Ablation Study of Neural Network Based General Temporal Point Processes Model

Geng Shao, Gloria Wan, Rebecca Wang
Department of Electrical and Computer Engineering
Department of Comupter Science
McGill University
geng.shao@mail.mcgill.ca
yueyang.wan@mail.mcgill.ca
zhaoyue.wang@mail.mcgill.ca

Abstract

In this project, we did the ablation study of the paper "Fully Neural Network based Model for General Temporal Point Processes" by Takahiro Omi from the 2019 NeurIPS accepted papers. This paper discusses Recurrent Neural Network (RNN) based models developed for point processes. RNN based models usually assume a specific functional form for the time course of the intensity function of a point process. This paper proposed a novel model where the time evolution of the conditional intensity function is formulated based on a neural network rather than assuming a specific functional form. We confirmed that the conclusions of the paper are supported. This new model overall achieves superior performances compared to the previous state-of-the-art methods for synthetic datasets. However because of the nature of the neural network where it requires a training process, it has a higher variance and highly depends on the process used to generate the data. Typically it performs better when the process is more complex, specifically stationary renewal, non-stationary-renewal, and Hawkes Processes. We show that tanh is a better activation function for the Neural Network and the 64 units per hidden layer setting in this paper performs worse compared to 16 units and 32 units, reducing the layer does not only increase the accuracy but also decreases the computational power required. Thus, we purposed ways to optimize the model and improve the result.

1 Introduction

1.1 Background and Motivation

Ablation study is a great way to analyze a machine learning model. By removing and modifying some of the features to see how it affects performance, it shows which parts of the model are especially crucial to the results and which parts are not (1). We choose to do an ablation study on the paper "Fully Neural Network based Model for General Temporal Point Processes". This paper is interesting as it purposed a new approach to the current state of the art model for the temporal point process. A temporal point process is a random process whose realizations consist of the time of isolated events. (2) The more widely adapted understanding views temporal point processes as how long events occur where the events themselves are spaced according to a discrete set of time parameters. The behavior of a temporal point process N is typically modeled by specifying its conditional intensity $\lambda = \lambda(t)$. The intensity functions of many point processes involve two components: the background and the effect of historical events. Formerly, models for point processes use parametric forms for the conditional intensity function that are carefully designed by researchers' prior knowledge to capture

Submitted to 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Do not distribute.

the character of the dataset in the study. However, this gives rise to a limitation where parametric forms of point process are specialized and restricted for arbitrarily distributed event data which tends to be oversimplified or even infeasible for capturing the problem complexity in real applications. Therefore, a Recurrent Neural Network (RNN) is used to obtain a compact representation of the event history. The conditional intensity function is then modeled as a function of the hidden state of the RNN. Consequently, the RNN based models outperform the linear and/or parametric models in prediction performance (3)(4). This paper brings it one step further by purposing a new model. It is innovative in a way that employs a neural network to represent the background time series of the intensity function in a general manner. In their experiment, a feed-forward neural network is applied to integrate the intensity function and then the intensity function is obtained through the differential. Through an average negative log-likelihood function, the authors get rid of any numerical approximations in the model of intensity function. (5) As a result, the expressive ability of recurrent Neural Network is improved, and the potential deterioration of the incorrect function-form assumptions can be nearly eliminated. Finally, the authors defend the superiority of their method through comparison with previous state-of-the-art methods. We are interested in this paper as we want to find out whether deep learning techniques can indeed defeat traditional mathematical methods in point processes, a specialized topic that used to only depends on pure mathematical functions, equations and accumulated knowledge of individual researchers. It might show us how machine learning can be revolutionary. The code and raw data of our work can be found in (6).

2 Related Works

2.1 Fully Neural Network based Model for General Temporal Point Processes, Omi et al.

Omi et al., 2019 purposed a novel model for point processes (5) While other RNN based models usually assume a specific functional form for the time course of the intensity function of a point process (e.g., exponentially decreasing or increasing with the time since the most recent event), this model uses a feedforward neural network to obtain the intensity function as its derivative (Figure.1). The backpropagation through time (BPTT) is employed to obtain the gradient of the loglikelihood function. Rather than directly modeling the hazard function, this paper defines cumulative hazard function $\Phi(\tau|h_i)$ as follows:

$$\Phi(\tau|h_i) = \int_0^\tau \phi(s|h_i)ds$$

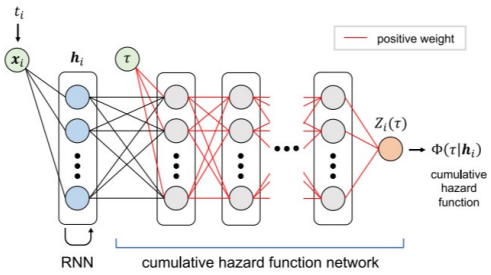

Figure 1: Network architecture to output the cumulative hazard function.(5)

The paper demonstrates that Neural Network Based Model (NN) outperforms parametric models (5) by a notable margin on stationary renewal (S_Renewal), non-stational renewal (N_Renewal), Hawkes Process 1 (Hawkes1) and Hawkes Process 2 (Hawkes2). It is as effective as or less effective compared to other models on stationary Poisson (S_Poisson) and non-stationary Poisson (N_Poisson) and self correcting (SC).

2.2 Modeling The Intensity Function Of Point Process Via Recurrent Neural Networks, Xiao et al.

In this paper (7), RNN is used to model the background of intensity function of point processes. The units of RNN is aligned with time series indexes while the history effect is modeled by another RNN

whose units are aligned with asynchronous events to capture the long-range dynamics. The authors point out the advantage being that the whole model with event type and timestamp prediction output layers can be trained end-to-end. For example, we could pre-train the model to fit real-life data before using it. The main difference between these two papers is that while Shuai et al, 2017 focuses on preserving temporal dynamic behavior of event sequences using RNN (Figure.2), while the paper we are studying uses NN in attempt to construct a even more versatile and flexible intensity function expression. Using NN also reduces the problematic issue of gradient vanishing or explosion due to history dependence for a long sequence.

Overall, in the study of point processes, we see a trend of combining traditionally mathematical methods with machine learning. From both papers, we can see that the goal is to allow a black-box treatment for modeling the intensity which is often a pre-defined parametric form in point processes. This allows the algorithms to be easier to use for people without prior domain knowledge and reduces false assumption errors.

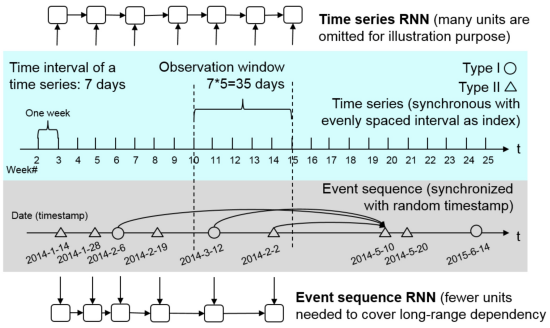

Figure 2: Synergically modelling time series and event sequence. Using RNN in intensity function.(7)

3 Dataset

For synthetic data, the authors generate seven random processes. There are S_Poisson, N_Poisson, S_Renewal, N_Renewal, SC, Hawkes1 and Hawkes2.Codes which generates each random processes are provided so these seven random processes are available for reproducing results and conducting ablation sutdy.

For real data, the authors provide four real datasets (5) without related code so we decided to write the code by ourselves. The first dataset needs purchasing and the link of the fourth dataset is broken. For the second dataset, we have prepossessed the dataset as described in (5). Unfortunately, this dataset was updated on 28th Nov. Due to the difference caused by updating, we fail to reproduce the same results but we still upload the code of prepossessing in code.zip. The size of the third dataset is too big and overwhelms Google Colab. We cannot find sufficient computation resources to preprocess. But the idea of prepossessing is still similar to the second dataset so our code should work on it.

The testing of real datasets is discouraged at the beginning due to uncontrolled reasons so we mainly focused on synthesis data in this project.

4 Proposed Approach

We applied the code provided by the paper in GPU environment in Google Colabto keep the consistency between our experiments and the authors' original work, we will still use the synthetic. We only used the synthetic data as the real data is not accessible. We want to investigate how deleting and changing the factors of the feedback neural network will change the performance of the model. More specifically, we aim at investigating whether the parameters used in the original paper are important. And if it is important to the result, whether it can produce the optimized result. Lastly, if the parameters does not produce the best result, we intend to find the better setting. Before performing the ablation tasks, first we will replicate the result from the paper using the code published by the

author to affirm that the experiment and conclusion are valid. Then, we proceed to examine this project in two aspects: 1. hyper-parameter explorations, 2. different model components' utilities.

4.1 Parameters

We want to explore how the changes in some hyper-parameters in the model affects the result. We keep in mind that we do not want to change parameters that contribute to the framework of the model as we still want to maintain the overall structure and framework of the given model. Therefore we selected the following features we think are appropriate. The parameters and its respective changes are listed as followed: Training size: 20%, 40%, 60%, 80%. Learning rate: 0.1, 0.01, 0.05, 0.001, 0.005. Activation function: tanh, ReLU, sigmoid, softplus. number of hidden layers: 2,3,4,5,6. number of units per hidden layers: power of 2 from 2 to 1024. optimizer: RMSprop, Nadam, Adam. The original setting is training size= 80%, learning rate = 0.1, activation function = tanh, number of hidden layers = 2, number of units per hidden layers = 64, optimizer = Adam.

Our method of experiments is to change one parameter each time, keeping the rest of the parameters controlled. By doing so allows any changes in the result to sole depends on the features we want to examine and minimize confusion. For the activation function we only modified those of the neural network and kept the rest as in the original report. To keep the model robust, we do not modify the parameters for RNN. We use MNLL to evaluate whether the quality of calculated conditional intensity functions. Smaller MNLL is, better the calculation is. MAE is not preferred because regarding evaluation, MNLL surpasses MAE for MNLL has a more obvious tendency.

4.2 Components' Utilities

We are interested in which part of the NN contributes the most into producing a better result. An NN model consists of one input layer, several hidden layers and one output perceptron. Every layer could have its own contribution to the final results. The input, hidden and output weights are limited in positive value because the authors of the paper think that in this way the cumulative hazard function will make sense. But we think the cumulative hazard function will make sense as long as it is monotonically increasing, which does not imply that every weight needs to be positive. We hence modified the original model into three models: one without hidden layer, one without neither input nor hidden layer and one without positive limitation of weight. By comparing whose MNLLs increase the most, we can say which part is the most important one.

5 Results

5.1 parameters

5.1.1 Training Size

To test the robustness of the model, we try to explore the effect of training size on the value of MNLL. We plot the MNLL versus seven random processes curve for each of the four models at different training sizes, the testing training sizes are chosen to be 20%, 40%, 60% and 80%. The following figure shows the results for NN model, the figures for the rest of the models have similar shapes and trends so they are not shown here, but they can be found in (8), which is an appendix for extra graphs.

From these curves (including three more figures in (8)) it is clear to see that the size of training set does not have much effect on the MNLL of these random processes, the four curves are nearly overlapped in each figure, except for the Hawkes Processes. Even for the Hawkes Processes, the differences in MNLL among different training sizes are within 10%, which are not significant. For Hawkes1, it can be observed that, no matter which model it uses, the training size of 40% always gives the lowest MNLL and 80% gives the highest, whereas in Hawkes2, 80% always gives the lowest MNLL and 60% gives the highest. This indicates that the Hawkes1 random process may be a little over trained using the author's original setting, but only 10% higher than its lowest MNLL may be acceptable by the author since training size of 80% works well for all other random processes.

There is no general conclusion can be drawn regarding what should be the best training size to use, it mostly depends on the characteristic of the random process itself. However, due to the fact that the

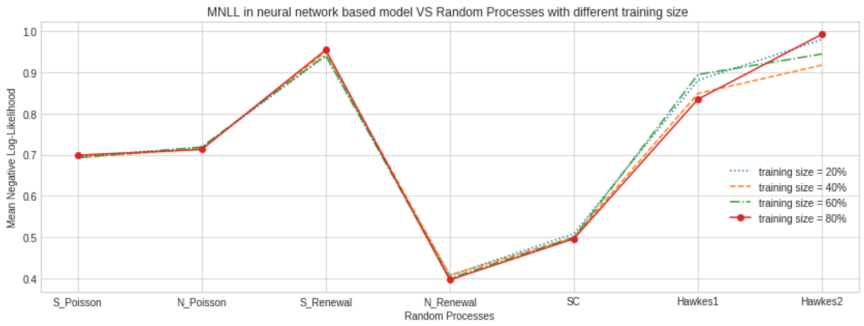

Figure 3: Influence of changing the training size on the MNLL.

overall performance of these four models are not sensitive to the change of training sizes, we can say that the model as a whole is sufficiently robust.

5.1.2 Learning Rate

The author set the learning rate to be 0.001, we test four other different learning rates: 0.1, 0.05, 0.01 and 0.005 while keeping all other hyper-parameters unchanged. The following figures plot MNLL against random processes at different learning rates for NN model and for constant model, the other two can be found in (8).

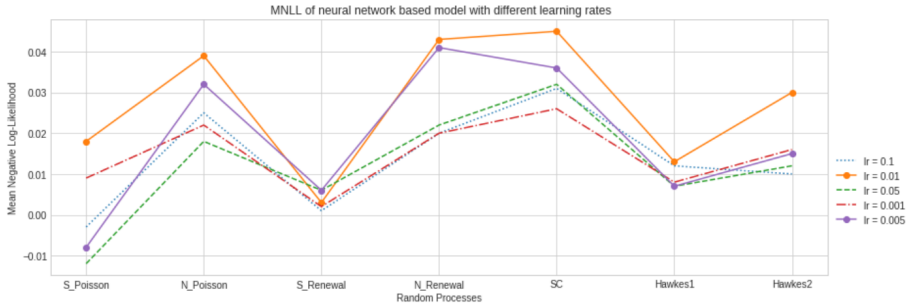

Figure 4: Influence of changing the learning rate to the MNLL for 7 different random processes using NN model

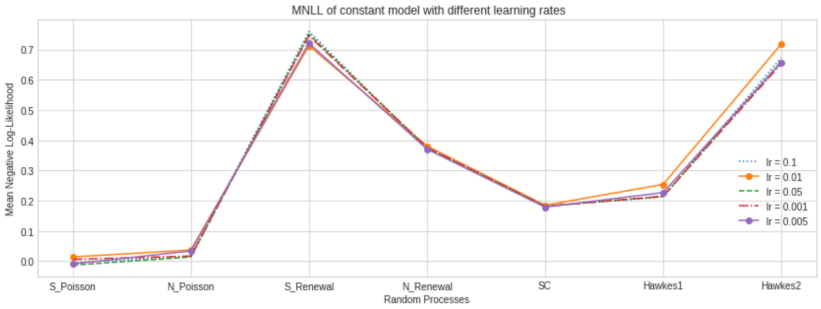

Figure 5: Influence of changing the learning rate to the MNLL for 7 different random processes using constant model

From these figures, it can be seen that the values of MNLL depend more on which model we use rather than the value of the learning rate. For example, the MNLL of S_Renewal process reaches below 0.01 for NN model but goes above 0.7 for constant model at all learning rates. Within a certain model, the effect of learning rate on MNLL is relatively small, the fluctuation is no more than 10%. i Specifically looking at the figure for NN model, one thing worth noticing is that this model has

Table 1: MNLLs for Three Different Optimizers (Blue Stands for minimum and Red Stands for Maximums in each Column)

	Name of Random Processes						
	S-Poisson	N-Poisson	S-Renewal	N-Renewal	SC	Hawkws1	Hawkws2
Adam(Original Model)	0.012	0.030	0.004	0.021	0.035	0.009	0.015
RMSprop	0.013	0.02	0.004	0.023	0.038	0.01	0.014
Nadam	-0.004	0.019	0.002	0.025	0.035	0.016	0.012

greatly reduced MNLL to 0.045 and below, this is approximately 10 times reduction. This result is confirmed with the author's conclusion, which states that the NN model is the best model among these four. In this figure it is also clearer to see that a learning rate of 0.01 generally produces the worst results (i.e. highest MNLL), and results from 0.1, 0.05 and 0.001 are very similar.

5.1.3 Activation Functions

As the author of the paper states that their NN model performs the best over all four models, we test some parameters that is specifically influential for NN model. We try several common activation functions to replace the originally-used "tanh". The resulting MNLL for each random process is shown in Figure.6.

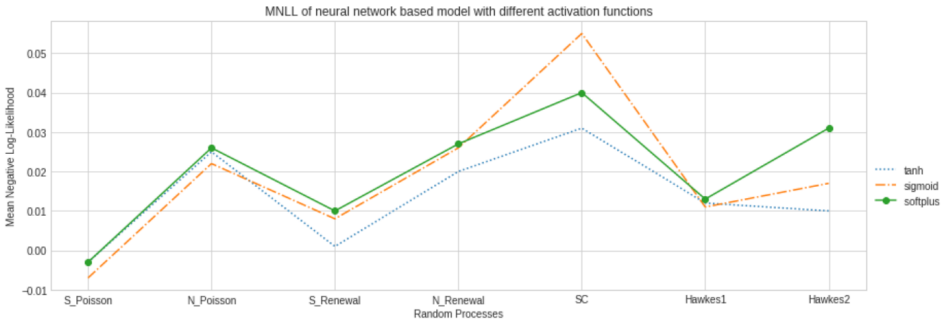

Figure 6: Influence of changing the activation function on the MNLL (without ReLU)

Among the three activation functions we use, ReLU is significantly inferior to the others in NN model. As the resulting MNLL obtained from ReLU is much higher than that obtained from others, plotting these four curves altogether in the same figure shield the trends for sigmoid, softplus and tanh. This is why we only plot the curves without ReLU presented, the seven corresponding values for ReLU are 2.255, 0.59, 0.078, 0.297, 2.94, 1.212 and 0.924. From Figure.6, we can confirm that the author's choice is indeed the best in general, it is the most optimal activation function for most of the random processes. In addition, we also plot the same curves for constant model, exponential model and piece-wise constant model to demonstrate that the tuning in NN model will not affect other models' behavior (figures in (8)).

5.1.4 Optimizer

Optimizer searches the extremums and hence finds the optimization for neural network. The default optimizer is Adam which is RMSprop with momentum. We compare the performance of Adam with RMSprop and Namdam which is RMSprop with Nesterov momentum (9). The results are in Table.1

From Table.1, RMSprop has the worst performance: three of seven MNLLs are the largest and no smallest MNLLs are in RMSprop. Nadam has one more smallest MNLL and one less largest MNLLs then Adam. However, the results of RMSprop, Adam and Nadam differ slightly in general, so we can conclude that NN proposed by authors are robust to optimizer change.

5.1.5 Number of NN Hidden Layers

Firstly, we focus on the influence of the number of hidden layers. The results are in Figure.7. We see that each random process have the respective optimized number of layers. N_renewal and SC prefer the least number of layers. S_Poisson, S_Renewal and Hawkws1 reach the smallest MNLL when number of layers is 3. Hawkws2 needs 5 layers to output its best MNLL. Overall, most smallest MNLLs occur at small number of hidden layers while increment in hidden layers may bring overfitting which degrades the performance.

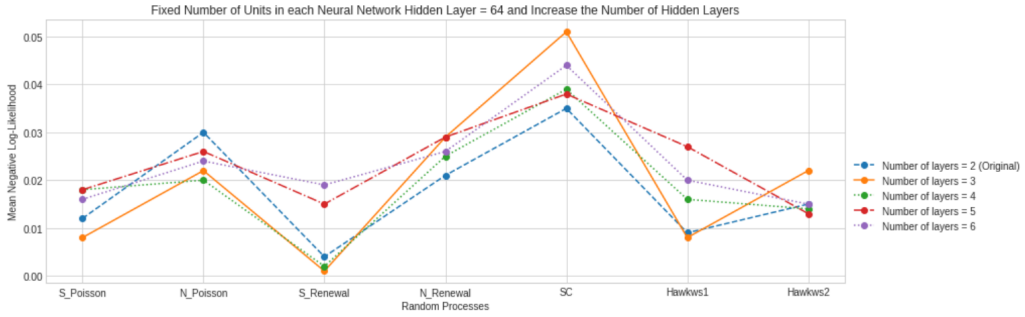

Figure 7: Influence of changing the number of hidden layers on the MNLL.

5.1.6 Numbers of Sizes of NN Hidden layers

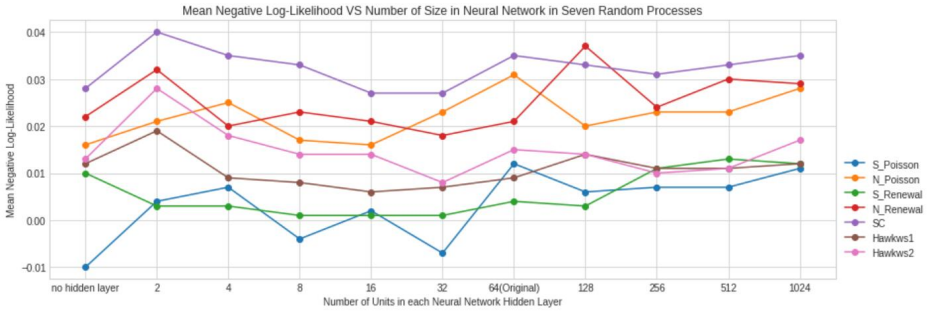

Figure 8: Influence of changing the size of hidden layers, i.e the number of units in each hidden layers on the MNLL.

What surprises us is that the default value used by authors, 64, doesn't give the smallest MNLL in all seven random processes. We find S_Poisson, N_Renewal and Hawkws1 achieve their smallest MNLLs when the number of units is 32. N_Poisson, S_Renewal, SC and Hawkws2 achieve their smallest MNLLs when the number of units is 16. This result also indicates that the authors' model can be simplified without loss of accuracy and even a superior result can be obtained through simplification.

Then we step into the individual process and see the difference between NN and the other three models when the size of hidden layers changes (Corresponding graphs in section 5 in (8)) . We can see that in S_Poisson and N_Poisson, MNLLs of NN are not always the lowest and even can become the largest in some sizes of hidden unit, especially obvious when sizes are 4, 64 and 1024 . When sizes are 2 and 128. the performance of NN falls behind exponential model in SC and the performance of NN falls behind piecewise constant model in Hawkws1. So despite two Poisson processes, exponential and piecewise constant model can be competitive in the certain sizes. In the other three models, N_Renewal, S_Renewal and Hawkws2, NN successfully defends its superiority and generates large margins of 0.2 to 0.3.

Table 2: MNLL when Remove One Critical Part (Blue Stands for minimum and Red Stands for Maximums in each Column)

	Name of Random Processes						
	S-Poisson	N-Poisson	S-Renewal	N-Renewal	SC	Hawkws1	Hawkws2
Original Model	0.012	0.030	0.004	0.021	0.035	0.009	0.015
No Hidden Layer	-0.01	0.016	0.01	0.022	0.028	0.012	0.013
Only One Perceptron	0.088	0.164	0.065	0.211	0.189	0.109	0.13
Remove Positive Limitation	0.004	0.014	0.002	0.021	0.029	0.009	0.01

5.2 Investigation of Components' Utilities

From Table.2, positive limitation of weights seem to be unhelpful for the model because removing it could bring back a better MNLL. Meanwhile, when we remove both input and hidden layers, the results deteriorate but simply removing hidden layers could improve the result slightly. It indicates that the structure of NN, that multiple perceptrons at the same layer, is critical to the advantaged result for even one layer of multiple perceptrons could decrease MNLLs sharply. By the way, the result in no-hidden layer again proves that the authors' model is over-complicated because the disappearance of the hidden layers decreases the MNLLs in four of seven random processes.

6 Discussion and Conclusion

In this study, we confirm the result of the paper. The NN model outperforms the other models by a notable margin on S_Renewal, N_Renewal, hawkes1 and hawkes2. It is as effective as exponential model and piece-wise constant model in self correcting model, which are all notable better than constant model. It is usually as effective as or less effective compared to other models on S_Poisson and N_Poisson. We think this is due to the nature of the neural network. It is a much more complex model and requires a training process. Therefore, it shows greater advantage over the other models on more complex datasets like hawkes1 and hawkes2. For simple datasets, it is sufficient to assume a specific functional form for the time-course of the hazard function. It is also clear that the constant model overall performed worse on all datasets. Therefore, for simpler datasets, exponential model and piece-wise constant model should be used as they can achieve both higher accuracy and requires less computational power. For more complex datasets, NN model presents its advantage. It should therefore be used. In addition, since we have shown that reducing the units in a hidden layer both increases accuracy and reduced computational power, we should modify the NN model.

Combining all the ablation work, We found out that the learning rate and training set percentage have little to no effect on the result. Therefore they are not important features to be considered. We found out that for each process, considering the computation cost, the preferred model(s) differ. For S_Poisson and N_Poisson, constant model is enough to produce a good estimation of conditional intensity function. For S_Renewal, N_Renewal and Hawkes2, NN is necessary to produce satisfied MNLLs. Lastly, in SC and Hawkes1, piece-wise constant model is competitive with NN because it can approximate conditional intensity function with close MNLLs but with fewer computation resources.

In conclusion, although NN model proves to be useful in the certain situations, it highly depends on the nature of the data. Therefore a well-knowledged individual is still needed to determine whether NN model is appropriate to employ. Therefore, in the short term, NN model cannot yet replace a specific model driven by domain knowledge for point processes. Looking forward, we want to extend our scope of investigating to other deep learning frameworks to map the intensity function. Inspired by Xiao et al, 2019, we should compare the effect of feed-forward NN and RNN on real-life data. To tackle the possible issue of gradient vanishing or exploding, other machine learning methods such as LSTM or even simpler methods that require less computational force such as LR. We want to see whether they differ drastically in different situations and if so, which one performs better in a given situation.

References

[1] R. Meyes, M. Lu, C. W. de Puiseau, and T. Meisen, "Ablation studies in artificial neural networks," *arXiv preprint arXiv:1901.08644*, 2019.

[2] D. R. Brillinger. "temporal point process.". [Online]. Available: "http://mathworld.wolfram.com/TemporalPointProcess.html"

[3] H. Jing and A. J. Smola, "Neural survival recommender," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 2017, pp. 515–524.

[4] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song, "Recurrent marked temporal point processes: Embedding event history to vector," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1555–1564.

[5] T. Omi, N. Ueda, and K. Aihara, "Fully neural network based model for general temporal point processes," *arXiv preprint arXiv:1905.09690*, 2019.

[6] G. S. Gloria Wan, Rebecca Wang. "code and raw data of our work.". [Online]. Available: "https://github.com/Schrscat/NeurIPS-2019-Reproducibility-Challenge"

[7] S. M. C. X. Y. H. Z. Shuai Xiao, Junchi Yan, "Modeling the intensity function of point process via recurrent neural networks," *arXiv preprint arXiv:1901.08644*, 2017.

[8] G. S. Gloria Wan, Rebecca Wang. "extra information and graphs.". [Online]. Available: "https://docs.google.com/document/d/1RsSosa1vMYsDXZxH6Gh_7_LvG1TXcOeWEjKbfdbgzAw/edit?usp=sharing"

[9] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, 2013, pp. 1139–1147.

[10] A. Leon-Garcia, "Probability, statistics, and random processes for electrical engineering," 2017.