Instill Symbolic Knowledge into Connectionist Models

Jianyu Zhan
nasa4836@gmail.com

Abstract

Artificial neural networks — *a.k.a.* the *connectionist systems* — exhibit powerful expressiveness, desirable efficiency and satisfactory robustness, it is thus believed that it should be combined with symbolic systems — another paradigm for pursuing human-level intelligence — to construct a more intelligent agent, which is capable of learning and reasoning. In this article, we discuss the merits and limitations of both paradigms, and discuss the challenges confronted while integrating them and the approaches to tackle them. The contribution of this article is to propose a new approach to address one of the key challenges: instilling symbolic knowledge into connectionist models. The author believes this would be a promising direction, because it is inspired by the transitive isomorphism among *formal logic*, *programming* and *neural networks*, and has the natural duality in it, which would facilitate the reverse operation: distilling learned knowledge from the connectionist models into symbolic ones, that is to acquire new propositions.

1 Introduction

The breakthrough of training of deep belief networks[16, 5] in 2006 marks the revival of neural networks, and the triumph of a convolutional network called AlexNet[21] at ILSVRC2012[1] Contest ignites the sheer enthusiasm of extensive research on deep neural networks architecture in academia and industry. The ensuing marvelous advance of neural networks in various areas ranging from speech recognition, human level machine translation, handwritten text generation, to image classification, image captioning, and even widely believed unbeatable game playing like go, marks the arrival of a new era for deep learning.

The overwhelming success of deep learning in such broad areas at a short amount of time period really piques people's interest of knowing the underlying magic and the power of deep neural networks. The usual informal narrative of neural networks depicts it as a repeated process of warping of data in geometric space such that it can learn the true representation. The famous *Univer-*

[1] ImageNet Large Scale Visual Recognition Challenge

sal Approximation Theorem[19, 11] formally depicts this physical analogy and guarantees that a two-layer perceptron with sufficiently many hidden units can approximate any *borel measurable function* to any desired degree of accuracy. But this perspective fails to capture the essence of deep architecture. Actually, kernel machines, such as Support Vector Machines(SVMs) are shallow architectures, and the similar *Universal Approximation Theorem* for SVM has also been derived[14], but "shallow nature of kernel machines leads to fundamentally inefficient representation"[6]. The real power of neural networks lies in its *deepness*. Some recent researches on this topic have revealed some theoretical insights on the expressiveness and efficiency of deep architectures[7, 22, 29]. We will elaborate on this in later section.

The exciting reality and promising theoretical result of deep neural networks symbolizes the the third-time Connectionism resurgence, the first-time being in 1950s[10] at the dawn of Artificial Intelligence, and the last time being in 1980s, with the discovery of a new efficient training method called *BackPropogation*[30]. Human-like intelligence is always our ultimate pursuit, and the ups and downs of the neural networks - the pivotal connectionist model represent the spiral advance of human's understanding of human intelligence. Now it seems a correct direction, with the incomparable feat ranging from the perceptual tasks(i.e., vision, audition), which are, by Hubert Dreyfus's argument[9], *unconscious instincts*, to cognitive tasks(e.g., machine translation). But we have a little unease in that some more intelligent tasks are still in doldrums. That is reasoning, which is "algebraically manipulating previously acquired knowledge in order to answer a new question"[8]. The adjective *algebraic* unveils its symbolic essence. Symbolism was the dominant paradigm of AI research from the mid-1950s to the late 1980s, and it was believed that human intelligence can be modeled on manipulation of symbols. The *Physical Symbol System Hypothesis*[25, 26] asserts that "A physical symbol system has the *necessary* and *sufficient* means for general intelligent action".

The *Physical Symbol System Hypothesis* seems quite a strong assertion. Given that currently subdued symbolic AI research compared to the versatile deep neural networks, a natural question pops up: Can we somehow combine the Connectionism with Symbolism to achieve strong AI? The answer seems promisingly yes! There was a long-existent AI community striving for *Neural-Symbolic Integration*[12], to promote the research in current AI weakness: learning and reasoning. But the viability doesn't warrant the necessity of doing so, the argument of favoritism of this method would be the goal of this article.

In this article, we will devote to the discussion of *Neural-Symbolic Integration* and try to shed light on a viable direction of research on the integration of symbolic knowledge into the neural networks to bring the ability of reasoning in the connectionist model, which lays the foundation for a possible and viable route to build a more intelligent agent. Section 2 is an elaborated discussion the merits of connectionist models and the limitations of pure symbolic systems. Section 3 discuss *Neural-Symbolic Integration*, and challenges of integrating them, and the existent approaches to tackle them. Section 4 proposes a new research approach that highlights a viable model for integration of symbolic knowledge into the

neural network model. Section 5 draws some conclusions.

2 Connectionist Models and Symbolic Systems

2.1 Merits of Connectionist Models

We will start with the discussion of merits of the connectionist models from the perspective of what properties of a general intelligent agent should be expected to have, and as we will see in later section, this sheds light on how we can do to instill the symbolic logic knowledge into neural networks. We discuss several merits of the neural networks.

2.1.1 Expressiveness

Though the exact biological model that human intelligence builds on has not been totally deciphered, it is no doubt that the biologically inspired neural networks does draw a profound analogy between its structure and human brain neurons inter-connections(Figure 1).

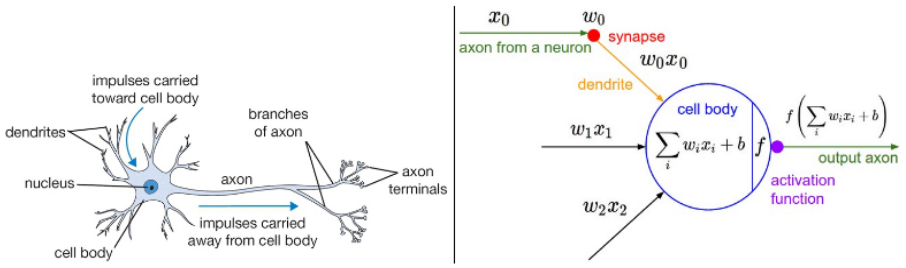

Figure 1: The analogy between biological neuron (left) and its mathematical model (right)[3]

More interestingly, a neuroscience paper[32] on Nature that presents an experiment, in which a ferret's auditory cortex was rewired to make retinal projections route into auditory pathway, and in the end the ferret induces visual orientation. This shows some dynamic rewire of neuron connections, which suggest the human brain is to some degree a *universal learning machine*. The *Universal Approximation Theorem* provides theoretical guarantee for neural networks' representation capacity. Sure we could not draw a direct parallel between these two concepts, but it at least provides a necessary condition for neural networks to act as a intelligent agent.

[3]This figure is from `https://cs231n.github.io/neural-networks-1/`.

2.1.2 Efficiency

For the efficiency metric, we take the amount of resources(neurons, parameters, etc) to measure neural networks' efficiency. Informally speaking, to learn the manifold, the higher degree of curvature of the manifold(more twist or turns), the more parameters(thus neurons) are needed to cover them, in turn more training examples are needed. More specifically, "they would require a large number of pieces to be well represented by a piecewise-linear approximation. Since the number of pieces can be made to grow exponentially with the number of input variables, this problem is directly connected with the well-known curse of dimensionality for classical algorithms"[6].

The method to overcome this problem, which actually makes deep neural networks stand out, is composition of many non-linearities. This is called the *combinatorial swindle*[22], that is, replacing exponentiation with multiplication. Take the example from [22]: if there are say $n = 106$ inputs taking $v = 256$ values each, this swindle cuts the number of parameters from v^n to $v \times n$ times some constant factor. This same paper provides the perspective from physics to elucidate the success of compositionality in neural networks: the hierarchical process. Indeed, this casual but powerful pattern exists everywhere in the physical world, for example, the object hierarchy: elementary particles forms atoms, which in turn form molecules, which in turn forms cells, etc. This hierarchical captures the very nature of the compositionality: the lower layers depicts the concrete and simple concepts, while as the layers stack up, more abstract and complex concepts emerge. For those curious, this paper[35] visually demonstrates how a convolutinoal neural network works and exhibits the idea we are talking about here.

This importance of compositionality is also captured and emphasized on by another paper[29] investigating the black-box of neural networks. This might provide another perspective of compositionality: recurrent neural networks(RNNs), which are loop-wired feed-forward neural networks, are practically universal Turing computers. And "all computable functions (by a Turing machine) are recursive, that is composed of a small set of primitive operations."[29].

2.1.3 Robustness

For robustness, the author would like to highlight two points: the *elaboration tolerance* and *graceful degradation*.

The concept of *elaboration tolerance* comes from John MaCathy, which is described as "A formalism is elaboration tolerant to the extent that it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances"[24]. Among other things, one of the properties of elaboration tolerance is generalization, which we would not elaborate here, but the author would like to point out that a plethora of research on *regularization*[36], which are addressing the generalization problem. Another hot topic that relates to elaboration tolerance is *Transfer Learning*[28], which is trying to more efficiently solve new tasks by leveraging experiences learned

from old tasks.

The concept of *graceful degradation* is defined as the tendency of neural networks to tolerate damage, be it biological or artificial. Human electroencephalogram(EEG) reveals that brain neural networks are simultaneously active, and parts of it damage would only vitiate the functionality that area it is responsible for. The same requirement is posed on the artificial neural network, and there is a powerful regularization method[34] by randomly masking out some non-output neuron and effectively create a bagging of several subnetworks during training, which precipitates "making predictions by averaging over multiple stochastic decisions implements a form of bagging with parameter sharing"[13]. This device provides really nice semantics of graceful degradation when deployed.

2.2 Symbolic Systems

2.2.1 Brief Introduction

Symbolic systems were proposed by Allen Newell and Herbert Simon in 1976[26]. In this paper, they claimed "Symbols lie at the root of intelligent action, which is, of course, the primary topic of artificial intelligence."To further facilitate later discussion, we briefly define a symbolic system.

Definition 1. A *symbolic system* consists of:

(a) **Symbols**: a set of entities, which are physical patterns.
(b) **Expressions**: composed of a number of instances (or tokens) of symbols related in some physical way.
(c) **Processes**: operations on expressions to produce other expressions: creation, modification, reproduction and destruction.

And it possess two properties[26]:

> **Designation**: An expression designates an object if, given the expression, the system can either affect the object itself or behave in ways dependent on the object.
>
> **Interpretation**: The system can interpret an expression if the expression designates a process and if, given the expression, the system can carry out the process.

Note these two properties "must also meet a number of additional requirements, of completeness and closure"[26].

Based on this definition, we claim that such a symbolic system solves problems by generating potential solutions with *heuristic search*, more specifically, by extracting useful information from a problem domain and using that information to guide their search.

2.2.2 The Limitation of Symbolic Systems

While exercising above symbolic systems as an intelligent agent, one would quickly confront two tricky and profound problems.

One is the combinatorial explosion problem, which is used to describe the rapid growth of the complexity of the search space of the problem domain. This problem is usually mitigated by pruning to eliminate some search directions totally or by carefully refined the heuristic search method.

The other one is a more profound problem, called *Symbol Grounding Problem*, first described by John Searle in his famous mental experiment *The Chinese Room Argument* in 1980[31], and latter specifically addressed by Stevan Harnad in [15]. The key idea behind this experiment is that by simply following the formal manipulation rule of the symbolic system, the one in the Chinese Room can answer the Chinese question raised by an outside observer without even knowing Chinese.That is, the "semantic interpretation of a formal symbol system is not intrinsic to the system, but just parasitic on the meanings in our heads"[15].

There are several grounding schemes proposed to address this problem, and they are beyond the scope of this article. Interested readers shall be pointed to Charles Sanders Peirce's and Stevan Harnad's works. As for our purpose of *Neural-Symbolic Integration*, the author would like to highlight Harnad's proposal that cognition should be modeled by *top-down*(symbolic) approach and meet *bottom-up* (sensory) approaches somewhere in between[15]. This looks like a hybrid *Neural-Symbolic Integration* scheme, and we would like to investigate the integrated scheme. But his idea did shed light on further research direction.

3 Neural-Symbolic Integration

Now we can turn to discuss the *Neural-Symbolic Integration*. The pursuit of human intelligence had been a driving force in establishing Artificial Intelligence(AI) as a discipline at the Dartmouth Conference in the summer of 1956. However, its progress was gradually subdued during the ensuing half century in comparison to what we call the Weak AI, as is witnessed especially by the current rejuvenation of Deep Learning. It turned out that the grand goal of AGI would be too hard to achieve than was expected. This is definitely a cross-discipline endeavor should it be realized. But a clear direction towards AI is that such a intelligent agent must have the ability to learn from examples and store the learned knowledge and then use the knowledge to reason upon encountering new environment or to make decision for activities. This is what *Neural-Symbolic Integration* seeks for. the author summarizes the reasons below:

1. We have discussed the blossom of Deep Learning in various areas in the Introduction section and also have showed the merits of neural networks as a full-fledged and robust learning agent in Section 2.1.
2. The *Physical Symbol System Hypothesis* asserts a strong *necessary* and *sufficient* condition guarantee of symbolic system for an intelligent agent.

3. One of the serious problem of symbolic system is the *Symbol Grounding Problem*, which we have discussed in Section 2.2.2. Harnad proposed[15] that symbolic representations should be grounded bottom-up by means of non-symbolic representations:iconic representations — sensory projections of objects — and categorical representations — invariant features of objects somehow warrants the integration with connectionist models.

3.1 Schematic Neural-Symbolic Integration

So what does a Neural-Symbolic system looks like? A great survey paper on this topic[3] depicts a Neural-Symbolic learning cycle as in Figure 2. The left-side Symbolic System is usually expressed in formal logic, a mature field that has been studied extensively and is based on solid mathematics logic background. This field is called *Logic Programming*[23] when we write these formal logic clauses in programming language. The right-side Connectionist System is the (deep) neural networks that we are familiar with. The important part is the *Representation* arrow going from left to right, that is to instill the symbolic information into Connectionist System, which is huge challenge that we will discuss in detail later. The opposite *Extraction* arrow going from right to left is to distill the *distributed* knowledge that is learned during training or inference(Reasoning) out of Connectionist System. By *distributed* the author means the learning process of a neural network is basically adjusting the weight of connections between neurons, and so the learned knowledge in stored distributively across all connections. This *Extraction* procedure is important and valuable, because it broadens our understanding of the knowledge that we might have never learned from the symbolic system before. We can further process these acquired knowledge by human experts and then feed back to the system. This closed circle empowers the Neural-Symbolic system the ability that pure symbolic systems or pure connectionist systems have never achieved. Note that these two arrows is to some degree a duality property, and that may inspire us that it would better to have sort of symmetry between these two opposite processes.

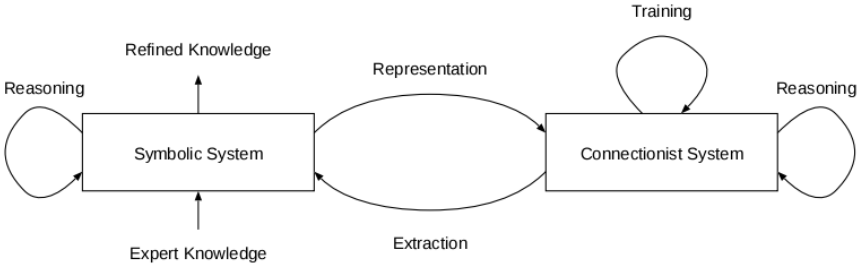

Figure 2: A neural symbolic learning cycle[4]

3.2 Challenges and Related Works

Now we can discuss the challenges of integrating symbolic knowledge into connectionist models, that is the *Representation* arrow in Figure 2.

In contrary to the order we define the First-Order and Propositional Logic in last section, the first symbolic system we tackle is the simpler Propositional Logic.

3.2.1 First-Order and Propositional Logic Preliminaries

To help readers to form a more concrete idea of what the challenges are while integrating symbolic knowledge into connectionist models and the approaches to tackle them, this section briefly recalls notations and definitions concerning a symbolic system, called *First-Order Logic*, in a top-down method for introducing concepts. For more rigorous definitions, please refer to [23].

In such a First-Order Logic symbolic system, we operates on a textitlogic program $\mathcal{P}$.

Definition 2. A *logic program* $\mathcal{P}$ is a finite sequence of alphabet letters, which consist of classes of symbols:

(a) **variables**(*e.g., a, x*);
(b) **constants**(*e.g., 1, Apple*);
(c) **function symbols**(*e.g., f, g*);
(d) **predicate symbols**(*e.g., isHuman(x)*);
(e) **connectives**: $\neg, \wedge, \vee, \leftarrow, \Leftrightarrow$, which mean *negate*, *and*, *or*, *implication* and *equivalent*, resepctively;
(f) **quantifiers**: $\forall, \exists$, which means *existential* and *universal*, respectively.

A logic program $\mathcal{P}$ structurally consists of a finite set of *clauses*.

Definition 3. A *clause* is of form:

$$A \leftarrow L_1, L_2, ..., L_n,\ n \geq 0;$$

where A is *head* and $L_1, ..., L_n$ is *body* of the *clause*. A is a *atom*, and $L_i, 1 \leq i \leq n$ is a *literal*.

Definition 4. An *atom* a is short for *atomic formula*, defined as $p(t_1, ..., t_n)$, where p is a *n-ary predicate symbol*, and $t_1, ..., t_n$ are *terms*.

Definition 5. A *term* is defined recursively as follows:

(a) A *variable* is a *term*;
(b) A *constant* is a *term*;
(c) f is an *n-ary function symbol* and $t_1, ..., t_n$ are *terms*, then $f(t_1, ..., t_n)$ is a *term*.

Definition 6. A *literal* is an *atom* or the negation($\neg$) of an *atom*.

[4]This figure is from [3].

We have defined (not comprehensively) the declarative syntax of a *logic program* $\mathcal{P}$. As for its semantics, or its meaning, we need to assign truth value($True$ or $False$) to the the *head* and *body* of each *clause*. This is called *interpretion*. We concern a very simple one called *Herbrand Interpretation*.

Definition 7. A *Herbrand Interpretation* I for a First-Order program is like this:

(a) every *connective* and every *quantifier* is interpreted as its fixed meaning, as is listed above;
(b) every *constant* is interpreted as itself;
(c) every *function symbol* is interpreted as the function that applies it
(d) every *predicate symbol* is interpreted by **grouding** the variables, that is binding *variables* with *constants*, to make it an *atom*.

So, according to (d), we need to pre-assign meanings(truth values) for these *atoms*. This set of *atoms* is called *Herbrand Base*, denoted as B_p.

Now that we have this **countable**(but not finite) set *Herbrand Base* , then by the definition of *Herbrand Interpretation*, we have a countable set of *Herbrand Interpretation*. So by this means, we have an interpretation for a *clause*:

Definition 8. The *meaning function*, or *semantic operator* $T_{\mathcal{P}} : 2^{B_P} \to 2^{B_P}$ is defined as follows: Let I be an interpretation and A a *grounded atom*. $A \in T_p(I)$ iff there exists a ground instance $A \leftarrow L_1, L_2, ..., L_n$ of a clause in $\mathcal{P}$ such that for all $1 \leq i \leq n$ we find $L_i \in I$.

Finally, we talk about a special case of *First-Order Logic*, that is *Propositional Logic*. It is defined by trimming the *predicate symbols* and *quantifiers* from *First-Order Logic*. It is much simpler and thus weaker in its expressiveness.

3.2.2 Challenges and Solutions

Now we can discuss the challenges of integrating symbolic knowledge into connectionist models, that is the *Representation* arrow in Figure 2.

In contrary to the order we define the *First-Order* and *Propositional Logic* in last section, the first symbolic system we tackle is the simpler *Propositional Logic*.

3.2.2.1 Propositional Logic Case

Now that *Propositional Logic* is a trimmed-off version of *First-Order Logic*, per the definition of *Herbrand Interpretation* above, it has no *Herbrand Base*, so we only need to consider the **finite** number of *function symbols* and **finite** number of *constants*, which means there are **finite** number of *atoms*, which in turn means there are only **finite** number of interpretations, thus a **finite** set of *meaning functions* for *clauses*.

So we can easily encode a *Propositional Logic* program by encoding its *meaning function* $T_{\mathcal{P}}$, instead of the program itself. More specifically:

1. In the input layer(the body), we assign one neuron for each *atom*.
2. In the output layer(the head), we also assign one neuron for each *atom*. The activation of neurons in these two layers represent the truth value of the corresponding *atom*.
3. For each *clause* of a *logic program* $\mathcal{P}$, there is a hidden layer neuron.
4. Connect the neuron in input layer(the body) to the hidden layer neuron, then to corresponding neuron in output layer(the head), according to the *clause*.

See Figure 3 for an example. This approach is adopted by [17].

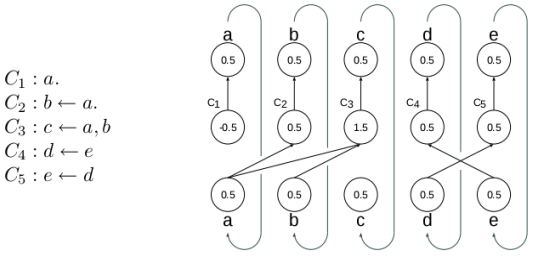

Figure 3: A Propositional program and the corresponding neural network[5]

3.2.2.2 First-Order Logic Case

This case is much trickier. Again, the idea is to represent the *meaning function* $T_{\mathcal{P}} : I_{\mathcal{P}} \to I_{\mathcal{P}}$ instead of the program $\mathcal{P}$ directly. The problem is that now we have **infinite**(but countable) number of *Herbrand Intepretation*. So it is not computationally feasible to encode the $T_{\mathcal{P}}$ as is done for the *Propositional Logic* case.

To tackle this problem, [18] maps the infinite but countable interpretations to real numbers by a *injective* mapping $R : 2^{B_p} \to \mathbb{R}$. This is feasible because the interpretation set is countable. And then it can map the *meaning function* $T_{\mathcal{P}}$ to a real value function $\overline{f}_p$, as is depicted in Figure 4. After we have this real value function $\overline{f}_p$, according to [11], we can construct a *feed forward network* that approximates $\overline{f}_p$ to this arbitrary degree of accuracy. So basically, [18] has formally proved theoretically that such a representation for a *First-Order Logic* program exists. However, this paper doesn't propose a constructive approach to do so. And also note that this paper constrains the kind of *First-Order Logic* program to be a so-called *acyclic program*.

Later in [4], such a constructive approach has be proposed, following the scheme in [18]. But this approach has some limitation due to the problem that the hardware floating point precision is very limited.

[5]This figure is from [3].
[6]This figure is from [18].

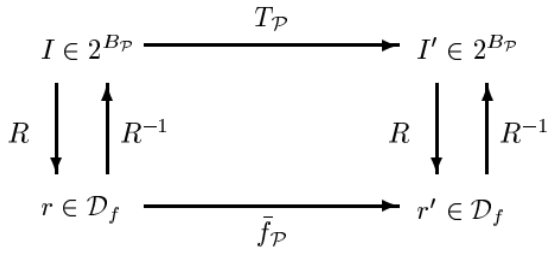

Figure 4: The relation between $T_{\mathcal{P}}$ and $f_{\mathcal{P}}$ [6]

There is another approach presented in [2], which representing a *First-Order Logic* program by means of *fibring neural networks*. In a *fibring neural network*, the activation of a neuron may influence other neruons by changing their weights. This approach also followed the the scheme in [18]. All in all, this approach addresses this problem in a compositional way of combining simple modules(fibring neural networks) into a ensemble.

4 New Research Direction

In this section, the author would like to propose a new research direction for integrating integrating symbolic knowledge into connectionist models.

4.1 Problems of Existing Approaches

In Section 3.2.2.2, we discuss the two constructive approaches to represent a *First-Order Logic* program and their problems. The problem exist in that fact that both follow the scheme in [18], that is, by encoding its *meaning function* $T_{\mathcal{P}}$, instead of the program itself. This is a ingenious idea though, however, in order to do so, a constriant was posed on the class of *First-Order Logic* programs that can be represented by neural networks: *acyclic logic programs*[1]. They are a subclass of the locally stratified programs, which enjoy several desirable properties like the fact the for each acyclic program $\mathcal{P}$ the meaning function $T_{\mathcal{P}}$ has a unique fixed point $M_{\mathcal{P}}$, which is a *minimal model*(*i.e., interpratation*). Put it simple, the contraint of *acyclic program* is to ensure $T_{\mathcal{P}}$ contracts to a fixed meaning.

The other problem is that to map $T_{\mathcal{P}}$ to real numbers in order to tackle the *infinite Herbrand Intepretation* problem, the $R : 2^{B_p} \to \mathbb{R}$ is required to be *injective*, which would be too strong.

The author tries to propose a new research direction, that we just encode the program itself instead of the meaning funciton $T_{\mathcal{P}}$. By this way, we can naturally eschew the problems above. The new idea comes fromand *The Curry-Howard Isomorphism*[33] and the relation between neural networks and Functional Programming languages[27].

4.2 The Curry-Howard Isomorphism

The awesome *Curry-Howard Isomorphism* is the relationship between *computer programs* and *mathematical proofs*, first discovered by the American mathematician Haskell Curry and logician William Alvin Howard. It unveils the profound isomorphism between two seemly unrelated fields.

To put it simplely and informally, it states that *a proof is a program, and the type of the program is analogous to the theorem(proposition) to be proved*. This means to prove a theorem, we are effectively finding a function whose type corresponding to the theorem(proposition)!

To find such a program, more detailed isomorphism between two fields are formalized, as depicted in Table 1. Note, for the program, the author will use the *Haskell* syntax[7].

Formal Logic	**Haskell Program**
universal quantifier($\forall$)	generalized product type(Π)
existential quantifier($\exists$)	generalized sum type(Σ)
implication($\leftarrow$)	function type($func :: a \rightarrow b$)
conjunction($\wedge$)	product type((a, b))
disjunction($\vee$)	sum type($data\ Either\ a\ b$)
True clause	unit type
False clause	bottom type

Table 1: The correspondence between formal logic and programs

So from this fact, recall our definition of *First-Order Logic* program in Section 3.2.1, the $T_{\mathcal{P}}$ of the clause

$$A \leftarrow L_1, L_2, ..., L_n,\ n \geq 0$$

is equivalent to define a *Haskell* function:

$$foo :: (L_1, L_2, ..., L_n) \rightarrow A$$

where "::" means "has type of...".

Literally, this function means "*Find a function foo, which accpts a sequence of arguments of type $L_1, L_2, ..., L_n$, respectively, and return a type of A*". Then here comes a new question:

How to represent such a function in a connectionist model?

The answer comes from the analogy between *functional programming* and neural networks.

[7]Actually, functional programming is one of the outcome of *The Curry-Howard Isomorphism*, and *Haskell* is a popular and versatile pure functional language.

4.3 The Functional Programming Perspective of Neural Networks

One distinctive perspective proposed in [27] is to draw analogy bwtween neural networks and *functional programming*. To quote the narrative in that article:

> *With every layer, neural networks transform data, molding it into a form that makes their task easier to do. We call these transformed versions of data "representations. Representations correspond to types. ...Just as two functions can only be composed together if their types agree, two layers can only be composed together when their representations agree.*

Compositionality is of key importance to *functional programming*[20]. So the keen insight of [27] is that it captures the *Compositionality* of layers of neural networks(which we've asserted that it is also key to the success of neural networks in Section 2.1.2) is equivalent to the *composition* of functions in *functional programming*.

More than that, [27] further points that "*neural network patterns are just higher order functions that is, functions which take functions as arguments*". That means some common *nerual netwrok patterns* works just like building block for composing a more complex networks. For example, an *Encoding RNN*(Figure 5) is just like the *foldl* function in *Haskell*.

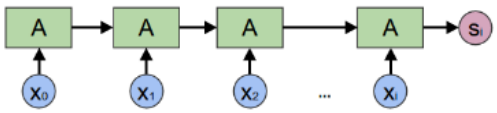

Figure 5: An Encoding RNN, which is like the $foldl :: (b \to a \to b) \to b \to [a] \to b$ in *Haskell*, which accpets *three* arguments: one *function*, one *accumulator*, and one *array*. What it does is quite simple: iterating this array, applying the function to every element of the array, and accumulating the result into the *accumulator*, and finally returning the *accumulator*. This is what exactly a RNN is doing: recurrently comsue a sequence of input and *fold* into the final outcome.

4.4 The Final Approach

So compare the function *foo* with *foldl*, we can just encode *foo* in *foldl*:

$$foo = foldl\ and\ True\ (L_1, L_2, ..., L_n)$$

Accordingly, we can encode this *logic program clause* in a RNN! So the final encoding of the logic program can be a ensemble of RNNs. Specifically, if a term

in the *body* of is a *ground atom*, it is just like we have a constant RNN input; if a term in the *body* is a *predicate symbol*, it is another RNN, and we just connect its output to this RNN.

The benifits of this approach are:

1. We get rid of constraint of *acyclic programs*.
2. We can encode any *logic program* in a bottom-up compositional way, following a definite procedure, which may be learned by a *Meta-Learning* system.
3. We have discussed in Section 3.1 that the *Representation* and *Extraction* phases are kind of duality. Using this compositional method, we are assumed to easily *decompose* the structure, that is, to distill the newly-learned knowledge in the connectionist model to symbolic forms: some new proposition! This direction worths future research.

5 Conclusions

In this article we discuss in deatil the merits of neural networks(the connectionist model) and the limitations of symbolic systems, and to combine the strength and weakness of both system, we also discussed the challenges, and the approaches to address them. We propose a new approach to instill the symbolic knowledge into connectionist models, inspired by the analogy between the *formal logic* and *programming*, and the analogy between *functional programming* and *neural networks*. The author believes that his transitive isomorphism shed light on the future research on this *Neural-Symbolic Integration*. However, we still lack empirical research for this approach, and it remains to be seen the real stregnth and problem of this approach.

References

[1] Apt, K. R., and Bezem, M. Acyclic programs. *New generation computing 9*, 3-4 (1991), 335–363.

[2] Bader, S., Garcez, A. S. d., and Hitzler, P. Computing first-order logic programs by fibring artificial neural networks. In *FLAIRS Conference* (2005), pp. 314–319.

[3] Bader, S., and Hitzler, P. Dimensions of neural-symbolic integration-a structured survey. *arXiv preprint cs/0511042* (2005).

[4] Bader, S., Witzel, A., and Hitzler, P. Integrating first-order logic programs and connectionist systems-a constructive approach. In *Proceedings of the IJCAI-05 Workshop on Neural-Symbolic Learning and Reasoning* (2005).

[5] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems* (2007), pp. 153–160.

[6] Bengio, Y., LeCun, Y., et al. Scaling learning algorithms towards ai. *Large-scale kernel machines 34*, 5 (2007), 1–41.

[7] Bianchini, M., and Scarselli, F. On the complexity of shallow and deep neural network classifiers. In *ESANN* (2014).

[8] Bottou, L. From machine learning to machine reasoning. *Machine learning 94*, 2 (2014), 133–149.

[9] Dreyfus, H. L. What computers can't do.

[10] Frank, R. The perceptron a perceiving and recognizing automaton. *tech. rep., Technical Report 85-460-1* (1957).

[11] Franklin, J. On the approximate realization of continuous mappings by neural networks. *Neural Networks 2* (1989), 183–192.

[12] Garcez, A. S., Lamb, L. C., and Gabbay, D. M. *Neural-symbolic cognitive reasoning*. Springer Science & Business Media, 2008.

[13] Goodfellow, I., Bengio, Y., and Courville, A. Deep learning. book in preparation for mit press. *URL¡ http://www. deeplearningbook. org* (2016).

[14] Hammer, B., and Gersmann, K. A note on the universal approximation capability of support vector machines. *Neural Processing Letters 17*, 1 (2003), 43–53.

[15] Harnad, S. The symbol grounding problem. *Physica D: Nonlinear Phenomena 42*, 1-3 (1990), 335–346.

[16] Hinton, G. E., Osindero, S., and Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural computation 18*, 7 (2006), 1527–1554.

[17] Hölldobler, S., Kalinke, Y., Ki, F. W., et al. Towards a new massively parallel computational model for logic programming. In *In ECAI94 workshop on Combining Symbolic and Connectioninst Processing* (1991), Citeseer.

[18] Hölldobler, S., Kalinke, Y., and Störr, H.-P. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence 11*, 1 (1999), 45–58.

[19] Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural networks 2*, 5 (1989), 359–366.

[20] Hughes, J. Why functional programming matters. *The computer journal 32*, 2 (1989), 98–107.

[21] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[22] Lin, H. W., Tegmark, M., and Rolnick, D. Why does deep and cheap learning work so well? *Journal of Statistical Physics* (2016), 1–25.

[23] Lloyd, J. W. *Foundations of logic programming*. Springer Science & Business Media, 2012.

[24] McCarthy, J. Elaboration tolerance. In *Common Sense* (1998), vol. 98.

[25] Newell, A. Physical symbol systems. *Cognitive science 4*, 2 (1980), 135–183.

[26] Newell, A., and Simon, H. A. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM 19*, 3 (1976), 113–126.

[27] Olah, C. Neural networks, types, and functional programming, 2015. *URL http://colah. github. io/posts/2015-09-NN-Types-FP*.

[28] Pan, S. J., and Yang, Q. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering 22*, 10 (2010), 1345–1359.

[29] Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., and Liao, Q. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review. *International Journal of Automation and Computing* (2017), 1–17.

[30] Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. Learning representations by back-propagating errors. *Cognitive modeling 5*, 3 (1988), 1.

[31] Searle, J. R. Minds, brains, and programs. *Behavioral and brain sciences 3*, 3 (1980), 417–424.

[32] Sharma, J., Angelucci, A., and Sur, M. Induction of visual orientation modules in auditory cortex. *Nature 404*, 6780 (2000), 841.

[33] Sørensen, M. H., and Urzyczyn, P. *Lectures on the Curry-Howard isomorphism*, vol. 149. Elsevier, 2006.

[34] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research 15*, 1 (2014), 1929–1958.

[35] Zeiler, M. D., and Fergus, R. Visualizing and understanding convolutional networks. In *European conference on computer vision* (2014), Springer, pp. 818–833.

[36] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530* (2016).