Some Thoughts on Logic, Declarative Programming, and Knowledge Representation

Michael Gelfond

January, 2021

My Scientific Interests

During last forty years of my scientific life I was mostly interested in two things:

- Achieving better understanding of the human mind and structure of the universe by discovery, refinement, and formalization of the basic categories of our language and thought.
- Developing a good methodology for the design and implementation of transparent and elaboration tolerant software systems.

The first desire led me to a traditional activity of a logician.

I started with working in the area of constructive logic, concerned with nature of mathematical objects and their existence, and moved to trying to better understand things like rational beliefs, causality, etc.

This activity did not require any justification. Pure joy of replacing chaos by order and clarity was enough.

The impact of logician's activity, however, may be much broader and more consequential.

G. Leibniz viewed this activity as second in importance only to the activities of prophets and kings.

Universal Symbolism

In the 17th century Leibniz proposed the idea of developing *universal symbolism* (*Characteristica Universalis*):

- **a *universal notation* by use of which any item of information whatever can be recorded in a natural and systematic way,**
- **a *means of manipulating the knowledge* thus recorded in a computational fashion, so as to reveal its logical interrelations and consequences.**

More Citations:

"*A man who is neither a prophet nor a prince can ever undertake any thing of greater good to mankind and more fitting for divine glory*".

“*mankind is still not mature enough to lay claim to the advantages which this method could provide.*”

Among those he included much better understanding of nature and man, great scientific progress, and even end to religious strife and wars, achieved by reducing disputes to computation.

We still have no Universal Symbolism and not yet learn to gracefully use parts of the symbolism which have already been developed.

However, logicians interested in good of mankind and/or divine glory continue striding toward the goal.

Perhaps eventually we will mature enough to fully develop and use the method.

In this talk I'll discuss several steps in this development (some big and some small) which, I believe, worth remembering and thinking about.

George Boole

Important step toward Leibniz goal was made by Boole, who believed that *"A successful attempt to express logical propositions by symbols, the laws of whose combinations should be founded upon the laws of the mental processes which they represent, would be a step towards a philosophical language."*

In his "An Investigation of the Laws of Thought" (1854) Boole did exactly that.

In this book certain type of reasoning was reduced to *Boolean Algebra* which is one of the foundations of our information age.

Axiomatization of Mathematics

In the first half of 20th century ideas of Boole and much earlier work of Euclid led to

- **Expanding Boolean Logic by quantifiers, new laws and inference rules (Frege).**
- **Finding small number of axioms of Set Theory from which (almost) all mathematical theorems can be deduced by a small number of inference rules (Russel and Whitehead).**

  (Godel showed that "*almost*" in this statement is inevitable.
  There are always going to be true propositions not captured by such axioms and rules.)

Beyond Mathematics

A. Tarski started a program of expanding axiomatic method from mathematics to other sciences.

The program, further advocated by J. McCarthy and others, so far proved to be most successful in Computer Science, where it lead to the development of

- **Declarative Programming Paradigm,**
- **Logic Based Approach to AI.**

Declarative Programming

Declarative Programming is an *attempt to apply axiomatic method of mathematics to computer science and to practice of programming* (and, hence, closely related to my second scientific interest).

It suggests design methodology according to which

- knowledge relevant to a given class of computational problems is represented by a theory (collection of axioms) in some precise mathematical language.
- A computational problem is reduced to computing consequences or models of this theory.
- General reasoning algorithms are used to compute these consequences or models.

Declarative Languages

Representation of knowledge requires *formal languages – means for thinking about computational problems and for communicating information about them*.

Languages are often divided into two basic types:

- Algorithmic languages describe sequences of actions to be performed to achieve some goal.
- Declarative languages describe properties of objects an agent can use to form rational beliefs, reason and act toward achieving the desired goal.

Deductive Databases: the Family Example

***Given*:**
a database containing entries for three people - John, Mary, and their son, Sam.

$$father(john, sam).$$
$$mother(mary, sam).$$

***Problem*:**

Teach the database a new family relation, *parent*, and check if the term is understood.

The following two statements can be viewed as a definition of relation *parent* written in a declarative language called *Datalog*:

$$\begin{array}{lcl} parent(X,Y) & \leftarrow & father(X,Y) \\ parent(X,Y) & \leftarrow & mother(X,Y). \end{array}$$

We use it together with database facts

$$\begin{array}{l} father(john, sam). \\ mother(mary, sam). \end{array}$$

To make sure that the program understood the notion of *parent* we test it by asking a number of questions to be answered by an *inference engine* of Datalog.

If the answers are satisfactory the program learned.

(This is exactly the method we use to check human understanding.)

Is John a parent of Sam?

$$?\mathbf{parent}(\mathbf{john}, \mathbf{sam}) \qquad \textit{Yes}$$

Who are Sam's parents?

$$?\mathbf{parent}(\mathbf{X}, \mathbf{sam}) \qquad X = john,\ X = mary$$

Why is this solution interesting?

- Thanks to our language, the thought process which led to this solution is very different from that used in traditional programming.
- The solution is short and easy to understand. Mathematical definition of parents is very close to the informal definition. No knowledge of data structures and search algorithms is required.
- The program has a high degree of elaboration tolerance. It is not difficult to expand.

Expanding the Program

To illustrate these features we expand the program by

(a) recursive definition of $\mathit{ancestor}$**:**

$$\begin{aligned}\mathit{ancestor}(X,Y) &\leftarrow \mathit{parent}(X,Y).\\ \mathit{ancestor}(X,Y) &\leftarrow \mathit{parent}(X,Z),\\ & \mathit{ancestor}(Z,Y).\end{aligned}$$

(b) Definition of relation $\mathit{numOfChildren}$**:**

$$\mathit{numOfChildren}(N,P) \leftarrow \mathrm{count}\{X : \mathit{parent}(P,X)\} = N$$

Here count **is a function on sets. (Such functions are called aggregates).**

Discussion

Recursive definition is one of basic categories of our language. It is very important for our thought process.

The semantics of Datalog (given in terms of minimal Herbrand model of a program) provides an accurate account of this category.

The language also allows aggregates but prohibits definitions which use recursion through aggregates.

Our program has no such definitions. It exhibits expected behavior and can be proven correct.

We still have no universally accepted semantics for recursive aggregates.

There is a number of competing approaches which may disagree on questions like:

What should be the models of programs

$$p(1) \leftarrow count\{X : p(X)\} \geq 0$$

and

$$p(1) \leftarrow count\{X : p(X)\} = Y, \ Y \geq 0 \quad ?$$

Can one of them be consistent and another inconsistent?

To answer we need better understanding of self-reference and methodology of language design.

More things to know about Datalog

- Datalog programs are easier to develop, maintain, and modify than traditional DBMS; for recursive queries they outperform such systems by orders of magnitude.
- Datalog inference engines often use traditional SQL techniques to get efficient access to huge collections of data.
- Extension of Datalog (e.g., Answer Set Prolog (ASP)) allow reasoning with incomplete information.
- Applications of Datalog include data integration, information extraction, networking, program analysis, security, cloud computing, machine learning, etc.

Declarative Programming in KR and AI

The next two examples illustrate applications of declarative programming to two classical AI problems:

- **Formalization of reasoning with defaults and**
- **Formalization of reasoning about actions and their effects.**

Both problems are crucial for formalizing commonsense reasoning and for the development of methodology for the design of intelligent agents capable of acting in a changing environment.

Reasoning with Defaults

In everyday reasoning we often use *defaults* - statements of the form:

"*normally, elements of class* C *have property* P".

It seems that a substantial part of our education consists of learning various defaults and methods of reasoning with defaults and their exceptions.

The problem of understanding and formalizing default reasoning remained unsolved for decades. It is mostly solved now.

An Example of Default Reasoning

Given a database:

$$\begin{array}{c} parent(bob, mary) \\ parent(john, sam) \end{array}$$

together with a default:

"*Parents normally love their children*"

one expects answers to queries

?love(john,sam)

?love(bob,mary)

to be *yes*.

Non-monotonic Consequence Relations

If later we learn that

“***John is an exception to this rule. He hates his son.***”

then our previous conclusion “$love(john, sam)$” will be withdrawn.

A consequence relation which allows a reasoner to withdraw previous conclusions given new information is called non-monotonic.

Non-monotonic consequence does not appear in mathematics and hence has not been studied in classical mathematical logic.

Non-monotonic Logics

Problem: ***Develop new logics with non-monotonic consequence relations.***

First such logics appeared around 1980 (McCarthy, Reiter, McDermott and Doyle).

ASP (Answer Set Prolog) is probably most popular such logic to date.

According to Stanford Encyclopedia of Philosophy ***the emergence of non-monotonic logics is one of the most significant developments both in logic and artificial intelligence*****.**

Defaults and Default Negation in ASP

Default "*parents normally love their children*" is represented as

$$\begin{array}{lcl} \mathrm{loves}(X,Y) & \leftarrow & \mathrm{parent}(X,Y), \\ & & \quad \mathbf{not}\ \neg \mathrm{loves}(X,Y). \end{array}$$

Representation contains two new logical connectives:

- **"$\neg p$", read as "p *is false*"**
- **"not p", read as "*there is no reason to believe that* p *is true*"**

The rule reads: ***If* X *is a parent of* Y *and there is no reason to believe that he does not love* Y *then he does.***

Default Reasoning

Program

$$\begin{aligned} loves(X, Y) \quad \leftarrow \quad & parent(X, Y), \\ & \textbf{not } \neg loves(X, Y) \end{aligned}$$

$$parent(john, sam)$$

entails

$$loves(john, sam).$$

But, addition of a new fact:

$$\neg loves(john, sam)$$

forces the program to replace this conclusion by

$$\neg loves(john, sam).$$

Consistency-Restoring Rules

A general solution to representing defaults with exceptions uses a new language construct, *consistency-restoring rule* (cr-rule), which has the form:

$$l \overset{+}{\leftarrow} body$$

It says that
if the reasoner associated with the program believes the body of the rule, then it "may possibly" believe that its head is true.

However, this possibility may be used only if there is no way to obtain a consistent set of beliefs by using only regular rules of the program.

Capturing Indirect Exceptions

Consider a program

$$\begin{array}{lcl} p(X) & \leftarrow & c(X), \text{not } \neg p(X). \\ q(X) & \leftarrow & p(X). \\ c(a). & & \end{array}$$

which entails $p(a)$ and $q(a)$ but, after addition of

$$\neg q(a)$$

becomes inconsistent.
The consistency can be restored by cr-rule

$$\neg p(X) \overset{+}{\leftarrow} c(X).$$

allowing indirect exceptions to the default.

Discussion

The example shows how a long standing problem in knowledge representation found a simple solution via the development of logical languages containing new non-monotonic language constructs, such as default negation and cr-rules.

Currently, we have powerful languages with non-monotonic inference, reasonably efficient reasoning systems, and non-trivial mathematical theory.

The next unexplored step is the axiomatization of large parts of commonsense knowledge.

Representing Effects of Actions

To reason about the story

"*Initially, John was in the corridor holding a book. Then he stepped in the office*"

we need to axiomatize effects of action $\mathsf{move}(\mathsf{P}, \mathsf{L})$**.**

The initial state, S_0 **can be described as**

$$\begin{array}{l} \mathsf{holds}(\mathsf{in}(\mathsf{john}, \mathsf{corridor}), 0) \\ \mathsf{holds}(\mathsf{in}(\mathsf{book}, \mathsf{corridor}), 0) \\ \mathsf{holds}(\mathsf{holding}(\mathsf{john}, \mathsf{book}), 0) \\ \mathsf{occurs}(\mathsf{move}(\mathsf{john}, \mathsf{office}), 0) \end{array}$$

(We use 0 **and** 1 **as the initial and final time-steps.)**

Effects of Actions

This axiomatization, together with the initial state, should be able to conclude

$$\begin{array}{l} holds(in(john, office), 1) \\ holds(in(book, office), 1) \\ holds(holding(john, book), 1) \end{array}$$

To achieve this we formalize

(a) direct and indirect effects of $move$ **and**

(b) Inertia Axiom which says:
"things normally remain as they are."

Direct and Indirect Effects

Direct effect:

$$\mathsf{holds}(\mathsf{in}(P, L), T+1) \leftarrow \mathsf{occurs}(\mathsf{move}(P, L), T)$$

***Indirect effects* are consequences of constraints:**

$$\begin{array}{lll} \mathsf{holds}(\mathsf{in}(O, L), T) & \leftarrow & \mathsf{holds}(\mathsf{in}(P, L), T) \\ & & \mathsf{holds}(\mathsf{holding}(P, O), T) \end{array}$$

$$\neg \mathsf{holds}(\mathsf{in}(X, L_2), T) \leftarrow \mathsf{holds}(\mathsf{in}(X, L_1), T), L_1 \neq L_2$$

The axioms elegantly describe changes the action causes in the domain but say nothing about things which remain unchanged.

Frame Problem and Its Solution

The challenge of finding succinct representation of what does not change constitute a famous *Frame Problem* (McCarthy and Hayes, 1969).

As a solution they suggested the use of *Inertia Axiom* (Leibniz) — "*Things tend to stay as they are*".

The axiom is a default and no one knew how to represent the defaults. Now we do.

$$\begin{array}{lcl} \mathsf{holds}(F, T+1) & \leftarrow & \mathsf{holds}(F, T) \\ & & \textbf{not } \neg\mathsf{holds}(F, T+1) \end{array}$$

$$\begin{array}{lcl} \neg\mathsf{holds}(F, T+1) & \leftarrow & \neg\mathsf{holds}(F, T) \\ & & \textbf{not } \mathsf{holds}(F, T+1) \end{array}$$

Discussion

As expected, the resulting program derives

$$\begin{array}{ll} holds(in(john, office), 1) & \neg holds(in(john, corridor), 1) \\ holds(in(book, office), 1) & holds(holding(john, book), 1) \end{array}$$

The solution works for a large range of discrete dynamic systems, i.e., state-action-state transition diagrams.

Direct effects of actions, constraints, and the inertia concisely describe the diagram.

This description forms the basic part of programs which find plans for achieving the agent's goal, explanations for unexpected events, etc. This methodology is now used in robotics, decision support, question-answering, etc.

ASP in Applications

Development of ASP based logics capable of representing and reasoning with recursive definitions, aggregates, defaults, effects of actions, etc., was going hand in hand with discovery, optimization, and implementation of reasoning algorithms.

This led to the emergence of remarkably efficient reasoning systems which made ASP a practical tool of declarative programming.

This is not the focus of my talk but, for illustrative purposes, in the next few slides I will briefly comment on one such application.

An Industrial Application

A story by Gerhard Friedrich – former head of the Department for Configurations and Diagnosis at Siemens, Germany.

Worked on configuration problem for more than 20 years. Used different methods, including procedural programming, constraint programming, etc.

The problem is difficult. E. g. in telecommunication domain, just focusing on hardware:

- **Up to 30.000 modules of 200 types**
- **Up to 1.000 frames of 50 types**
- **Up to 200 racks of 20 types**
- **Up to 10.000 cables of 50 types**

Recent use of ASP allowed

- Reduction of initial development cost by 66%
- Reduction of yearly maintenance cost by 80%
- Productivity increase by 300% (no additional staff)
- Enhanced user interaction: explanations, incremental configuration, repair ...

In this case ASP delivered on its promise.

Friedrich gave an example of ASP program consisting of 12 rules.

Quote: " *By this simple ASP program, we solved real world instances which could NOT be solved by the in-house tool*"

In his estimate "*ASP is a historical leap for AI*".

In my judgment, we are still very far from realizing the ASP potential for industrial applications, but are moving in the right direction. We need to

(a) improve efficiency of algorithms and implementations for ASP and its extensions and

(b) teach logic!

Understanding Uncertainty

One of the most important tools for understanding uncertainty is the theory of *probability*.

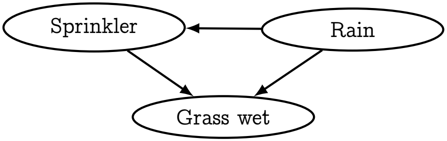
This was a focus of *probabilistic approach to AI*, which developed powerful knowledge representation tools and reasoning methods utilizing graphs and Bayesian reasoning.

In this approach probability of a proposition A is understood as the degree of agent's belief that A is true.

Here is an example of graphical representation of probability:

Causal Bayesian Network: Example

	Sprinkler	
Rain	T	F
F	0.4	0.6
T	0.01	0.99



Rain	
T	F
0.2	0.8

		Grass wet	
Sprinkler	Rain	T	F
F	F	0.4	0.6
F	T	0.01	0.99
T	F	0.01	0.99
T	T	0.01	0.99

This is a remarkable way to succinctly describe probabilistic models.

- Possible worlds – assignments of values to variables, e.g $\{rain, \neg sprinkler, grassWet\}$.
- Probability measures of the worlds (given by tables).
- Probability function defined on sets of such worlds.

Arrows between the nodes indicate *causal dependencies* which allows to distinguish probability of events conditioned on observations from probability of events conditioned on deliberate actions.

Work on causal Bayesian Network and Structural Equations clarified the relationship between probability and causality and led to great achievements in AI, statistics and other areas dealing with decision making.

For a long time logical and probabilistic approaches to AI developed independently, but now there is a substantial amount of interaction.

I believe that we actually *have one single paradigm combining both, logical and probabilistic reasoning.*

Boole on Probability and Logic

This view on relationship between logic and probability goes back to Boole.

Common foundations are evident from the full title of Boole's book: *"An Investigation of the Laws of Thought on which are founded the Mathematical Theories of Logic and Probabilities".*

He viewed probability of an event as the "*degree of belief that it has taken place, or that it will take place*".

Reasoning about probability is simply reasoning about degrees of belief.

Logic and Probability under one Roof

In the original interpretation of ASP a program's model does not represent possible state of the world but rather possible state of agent's beliefs.

Hence, ASP has three degrees of belief:

- $p \in M$ – **the agent believes that** p **is true,**
- $\neg p \in M$ – **agent believes that** p **is false,**
- **otherwise, agent neither believes nor disbelieves** p**.**

Plog is an extension of ASP for reasoning with probabilistic knowledge. It allows arbitrary degrees of belief from $[0, 1]$**.**

Here is an example.

Example: Monty Hall Problem

A player selects one of three closed doors, behind one of which there is a prize.

After selection is made, Monty is obligated to open one of the remaining doors which does not contain the prize.

The player can switch his selection to the other unopened door, or stay with his original choice.

Does it matter if he switches? The answer is *yes*.

However, the lady who published the solution received thousands of letters from her readers—the vast majority of which, many with PhD in math, disagreed with her answer.

This phenomenon was clearly recognized by Boole who wrote:

"I think it to be one of the peculiar difficulties of the theory of probabilities, that its difficulties sometimes are not seen. The solution of a problem may appear to be conducted according to the principles of the theory as usually stated; it may lead to a result susceptible of verification in particular instances; and yet it may be an erroneous solution."

One possible way to alleviate the difficulty is to define probability with respect to explicitly stated knowledge base of the reasoner.

Problem Description

Player's knowledge in P-log:

$$\begin{aligned} \text{Declarations}: \quad & \text{doors} = \{1,2,3\} \\ & \text{selected}, \text{prize}, \text{open} : \text{doors} \\ & \text{canOpen} : \text{doors} \rightarrow \text{boolean} \end{aligned}$$

$$\begin{aligned} \text{Rules}: \quad \text{canOpen}(D) \leftarrow & \ \text{not} \ \neg\text{canOpen}(D) \\ \neg\text{canOpen}(D) \leftarrow & \ \text{selected} = D \\ \neg\text{canOpen}(D) \leftarrow & \ \text{prize} = D \end{aligned}$$

The first rule is the default: *"Normally, a door can be opened"*.

The next two rules are exceptions to this default.

Probabilistic Part:

Values of variables $prize$ **and** $selected$ **are chosen at random from the set of all doors.**

$random(prize)$
$random(selected)$

Value of $open$ **is chosen at random from the set of doors which can be open according to the rules of the game:**

$random(open : \{X : canOpen(X)\})$

According to the semantics of P-log:

$random(prize)$

can be replaced by rule

$$prize = 1 \textbf{ or } prize = 2 \textbf{ or } prize = 3$$

and $random(open : \{X : canOpen(X)\})$ **by rules**

$$open = 1 \textbf{ or } open = 2 \textbf{ or } open = 3$$

$$\leftarrow \neg canOpen(X), open(X)$$

Player selects the first door

Add

$$\text{do}(select = 1)$$

to the program.

The new program defines the following possible worlds:

$W_1 = \{sel = 1, prize = 1, canOpen(2), canOpen(3), open = 2\}$

$W_2 = \{sel = 1, prize = 1, canOpen(2), canOpen(3), open = 3\}$

$W_3 = \{sel = 1, prize = 2, canOpen(3), open = 3\}$

$W_4 = \{sel = 1, prize = 3, canOpen(2), open = 2\}$

Probabilistic measures of W_1 and W_4 are

$$\mu(W_1) = 1/3 \times 1/2 = 1/6$$
$$\mu(W_4) = 1/3 \times 1 = 1/3$$

Player observes Monty's actions, makes the decision

Suppose Monty opened door 2 and the player recorded observations:

$$\mathrm{obs}(\mathrm{open} = 2). \qquad \mathrm{obs}(\mathrm{prize} \neq 2).$$

Updated program has two possible worlds: W_1 **and** W_4.

Since $\mu(W_4) = 2 \times \mu(W_1)$

changing the door doubles player's chances to win.

Discussion

Rules of the game written in P-log are close to their informal descriptions.

In fact, solution to the problem consists in simply stating these rules. The rest is automatic.

Its worth noting that if rule

$$\neg canOpen(D) \leftarrow \; selected = D$$

were removed from the program then changing the door would not change the chance of winning.

I suspect that confusions surrounding this and other similar problems are often caused by the absence of precise specification.

Syntax and semantics of P-log are close to that of ASP.

Possible worlds **of the program are answer sets of its ASP translation.**

Probabilistic measure is defined in natural way using the Indifference Principle: ***Possible outcomes of a random experiment are assumed to be equally probable if we have no reason to prefer one of them to any other.***

There are also causal probability statements of the form

$$pr(A|_c B) = v$$

which are used for computing probabilistic measures.

Another Example

Program Π below represents knowledge about whether a certain rat will eat arsenic today and whether it will die today.

$$\begin{array}{l}
arsenic, death : boolean \\
random(arsenic) \\
random(death) \\
pr(arsenic) = 0.4 \\
pr(death \mid_c arsenic) = 0.8 \\
pr(death \mid_c \neg arsenic) = 0.01
\end{array}$$

Index c in symbol $|_c$ indicates causal dependency. It's use in the program expresses the fact that the rat's consumption of arsenic carries information about the cause of its death.

The program demonstrates the difference between observations and deliberate actions. It entails:

$$\begin{aligned} &P_\Pi(\mathsf{arsenic}) = 0.4 \\ &P_\Pi(\mathsf{arsenic} \mid \mathsf{obs}(\mathsf{death})) = 0.982 \\ &P_\Pi(\mathsf{arsenic} \mid \mathsf{do}(\mathsf{death})) = 0.4 \end{aligned}$$

where P_Π is the probability function defined by Π.

Observation of death increases our belief that the rat had eaten arsenic.

Deliberately killing the rat does not.

As pointed out by Pearl, the difference between conditioning on observations and on deliberate actions is very important in decision making and other types of causal reasoning.

Discussion

It is worth noting that conditional probability $P_\Pi(A \mid B)$ is defined simply as $P_{\Pi \cup B}A$.

So, in some respects it is more general than the usual one.

We can condition on *defaults, rules introducing new terms, deliberate actions in the sense of Pearl*, etc., which is not possible in "classical" probability theory.

While conditioning on observations only eliminates possible worlds of the program it is not necessarily the case in general.

Discussion

Consider program Π

$$\begin{aligned}&a, b : boolean\\&a \leftarrow \textbf{not } b\\&random(a) \leftarrow b\end{aligned}$$

The program has one possible world $\{a\}$**. Hence**

$$P_{\Pi}(a) = 1.$$

But expanding the program by b **produces a new possible world containing** $\neg a$**. We have**

$$P_{\Pi \cup \{b\}}(a) = 0.5$$

Addition of new information changed our probabilistic model.

Discussion

These features, which are not available in “classical” probability, clarify subtle questions about probabilistic reasoning and facilitate modeling of probabilistic domains.

There is a number of other interesting languages combining logical and probabilistic reasoning.

There are interesting applications to learning, robotics, and other domains.

Unfortunately, the number of people working in these areas is comparatively small and many important questions related to this topic remain unanswered.

Learning

So far we assumed that knowledge needed to design an agent can be extracted from experts and obtained by introspection.

This is not always the case. We may need to extract relationship between variables from a large collection of unstructured data.

There are many approaches to developing learning algorithms which address this task.

The idea is to set up basic parameters about the data and train the program to learn on its own by recognizing patterns.

Neural Networks

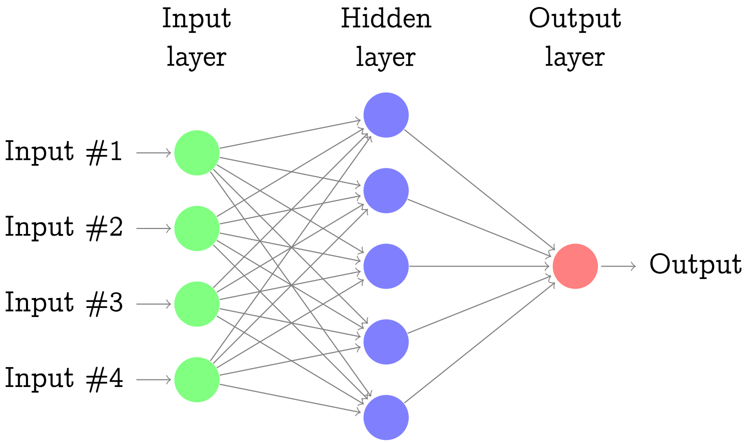
Important recent advances in learning algorithms are based on the notion of *neural network*.

It goes back to Frank Rosenblatt (1958) who introduced two layers neural network, which defined a particular type of functions.

It was quickly shown that such networks cannot predict the outputs of XOr logic gates, which slowed down research in this area.

It was later realized that the use of additional layers allow networks to represent very large class of functions.

A neural network



There are nice algorithms which, *given parameters* X *and* Y *and a training data from a large data set* D*, learn a neural network* N *which approximates the relationship between* X *and* Y *in* D.

For a long time these algorithms, however, were too slow.

In the last decade, the efficiency of reasoning with neural networks increased dramatically.

This, together with availability of huge amount of data, led to the development of the field of Deep Learning with its remarkable applications.

Logic and Deep Learning

Logic is beginning to be used to speed up learning algorithms and explain their results.

The basic idea is to supply the algorithm with knowledge base T **describing known dependencies between** X **and** Y **and use these dependencies during the learning process.**

A number of recent results show that this method substantially increases efficiency of learning a neural net N **which approximates the relationship between** X **and** Y **in** D**.**

The net N can be used to discover a new set of dependencies R such that neural network built using knowledge base $\mathsf{T} \cup \mathsf{R}$ instead of T defines essentially the same function as N.

These newly discovered properties of data can help to explain the algorithm's results.

It would be interesting to see accurate formulations and proofs of correctness for this type of algorithms.

It is also interesting to investigate if deep learning can help to learn common sense axioms which are currently discovered by introspection.

This type of work is new and more insights will be needed before we really understand the picture.

Conclusion

One goal of this talk was to bring to my mind and to the minds of my listeners clearer awareness of a continuous line of thought aimed at eventual realization of Leibniz Dream.

The progress always looks slower than we want but in reality our understanding increases at a fast rate.

To those of you who are interested in making Leibniz Dream come true my advise is: "*keep calm and carry on*".
Remember that your work is second only to that of Prophets and Kings (and since no true Prophets and Kings are currently left in the world, it is second to none.)

Some Problems to Solve

- Develop better understanding of causality.
- Figure out how large amount of knowledge can be organized to allow an agent effectively identify and access its parts relevant to the agent's goals.
- Axiomatize and combine together various pieces of commonsense knowledge. Concentrate at some basic notions such as *crossing the boundary*, *exchange*, *refinement*, *rights and obligations*, etc.
- Better understand traditional topics of logic such as relationship between knowledge and belief, beliefs and actions, the nature of counterfactual, etc., and bring them to the realm of Declarative Programming.

- Study a logical theory together with the set of operations defined on it (similar to move from data structures to abstract data types).
- Study behavior of logical systems in the context of agent architecture.
- Figure out when one theory can be viewed as a refinement of another, etc.

THANK YOU