Derivatives of Turing Machines and Inductive Inference

Daniel Murfet

University of Melbourne

⌜based on joint work with J. Clift "Derivatives of Turing machines in Linear Logic" arXiv: 1805.11813⌟

therisingsea.org

Talk Outline

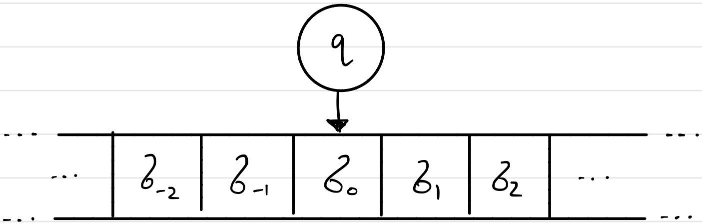
1. Derivatives of Algorithms?
   - Turing machines, Universal Turing machines
   - The Ehrhard-Regnier derivative
2. The problem of inductive inference / program synthesis
   - General problem statement
   - The probabilistic approach
3. Program synthesis via naive probability

What is an algorithm?

Def^n A Turing Machine M on alphabet Σ and set of states Q (both finite sets, with special symbols blank $\square \in \Sigma$ and $0, 1 \in \Sigma$ and states START, HALT $\in Q$) is a function (so $M = (\Sigma, Q, \delta)$)

$$\delta : \Sigma \times Q \longrightarrow \Sigma \times Q \times \{\text{LEFT}, \text{RIGHT}, \text{STAY}\}$$

which describes the time evolution of a configuration



$\in \Sigma^{\mathbb{Z}} \times Q$

(all but finitely many σ_i are blank)

$((\sigma_i)_{i \in \mathbb{Z}}, q)$

What is an algorithm?

<u>Defⁿ</u> Given a Turing machine M we write $M(x) = y$ for $x, y \in \{0,1\}^*$ if on a configuration (x, START) the machine reaches the configuration (y, HALT) after finitely many steps.

<u>Theorem</u> (Turing) There exists a Turing machine $\mathcal{U}$ such that for every Turing machine M there exists a string $c_{\mathcal{U}}(M) \in \{0,1\}^*$ such that for all $x \in \{0,1\}^*$

- if $M(x) = y$ then $\mathcal{U}(c_{\mathcal{U}}(M), x) = y$, and
- if M does not halt on input x, then $\mathcal{U}$ does not halt on $(c_{\mathcal{U}}(M), x)$

• Such a machine is called a <u>Universal Turing Machine</u> (UTM).

What is an algorithm?

Example The Turing Machine M has $\Sigma = \{\square, 0, 1\}$, $Q = \{\text{START}, \text{HALT}\}$

$$\delta(x, \text{HALT}) = (x, \text{HALT}, \text{STAY})$$

$$\delta(0, \text{START}) = (1, \text{START}, \text{RIGHT})$$

$$\delta(1, \text{START}) = (0, \text{START}, \text{RIGHT})$$

$$\vdots$$

$$M(0110) = 1001$$
$$M(1010) = 0101$$
$$\vdots$$

The code of this Turing machine is the string $c(M)$:

"0 START 1 START RIGHT ▨ 0 HALT 0 HALT STAY ▨ 1 START ⋯"

and by choosing binary encodings we may assume $c(M) \in \{0,1\}^*$.

The Ehrhard-Regnier derivative

$$\begin{tikzcd}
\{\text{Turing machines}\} \arrow[r, shift right, "\text{Turing}"'] & \{\lambda\text{-calculus terms}\} \arrow[l, shift right] \arrow[d]
\end{tikzcd}$$

$(\lambda x.(\lambda y.x)),\ (\lambda y.y), \ldots$

$$\begin{tikzcd}
M \arrow[r, mapsto] & t(M) \arrow[dl] \\
D_i\, t(M)\cdot u &
\end{tikzcd}$$

$D_i \circlearrowright \{\text{Differential } \lambda\text{-calculus terms}\}$

$D_1\,(\lambda x.(\lambda y.x))\cdot u, \ldots$

<u>Question</u>: *what do these derivatives mean? And what are they good for?*

- T. Ehrhard, L. Regnier "*The differential λ-calculus*" *Theoretical Computer Science* 309, p. 1-41, 2003.

A semantic approach via Linear Logic

language of closed symmetric monoidal categories with cofree coalgebras

$$
\begin{tikzcd}
\{\text{Turing machines}\} \arrow[r, "\text{encode}"] & \{\text{linear logic proofs}\} \arrow[d, "\text{add dual numbers}"] \arrow[dr, bend left, "[\![-]\!]"] & \\
& \{\text{differential linear logic proofs}\} \arrow[r, "[\![-]\!]"] \arrow[loop below, "\frac{\partial}{\partial x}"] & \mathrm{Vec}_k
\end{tikzcd}
$$

based on work of Girard

Ehrhard-Regnier derivative

Sweedler semantics (based on cofree coalgebras)

Inductive Inference

- (Informal) Given several instances of a pattern, infer the pattern

- Given $(x_1, y_1), \ldots, (x_n, y_n)$ with $x_i, y_i \in \{0,1\}^*$, infer an algorithm M such that $M(x_i) = y_i$ for $1 \leq i \leq n$.

  ⌜inductive: we take $M(x)$ as a prediction for unobserved x⌟

  - What do we mean by "algorithm"?
  - There are infinitely many such M's $\Rightarrow$ prefer simpler ones
  - But what does "simpler" mean?
  - For technical reasons we may also bound M's runtime (i.e. we assume the pattern is <u>effectively</u> computable) and allow $M(x_i) = y_i *$.

Inductive Inference, formalised

- Every UTM $\mathcal{U}$ gives a "parametrisation" of the set of algorithms

$$\{\text{algorithms}\} := \{\text{Turing machines}\} \xrightarrow[\cong]{c_{\mathcal{U}}} \{0,1\}^*, \text{ i.e. codes}$$

- Kolmogorov, Solomonoff: algorithm M is <u>simpler</u> than algorithm N if

$$|c_{\mathcal{U}}(M)| < |c_{\mathcal{U}}(N)|$$

- <u>Inductive Inference</u>: given $\mathcal{E} = \{(x_i, y_i, t_i)\}_{i=1}^{N}$ with $x_i, y_i \in \{0,1\}^*$ and $t_i \in \mathbb{N}$, find the shortest code $C \in \{0,1\}^*$ such that $\mathcal{U}(C, x_i) \overset{t_i}{=} y_i *$ for all $1 \leq i \leq N$. Call any such C a <u>solution</u> of $\mathcal{E}$.

meaning a string with prefix y_i

Error propagation

- Inductive Inference: given $\mathcal{E} = \{(x_i, y_i, t_i)\}_{i=1}^N$ with $x_i, y_i \in \{0,1\}^*$ and $t_i \in \mathbb{N}$, find the shortest code $C \in \{0,1\}^*$ such that $\mathcal{U}(C, x_i) \overset{t_i}{=} y_i$ * for all $1 \leq i \leq N$. Call any such C a solution of $\mathcal{E}$.

- Logician's solution enumerate codes by length and try them one-by-one!

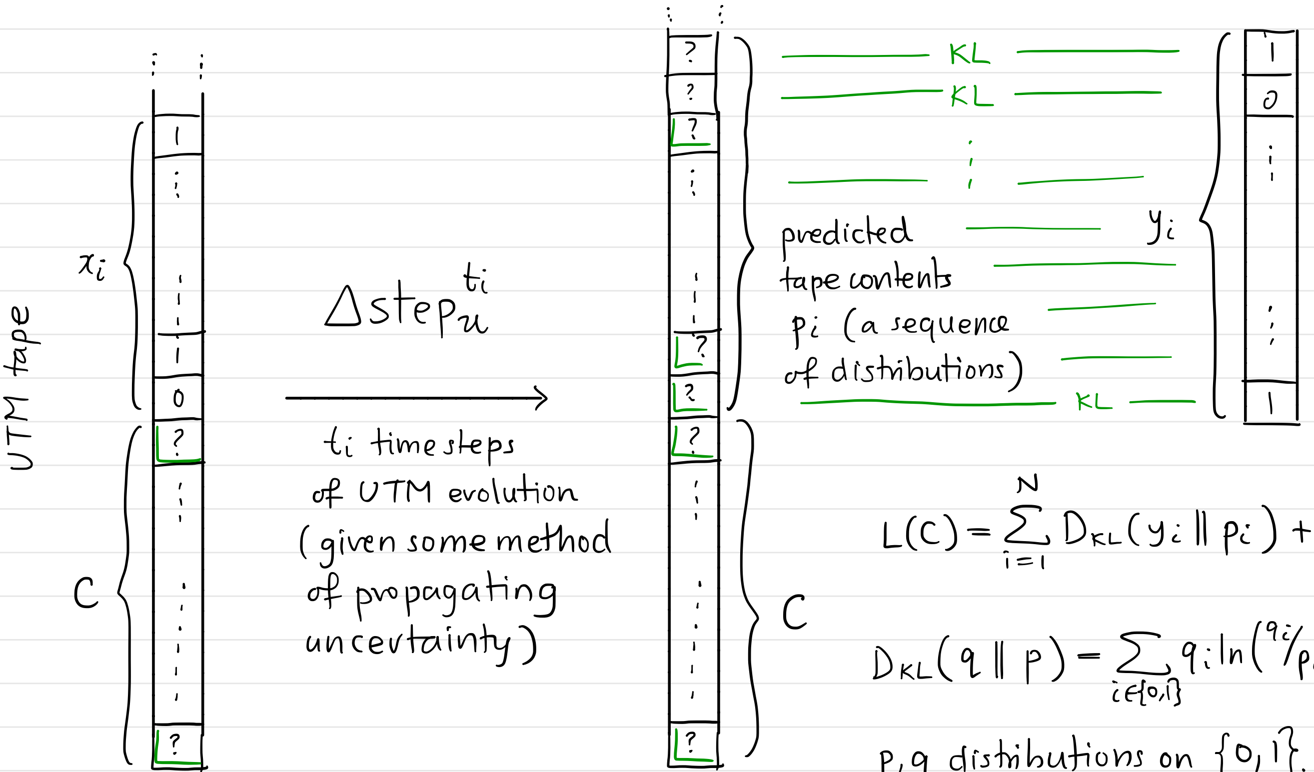
- Probabilist's solution view $C = (C_0, C_1, \ldots)$ as a sequence of distributions over $\{0,1\}$ and vary C so as to minimise a loss

$$L(C) = \sum_{i=1}^{N} D_{KL}\left(y_i \,\|\, \Delta\mathrm{step}_{\mathcal{U}}^{t_i}(C, x_i) \right) + \text{regularisation}$$

KL divergence

propagation of uncertainty in C through UTM

Error propagation



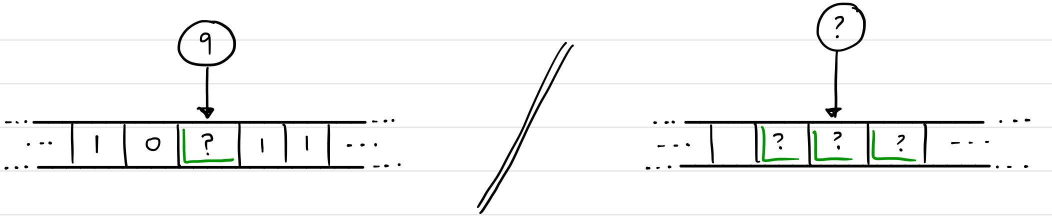
$$L(C) = \sum_{i=1}^{N} D_{KL}(y_i \,\|\, p_i) + \cdots$$

$$D_{KL}(q \,\|\, p) = \sum_{i \in \{0,1\}} q_i \ln\left(q_i / p_i\right)$$

p, q distributions on $\{0,1\}$.

Error propagation

- **Question**: how to propagate error/uncertainty through a TM?



t steps

Def^n Given a finite set Z, let $\Delta Z :=$ { probability distributions over Z }.

$$\underbrace{(\Delta\Sigma)^{\mathbb{Z}} \times \Delta Q}_{\text{configurations with uncertainty}} \xrightarrow{\Delta\mathrm{step}_M} (\Delta\Sigma)^{\mathbb{Z}} \times \Delta Q$$

i.e. what should this be?

- Standard probability: sample from distributions representing state and tape squares, run the machine for t steps, amalgamate results

$$\mathbb{P}(\, S(t+1) = q \,) = \sum_{\substack{z \in \Sigma,\, q \in Q \\ \delta(z, q') = (?, q, ?)}} \mathbb{P}(\, Y_0(t) = z \wedge S(t) = q' \,)$$

state at time $t+1$ symbol under head state.

Suppose the code $C = (C_0, C_1, C_2, \dots)$ depends on $C_j = x_j \cdot 0 + (1 - x_j) \cdot 1$, so

$$\frac{\partial}{\partial x_j} L(C) = \sum_{i=1}^{N} \frac{\partial}{\partial x_j} D_{KL}\Big(y_i \,\|\, \Delta\mathrm{step}_u^{t_i}(C, x_i) \Big) + \cdots$$

$$= \text{a function of } \frac{\partial}{\partial x_j} \Delta\mathrm{step}_u^{t_i}(C, x_i)$$

Unfortunately ∇L is useless for solving the inductive inference problem!

- <u>Naive probability</u> we can define an alternative Δstep propagating uncertainty through a TM, by making certain "naive" conditional independence hypotheses, e.g.

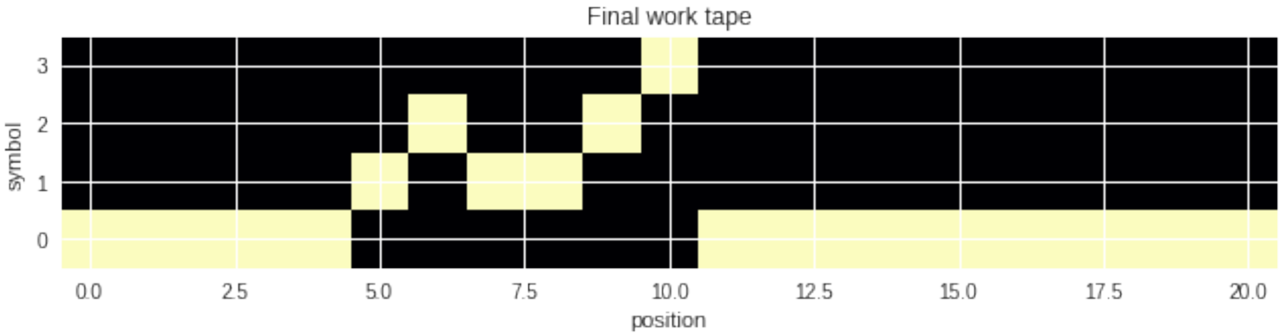
$$\mathbb{P}_{nv}(\ S(t+1) = q\ ) = \sum_{\substack{z \in \Sigma,\, q \in Q \\ \delta(z,q') = (?,q,?)}} \mathbb{P}_{nv}(\ Y_0(t) = z\ )\, \mathbb{P}_{nv}(\ S(t) = q'\ )$$

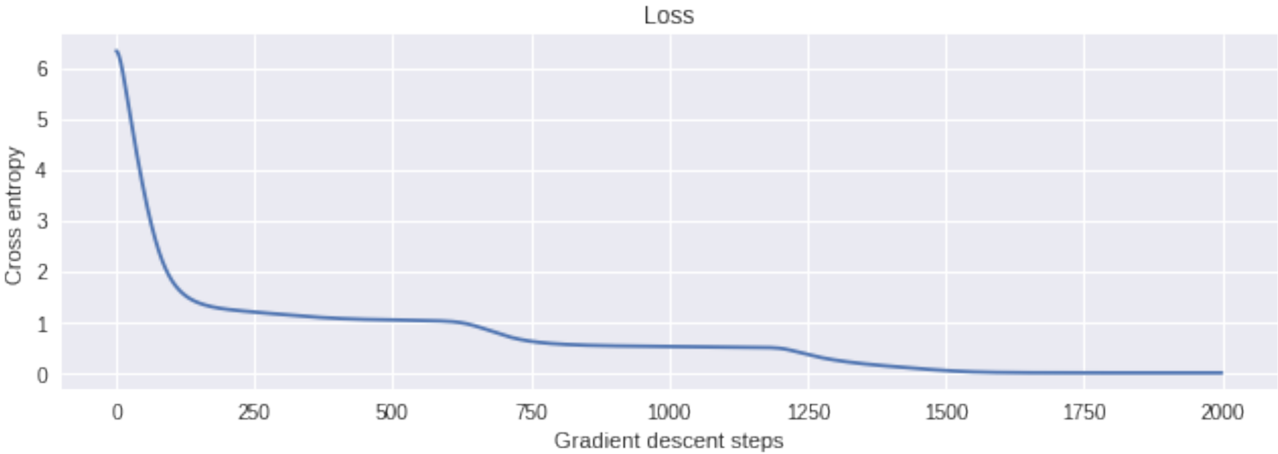
state at time t+1 symbol under head state.

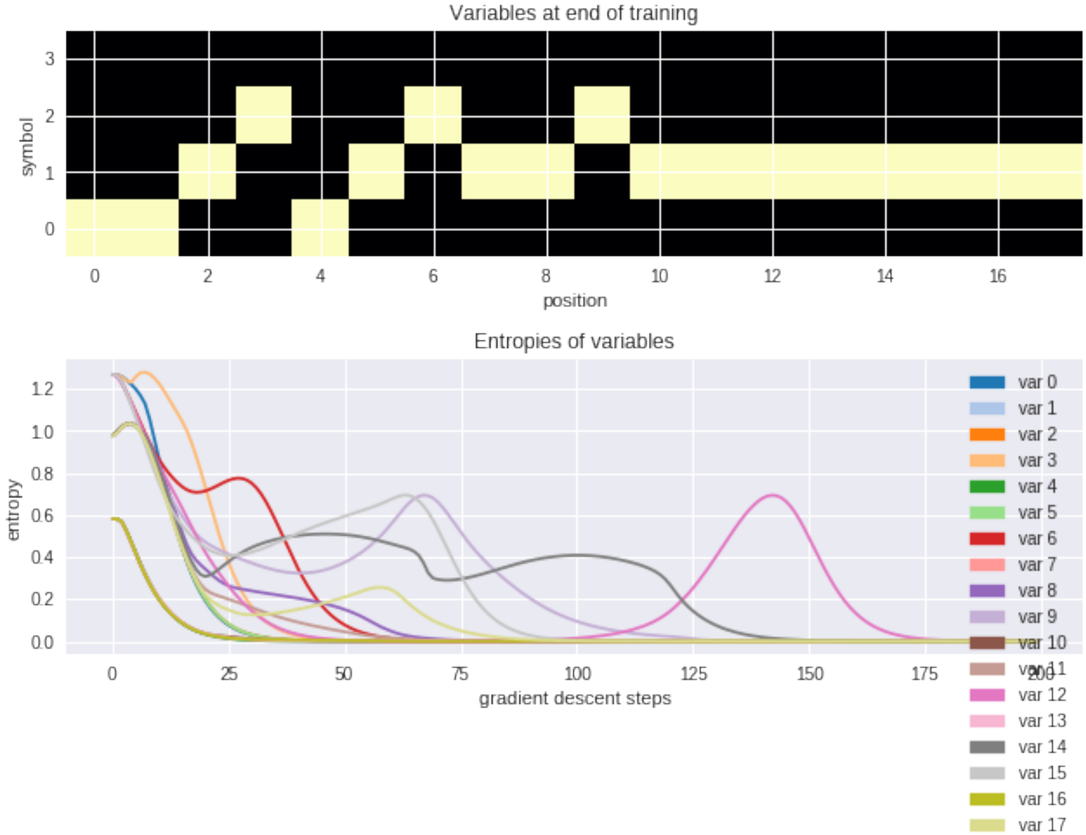
<u>Theorem</u> (Clift-M) The propagation of naive probability

$$\Delta\mathrm{step}_M : (\Delta\Sigma)^{\mathbb{Z}} \times \Delta Q \longrightarrow (\Delta\Sigma)^{\mathbb{Z}} \times \Delta Q$$

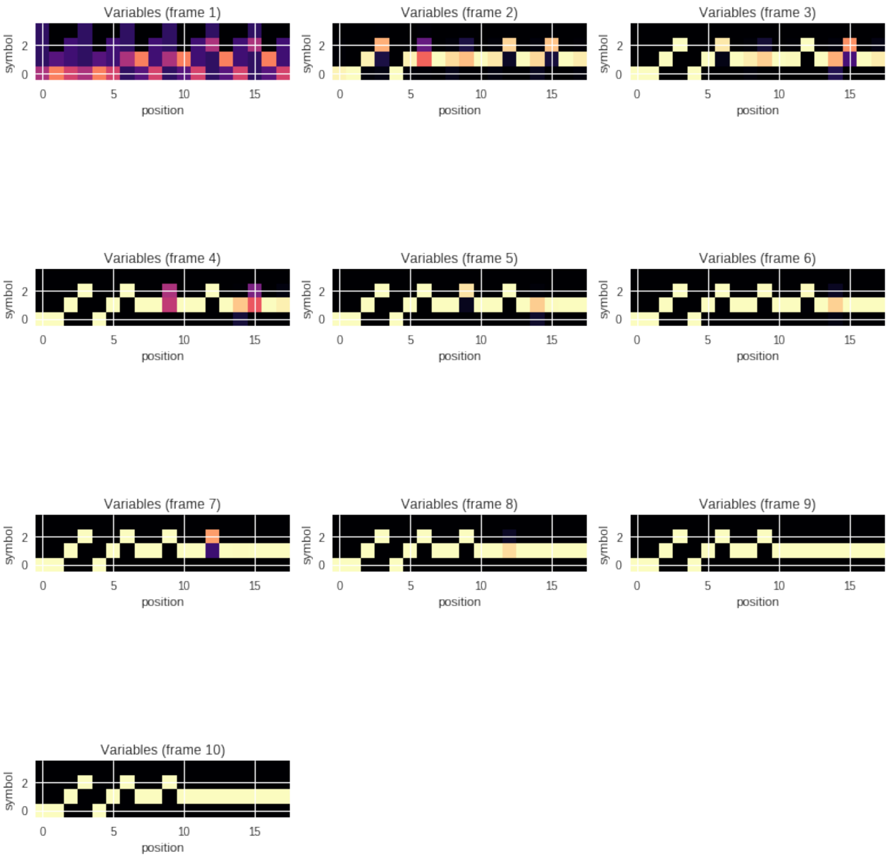
arises from the denotational semantics of linear logic, and the Ehrhard-Regnier derivative of M computes $T(\Delta\mathrm{step}_M)$.


Final work tape
symbol
position
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
1
2
3


Loss
Cross entropy
Gradient descent steps
0
250
500
750
1000
1250
1500
1750
2000
0
1
2
3
4
5
6



Variables at end of training
symbol
3
2
1
0
0
2
4
6
8
10
12
14
16
position
Entropies of variables
entropy
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
25
50
75
100
125
150
175
200
gradient descent steps
var 0
var 1
var 2
var 3
var 4
var 5
var 6
var 7
var 8
var 9
var 10
var 11
var 12
var 13
var 14
var 15
var 16
var 17


Variables (frame 1)
Variables (frame 2)
Variables (frame 3)
Variables (frame 4)
Variables (frame 5)
Variables (frame 6)
Variables (frame 7)
Variables (frame 8)
Variables (frame 9)
Variables (frame 10)
symbol
0
2
position
0
5
10
15

References

- (DeepMind) R. Evans, E. Grefenstette "Learning explanatory rules from noisy data" Journal of AI Research 61 (2018) 1-64.

- (Microsoft) A. Gaunt et. al "TerpreT: a probabilistic programming language for program induction" 2016.

The Ehrhard-Regnier derivative

$$
\begin{tikzcd}
\{\text{Turing machines}\} \arrow[r, shift right, "\text{Turing}"'] & \{\lambda\text{-calculus terms}\} \arrow[l, shift right] \arrow[d] \\
& \{\text{Differential } \lambda\text{-calculus terms}\} \arrow[loop left, "D_i"]
\end{tikzcd}
$$

$(\lambda x.(\lambda y.x)),\ (\lambda y.y), \ldots$

$D_1\,(\lambda x.(\lambda y.x))\cdot u, \ldots$

$$
\begin{tikzcd}
M \arrow[r, mapsto] & t(M) \arrow[dl] \\
D_i\,t(M)\cdot u &
\end{tikzcd}
$$

<u>Question</u> : what do these derivatives mean? And what are they good for?

they compute rates of change of naive probability

(potentially) program synthesis

∇L computed by Ehrhard-Regnier