CTS^2: Time Series Smoothing with Constrained Reinforcement Learning

Yongshuai Liu YSHLIU@UCDAVIS.EDU
Xin Liu XINLIU@UCDAVIS.EDU
The University of California, Davis, 1 Shields Ave, Davis, CA 95616, USA

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

Time series smoothing is essential for time series analysis and forecasting. It helps to identify trends and patterns of time series. However, the presence of irregular perturbations disrupt the time series smoothness and distort information. The goal of time series smoothing is to remove these perturbations while preserving as much information as possible. Existing smoothing algorithms have complete freedom to make corrections to the data points which often over smooth the time series and lose information. None of them considers constraining data corrections to the best of our knowledge. Moreover, most existing methods either do not smooth in real-time or their parameters need to be hand-tuned in different scenarios. To improve smoothing performance while considering data correction constraints, we propose a **C**onstrained reinforcement learning-based **T**ime **S**eries **S**moothing method, or CTS^2. Specifically, we first formulate the smoothing problem as a Constrained Markov Decision Process (CMDP). We then incorporate data correction constraints to restrict the amount of correction at each point. Finally, we learn a policy network with a linear projection layer to smooth the time series. The linear projection layer ensures that all data corrections satisfy the data correction constraints. We evaluate CTS^2 on both synthetic and real-world time series datasets; our results show that CTS^2 successfully smooths time series in real-time, satisfies all the correction constraints, and works efficiently in a variety of scenarios.

Keywords: Multivariate time series, Smoothing, Constraint, Deep reinforcement learning

1. Introduction

A time series is a sequence of discrete-time data points indexed temporally. It is a common data type with increasing applications. For example, the MEMS gyroscope and accelerometer data are widely used in smartphones to monitor human activity Lu and Liu (2015); Qu et al. (2015) and direct video stabilization Hu et al. (2020). Moreover, ECG Muhammad et al. (2020) and speech signal Lu et al. (2017) are essential indicators for human health and the stock curves exhibit the main trend of the financial market Islam and Nguyen (2020).

The time series contains trend information, which can be used to forecast the following data points. However, this trend information may be distorted due to data perturbations which are unavoidable in data collection. For example, the environmental noise caused by temperature or humidity affects the time series generated by sensors. In time series forecasting, the presence of perturbations can harm predictions Cheng et al. (2021).

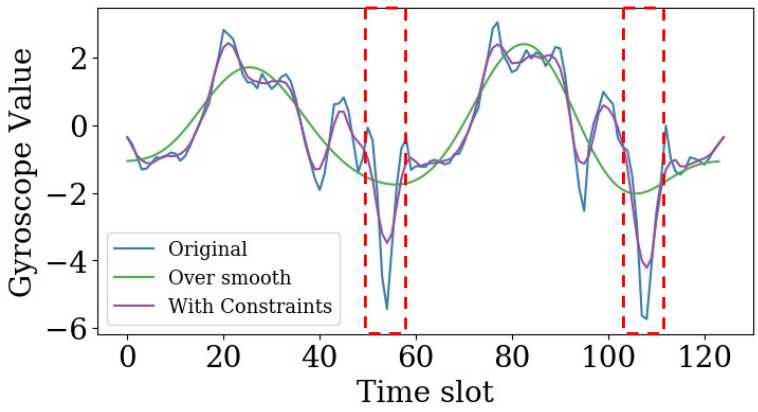


Figure 1: Gyroscope time series demonstration

Time series smoothing is to reduce or eliminate the effect of these perturbations, while preserving as much information as possible and thus allowing important trends and patterns to stand out Liu et al. (2018). Currently, the main direction for achieving smooth time series is to design an algorithm that corrects data perturbations, such as low-pass filters Wesseling (1991) and polynomial fitting Savitzky and Golay (1964) methods. These traditional methods are effective in some circumstances Katris (2021), but have limitations.

First, none of the methods considers data correction constraints while smoothing the time series. Those methods usually have complete freedom to make changes to the data points in time series. However, such unconstrained smoothing methods can potentially over smooth the time series and lose information. A motivating example shows in Fig. 1. The original time series comes from one dimension of a gyroscope. The data shows obvious periodicity and the peak areas may contain important information, e.g. go upstairs. If the time series is overly smoothed, it could remain the global trending information but it would omit the local peak information. Losing such local features can be problematic to certain applications, e.g. inaccurate step count in the above example. In medical applications for diagnosis, losing such information can result in erroneous diagnoses. In video stabilization application Huang and Onnela (2020), over smoothed video frames can out of capture screen. To address this issue, we involve predefined constraints, which could derive from domain knowledge, to restrict the amount of correction allowed at each data point.

Second, these methods usually require many hyperparameters and careful manual tuning in different scenarios. For example, in polynomial fitting methods Savitzky and Golay (1964), a higher-order fitting will over smooth the data, while a lower order fitting will under smooth the data. In addition, hyperparameters cannot be reused in different scenarios, necessitating hyperparameter re-tuning if scenarios change.

Last, most algorithms (e.g. low-pass filter Wesseling (1991)) are used for post-processing and cannot output the smooth time series in real-time. For example, in video camera applications which is widely used in mobile phones, it is often desirable to store the processed video in real time, instead of doing post video processing because of storage limitations.

To address the limitations, we study time series smoothing with correct constraints. Furthermore, we aim to handle the problem in real time and without hand tuned hyperparameters in different scenarios. Specifically, we first formulate the problem as a Constrained Markov Decision Process (CMDP) by setting appropriate states, actions and rewards. We

process the time series with a fixed-size sliding window. In our formulation, states are the data points in the current window, actions represent how to correct the data, and rewards are defined as the smoothness of the current window. Second, we incorporate data correction constraints to ensure that corrected data points are not too far from their original counterparts. Third, we design a reinforcement learning (RL) policy network with a linear projection layer to re-target the data points. The layer projects all infeasible actions to the feasible space, ensuring that the aforementioned constraints are never violated.

In summary, our contributions are as follows:

- We propose CTS^2, an RL algorithm to smooth multivariate time series. To the best of our knowledge, our work is the first to consider constraints in time series smoothing and it is easy to integrate with existing RL methods.
- Our work smooths the time series with a very short delay. It is fair to say CTS^2 can achieve real-time smoothing after the policy learning.
- The parameters of CTS^2 can easily be reused in new scenarios and it reduces efforts needed to fine-tune the parameters manually.
- We conduct extensive experiments to compare CTS^2 with traditional methods on both synthetic and real-world datasets. CTS^2 outperforms the baselines with zero constraint violations and achieves similar smoothness.

2. Related work

2.1. Time Series Smoothing

Many smoothing methods exist, such as moving average methods Singh et al. (2013), exponential smoothing methods Kammeyer and Kroschel (1998), Savitzky–Golay filters Savitzky and Golay (1964) and Fourier filters Wesseling (1991).

Moving Average: Moving average methods generate a series of averages by taking the mean of values in the time series within a sliding window. Taking these averages makes the time series smoother and can even extract certain information contained in the time series.

Formally, assuming we have a time series $O_1, O_2, O_3, ..., O_t$, we compute D_t from the average of last kth elements: $D_t = \frac{O_{t-k}+O_{t-k+1}+...+O_t}{k}$, where k is a smoothing parameter.

Exponential Smoothing: In the moving average methods above, we assign equal weight to each point in the average. Exponential smoothing, on the other hand, assigns exponentially decreasing weights to past data points; in other words, recent points are weighted more heavily than older data points. Intuitively, exponential smoothing attempts to acknowledge older information while prioritizing more recent data. Exponential smoothing derives new points with previous data recursively: $D_1 = O_1, D_t = \alpha * O_t + (1-\alpha) * D_{t-1}, i > 1$, where $\alpha \in [0, 1]$ is a discount parameter, O_t is the original time series point and D_t is a point of the smoothed time series.

Savitzky–Golay (SG) Filter: The SG filter is a classic data smoothing method based on local least-squares polynomial approximation. It fits a polynomial to a set of input data points and then evaluates the resulting polynomial at a single point within the approximation interval. The SG filter has no convergence issues and maintains the shape

of the time series better. Because of its performance benefits, SG filters are widely applied for time series smoothing.

Fourier Filter: Fourier filtering methods (low-pass filtering), are based on the Fourier transform. By translating a time series from the time domain into the frequency domain and suppressing the high-frequency components, Fourier filters achieve a smoothing effect. The performance of the filter is dependent on the cut-off frequency, a hyperparameter.

2.2. Constrained Reinforcement Learning

In our work, we consider an RL problem with instantaneous action constraints Liu et al. (2021b, 2020b), which require the action to satisfy a condition in each time step. One approach to handle instantaneous constraints is to project the actions in each step to the feasible space. Optlayer Pham et al. (2018) is proposed to restrict actions for robotics running in the real world. It projects the infeasible output of the actor-network to a feasible action by adding a new layer after the actor-network. This new layer aims to find the closest feasible action that satisfies all constraints in L_2 distance. In Bhatia et al. (2019), the authors propose approximate linear projected approaches that aim to satisfy resource allocation constraints. A similar idea is applied in Liu et al. (2020a, 2021a). The only different is that some constraints can not be explicitly checked due to the complexity of the system. To handle these unknown instantaneous constraints, they train another neural network in advance to predict their values; the constraints can then be easily checked.

3. Preliminary

3.1. Constrained Markov Decision Process (CMDP)

CMDP is an extension of Markov Decision Processes (MDP) by adding a cost function. CMDPs are represented by a tuple $(S, A, R, P, \mu, \gamma, C)$ Sutton and Barto (2018), where S is the set of states, A is the set of actions, $R : S \times A \times S \mapsto \mathbb{R}$ is the reward function, $C : S \times A \times S \mapsto \mathbb{C}$ is the cost function. The cost function defines an instantaneous constraint on the actions $c(s, a) \leq \epsilon$. If $a \in A$ satisfies all constraints, a is feasible.

Furthermore, $P : S \times A \times S \mapsto [0, 1]$ is the transition probability function, where $P(s^{'}|s, a)$ is the transition probability from state s to state $s^{'}$ upon taking action a, $\mu : S \mapsto [0, 1]$ is the initial state distribution and γ is the discount factor for future reward. A policy $\pi : S \mapsto \mathcal{P}(A)$ is a mapping from states to a probability distribution over actions and $\pi(a|s)$ is the probability of taking action a in state s. We write a policy π as π_θ to emphasize its dependence on the parameter θ (e.g., a neural network policy with parameter θ).

For a CMDP, the goal is to find a policy π_θ which maximizes the discounted cumulative reward while satisfying all constraints. Formally, the objective is

$$\begin{aligned} \max_{\theta} \ J_R^{\pi_\theta} &= \mathbb{E}_{\tau \sim \pi_\theta}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})], \\ s.t. \quad & c_i(s_t, a_t) \leq \epsilon_i, \forall i \in [K], \end{aligned} \tag{1}$$

where ϵ_i is the limit for each constraint, $\tau = (s_0, a_0, s_1, a_1...)$ denotes a trajectory, and $\tau \sim \pi_\theta$ means that the trajectories are sampled from the policy π_θ.

For a trajectory starting from state s, the value function of state s is $V_R^{\pi_\theta}(s)$. The action-value function of state s and action a is $Q_R^{\pi_\theta}(s,a)$ and the advantage function is

$$A_R^{\pi_\theta}(s,a) = Q_R^{\pi_\theta}(s,a) - V_R^{\pi_\theta}(s). \tag{2}$$

3.2. Policy Gradient Methods

Policy gradient Sutton et al. (2000) methods are applied to find an optimal policy of an unconstrained MDP problem. These methods involve calculating the gradient of the objective in Eq. (1), $\bigtriangledown J^{\pi_\theta} = \mathbb{E}_t[\bigtriangledown_\theta \log \pi_\theta(a_t|s_t)A_t]$, where π_θ is the current policy under parameter θ and A_t is the advantage function (Eq. (2)) at time step t. Thereafter, θ is updated as $\theta = \theta + \eta \bigtriangledown J^{\pi_\theta}$, where η is the learning rate.

Trust Region Policy Optimization (TRPO) Schulman et al. (2015) is an advanced policy gradient method that was proposed to achieve monotonic improvement of the new policy based on the results of the previous policy. The objective is approximated with a surrogate function combined with the Kullback Leibler (KL) divergence Kullback and Leibler (1951):

$$\begin{aligned} \max_\theta \ & L^{TRPO}(\theta) = \mathbb{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t], \\ s.t. \quad & \mathbb{E}_t[KL[\pi_{\theta_{old}}(a_t|s_t), \pi_\theta(a_t|s_t)]] \leq \delta, \end{aligned} \tag{3}$$

where δ is the step size limitation.

4. Multivariate Time Series Smoothing Formulation

We formulate a time series $O = (O_1^i, O_2^i, ..., O_t^i, ...)$ as n-dimensional time-aligned data, where i represents the data dimension and t is the time index. When a time series O is collected over time, imperfections in the data collection process lead to random perturbations in the resultant time series; we aim to reduce these perturbations in the original time series O and output a smoother time series $D = (D_1^i, D_2^i, ..., D_t^i, ...)$. Moreover, the data correction should satisfy certain constraints that we will describe later.

For clarity, consider the toy example of a one-dimensional time series with nine-time slot points, as seen in Fig. 2. The red curve is the original time series with perturbations and the green one is the smooth output. Our algorithm translates the original red points, O, to the green ones, D, and makes the curves as smooth as possible. We quantify the smoothness of a curve by the sum of all angles constructed by any three continuous points. For example, in Fig. 2, the angle for the three continuous points D_3, D_4, D_5 is $\alpha = \arccos(\frac{\overrightarrow{D_4D_3} \cdot \overrightarrow{D_4D_5}}{|\overrightarrow{D_4D_3}||\overrightarrow{D_4D_5}|})$. With this definition, we can say that the time series is smoother when the sum is high and vice versa. The sum of angles is a representative metric to quantify the smoothness of time series in the literature Froeb and Koyak (1994). Our algorithm is not limited to the metric. Different metrics are evaluated in the Sec. 6.2.

In addition to smoothing the time series curves, the data correction process has to satisfy the following linear constraints:

- The correction for any dimension of any point cannot exceed a limitation:

$$|D_t^i - O_t^i| \leq \epsilon_1^i. \tag{4}$$

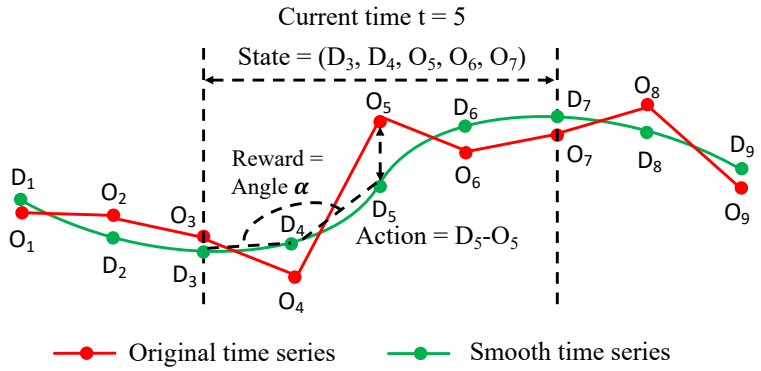


Figure 2: Mapping Time Series Smoothing to CMDP

- The weighted multi-dimensional correction for any point cannot exceed a limitation across all dimensions:

$$\sum_{i=1}^{n} \omega_i |D_t^i - O_t^i| \leq \epsilon_2. \tag{5}$$

  where ϵ_1^i and ϵ_2 are limitations and ω_i is the weight, they are predefined depending on the need or domain knowledge.

The intent is to ensure that the correction does not modify the original data too much, and thus preserve as much original information as possible.

4.1. Mapping to CMDP

In order to map the time series smoothing problem to a CMDP, we process the time series with a fixed window size W, which needs to be greater than two. The window starts from the first point and shifts one point in the next time slot. We have the following definition, as demonstrated in Fig. 2, e.g. $W = 5$:

- State: Assuming that the current time slot is t, the state s_t^i for each dimension is $s_t^i = (D_{t-2}^i, D_{t-1}^i, O_t^i, O_{t+1}^i, ..., O_{t+W-3}^i)$. For all dimensions, we concatenate all the state vectors together, thus state $s_t = (s_t^1, s_t^2, ..., s_t^n)$. The fist two points (D_{t-2}^i, D_{t-1}^i) are the corrected points from past two time slots which cannot be changed, the middle point O_t^i is the current original data point which we are dealing with and the last $W-3$ points $(O_{t+1}^i, ..., O_{t+W-3}^i)$ are future data points, which means CTS^2 needs $W-3$ time slot delay to output the smooth point. In each time slot, CTS^2 can get the corrected point D_t^i for the current time t. When $t \leq 2$, $D_1^i = O_1^i, D_2^i = O_2^i$.

- Action: The policy action a_t^i is defined as the correction amount on the current time point O_t^i. For all dimensions, we concatenate all the corrections together. Thus we have the action $a_t = (a_t^1, a_t^2, ..., a_t^n)$. Given an the action a_t^i from a policy, we can compute the smoothed point as $D_t^i = O_t^i + a_t^i$.

- Reward: The reward function at a time t is defined as the angle constructed by the first three continuous points $D_{t-2}^i, D_{t-1}^i, D_t^i$. We derive it through the vector $\overrightarrow{D_{t-1}^i D_{t-2}^i}$ and $\overrightarrow{D_{t-1}^i D_t^i}$, where D_t^i is the current corrected data point getting from $D_t^i = O_t^i + a_t^i$. For

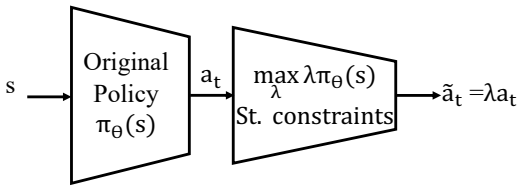


Figure 3: Instantaneous constraints projection layer

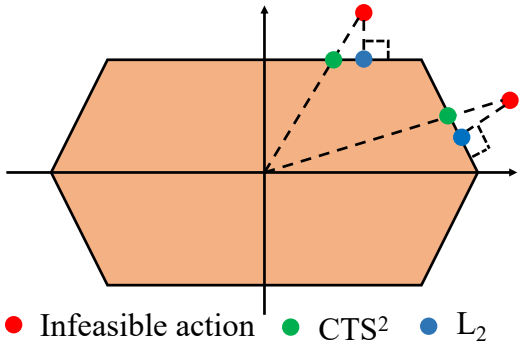


Figure 4: The intuition of CTS^2.

all dimensions, the reward is the sum of all angles. The intuition is that the corrected data points $D^i_{t-2}, D^i_{t-1}, D^i_t$ are smoother if the reward is larger. The smoothness of the entire time series is calculated with the total reward accumulated with time, which is also the objective of the policy (Eq. 1).

Besides obtaining a smooth time series by maximizing the cumulative reward expectation, the actions have to satisfy the instantaneous constraints that are listed above (Eq. 4 and 5). Next, we discuss how to learn the constrained policy.

5. CTS^2: Time Series Smoothing with Constraints

We base our algorithm on TRPO (Eq. 3) to learn a policy that maximizes the cumulative reward. The policy π_θ takes a state s_t as input and outputs an action $a_t = \pi_\theta(s_t)$. TRPO, however, is unable to handle the constraints alone.

In order to handle the linear constraints (Eq. 4 and 5), we further extend our policy by adding an additional, linear projection layer after the original policy network π_θ. In each step, if the output of the original unrestricted policy $\pi_\theta(s)$ violates the constraints, this linear projection layer projects this action $a_t = \pi_\theta(s)$ to a new, feasible action $\widetilde{a}_t$. The role of the linear projection layer is to find a positive scalar λ that scales an action to respect the constraints. The new action is then $\widetilde{a}_t = \lambda a_t$. Formally, we have

$$\begin{aligned} &\arg\max_\lambda \lambda\pi_\theta(s), \\ \text{s.t.} \quad &|\lambda a^i_t| \le \epsilon^i_1, \\ &\sum_{i=1}^{n} \omega_i |\lambda a^i_t| \le \epsilon_2. \end{aligned} \tag{6}$$

The network structure is shown in Fig 3 and the pseudo-code is shown in Algorithm 1.

5.1. Computation Complexity and Practical Implementation

The typical way to address instantaneous constraints is to find the closest L_2 projection by solving the following Quadratic Programming (QP) problem Pham et al. (2018):

$$\begin{aligned} &\min_{\widetilde{a}} \frac{1}{2} \|\widetilde{a} - \pi_\theta(s)\|^2 \\ &s.t.\ \widetilde{a} \text{ is feasible} \end{aligned} \tag{7}$$

Algorithm 1 Policy learning of CTS^2

Input: Initialize policy π with parameter $\theta = \theta_0$. Set the hyperparameter KL divergense step size and learning rate α for TRPO, set the sliding window size W for the time series. and set the maximum iteration number I and episode number E for each iteration.

Output: The policy parameters θ

```
Initialize the computational graph structure.
for Iterations=0,1,2,...I do
  for Episodes=0,1,2,...E do
    Sample a multivariate time series O with noise and drift
    for t=3,... do
      Sample the state s_t = (s_t^1, s_t^2, ..., s_t^n)
      Get the feasible action ã_t = λ a_t with Eq. 6
      Achieve the reward by calculating the sum of angular rates in all dimensions.
      Process the states, actions and rewards to advantage, etc
    end for
  end for
  Update the policy parameters with gradient θ_{k+1} = θ_k + α ▽_θ L^{CLIP}(θ).
end for
Return policy parameters θ = θ_{k+1}
```

Because the objective is strictly convex and the constraints are linear in our case, Eq. 7 can be solved by the KKT conditions Boyd and Vandenberghe (2004), and a unique solution exists. The main drawback is computational complexity as this optimization occurs in every infeasible step and most RL methods require a large number of samples.

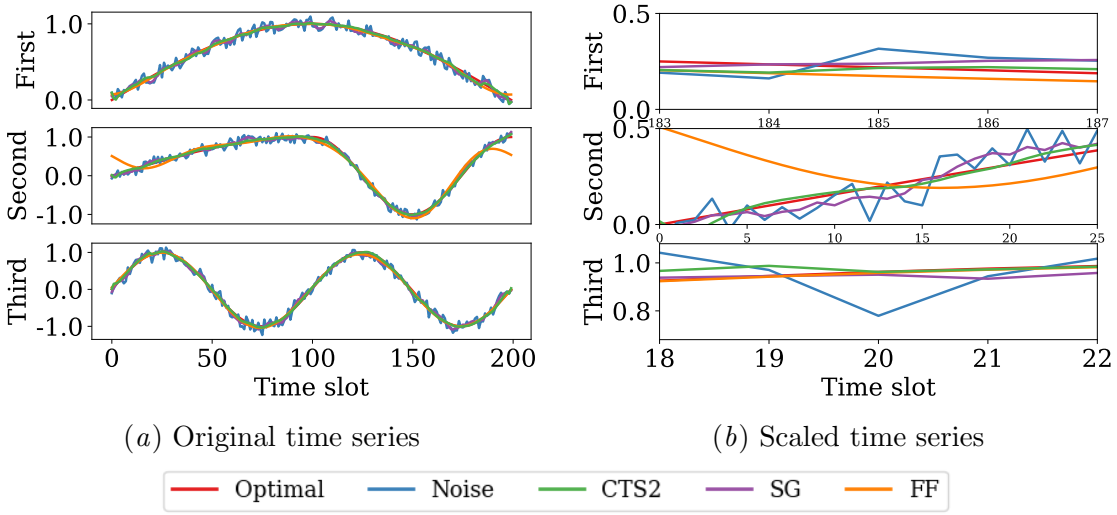
Since the constraints in our case are a linear combination, CTS^2 reduces the computation complexity by doing a linear approximation. The idea is that we do not need the optimal $L2$ projection of actions; instead, we just need any projection function which can respect the constraints. Empirically, the projection derived by CTS^2 is not very far from the optimal $L2$ projection. The intuition behind this is shown in Fig. 4 in the case of two-dimensional actions. The brown area is the feasible space. CTS^2 and L_2 both project the infeasible action to the feasible boundary; the benefit of CTS^2 is that Eq. 6 is approximately solved much more quickly in the following way.

In practice, the first constraint $|\widetilde{a}_t^i| \leq \epsilon_1^i$ is easy to enforce by clipping the original action a_t^i, so we first get an intermediate action $\hat{a}_t^i = \text{clip}(a_t^i, -\epsilon_1^i, \epsilon_1^i)$. Omitting the first constraint, Eq. 6 can be easily solved with $\lambda = \frac{\epsilon_2}{\sum_{i=1}^n \omega_i |\hat{a}_t^i|}$. The procedure only needs two steps. It is faster than the L_2 projection.

6. Experiments

In the experiment, we demonstrate the following properties of CTS^2:

- CTS^2 smooths multivariate time series with similar smoothness as traditional methods. Moreover, it never violates the constraints, while traditional methods cannot.
- CTS^2 smooths the time series across various scenarios with reusable hyperparameters.



(a) Original time series (b) Scaled time series

Figure 5: CTS^2 performance for synthetic time series with Gaussian noise.

- CTS^2 outputs the smooth data points with a very short delay. It is fair to say CTS^2 can achieve real-time smoothing after the policy learning.

We conduct experiments and compare CTS^2 with the widely used Savitzky–Golay (SG) Filter Savitzky and Golay (1964) and Fourier Filter (FF) Wesseling (1991) on both synthetic and real-world datasets. All policy neural networks use the same structure. They consist of two fully connected layers with 64 and 32 nodes, respectively. Training a policy can be time-intensive, but once trained, it can produce smooth data points in milliseconds in the inference phase. The source code is provided in supplementary material.

6.1. Synthetic results

We first run the experiment in synthetic time-series. We generate three-dimensional time series from optimal sin and cos functions and then add Gaussian and uniform random noise separately to the time series. For each dimension, we generate 200 sequential points, which we define as one episode. We segment these 200 points to four different slots, e.g. $0 \sim 49, 50 \sim 99, 100 \sim 149, 150 \sim 199$. For each slot, we either generate from sin function or cos function. In such way, we can generate time series with different shapes by combing sin and cos functions. The optimal sin and cos function range in $[-1, 1]$ and the noise range in $[-0.1, 0.1]$. The Gaussian noises are truncated if they are out of the range.

We follow the formulation as discussed in Sec. 4 to construct the CMDP. The policy actions are restricted with the flowing predefined constraints:

$$|a_t^i| \leq 0.1, i = x, y, z,$$
$$2|a_t^x| + |a_t^y| + |a_t^z| \leq 0.3.$$

In other words, the data correction cannot exceed 0.1 for each individual data point and their weighted sum cannot exceed 0.3.

Figure 5 shows the smoothing results for time series with Gaussian noise when the sliding window size is 5 and the policy is updated with 500 iterations, each containing

Table 1: Performance of different methods

Methods	Sum of angles	Constraints validation ratio	Sliding window size	Training time
CTS^2	1862.42	**0**	**5**	**118 minutes**
L_2 Projection	1862.15	0	5	134 minutes
SG(5,1)	1848.82	0.035	5	-
SG(45,5)	1863.90	0.05	45	-
FF	**1865.15**	0.28	whole episode	-

50 000 episodes. In next section, we will explore more hyperparameters. Specifically, we show the original smoothing results for all three dimensions in Fig. 5(a)subfigure. To be clear, we highlight the key advantages of CTS^2 in Fig. 5(b)subfigure by scaling the figures . We can see that CTS^2 almost achieves the same curves as the optimal lines of sin and cos functions and it satisfies all the constraints. Since the sliding window size is 5, CTS^2 takes only two delay points to inference the smooth data point.

To be fair, we set the sliding window size of SG to be 5 as well and the fitting order is 1 (we search for order with 1, 2, 3, and 4. The order 1 gets best results). We can see that SG(5,1) cannot smooth the curve well enough and it violates the constraints in some cases (e.g. in the third dimension, the point indexed with 20 violates the constraint $|a^z_{20}| = 0.144 \geq 0.1$). Although we can get a smoother time series with SG using a larger window size (e.g, 45), it increases the number of delay points, which runs counter to the real-time requirement.

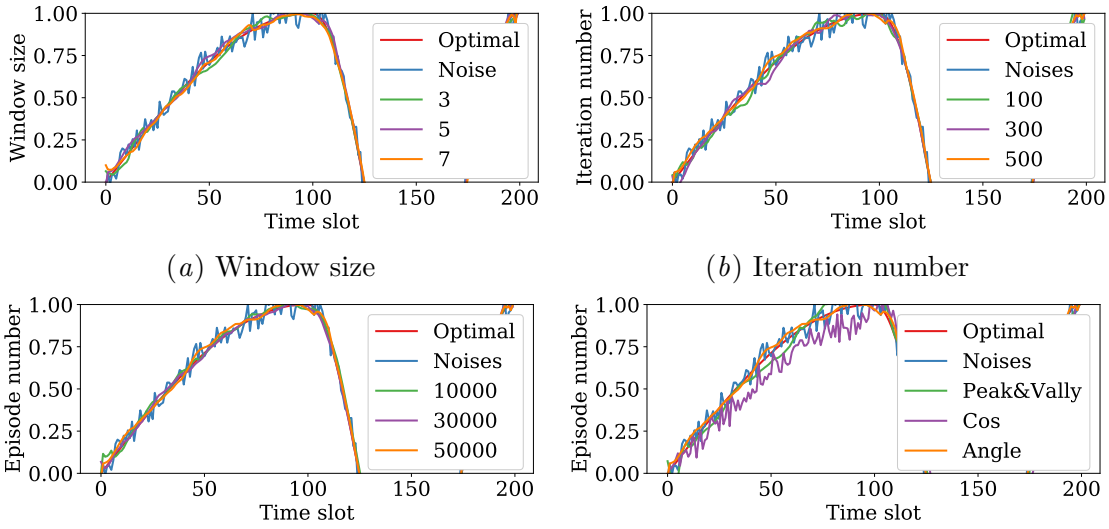
It appears that the FF smooths the curve by removing the highest 5 frequencies (we search for 5, 10, 15, 20 and the parameter 20 gives us the best results), however, it violates the constraints in most cases, For example, the first-dimensional point indexed with 185 and the second-dimensional points indexed between $[0, 25]$ violate the constraints. Worse still, FF does post-processing by using the whole episode, which is not real-time as well.

CTS^2 works well for time series with uniform noise as well as the Gaussian ones. It also works well in different time series shapes. We omitted the figures since the page limitation. We use the same parameters to process the time series with different noises and different shapes separately. The parameters of CTS^2 can be easily reused to new scenarios and we do not need to take effort to fine-tune the parameters manually. On the contrary, we have to fine tune parameters for SG (e.g. window size and order number) and FF (e.g. number of clipping frequency) in different scenarios to get desirable results, which is time consuming.

In addition, we run more experiments with time series with Gaussian noise and summarize the statistical results in Table 1; we compare the following aspects of the algorithms:

- **Sum of angles** is the metric to quantify the smoothness of a time series. For each episode, we sum all the angles constructed by any continuous tree points (see Fig. 2). The higher the sum, the smoother the time series.

- **Constraints validation ratio** computes the percentage of data points that violate the predefined constraints.



(a) Window size (b) Iteration number

(c) Episode number (d) Reward metric

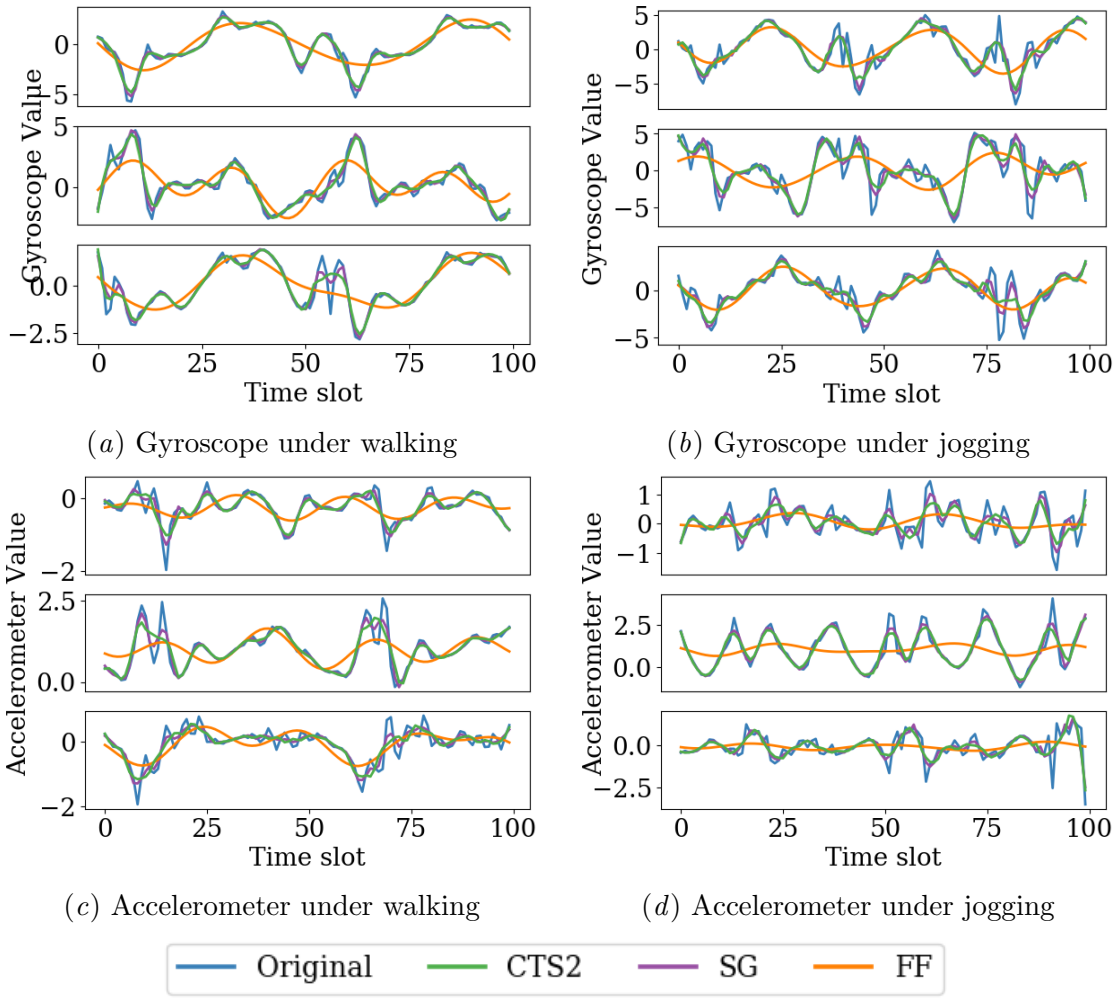
Figure 6: The impact of different hyperparameters

- **Sliding window size** represents the number of data points we need to process the time series in each step. Usually, the smaller the better. A smaller size reduces the number of delayed data points, which aids in smoothing the time series in real-time.
- **Training time** is to show the computational improvement versus the standard $L2$ projection layer Pham et al. (2018), as explained in Sec. 5.1. Our evaluation is supported with the Intel(R) Xeon(R) Platinum 8168 processor and we use 72 parallel threads to do the data sampling in the training phase.

Those metrics are evaluated with different episodes in different shapes and we use same hyperparameters (e.g. window size, iteration number, episode number in each iteration) for CTS^2; the average results can be seen in Table 1. We can see that the CTS^2 and L_2 projections achieve a fair high sum of angles and never violate the constraints. In addition, they can output the smooth time series with a window size of 5, which only contains two delay points. Moreover, CTS^2 runs faster than the L_2 projection since it reduces the complex QP in each step by doing a linear approximation.

6.2. Hyperparameter Tuning

In this section, we evaluate the performance of CTS^2 under different hyperparameters, e.g. window size, iteration number and episode number in each iteration. Moreover, we also show the performance under different smoothness metric. In the evaluation, we use synthetic data with Gaussian noise and we do grid search for the hyperparameters. The results are shown in Fig 6. We demonstrate the results for one demonstration. We can conclude that we can achieve fair good but with some error results when the window size is 3 and the window size is the larger the better. The window size with 7 is not necessary. For the iteration number and episode number, we can get desirable results when they are 500 and 50 000 separately. Smaller numbers are not quit enough. We also evaluate with



(a) Gyroscope under walking

(b) Gyroscope under jogging

(c) Accelerometer under walking

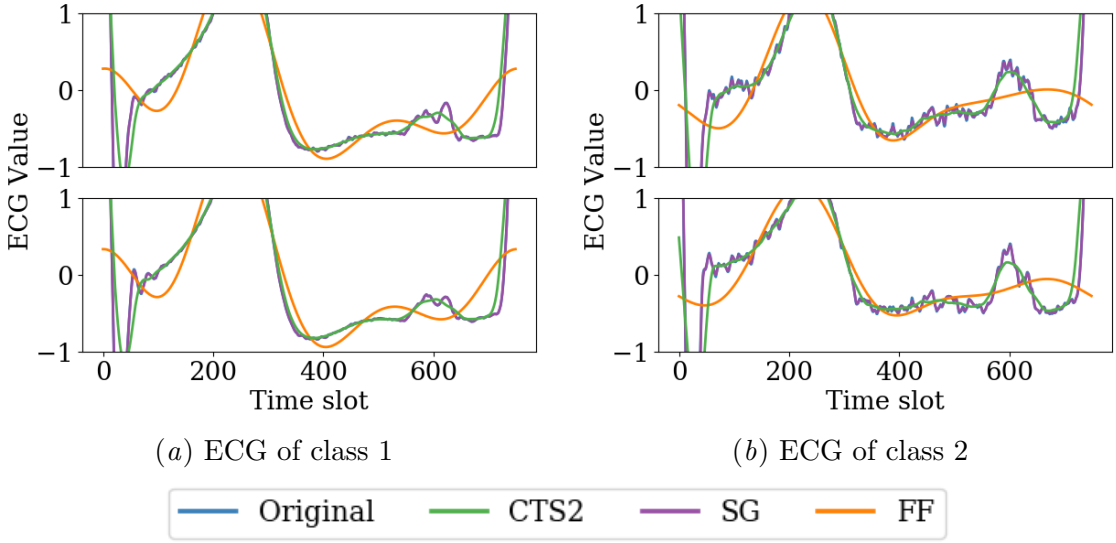
(d) Accelerometer under jogging

Figure 7: CTS^2 performance for Gyroscope and Accelerometer time series

different reward metrics: number of peak and valley, the cos value and the angle, we can see the angle metric can get the best results.

6.3. Real-world results

We use the open-source datasets Malekzadeh et al. (2019) and Bagnall et al. (2017) to evaluate CTS^2. Malekzadeh et al. (2019) includes time series data generated by gyroscope and accelerometer sensors. A total of 24 volunteers with different age, gender, weight and height were invited to collect data while performing different activities: go downstairs, go upstairs, walking, jogging, sitting, and standing. The data was collected with iPhone 6s smartphones which are common and widely used devices. Bagnall et al. (2017) is a dataset that contains large amount of different time series, e.g. data of ECG, traffic, earthquake, etc. It is originally designed for time series classification. The data could be univariate or multivariate. They also have different lengths, sizes, and classes. CTS^2 performance well on most of them. To save page space, we demonstrate results with dataset



(a) ECG of class 1 (b) ECG of class 2

Figure 8: CTS^2 performance for ECG time series under different classes

'NonInvasiveFetalECG'. The dataset was collected corresponding to the record of the ECG from left and right thorax, so the data contains two dimensions. It also includes 42 classes. Furthermore, the training size is 1800 and the test size is 1965. Since our purpose is doing smoothing but not classification, so we use all 3500 time series to train our model and left 265 time series to test the performance. For each time series, it contains 750 data points. Before evaluating our CTS^2, we do data augmentation by dividing the long time series into small slots. Each slot overlaps with near slots by shifting one point. Therefore, we got enough data to train the CTS^2.

For the gyroscope and accelerometer data, the actions are restricted with:

$$|a^n| \leq 0.1 * max(|O_t^n|), n = x, y, z,$$
$$2|a^x| + |a^y| + |a^z| \leq 0.1 * (max(|O_t^x|) + max(|O_t^y|) + max(|O_t^z|)).$$

where $max(|O_t^n|)$ is the maximal absolute value of any O_t^n.

Similarly, the ECG action is restricted with:

$$|a^n| \leq 0.1 * max(|O_t^n|), n = x, y,$$
$$2|a^x| + 3|a^y| \leq 0.2 * (max(|O_t^x|) + max(|O_t^y|))).$$

The smoothing results of gyroscope, accelerometer and ECG time series are shown in figure 7 and 8. To make a better demonstration, we limit the value of ECG to $[-1, 1]$. Moreover, CTS^2 use the same hyperparameters as learned form Sec. 6.2 in different scenarios, which further demonstrate that the parameters are reusable. For SG and FF, we grid search parameters in each case. Compared to the SG and FF, CTS^2 achieves an equivalently smooth performance while satisfying all constraints. Moreover, CTS^2 only need 2 delay data points to output smoothing ones, while SG and FF often need more than that.

7. Discussion

To the best of our knowledge, CTS^2 is the first work that smooths the time series while considering data correction constraints and it can be applied to general types of time series. One limitation compared with existing methods is that the RL policy must be pre-trained with a fair amount of data. A way to alleviate this concern is to train the policy with offline data; regardless, most time series data is not expensive to obtain. Additionally, there are a number of areas in RL that deal with the sample efficiency problem, such as efficient exploration Badia et al. (2020) and meta-learning Gupta et al. (2018). Applying these methods to CTS^2 would be an interesting future direction. Another concern people may have is that CTS^2 still needs a fair amount of hyperparameters, which is true. However, the hyperparameters can be reused to different scenarios as shown in Sec. 6.2, which means we do not need to take efforts to hand tuned for them across various scenarios.

8. Conclusion

In this paper, we propose a **C**onstrained reinforcement learning-based **T**ime **S**eries **S**moothing method, CTS^2, to smooth time series while considering data correction constraints. Specifically, we formulate the constraint smoothing problem as a CDMP and add a linear policy network layer to project all infeasible actions into feasible ones. CTS^2 ensures that the data correction never violates the constraints, whereas traditional methods cannot. Meanwhile, the smoothness achieved with CTS^2 is equivalent to traditional smoothing methods. Another benefit of CTS^2 is that the parameters can be easily reused to different scenarios and it is not necessary to manually adjust each parameter. In addition, CTS^2 smooths time series with a short delay. Although CTS^2 takes some data to learn the policy, it can smooth time series in real time once the policy has been learned.

Acknowledgments

The work was partially supported by NSF through grants IIS-1838207, CNS 1901218, OIA-2040680, 2134901, and USDA-020-67021-32855.

References

Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, et al. Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038*, 2020.

A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31:606–660, 2017.

Abhinav Bhatia, Pradeep Varakantham, and Akshat Kumar. Resource constrained deep reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 610–620, 2019.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Jian Cheng, Yu Yang, Niaoqing Hu, Zhe Cheng, and Junsheng Cheng. A noise reduction method based on adaptive weighted symplectic geometry decomposition and its application in early gear fault diagnosis. *Mechanical Systems and Signal Processing*, 149:107351, 2021.

Luke Froeb and Robert Koyak. Measuring and comparing smoothness in time series the production smoothing hypothesis. *Journal of econometrics*, 64(1-2):97–122, 1994.

Abhishek Gupta, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Unsupervised meta-learning for reinforcement learning. *arXiv preprint arXiv:1806.04640*, 2018.

Xiao Hu, Daniel Olesen, and Per Knudsen. Gyroscope aided video stabilization using nonlinear regression on special orthogonal group. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2707–2711. IEEE, 2020.

Emily J Huang and Jukka-Pekka Onnela. Augmented movelet method for activity classification using smartphone gyroscope and accelerometer data. *Sensors*, 20(13):3706, 2020.

Mohammad Rafiqul Islam and Nguyet Nguyen. Comparison of financial models for stock price prediction. *Journal of Risk and Financial Management*, 13(8):181, 2020.

Karl-Dirk Kammeyer and Kristian Kroschel. *Digitale signalverarbeitung*, volume 5. BG Teubner Verlag, 1998.

Christos Katris. A time series-based statistical approach for outbreak spread forecasting: Application of covid-19 in greece. *Expert Systems with Applications*, 166:114077, 2021.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Yongshuai Liu, Jiyu Chen, and Hao Chen. Less is more: Culling the training set to improve robustness of deep neural networks. In *International Conference on Decision and Game Theory for Security*, pages 102–114. Springer, 2018.

Yongshuai Liu, Jiaxin Ding, and Xin Liu. A constrained reinforcement learning based approach for network slicing. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pages 1–6. IEEE, 2020a.

Yongshuai Liu, Jiaxin Ding, and Xin Liu. Ipo: Interior-point policy optimization under constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4940–4947, 2020b.

Yongshuai Liu, Jiaxin Ding, and Xin Liu. Resource allocation method for network slicing using constrained reinforcement learning. In *2021 IFIP Networking Conference (IFIP Networking)*, pages 1–3. IEEE, 2021a.

Yongshuai Liu, Avishai Halev, and Xin Liu. Policy learning with constraints in model-free reinforcement learning: A survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, 2021b.

Li Lu and Yongshuai Liu. Safeguard: User reauthentication on smartphones via behavioral biometrics. *IEEE Transactions on Computational Social Systems*, 2(3):53–64, 2015.

Li Lu, Lingshuang Liu, Muhammad Jawad Hussain, and Yongshuai Liu. I sense you by breath: Speaker recognition via breath biometrics. *IEEE Transactions on Dependable and Secure Computing*, 17(2):306–319, 2017.

Mohammad Malekzadeh, Richard G. Clegg, Andrea Cavallaro, and Hamed Haddadi. Mobile sensor data anonymization. In *Proceedings of the International Conference on Internet of Things Design and Implementation*, IoTDI '19, pages 49–58, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6283-2. doi: 10.1145/3302505.3310068. URL http://doi.acm.org/10.1145/3302505.3310068.

Ghulam Muhammad, M Shamim Hossain, and Neeraj Kumar. Eeg-based pathology detection for home health monitoring. *IEEE Journal on Selected Areas in Communications*, 2020.

Tu-Hoa Pham, Giovanni De Magistris, and Ryuki Tachibana. Optlayer-practical constrained optimization for deep reinforcement learning in the real world. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6236–6243. IEEE, 2018.

Hong Qu, Xiurui Xie, Yongshuai Liu, Malu Zhang, and Li Lu. Improved perception-based spiking neuron learning rule for real-time user authentication. *Neurocomputing*, 151: 310–318, 2015.

Abraham Savitzky and Marcel JE Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8):1627–1639, 1964.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

Omkar Singh, Ramesh Kumar Sunkaria, et al. Ecg signal denoising based on empirical mode decomposition and moving average filter. In *2013 IEEE International Conference on Signal Processing, Computing and Control (ISPCC)*, pages 1–6. IEEE, 2013.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

P Wesseling. A survey of fourier smoothing analysis results. In *Multigrid Methods III*, pages 105–127. Springer, 1991.