Differentiable Kernel Regression using Convolutional Neural Networks

Narges

David

Abstract

Kernel methods have relied on parametric kernels and cross validation to recover the kernel family and optimal parameters. This severely limits the power of these methods in kernel density estimation, regression, and any consequent tasks. In this paper, we provide a simple formulation of kernel regression based on convolution, which enables us to learn the kernel function and parameters directly from temporal data. We show how learning kernels improves the quality of density estimation and regression, compared to traditional cross validation within parametric families of kernels. Moreover, we show our method can easily extend to multivariate input with structural and temporal dependencies, and is able to recover dependency structure on the input time series automatically.

1 Introduction

In several areas of data science and machine learning, tractability of computations has often been traded off with simplifying assumption on the dependencies and distribution of the random variables of the model.[] [] [] [] As the size of datasets are growing, these assumptions begin to hurt more than they help [] [] []. Kernel density estimation[] is one of the methods which does not impose most of the simplifying assumptions on the distribution of the data[], and has been previously studied in the context of density estimation, regression, causal inference, conditional distribution modeling, among others. Kernel regression[] in particular, is a general method for estimating any function, given samples from the true distribution and a kernel similarity function.

While the theory of kernel regression allows the use of any positive semi definite kernel, in practice, methods only cross-validate over a few well-known kernel functions such as Radial basis(Gaussian,) Laplace, etc. and fail to consider the entire space of legal kernels. Of course when one leaves this step to cross-validation nothing more is reasonable to expect. In this paper, we try to liberate the kernel regression method from searching within a set of pre-defined kernel families.

In particular, we focus on irregularly measured temporal time series, and show how to construct a differentiable formulation of kernel density estimation for single time series as well as multiple dependent time series. Our method is limited to discrete time series, and we use a differentiable formalization and gradient descent to estimate the kernel function over its domain. As we show, not only this method is easily able to recover multivariate kernels, it can also recover spatio-temporal dependencies between the random variables.(maybe! we'll see)

2 Kernel Regression

Imagine the input to be samples from D time series, each sampled irregularly. An example context would be different types of lab measurements for a patient at different time points across their life.

We denote the samples as $x^1_{t^1_1}$, $x^1_{t^1_2}$, ..., $x^1_{t^1_{n_1}}$, ...,$x^D_{t^D_1}$, $x^D_{t^D_2}$, ..., $x^D_{t^D_{n_D}}$, where x^d refers to time series d and t^d_1,... $t^d_{n_d}$ refer to the time points over which time series d is sampled.

Kernel regression provides a general formalism for estimating any function with additive noise. Let's start from a single time series, $x(t)$. Kernel regression assumes the following.

$$x = f(t) + \epsilon$$
$$\epsilon \sim N(0, \sigma^2)$$

Given observed samples x_{t_1},...x_{t_n} from the series, general function regression with additive noise lets us estimate the value of x at a new time point t_{new} as follows.

$$x(t_{new}) = \mathbf{E}_{x \sim P(x|t=t_{new})}[x]$$
$$\mathbf{E}_{x \sim P(x|t=t_{new})}[x] = \int_x xP(x|t=t_{new})dx = \int_x x\frac{P(x, t=t_{new})}{P(t_{new})}dx$$

At this point, one can use kernel density estimation to estimate the probabilities $P(x, t=t_{new})$ and $P(t_{new})$ from the training data. Nadaraya[] and Watson[]showed that using a positive semidefinite kernel function $K(t, t')$, the nonparametric regression formulation is reduced to:

$$\mathbf{E}_{x \sim P(x|t=t_{new})}[x] = \frac{\sum_{i=1}^n x_{t_i} K(t_{new}, t_i)}{\sum_{i=1}^n K(t_{new}, t_i)} \tag{1}$$

We can now rewrite the nonparametric regression using convolution operator. To be able to use functional notation, we first write the sequence of observed samples x_{t_1},...,x_{t_n} as a function: $\bar{X}_{train}(t)$ = $\sum_{i=1}^n x_{t_i}\delta(t, t_i)$, where $\delta(t, \tau_0) = 1$ when $t = \tau_0$, and 0 otherwise.

Denoting convolution operator as $*$, i.e. $(k * f)(t) = \int_\tau K(t-\tau)f(\tau)d\tau$, the numerator of the kernel regression is equal to the following.

$$\sum_{i=1}^n x_{t_i} K(t_i - t_{new}) = (K * \bar{X}_{train})(t_{new})$$

The denominator $P(t_{new})$ can similarly be written as a convolution of the kernel function with a sequence of 1s at each point at which we have a sample, denoted as $I(\bar{X}_{train} : observed)(t) = \sum_{i=1}^n \delta(t, t_i)$.

$$\sum_{i=1}^n K(t_{new}, t_i) = (K * I(\bar{X}_{train} : observed))(t_{new})$$

So the kernel regression formulation of Nadaraya and Watson reduces to the following formulation.

$$\mathbf{E}_{x \sim P(x|t=t_{new})}[x] = \frac{(K * \bar{X}_{train})(t_{new})}{(K * I(\bar{X}_{train} : observed))(t_{new})}$$

We summarize this formulation in figure 1. The benefit of this formulation, is that we can now differentiate this ratio with respect to each $K(\tau)$ at each position τ within the kernel domain, using function composition and back-propagation[]. We can also compose this differentiable kernel regression module within any subsequent differentiable operators and perform multiple tasks such as classification, reconstruction or beyond via gradient descent and back-propagation. In this paper, we used leave-one-out imputation mean squared error as the loss function. In practice, we assume the domain of K is bounded over $[-M, M]$ range, therefore the learning task will have $2M + 1$ parameters.

Our formulation easily extends beyond single time series. Assuming that we now have D time series, we can assume the kernel K is a matrix of size $D \times (2M + 1)$, and optimize each kernel value between series i at time r and series j at time s. The formulation of the kernel regression then becomes a 2D convolution of this kernel matrix with all time series' observed points in the numerator, divided by the 2D convolution of the kernel matrix with a binary matrix encoding which series at which time point does have a nonzero observation. To our knowledge, this is the first formulation which allows modeling of any possible interactions between different time series and input variables.

Figure 2 shows the architecture of the entire model for the kernel regression.

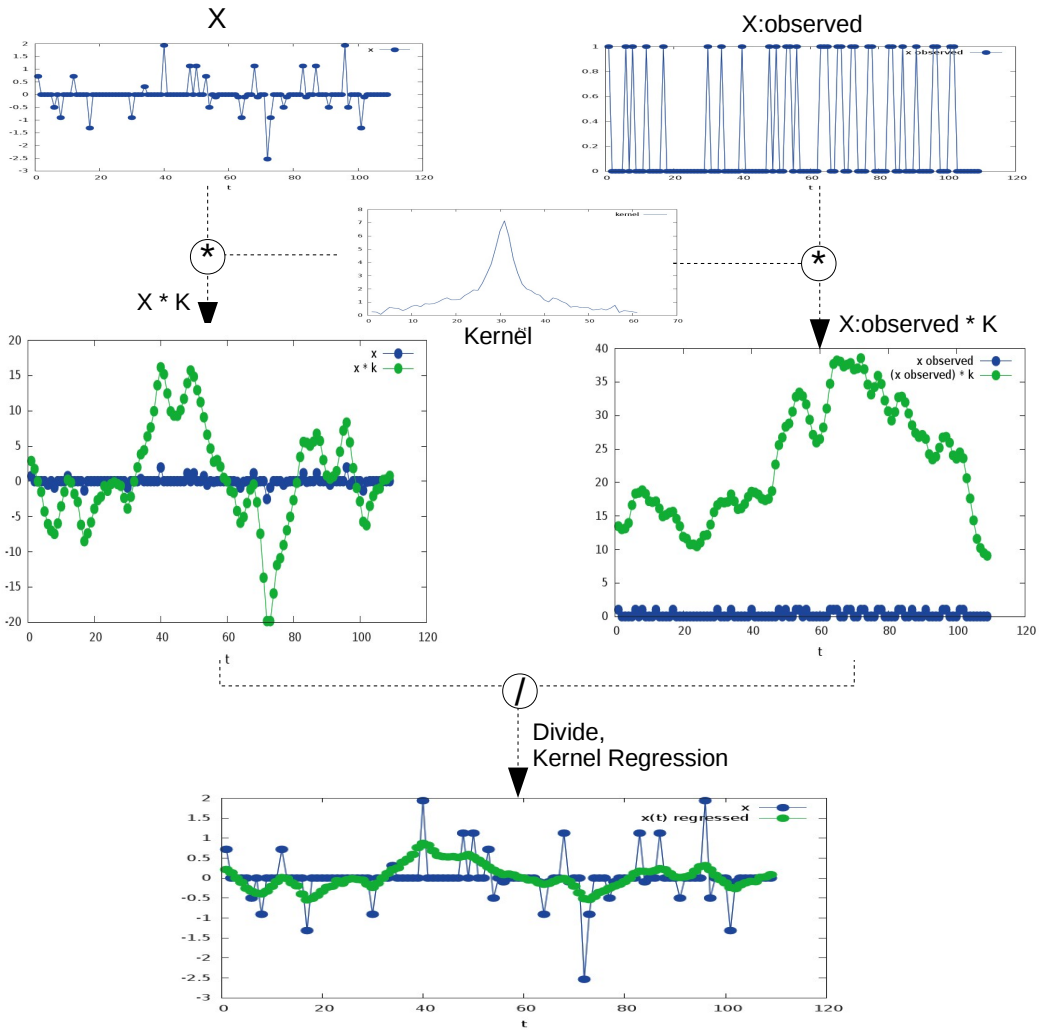
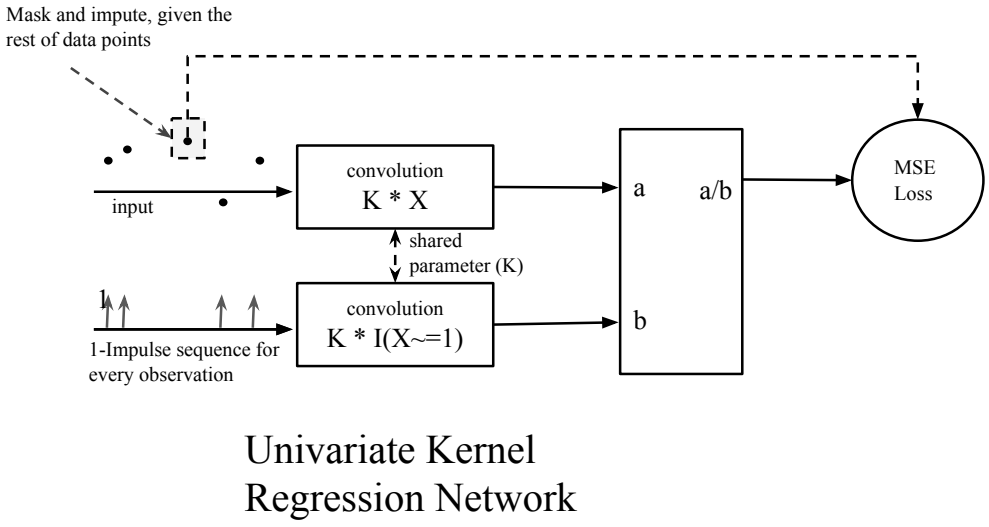


Figure 1: Visualization of an example data series and kernel regression as convolution.

3 Related Work

traditional kernel methods, GP, univariat2e and multivariate. (For GP) http://www.cs.toronto.edu/ rgrosse/icml2013-gp.pdf code: https://github.com/jamesrobertlloyd/gp-structure-search



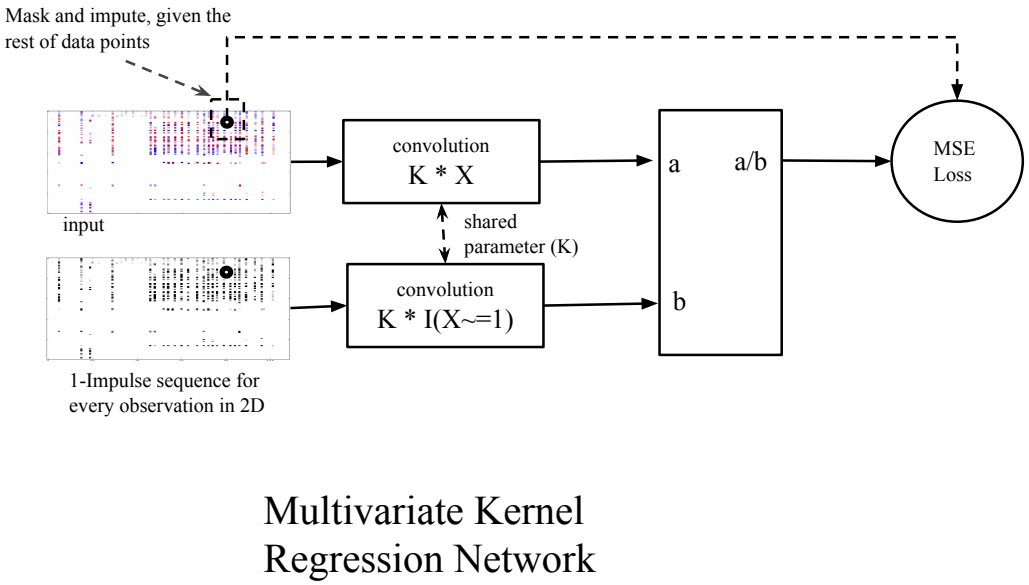


Figure 2: Architecture for kernel regression.

4 Experiments and Results

4.1 Data

We normalize each series to be zero mean, unit variance (each measurement per person normalized on its own).. more details later.. Training data is 5K, and validation set is another 5K. There are 100 lab measurements over 109 months.

4.2 Synthetic Data

Sampling multiple codependent Gaussian processe (I know how to do it) How about other forms of dependencies??

4.3 Data Augmentation for Training

In order to successfully train the model with finite training data, we augmented each time series by adding Gaussian noise to each observation, and wiggling the time of each observation by a small amount, randomly drawn from another Gaussian distribution with standard deviation of 1. This step is very important for convergence of the method.

4.4 Technical Details of the implementation

In order for the back-propagation to remain stable, we limited the derivative of the ratio such that if the magnitude of denominator was smaller than an epsilon, we did not back-propagate its derivative. We also filtered the gradients within range of [-10 +10], as higher values would potentially cause numerical instability. The effect of the limit was that the convergence would be slower, but more stable.

5 References