> The greatest benefit to recognizing patterns is that it gives you a pathway to power, and a ladder out of chaos.
> *Tony Robbins*

Here we introduce a framework to describe the computation process of pattern recognition.

0.1 Pattern Recognition

In this subsection, we give intuitions about what pattern recognition is and we build a connection between pattern recognition and pattern matching in functional programming.

Intuitively pattern recognition is a function that is (using classical computers)

- computable. For the very least, in polynomial time and space.
- learnable. Either theoretically or empirically.

The importance of computability is needless to say. Learnability is critical for scaling, because it's much cheaper to train on machines than hiring millions of programmers to hand write everything.

Examples of pattern recognitions are

- Linear function. Computability is obviously. It's learnable both theoretically and empirically.
- Various tasks in computer vision. Empirically shown by neural networks.
- Various tasks in natural language processing. Empirically shown by neural networks.

Neural networks themselves are not viewed as "pattern recognition" because they can be hard to learn (the learning problem in neural networks is NP-complete, shown in Judd 1987, 1990).

0.2 Computation Graph

In this subsection, we give definitions of computation graphs in preparation for pattern recognition graph. We also show functions described by computation graph are more general than normal pure functions in programming langauges.

First, we use graph to represent a complicated function composed from simple ones. We use graph instead of DSL for the benefit of clarity.

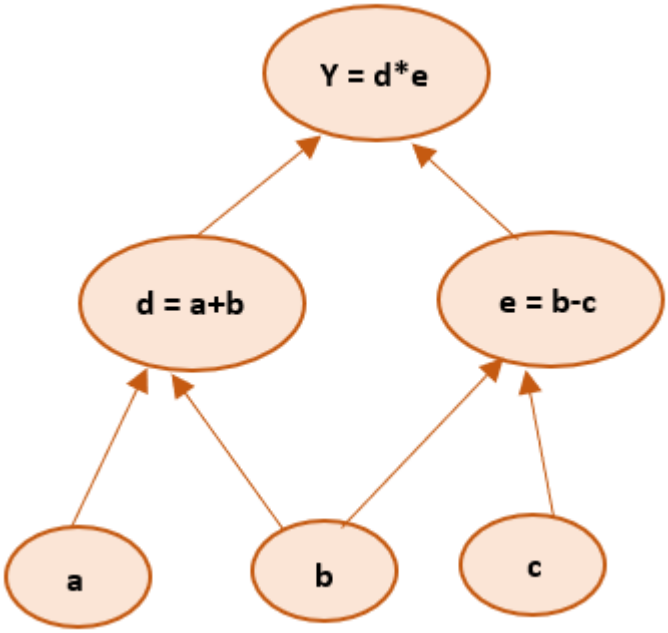
This is not much different from those graphs for neural networks.

Definition (Computation Graph)**.** A computation graph is a directed graph with nodes being types, and for a node T, with a nonempty list of incoming edges, the starting node type of which are $S_1, \cdots, S_n$, there is a function

$$f : (S_1, \cdots, S_n) \to T$$

.

All the nodes without incoming edges are seen as inputs, and all the nodes without outgoing edges are seen as outputs.



Claim: *functions described by computation graph are more general than normal pure functions in programming langauges.*

Here normal means the body of the function definitions is upper bounded, say by 30 lines (3 lines would be ideal for clean code).

Remark. *In Rust, there is a library called "salsa" where one can express the computation graph efficiently with memoization and value sharing. However, one needs to use macro and still need to keep track of many things.*

0.3 Pattern Recognition Graph

0.4 Modular Pattern Recognition Graph

Pattern recognition graph is a specialized computational graph such that it's computable, learnable, explainable and well-suited to a specific domain.

(Explainability is possible because we only allow nodes to be associated with explainably types, i.e. a type such that the value of that type has clear meaning)

To avoid unnecessary confusion from over abstraction, we discuss computer vision and natural language separately, but the similarity can still be easy to see.

Definition (Pattern Recognition Graph for Image Input)**.** The input is 2D rasterized image, i.e. a $M \times N$ matrix with each entry being RGB value.

0.5 Evolving Modular Pattern Recognition Graph

0.6 Expressive Power

0.6.1 Relational Database Query

0.6.2 Type Checking

0.6.3 Formal Verification and Theorem Proving

0.7 Transpilation to Neural Networks

0.8 Transpilation to More Efficient and Robustness Programs