# Appendix for Concurrent Multi-Label Prediction in Event Streams

## A Model Implementation and Training Details

We implement the transformer module for TCMBN with partial adaption from the Transformer Hawkes Process Zuo et al. [2020] [1]. Training of TCMBN is performed by mini-batch stochastic gradient descent given training event sequences, validated by dev subset and test on the test subset. The ratio of the three sets is 60-20-20. Adam [Kingma and Ba, 2014] optimizer is used to train our model. The experiment setting for hyperparameters in TCMBN for multi-label prediction is given in Table 1. Baseline model Transformer-LG is performed with the same set of hyperparameters. Baseline model RNN-MBN is implemented partially from Intensity-free Point Process Shchur et al. [2019] yet with CMBN module of 64 components and is trained with default setting [2]. (While other configurations for the above baselines are possible, their empirical performance is inferior.) The best threshold $\lambda = \{0.1, 0.01, 0.001\}$ for each model is chosen based on dev set when domain knowledge loss is injected. We inject such loss in the modeling of synthetic data (PM and HM) and Defi. We set each $q$ to be 1 for the above datasets. In addition, we set $\lambda = 0$ for the rest of datasets. All our experiments are performed on a private server with TITAN RTX GPU. We attach our code in the supplementary material package.

## B Synthetic Sequence Generation

For synthetic data generation, we generate 1000 sequences according to the following procedure. The dynamics for a generated event sequence is driven by either a Poisson or a Hawkes Process with exponential kernel. We use a constant rate of 0.2 and time horizon of (0,100] to generate Poisson timestamps. We use baseline rate of 0.2, decay constant of 0.5, and adjacency matrix of [0.1] as described in the tick library of Hawkes model with exponential kernel.[3]

We partition 5 labels into two groups: one group of 2 and the group of 3. The un-normalized probability of co-occurence of labels for each event epoch depends only on the number of historical occurrences. Specifically, for each label, we check the counts of each specific label in each group in the history and obtain an un-normalized probability.

[1] https://github.com/SimiaoZuo/Transformer-Hawkes-Process

[2] https://github.com/shchur/ifl-tpp

[3] https://x-datainitiative.github.io/tick/modules/hawkes.html

Table 1: Hyperparameters for TCMBN. n_layers: the number of blocks for multi-headed self-attention module; d_model: the dimension of the value vector after attention has been applied; n_head: the number attention heads; d_inner: the dimension of the hidden layer of the feed forward neural network; d_v: the dimension of the value vector; d_k : the dimension of the key vector;num_comps : number of mixture of Bernoulli components.

| Parameter | PM/HM | Synthea | Dunnhumby | Mimic3 | Defi |
|---|---|---|---|---|---|
| batch_size | 8 | 8 | 8 | 16 | 8 |
| n_head | 4 | 4 | 6 | 1 | 1 |
| n_layers | 1 | 1 | 1 | 1 | 1 |
| d_model | 64 | 64 | 32 | 64 | 32 |
| d_inner | 32 | 32 | 16 | 32 | 16 |
| d_v | 32 | 32 | 16 | 32 | 16 |
| d_k | 32 | 32 | 16 | 32 | 16 |
| dropout | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 |
| epoch | 100 | 100 | 100 | 100 | 100 |
| num_comps | 64 | 64 | 64 | 64 | 64 |
| learning_rate | 0.0002 | 0.002 | 0.01 | 0.0015 | 0.006 |

We generate each event label according to the normalized probability distribution which serves as mean of multivariate Bernoulli distribution with diagonal covariance. Together we combine timestamps and labels to form our PM and HM datasets by performing 5 simulation, each of which consists of 1000 event sequences. We show more details in Algorithm 1 with pseudo code in Python for PM; HM can be obtained similarly.

# C Models for Structural Learning

## C.1 Synthetic Data

We describe the procedure to simulate the binary data. From the Ising model below, when the node dimension M = 5, 6 out of 10 parameters $\Theta_{ij}$ are randomly chosen with mixed coupling, i.e. $\Theta_{ij} = \pm 0.5$ with equal probability [Ravikumar et al., 2010].

$$P(x_1, x_2, ..., x_M) = \exp(\Theta_0 + \sum \Theta_i x_i + \sum_{i<j} \Theta_{ij} x_i x_j) \tag{1}$$

Then the probability mass function of categorical distribution (of $2^M$ configurations) were calculated and samples were generated accordingly [De Canditiis, 2019]. For M = 8, 17 out of 28 parameters are randomly chosen with mixed coupling. For large M, Gibbs sampling can be performed. The Matlab implementation is available online[4].

[4]https://www.iac.cnr.it/ danielad/software.html

**Algorithm 1:** Poisson Concurrent Event Generator

**Input:** Given label set: $\mathcal{L}$, $|\mathcal{L}| = 5$, A generated univariate Possion Process data $D_t = \{t_i\}_{i=1}^{n}$

**Output:** A generated multi-label sequence $D_y$

$\mu$ = [0.5,0.5,0.5,0.5,0.5]
$\mu_{new}$ = [0,0,0,0,0]
$D_y$ = []
**for** $i \leftarrow 1$ **to** $n$ **do**
    sample = diag_bernoulli_generator($\mu$)
    **while** *sum(sample) == 0* **do**
        sample = diag_bernoulli_generator($\mu$)
    **end**
    $D_y$.append(sample)
    counts = np.sum(np.array($D_y$),axis=0)
    **if** *(counts[0]+counts[1]) != 0* **then**
        $\mu_{new}$[0] = counts[0]/ (counts[0]+counts[1])
        $\mu_{new}$[1] = counts[1]/ (counts[0]+counts[1])
    **end**
    **if** *(counts[2]+counts[3]+counts[4]) != 0* **then**
        $\mu_{new}$[2] = counts[2]/ (counts[2]+counts[3]+counts[4])
        $\mu_{new}$[3] = counts[3]/ (counts[2]+counts[3]+counts[4])
        $\mu_{new}$[4] = counts[4]/ (counts[2]+counts[3]+counts[4])
    **end**
    **for** $k \leftarrow 1$ **to** $\mathcal{L}$ **do**
        **if** *$\mu_{new}$[k] != 0* **then**
            $\mu$[k] = $\mu_{new}$[k]
        **end**
    **end**
**end**
**Return:** $D_y$

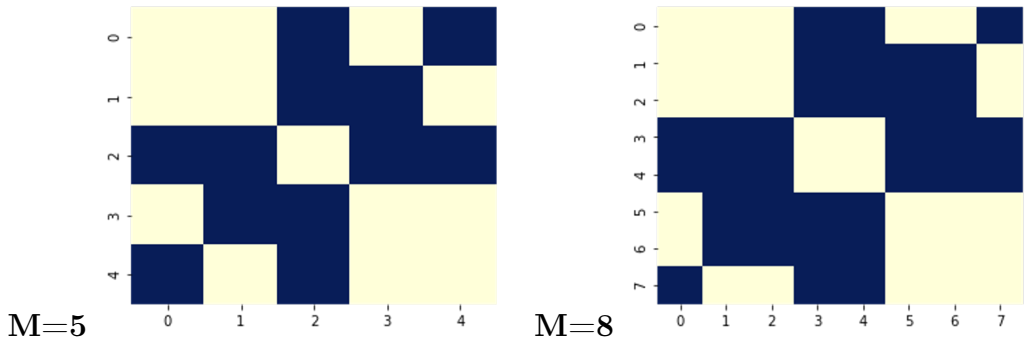


Figure 1: Ground truth structures of simulated data for 5 nodes and 8 nodes. Each black block indicates $\Theta_{ij} \neq 0$, while each yellow indicates $\Theta_{ij} = 0$.

## C.2 (T)CMBN-LSSA Setting

We used CMBN component from TCMBN for binary structural discovery and fed the IID data into the model to learn a covariance matrix on Pytorch. The learned matrix is fed into LSSA, a program on Matlab to compute the estimated precision matrix. Table 2 shows the parameters we set for the two-step procedure.

We obtained the covariance matrices on Dunnhumby dataset at the two timestamps $i = 1$ and $i = 101$ , solved the LSSA with penalty parameter $\gamma = 0.000005$ on Matlab.

Table 2: Parameters of (T)CMBN-LSSA on simulation experiments and Illinois Votes dataset

| nodes | samples | mixtures | epochs | batchsize | penalty $\gamma$ |
|---|---|---|---|---|---|
| 5 | 150 | 2 | 5000 | 150 | 0.001 |
| 5 | 300 | 2 | 5000 | 300 | 0.001 |
| 5 | 500 | 2 | 10 | 500 | 0.001 |
| 8 | 150 | 4 | 5000 | 150 | 0.0005 |
| 8 | 300 | 4 | 5000 | 150 | 0.001 |
| 8 | 500 | 4 | 100 | 500 | 0.001 |
| 18(votes) | 1246 | 8 | 200 | 150 | 0.001-0.015 |

## C.3 L-N-M, M-I and NeurISE

We ran L-N-M and M-I on Matlab according to the work by De Canditiis [2019] and the Matlab program by the author. [4] Specifically, we used L-N-M to indicate Wainwright-max procedure [Höfling and Tibshirani, 2009] to symmetricize the logistic neighborhood approach :the estimated quantities are not symmetric, i.e. $\widehat{\Theta}_{ij} \neq \widehat{\Theta}_{ji}$. In addition, a 10-fold CV procedure was applied to select the best penalty parameter for each node.

The M-I model evaluates the empirical covariance matrix, numerically invert it and perform a threshold on elements of the precision matrix. The thresholds we used in the experiments are 0.2 and 0.4 as two quantiles to zero out the entries of $\widehat{\Theta}$.

We trained NeurISE[Lokhov et al., 2020] on Google Colab with the default implementation and setting by the authors.[5] The produced $\widehat{\Theta}$ were symmetricized by the Wainwright-max procedure. We also used the two quantiles 0.2 and 0.4 to zero out the entries of $\widehat{\Theta}$.

## C.4 Illinois Votes

We analyze the different votes of the Illinois House of Representatives during the period of the 114-th and 115-th US Congresses (2015-2019), which is available at voteview.com Lewis et al. [2019]. The Illinois House of Representatives has 18 seats (one per district), each one corresponds to a US Representative belonging to the Democratic or the Republican parties. An imputed version is available online[6] Le Bars et al. [2020]. As shown in Figure 2, the recovered graphs on the shrinkage path where we increase the penalty parameter from 0.001 to 0.015, are getting closer and closer to the real situation where representatives of Democratic party (blue) and representatives of Republican party (red) are more strongly "connected" within the party than outside. The two parties are connected by 10-th seat, a supercollaborator, a role that some representatives get by acting more independently and position themselves in the middle of the parties. This result agrees with early work Le Bars et al. [2020].

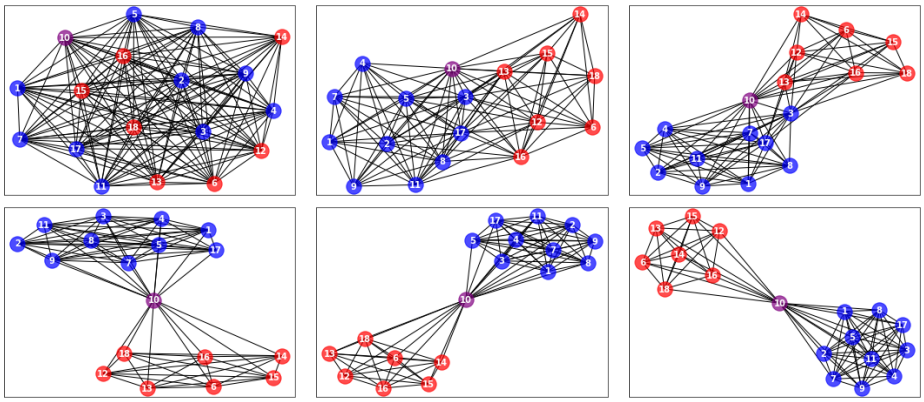

Figure 2: Recovered graphs for Illinois Votes data. Upper panel: From left to right the penalty parameter $\gamma$ is 0.003, 0.006, 0.008; Lower panel: From left to right the penalty parameter $\gamma$ is 0.011, 0.013, 0.015. When $\gamma = 0.011$, the Republican cluster (red) and the Democratic cluster (red) are formed, connected by a collaborator seat (purple).

[5] https://github.com/lanl-ansi/NeurISE
[6] https://github.com/BatisteLB/TVI-FL

# D Proof

## D.1 Theorem 1: Transformer with Temporal Encoding

Consider the type of transformer block described by Yun et al. [2019]. Our proof for a transformer with temporal encoding as a universal approximating function for any continuous function for sequence-to-sequence with compact support follows similarly as Proof of transformer with position encoding (Theorem 3 in Yun et al. [2019]). Without loss of generality, consider a sequence with timestamps $\{t_1, t_2, ..., t_n\}$ and let $t_i$ being integer values and $t_i < t_j$ for all $i < j$. We choose a $d$-dimensional Temporal encoding for the $n$ event epochs to be the following:

$$\mathbf{T} = \begin{bmatrix} 0 & t_2 - t_1 & \dots & t_n - t_1 \\ 0 & t_2 - t_1 & \dots & t_n - t_1 \\ \vdots & \vdots & & \vdots \\ 0 & t_2 - t_1 & \dots & t_n - t_1 \end{bmatrix}$$

This guarantees for all rows the coordinates are monotonically increasing and input values can be partition into cubes. The rest of the proof follows directly from proof of Theorem 3 by replacing $n$ with $t_n$ by performing quantization by feed-forward layers, contextual mapping by attention layers and function value mapping by feed-forward layers.

## D.2 Theorem 2: Existence of a Transformer Network that Separates Points

We prove Theorem 2 by contradiction. Given any function $f$ in the class of continuous sequence-to-sequence functions $\mathcal{F}$ that maps $\mathbf{X} \in R^{d \times L}$ to $\mathbf{H} \in R^{d \times L}$ with compact support. With loss of generality, consider $d = 1$. Suppose $\mathbf{X}_i \neq \mathbf{X}_j$ for some $i \neq j$ and $i, j \in \{1, 2, 3, 4, ..., L\}$. Given some $f \in \mathcal{F}$ that maps $\mathbf{X}$ to certain $\mathbf{H}^f$, where the $i, j$ entries differ by $|\mathbf{H}_i^f - \mathbf{H}_j^f|$. For any transformer that cannot separate $(\mathbf{X}_i, \mathbf{H}_i)$ and $(\mathbf{X}_j, \mathbf{H}_j)$, that is $\mathbf{H}_i = \mathbf{H}_j$. The transformer cannot approximate the function $f$ with $\epsilon$ distance since there is a always gap of $|\mathbf{H}_i^f - \mathbf{H}_j^f|$ with respect some entry-wise $l^p$ norm. Thus universal approximating property fails.

# References


Daniela De Canditiis. Learning binary undirected graph in low dimensional regime. *arXiv preprint arXiv:1907.11033*, 2019.

Holger Höfling and Robert Tibshirani. Estimation of sparse binary pairwise markov networks using pseudo-likelihoods. *Journal of Machine Learning Research*, 10(4), 2009.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Batiste Le Bars, Pierre Humbert, Argyris Kalogeratos, and Nicolas Vayatis. Learning the piece-wise constant graph structure of a varying ising model. In *Int. Conf. on Machine Learning*, pages 675–684. PMLR, 2020.

Jeffrey B Lewis, Keith Poole, Howard Rosenthal, Adam Boche, Aaron Rudkin, and Luke Sonnet. Voteview: Congressional roll-call votes database. *See https://voteview.com/(accessed 27 July 2018)*, 2019.

Andrey Y Lokhov, Sidhant Misra, Marc Vuffray, et al. Learning of discrete graphical models with neural networks. *arXiv preprint arXiv:2006.11937*, 2020.

Pradeep Ravikumar, Martin J Wainwright, and John D Lafferty. High-dimensional ising model selection using l1-regularized logistic regression. *The Annals of Statistics*, 38(3):1287–1319, 2010.

Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. Intensity-free learning of temporal point processes. *arXiv preprint arXiv:1909.12127*, 2019.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J Reddi, and Sanjiv Kumar. Are transformers universal approximators of sequence-to-sequence functions? *arXiv preprint arXiv:1912.10077*, 2019.

Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. Transformer hawkes process. In *International Conference on Machine Learning*, pages 11692–11702. PMLR, 2020.