CS230

Using Convolutional Neural Networks for Deconvolution: a Novel Approach for Understanding Datasets at Finer Timescales

Greg Weaver*
Department of Computer Science
Stanford University
gregweav@stanford.edu

Abstract

I propose a new method for deconvolving random variables. By expanding the ideas of Central Limit Theorem, I find that distribution functions behave uniquely as they are convolved. In this paper, I create a technique to capture this behavior. Also, by taking large samples of this behavior, I train a 1-dimensional convolutional neural network that takes in this behavior with the label of what probability distribution that was convolved to produce this behavior. After training on 10000 samples, I get a 91 percent accuracy on classifying the distribution family after it has been convolved 10 times. In essence, I show some preliminary results that allows one to deconvolve without the use of filters.

1 Introduction

A pervasive problem in the field of modeling time series is that the object that one is measuring is changing in between observations. For example, consider the simple problem of trying to model a man's walking velocity at a time scale of every minute when only given his displacement every 10 minutes. Assuming that the man changes his velocity every minute, between observation the variable X that represents his displacement is the result of the convolution of 10 random variables.

Typically, the problem of deconvolution involves taking the inverse Fourier Transform of the dataset, which requires a filter of some sort. However, I show that this problem can be solved without filters, through the use of machine learning. To accomplish this, I generate useful signals from the dataset, to be described later, and insert this signal into one-dimensional convolutional neural network. The output of this network is the label of which the network believes the dataset is governed by prior to the convolution.

2 Related work

The reason why one can not simply take estimates from the raw dataset is because of a statistical concept known as Central Limit Theorem. Central Limit Theorem states that most distributions, when convolved (or summed) converge towards a normal distribution. Specifically, Central Limit Theorem is the description of the of an asymptotic behavior. CLT has been proven in a variety of ways, of which one of the most well-known is the use of Stein's method (Stein, 1971).

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

An interesting solution to finding the finer-scale distribution from these tracks is conceptually outlined in the Berry-Esseen theorem (Esseen, 1956). This theorem postulates about asymptotic behavior of fit towards normal as random variables are summed by putting bounds on the rate of convergence to a normal distribution of an unknown posterior distribution. Specifically, Berry-Essen theorem states s that if $X_1, X_2, ...$ is a sequence of independent identically distributed random variables whose values are normalized to $E(X_i)$ = 0, $E(X_{variance})$ = 1, then the cumulative distribution function of $1/sqrt(n) \sum_{i=1}^{100} X_i$ differs from the unit normal distribution by at most $K\beta/\sqrt(n)$ where K is a constant that is usually 2 (Berry, 1941, Esseen, 1956). Not only has this asymptotic been shown in single-variable cases, but also in multi-variable cases as well (Götze, 1991).

With the Berry-Esseen way of thinking about convolution as a behavior that happens as variables are summed, I use the Kolmogorov-Smirnov statistic as used in Stein's method as a way to ask a simple question: is the rate of convergence to other different families of distributions unique to the distribution that is being convolved?

Moreover, are the patterns of convergence to other distributions unique to the family as well (i.e. if we keep summing a random variable from distribution x, and we measure the goodness-of-fit to a distribution that is not normal, how does this number progress through summation)? Understanding these questions could mean that one could look at a dataset and deduce the family of distribution of the random variable in that dataset at a time-scale that is much shorter than the latency of observations present in the dataset, without the use of inverse Fourier transforms and filters. Specifically, the use of the Fourier Transform requires many other parameters to be known, as the work of Jakob Sohl has shown (2012). Sohl has stressed that the bounds on the range of deconvolution of the dataset must be tight, to the point that the loss of knowledge of the bounds rapidly decays at a rate of $1/\sqrt(n)$, where n is the error of the bound. Classifying the signals that I have generated encounters no such problem.

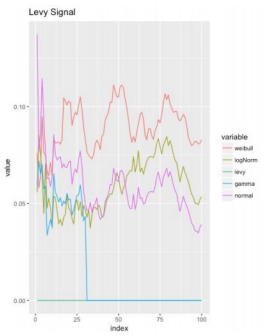
3 Dataset and Features

Extending this reasoning out, I show that the asymptotic behavior of summation not only goes towards normal (the Kolmogorov-Smirnov statistic for the fit of a particular distribution, which in this case, is the normal distribution), but also to unique constants of fit (and patterns) described for a variety of distributions.

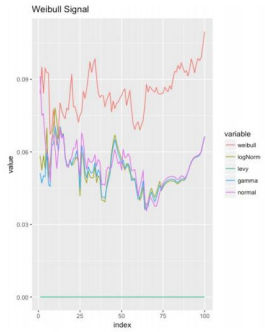
To show this, I implement the method as follows. Given a set of P observations, denoted as v, I create a 100 x P matrix, denoted as A, where row 1 is v and rows 2-100 are permutations of v. I then take the cumulative row sum, transforming the matrix A to B such that row $B[i,]$ = $A[1,] + A[2,] + A[3,]$ + ... $A[i,]$. Thus, the ith row in B is the summation of in i permutations of v. Then for each row in B, I then take maximum likelihood estimates for 5 different distributions (normal, lognormal, levy, Weibull, and Cauchy), which maximizes the probability $\prod_{i=1}^{100} f(x_i|\theta)$, where x_i is a single value in the row $B[i,]$ and θ is the set of parameters that describe the distribution to be fitted, which in this case is one of either normal, lognormal, levy, Weibull, and Cauchy.

The maximum likelihood estimate algorithm finds the best parameters that describe the data set given. I avoided some other techniques because it has shown that they have bias towards certain parameters for estimations. Using these hypothetical parameters found, I then take the Kolmogorov-Smirnov statistic that compares the hypothetical best fit parameters given from MLE to the the row at $B[i,]$ to measure how good the fit was for each of the five distributions. Thus, each row gives 5 numbers describing the fit for each distribution. Going through each row and finding these statistics, a 5 100-length vectors are created, which describes how the fit of five distributions acts as summation progresses.

As one can see in the figures provided, it seems that the five signals derived using the aforementioned methods results in signals that are noticeably different from one another. To create the dataset that the convolutional neural network will be eventually be trained on, I simulate 10000 examples of these signals, of which are split up between 2000 examples of Levy, weibull, lognorm, normal and gamma distributions. I also generate 1000 examples for a test set that has 200 of the same split among each of the five distribution as the train set. It is interesting to note that normalizing the signals resulted in drastically reduced classification rates in the neural network, by a rate of almost 30 percent.

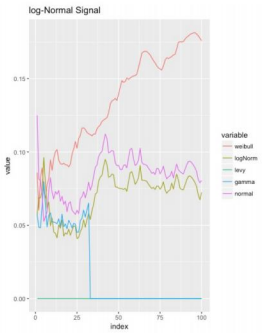


(a) Progression of fits as Levy distribution is convolved.

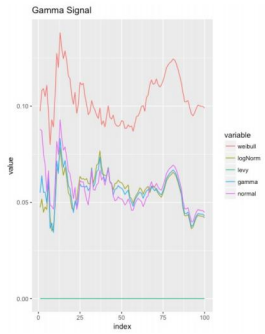


(b) Progression of fits as Weibull distribution is convolved.

Figure 1: Index represents the i^{th} summation of the dataset. Y-axis represents KS-Stat value



(a) Progression of fits as log-Normal distribution is convolved.



(b) Progression of fits as Gamma distribution is convolved.

Figure 2: Index represents the i^{th} summation of the dataset. Y-axis represents KS-Stat value

4 Methods

To classify these signals, I try out a variety of network architectures and see which one has the best classification accuracy. I first tried a vanilla neural network and then settled on a certain architecture within the class of 2-layer 1-D convolutional neural network. The error function that was used to train the network was the standard cross-entropy loss.

Training occurs in the network by taking repeated passes over the training set, measuring the difference in the labels put out by the network to the key given in the training set. By taking the error between the two (cross-entropy loss), the network is trained over the course of 40 epochs.

It is important to note that expressing all five of the signals as the concatenation of the dataset drastically reduced the expressive power of these signals. By creating a single 500-length vector for the dataset, my classification accuracy was ruined, getting random odds (20 percent) on every training iteration.

5 Experiments/Results

Given that the training of the neural networks was done on my CPU in only 30 - 40 minutes, I kept mini-batch sizes down to 32 and was able to consistently drive down the loss function at every epoch. Using a learning rate of .09 got the best results for quick learning and high accuracy.

Finding the best classification rates meant finding a balance between a large enough network to account for the complexity of the data while also avoiding the fact that a large network that would simply over fit the data. For example, as seen in figure 4, deeper networks over fit the data such that classification rate was reduce by roughly 10 percent. Also note that networks with larger convolutional

Model	Train Acc.	Test Acc.	Tra. Size	Test Size
Vanilla, two-layer	.97	.87	10000	1000
2 Conv1D layers+relu	.92	.91	10000	1000
3 Conv1D layers+sig	.997	.8	10000	1000
2 Conv1D layers + str = 5	.772	.32	10000	1000

Figure 3: Table of values matching network architectures to accuracy.

Table 1: Confusion Matrix of Test Behavior

	Levy	Weibull	LogNormal	Gamma	Normal
Levy	190	0	1	3	0
Weibull	5	188	2	15	6
LogNormal	12	0	178	3	12
Gamma	3	12	4	174	2
Normal	0	0	15	5	180

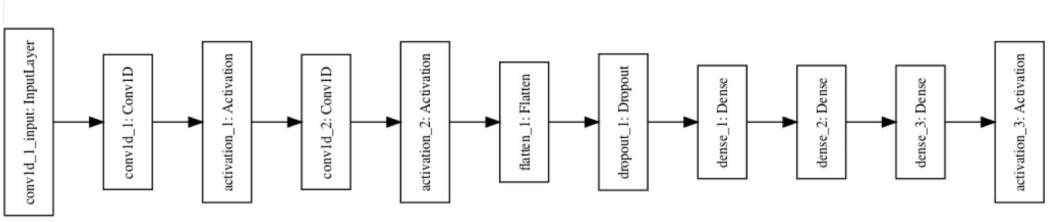


Figure 4: The convolutional architecture that resulted in the best classification

strides did not have similar performance as those with smaller strides. I hypothesize that convolutional layers with larger strides did not have enough detail gleaned in the early convolutional stages of the network to have a representation in the network that was able to generalize on the test set.

The network architecture as seen in figure 5 resulted in the highest accuracy, whose performance did not vary much from its performance on the training set. Because of this, I do not think that I am over-fitting the data due to the very small variance between the performance on the training set and the held-out test set.

Note the confusion matrix in table 1, where the vertical axis has the true labels and the horizontal axis has the labels output by the network. true label from the test data. It can be seen that the discrimination that the network had was between weibull and gamma and also between log-normal and normal. This is consistent with what I would predict: the difference between gamma and weibull and also between log-normal and normal are small (at least visually). However, the difference between gamma and weibull is larger than the normal vs log-normal, and this is also surprising reflected in the slight increase in discrimination between these two.

Let us take a note of what I have just shown. The dataset that I trained the classifier has a signal derived for a dataset that, for every data point in the dataset, it is the summation of ten random variables. In other words, the variable is changing ten times between observations, and the difference between examples within training examples will look largely normal. However, based on the signal generated using the methods described above, we can classify the dataset to its deconvolved family of distribution at a rate of 91 percent of the time.

Surprised by this development, I went into the offices of a few statistics professors here at Stanford to see whether there has been any precedent to the work that I have laid out here, including Sourav Chatterjee (expert in Central Limit Theorem), Emmanuel Candes (creator, along with Terrance Tao,

of compressed sensing, of which this work has applications to as well), Guenther Walther, and David Siegmund. They were surprised by the results, and said that they hadn't seen anything related to this work within statistics. Sourav, whose career partly involves the CLT, was particularly surprised.

6 Conclusion/Future Work

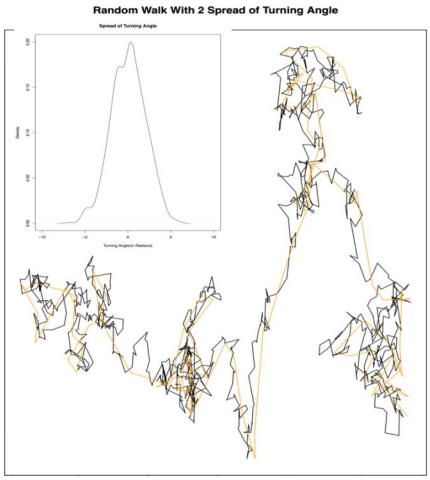


Figure 5: A random walk with the turning angle shown. Essentially, this neural net is able to understand what distribution governs the displacements in the black line, given only the orange line.

In figure 5, note the black line and the orange line. Assuming the classification of the neural network is correct, we can understand the distribution of the black lines, given only the orange lines. This insight from this toy example offers applications towards random walk theory and predicting stock value (which relies on random walks to underlie many abstractions). For example, one can greatly increase profits of high-frequency trading if one has a greater knowledge of the R.V. at a finer time scale, and thus can manage risk better in between stock updates.

In essence, in this work I develop a way to create signals from datasets that have been convolved many times and for the most part already look like a normal distribution due to the central limit theorem. These signals I show are sufficiently unique to the original density function that was convolved to produce that dataset by training a neural network classifier that can classify what density function was convolved, receiving an accuracy of 91 percent on a held-out test set.

This result has interesting theoretical implications towards understanding deconvolution, and also towards applying these methods towards new problems involving time series and random walk theory.

The next step towards realizing the representational power of these signals lies how robust these signals are to linear combinations of a host of other distributions. Thus, the next step in research for this method that I can see is testing to see whether one can train a 1-d convolutional neural network to predict what specific linear combination of density functions that have been convolved. The first step towards accomplishing this would be to change the final layer of the network to have a ReLu activation function, which would allow for multiple outputs to be outputted and also allow for linear combinations to be expressed by the linear scaling nature of ReLu. That next step, if realizable, would be a big step forward in understanding the how much potential these signals have in deconvolving random variables.

7 Contributions

I have done all the work described above.