Sparse arrays of signatures for online character recognition

Benjamin Graham
Dept of Statistics, University of Warwick, CV4 7AL, UK
b.graham@warwick.ac.uk

December 8, 2013

Abstract

In mathematics the *signature* of a path is a collection of iterated integrals, commonly used for solving differential equations. We show that the path signature, used as a set of features for consumption by a convolutional neural network (CNN), improves the accuracy of *online* character recognition—that is the task of reading characters represented as a collection of paths. Using datasets of letters, numbers, Assamese and Chinese characters, we show that the first, second, and even the third iterated integrals contain useful information for consumption by a CNN.

On the CASIA-OLHWDB1.1 3755 Chinese character dataset, our approach gave a test error of 3.58%, compared with 5.61%[4] for a traditional CNN. A CNN trained on the CASIA-OLHWDB1.0-1.2 datasets won the ICDAR2013 Online Isolated Chinese Character recognition competition.

Computationally, we have developed a sparse CNN implementation that make it practical to train CNNs with many layers of max-pooling. Extending the MNIST dataset by translations, our sparse CNN gets a test error of 0.31%.

Keywords: online character recognition, signature, iterated integrals, convolutional neural network

Condensed running title: Sparse arrays of signatures

1 Introduction

Two rather different techniques work well for online Chinese character recognition. One approach is to render the strokes into a 40×40 bitmap embedded

in a 48×48 grid, and then to use a deep convolutional neural network (CNN) as a classifier [4]. Another is to draw the character on an 8×8 grid, and then in each square calculate a *histogram* measuring the amount of movement in each of the 8 compass directions, producing a 512-dimensional vector to classify [2].

Intuitively, the first representation records more accurately *where* the pen went, while the second is better at recording the *direction* the pen was taking. We attempt to get the best of both worlds by producing an enhanced picture of the character using the path *iterated-integral signature*. This value-added picture of the character records the pen's location, direction and the forces that were acting on the pen as it moved.

CNNs start with an input layer of size $N \times N \times M$. The first two dimensions are spatial; the third dimension is simply a list of features available at each point; for example, $M = 1$ for grayscale images and $M = 3$ for color images. When calculating the path signature, we have a choice of how many iterated integrals to calculate. If we calculate the zeroth, first, second, ..., up to the m-th iterated integrals, then the resulting input vectors are $M = 1 + 2 + 2^2 + \cdots + 2^m$ dimensional.

This representation is *sparse*. We only calculate path signatures where the pen actually went: for the majority of the $N \times N$ spatial locations, the M-dimensional input vector is simply taken as all-zeros. Taking advantage of this sparsity makes it practical to train much larger networks than would be practical with a traditional CNN implementation.

2 Sparse CNNs: DeepCNet(l, k)

Inspired by [4], we have considered a simple family of CNNs with alternating convolutional and max-pooling layers. Let DeepCNet(l, k) denote the CNN with

- an input layer of size $N \times N \times M$ where $N = 3 \cdot 2^l$,
- k convolutional filters of size 3×3 in the first layer,
- nk convolutional filters of size 2×2 in layers $n = 2, \ldots, l$
- a layer of 2×2 max-pooling after each convolution layer, and
- a fully-connected final hidden layer of size $(l + 1)k$.

For example, DeepCNet$(4, 100)$ is the architecture from [4] with input layer size $N = 48 = 3 \cdot 2^4$ and four layers of max-pooling:

input-100C3-MP2-200C2-MP2-300C2-MP2-400C2-MP2-500N-output

For general input the cost of the forward operation, in particular calculating the first few hidden layers, is very high. For sparse input, the cost of calculating the lower hidden layers is much reduced, and evaluating the upper layers becomes the computational bottleneck.

When designing a CNN, it is important that the input field size N is strictly larger than the objects to be recognized; CNNs do a better job distinguishing features in the center of the input field. However, padding the input in this way is normally expensive. An interesting side effect of using sparsity is that the cost of padding the input disappears.

2.1 Character scale $n \approx N/3$

For character recognition, we choose a scale n on which to draw the characters. For the Latin alphabet and Arabic numerals, one might copy MNIST and take $n = 20$. Chinese characters have a much higher level of detail: [4] uses $n = 40$, constrained by the computational complexity of evaluating dense CNNs.

Given n, we must choose the l-parameter such that the characters fit comfortably into the input layer. DeepCNets seem to work best when n is approximately $N/3$. There are a couple of ways of justifying the $n \approx N/3$ rule:

- To process the $n \times n$ sized input down to a zero-dimensional quantity, the number of levels of 2×2 max-pooling $l = \log_2(N/3)$ should be approximately $\log_2 n$.
- Counting the number of paths through the CNN from the input to output layers reveals a plateau; see Figure 1. Each corner of the input layer has only one route to the output layer; in contrast, the central $(N/3) \times (N/3)$ points in the input layer each have $9 \cdot 4^{l-1}$ such paths.

2.2 Sparsity

For DeepCNets with $l \geqslant 4$, training the network is in general hard work, even using a GPU. However, for character recognition, we can speed things up by taking advantage of the sparse nature of the input, and the repetitive nature of CNN calculations. Essentially, we are memoizing the filtering and pooling operations.

First imagine putting an all-zero array into the input layer. As you evaluate the CNN in the forward direction, the translational invariance of the input is propagated to each of the hidden layers in turn. We can therefore

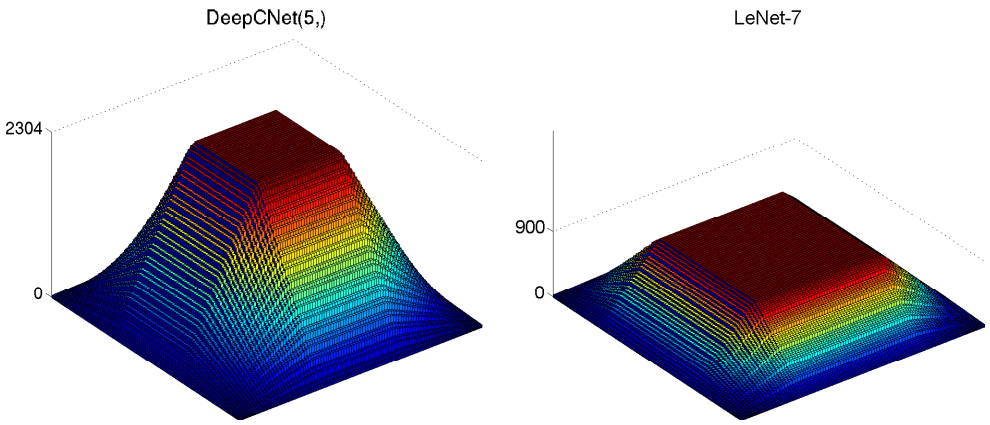

Figure 1: A comparison of the number of possible paths from the input layer (96×96) to the fully connected layer for $l = 5$ DeepCNets and LeNet-7 [8]. The DeepCNet's larger number of hidden layers translates into a larger number of paths, but with a narrower plateau.

think of each hidden variable as having a *ground state* corresponding to receiving no meaningful input; the ground state is generally non-zero because of bias terms. When the input array is sparse, one only has to calculate the values of the hidden variables where they differ from their ground state.

To forward propagate the network we calculate two types of list: lists of the non-ground-state vectors (which have size M, k, $2k$, ...) and lists specifying where the vectors fit spatially in the CNN. This representation is very loosely biologically inspired. The human visual cortex separates into two streams of information: the dorsal (*where*) and ventral (*what*) streams.

2.3 MNIST as a sparse dataset

To test the sparse CNN implementation we used MNIST [7]. The 28×28 digits have on average 150 non-zero pixels. Placing the digits in the middle of a 96×96 grid produces a sparse dataset as 150 is much smaller than 96^2.

It is common to extend the MNIST training set by translations and/or elastic distortions. Here we only use translations of the training set, adding random shifts of up to ± 2 pixels in the x- and y-directions. Training a very small-but-deep network, DeepCNet$(5, 10)$, for a very long time, 1000 repetitions of the training set, gave a test error of 0.58%. Using a GeForce GTX 680 graphics card, we can classify 3000 characters per second.

We tried increasing the number of hidden units. Training DeepCNet$(5, 30)$ for 200 repetitions of the training set gave a test error of 0.46%.

Dropout, in combination with increasing the number of hidden units and the training time generally improves ANN performance [6]. DeepCNet$(5, 60)$ has seven layers of matrix-multiplication. Dropping out 50% of the input to the fourth layer and above during training resulted in a test error of 0.31%.

3 The sparse signature grid representation

The expression of the information contained in a path in the form of iterated integrals was pioneered by K.T. Chen [3]. More recently, path signatures have been used to solve differential equations driven by rough paths [11, 12]. The signature extracts enough information from the path to solve any linear differential equation and uniquely characterizes paths of finite length [5].

The signature has been used in sound compression [10]. A stereo audio recording can be seen as a highly oscillating path in $\mathbb{R}^2$. Storing a truncated version of the path signature allows a version of the audio signal to be reconstructed.

Although computing the signature of a path is easy, the inverse problem is rather more difficult. The limiting factor in [10] was the lack of an efficient algorithm for reconstructing a path from its truncated signature when $m > 2$. We side-step the inverse problem by learning to recognize the signatures directly.

3.1 Computation of the path signature

Let $[S, T] \subset \mathbb{R}$ denote a time interval and let $V = \mathbb{R}^d$ with $d = 2$ denote the writing surface. Consider a pen stroke: a continuous function $X : [S, T] \to V$. For positive integers k and intervals $[s, t] \subset [S, T]$, the k-th iterated integral of X is the d^k-dimensional vector (i.e. a tensor in $V^{\otimes k}$) defined by

$$X^k_{s,t} = \int_{s<u_1<\cdots<u_k<t} 1\, dX_{u_1} \otimes \cdots \otimes dX_{u_k}.$$

By convention, the $k = 0$ iterated integral is simply the number one. The $k = 1$ iterated integral is the displacement of the path. The $k = 2$ iterated integral is related to the curvature of the path.

As k increases, it is a case of diminishing returns. The iterated integrals increase rapidly in dimension whilst carrying less information about the large scale shape of X. We therefore consider truncated signatures. The *signature*, truncated at level m, is the collection of the iterated integrals,

$$S(X)_{s,t} = (1, X^1_{s,t}, X^2_{s,t}, \ldots, X^m_{s,t}).$$

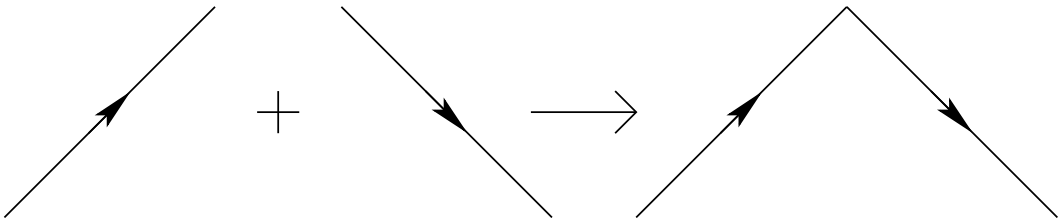

Figure 2: Concatenating paths and the corresponding $m = 2$ signatures.

With $d = 2$, the dimension of this object is

$$M := 1 + d + \cdots + d^m = 2^{m+1} - 1. \tag{1}$$

Let $\Delta_{s,t} := X_t - X_s$ denote the path displacement. Thinking of $\Delta_{s,t}$ as a row vector, the tensor product corresponds to the Kronecker matrix product (`kron` in MATLAB). When X is a straight line, the signature can be calculated exactly:

$$X^1_{s,t} = \Delta_{s,t}, \quad X^2_{s,t} = \frac{\Delta_{s,t} \otimes \Delta_{s,t}}{2!}, \quad X^3_{s,t} = \frac{\Delta_{s,t} \otimes \Delta_{s,t} \otimes \Delta_{s,t}}{3!}, \quad \ldots \tag{2}$$

Joining together two paths to form a longer path results in a type of convolution of the signatures; see Figure 2. With $s < t < u$, $S(X)_{s,u}$ is calculated from $S(X)_{s,t}$ and $S(X)_{t,u}$ using the formula

$$X^k_{s,u} = \sum_{i=0}^{k} X^i_{s,t} \otimes X^{k-i}_{t,u}, \quad k = 0, 1, 2, \ldots, m. \tag{3}$$

Using (3) inductively we can calculate the signature for any piecewise linear path.

3.2 Representing pen strokes

Each character collected by an electronic stylus is represented by a sequence of pen strokes. Each stroke is represented by a sequence of points that we will treat as a piecewise linear path.

Recall that $n \approx N/3$ denotes the scale at which characters will be drawn. We use another parameter $\delta = n/5$ to describe very approximately the scale of path curvature.

Here is an algorithm that takes a character and uses the first m iterated integrals to construct an $N \times N \times M$ CNN-input-layer array (1).

– Initialize an array of size $N \times N \times M$ to all zeros. Think of this as an $N \times N$ array of vectors in $\mathbb{R}^M$; the first two dimensions correspond to the writing surface, and the third corresponds to the elements of the signature.
– Rescale the character to fit in an $n \times n$ box placed in the center of the $N \times N$ input layer. Let $X_i : [0, \text{length}(i)] \to [0, N]^2$ denote the i-th character stroke, parameterized to have unit speed.
– Let $X_i(t)$ denote a point moving along the i-th stroke. Mapping $X_i(t)$ into the $N \times N$ grid, store $S(X_i)_{t-\delta,t+\delta}$ in the appropriate column of the array.

Note that the first element of the signature corresponds to the zero-th iterated integral which is always a one. Thus if we look at the first $N \times N \times 1$ slice of our 3D array we see a 2D bitmap picture of the character. If $m \geqslant 1$, the next two layers contain the first iterated integrals: they encode the direction the pen was moving. If $m \geqslant 2$, the next four layers encode the second iterated integrals, and so on.

4 Results

4.1 10, 26 and 183 character classes

We will first look at three relatively small datasets [1] to study the effect of increasing the signature truncation level m.

– The Pendigits dataset contains handwritten digits 0-9. It contain about 750 training characters for each of the ten classes.
– The UJIpenchars database includes the 26 lowercase letters. The training set contains 80 characters for each of the 26 classes.
– The Online Handwritten Assamese Characters Dataset contains 45 samples of 183 Indo-Aryan characters. We used the first k handwriting samples as the training set, and the remaining $45 - k$ samples for a test set ($k = 15$ or 36).

To make the comparisons interesting, we deliberately restrict the DeepCNet k and l parameters. The justification for this is that increasing k and l is computationally expensive. In contrast, increasing m only increases the cost of evaluating the first layer of the CNN; in general for sparse CNNs the first layer represents only a small fraction of the total cost of evaluating the network. Thus increasing m is cheap, and we will see that it tends to improve test performance.

We tried smaller and larger networks, and using the training set with and without increasing its size by affine transformations (a random mix of scalings, rotations and translations). The table shows the test set error rates.

Dataset	DeepCNet	n	Transforms	$m=0$	$m=1$	$m=2$	$m=3$
Pendigits	(3,10)	10	×	3.37%	1.60%	1.32%	1.09%
Pendigits	(5,50)	32	✓	0.94%	0.43%	0.40%	0.40%
UJI lowercase	(4,10)	20	×	18.3%	16.3%	15.3%	13.4%
UJI lowercase	(5,50)	32	✓	7.4%	6.6%	6.9%	6.9%
Assamese-15	(5,20)	32	×	48.9%	40.3%	39.8%	34.8%
Assamese-15	(5,50)	32	✓	12.5%	11.9%	11.0%	11.0%
Assamese-36	(6,50)	64	✓	2.2%	2.3%	1.6%	2.3%

The results show that the first and second, and even the third iterated integrals carry information that CNNs can use to improve generalization from the training to the test set.

4.2 CASIA

The CASIA-OLHWDB1.1[9] database contains samples from 300 writers, allocated into training and test sets. It contains samples of the 3755 GBK level-1 Chinese characters.

A test error of 5.61% is achieved using a deep CNN applied to pictures drawn from the pen data [4]. Their program took advantage of a couple of features that are often used in the context of CNNs. Rather than simply drawing a binary bitmap of the character, they convolved the images with a Gaussian blur. They also used elastic distortions.

We trained a DeepCNet$(6, 100)$ with $m = 2$ and $n = 60$. We went through the training data 80 times. For the first 40 repetitions, we randomized the placement of the training characters is a small neighborhood of the center of the input layer. This gave a test error of 4.44%. We then applied affine transformations to the training characters for another 40 repetitions, giving a test error of 4.01%.

Modifying the above network by adding dropout—with dropout per level of $0, 0, 0, 0.1, 0.2, 0.3, 0.4, 0.5$—resulted in a test error to 3.58%.

5 Conclusion

We have studied two methods for improving the performance of CNNs for online handwriting character recognition, enhancing the pictures with signature information and using sparsity to increase the depth of the networks we can train. They work well together on a variety of alphabets.

This work raises a number of questions:

– Besides the path signature, there are many other ways of describing the shape of a path. You could calculate 8-direction histograms [2] to give a

sparse $N \times N \times 8$ input layer. Or you could use an unsupervised learning algorithm to characterize path segments. What is the best way of describing paths for CNNs?
– Can our approach be applied to natural images using curves extracted from the image by some deterministic curve tracing algorithm?
– Our CNN is sparse with respect to the two spatial dimensions, but not in terms of the third feature-set dimension. Predictive Sparse Decomposition [13] results in sparse feature vectors. Can doubly-sparse CNNs be built to recognize images more efficiently?

6 Acknowledgement

Many thanks to Fei Yin, Qiu-Feng Wang and Cheng-Lin Liu for their hard work organizing the ICDAR2013 competition.

References

[1] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[2] Z. L. Bai and Q. A. Huo. A study on the use of 8-directional features for online handwritten Chinese character recognition. In *ICDAR*, pages I: 262–266, 2005.

[3] Kuo-Tsai Chen. Integration of paths—a faithful representation of paths by non-commutative formal power series. *Trans. Amer. Math. Soc.*, 89:395–407, 1958.

[4] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649, 2012.

[5] Ben Hambly and Terry Lyons. Uniqueness for the signature of a path of bounded variation and the reduced path group. *Ann. of Math. (2)*, 171(1):109–167, 2010.

[6] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[7] Yann Lecun and Corinna Cortes. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[8] Yann LeCun, Fu-Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of CVPR'04*. IEEE Press, 2004.

[9] C.-L. Liu, F. Yin, D.-H. Wang, and Q.-F. Wang. CASIA online and offline Chinese handwriting databases. In *Proc. 11th International Conference on Document Analysis and Recognition (ICDAR), Beijing, China*, pages 37–41, 2011.

[10] T. Lyons and N. Sidorova. Sound compression—a rough path approach. In *In Proceedings of the 4th International Symposium on Information and Communication Technologies*, pages 223–229, 2005.

[11] Terry Lyons and Zhongmin Qian. *System control and rough paths*. Oxford Mathematical Monographs. Oxford University Press, Oxford, 2002. Oxford Science Publications.

[12] Terry J. Lyons, Michael Caruana, and Thierry Lévy. *Differential equations driven by rough paths*, volume 1908 of *Lecture Notes in Mathematics*. Springer, Berlin, 2007.

[13] Marc'Aurelio Ranzato, Christopher S. Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In Bernhard Schölkopf, John C. Platt, and Thomas Hoffman, editors, *NIPS*, pages 1137–1144. MIT Press, 2006.