Lecture Note on the Husky Project

Xiyu Zhai

Contents

1 Introduction

The husky project (`https://github.com/ancient-software/husky`) is currently a project in computer vision, which starts five years ago even before my PhD.

(Pdf of lecture note: `https://github.com/ancient-software/husky/blob/main/sparks/impress/lecture1.pdf`)

(Latex Source of lecture note: `https://github.com/ancient-software/husky/blob/main/sparks/impress/lecture1.tex`)

Warning: this is work in progress. It shall take a long time to complete this note.

1.1 Motivation

Example (Linear Classification)**.** $f_0(x) = w_0^\top x$.

This is easy.

Just linear algebra.

Mathematically speaking, no interesting mathematical structure is involved.

Let Γ be a set, let X, W be input space and parameter space (can be $\mathbb{R}^n, \mathbb{R}^m$ respectively) and consider $F : X \times W \times \Gamma \to Y$ such that the ground truth f_0 is given by

$$f_0(x) = \sup_{\gamma \in \Gamma} F(x; w_0, \gamma) \tag{1}$$

where fixed unknown parameter $w_0 \in W$ and F is easy to compute.

(It's more proper to write in dependent type: $F : (x : X) \to (w : W) \to \Gamma(x, w)$ to mean that Γ is dependent on preceding x, w.)

If we only care about whether f_0 is above a certain threshold v_0, then

$$f_0(x) > v_0 \Leftrightarrow \exists \gamma \in \Gamma, F(x; w_0, \gamma) > v_0 \tag{2}$$

which can be seen as "NP"-problem.

We can make everything boolean by setting

$$f_0(x) := \exists \gamma \in \Gamma, F(x; w_0, \gamma) \tag{3}$$

We can make this harder by considering

$$f_0(x) := \forall \gamma_1 \in \Gamma, \exists \gamma_2 \in \Gamma, F(x; w_0, \gamma_1, \gamma_2) \tag{4}$$

and maybe even harder to include the problem of Go game.

There are two difficulties:

- statistical one. w_0 is unknown
- computational one. How to get an efficient approximation of f_0 because its original mathematical form is not obviously computable.

When $f(\gamma_0, x, w_0)$ is ϵ-continuous for any γ_0, w_0, so is f_0, then $f_0 \in$, so we can put f_0 in some continuous function class, then we can apply traditional machine learning or deep learning to it.

In fact, machine learning can still be applied even if the original problem doesn't really have any statistical element, i.e. W is univalent. This is basically what AlphaGo does. It's a way of approximate a function so that it's accurate with high probability.

The problem with this approach is that it's not optimal, overly complicated, leads to blackbox. For domain specific F, there might be better ways.

The project is now about finding these better ways for those F in computer vision.

Example (Mnist)**.**

Example (Imagenet)**.** Computer Graphics.

Let's see how mathematicians handle similar conditions.

Example (Group isomorphism)**.**

Definition (Group)**.** A group is a set G together with a multiplication $M : G \times G \to G$ such that

(1) identity

(2) inverse

(3) associative

Fix a finite group G_0, let $X = \mathcal{G}$ be a set of finite group, let W be univalent, and γ consider

$$F(x; w_0, \gamma) := \gamma : x \to G_0 \text{ is a group isomorphism} \tag{5}$$

todo: explain various ways to approximate F.

Example (Topological Space)**.**

Definition (Topological Space)**.** A topological space is a set X together with a set of subsets (called open sets) τ such that

- $\emptyset, X \in \tau$
- $\forall U_1, U_2 \in \tau$, $U_1 \cap U_2 \in \tau$
- $\forall \{U_i : i \in I\} \subset \tau$, $\bigcup_{i \in I} U_i \in \tau$

Definition (Cech Cohomology of Constant Real Valued Sheaf)**.** Let X be a topological space, for each open set U, let $\mathscr{F}(U)$ be the set of real valued constant functions on U.

Let $\mathcal{U}$ be a finite open cover of X ($\mathcal{U}$ contains a finite number of open sets and their union is X).

For $I \subset \{1, \cdots, n\}$ define $U_I = \cap_{i \in I} U_i$, then the Cech complex is defined by

$$0 \to \prod_{|I|=1} \mathscr{F}(U_I) \to \prod_{|I|=2} \mathscr{F}(U_I) \to \cdots \to \prod_{|I|=i} \mathscr{F}(U_I) \to \cdots \tag{6}$$

The maps are defined as follows. The map from $\mathscr{F}(U_I) \to \mathscr{F}(U_J)$ is 0 unless $I \subset J$, i.e., $J = I \cup \{j\}$. If j is the kth element of J, then the map is $(-1)^{k-1}$ times the restriction map res_{U_I, U_J}.

Define $H^i_{\mathcal{U}}(X, \mathscr{F})$ to be the ith cohomology group of the complex. (todo: explain this)

todo: explain example

$0 \to \mathbb{R}^2 \to \mathbb{R}^2 \to 0$

$H^0_{\mathcal{U}_k} = H^1_{\mathcal{U}_k} = \mathbb{R}$, which agrees with the singular cohomology.

1.2 Composition

It currently comprises mainly of three parts (three totally different fields):

(1) new algorithms and ways of feature constructions that are geometrically more appropriate for shape analysis;

  Remark. *todo: explain what is geometrically appropriate*

(2) a new machine learning framework

(3) a new programming language that is critical for interactively implementing and improving efficient and fully interpretable models.

  Remark. *PL is generalizable to other domains, like NLP, Theorem Proving, Robotics, RL, Computer Graphics, etc. I'm in the way of overhauling the type system to introduce Monad for NLP and Robotics and RL and Prop for theorem proving.*

  As a result, this PL is an extremely ambitious project that must have all good things from languages in different domains, C++/Rust/Zig from system level programming

Remark. *todo: explain why this is difficult for previous experts in the field.*

1.3 Difference

Pattern Theory `https://en.wikipedia.org/wiki/Pattern_theory`

Traditional Computer Vision. Handcrafted feature descriptors such as SIFT and SURF are fed to traditional machine learning classification algorithms such as Support Vector Machines and k-Nearest Neighbours to solve the aforementioned CV problems.

Computation: matrix (mostly)

Biggest problem: doesn't work well for large datasets.

Deep Learning. End to end training.

Computation: matrix

Biggest problem: doesn't explain well, totally not scientific.

Husky. Interactively construct features that is interpretable and efficient, first purely by hand then automate more and more in the process. Feature is no longer just a map from input to a vector space, it's a map from input to a type, which encodes information in an domain specific, interpretable and efficient way.

`https://en.wikipedia.org/wiki/Fock_space`

For image classification, the training process is replaced by the following iterative developing process (with labeling being part of it)

- Replace the goal of finding an all-powerful classifier by finding for each class a one-vs-all classifier.

  Remark. *When the number of classes is really large, we will use LSH(locally sensitive hashing)-like schemes. But for imagenet, this is not necessary.*

- For each class, try to represent the one-vs-all classifier by a cascade, i.e. a sequence of partial classifiers together with a one-vs-all classifier in the end.

  todo: explain partial classifier

- Partial classifiers are first constructed by hand interactively. Constantly interact with the debugger.

The core advantages are

- The training process doesn't require GPU and is inherent cheap to compute, everyone with a normal computer can do it.
- Agile. For any small change, the result is instantaneously updated and one can reason why it behaves that way. In contrast, in deep learning or tradition

its inference is much faster and less memory demanding because everything is interpretable so there are more chances for optimization and.

However, this process maps to a drastically different set of requirements on development tools which is too advanced for existing programming languages to handle:

- Require the ability to easily write system level programming because we are not using matrices

  Remark. *This rules out python.*

- Incrementally compute the features in a functional way, i.e. instant response even for imagenent

  Remark. *This rules out every existing language, especially those requiring long compilation time like C++/Rust/Zig. But python and julia are kindof okay because they can run in a notebook environment. The only problem is that notebook is procedural, not functional, can't scale every well.*

- One-click visualization of one or more features (possibly restricted to a branch or a datapoint)

  Remark. *This rules out every existing language.*

- Higher order functions for automation: one expr for constructing and training a model

  Remark. *This rules out every existing language.*

- Automatic memory management.

  Remark. *This rules out every existing language except Rust.*

- Automatic lazy value cache management across domains

  Remark. *This rules out every existing language. Haskell has features that sounds similar, but isn't good enough.*

- Rigorous type system to avoid distracting programmer from the task at hand

  Remark. *This rules out python.*

- Being friendly to customized compilation, with targets being possibly embedded device.

  Remark. *This rules out python, julia.*

So that's why I feel it necessary to develop a new language.

1.4 Future

The future is going to be like:

- in two years, imagenet is done in husky by myself, totally interpretable and as accurate and 100 times faster for inference; the development only requires a moderate computer and it doesn't need GPU for training and inference.

  Rougly speaking, it's going to be like:

  - 4 months recognise husky
  - 4 months recognise 9 more classes
  - 4 months recognise 90 more with automation
  - 4 months recognise 900 more with much better automation
  - 8 months for improvement on all fronts

- gains popularity because husky doesn't require GPU and is a new programming language that is much more easier to make right than python, and can publish papers because people are convinced.

- husky applies to Computer Graphics(GAN, etcs.), RL, Robotics

- I will raise funding for NLP and theorem proving, in 5 to 30 years will replace large language models (transformers, etc)

1.5 Now

The current state of the project is

- theoretical stuffs are clear (nothing changes significantly from two years ago); mathematically I'm fully convinced that things will work. However, to be convincing for other people with less mathematical maturity, this is probably not enough. But it doesn't really matter because I'm going to be the one who can make essential contribution to the project due to the complexity of things.
- a minimal (barely) working version of language is there, for which I wrote many code in the last two years.
  - first year work C++
  - second year Rust
- have only spent very limited time in actually making it work
  - using the C++ version a model for mnist which is 97% accuracy for half of the data
  - Mnist using the latest version in progress. Not in a hurry, because people aren't convinced by Mnist no matter how good the result is. Will do it when I have the mood.
  - Imagenet in progress,

  Remark. *todo: explain what percentage means*

1.6 Past

It initially is very different, the path is

- deep learning theory
- algorithm + machine learning
- geometric algorithm
- programming language

2 Inference Process for Image Classification

2.1 Domain Specific Definitions

Given an input space X, a label space Y, an unknown function $f : X \to Y$, we want to approximate it with high accuracy.

Remark. *We can possibly collect data on demand or have everything ready at the beginning. For inference purpose, we really don't need to care.*

Definition (Feature)**.** A feature is map from X to a type S.

Remark. *A type, rougly speaking, is a set with a specific computer representation for its elements.*

Remark. *In traditional computer vision, and*

Remark. *In the current implementation, features are computed lazily and there is a global sheet caching (selectively) the values computed. The language allows custom computational implementation for different compilation targets based on the semantic information provided by the code. We will revisit this topic in greater details in later sections.*

Definition (Option)**.** Given a type T, Option T means $T \sqcup \{\text{Null}\}$ roughly speaking.

2.2 Mnist

see `https://github.com/ancient-software/husky/blob/main/sparks/impress/vision_a_programming_approach.md`

2.3 Imagenet(tentative)

todo