Report

My work consisted of synthetic datasets generation and running the experiments of existing models (baseline + 3 alternatives) on them.

The root directory and location of all the experiments:
root = Cohortney/src/mds20_cohortney-main/

1. Data generation

a. Hawkes data generation

root/data_generation-v[1/2].ipynb

Resulting datasets location:
root/data/simulated_Hawkes/K_[m]_C_[n]*
root/data/simulated_Hawkes/tmp_sin_K_[m]_C_[n]_[0/1/2]**
root/data/simulated_Hawkes/tmp_trunc_K_[m]_C_[n]_[0/1/2]***
* m - number of clusters, n - number of event types (we took k=2..5, n=5)
, * datasets with sine-like and piecewise constant impact functions respectively. They were generated the same way as in the original paper[1]:

> We generate two synthetic data sets with various clusters using sine-like impact functions and piecewise constant impact functions respectively. In each data set, the number of clusters is set from 2 to 5. Each cluster contains 400 event sequences, and each event sequence contains 50 ($= M_n$) events and 5 ($= C$) event types. The elements of exogenous base intensity are sampled uniformly from $[0, 1]$. Each sine-like impact function in the k-th cluster is formulated as $\phi^k_{cc'} = b^k_{cc'}(1-\cos(\omega^k_{cc'}(t-s^k_{cc'})))$, where $\{b^k_{cc'}, \omega^k_{cc'}, s^k_{cc'}\}$ are sampled randomly from $[\frac{\pi}{5}, \frac{2\pi}{5}]$. Each piecewise constant impact function is the truncation of the corresponding sine-like impact function, i.e., $2b^k_{cc'} \times \text{round}(\phi^k_{cc'}/(2b^k_{cc'}))$.

b. Non-Hawkes data generation

root/data_generation-v[1/2].ipynb

Resulting datasets location:
root/data/simulated_non_Hawkes/K_[m]_C_[n]

Maybe, it was hard to beat the baseline (DHMP) because it was designed for specific datasets - Hawkes ones. The idea was to generate another type of datasets - non-Hawkes (non-additive, non-positive effect).

These data was generated using the idea of rejection sampling (generate some process and then take points from it with the probability lambda_i / lambda*) like in algorithm 2 in the article [2]. They make a random timestep and sample one point of each type and from all that satisfy the inequality with lambda, they take the earliest one. (it has a detailed description in section b2 of the article).
As for the form of the process itself - lambda_k (t), I took baseline[k_-1] + np.sin (t) * (1-np.cos (t + k)))) and put a Softplus on top for non-negativity.

Algorithm 2 Data Simulation (thinning algorithm)

Input: interval $[0, T]$; model parameters
$t_0 \leftarrow 0$; $i \leftarrow 1$
while $t_{i-1} < T$: ▷ *draw event i, as it might fall in [0,T]*
  for $k = 1$ **to** K : ▷ *draw "next" event of each type*
    find upper bound $\lambda^* \geq \lambda_k^i(t)$ for all $t \in (t_{i-1}, \infty)$
    $t \leftarrow t_{i-1}$
    repeat
      draw $\Delta \sim \text{Exp}(\lambda^*)$, $u \sim \text{Unif}(0, 1)$
      $t \mathrel{+}= \Delta$ ▷ *time of next proposed event*
    until $u\lambda^* \leq \lambda_k^i(t)$ ▷ *accept proposal with prob* $\frac{\lambda_k^i(t)}{\lambda^*}$
    $t_{i,k} \leftarrow t$
  $t_i \leftarrow \min_k t_{i,k}$; $k_i \leftarrow \text{argmin}_k t_{i,k}$ ▷ *earliest event wins*
  $i \leftarrow i + 1$
return $(k_1, t_1), \ldots (k_{i-1}, t_{i-1})$

2. The models

Let me briefly describe all compared models

a. Dirichlet Mixture Model of Hawkes Process[1] (DHMP)

It is our baseline. An implementation of the algorithm from the original paper (their implementation is available on Matlab).

The first version of this model (made during the course) had not correct basis functions. I made them adaptive, as in the original paper[3] (section 4.5, algorithm 2).

Algorithm 2 Selecting basis functions

1: **Input:** $\mathcal{S} = \{s_c\}_{c=1}^C$, residual's upper bound ϵ.
2: **Output:** Basis functions $\{\kappa_{\omega_0}(t, t_m)\}_{m=1}^M$.
3: Compute $\left(\sum_{c=1}^C N_c\sqrt{2\pi h^2}\right) e^{-\frac{\omega^2 h^2}{2}}$ to bound $|\hat{\lambda}(\omega)|$.
4: Find the smallest ω_0 satisfying $\int_{\omega_0}^{\infty} |\hat{\lambda}(\omega)| d\omega \leq \epsilon$.
5: The proposed basis functions $\{\kappa_{\omega_0}(t, t_m)\}_{m=1}^M$ are selected, where ω_0 is the cut-off frequency of basis function and $t_m = \frac{(m-1)T}{M}$, $M = \lceil \frac{T\omega_0}{\pi} \rceil$.

where $h = (\frac{4\hat{\sigma}^5}{3\sum_c N_c})^{0.2}$,
$\kappa_\omega(t, t_m) = \exp(-(t - t_m)^2/(2\sigma^2))$
$\omega = \sigma^{-1}$.

A detailed description is given in section 8.3.
My implementation of basis functions selection is here:
root/src/DMHP/HP.py

b. Convolutional Autoencoder (CAE)

Firstly, we pass the partitions obtained from Cohortney to encoder, get the latent features from it, then we pass the reshaped latent data to the Kmeans algorithm.

c. Deep Clustering

The training consists of the following steps. On each epoch of the training phase, deep features of sequences are computed with convnet. These features are used to cluster sequences using the KMeans algorithm. The top fully-connected classifier is then initialized and put onto the top of the model. The obtained pseudolabels are then used for supervised-manner training of the model.

d. Optimal Transport (OT)

It is an adaptation of a state-of-the-art approach for image unsupervised classification proposed in (Asano et al., 2020).
The original algorithm is a simultaneous clustering and representation learning:
Step 1: representation learning via cross-entropy minimization
Step 2: self-labelling (solve an optimal transport problem to obtain the pseudo-labels)
As a model, they used AlexNet and ResNet with multiple clustering heads. About this work, we got partitions obtained from Cohortney and then trained convolutional filters followed by linear layer and predicting linear layer on top of those partitions.

3. Experiments

root/test_[model_name]-[dataset_name].ipynb

The results of the experiments on synthetic data can be viewed in the following table. To compare the models, I used clustering purity.

DeepCluster:
A convolutional neural network with batch normalizations. For the main part of the model we use Adam optimizer and for the top layer, we use SGD optimizer with a momentum equal 0.9. We train the model for 40 epochs with learning rate=1e-4, weight decay=1e-4, batch size=64. Deep Clustering is applied over features obtained from Cohortney partition with n = 8.

CAE:
A convolutional autoencoder with 5 convolutions and batch normalizations. The kernel size = 3. Trained with Adam optimizer, learning rate = 0.003, batch size = 50, during 200-300 epochs. CAE Clustering is applied over features obtained from Cohortney partition with n = 8.

OT:
This model was trained with Adam optimizer and adjusted learning rate during 40 epochs.

I computed the mean and the standard deviation of Purity over 5 runs (below). K denotes a number of clusters and C denotes a number of classes. The results are marked bold when they are better than ones of alternative methods.

data	DHMP (paper)	DHMP	CAE	OT	DeepCluster
K=2 C=5	-	**0.9998+-0.0005**	0.9940+-0.0048	0.6094+-0.0731	0.8185+-0.2035
K=3 C=5	-	**0.8703+-0.0251**	0.7977+-0.0520	0.8038+-0.0125	0.7375+-0.0544
K=4 C=5	-	0.7374+-0.0797	**0.7433+-0.0452**	0.5393+-0.0350	0.5799+-0.0388
K=5 C=5	-	**0.6131+-0.0390**	0.6126+-0.0545	0.4803+-0.0197	0.4092+-0.1183
sine-like					
K=2 C=5	0.9898	0.9568+-0.0481	0.9252+-0.0396	**0.9873+-0.0051**	**0.9800+-0.0119**
K=3 C=5	0.9683	**0.9850+-0.0008**	0.8565+-0.0461	0.8075+-0.1103	0.7272+-0.0798
K=4 C=5	0.9360	**0.8381+-0.0325**	0.7015+-0.0184	0.7977+-0.0520	0.5774+-0.0483
K=5 C=5	0.9055	**0.6230+-0.0535**	0.5976+-0.0120	0.4313+-0.0545	0.4123+-0.0196
truncated					
K=2 C=5	0.8085	**1.0000+-0.0000**	0.9965+-0.0049	**1.0+-0.0**	**1.0000+-0.0000**
K=3 C=5	0.7715	**0.9717+-0.0000**	0.7097+-0.1159	0.6883+-0.0475	0.5095+-0.1320
K=4 C=5	0.7056	**0.9912+-0.0006**	0.8048+-0.0738	0.7491+-0.1022	0.6496+-0.0383
K=5 C=5	0.6774	**0.8732+-0.0882**	0.6307+-0.0598	0.6060+-0.0851	0.4985+-0.0478

Training of the Deep Cluster and Optimal Transport required hours for each experiment, while CAE was a lot faster, that's why I worked with it more than with the other models. It was very hard to beat the baseline.

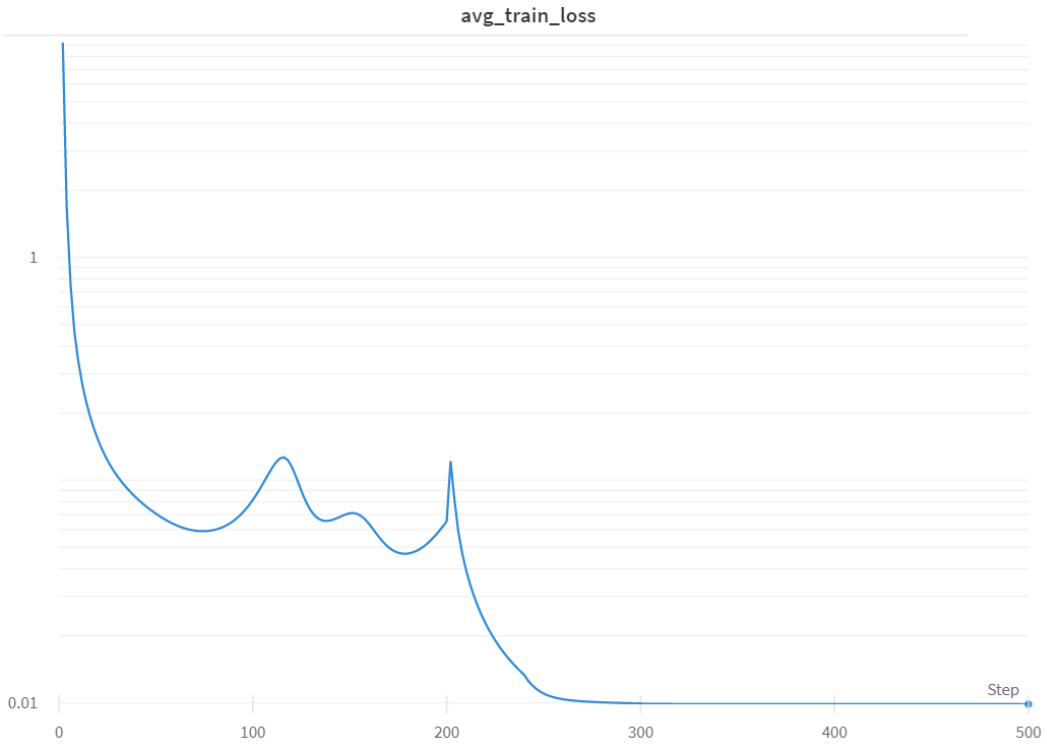
Then, I regenerated the dataset in another way (the same logic, but another random seeds in kernels) (the first version of code is in root/data_generation-v1.ipynb, the second one in root/data_generation-v2.ipynb). The results of DHMP on it became closer to the DHMP's one from the original paper. But CAE still did not beat it.

data	DHMP	CAE
sine-like		
K=2 C=5	**0.9807+-0.0022**	0.9528+-0.0005
K=3 C=5	**0.9925+-0.0008**	0.9907+-0.0003
K=4 C=5	**0.7050+-0.0069**	0.6261+-0.0172
truncated		
K=2 C=5	**0.9345+-0.0010**	0.6210+-0.0576
K=3 C=5	**0.6733+-0.0025**	0.6443+-0.0451
K=4 C=5	**0.7084+-0.0034**	0.5105+-0.0573
K=5 C=5	**0.7572+-0.0008**	0.5890+-0.0688

Initially, the experiments with CAE were made with Adam optimizer and learning rate = 0.003. Then, I experimented with an adaptive learning rate for 3 clusters (below)

and trained the model during 500 epochs, but it did not improve the results of the clustering task - maybe, because the resulting representations are not so good clusterable. So, a better autoencoder does not mean a better clustering model, it seems to be a more complicated task requiring a more detailed analysis.

```
def adjust_lr_K3_C5(epoch):
    if epoch < 100:
        return 0.003
    if epoch >= 100 and epoch < 120:
        return 0.0003
    if epoch >= 120 and epoch < 150:
        return 0.000003
    if epoch >= 150 and epoch < 200:
        return 0.0000003
    else:
        return 0.00000003
```



Literature

1. *A Dirichlet Mixture Model of Hawkes Processes for Event Sequence Clustering*. Section 5.1.
2. *The Neural Hawkes Process: A Neurally Self-Modulating Multivariate Point Process Hongyuan*. Appendix, section b2, algorithm 2
3. *Learning Granger Causality for Hawkes Processes*. Section 4.5, algorithm 2; section 8.5.