Jump Detection Involving Convolutional Neural Network

October 18, 2019

Abstract

In this paper we mainly present methods based on convolutional neural networks (CNNs), and model combining CNNs and polynomial annihilation to solve jump detection problem on functions in one dimensional, two dimensional and three dimensional spaces. We compare the performances of CNN and polynomial annihilation, and illustrates the model is available for functions with different domain and uniform grid. Besides, we also come up with method for jumps in first derivative detection. We also mention that the approaches can be extended to high dimensional space.

1 Introduction

The problem of jump detection has attracted intensive concern for a long time. A function $f \in \mathbb{R}^n$ is continuous at x_0, if and only if $\lim_{x \to a^-} f(x) = \lim_{x \to a^+} f(x) = f(a)$, otherwise x_0 is called a discontinuity for f. There are mainly three types of discontinuity: removable discontinuity, jump discontinuity, and infinite discontinuity. Typically the point x_0 is a jump when $\lim_{x \to a^-} f(x) \neq \lim_{x \to a^+} f(x)$. The aim for solving detection problem is to find the locations of jumps given a function. In this paper we only focus on jump detection problem.

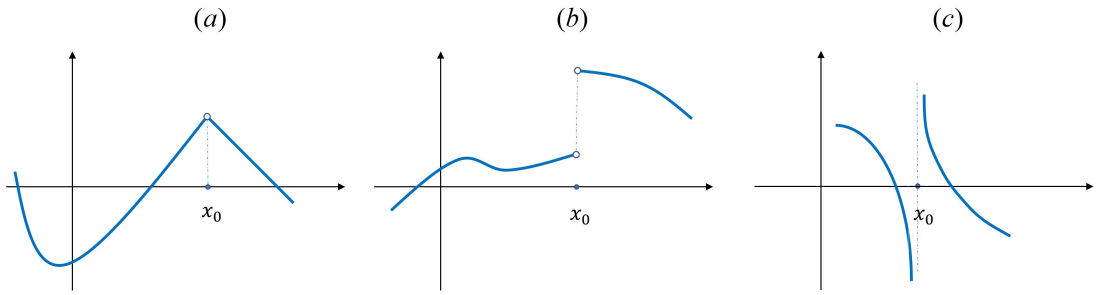


Figure 1: Three types of discontinuity. (a) is removable discontinuity, (b) is jump discontinuity, (c) is infinite discontinuity.

There are many applications of jump detection. The most common study is edge detection in vision field, another essential application is in the numerical simulation of partial differential equations. Many numerical methods for solving partial differential equations are closely associated to the smoothness of domain, therefore known the location of jumps are of interest.

There are lots of literature talking about the jump detection. Detection methods on high dimensional space are also discussed ([1],[13]). Some studies extend the detection problem to detection for discontinuity in derivatives ([7]). Some of those methods use properties of wavelets ([2],[14]), some construct detecting filters ([9],[13]). In recent years, polynomial annihilation detection method ([6],[7],[10]) is a numerical approach especially for jump detection. It is first introduced in "Polynomial Fitting For Edge Detection in Irregularly Sampled Signals and Images" ([10]). The method is mainly based on the property of Taylor expansion. This method can also been extended to discontinuity detection in derivatives.

The following paper is organized in this way:
Section 2 briefly introduces two methods: convolutional neural network (CNN) and polynomial annihilation, which will be used in our models. In section 3, a detection approach based on CNN in one dimensional space, and two dimensional detection approaches involving CNNs and polynomial annihilation method are specifically described. In section 4 we show experimental results and elaborate that the approaches can be adapt to any domain and grid. Section 5 is a supplement talking about some details in experiments. In Section 6 we bring up some extended problems and their solutions, such as how to detect discontinuity in derivatives, and how will the model work in high dimensional space. Other implement details will be in the appendix, including data generation, architectures of models mentioned in the paper.

2 Background

2.1 Introduction to CNN

Convolutional Neural Network(CNN) is a special type of neural network. It was first introduced for solving the problem of document recognition([16]), and becomes well-known thanks to the success of its applications on image classification([5]). Neural network is a representation of a sequence of functions, its architecture consists of multiple layers made of neurons with learnable parameters. A CNN may contains convolutional layer, pooling layer, batch normalization layer and so on. Convolutional layer is the core layer in CNN.

2.1.1 Convolutional Layer

A mathematical definition of convolution is

$$s(t) = (x * w)(t) = \int x(a)w(t-a)dw \tag{1}$$

$s(t)$ is an estimate of the position of spaceship calculated by a weighted average operation $w(a)$. The action of convolutional layer is similar to the definition, which aims to get a weighted average of a certain range of input by an operation called kernel, or filter. Besides, in a convolutinoal layer, a stride is the size of the step the kernel moves each time, and an activation function is a non-linear transformation to the input. Sometimes to avoid the shrink of output (or feature map of the layer), we add zeros surround the input, those zeros are call zeros padding.

Let the input to the $i'th$ convolutional layer be $X^{(i-1)} \in \mathbb{R}^{n_{i-1} \times m_{i-1} \times K_{i-1}}$ with input height n_{i-1}, width m_{i-1} and K_{i-1} channels.

Assume there are K_i kernels in the convolutional layer. Denote the kernel set as $\{Ker_k\}_{k=1,\cdots K_i}$, $Ker_k \in \mathbb{R}^{h \times w}$. let $k'th$ kernel has weight matrix $W_i^{(k)}$. Denote the activation function as $\sigma(x)$. The stride of height and width are $d_i^{(1)}, d_i^{(2)}$. Usually stride and kernel size are decided by the input size and padding.

With the set up above, let the output of this layer be $X^{(i)} \in \mathbb{R}^{n_i \times m_i \times K_i}$. The element $x_{a,b,k}^{(i)}$ in the output is the calculated by the product of $W_i^{(k)}$ and the corresponding patch $X_{(a,b)}^{(i-1)}$ with size $h \times w \times K_{i-1}$ extracted from $X^{(i-1)}$, plus a bias $b_i^{(k)}$. The outcome is then transformed by the activation function σ. In CNN the most widely used activation function nowadays is rectified linear units (ReLU), $\sigma(x) = max(0, x)$.

$$x_{a,b,p}^{(i)} = \sigma(< W_i^{(k)}, X_{(a,b)}^{(i-1)} > + b_i^{(k)}) \tag{2}$$

where $< \cdot, \cdot >$ is tensor inner product:
For 2 tensors $A, B \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, the inner product between two tensors is

$$< A, B >= \sum_{i_1, i_2, \cdots, i_d} a_{i_1, i_2, \cdots, i_d} b_{i_1, i_2, \cdots, i_d} \tag{3}$$

2.1.2 Pooling Layer

A pooling function calculates and outputs a summary statistic for the inputs in the neighborhood of a certain location. The statistic is usually calculated by taking the averages (Average Pooling) or maximum values (Max Pooling). Define the input for $i'th$ pooling layer as $X^{(i-1)}$ with size $\mathbb{R}^{n_{i-1} \times m_{i-1} \times K_{i-1}}$, and pooling size is $s_i^{(1)} \times s_i^{(2)}$, with stride $(d_i^{(1)}, d_i^{(2)})$ and activation function σ. For max pooling function, the element

$x_{a,b}^{(i)}$ in the output $X^{(i)}$ is calculated by taking the maximum value of the corresponding patch $X_{(a,b)}^{(i-1)}$ with size $h \times w \times K_{i-1}$ extracted from $X^{(i-1)}$, then the maximum value is transformed by the activation function σ.

$$x_{a,b}^{(i)} = \|X_{(a,b)}^{(i-1)}\|_{max} \tag{4}$$

2.2 Introduction to polynomial annihilation detection

Polynomial annihilation detection is mainly based on the property of Taylor expansion. The detection method essentially approximates the remaining terms of the Taylor series expansion of $f(x)$ which rapidly converges to zero, while this method converges to some scaled value of jump heights at the jump location.([7])

let S be a set of discrete points in Ω, where Ω is a bounded domain in $\mathbb{R}^d$. f is a piecewise smooth function only known on S. Every point is defined as $x = (x_{(1)}, ...x_{(d)}) \in \mathbb{R}^d$. The polynomial annihilation is to construct a function $L_m f(x)$, $m \in \mathbb{N}$, s.t. if x is away from discontinuities, $L_m f(x) \to 0$. Therefore using a threshold on $L_m f(x)$ we can determine the location of discontinuity.

To determine the value of $L_m f$ on x, neighboring information is employed. For a pre-determined $m \in \mathbb{Z}^+$, denotes the space of all polynomials of degree $\leq m$ in $d \in \mathbb{N}$ variables as Π_m, and denote the dimension of Π_m as m_d:

$$m_d = \begin{pmatrix} m+d \\ d \end{pmatrix}$$

for $\forall x \in \Omega$, pick a set $S_x := S_{m_d,x} := \{x_1, ...x_{m_d}\}$, which is a local set of m_d points around x.
The next step is to solve a linear system for coefficients $c_j(x)$

$$\sum\nolimits_{x_j \in S_x} c_j(x) p_i(x_j) = \sum_{|\alpha|_1 = m} p_i^{(\alpha)}(x),\ \alpha \in \mathbb{Z}_+^d \tag{5}$$

where $\alpha \in \{(\alpha_1, \cdots \alpha_d) : \alpha_1, \cdots \alpha_d \in \mathbb{Z}_+\}$, p_i, $i = 1, ...m_d$ is a bases of Π_m (In this work we use a polynomial basis). $p_i^{(\alpha)}(x)$ is taking partial derivative of $p_j(x)$:

$$p_i^{(\alpha)}(x) = \frac{\partial^{\alpha_1 + \cdots + \alpha_d} p_i(x)}{\partial {x_{(1)}}^{\alpha_1} \cdots \partial {x_{(d)}}^{\alpha_d}} \tag{6}$$

Typically in 1-dimensional case,

$$\sum\nolimits_{x_j \in S_x} c_j(x) p_i(x_j) = p_i^m(x),\ i = 1, ...m \tag{7}$$

The solution of the equation system is existing and unique. After calculating the coefficients, the approximation function $L_m f(x)$ is constructed by

$$L_m f(x) = \frac{1}{q_{m,d}(x)} \sum\nolimits_{x_j \in S_x} c_j(x) f(x_j) \tag{8}$$

Where $q_{m,d}(x)$ is a normalization factor.
The approximation to the actual jump heights of $L_m f$ is guaranteed by theorem 3.1 in the paper. In one dimensional case, constrain $S \subset [a,b]$ and discontinuity set $J = \{\xi : a \leq \xi \leq b\}$. Define the local jump function of f as

$$[f](x) := f(x+) - f(x-) \tag{9}$$

$$h(x) := max\{|x_i - x_{i-1}| : x_{i-1}, x_i \in S_x\} \tag{10}$$

If the normalization factor $q_m(x)$ is determined to be $\sum_{x_j \in S_x^+} c_j(x)$, where

$$S_x^+ := \{x_j \in S_x | x_j \geq x\} \tag{11}$$

The number of points in S_x is $m+1$, then the theorem states that $L_m f$ is approximate to the local jump function with high accuracy:

$$L_m f(x) = \begin{cases} |f|(\xi) + \mathcal{O}(h(x)), \ x_{j-1} \leq \xi, x \leq x_j, \\ \mathcal{O}(h^{min(m,k)}(x)), \ f \in C^k(I_x) \quad for \ k > 0 \end{cases} \tag{12}$$

I_x is the smallest closed interval s.t. $S_x \subset I_x$.

The strength of polynomial annihilation is it can adapt to any irregular data and domain. Moreover, this method can be extended to jump detection of functions on $C^\gamma[a,b]$, $\gamma \in 1, 2, \cdots$.

3 Detection methods

3.1 General notations

In this section we introduce notations that would be used in the following descriptions. In a d-dimensional space, each point $x = (x_{(1)}, \cdots x_{(d)}) \in \mathbb{R}^d$. Ω is a bounded domain in $\mathbb{R}^d$. Let S be a set of finite discrete points in Ω, $|S| = K$. Therefore for a piece-wise smooth function f is known only on set S.

We use a pre-determined grid to define set S. A grid G on Ω consists of K points and N unit grids. Denote jump set of f as $J = \{\xi_k | 1 \leq k \leq m_f, \xi_k \in \Omega\}$, where m_f is the number of discontinuities in function f on Ω.

3.2 One-dimensional approach

In this section we come up with an new approach using CNNs for detection. The goal of this approach is to construct a function on Ω, which can approximately represent the location of discontinuities, the idea of which is similar to polynomial annihilation detection. The model is expected to tell if there is discontinuity for any unit grid.

In one dimensional space, the number of unit intervals $N = K - 1$. For a piece-wise smooth function f known only on set S, the jump location set is defined as

$$J = \{\xi_k | 1 \leq k \leq m, a \leq \xi_k \leq b\} \tag{13}$$

where a and b are boundaries of the function. Define the vector v_f as

$$v_f = \{f(x_k)\}_{k=1,\cdots N} \tag{14}$$

This is a vector containing all values of function f known on point set S. Based on the set, we use an indicator vector as ground truth to represent the locations of jumps:

$$y_f = (y_{(1)}, y_{(2)}, ..., y_{(K-1)}) \tag{15}$$

where each element is corresponding to a unit grid:

$$y_{(i)} = \delta(x_i \leq \xi_k < x_{i+1}, \forall 1 \leq k \leq m),\ i = 1, ...K - 1 \tag{16}$$

where $\delta(\cdot)$ is characteristic function. Here $y_{(i)}$ is an indicator of the interval $[x_i, x_{i+1})$ whether it contains a discontinuity or not.It might be noted that besides using 0 and 1 indicator, we can use other values.

In the first step of prediction, first we standardize v_f and treat it as the CNN input, the input size is $1 \times K$. The reason for standardization is to avoid the ineffectiveness caused by different scales in functions. The standardization formula is

$$v_f^1 = \frac{v_f - \mu_{v_f}}{\sigma_{v_f}} \tag{17}$$

Where μ_{v_f} is the mean of v_f and σ_{v_f} is the standard deviation.
y_f is the ground truth used for model training, or the function we want to approximate by the output, so the CNN output size is $K - 1$.
Define the output of CNN as $\mathcal{N}(v_f)$, which is a $K - 1$-length vector. As for objective function, we use Root mean squared loss:

$$RMSE = \sqrt{\frac{1}{K-1}\|y_f - \mathcal{N}(v_f)\|_2^2} \tag{18}$$

Given any test function f_t, get the output $\mathcal{N}(v_{f_t})$ from the trained CNN model. The output of an effective model should be a good approximation function to the ground truth of f_t. Pre-determine a threshold t. For each element $\mathcal{N}(v_{f_t})_i$, $i = 1, ...K - 1$, if $\mathcal{N}(v_{f_t})_i > t$, then the interval $[x_i, x_{i+1})$ is considered to contain a discontinuity.
Briefly summary the steps of one-dimensional CNN approach:
0. Generate training data and their ground truths, standardize the training data and train a CNN model $\mathcal{N}$.
1. In prediction procedure, first standardize the testing function f_t.
2. Get the output $\mathcal{N}(v_{f_t})$ from the trained CNN.
3. Choose a threshold t, the interval $[x_i, x_i + 1)$ is predicted to contain a discontinuity if $\mathcal{N}(v_{f_t})_i > t$.

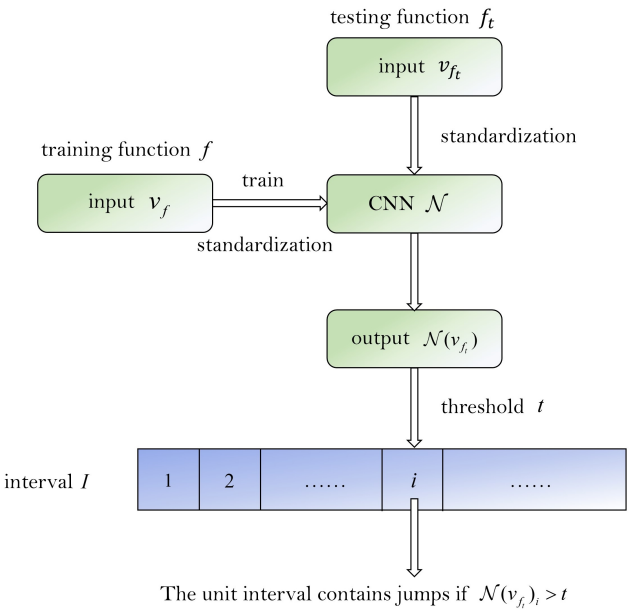


Figure 2: Diagram for 1-d approach. The output y has length $K-1$, each element is corresponding to the detection result of an unit interval $[x_i, x_{i+1})$.

3.2.1 General model for any domain and uniform grid

From the model, it is obviously the domain doesn't have effect on detection since the information of grid size and domain is not a input to the model. For example, when we shift function from $[-1, 1]$ to $[2, 3]$, there is no difference for the input to the model. As for the grid size, although the input may reflect some attributes of functions, the grid size is not a direct input. During the experiment, we generate functions with different domain and grid size. The result also shows that the model has same detection rates on different domain. Unless the grid is too sparse, it may cause a problem of lack of resolution. We will talk about this in the next section. Generally speaking, we don't need to train different models to fit various domains, length of inputs or grids.

In general cases, if the domain is $\Omega^{'}$, with $K^{'}$ points, we can adopt a sliding window with step sides d to detect on each grid which has the same size as G. Particularly, if $K^{'} < K$, first we extend the data to K points by adding values on both sizes, otherwise we let the input keep the same as original.

The general prediction procedure is described as the following:
0. Standardize the function.
1. If function v has length $K^{'} < K$, then we extend the right side with number of points $\lfloor a = (K - |v|)/2 \rfloor$,and right side with number of points $b = K - |v| - a$, where the values

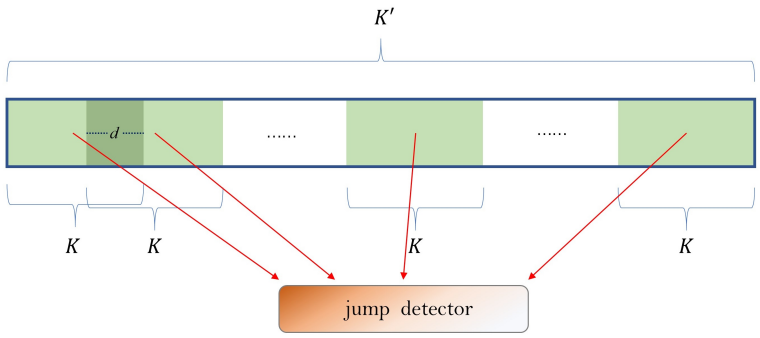


Figure 3: In general case, use a sliding window and apply CNN model on each piece.

of left added points is equal to the value of first points in function, the values of right added points is equal to the value of last points in function. If length is greater or equal to K, keep it the same. Call the new vector v^*

2. From the first point v_1^*, apply the model to first piece $(v_1^*, \cdots, v_K^*)$, record the index of output that greater than threshold t. Slide d points, and repeat the operation, until sliding to the last piece, the tail point of which is the last point of v^*.

3. Combine all the locations that indicating containing jumps in the function, this would be the final outcome.

3.3 Two-dimensional Approach

In two dimensional space, a function f is on a domain $\Omega \subset \mathbb{R}^2$. Assume there are K_1 points evenly aligned to the height of the grid, and K_2 points to the width, then $|S| = K_1 \times K_2$, $N = (K_1 - 1) \times (K_2 - 1)$. Denote the point at $i'th$ row and $j'th$ column of grid G as $x_{i,j}$. We define the set of all four vertices of each unit grid as

$$C = \{c_{i,j} | c_{i,j} = \{x_{i,j}, x_{i,j+1}, x_{i+1,j}, x_{i+1,j+1}\}, i = 1, ...K_1 - 1, j = 1, ...K_2 - 1\} \quad (19)$$

Define $K_{c_{i,j}}$ as the convex hull of the set of vertices $c_{i,j}$.

The initial idea is the same as in one dimensional case, which is applying a trained CNN to get an output that approximates the actual locations of jumps. However In high dimensional case, the computational costs raise rapidly for both CNN and numerical methods. Since the last layer of CNN is usually a dense layer, with the output size increases exponentially, the number of parameters could become very large. Therefore a coarse-to-fine strategy might be helpful to avoid curse of dimensionality.

3.3.1 One step detection

In one step detection approach, we still apply a single CNN to predict if a certain range of locations contain discontinuity. First we assign order index to each unit grids in G by row first. Using elements in set C as representations of unit grids, the order of unit grids becomes $(c_{1,1}, c_{1,2}, \cdots c_{1,K_2-1}, c_{2,1}, \cdots c_{K_1-1,1}, \cdots c_{K_1-1,K_2-1})$. We assign index $1, 2, \cdots (K_1-1)(K_2-1)$ to the corresponding unit grids, and denote unit grids as $\{G_i\}_{i=1,\cdots N}$. This step makes it convenient to construct a ground truth vector, with each element in the vector corresponding to a certain unit grid. if there is discontinuity inside the unit grid, then the indicator function is 1, otherwise is 0.

$$y_f = (y_{(1)}, y_{(2)}, ..., y_N) \tag{20}$$

$$y_{(i-1)\times(K_2-1)+j} = \delta(\xi_k \in K_{c_{i,j}}, \quad \forall 1 \le k \le m) \tag{21}$$

$i = 1, ...K_1 - 1$, $j = 1, ...K_2 - 1$.
Define v_f as a matrix of all values in function f on S:

$$\begin{bmatrix} f(x_{1,1}) & f(x_{1,2}) & \cdots & f(x_{1,K_2}) \\ \vdots & \vdots & \ddots & \vdots \\ f(x_{K_1,1}) & f(x_{K_1,2}) & \cdots & f(x_{K_1,K_2}) \end{bmatrix} \tag{22}$$

With a CNN model $\mathcal{N}$ trained by functions f and ground truths v_f, The detection approach is the same as it in one dimensional case. For a testing function f_t and output $\mathcal{N}(v_{f_t})$, if $\mathcal{N}(v_{f_t})_i \ge t$, we predict there is a discontinuity in the corresponding unit grid.

3.3.2 Two Step Detection

However, due to the large amount of parameters in CNN and computational cost for detection on each unit grid in the first method, a fungible strategy instead is coarse-to-fine. We split the whole detection problem into two parts:
1. Instead of detecting discontinuity on each unit grid directly, we first apply a CNN detector on a coarse range of position. we divide the whole grid G into several smaller grids. Denote the set of all those smaller grids as $G_s = \{G_i^s | i = 1, \cdots N_G\}$. The index of smaller grids are also assigned by row first. G_s should cover all unit grids in G and ideally each two grids in G_s should have no overlaps (However the points can be shared for different grid G_i^s). Therefore the first stage of the problem becomes detecting discontinuity in every G_i^s in G_s.
2. For each G_i^s predicted containing discontinuities through the first step, we apply another detector on G_i^s to detect discontinuity by each unit grids in G_i^s.

Through the coarse-to-fine strategy, computational cost is reduced for those G_i^s which are predicted not containing any discontinuity in the first step.
For the first stage, we still use CNN for detection; we adopt two different approaches for

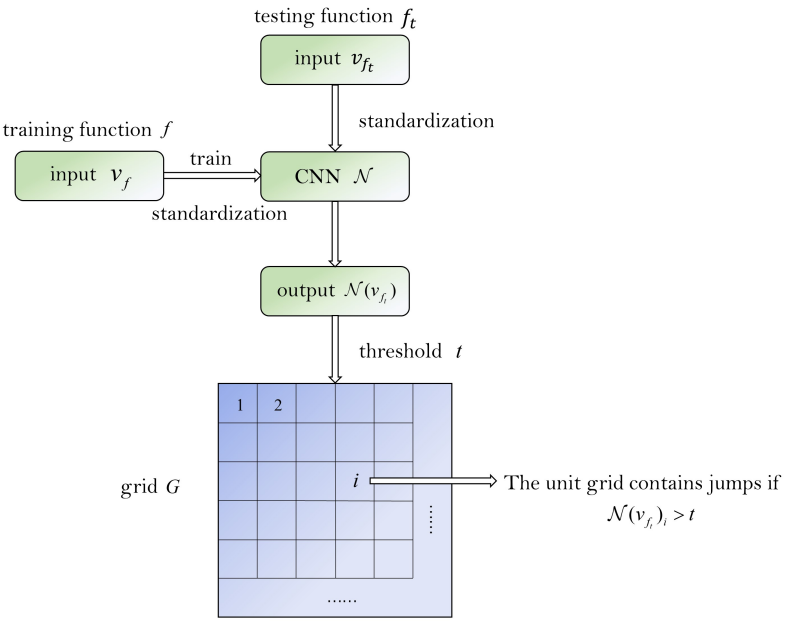


Figure 4: 2-d approach with one step. The procedure is almost the same as it in 1-d approach, except we need to pre-assign order index to each unit grid to let it be corresponding to a certain element in the ground truth vector.

the second stage: CNN and Polynomial annihilation method, and compare their performances.

a. CNN+CNN approach (2CNNs)
In the first step, divide grid G into $N_1 = \frac{K_1-1}{L_1} \times \frac{K_2-1}{L_2}$ smaller grids, where we decide every smaller grid G_i has L_1 unit grids in column and L_2 unit grids in row, therefore each smaller grid contains $L_1 \times L_2$ unit grids. To train a CNN for the first step, firstly f still need to be standardized.The ground truth in first stage is

$$y_f = (y_{(1)}, y_{(2)}, ..., y_{(N_1)}) \tag{23}$$

Each $y_{(i)}$ is an indicator function whether the corresponding grid G_i^s contains discontinuities. The training input v_f is kept to be the same as before.
With a trained CNN model called $\mathcal{N}_1(\cdot)$, apply it on the testing function f_t to get the output $\mathcal{N}_1(v_{f_t})$. Set a threshold t_1, If $\mathcal{N}_1(v_{f_t})_i \geq t_1$, then add the grid G_i^s into a candidate set called G_c for the next step.
For the second stage, we train a new CNN model called $\mathcal{N}_2(\cdot)$ with input size $(L_1+1) \times (L_2+1)$, since there are (L_1+1) points in column and (L_2+1) points in row on every smaller grid G_i^s. The output size is the number of unit grids in G_i^s, which is denoted as $N_2 = L_1 \times L_2$. Every training data is a vector of all known points on those G_s that the area inside actually contains discontinuities. For example, in a training function f, if there are discontinuities in grid G_i^s, then we treat all known values on G_i^s as a matrix v_f^i to be a training input. The ground truth vector for this input is defined to be

$$y_f^i = (y_{(1)}^i, y_{(2)}^i, ..., y_{(N_2)}^i) \tag{24}$$

$y_{(j)}^i$ indicates whether there is a discontinuity in $j'th$ unit grid in G_i^s or not.
With these set up, a CNN model $\mathcal{N}_2(\cdot)$ can be trained. The training samples are function values on all G_i^s containing discontinuities of all training functions. With a testing function f_t and the candidate set G_c already been constructed through the first step, we apply $\mathcal{N}_2(\cdot)$ to each element in G_c, thus detect on all unit grids in smaller grids that considered to contain discontinuities with high probability.

b. CNN+Polynomial annihilation approach (CNNPoly)
The first stage is same as it in 2CNNs method. For the second stage, we perform polynomial annihilation method on each candidate grid in G_c for the center point of every unit grids, with a threshold t_2. For detecting on grid $G_i^s \in G_c$, Denote the center point of a cell to be x, and all points on G_i^s forms a set $S_{G_i^s}$. Construct the approximation function $L_m f(x)$ based on m_d closest points around x_i^s in the set $S_{G_i^s}$. If $L_m f(x) \geq t_2$, the corresponding cell is predicted to have discontinuity.

3.3.3 General model for any domain and grid

Similar to one dimension situation, for function on domain Ω' with grid size ${K_1}' \times {K_2}'$:
0. Standardize the function.

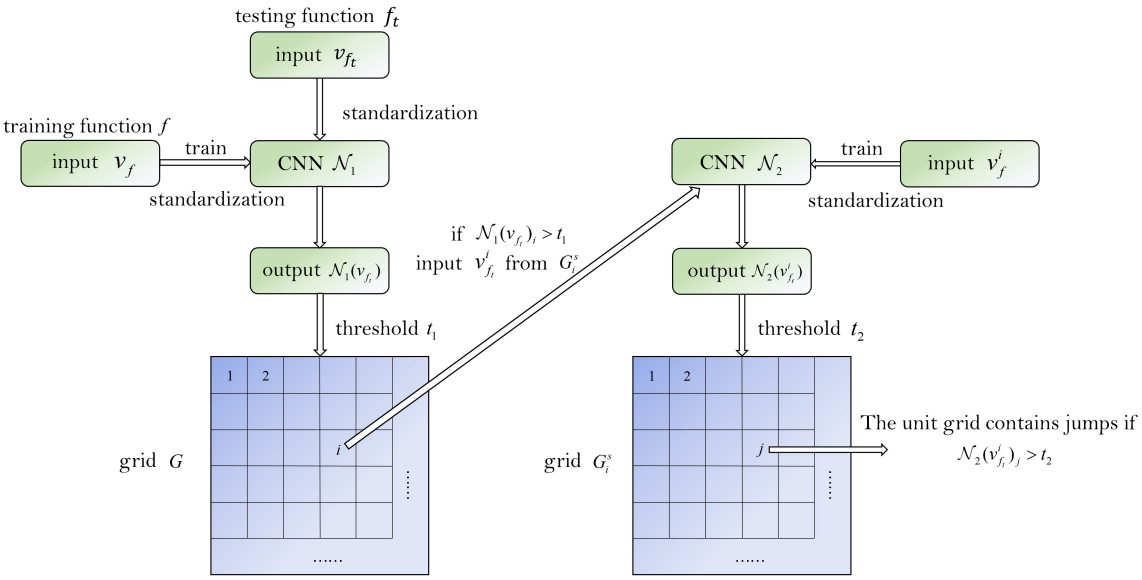


Figure 5: Diagram of 2-d approach with two detection steps: CNN+CNN method.

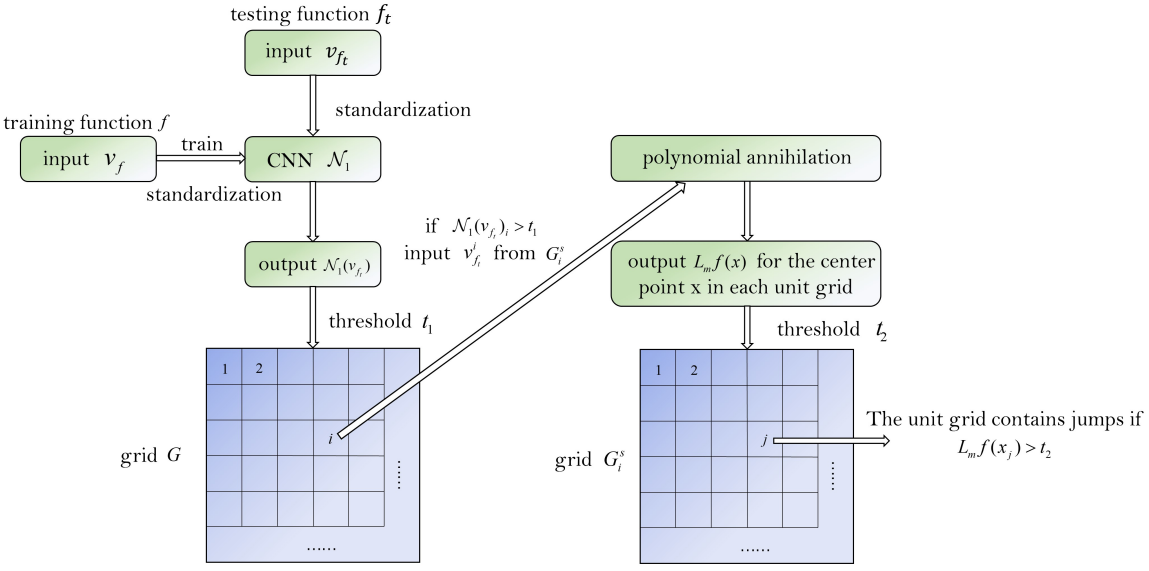


Figure 6: Diagram of 2-d approach with two detection steps: CNN+Polynomial annihilation method.

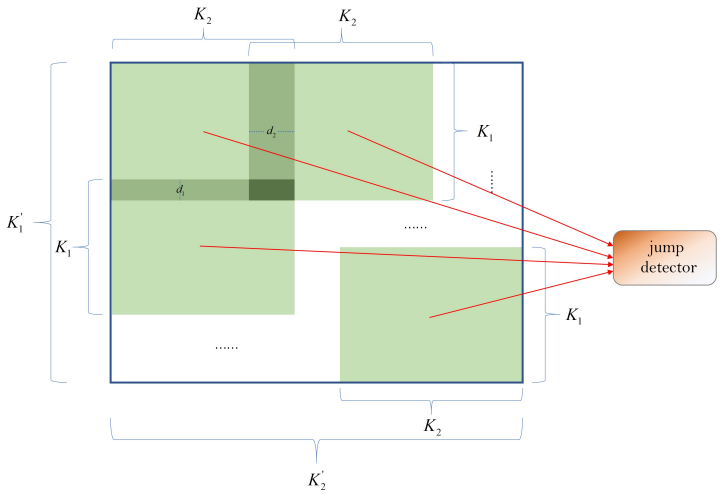


Figure 7: This figure explains how the sliding window works on general function in two-dimensional space.

1. If function v has length $L_1{}'$ and $L_2{}'$. If $K_1{}' < K_1, K_2{}' \geq K_2$,then extend the height to K_1, with top add $a = (K_1 - K_1{}')/2$ points, bottom $b = K_1 - K_1{}' - a$ points. $K_2{}' < K_2, K_1{}' \geq K_1$, extend the width to K_1, with left add $a = (K_2 - K_2{}')/2$ points, right $b = K_2{}^2 - K_2 - a$ points. If $K_1{}' < K_1, K_2{}' < K_2$, then extend it to a $K_1 \times K_2$ rectangle, value of each points added are the value of its nearest points in the original function. Each time the sliding window remove d_1 points by column or d_2 points by row.
2. From the first point v_1^*, apply the model to first piece of data $(v_1^*, \cdots, v_K^*)$, record the index of output that greater than threshold t. use a sliding window with training grid size to repeat the operation, until last piece, the last point is the last point of v^*. To avoid mis-prediction caused by adding artificial data, if the sliding window is not covered by real data, we move it to the nearest location that inside the original grid.
3. Combine predictions from all pieces of input, it is the final prediction. In this way we detect any two-dimensional bounded functions with limited values.

4 Experimental results

We use tensorflow and keras for CNN implementation on Ohio super computer owens, which has login nodes Intel Xeon E5-2680 (Broadwell) CPUs with 28 cores per node and 256GB of memory per node. GPU computing is supported by CUDA10.0.31. The P100 "Pascal" is a NVIDIA GPU with a compute capability of 6.0. Each P100 has 16GB of on-board memory and there is one GPU per GPU node.

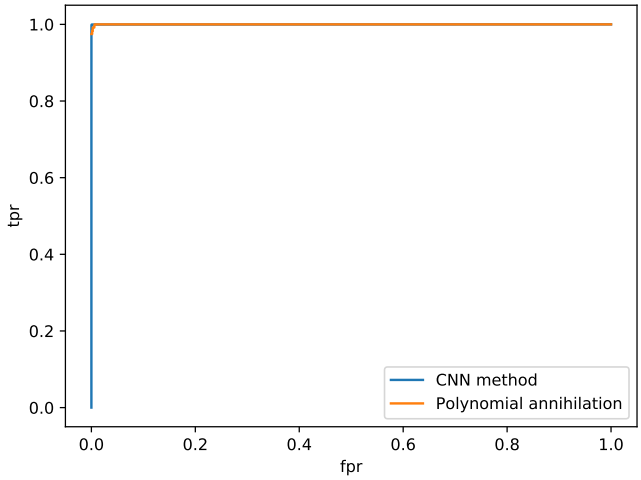


Figure 8: ROC curves of CNN and numerical methods.The y-axis is true positive rate (TPR), the x-axis is false positive rate (FPR). A true case(T) is an unit interval $[x_i, x_{i+1})$ really contains a jump, a false case(F) is there is no jump in the interval, thus true Positive(TP) means the situation that the unit interval actually containing a jump has been detected having a jump by the model, which is equivalent to $\mathcal{N}(v_f)_i > t$ and false Positive(FP) means an interval that does not contain a jump but $\mathcal{N}(v_f)_i > t$.

4.1 One dimensional approach

4.1.1 One dimensional simulation

Data generation procedure is described in appendix A.1. We generate 1000,000 training functions, with the largest allowed number of jumps in each function $M = 3$. Each input data contains 202 points on $\Omega[-1, 1]$. The architecture of model is described in appendix B.1.

We use 10,000 testing functions in the same domain and grid as training data to evaluate the performance of Both CNN approach and polynomial annihilation. From the ROC figure, the two curves have very large area under the curves, and are nearly overlapped with each other, which means both of them show good performance. Therefore both of two methods can solve the one-dimensional detection problem well.

4.1.2 Model robustness on data with more jumps

In this section, we apply the trained CNN model on testing functions with $M = 6$. By viewing prediction figures, the result reflects the model is robustness when there are more

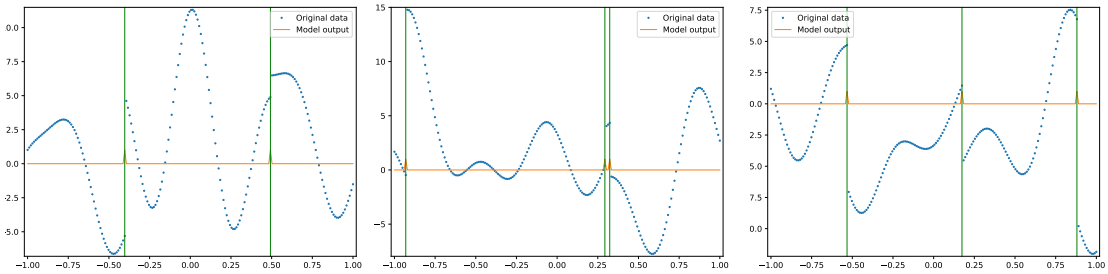


Figure 9: Some detection results using trained CNN model. The blue curve is the real function on 202 points in $[-1, 1]$, and the green vertical lines are the left boundary point of intervals containing jumps. The orange curve is the output of CNN model, which has 201 points. We can see from the figure that near the location of jumps, the output values are approaching to 1, otherwise they are nearly 0.

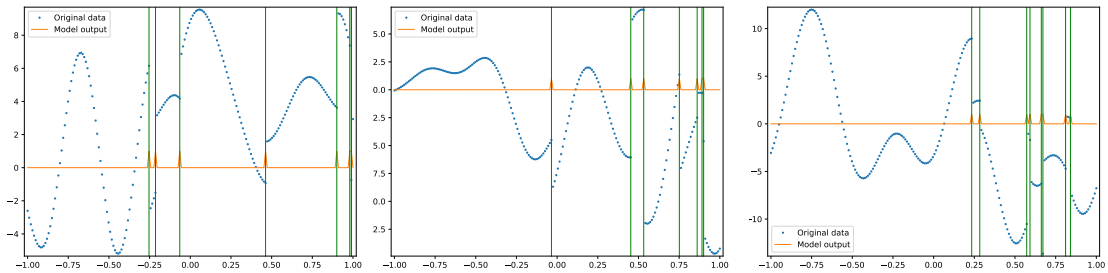


Figure 10: Testing results on function generated with $M = 6$ with CNN trained only on functions with $M = 3$. The function f_t is consisting of blue discrete points, where the yellow curve is the output $\mathcal{N}(v_{f_t})$. The green vertical lines are across the left boundary point of the unit intervals that actually contain jumps. We can see that when near the location of discontinuity, the values of $\mathcal{N}(v_{f_t})$ is approaching to 1; when being far away from jumps, the output values are nearly 0.

jumps in the function, which never shows up in training data. This result demonstrates the CNN approach can be used to detect one dimensional functions with any number of jumps as long as having enough resolution.

4.1.3 General cases

Using sliding window, we can get predictions of functions on different domains and grid sizes. We generate testing data on different grids, thus the domain is also changed during testing. we set sliding step d to be $d = 100$.According to the figure, when data is relatively sparse comparing to training data, the output seems to contain some noise. With grid gradually becoming dense, the prediction becomes essentially accurate. This results shows that the trained model can be adapted to any bounded function

with enough resolution, therefore we don't need to train models with different hyper-parameter setups.

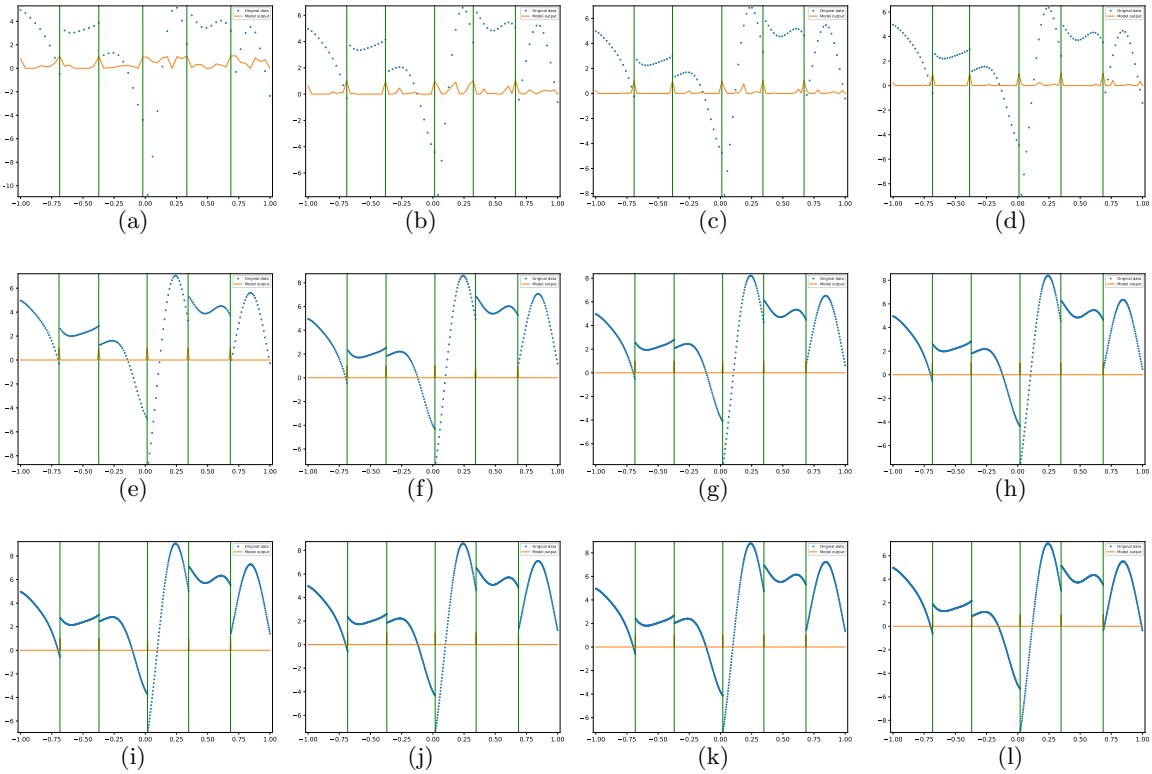


Figure 11: 13 inputs are generated from the same function on $[-1, 1]$, by picking different number of data points evenly. From figure (a) to figure (l), the numbers of points picked are 52,72,92,102,202,302,402,502,602,702,802,902.

4.2 Two dimensional approach

4.2.1 One step detection approach

In two-dimensional space, define $\Omega = [0, 1]^2$. The grid G contains 101×101 uniform points, therefore $N = 100 \times 100$. In this case we only consider jump curves between two functions f_1 and f_2.

Using generation methods described in appendix A.3, we generate 64000 training functions by line cut, and 64000 training functions by circle cut. Although there are so many different shapes of jump curves, we will see later that the CNN model trained on these two types of functions still have good performance for detection on functions with other shapes of jump curve.

We tried 4 sizes of L_1 and L_2 to divide the whole grid into smaller grids:$L_1 = L_2 = 1$, $L_1 = L_2 = 2$, $L_1 = L_2 = 4$, $L_1 = L_2 = 10$. The last 3 grid can only recognize if there are jumps in a large range. Therefore the corresponding number of a smaller grid G_i^s for a function in each situation is 10000, 2500, 625 and 100. For these 4 different sizes, the CNN models have the same architecture, except the output sizes are 10000, 2500, 625 and 100. The architecture is described in B.3.

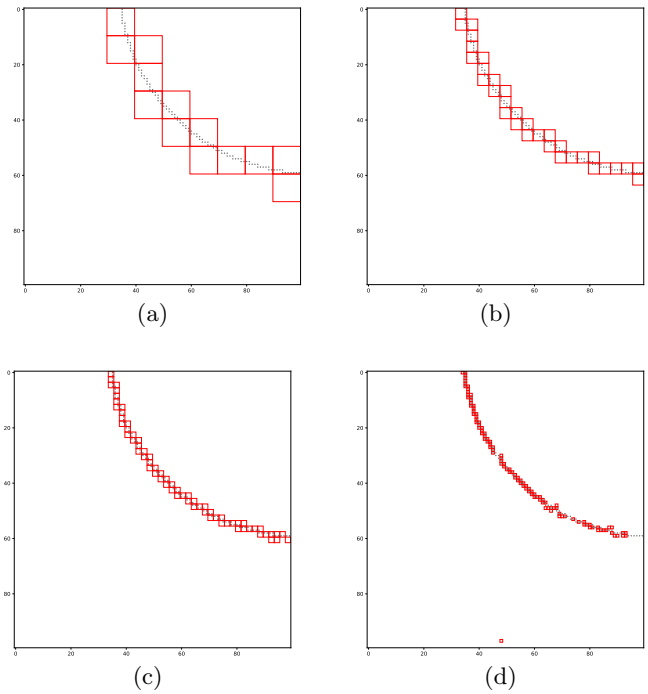


Figure 12: (a),(b),(c),(d) are the testing results for the same function with $L_1 = L_2$=10,4,2,1. Set threshold $t = 0.25$, for each unit grid, if its corresponding output value is greater than t, stroke the edges of the unit grid with red color; For those unit grids that actually contain a discontinuity, we draw a black point at the center of the unit grid.

4.2.2 Two step detection approach

a. CNN+CNN(2CNNs) approach We choose $L_1 = L_2 = 10$ for the first step model, since it's model has shown a high prediction accuracy in the above experiment, and also saves computational cost for the second step. G_i^s consists of 10×10 unit grids, therefore it has 11×11 points. The architecture of $\mathcal{N}_2(\cdot)$ is described in appendix B.3.

Set the thresholds $t_1 = 0.28$, $t_2 = 0.22$. In this case, for a testing function f_t and its candidate set G_c from the first step, apply the trained $\mathcal{N}_2$ to each grid in the set. If

$\mathcal{N}_2(v^i_{f_t})_j \geq t_2$, the corresponding $j'th$ unit grid in G^s_i is claimed to contain a jump.

b. CNN+Polynomial annihilation(CNNPoly) approach
For $\forall G^s_i \in G_c$, apply polynomial annihilation on center points $x_{i,j}$ for each unit grid c_j in G^s_i. Let $m_d = 3$, and threshold of CNN is $t_1 = 0.28$, threshold for Polynomial annihilation is $t_2 = 0.28$. If $L_m f_{G^s_i}(x_{i,j}) > t$, the model claims the $j'th$ unit grid in G^s_i contains a jump.
The figure shows some detection results of 2CNNs and CNNPoly. They both show good performances on detecting jump curves for testing function.

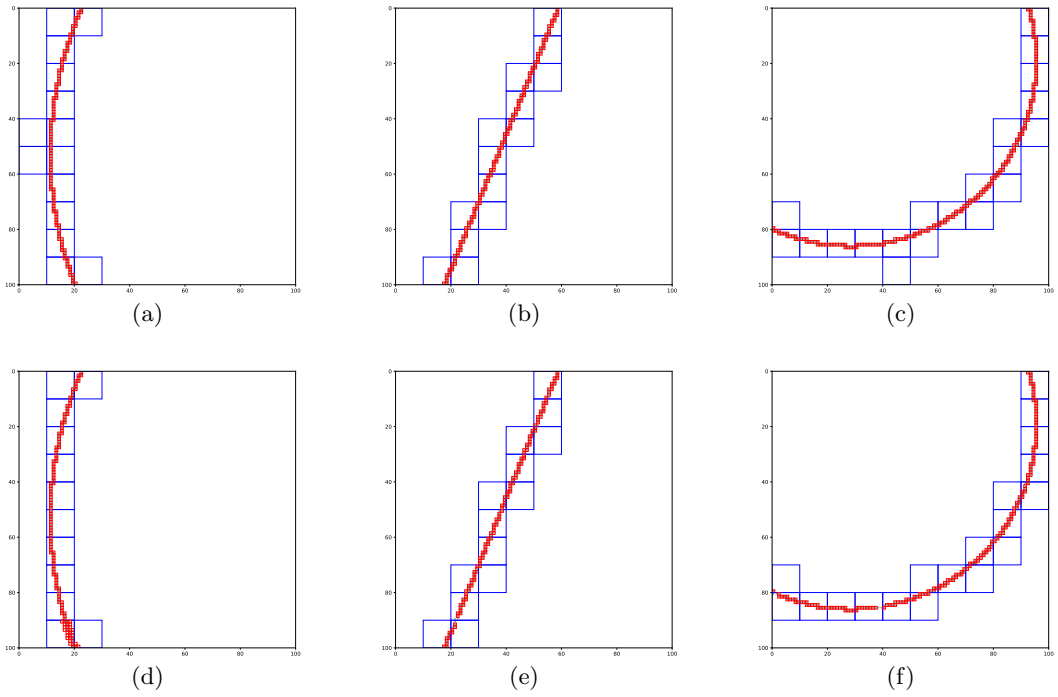


Figure 13: Compare the results between CNN+CNN method (a) (b) (c), and CNN+polynomial annihilation method (d) (e) (f). In the first stage, if the model claims a small grid G^s_i to contain jumps, stroke the outside frame with blue. For the second stage, if the model predicts a unit grid contains a jump, stroke the edges of the outside frame with red.

4.3 Model robustness on new shape of jump curve

To test the model robustness for functions with new shaped curve, we generate data with jump curve being a triangle. Set $t_1 = 0.28, t_2 = 0.28$ for CNNPoly, $t_1 = 0.28, t_2 = 0.22$ for 2CNNs.

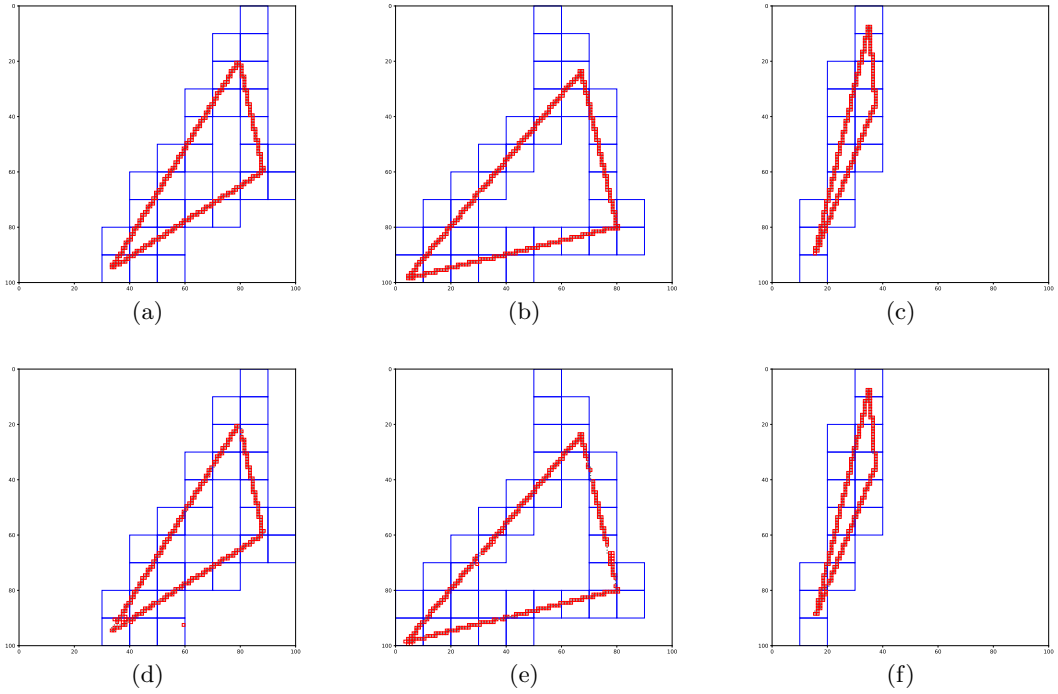
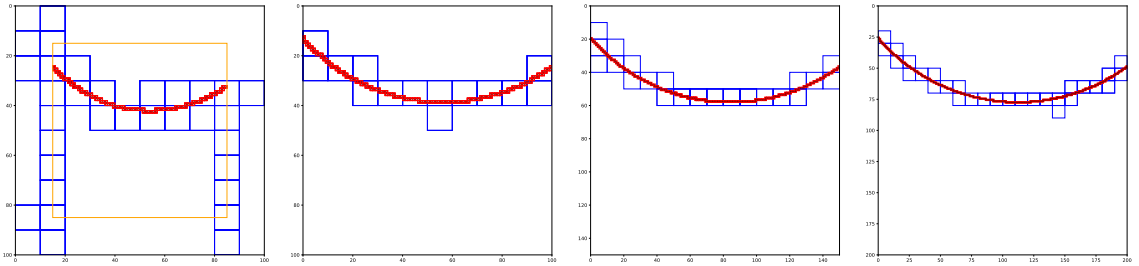


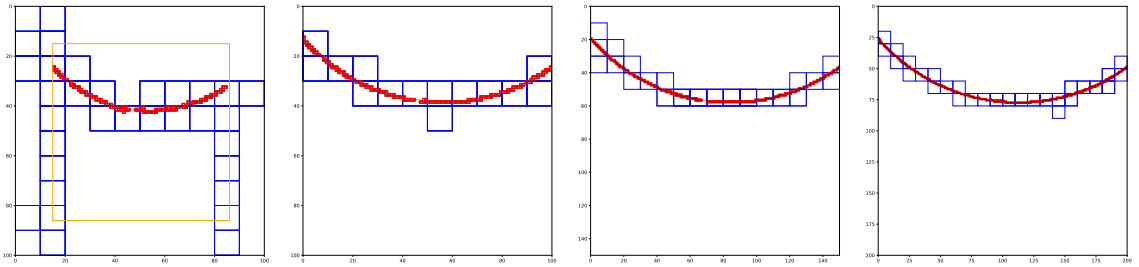
Figure 14: Apply both detection methods to a new type of testing data with jump curve being a triangle. (a) (b) (c) are by 2CNNs, (d) (e) (f) are predictions by CNNPoly, Generally both methods can detect the curves.

4.4 General function detection

We also test functions with different domains and grids by both 2CNNs and CNNPoly method. We generate a set of testing data from the same function by picking different number of points. Set $d_1 = d_2 = 90$, $t_1 = 0.4, t_2 = 0.12$ for CNNPoly, and $d_1 = d_2 = 90$, $t_1 = 0.4, t_2 = 0.2$ for 2CNNs. The result shows when we have enough resolution, the detection result can be very accurate.



(a) 2CNN,length=71 (b) 2CNN,length=101 (c) 2CNN,length=151 (d) 2CNN,length=201



(e) CNNPoly, length=71 (f) CNNPoly, length=101 (g) CNNPoly, length=151 (h) CNNPoly, length=201

Figure 15: function on $[0, 1]^2$, with grid size 71×71,101×101,151×151, 201×201.The blue boxes are the predictions at the first stage, the red boxes are the predictions at the second stage. Particularly , the yellow box in the first subplot is the original range of input.

4.5 Comparison on CNNPoly and 2CNNs methods

According to experiments carried out above, both methods can solve the detection problem. From the perspective of computational cost, CNNPoly runs much slower than 2CNNs and also has much more computational complexity. Besides, CNNPoly are more likely to produce oscillations when the jump is obvious. However, 2CNNs is not as robust as CNNPoly since it usually brings false predictions. However, a guess is that when we have enough data, this problem in CNN can be solved. Generally speaking, using 2CNNs might be a good way if don't need very accurate prediction results.

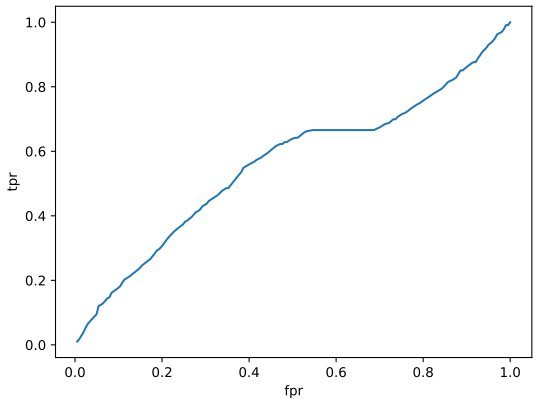


Figure 16: ROC curve on testing data by a trained NN. Therefore this model cannot solve the detection problem.

5 Experiment details

5.1 An experiment using NN instead of CNN

Someone may ask if deep neural network can also solve this problem. Theoretically a neural network should also achieve same outputs, since CNN can be regarded as a special NN with some weights set to be 0. However, in real implementation, there is a problem about local minimum in optimization procedure. We carry out an experiment using NN instead of CNN on the one dimensional detection problem, the architecture is described in appendix B.2.

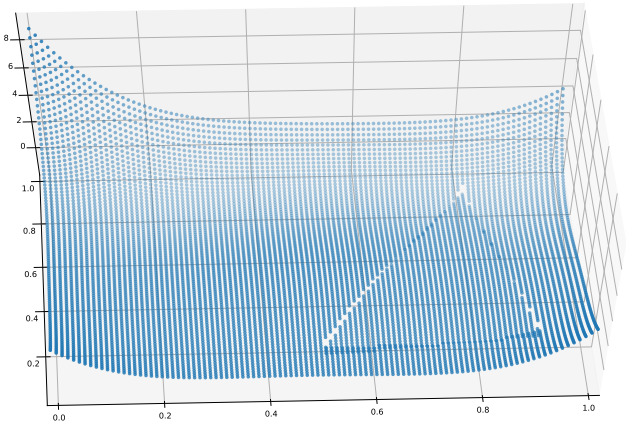
After training with 10000 data for 500 epoch, the loss become stable at around 0.0737, which is approximately the value when all predictions on the training data set are 0, no positive results. Also based on ROC curve, it seems to not work.

5.2 Boundary detection problem

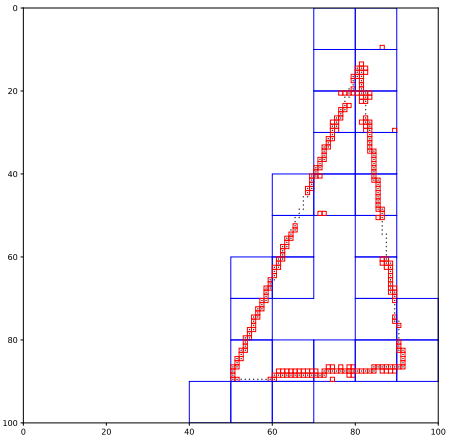
Kernel size usually affects CNN effectiveness. In this specific detection problem, we found that the kernel size might influence the boundary jump detection. When considering the kernel size in model, we found a phenomenon that when we use kernel size $k = 6$, we cannot detect a jump if the jump is near boundary. When we change $k = 3$, most of jumps near boundary can be detected. When we change $k = 2$, all jumps in testing functions can be detected by adjusting threshold. This might because the detection procedure need enough local information of both size of jumps.

5.3 Function closeness and flipping

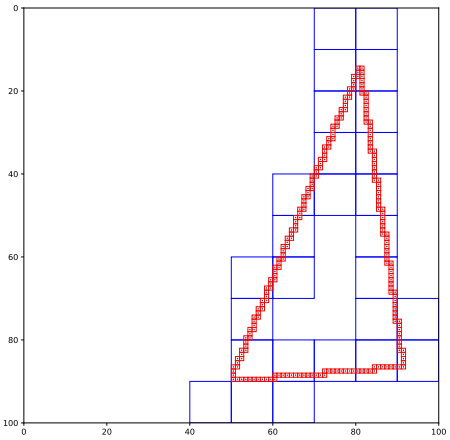
Sometimes jump can be very small because two functions are flipping or two functions are too close to each other. In detection, sometimes we cannot detect those jumps with either CNNPoly, 2CNNs or Polynomial annihilation unless we set a very small threshold, which would cause larger mis-detection problem.



(a)



(b)



(c)

Figure 17: (a) is the 3D plot of a function with functions closed to each other somewhere, (b) is CNNPoly prediction with two thresholds=0.28,0.28, (c) is 2CNNs prediction with two thresholds=0.28,0.22. In the neighborhood of location that two functions are closed to each other, both approaches have difficulty to detect jumps, while CNNPoly is more likely to be influenced.

5.4 threshold chosen

During prediction procedure, we need a threshold in each model to determine an output. We use a small set of data to estimate optimal thresholds. Firstly we set a goal function $F = |\frac{N_{pred}}{N_{total}} - 1|$, where N_{pred} is the number of predicted jumps in all data, and N_{total} is the number of total jumps in data. we want to find a threshold within $[0, 1]$ so that F is minimum. When $N_{pred} = N_{total}$, the value reach 0, which is the minimum value in parameter space.

For one dimensional detection model and 2CNNs model, we use 10000 new data for calculation. For CNNPoly method in two dimensional data, only 1000 data used, since it took too long to calculate over the whole data set.

Under those conditions, the optimal threshold for one-dimensional problem is 0.97, for the first step model of two-dimensional problem is 0.28, the threshold value for the second step in CNNPoly is 0.22, and for the second step in 2CNNs is 0.28.

However, the threshold should still be adjusted based on real cases. From last discussions, there is phenomenon especially appearing in polynomial annihilation, when the jump is small, the threshold should also be relatively small, which also occurs in CNN model. Therefore, users should choose threshold based on type I error and type II error. However, this problem might be solved after enough training.

6 Extended problems

6.1 Jumps in first derivative detection

Sometimes people also care about discontinuity in derivatives of functions. Previously we mentioned that polynomial annihilation can detect jumps in any derivatives. Method involving CNN can also be applied in those problems. Moreover, we can distinguish jumps in different derivatives when they appear at the same time in a function. Typically, we focus on the problem with jumps and jumps in the first derivative (also known as continuous not differentiable) detection.

A function is differentiable at x in one dimensional case if $lim_{h\to 0}\frac{f(x+h)-f(x)}{h}$ exists, otherwise it's none differentiable. If the function at x is continuous but not differentiable, it is a jump in the first derivative.

6.1.1 Detection approach

Now we redefine discontinuity and continuous not differentiable point set as $\mathcal{E} := \{e_i, i = 1, ...l_f + m_f\} = \{C_1, ...C_{l_f}, \xi_1, ..., \xi_{m_f}\}$,$l_f + m_f \leq M$.

If it contains a continuous but non-diferentiable point, assign it to label t_2; if it contains a jump, then we assign the interval with label t_3; otherwise assign it with label t_1. Usually t_1, t_2 and t_3 are be different values. Therefore we can construct a ground truth vector with value of each element being the label of its corresponding unit interval. Other set up are the same as one dimensional jump detection.

$$y_f = (y_{(1)}, y_{(2)}, ..., y_{(K-1)}) \tag{25}$$

$$y_{(i)} = \begin{cases} t_1, \ \textit{contains only smooth function.} \\ t_2, \ x_i \leq C_k < x_{i+1}, \forall 1 \leq k \leq l_f \\ t_3, \ x_i \leq \xi_k < x_{i+1}, \forall 1 \leq k \leq m_f \end{cases} \tag{26}$$

6.1.2 Simulation results

In simulation procedure, generate 1000,000 training data on domain $[-1, 1]$ with $K = 202$, $M = 3$ and $t_1 = 0, t_2 = 1, t_3 = 2$. The probability p that a non-derivative point is a jump is set to be 0.75, thus the probability that a non-derivative point is a jump is 0.25.

According to the data generation approach in appendix A.2, $min|e_i - e_{i+1}| \geq \frac{1}{25}$ for $\forall i > 1, ...l_f + m_f - 1$. Therefore for each unit interval $[x_i, x_{i+1})$, it would at most contain one discontinuity or one continous not differentiable point.

For prediction, we set two thresholds t_1 and t_2. If the $i'th$ element in the output $\mathcal{N}(v_{f_t})_i \leq t_1$, $[x_i, x_{i+1})$ only contains a smooth piece of function; if $t_1 < \mathcal{N}(v_{f_t})_i \leq t_2$, we claim that the interval contains a continuous not differentiable point. If $\mathcal{N}(v_{f_t})_i > t_2$, the model indicates the interval contains a jump.

For testing data, e set the probability of being a continuous non-differentiable point other than a discontinuity $p = 0.8$, which means a non-differentiable point has probability 0.8 to be a jump in first derivative and 0.2 to be a jump for f. The testing result shows that changes in p have no effect on prediction result.
Certainly the ground truth values are not necessary to be 0,1,2. Therefore we also try different labeling ways for training and testing. We train all of those models on the same 1000,000 training data and test on 10000 testing data. According to ROC curve, labeling strategy are relatively flexible. However, it's might be better to assign values in a reasonable way: Since the property of continuous not differentiable points are closer to smoothness, while the property of jump can be very different, the label assigned may be better follow a order of smoothness, jump in first derivative and jump for more effective training.

6.2 Generalization to high dimensional problems

The detection method involving CNN can be extended to high dimensional space. Since there are exponential increases in computation costs, we prefer approaches that balance

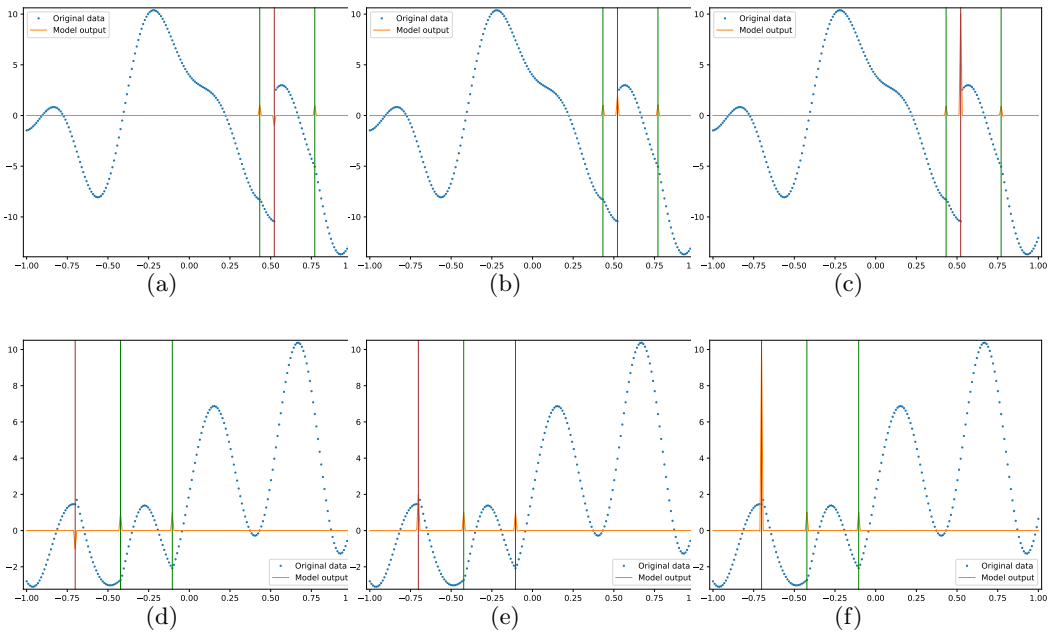


Figure 18: Results on testing functions. a and d are predicted by model trained with label "0,-1,1"; b and e are predicted by model trained with label "0,1,2", c and f are predicted by model trained with label "0,1,10". The function f_t is consisting of blue discrete points, the orange curve is the output $\mathcal{N}(v_{f_t})$. The green vertical lines are across the exact left boundary point of the unit intervals that actually contain discontinuity, and the red vertical lines are across the left boundary point of the unit intervals which contain continuous not differentiable points.

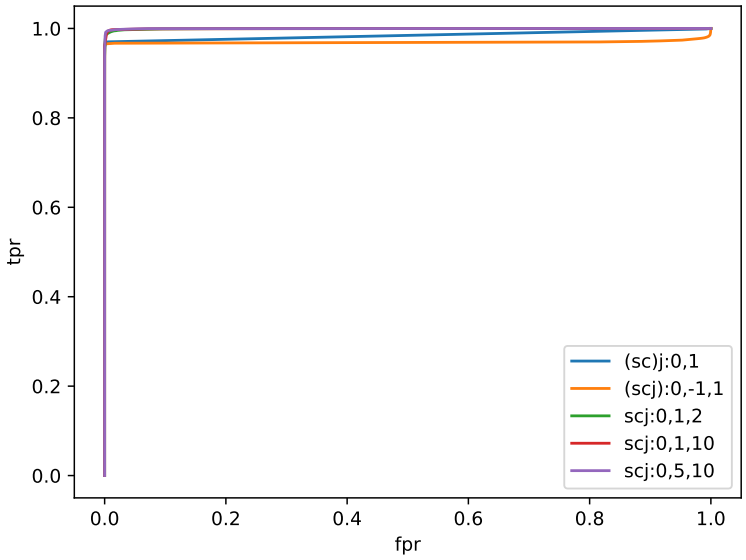
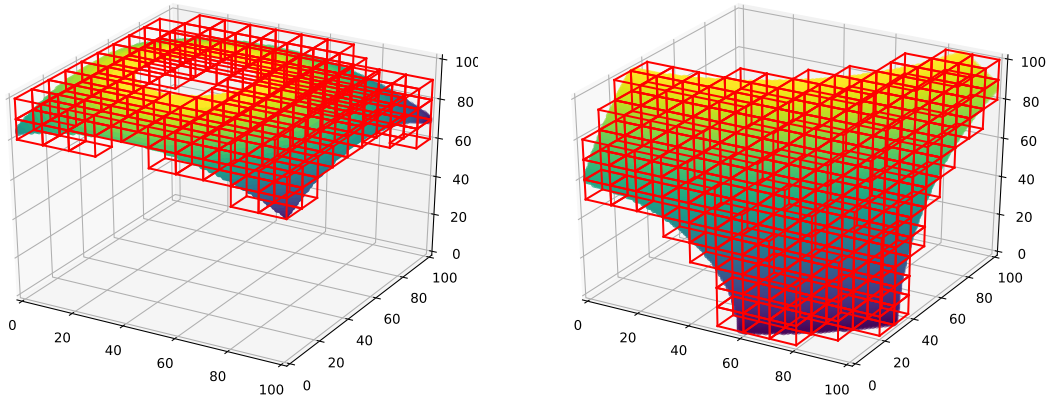


Figure 19: The label "scj: a,b,c" means assign label a to the unit interval with only smooth piece of function; assign b to the interval with continuous not differentiable points; assign c to interval with jumps. Especially "(sc)j:a,b" means assign label to the unit interval with only smooth piece of function and interval with continuous not differentiable with the same label a. From the ROC curve we can see with enough training, if the values assigned follow an order described in the above paragraph, then the ROCs almost overlapped and have the largest area under the curve. But in general, it seems the effect of labeling is small.

between accuracy and computation. From two dimensional case, we find both CNNPoly and 2CNNs have good performances on detection as well as relative low computation costs. Therefore it still might be a good choice to apply those two approaches to high dimension.
For n-dimensional problem ($n \geq 2$), if we pick uniform point number to be $(K_1, K_2, ...K_n)$ for each dimension, the unit grid number is $N = (K_1 - 1) \times (K_2 - 1)... \times (K_n - 1)$ on $[-1, 1]^n$. (The detection approach is the same for detection on other domains.) Firstly we determine a suitable Hyper-rectangle size to be $(L_1, L_2, ...L_n)$ with unit in number of cells, where $K_i - 1$ can be divided by L_i, therefore we divide the grid G into smaller grids and can repeat the 2CNNs and CNNPoly approaches.

As an example, we train a CNN model to detect jumps in each small cubic, which is the first step of both detection methods. The architecture of model is described in appendix B.4.
After detecting the larger grid with size $10 \times 10 \times 10$, we choose threshold $t = 0.3$. For



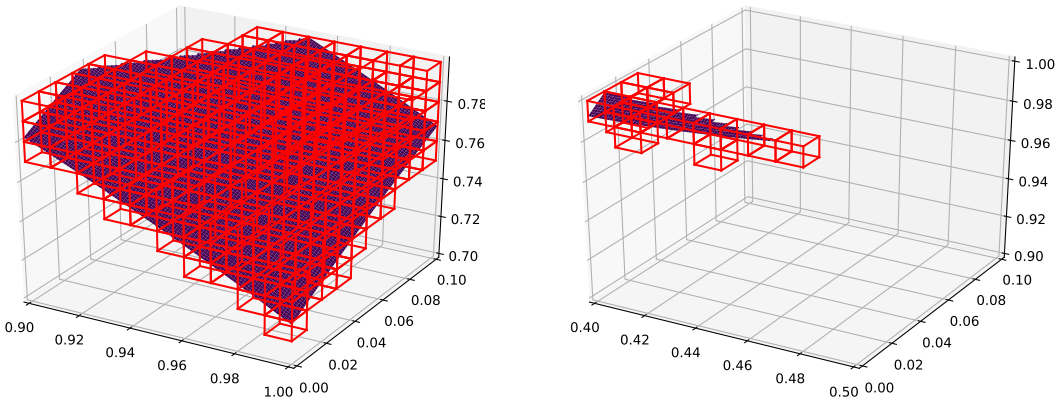
(a) (b)

Figure 20: Prediction results of a three-dimensional function. The red boxes indicate that there are jumps inside.

those predicted score greater than 0.3, we apply polynomial annihilation to the larger grid.

6.3 training with height

In one-dimensional data, annihilation polynomial method can indicate how large the jump is, as well as the jump direction. With a little modification on labeling, CNN models can also achieve the same effect.



(a) (b)

Figure 21: Prediction results by polynomial annihilation on a larger grid predicted by first step using CNN. The red boxes are unit grid which indicates that there are jumps inside.

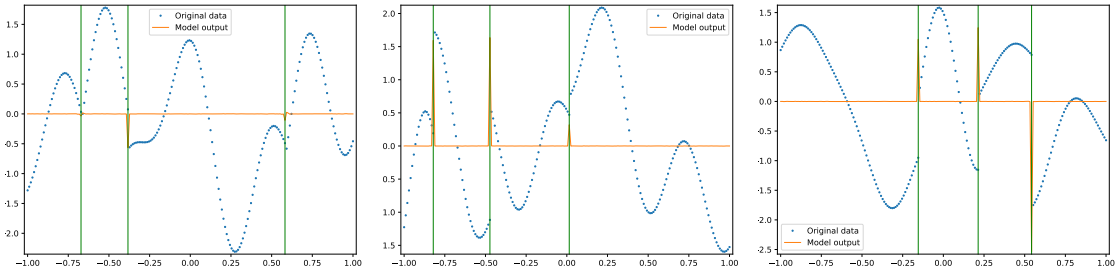
Instead of labeling the ground truth using indicator function, we modify it using the real height at the point. the jump height $h(x)$ at point x is defined as

$$h(x) = f_+(x) - f_-(x) \tag{27}$$

The ground truth $\{y_{(i)}\}$ then is defined as

$$y_{(i)} = max(h(x), \forall x \in [x_i, x_{i+1})) \tag{28}$$

According to the labeling method, we should avoid 2 jumps in the same unit grid in this case. Since during training and prediction, we first standardize each function, therefore



(a) (b) (c)

Figure 22: Prediction results of CNN model with height labels. The model can predict both height and direction of jumps.

the outcome only provides a standardized predicted height. However we can recover the

predicted height to original scale by the following equation:

$$h_{ori}(x) = h_{scale}(x) \times \sigma_{v_f} + \mu_{v_f} \tag{29}$$

We calculate the mean squared error on 10,000 scaled data and there original data. We only consider the points that is really a jump point. The scaled error is 1.33e-05, and the original error is 1.32e-05. Both values are very small, which implies the model has very accurate predictions on heights of jumps.

7 Conclusion and discussion

In this paper we come up with methods for jump detection particularly in one-dimensional and two-dimensional space, and also illustrates that the approaches involving CNN can be also applied to high-dimensional space. In one dimensional case, we also apply CNN to detect jump in first derivatives. Moreover, the combination of CNN and polynomial annihilation detection can balance between computational costs and accuracy. All the experiment results shows that jump detection involving CNN can get good performances.

8 Reference

References

[1] Zhang, Guannan, et al. "Hyperspherical sparse approximation techniques for high-dimensional discontinuity detection." SIAM review 58.3 (2016): 517-551.

[2] Suresh, V., et al. "Denoising and detecting discontinuities using wavelets." Indian Journal of Science and Technology 9 (2016): 19.

[3] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521, no. 7553 (2015): 436.

[4] Burkardt, John, Clayton Webster, and Guannan Zhang. "A Hyperspherical Method for Discontinuity Location in Uncertainty Quantification." (2013).

[5] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In Advances in neural information processing systems, pp. 1097-1105. 2012.

[6] Jakeman, John D., Richard Archibald, and Dongbin Xiu. "Characterization of discontinuities in high-dimensional stochastic problems on adaptive sparse grids." Journal of Computational Physics 230, no. 10 (2011): 3977-3997.

[7] Archibald, Rick, Anne Gelb, and Jungho Yoon. "Determining the locations and discontinuities in the derivatives of functions." Applied Numerical Mathematics 58.5 (2008): 577-592.

[8] Esposito, Joel M., and Vijay Kumar. "A state event detection algorithm for numerically simulating hybrid systems with model singularities." ACM Transactions on Modeling and Computer Simulation (TOMACS) 17, no. 1 (2007): 1.

[9] Wei M., De Pierro A. R.,Yin J., Iterative methods based on polynomial interpolation lters to detect discontinuities and recover point values from Fourier data, IEE Trans. on Signal Process. 53, no. 1 (2005) 136-146.

[10] Archibald, Rick, Anne Gelb, and Jungho Yoon. "Polynomial fitting for edge detection in irregularly sampled signals and images." SIAM journal on numerical analysis 43, no. 1 (2005): 259-279.

[11] Allasia G., Besenghi R., De Rossi A., A scattered data approximation scheme for the detection of fault lines. In Mathematical methods for curves and surfaces (Oslo, 2000), 2534, Innov. Appl. Math., Vanderbilt Univ. Press, Nashville, TN, 2001.

[12] Gutzmer T., and A. Iske, Detection of discontinuities in scattered data approximation. Numerical Algorithms 16 (1997) 155-170.

[13] Jain, Ramesh, Rangachar Kasturi, and Brian G. Schunck. Machine vision. Vol. 5. New York: McGraw-Hill, 1995.

[14] Bozzini M., F. De Tisi, M. Rossini:Irregularity detection from noisy data with wavelets. In: Wavelets, Images, and Surface Fitting, pp. 5966, Wellesley MA: A. K. Peters 1994.

[15] Lee D., Coping with Discontinuities in Computer Vision: Their Detection, Classication and Measurement IEEE Transaction on Pattern Analysis and Machine Intelligence, 12 (1990) 321-344.

[16] LeCun, Yann, et al. "Backpropagation applied to handwritten zip code recognition." Neural computation 1.4 (1989): 541-551.

[17] Canny, John. "A computational approach to edge detection." In Readings in computer vision, pp. 184-203. Morgan Kaufmann, 1987.

Appendix A Data generation

A.1 One-dimensional data with jumps

To generate functions containing jumps on domain $\Omega = [\omega_0, \omega_1]$, set $m_f \leq M$ as the maximum number of jumps in a function f, where M is a pre-defined largest number of jumps allowed in a function. N points are chosen uniformly on function f. Set $\xi_0 = x_1$, $\xi_{m_f+1} = x_K$.
Firstly, define the jump set:

$$J = \{\xi_k | 1 \leq k \leq m_f, \xi_k \in (x_1, x_K)\} \tag{30}$$

For $\forall k' > k, 1 \leq k' \leq m_f$, we have $\xi_{k'} > \xi_k$.
It should be noted that in this generating process, the jumps are allowed fall into the same unit interval.
A piece-wise smooth function $f(x)$ is defined as:

$$f(x) = \begin{cases} f_i(x), & if \quad x \in [\xi_{i-1}, \xi_i), \quad 1 \leq i \leq m_f + 1 \\ f_{m+1}(x), & if \quad x = \xi_{m_f+1} \end{cases} \tag{31}$$

Where $f_i(x)$ is generated by Fourier expansion:

$$f_i(x) = a_0 + \sum_{n=1}^{15} (a_n cosnx + b_n sinnx) \tag{32}$$

In the above equation, $a_0 \in [0, 1]$ is generated from uniform distribution $Unif(0, 1)$, and every element in $\{a_n\}$, $\{b_n\}$ follows a normal distribution $N(0, 1)$ and is independent from each other.
To generate a function f, m_f is determined by discrete uniform distribution on $\{0, 1, \cdots, M\}$, thus the probabilities that the function contains $0, 1, \cdots, M$ jumps are set to be equivalent to $\frac{1}{M+1}$. Next, determine each elements in the location set of jumps $\{\xi_1, \cdots, \xi_{m_f}\}$ by uniform distribution on Ω. At last, use the above equations to generate the piece-wise functions. The K points are the final input for function f.
According to the necessary and sufficient condition for Taylor expansion, a function should have infinite number of continuous derivatives at the expanded point. Therefore it's impossible to produce infinite discontinuity. Besides, we set random positive numbers to extend the gap between two pieces of smooth functions to prevent producing continuous not differentiable points.
In 20,000 training data, we set $M = 3$, $\Omega = [-1, 1]$, $K = 202$.

A.2 One-dimensional data with jumps and discontinuities in first derivative

To create a discontinuity in first derivative at x of function f, we first treat x is a jump location $x = \xi_k \in J$. Therefore we can construct a function $f_0(x)$ exactly by the generation method for jumps. Next, we modify function values at the right side of x, which is the piece of smooth function after point x:

$$f(x) = \begin{cases} f_0(x), & \omega_0 \leq x < \xi_k \\ f_0(x) + f_{k-1}(\xi_k) - f_k(\xi_k), & \xi_k \leq x \leq \omega_1 \end{cases} \tag{33}$$

Therefore, to generate a function with jumps and discontinuities in first derivative, first we generate a function with jumps only, then set a probability p, so that each jump has probability p to be changed into a discontinuity in first derivative.
For training data, set $M = 3$, $\Omega = [-1, 1]$ and $K = 202$.
To prevent jump and discontinuities in first derivative fall into the same unit grid, which would cause the labeling problem, we add a constrain that for the set of all jumps and

discontinuities in first derivative $\mathcal{E} := \{e_i, i = 1, ...l_f + m_f\} = \{C_1, ...C_{l_f}, \xi_1, ..., \xi_{m_f}\}$, $min|e_i - e_{i+1}| \geq \frac{1}{25}$ for $\forall i > 1, ...l_f + m_f - 1$. Therefore for each unit interval $[x_i, x_{i+1})$, it would at most contain one discontinuity or discontinuity in first derivative.

A.3 Two-dimensional data with jumps

In two-dimensional space, let the grid G contain $K_1 \times K_2$ uniform points, therefore $N = (K_1 - 1) \times (K_2 - 1)$. In this case we only consider jump curves between two functions f_1 and f_2. f_1, f_2 are combination of Legendre Polynomial: $P_n(x)$, $x \in S$, which can be defined by contour integral:

$$P_n(z) = \frac{1}{2\pi i} \int (1 - 2tz + t^2)^{-1/2} t^{-n-1} dt \tag{34}$$

$$f_1, f_2 = \sum_{m+n \leq norder} a_{(m,n)} P_m(x) P_n(y) \tag{35}$$

where norder is highest order of polynomial, here we set norder=10. and $a_{(m,n)}$ follows distribution $N(0, 10)$.
To split Ω into two parts and map f_1, f_2 on them, two splitting methods are used. One is using a line as split line, the other is using a circle as split line.
1. Function generated by line cut.

$$f(x) = \begin{cases} f_1(2x - 1, 2y - 1), & if \quad y \leq ax + b \\ f_2(2x - 1, 2y - 1), & if \quad y > ax + b \end{cases} \tag{36}$$

For training data we set $\Omega = [0, 1]^2$, the jump curve is $\{(x, y)|y = ax + bx \in [0, 1],\ y \in [0, 1]\}$
2. Function generated by circle cut.

$$f(x) = \begin{cases} f_1(2x - 1, 2y - 1), & if \quad (x - \theta_x)^2 + (y - \theta_y)^2 \leq r^2 \\ f_2(2x - 1, 2y - 1), & if \quad (x - \theta_x)^2 + (y - \theta_y)^2 > r^2 \end{cases} \tag{37}$$

For training data we set $\Omega = [0, 1]^2$, the jump curve is $\{(x, y)|(x - \theta_x)^2 + (y - \theta_y)^2 = r^2, \theta_x \in [-2, 2], \theta_y \in [-2, 2], r \in [0.5, 1.5]\}$
For training data, we set $K_1 = K_2 = 101$. The inputs are only values of vertices of the grid.

A.4 Three-dimensional data with jumps

Define grid G on domain Ω, with each dimension containing K_1, K_2, K_3 points. Similar to two-dimensional data generation, we only consider a jump curve between two functions f_1 and f_2, which is identified by a ball surface:

$$f_1, f_2 = \sum_{m+n+k \leq norder} a_{(m,n)} P_m(x) P_n(y) P_k(z) \tag{38}$$

where norder is highest order of polynomial, here we use 3. and $a_{(m,n)}$ follows distribution $N(0, 10)$.
When $\Omega = [0, 1]^2$, the jump curve is $\{(x, y)|(x - \theta_x)^2 + (y - \theta_y)^2 + (z - \theta_z)^2 = r^2, \theta_x \in [-2, 2], \theta_y \in [-2, 2], \theta_z \in [-2, 2], r \in [0.5, 1.5]\}$
For training data, set K_1, K_2, K_3=101.

Appendix B Architecture of CNN Models

B.1 Architecture of one-dimensional CNN model

The architecture of models to detect jumps and discontinuities in first derivative in one-dimensional space is as the following:

Layer	input size	kernel size	num of kernel	stride	activation function	output size
conv1	202	2×1	24	1	ReLu	201×24
conv2	201×24	2×24	24	1×24	ReLu	200×24
conv3	200×24	2×24	24	1×24	ReLu	199×24
conv4	199×24	2×24	24	1×24	ReLu	198×24
conv5	198×24	2×24	24	1×24	ReLu	99×24
dense						201

During training process, set an dropout layer before dense layer with probability 0.2.

B.2 Architecture of one-dimensional NN model

Layer	activation function	output size
dense1	ReLu	300
dense2	ReLu	200
dense3	ReLu	100
dense4	ReLu	100
dense5	ReLu	100
dense6		201

During training process, set an dropout layer before dense layer with probability 0.2.

B.3 Architecture of two-dimensional CNN model

B.3.1 one steps model

Models with different output size have the same architecture.

Layer	input size	kernel/pooling size	num of kernel	stride	activation function	output size
conv1	101×101	4×4	32	1×1	ReLu	98×98×32
maxpooling1	98×98×32	2×2		2×2		49×49×32
conv2	49×49×32	2×2×32	32	1×1×32	ReLu	48×48×32
conv3	48×48×32	2×2×32	32	1×1×32	ReLu	47×47×32
conv4	47×47×32	2×2×32	32	1×1×32	ReLu	46×46×32
dense						N

Where $N = 100, 625, 2500, 10000$.

B.3.2 First CNN model in two-step detection

Layer	input size	kernel/pooling size	num of kernel	stride	activation function	output size
conv1	101×101	4×4	32	1×1	ReLu	98×98×32
maxpooling1	98×98×32	2×2		2×2		49×49×32
conv2	49×49×32	2×2×32	32	1×1×32	ReLu	48×48×32
conv3	48×48×32	2×2×32	32	1×1×32	ReLu	47×47×32
conv4	47×47×32	2×2×32	32	1×1×32	ReLu	46×46×32
dense				100		

B.3.3 Second CNN model in two-step detection

Layer	input size	kernel size	num of kernel	stride	activation function	output size
conv1	100×100	2×2	32	1×1	ReLu	10×10×32
conv2	10×10×32	2×2×32	32	1×1×32	ReLu	9×9×32
conv3	9×9×32	2×2×32	32	1×1×32	ReLu	8×8×32
dense						100

During training process, set an dropout layer before dense layer with probability 0.1. The training data are all non-overlapped $10 * 10$ grids of 64000 training functions.

B.4 Architecture of three-dimensional CNN model

Layer	input size	kernel/pooling size	num of kernel	stride	activation function	output size
conv1	101×101×101	5×5×5	12	5×5×5	ReLu	20×20×20×12
maxpooling1	20×20×20×12	2×2		2×2		10×10×10×12
conv2	10×10×10×12	2×2×2×12	6	2×2×2	ReLu	5×5×5×6
dense						1000

During training process, set an dropout layer before dense layer with probability 0.2.