Vision: A Programming Approach

Xiyu Zhai, Jiang Xin, Jian Qian, Haochuan Li, Xiao Ma, Sasha Rakhlin, Piotr Indyk

Contents

Abstract

We introduce a fundamentally novel computer vision methodology that can explore possibilities beyond traditional computer vision and end-to-end deep learning. Using just CPUs, it can develop prediction programs with equal or better accuracy, efficiency and robustness, and full explainability. The effective computation needed in training is basically ignorable as opposed to that of deep learning. However, the paradigm is difficult to implement using existing programming languages, we invented a new general purpose programming language called Husky for this purpose.

1 Introduction

The field of computer vision has witnessed a lot of success primarily powered by deep learning. But what we get is still far from ideal.

The argument is as follows: there exists statistical and computational difficulty in prediction in pattern recognition tasks in computer vision, but deep learning attempts to address both classes of problems using only statistical function fitting.

So inevitably, deep learning leads to overparametrized model, slow to predict and costly to train and demand a lot of data and is not robust due to not incorporating domain specific data.

For analysis of MNIST, see later sections.

2 Notions

We shall use pattern recognition to mean the prediction process.

What we do is not machine learning, but a more generalized feature engineering which includes machine learning as a subset.

3 Motivation

TODO

4 A New Paradigm

The typical setup of machine learning theory states everything in mathematical terms. However, for many machine learning problem, especially those involved with pattern recognition, even the prediction(inference) involves nontrivial computation.

So the difficulty is twofold: statistical and computational.

What traditional machine learning and deep learning do is to solve problems of two different nature using statistical function fitting. This works in practice, but the model is slow, large, not robust, inexplainable, costly to train.

We claim there are far better approaches.

This section describe theoretically a new paradigm. We will explain how we reformulate machine learning setup in a way so that computational structure is captured. We will also present our solutions to the newly formulated problems, which combines the good of both the programming world and the machine learning world.

The following sections will be about how it works in practice.

4.1 Computational Difficulties in Prediction

The typical setup of machine learning is like:

first you have an input space $\mathcal{X}$ and output space $\mathcal{Y}$, and a set of functions $\mathcal{H}$ from $\mathcal{X}$ to $\mathcal{Y}$. There is an unknown function $f_0 : \mathcal{X} \to \mathcal{Y}$. There is a distribution $\mathcal{P}$ over $\mathcal{X}$, and i.i.d samples x_i with $y_i = f_0(x_i)$ according to $\mathcal{P}$. Then the problem is to choose an element in $\mathcal{H}$ that best approximates this unknown function f_0.

The problem with this formulation is that everything is stated in mathematically terms without mentioning its computational information.

In many cases, we can choose a very concise $\mathcal{H}$ containing the ground truth, however functions in $\mathcal{H}$ might not be a function easy to compute, i.e. for $f \in \mathcal{H}$, it might take significant to compute for a given x the value of $f(x)$.

One possibility is that $f(x)$ could be of NP-form. Suppose that Γ is a certificate space, and $\mathfrak{s} : \mathcal{X} \times \mathcal{Y} \times \Gamma \to \mathbb{R}$ is a score function that is reasonably easy to compute, with $\mathfrak{s}(x, y; \gamma)$ representing how credible x, and suppose that

$$f(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} \sup_{\gamma \in \Gamma} \mathfrak{s}(x, y; \gamma), \tag{1}$$

then $f(x)$ might be hard to compute.

We claim this characterizes exactly the MNIST dataset.

Example (MNIST)**.** Here we describe briefly a function in mathematical terms which we believe is the ground truth for the MNIST dataset. Details can be seen in appendix.

We take the convention that the fill of the digits is white and the background is black.

The image is represented by a $[0, 1]$-valued 28×28 matrix $I = (I_{ij})_{0 \le i \le 27, 0 \le j \le 27}$.

- let's first consider how to describe an image of **digit one of the simplest kind**, which constitutes 95% of all images of digit one.

  A typical image looks like this:

  [an image here]

  Think about how it's drawn. The person when writing down a digit one like this has an ideal version in mind, a straight line that is almost vertical. So take Γ_1 to be the set of straight lines with slopes satisfying some easy constraint. Then we should define $\mathfrak{s}(x;\gamma)$ for $\gamma \in \Gamma_1$ such that

  - for "most" points over γ, it's surrounded by white pixels;
  - for "most" non-white pixels, it's away from γ.

  One choice could be

  $$\mathfrak{s}(x;\gamma) = a_1 \int_0^1 \max_{(i,j)\in[27]\times[27]} 1_{\|\gamma(t)-(i,j)\|_2<\epsilon} I_{ij} dt - a_2 \sum_{(i,j)\in[27]\times[27]} 1_{I_{ij}<0.5} \mathrm{dist}((i,j),\gamma) \tag{2}$$

  where ϵ is an appropriate small number and $a_1, a_2 > 0$ are appropriate coefficients.

  In fact the function applies when γ is any path, not necessarily a straight line. Formally it is defined over the path space (without basepoint) $\Gamma = M([0,1],[0,1]^2)$.

  The dimensionality of the configuration space is 6, which can possibly be reduced to 5.

- now let's consider a harder case, **digit seven of the simplest kind**.

  A typical image looks like this:

  [an image here]

  Here we consider all continuous $\gamma : [0,2] \to [0,1]^2$ such that $\gamma|_{[0,1]}, \gamma|_{[1,2]}$ are straight line segments. Additionally, there should be some constraint on the positions of $\gamma(0), \gamma(1), \gamma(2)$ such that $\overline{\gamma(0)\gamma(1)}$ is very close to being horizontal, and $\overline{\gamma(1)\gamma(2)}$ should be roughly vertical downward.

  The score function $\mathfrak{s}$ is the same.

  The dimensionality of the configuration space is 6, which can actually be reduced to 5.

- now let's consider a much harder case, **digit zero**.

  We consider smooth curves $\gamma : [0,L] \to [0,1]^2$ with arc length parametrization such that the mean curvature is always nonnegative, i.e.

  $$\|\gamma'(t)\| \equiv 1 \tag{3}$$

  and

  $$\gamma''(t) \times \gamma'(t) \geq 0 \tag{4}$$

  Additionally, we require that $\gamma(0)$ is very close to $\gamma(L)$.

  We should also require that γ is nondegenerate, which can be characterized by isoperimetric inequality.

  All these γ form Γ_0.

  And we still use the same score funtion $\mathfrak{s}$.

- **general case**. Fix a graph $G = (V,E)$. Give it a natural topological structure and identify each e with $[0,1]$. For each $e \in E$, we give assign a σ_e which is one of the following classes of curves:

  - nonconvex but not straight
  - nonconcave but not straight
  - straight.

  We define the total space as

  $$\Omega_G = \left\{\gamma \in M(G,[0,1]^2) : \forall e \in E, \gamma|_e \in \sigma_e\right\}. \tag{5}$$

  Then the configuration space Γ_G is a subset of Ω_G such that it is given by a boolean function s in the sense that

  $$1_{\Gamma_G}(\gamma) = s((\gamma(v))_{v\in V}, ((\gamma|_e'(0), \gamma|_e'(1), \mathrm{dist}(\gamma|_e, \overline{\gamma|_e(0)\gamma|_e(1)})))_{e\in E}) \tag{6}$$

4.2 Rethinking Hypothesis Space

4.3 Rethinking Training

4.4 Challenge of the New Paradigm

The primary challenges are

- efficient geometric algorithms are not straightforward to code, prone to bug and corner cases

- need to write system level code
- need to write high level code for feature construction, possibly with automation, and even automation of automation
- visualization of features, possibly over samples within a certain branch
- fast iteration
- large dataset
- a strong type system that expresses feature, and feature composition

Current languages like python, C++, Rust are far from being able to satisfies the needs, which is one of the primary reasons why the paradigm hasn't been discovered yet.

It's obvious that a new language is needed. It took two years for the Husky programming language to be created for this purpose.

5 Design of the Husky Programming Language

This deserves a separate long paper. But for the sake of logic completeness, we give a brief overview here.

The language itself will be general purpose. But here we focus on its usage in computer vision.

5.1 syntax

The syntax is pythonic.

One can define a feature by

```
def a_feature -> i32:
  1
```

Listing 1: Python example

5.2 type system

5.3 compilation and evaluation

5.4 debugger

6 MNIST

7 ImageNet

8 Advantages over Pure Deep Learning

8.1 Extensible

In deep learning, to test a new idea, one typically has to rerun the experiments over a large datasets using multiple GPUs, waiting for days the new results.

But in husky, one can try ideas instantaneously, based on previous people's result.

The difference is because in deep learning, everything is mixed together, a change in any parameter leads to uncontrollable change in prediction for any input. Everything is fully inexplainable.

But in Husky, a model is no different from a computer software, which is divided into packages, modules, features and functions. It's easy to isolate the difficulty, and separate things of different natures. Everything is fully explainable.

8.2

9 Future

9.1 Other Tasks in Computer Vision

One thing great about Husky is that it inherently does segmentation and detection even during classification tasks. So it would be easy to do them.

The sketching algorithm generalizes easily to 3D, resulting in fundamentally novel shape analysis. So 3D tasks are doable too.

9.2 Go Beyond Computer Vision

A Proof Details

B Language Design Details

C Mnist Details

D ImageNet Details

E Contributions

This section is for listing individual contributions.

Xiyu Zhai's contribution can be summarized as

- line sketching algorithm and convex concave components and proof of its stability
- theoretical formulation of the computational difficulty in prediction
- the perspective of field vs particle
- type theory for features
- lazy feature computation
- most of the design, implementation and maintenance of the Husky programming language
- mnist, imagenet experiments
- realization that decision list is better than decision tree

Jiang Xin's contribution can be summarized as

- help with mnist, imagenet experiments

Jian Qian's contribution can be summarized as

- clarify theoretical formulation

Haochuan Li's contribution can be summarized as

- clarify theoretical formulation

Xiao Ma: TBA.

Sasha Rakhlin provides insights from learning theory. Fundamental machine learning concepts still apply.

Piotr Indyk provides insights from applied algorithms. The computation is inspired by LSH in the early days.