TTIC 31230, Fundamentals of Deep Learning

David McAllester, April 2017

In Search of AGI

Universal Models of Computation

Universal Knowledge Representation

Is There a Universal Architecture?

Noam Chomsky: By the no free lunch theorem **natural language grammar is unlearnable without an innate linguistic capacity**. In any domain a strong prior (a learning bias) is required.




Leonid Levin, Andrey Kolmogorov, Geoff Hinton and Jürgen Schmidhuber: **Universal learning algorithms exist. No domain-specific innate knowledge is required.**

Is Domain-Specific Insight Valuable?

Fred Jelinek: Every time I fire a linguist our recognition rate improves.

C++ as a Universal Architecture

Let h be any C++ procedure that can be run on a problem instance to get a loss where the loss is scaled to be in $[0, 1]$.

Let $|h|$ be the number of bits in the Zip compression of h.

Free Lunch Theorem: With probability at least $1-\delta$ over the draw of the sample the following holds *simultaneously* for all C++ programs h and all $\lambda > 1/2$.

$$\ell(h) \leq \frac{1}{1-\frac{1}{2\lambda}} \left(\widehat{\ell}(h) + \frac{\lambda}{N} \left((\ln 2)|h| + \ln \frac{1}{\delta} \right) \right)$$

The Turing Tarpit

The choice of programming language does not matter.

For any two Turing universal languages L_1 and L_2 there exists a compiler $C : L_1 \to L_2$ such that

$$|C(h)| \leq |h| + |C|$$

(Differentiable) Universal Computation

The Circuit Model

The Von Neumann Architecture ("Neural Turing Machines")

Functional Programming

Logic Programming

Universal Knowledge Representation

Knowledge Graphs (Freebase, Wikidata)

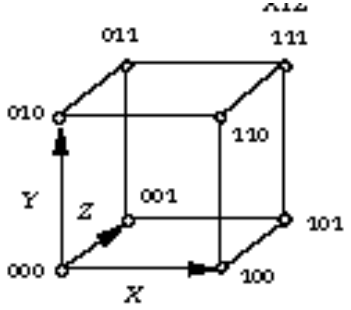
Natural Language as Knowledge Representation

The Foundations of Mathematics

The Concept of Truth.

Universal Function Representation Theorems

Consider continuous functions $f : [0,1]^N \to \mathbb{R}$



$\xrightarrow{f}$ $\mathbb{R}$

Given the corner values, the interior can be filled.

$$f(x_1,\ldots,x_N) = \prod_{y_1,\ldots,y_n,\, y_i\in\{0,1\}} f(y_1,\ldots,y_n) \prod_{i=1}^{N} (x_i y_i + (1-x_i)(1-y_i))$$

Hence each of the 2^N corners has an independent value.

The Kolmogorov-Arnold representation theorem (1956)

For continuous $f : [0,1]^N \to \mathbb{R}$ there exists continuous $g_i, h_{i,j} : \mathbb{R} \to \mathbb{R}$ such that

$$f(x_1, \ \ldots, \ x_N) = \sum_{i=1}^{2N+1} g_i \left(\sum_{j=1}^{N} h_{i,j}(x_j) \right)$$

$$f(x_1, x_2) = \begin{cases} & g_1(h_{1,1}(x_1) + h_{1,2}(x_2)) \\ + & g_2(h_{2,1}(x_1) + h_{2,2}(x_2)) \\ \vdots & \\ + & g_5(h_{5,1}(x_1) + h_{5,2}(x_2)) \end{cases}$$

A Simpler, Similar Theorem

For (possibly discontinuous) $f : [0,1]^N \to \mathbb{R}$ there exists (possibly discontinuous) $g, h_i : \mathbb{R} \to \mathbb{R}$.

$$f(x_1,\ \ldots,\ x_N) = g\left(\sum_i h_i(x_i)\right)$$

Proof: Select h_i to spread out the digits of its argument so that $\sum_i h_i(x_i)$ contains all the digits of all the x_i.

Cybenko's Universal Approximation Theorem (1989)

For continuous $f : [0,1]^N \to \mathbb{R}$ and $\varepsilon > 0$ there exists

$$F(x) = \alpha^\top \sigma(Wx + \beta)$$

$$= \sum_i \alpha_i \sigma \left(\sum_j W_{i,j}\, x_j + \beta_i \right)$$

such that for all x in $[0,1]^N$ we have $|F(x) - f(x)| < \varepsilon$.

How Many Hidden Units?

Consider Boolean functions $f : \{0,1\}^N \rightarrow \{0,1\}$.

For Boolean functions we can simply list the inputs $x^0, \ldots, x^k$ where the function is true.

$$f(x) = \sum_k \mathbb{1}[x = x^k]$$

$$\mathbb{1}[x = x^k] \approx \sigma\left(\sum_i W_{k,i} x_i + b_k\right)$$

A simpler statement is that any Boolean function can be put in disjunctive normal form.

Circuit Complexity theory

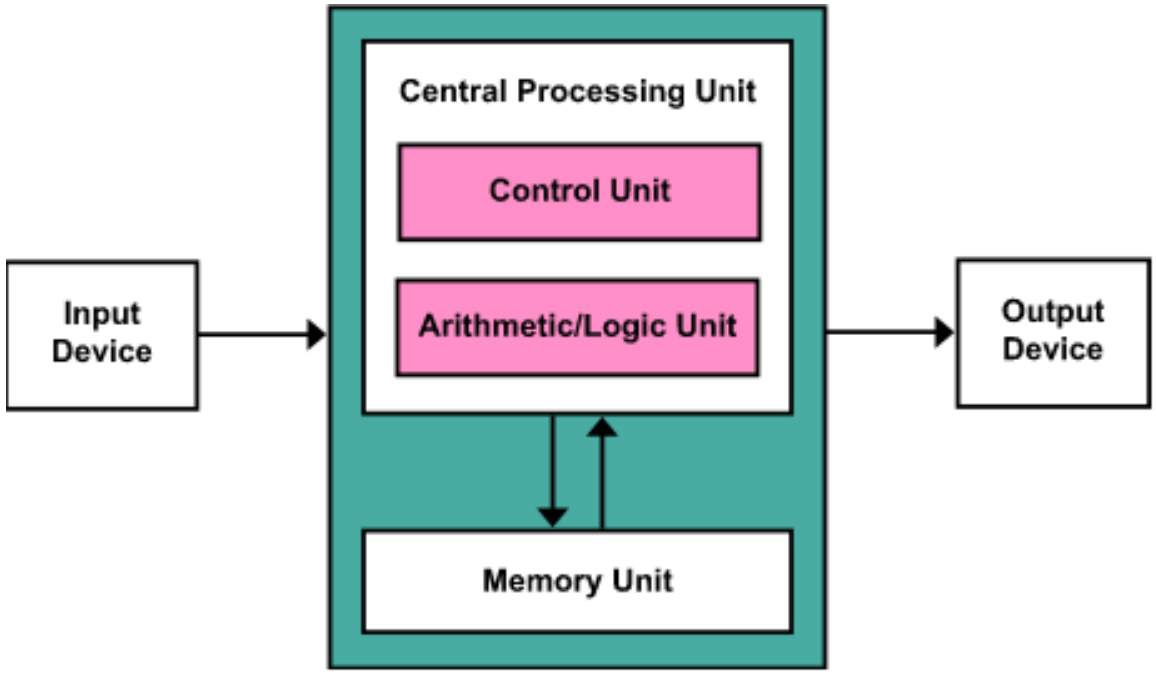
Building on work of Ajtai, Sipser and others, Hastad proved (1987) that any bounded-depth Boolean circuit computing the parity function must have exponential size.

Matus Telgarsky recently gave some formal conditions under which shallow networks provably require exponentially more parameters than deeper networks (COLT 2016).

Neural Turing Machines (NTM)

Neural Turing Machines Alex Graves, Greg Wayne, Ivo Danihelka, 2014

(Actually a differentiable Von Neumann architecture)



CPU Registers of the NTM

The CPU has a system of registers each of which stores a vector rather than a bit string.

All computation is differentiable.

Address registers (heads) hold an attention vector over all memory cells.

Read address registers are separate from write address registers.

An Execution Cycle

The CPU receives an external input.

The first step is to recompute the attention vectors in the read address registers.

The CPU computes a "key" k^h for each head h and an attention α_j^h

$$\alpha_j^h = \underset{j}{\text{softmax}}\ k^h \cdot M[j]$$

$$r^h = \sum_j \alpha_j^h M[j]$$

Reading from Memory

Once the read address vectors (attentions) have been computed we read a value for each read head.

$$v^h = \sum_j \alpha_j^h M[j]$$

After reading from memory the machine computes a key and attention for each write head.

Each write operation involves "forget" and "input" operation analogous to an LSTM.

Finally the controller emits an external output vector.

Neural Programmer Interpreter

Neural Programmers-Interpreter, Scott Reed and Nando de Freitas, ICLR, 2016

Neural Programmer Interpreter

We will use

$$
\begin{aligned}
i &\sim \text{procedure pointer (integer)} \\
p &\sim \text{procedure instruction (vector)} \\
a &\sim \text{arguments (sequence of integers)} \\
e &\sim \text{memory (array of vectors)} \\
h &\sim \text{CPU state vector}
\end{aligned}
$$

To execute a procedure call $\text{RUN}(i, a, e)$

$h \leftarrow 0$
$p \leftarrow M^{\text{Prog}}[i]$
Until $f_{\text{end}}(h)$: $\quad e, h \leftarrow \text{DoStep}(p, a, e, h)$
return(e)

DoStep

To compute DoStep(p, a, e, h)

$h \leftarrow f_{\text{LSTM}}(p, a, e, h)$

$k \leftarrow f_{\text{prog}}(h)$

$i \leftarrow \text{argmax}_i \; k^\top M^{\text{Key}}[i]$

If $i = 0$: $\quad e \leftarrow f_{\text{Env}}(p, f_{\text{Arg}}(h), e)$

else: $e \leftarrow \text{RUN}(i, f_{\text{Arg}}(h), e)$

return(e, h)

Non-Differentiable Steps

To execute a procedure call $\mathrm{RUN}(i, a, e)$

$h \leftarrow 0$
$p \leftarrow M^{\mathrm{Prog}}[i]$

Until $f_{\mathrm{end}}(h)$: $\quad e, h \leftarrow \mathrm{DoStep}(p, a, e, h)$

$\mathrm{return}(e)$

Non-Differentiable Steps

To compute DoStep(p, a, e, h)

$h \leftarrow f_{\text{LSTM}}(p, a, e, h)$

$k \leftarrow f_{\text{prog}}(h)$

$i \leftarrow \text{argmax}_i \; k^\top M^{\text{Key}}[i]$

If $i = 0$: $\quad e \leftarrow f_{\text{Env}}(p, f_{\text{Arg}}(h), e)$

else: $e \leftarrow \text{RUN}(i, f_{\text{Arg}}(h), e)$

return(e, h)

Training on Execution Traces

To train they use execution traces

$$e_t, i_t, a_t \Rightarrow i_{t+1}, a_{t+1}, f_{\text{end}}$$

Tail Recursion

Making Neural Programming Architectures Generalize Via Recursion, Jonathon Cai, Richard Shin, Dawn Song, ICLR, 2017

Allowing tail recursion, and explicitly labeling tail recursions in traces, significantly improves learning.

Bottom-up Logic Programming

Bottom-up logic programming is distinguished by its relationship to dynamic programming algorithms.

$$\text{At}(x) \Rightarrow \text{Reachable}(x)$$

$$\text{Reachable}(x) \wedge \text{Edge}(x, y) \Rightarrow \text{Reachable}(y)$$

This defines a linear time algorithm for reachability.

The CKY algorithm for context-free parsing

Consider a grammar defined by productions of the form $S \rightarrow$ NP VP and $N \rightarrow$ Kelly.

The $O(n^3)$ CKY parsing algorithm is defined by the following two rules.

$$X \rightarrow a \quad \wedge \quad S[i] = a \quad \Rightarrow X[i, i+1]$$

$$X \rightarrow YZ \quad \wedge \quad Y[i, j] \quad \wedge \quad Z[j, k] \quad \Rightarrow X[i, k]$$

Datalog

A set of inference rules each of which has antecedents and conclusions that are just predicates applied to variables is called a **datalog** program.

It can be shown that datalog "captures the complexity class P" — they can express **all and only** polynomial time decidable relations (provided the entities are assigned a total order).

General bottom-up logic programs, including expressions (terms), are Turing complete.

$$N(x) \Rightarrow N(s(x))$$

Universal Knowledge Representation

Knowledge Graphs (Freebase, Wikidata)

Natural Language as Knowledge Representation

The Foundations of Mathematics

The Concept of Truth.

Natural Language Semantics

Thousands of civilians have fled advances by Syrian government forces in eastern Ghouta as ...

Stanford Parse Tree

```
(NP (NP (NNS Thousands))
    (PP (IN of) (NP (NNS civilians))))
(VP (VBP have)
    (VP (VBN fled)
        (NP (NNS advances))
        (PP (IN by) (NP (NP (JJ Syrian)
                            (NN government)
                            (NNS forces))
                        (PP (IN in) (NP (JJ eastern)
                                        (NNP Ghouta)))))))
```

Stanford Dependencies

```
root(ROOT-0, fled-5)
  aux(fled-5, have-4)
  nsubj(fled-5, Thousands-1)
    nmod(Thousands-1, civilians-3)
      case(civilians-3, of-2)
  dobj(fled-5, advances-6)
  nmod(fled-5, forces-10)
    case(forces-10, by-7)
    amod(forces-10, Syrian-8)
    compound(forces-10, government-9)
    nmod(forces-10, Ghouta-13)
      case(Ghouta-13, in-11)
      amod(Ghouta-13, eastern-12)
```

Just Parantheses

```
(Thousands of civilians)
(have fled)
(advances (by (Syrian government forces))
          (in eastern Ghouta))
```

Reference

Thousands of civilians have fled advances by Syrian government forces in eastern Ghouta as Damascus makes rapid gains against the last major rebel enclave near the capital.

Damascus ⇒ Assad

Rapid Gains ⇒ advances-6

the last major rebel enclave ... ⇒ Ghouta

the capital ⇒ Damascus

Reference vs. Composition

Functional Programming is compositional

$$x = f(y, z)$$

The meaning of x is computed by f from the meaning of y and z.

But in language we typically have that $f(y, z)$ is a mention and x is its referent.

(the last (major rebel enclave) (near (the capital)))

$$x = (\text{the last } Q\ P)$$

Natural Language: Parsing

A Fast and Accurate Dependency Parser using Neural Networks, Danqi Chen and Christopher Manning, 2014.

$$
\begin{array}{l}
\vdots \\
s_3 \\
s_2 \\
s_1 \\
*\ \ b_1\ b_2\ b_3\ \ldots
\end{array}
\quad \overset{\text{push}}{\Rightarrow} \quad
\begin{array}{l}
\vdots \\
s_2 \\
s_1 \\
b_1 \\
*\ \ b_2\ b_3\ b_4\ \ldots
\end{array}
$$

Arc Transitions

$$\begin{array}{l} \vdots \\ s_3 \\ s_2 \\ s_1 \\ *\quad b_1\ b_2\ b_3\ \ldots \end{array} \quad \underset{\Rightarrow}{\text{LeftArc}} \quad \begin{array}{l} \vdots \\ s_4 \\ s_3 \\ s_1 \\ *\quad b_1\ b_2\ b_3\ \ldots \end{array} \qquad \text{Emits } s_1 \xrightarrow{\ell} s_2$$

$$\begin{array}{l} \vdots \\ s_3 \\ s_2 \\ s_1 \\ *\quad b_1\ b_2\ b_3\ \ldots \end{array} \quad \underset{\Rightarrow}{\text{LeftArc}} \quad \begin{array}{l} \vdots \\ s_4 \\ s_3 \\ s_2 \\ *\quad b_1\ b_2\ b_3\ \ldots \end{array} \qquad \text{Emits } s_2 \xrightarrow{\ell} s_1$$

Dependency Parsing Machine Configurations

A machine configuration

$$c = (s, b, A)$$

$s \sim$ stack

$b \sim$ buffer

$A \sim$ Dependency Arcs

Training

Construct a database of machine configurations labeled with actions.

Train an MLP with one hidden layer to a softmax over actions and train on cross entropy.

The input to the MLP is a concatenation of 18 word vectors defined in terms of the configuration

plus 18 corresponding part of speech vectors

and 12 parent edge label vectors.

The Foundations of Mathematics

variables, pairs	x	(e_1, e_2)	$\pi_i(e)$
functions	$\lambda x : \sigma\ e[x]$	$f(e)$	
Booleans	$P(e)$	$e_1 \doteq e_2$	$e_1 =_\sigma e_2$
	$\neg\Phi$	$\Phi_1 \vee \Phi_2$	$\forall x : \sigma\ \Phi[x]$
types	$\Sigma_{x:\sigma}\ \tau[x]$	$\Pi_{x:\sigma}\ \tau[x]$	$S_{x:\sigma}\ \Phi[x]$

The Base Case

The base case is given by **Bool**, **Set** and **Type** with **Bool** : **Type** and **Set** : **Type**.

$$\sigma \times \tau \doteq \Sigma_{\alpha:\sigma}\, \tau \quad \alpha \text{ not in } \tau$$

$$\mathbf{Graph} \doteq \Sigma_{\alpha:\mathbf{Set}}\, (\alpha \times \alpha) \to \mathbf{Bool}$$

Learning Guiding Evolution

In a 1987 paper entitled "How Learning Can Guide Evolution", Goeffrey Hinton and Steve Nowlan brought attention to a paper by Baldwin (1896).

The basic idea is that learning facilitates modularity.

For example, longer arms are easier to evolve if arm control is learned — arm control is then independent of arm length. Arm control and arm structure become more modular.

Learning Guiding Learning

If each "module" is learning to participate in the "society of mind" then each model can more easily accommodate (exploit) changes (improvements) in other modules.

Modularity (and abstraction) are fundamental principle of software design.

Levin's Universal Problem Solver (Levin Search)

Leonid Levin observed that one can construct a universal solver that takes as input an oracle for testing proposed solutions and, if a solution exists, returns it.

One can of course enumerate all candidate solutions.

However, Levin's solver is universal in the sense that it is not more than a constant factor slower than any other solver.

Levin's Universal Solver

We time share all programs giving time slice $2^{-|h|}$ to program h where $|h|$ is the length in bits of h.

The run time of the universal solver is at most

$$O(2^{-|h|}(h(n) + T(n)))$$

where $h(n)$ is the time required to run program h on a problem of size n and $T(n)$ is the time required to check the solution.

Here $2^{-|h|}$ is independent of n and is technically a constant.

Bootstrapping Levin Search

While Levin search sounds like a joke, Schmidhuber takes it seriously.

He has proposed ways of accelerating a search over all programs and has something called the Optimal Ordered Problem Solver (OOPS).

The basic idea is bootstrapping — we automate a search for methods of efficient search.

END