

Supplements to “Convolutional Non-Homogeneous Poisson Process and its Application to Wildfire Ignition Risk Quantification for Power Delivery Networks”

Guangzhou Wei¹

¹H. Milton Stewart School of Industrial & Systems Engineering,
Georgia Institute of Technology

Feng Qiu²

²Advanced Grid Modeling, Optimization and Analytics,
Argonne National Lab

Xiao Liu¹

¹H. Milton Stewart School of Industrial & Systems Engineering,
Georgia Institute of Technology

Supplement A: Derivation of $h(i, t)$

Based on the \mathcal{NC} operator (5) and the model (3), $h(i, t)$ in (4) can be written as

$h(i, t) = \sum_{n=1}^{\infty} \xi^n \mathcal{NC}^{(n)}\{c\}(i, t - n\Delta)$. The derivation is given as follows:

$$\begin{aligned}
 h(i, t) &= \xi \mathcal{NC}\{\log \lambda\}(i, t - \Delta) = \xi \mathcal{NC}\{c\}(i, t - \Delta) + \xi \mathcal{NC}\{h\}(i, t - \Delta) \\
 &= \xi \mathcal{NC}\{c\}(i, t - \Delta) + \xi^2 \mathcal{NC}\{\mathcal{NC}\{\log \lambda\}\}(i, t - 2\Delta) \\
 &= \xi \mathcal{NC}\{c\}(i, t - \Delta) + \xi^2 \mathcal{NC}^{(2)}\{\log \lambda\}(i, t - 2\Delta) \\
 &= \xi \mathcal{NC}\{c\}(i, t - \Delta) + \xi^2 \mathcal{NC}^{(2)}\{c\}(i, t - 2\Delta) + \xi^2 \mathcal{NC}^{(2)}\{h\}(i, t - 2\Delta) \quad (1) \\
 &\dots \\
 &= \sum_{n=1}^{\infty} \xi^n \mathcal{NC}^{(n)}\{c\}(i, t - n\Delta) + \lim_{n \rightarrow +\infty} \xi^n \mathcal{NC}^{(n)}\{h\}(i, t - n\Delta) \\
 &= \sum_{n=1}^{\infty} \xi^n \mathcal{NC}^{(n)}\{c\}(i, t - n\Delta).
 \end{aligned}$$

Supplement B: Representation of cNHPP using the Architecture of RNN

In Appendix B, we show that the proposed cNHPP can be represented using the architecture of a Recurrent Neural Network (RNN). Note that, the proposed linear model in (13) suggests that the intensity function at time t depends on the historical and current covariate information. Such a structure enables us to draw a connection between the proposed model and an RNN—a feed-forward neural network with an input layer, a recurrent layer, and an output layer.

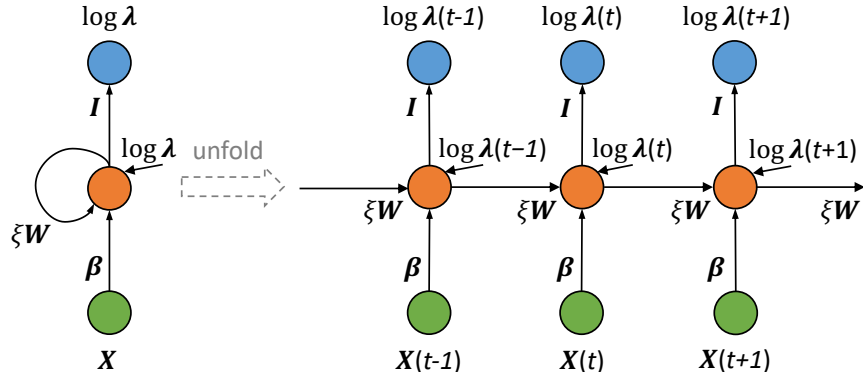


Figure 1: The RNN representation of the proposed cNHPP.

The RNN representation of the proposed model is shown in Figure 1. Motivated from the proposed model (13), the input layer takes the input of the $N \times (q + 1)$ covariate matrix $\mathbf{X}(\cdot)$, while the output layer outputs the intensity functions $\log \lambda(\cdot)$. The recurrent layer consists of the recurrent edge and hidden state, in which the recurrent edge repeatedly feeds the previous hidden state to the current state. The unfolded version of this RNN has a many-to-many structure (i.e., many inputs and outputs) that can be represented by the following forward-propagation equations (Fan et al., 2021):

$$\mathbf{h}(t) = \xi \mathbf{W} \mathbf{h}(t-1) + \mathbf{X}(t) \beta, \quad \mathbf{o}(t) = \mathbf{I} \mathbf{h}(t) \triangleq \log \lambda(t), \quad (2)$$

where $\mathbf{h}(t)$ and $\mathbf{o}(t)$ are, respectively, the hidden and output states at time t , the hidden-to-output connection \mathbf{I} is an identity matrix in our case, and the weight vector $\boldsymbol{\beta}$ and weight matrix $\xi\mathbf{W}$ are the input-to-hidden and hidden-to-hidden connections respectively. It is immediately seen that (2) implies that the intensity function at time t is given by the sum of $\mathbf{X}(t)\boldsymbol{\beta}$ (i.e., the current effects of covariates) and $\xi\mathbf{W}\mathbf{h}(t-1) = \xi\mathbf{W}\log\boldsymbol{\lambda}(t-1)$ (i.e., the cumulative effects through spatio-temporal dependency), which is exactly the same as the proposed model (13). Because this RNN representation is originated from the proposed model, we call it the model-inherited RNN (**mRNN**) in this paper.

In the **mRNN** (2), the parameters to be learned are the same as in the model (13), i.e., $\boldsymbol{\theta} = (\xi, \beta_0, \beta_1, \dots, \beta_q)^T$, which can be estimated through maximizing the log-likelihood function $\ell(\boldsymbol{\theta})$ in (16) using the back-propagation through time (BPTT). The BPTT provides a computational procedure for obtaining the gradients of the unknown parameters, which can be used to train the RNN with gradient-based techniques (Goodfellow et al., 2016).

Following this **mRNN** representation, computing $\{\log\boldsymbol{\lambda}(t)\}_{t=0}^T$ primarily involves multiplications between $\xi\mathbf{W}$ and $\{\mathbf{X}(t)\boldsymbol{\beta}\}_{t=0}^T$. This process requires $T+1$ multiplications between an $N \times N$ matrix and an N -vector. On the other hand, for the representation (13) in the main manuscript, $\log\boldsymbol{\lambda}(t) = (\sum_{k=0}^K \xi^k \mathbf{W}^k \mathbf{X}(t-k\Delta))\boldsymbol{\beta}$. To obtain $\log\boldsymbol{\lambda}(t)$, the main computation involves K multiplications between $\xi^k \mathbf{W}^k$ and $\mathbf{X}(t-k\Delta)$. Then, to obtain $\{\log\boldsymbol{\lambda}(t)\}_{t=0}^T$, we need $(T+1) \times K$ multiplications between an $N \times N$ matrix and an $N \times (p+1)$ matrix. As a result, leveraging the **mRNN** representation can reduce the computational cost. For illustrative purposes, we compare the computational time of evaluating $\{\log\boldsymbol{\lambda}(t)\}_{t=0}^T$ respectively based on (13) in the manuscript and its RNN representation. To ensure a fair comparison, we use PyTorch with the tensor data type to calculate $\{\log\boldsymbol{\lambda}(t)\}_{t=0}^T$ for both approaches. The dimensions of \mathbf{W} and $\mathbf{X}(t)$ are kept the same as in

Section 3.2, and the comparison result is shown in Figure 2. We see that the computational time for the proposed model increases almost linearly with the increase of the truncation number K . Even for $K = 1$, the computation cost of the proposed model is still higher than that of the **mRNN**. In the application example presented in Section 3.2, we set $K = 7$ and the computational time associated with the statistical model is approximately 12 times longer than using its RNN representation.

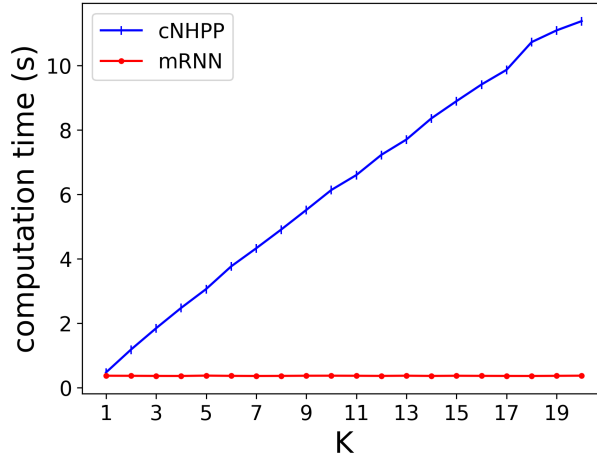


Figure 2: Comparison of the computational time of evaluating $\{\log \boldsymbol{\lambda}(t)\}_{t=0}^T$ respectively based on (13) in the manuscript and its RNN representation.

Supplement C

In Section 3.3.1, we implement the RNN with `PyTorch`, and employ the Adam optimizer (learning rate = 0.001) to train the model. Convergence of the loss function and unknown parameters are shown in the figure below where a total number of 20,000 epochs are trained.

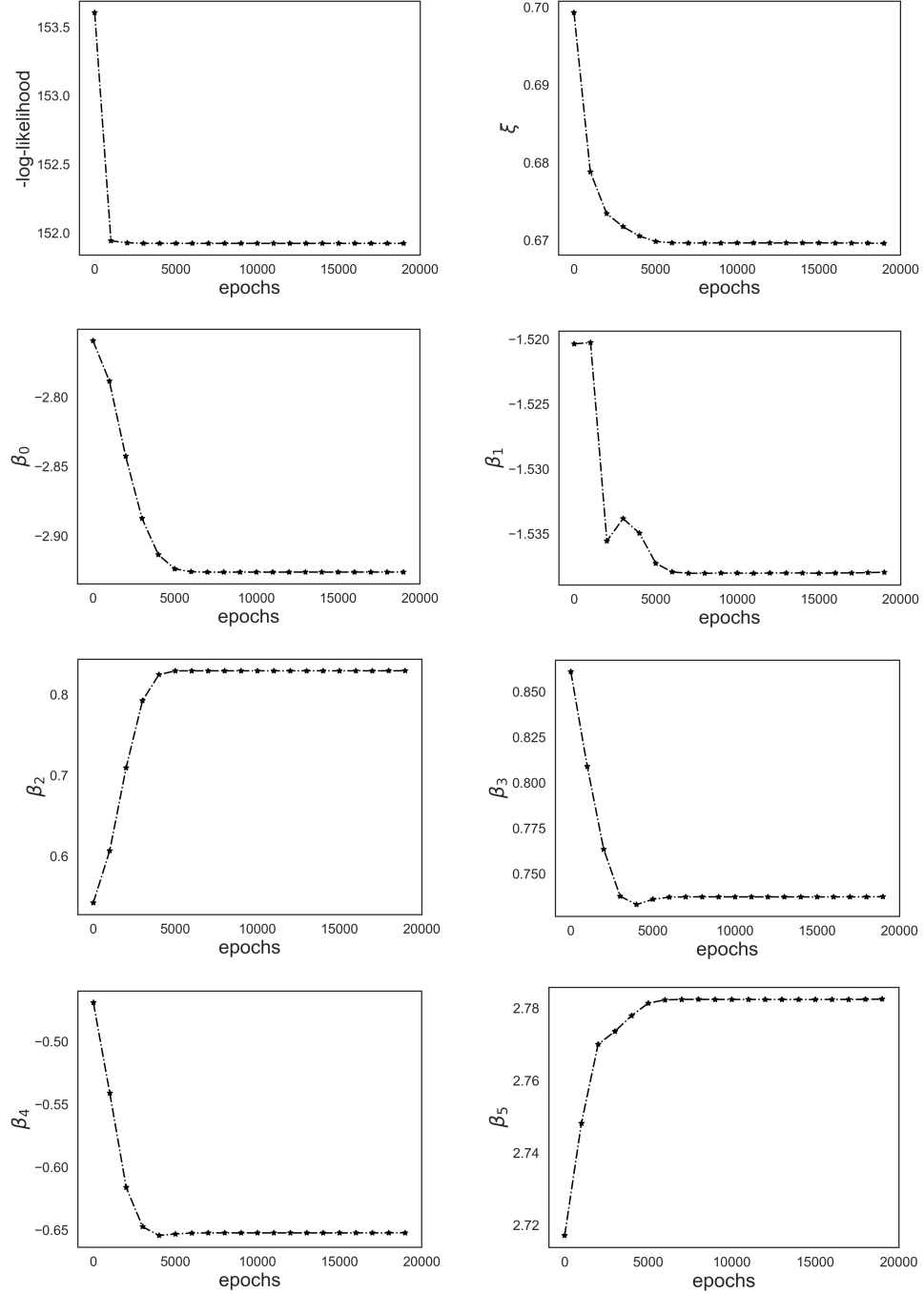


Figure 3: Convergence of the loss function and parameters for mRNN.

References

Fan, J., Ma, C., and Zhong, Y. (2021). A selective overview of deep learning. *Statistical Science*, 36(2):264.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.