Learning to Reason

PhD Thesis Proposal

Wojciech Zaremba WOJ.ZAREMBA@GMAIL.COM

Advisors:

Rob Fergus FERGUS@CS.NYU.EDU

Yann LeCun YANN@CS.NYU.EDU

New York University

Abstract

Neural networks have proven to be very powerful models for object recognition (Krizhevsky et al., 2012), natural language processing (Mikolov, 2012), speech recognition (Graves et al., 2013), and other tasks (Sutskever et al., 2014). However, there is still a huge gap between them and a truly intelligent system. I identify several qualities that an intelligent systems should possess, namely: (i) reasoning ability, (ii) the capability to integrate with external interfaces and (iii) small sample complexity. My thesis will focus on tackling these problems.

1. Introduction

It is clear that by improving the performance of current statistical learning systems, we will not be able to make them intelligent, even if they achieve 0% test error. This proposal explores what skills are necessary for statistical learning system to become "intelligent" and outlines attempts to address these skills.

While many qualities make up true intelligence, we focus on three skills that are notably lacking from current approaches: (i) the ability to reason, (ii) the capability to integrate with external interfaces and (iii) a small sample complexity. The goal is to address all of these problems within a single unified framework. This will be approaches to a wide range of tasks in different domains, to empirically verify its robustness.

For some of these tasks, exploratory work has been performed which gives good intuitions for the proposed work.

2. Reasoning abilities

Reasoning is the process of forming conclusions, judgements, or inferences from facts or premises. An artificial system that can reason should be able to understand high level concepts like:

- Scope
- Conditioning
- Nesting
- Repetitions
- Memorization

While each of this skills could be solved in particular domain, the resulting system would be of little use and would require strong assumptions about possible scenarios. Moreover, an intelligent reasoning system cannot be based only on predefined rules. Instead, it should be based on pattern matching, and the application of learned heuristic algorithms.

There are many domains where we can test the reasoning ability of our system to see if it is able to learn postulated concepts. These include (a) computer programs and (b) the derivation of mathematical theorems. Both of these are rich in scoping, branching, nesting, pattern repetition, and so on. Drawing high-level conclusions in such domains requires sophisticated reasoning. Eventually, the framework should be able to handle many different domains simultaneously.

2.1. Reasoning in computer programs

The first reasoning task that I intend to tackle is to train statistical models the meaning of computer programs. Consider one possible instantiation of this, where the task is to take a program as an input (e.g. character-by-character),

Input:
```
f=(8794 if 8887<9713 else (3*8334))
print((f+574))
```
Target: 9368.
Model prediction: 9368.

Figure 1. An exemplary program that we take as an input. The task is to predict the evaluation of the program. We have achieved a high level of performance on this task (Zaremba & Sutskever, 2014).

and predict the program output. This requires the statistical model to understand every single operand of a program. For instance, in the case of addition it involves bit shifts, and memorization of operations on digits. Moreover, programs can contain variable assignment, if-statements, and so on. We were able partially to address this problem (Zaremba & Sutskever, 2014). Figure 1 shows an exemplary program and its output, along with a prediction of the output obtained with recurrent neural network.

Although current results are promising, they are very limited. We are able to deal with programs that can be evaluated by reading them once from left-to-right, but generic programs are far more complex. Our reasoning system has to be able to evaluate for arbitrarily long time, if the task requires it. Moreover, it shouldn't be limited by a fixed memory size. Instead, memory should be available as an interface (see Section 3).

2.2. Reasoning in mathematics

It is known that theorem proving is an intractable task in computational sense. However, humans are able to prove theorems. They do this by employing prior knowledge, acquired from experience of solving other, related problems, and fitting known mathematical "tricks" to the new problem.

More formally, we can think about proofs as an application with multiple axioms that starts with a hypothesis that we want to prove. Thus mathematical skills can be perceived as learning prior over trees of axioms. This prior "suggests" which sequence of axioms is more likely to lead to a proof of the theorem.

My interest lies in training statistical machine learning systems to learn such priors. We have started with the very constrained mathematical domain of polynomials and demonstrate that here we can learn a prior over mathematical identities. Our recent work (Zaremba et al., 2014) concerns identities over polynomials over matrices. Figure 2 shows example of a simple identity over matrix polynomials. It states that, $\sum_{i,j}\sum_k A_{i,k}B_{k,j} = \sum_j\sum_k(\sum_i A_{i,k})B_{k,j})$. This an example of toy identity. Our system is able to derive much more sophisticated identities.

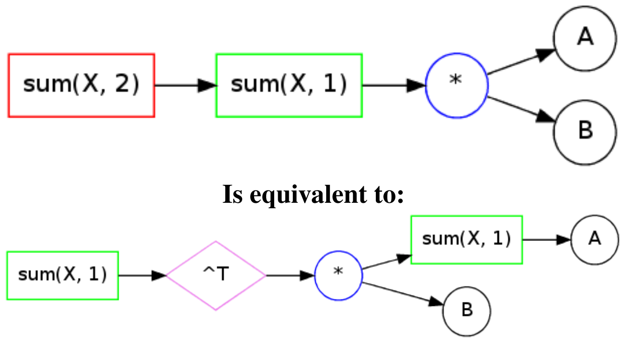

Figure 2. An example of two equivalent mathematical expressions. It states that $\sum_{i,j}\sum_k A_{i,k}B_{k,j} = \sum_j\sum_k(\sum_i A_{i,k})B_{k,j})$. (Zaremba et al., 2014) contains more sophisticated examples.

3. Interface learning

Neural networks are very powerful statistical models, however some concepts might be very difficult for them to express. For instance, nesting seems to be a very common operation, that allows one to deal with (1) object hierarchies, (2) natural language parsing, and (3) compositionality. The ability to nest concepts can be provided by adding a stack as an external interface to the neural network, with the neural network learning how to use such an interface.

More broadly, external interfaces could enhance neural networks capabilities, and also provide access to external resources, and datasets. Potentially useful interfaces include:

- stack
- FIFO
- an external memory (Weston et al., 2014; Graves et al., 2014)
- search engine
- database
- execution environment like python interpreter

The state space of many interfaces is massive (e.g. all possible search queries). Moreover, external interfaces are non-differentiable, These obstacles could potentially be addressed by learning a differentiable model that describes them (e.g. a neural network). This network would simulate the external interface for purpose of being differentiable.

4. Small sample complexity

Current deep learning systems suffer from a large sample complexity. This hinders the potential use of system for online learning (e.g. robots). It is expected that during the initial phase of learning any system without prior knowledge would need to consume a large number of samples. However, we hope that as training progresses, sample complexity should drop. But this is not observed in a current systems. I propose several approaches that allow sample complexity to be decreased. The meta-optimizer (see Subsection 4.4) is the desired solution, but I also propose some intermediate solutions that can also decrease sample complexity.

4.1. Automatic derivation of better models

Neural networks are designed in an ad-hoc fashion, using a researchers intuition about what architectures will result in a high prediction performance. One way of addressing this issue would be to fully understand why, and how the models work. However, mathematical analyses give little insight and it is unclear if it is even possible. I would like to automatically explore a large space of models, and evaluate them on various tasks. This involves trying different connectivity patterns, activation functions, and hyperparameter settings. To search over this discrete space, I intend to use genetic programming.

4.2. Augmentation marginalization

For simplicity, we focus on computer vision tasks.

We often train neural networks for multiple epochs, while the same samples are presented multiple times. The best performance is achieved when samples are slightly modified over the course of training. In case of computer vision, it is done by jittering images, rotating them, and so on. This is known as data-augmentation. Theoretically, a full image contains information about all updates obtained with all possible augmented samples. More formally, let x be an image, y its label and $x_1, x_2 \ldots x_k$, are its augmented copies. L is a loss function, and θ_i are parameters at step i. I would like to predict for a given x, the sum of derivatives for all data augmentations:

$$f(x, \theta_1) = \sum_{i=1}^{k} \partial_\theta L(x_k, y, \theta_i) \tag{1}$$

Instead of following gradient of $\partial_\theta L(x, y, \theta_1)$, we could follow $f(x, \theta_1)$. This could improve learning speed, as a single sample would provide information about all its augmented copies.

The function f might have its own parameters, and I would like to model it as a neural network which would be trained in a supervised fashion.

4.3. One-shot learning objective

Human are able to recognize a new object by seeing them just once. Contemporary deep learning systems are far from being able to do this, requiring hundreds or even thousands of examples. We propose a new formulation of neural network training that focus on training for the one-shot learning objective.

The main idea is to train one neural network to predict parameters of another neural network. More formally, let $\phi(x, W_{in}, \theta) = (y, W_y)$ be a neural network parametrized by θ. Weights $W_{in} \in \mathbb{R}^{f \times l}$ are a top layer weights, where f is a top layer feature size, and l is a number of labels. $W_y \in \mathbb{R}^f$ is a single weight vector used to predict class y.

For a given new class x', y', we compute $W_{y'} = \phi(x', None, \theta)$ to produce weights $W_{y'}$. We use this weights to classify other objects belonging to the class y'. This idea is currently being explored with my collaborators Elias Bingham and Brenden Lake.

4.4. Meta-optimizer

I would like to build a meta-optimizer that would be used to train a neural network. Such an optimizer would take as input the gradients of a neural network, and would decide on the next update step. The optimizer itself could be parameterized with another neural network, which would allow it to simulate any first order, gradient-based, learning algorithm like SGD, momentum, LBFGs etc. This implies that such a meta-optimizer should subsume all first order, gradient-based optimization techniques. This work is under progress, supervised by my collaborator Rahul Krishnan and Anna Choromanska.

5. Discussion

Tackling the aforementioned problems would take us much closer to the real intelligent systems, and defines three core pillars of Artificial Intelligence. However, there are many other problems which need to be solved and integrated to achieve a fully intelligent system, e.g. navigation, learning by imitation, cooperation, and many others.

References

Graves, Alex, Mohamed, Abdel-rahman, and Hinton, Geoffrey. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 6645–6649. IEEE, 2013.

Graves, Alex, Wayne, Greg, and Danihelka, Ivo. Neural turing machines, 2014.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Mikolov, Tomáš. *Statistical language models based on neural networks*. PhD thesis, Ph. D. thesis, Brno University of Technology, 2012.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.

Weston, Jason, Chopra, Sumit, and Bordes, Antoine. Memory networks, 2014.

Zaremba, Wojciech and Sutskever, Ilya. Learning to execute, 2014.

Zaremba, Wojciech, Kurach, Karol, and Fergus, Rob. Learning to discover efficient mathematical identities. *arXiv preprint arXiv:1406.1584*, 2014.