2024-04-15-Free Range Programming

Towards a Vulcan Mind Meld

Questions:

We have two LLMs, one trained and one empty.

How do you transfer training from the first LLM to the empty one?

The information probably consists of zillions of numeric weights. A different weight for each node in the LLM.

Would you transfer the weights serially, in a single stream of information?

If you were to press 10 fingers against the head of the empty LLM, would you use 10 serial streams instead of a single stream? Would the transfer process run 10 times faster?

How close together would the source and target need to be to use the 10-finger trick? At what separation distance do you have to stop using 10 fingers and resort to a single stream of data?

What is Lisp?

In my mind:

Lisp is: programming using ASTs (CSTs, actually, but, I quibble) and has no actual "higher level syntax"

Lisp is an assembler with recursive syntax instead of a line-oriented syntax

- assembler is "untyped" and allows great power while allowing you to blow your own feet off

Lisp is, ostensibly, about "list processing", but, in fact it is about "stack processing" ; CAR means "top of stack" and CDR means "the rest of stack, with the Top removed"

Lisp has only 2 types - Atom and List, which makes Lisp very convenient for Design (vs. Production Engineering (aka optimization)), since you don't need to get tangled up dealing with niggly details.

Lisp can accommodate any paradigm and is not restricted to FP, or OO, or class-based design, or whatever ; again, this makes Lisp very convenient for Design (corollary: a language that has a parenthesis-oriented syntax is not actually a Lisp if it strongly encourages programming in only a single paradigm, like FP or OO)

Lisp began life as a "dynamically typed" language, which means that it did type-checking only at runtime (this is *not* the same as being "untyped"), it is possible to (further) differentiate Atoms by type and to perform gradual type embellishment and gradual type-checking

(If my reference to Lisp as being an untyped assembler vs. to Lisp being typed is confusing, note that I consider "type checking" to be just another error check akin to "syntax error checking", you can check for machine-types (like int/float/double/etc.) and/or you can check for design-oriented type hierarchies or you can just use 2 simple types (like Atom and List) - Lisp allows you to turn the knob from "not much type checking" to "lots of type checking", depending on where you are in the Design cycle)

Appendix - See Also

See Also

References *https://guitarvydas.github.io/2024/01/06/References.html*
Blog *https://guitarvydas.github.io/*
Blog *https://publish.obsidian.md/programmingsimplicity*
Videos *https://www.youtube.com/@programmingsimplicity2980*
[see playlist "programming simplicity"]
Discord *https://discord.gg/Jjx62ypR (Everyone welcome to join)*
X *(Twitter) @paul_tarvydas*
More writing *(WIP): https://leanpub.com/u/paul-tarvydas*