Pattern Recognition in Procedural Knowledge

Comprehensive Exam

Breandan Considine

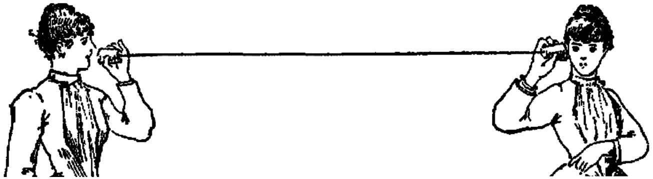
McGill University

breandan.considine@mcgill.ca

January 24, 2021

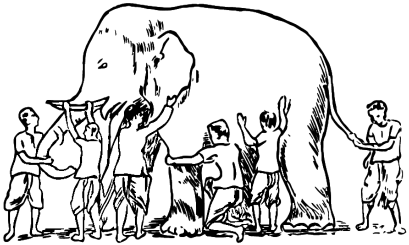
Knowledge is lost in translation

- First an end user describes a problem using natural language
- Next an expert translates the problem into a formal language
- Then an engineer implements the spec in a programming language
- Code is *not* the model, just one implementation of the model
- And the model itself is just an approximation of the real world
- System identification is hard, most abstractions are leaky



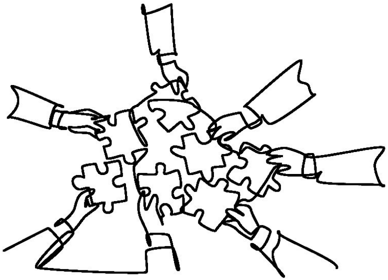
Knowledge is difficult to grasp

- The human cortex has a finite capacity working memory
- Many codebases are enormous bodies of collective knowledge
- Cannot be grasped entirely by a single knowledge worker
- Programs are not made of bits or bytes, but abstract procedures
- Code is simply one *artifact* or byproduct of programming
- Seek to understand the process, not the byproduct



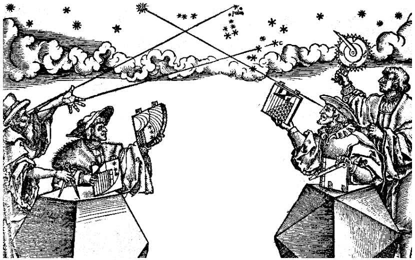
Knowledge does not compose

- Complex systems require participation from many stakeholders
- In order to scale up participation, we need programs to compose
- Impossible to build complex systems by gluing together spare parts
- Need better ways to facilitate collaborative knowledge engineering



Reason is a ladder to higher knowledge

- To facilitate human understanding, we must have better tools
- Tools should complement, not supplant human reasoning abilities
- Reasoning is a tool to access higher forms of knowledge
- Seek to augment human intelligence by automated reasoning



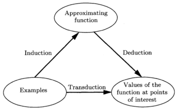
Learning and reasoning from first principles

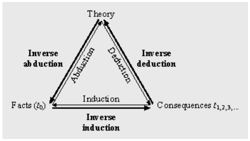
- Learning to program by imitating big code is a hopeless task!
- Such a system will only generate more code, increase technical debt
- Source code alone is fragile, contains too many brittle assumptions
- Where does code come from? Need to retrace its *provenance*
- Software more than just source code, many supporting *artifacts*
- Software engineering starts with pen and paper...

$$\frac{}{a \equiv a}\ \textit{Identity} \qquad \frac{a \equiv b}{b \equiv a}\ \textit{Symmetry} \qquad \frac{a \equiv b \quad b \equiv c}{a \equiv c}\ \textit{Transitivity} \qquad \frac{a \equiv b}{f(a) \equiv f(b)}\ \textit{Congruence}$$

Induction, Deduction, Abduction, Transduction, et al.

- ***duction**: From Latin ductio, ductionem, meaning to lead away
- **Conduction**: Transfer energy from one location to another
- **Induction**: Infer general rules from specific observations
- **Deduction**: Infer specific predictions from general rules
- **Abduction**: Infer higher order rules (theory) from specific facts
- **Transduction**: Infer specific predictions from specific observations
- **Production**: Infer a more complex expression from a simpler one
- **Reduction**: Infer a simpler expression from a more complex one





Logical Induction

- Inductive data types: `type Nat = Zero | Succ Nat`
- Inductive/recursive definitions: basis, induction, closure
- Sum-Product networks:

```
type SPN = SimpleDistribution
         | + (SPN a) (SPN a)
         | * (SPN a) (SPN a)
```

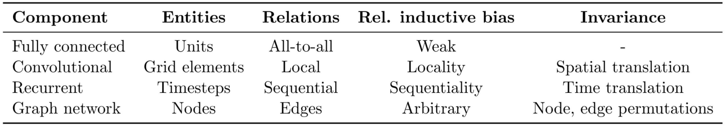
- Inductively defined graphs:

```
type Graph = Empty | Vertex
           | + (Graph a) (Graph a)
           | * (Graph a) (Graph a)
```

- Coinduction / corecursion

Induction in Machine Learning

- How should learners fill in the gaps between observations?
- Inductive biases allow a learning algorithm to prioritize one solution over another, irrespective of the observed data, e.g.:
  - Risk minimization (Vapnik, Chervonenkis, Arjovsky, et al.)
  - Minimum description (Solomonoff, Kolmogorov, Chaitin)
  - Maximum entropy (Shannon, Jaynes et al.)
  - Nearest neighbors (Fix, Hodges, Cover et al.)



Component	**Entities**	**Relations**	**Rel. inductive bias**	**Invariance**
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations

Table 1: Various relational inductive biases in standard deep learning components. See also Section 2.

Figure: From Battaglia et al. (2018), "Relational inductive biases"

What do these two formalisms share in common?

- Circuits propagate electricity through a copper graph
- Proofs propagate propositions through a logical graph
- Neural networks propagate error through a computation graph
- Bayes nets propagate uncertainty through a probabilistic graph
- Knowledge graphs propagate ideas through a hypergraph

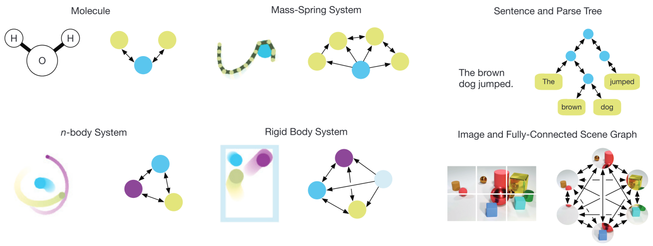


Figure: From Battaglia et al. (2018), "Relational inductive biases"

The Computational Trinity

The Christian doctrine of trinitarianism states that there is one God that is manifest in three persons, the Father, the Son, and the Holy Spirit, who together form the Holy Trinity. The doctrine of computational trinitarianism holds that [...] Logic, Languages, and Categories are but three manifestations of one divine notion of computation.

–Robert Harper (2011), The Holy Trinity

- Types ⇔ sets
- Proofs ⇔ programs
- Categories ⇔ graphs

Semiring algebras

$$\mathbf{D}_{st} = \overbrace{\bigoplus_{P \in P^*_{st}} \underbrace{\bigotimes_{e \in P} W_e}_{\text{Aggregate}}}^{\text{Accumulate}}$$

S	$\oplus$	$\otimes$	⓪	①	Path
$\mathbb{R} \cup \{\infty\}$	min	$+$	∞	0	Shortest
$\mathbb{R} \cup \{\infty\}$	max	$+$	$-\infty$	0	Longest
$\mathbb{R} \cup \{\infty\}$	max	min	0	∞	Widest

Static versus dynamic graphs

- Sometimes the graph is static (matrix multiplication)
- Just propagate simple values through it (e.g. $\mathcal{T} \in \mathbb{R}, \mathbb{B}, \mathbb{Z}, \mathbb{C}$)
- Sometimes the graph is dynamic (tensor contraction)
- Can propagate more complex objects (e.g. $\mathcal{T}^2, \mathcal{T}^3, \mathcal{T}^k$)
- Hyper-graph/edge replacement grammars (HRGs, FGGs, GGs et al.)

The algebra of graph replacement

- Semiring algebras (Gondran and Minoux)
- Graph grammars (Ariola, Plump et al.)
- Basically generalizes linear algebra to other structures

Valid and invalid procedures

- Not all procedural text is source code
- Not all source code is syntactically valid / well-formed
- Not all well-formed programs are semantically valid
- Not all semantically valid programs are error-free
- Not all error-free programs terminate
- Not all terminating programs yield the correct answer
- Not only "not all", but uncountably many counterexamples
- Unconstrained sampling is doomed to fail

Necessary assumptions

The expressive power of a programming language arises from its strictures and not from its affordances.

Robert Harper

- Bounded depth circuits
- Finite sample space
- All programs terminate

The trouble with pattern recognition: Part I

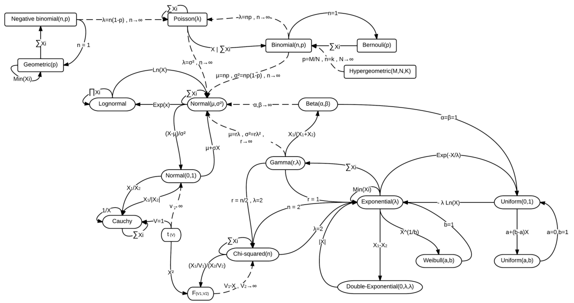
- Similar functions, dissimilar forms
- P1: Two functions may share the same meaning, but different forms
- E1: two programs may implement the exact same function, but share almost no syntax in common
- Q1: How do we recognize superficially dissimilar structures which share hidden meaning?

The trouble with pattern recognition: Part II

- Similar forms, dissimilar functions
- P2: Two objects may have a common form, but have vastly different meanings
- E2: two programs may differ by only a few tokens, but have drastically different meanings
- Q2: How do we disambiguate superficially similar structures?

Relations among probability distributions

- Knowledge is accumulated across generations of research
- Patterns have specific names, e.g. “Gaussian”, “Dirichlet”...
- Can we recover these relations from first principles?



Equivalence checking

	Deterministic	**Probabilistic**
Exact	Type Checking	Variable Elimination
	Model Checking	Probabilistic Circuits
Approximate	Software Testing	Monte Carlo Methods
	Dynamic Analysis	Bayesian Networks

Kernel methods

	$\Delta(f,g)$	$\varphi(x)$
Polynomial	$(\mathbf{f}^\intercal \mathbf{g} + r)^q$	$\left[\sqrt{\binom{q}{\mathbf{n}} r^{n_0}} \prod_k x_k^{n_k}\right]^\intercal_{\mathbf{n}\in\{\mathbf{n} \mid \mathbf{1}^\intercal \mathbf{n}=q\}}$
Gaussian RBF	$e^{-\frac{\lVert\mathbf{f}-\mathbf{g}\rVert^2}{2\sigma^2}}$	$e^{-\gamma x^2}\left[1, \sqrt{\frac{(2\gamma)^i}{i!}} x^i\right]^\intercal_{i\in(0,\dim(x)-1]}$
Subset	$\prod_{i=1}^n (f_i g_i + 1)$	$[\varphi_{\textsc{Poly}}(x)_A]^\intercal_{A\subseteq[1,n]}$
Substring	$\sum_{\sigma\in\Sigma^*}(f*\sigma)(g*\sigma)$	$\lvert\{i \mid \sigma = x_{i..(i+\lvert\sigma\rvert)}\}\rvert$
Subtree	$\Delta_{\mathrm{WL}}(f,g)$	$\begin{cases} \delta_k(t,x) & t \overset{?}{=} x \\ \varphi_t(\overleftarrow{x}) + \varphi_t(\overrightarrow{x}) & \text{otherwise.} \end{cases}$
Subgraph	$\Delta_{\mathrm{SS}}(\varphi^{(h)}(f), \varphi^{(h)}(g))$	$\textsc{Hash}(\{\{\varphi^{(i-1)}(u)\forall u \in \mathcal{N}(x)\}\})$