Vision: A Programming Approach (work in progress)

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

Abstract

Traditional feature engineering has been largely superseded by deep learning in many pattern recognition tasks in computer vision. However, tensor-based deep learning is still far from ideal. In fact, the perfect pattern recognition program that is accuracte, efficient, robust and explainable is not within its expressive power.

We invent a new generalized feature engineering approach allow us to create models close to optimal than deep learning in all aspects, accuracy, robustness, efficiency, explainability, maintainability. What one need for the new approach are only CPUs on one personal computer, zero GPUs, even for ImageNet. The core differences from traditional feature engineerrinig are a more appropriate geometric understanding for image pattern recognition and a creation of a new programming language called Husky.

Accumulative ...

Experiments show that it's as accurate as typical neural networks in MNIST and ImageNet, with inference 20 times faster for MNIST and 100 times faster for ImageNet. The approach is applicable for most tasks in computer vision. Accuracy is as ..., explainability is ..., robustness is ...

The programming language is even applicable for other AI domains, which will be the focus of future research.

This is work in progress and the list of authors is not exhaustive. It's open for collaboration. New collaborators will be added from time to time.

It dates back to more than five years ago, when Xiyu Zhai considers theoretically how to do computer vision. It has evolved into a mixture of theory, programming language, computer vision, system and machine learning.

The project already involves a huge amount of engineering, i.e. more than 2 years of a single PhD's dedicated work (without weekends of course). Yet another year of collaboration of a big team is needed for its completion. This draft is put on arXiv so that the project has a central reference. It will be frequently updated as we obtain new results. Also every member's contribution will be faithfully recorded.

Contents

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 Related Work

2.1 Traditional Feature Engineering

2.2 Bayesian

The Bayesian is roughly like that, one has

2.3 Deep Learning

2.4 Deep Learning Acceleration

3 Motivation

3.1 Inference Efficiency

3.2 Robustness

3.3 Interpretability

3.4 Development Cost

3.5 Labeling Cost

3.6 Maintanence

3.7 A Lack of Scientific Approach

Deep learning is an amazing, but it's anything but scientific, which is well known since the very beginning.

We're not against usage of deep learning, but it's ridiculous that it's the only thing that works.

Dominance by a non-scientific approach makes the whole field looks like the middle age.

Ideally there should be a spectrum of methods, from scientific but slow to implement to non-scientific but quick to make demo. We've discovered one end of the spectrum, it's time to discover the other end.

4 Non-Trivial Computation in Pattern Recognition

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.

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 $I = (I_{ij})_{0 \le i \le 27, 0 \le j \le 27}$.

- **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.

- **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.

- **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

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

  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

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

5 A New Paradigm for Pattern Recognition

5.1 Pattern Recognition VS Machine Learning

Pattern recognition is the process of recognizing patterns in data. It involves identifying patterns in data and using those patterns to classify or categorize the data. Pattern recognition can be used to identify trends, make predictions, or classify data into different categories.

Machine learning, on the other hand, is a type of artificial intelligence that allows systems to learn and improve their performance over time, without explicitly being programmed. Machine learning algorithms analyze data and use statistical techniques to find patterns and relationships in the data. These algorithms can then be used to make predictions or take actions based on the patterns they have identified.

In short, pattern recognition is a way of identifying patterns in data, while machine learning is a way of using those patterns to make predictions or take actions.

5.2 Rethink Hypothesis Space

The ultimate goal for pattern recognition is to get an accurate efficient program which is inherently discrete. The search space of program is computational expensive to explore. For example, we can consider the space of all programs that can be described by N characters. The number of samples we need is proportional to N. But the size of the space to search is exponentially large making it impossible to directly search in space. As a result, we have to find some rules to narrow the search space so that we can find an approximate solution to the problem in a limited time.

Machine learning and deep learning targets on a search space that within a continuous hypothesis space which is typically parametrized by a high dimensional vector space. The parametrization must be "smooth" enough so that end-to-end auto-optimization is possible. For example,, one can use gradient descent algorithm to optimize a machine learning model. However, the solution found by this method will be far from the optimal if the pattern recognition involves non-trivial computation.

Since the optimal computer program of AI is from a discrete hypothesis space where the hypothesis is constructed with characters of programming languages, we will construct a coding hypothesis space. A big challenge is that the discrete coding hypothesis space is hard to be optimized automatically from mathematical.

5.3 Feature Definition

In machine learning and pattern recognition, a feature is an individual measurable property or characteristic of a phenomenon[1].

5.4 Rethink Training

Let x_i be a finite set of features, f_i be morphisms,

we use a growing sheet to search for good features.

A sheet is a sequence of features y_i such that each y_i is a simple function call on previous ys.

For example,

```
y_1 = x_1
y_2 = f_5(x_1)
y_3 = f_1(y_1, y_2)
```

If f_i are add(+) and multiply($\times$), then we can represent all polynomials.

Of course, f_i are not that simple.

By growing sheet, we mean that we are going to add new members based on training data.

Consider the possibilities of choice for y_n, it's a finite set, we can limit it to a small enough subset.

In fact, if things are strongly typed, the choice is really small.

The selection is based on a heuristics, which can be realized by firstly vectorize all y_i and feeded it to a black box machine learning algorithm.

We require something on maximum dependency depth.

Then we finish by choosing the best one and pruning away unused features.

Note that this amounts to search for multiple programs simultaneously because a sheet can be pruned to an exponential number of programs.

This can be time consuming for large datasets, so we have to do localization, which is quite small actually.

Filtering for example can effectively reduce any dataset to small ones.

Note that the biggest advantage we have compared with theorem proving is that we have a good heuristics.

There is a lot of system level consideration because some morphisms can be hard to compute.

This as a whole can be seen as a morphism, we can use

$$\text{sheet\_search}(u_1, u_2, ..., u_i) \tag{7}$$

to represent the result obtained by the above process with $u_1, ..., u_i$ added into the set of x_i

6 Domain Specifics for Image Classification

6.1 Line Segment Sketching and Convex/Convex Components

6.2 Three Families of Features

6.3 Composition of Features

7 Unprecedented Engineering Challenges

7.1 Coding

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.

7.2 Static Analysis

7.3 Compilation and Evaluation

7.4 Debugging

8 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.

8.1 Syntax

The syntax is pythonic.

One can define a feature by

```
def feature_a:
    1
```

or equivalently,

```
feature_a = 1.
```

Another feature can be defined based on it,

```
def feature_b:
    a_feature + 1
```

or equivalently

```
feature_b = a_feature + 1
```

This will make the value of feature b on any input equal to the value of feature a plus 1.

8.2 Type System

The core invention is ascension.

A type system can be seen as a category $\mathcal{C}$. Consider a functor $\mathcal{F}$ from $\mathcal{C}$ to another category $\mathcal{D}$, the functor also gives mappings from $Mor(X, Y)$ to $Mor(\mathcal{F}(X), \mathcal{F}(Y))$ for

An ascension is a collection of functors like these.

8.3 Compiler and Evaluator

8.4 Debugger

9 Results on MNIST

9.1 Overview

9.2 Digit One

10 Results on ImageNet

10.1 Overview

10.2 Husky Dog

10.3 Samoyed Dog

10.4 Piano

11 Advantages over Pure Deep Learning

11.1 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.

11.2

12 Future

12.1 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.

12.2 Go Beyond Computer Vision

A Related Work Details

B Image Classification Domain Specific Details

B.1 Line Sketch Algorithms

C Language Design Details

D Mnist Details

D.1 Overview

D.2 Digit One

D.3 Digit Two

D.4 Digit Three

D.5 Digit Four

D.6 Digit Five

E ImageNet Details

E.1 Overview

E.2 Husky

E.3 Piano

E.4 Dog

E.5 Cat

E.6 Object

F 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.

References

[1] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.