnson, J.: Inferring and Executing Programs for Visual Reasoning. arXiv preprint, arXiv: 1705.03633 (2017)
9. Agrawal, A., et al.: Don’t just assume; look and answer: Overcoming priors for visual question answering. Proc. IEEE Conf. on Computer Vision and Pattern Recognition, pp. 4971–4980 (2018)
10. Johnson, J., et al.: CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. arXiv preprint, arXiv: 1612.06890 (2016)
11. Hudson, D.A., Manning, Ch.D.: GQA: A New Dataset for Real-World Visual Reasoning and Compositional Question Answering. arXiv preprint, arXiv: 1902.09506 (2019)
12. Goertzel, B.: Perception Processing for General Intelligence: Bridging the Symbolic/ Subsymbolic Gap. AGI’12, Springer: Lecture Notes in Computer Science, vol. 7716, pp. 79-88 (2012)
13. Goertzel, B.: OpenCog NS: A Deeply-Interactive Hybrid Neural-Symbolic Cognitive Architecture Designed for Global/Local Memory Synergy. Proc. AAAI Fall Symposium Series FS-09-01, pp. 63-68 (2009)
14. Potapov, A.S., et al.: Semantic image retrieval by uniting deep neural networks and cognitive architectures. AGI’18, Springer: Lecture Notes in Computer Science, vol. 10999, pp. 196-206 (2018)
15. Yi, K., at al.: Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding. arXiv preprint, arXiv: 1810.02338 (2018)
16. Differentiable Probabilistic Logic Networks. To be published