Predicting In-Game Events from Online Game Event Streams

CS229 Project Report

Eun Jee Sung
ejsung@stanford.edu

Abstract

Massive online battle arena (MOBA) games are becoming increasingly popular in terms of their number of active game players and professional league viewers. This paper proposes a model that can predict the *timing* and *type* of the next upcoming in-game event. By applying the temporal point process to the long short-term memory (LSTM) model, this study proposes a model that can predict the timing of the next immediate event within 1.43 seconds of error. Source code is available at https://github.com/eunjeeSung/cs229-lol-event-prediction.

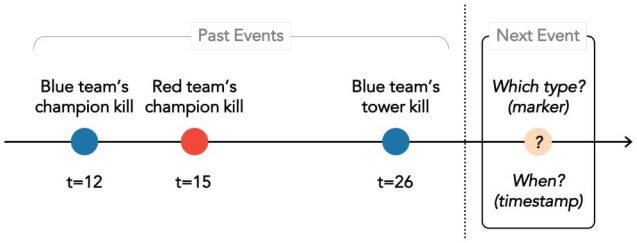

Figure 1: Visualization of problem definition

Introduction

Event data with marker information are being produced from a wide variety of domains, such as online services, financial transactions, and transportation (Du et al. 2016). For example, recommendation systems for commercial services have sought to find when and what type of product a customer would buy given the past sequence of purchases. Across such domains, the common goals of prediction are *when* and *what type* of event will happen in the future.

In the domain of e-sports and online gaming, an increasing body of research is being conducted on in-game event prediction (Yang et al. 2020). Especially with the rise in popularity of massive online battle arena (MOBA) games such as Dota2, League of Legends, and Honor of Kings in terms of their number of active game players and professional league viewers, studies have been conducted on MOBA in-game event predictions. In-game live event predictions can assist, in real time, the audience of professional league games for an enhanced viewing experience and help broadcasters produce better live highlight videos. Currently, professional MOBA league games are recorded by professional observers who follow and capture the major scenes of games and analyze in-game situations in real time. Thus, this study addressed the important task of predicting major online game situations, given a sequence of previous in-game events.

Given the availability of data APIs, this study focused on predicting the events of a MOBA game called League of Legends (LoL), a MOBA game developed by Riot Games. In LoL, 10 players each select champions with different traits and fight in 5v5 mode to destroy the other team's primary structure called Nexus. Its major in-game events include the purchase of core items, the destruction of the opponent team's towers and inhibitors, the death of champions, the death of major monsters on the map, and the increase of levels in players' skills. This study proposes a model that takes a sequence of events as the input and outputs which of these five types would happen next and at which timing.

Related Works

Game Event Prediction

Studies have been conducted on pre-game and in-game predictions of game results and events. Pre-game predictions have focused on predicting game results from pre-game features such as each team's selection of champions and each player's previous performance (Kim, Lee, and Chung 2020). Works focusing on in-game predictions have exploited in-game event streams to predict which of any of the 10 players will die within a five-second window (Katona et al. 2019) and train a game-playing bot using long short-term memory (LSTM) model and reinforcement learning (Lohokare, Shah, and Zyda 2020). Yang et al. (2020) tried to train a model to predict four tasks including the winning team of the whole game and the killer of the next immediate champion death using data from a MOBA game called Honor of Kings, but they did not predict the timing of specific events. Baldominos et al. (2016) developed a system that predicted player behaviors using a Variable-Order Markov model, but focused on building an efficient Hadoop-based system to support real-time prediction.

Temporal Point Process

To better predict the timing of an upcoming event, we adopted the temporal point process. The temporal point pro-

timestamp (m)	type	event_marker	user_id	...	posYPlayer10
...	...	...	...	...	...
7.41	ITEM_PURCHASED	3	9	...	7150
7.472	SKILL_LEVEL_UP	4	7	...	7771
7.513	CHAMPION_KILL	0	5	...	12564
...	...	...	...	...	...

Table 1: Example of the preprocessed data from a single game

cess is a mathematical framework that captures the arrival pattern of sequential events. Specifically, marked temporal point process seeks to predict event timings and their categories ("markers"). They have been successfully applied in various domains, such as seismology (Bray and Schoenberg 2013), online behavior modeling, and recommendation (Cai et al. 2018; Vassøy et al. 2019). Studies more closely related to the domain of online games and e-sports include one by Zhong et al. (2018), in which temporal point process was used to predicting in-game player activities in ice hockey games.

The core concept of temporal point process is the conditional intensity function $\lambda^*(t)$, which specifies the probability density that a new event will occur given a sequence of previous events. Traditional temporal point process models such as the Poisson process and the Hawkes process have been based on certain parametric assumptions about event arrival timings when modeling the conditional intensity of timings given a sequence of historical events. Recently, recurrent marked temporal process (RMTPP) was proposed to use recurrent neural network (RNN) instead to incorporate historical distribution and predict future event pairs (Du et al. 2016).

Problem Definition

This study uses a neural network to predict the timing and type of the next event (Figure 1). The i-th game produces a sequence of pairs (t_j^i, y_j^i) where t_j^i is the timing of the jth event of marker y_j^i in the game ($S_i = \{(t_1^i, y_1^i), (t_2^i, y_2^i), \ldots, (t_L^i, y_L^i)\}$). The goal was to build a model that can predict the next event (t_{j+1}^i, e_{j+1}^i) given a sequence of previous events $\{(t_1^i, y_1^i), (t_2^i, y_2^i), \ldots, (t_j^i, y_j^i)\}$. Here, y_j can be one of the five classes: 'CHAMPION_KILL,' 'ELITE_MONSTER_KILL,' 'BUILDING_KILL,' 'ITEM_PURCHASED', and 'SKILL_LEVEL_UP.'

Dataset

A dataset from 34,499 LoL games was collected using the official APIs from Riot Games. After preprocessing and filtering out games with insufficient log data (<1000 bytes), 22,001 games remained. Each row in our preprocessed game data file contains the following: (1) timestamp data of the event down to the millisecond, (2) event marker, and (3) nine indices of each player's state (9 indices x 10 players = 90 columns), resulting in a total of 92 columns. We divided timestamp by 60,000 so as to express it in the unit of minutes and to prevent overflow during the exponential calculation in our model (Table 1). The dataset was randomly splitted using 80-10-10% ratio to the train, validation, and test dataset.

The most important characteristic of our dataset and task was that the events were sparse. By their nature, in-game events are realized discretely on a continuous time horizon, making it hard to take snapshots of data in regular intervals. The time between two consecutive events in our dataset was irregular. This was because the raw data is comprised of the logs of events (Table 1). In addition, some types of events do not frequently happen in a game. For example, there could be less than 10 "monster kills" in a 35-minute game.

Methods

Feature Selection

Among the nine indices of each player's state, each player's X-axis and Y-axis position were selected as features of each event during the validation process. The ID of the player related with the event was also included. In addition, following the conventional practice of LoL live game broadcasting, the snapshot of the differences between the core indices of two teams at the time of each event (the difference in each team's total amount of gold collected, the number of minions killed, the number of jungle minions killed, the amount of XP and the players' levels) was engineered and added. In sum, together with the basic timing and type information, 28 features were used per event.

Model

Initially, a plain LSTM was used to predict event categories by inputting sequential data of fixed time intervals. The temporal sparsity and irregularity of the data was accounted for by filling in the "missing" timestamps with zero-filling. However, it resulted in a dataset mostly filled with zero indices and was preventing the model from learning useful patterns. Thus, we pivoted to include event timing as one of the explicit prediction goals.

The final model is an LSTM model that adopts RMTPP to predict the timing of the next event (Figure 2). First, the timing, type, and 26 auxiliary features are embedded separately and then concatenated. The concatenated embeddings are then fed into the LSTM layer. Outputs from the LSTM layer are then fed into the attention layer to better capture the

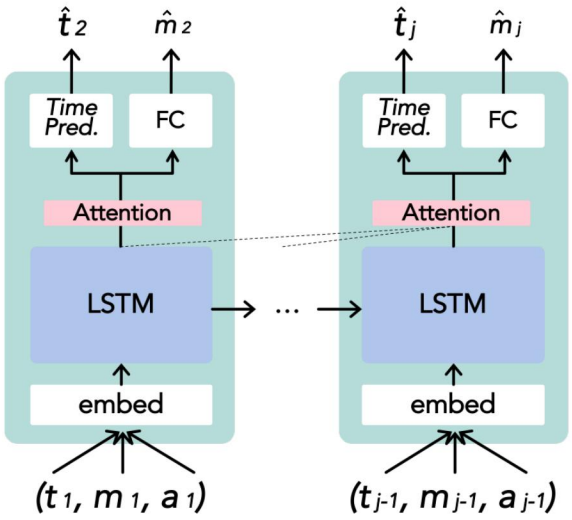

Figure 2: Proposed model (t, m, and a stand for timing, marker, and auxiliary features, respectively)

event sequence. Lastly, two heads follow the attention layer for final prediction: one for timing prediction and another for event type prediction. The parts are described in detail below.

Timing Prediction. Rather than predicting the timing of the next event directly, the best performing model predicts the *intensity* of event timing by adopting the concept from RMTPP (Du et al. 2016). During training, the negative log-likelihood of the target timing is minimized. Given the hidden state of the LSTM layer $\mathbf{h}_j$, the timing prediction layer outputs the likelihood that the next event will happen at time t.

$$\begin{aligned} f^*(t) &= f(t \mid h) \\ &= \exp\Big\{ \mathbf{v}^{t^T} \cdot \mathbf{h}_j + w^t(t - t_j) + b^t + \\ &\frac{1}{w}\exp(\mathbf{v}^{t^T} + b^t) - \frac{1}{w}\exp(\mathbf{v}^{t^T} + w^t(t - t_j) + b^t) \Big\} \end{aligned} \tag{1}$$

During the inference time, the timing of the next event is estimated by using the expectation

$$\hat{t}_{j+1} = \int_{t_j}^{\infty} t \cdot f^*(t) dt. \tag{2}$$

Marker Prediction. The marker of the next event is predicted by estimating the softmax multinomial distribution.

Training

Back Propagation Through Time is used to train the LSTM-based model. The model is unrolled for b steps through time and the shared parameters were updated sequentially in the back propagation step (Figure 3).

The total loss function is a sum of two loss terms. The timing loss L_t is calculated as the negative log-likelihood of the next predicted event timing.

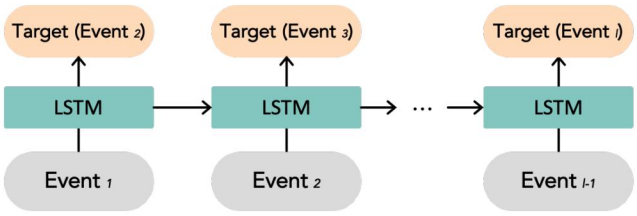

Figure 3: Unrolled version of the model

$$L_t = -\sum_{i=1}^{N}\sum_{j=1}^{L} \log f(d_{j+1}^{(i)} \mid \mathbf{h}_j) \tag{3}$$

Weighted Focal Loss (Lin et al. 2017) was used as the marker loss to overcome the overfitting due to class imbalance in the dataset. A class weight of [8.37, 83.37, 44.08, 1.83, 3.23] was used, which was derived by dividing the total number of events with the number of events in each class in the training set, so as to balance each class' contribution to classification result. The marker loss can be expressed as

$$L_m = \sum_{i=1}^{N}\sum_{j=1}^{L} -(1 - p_t)^{\gamma} \log(p_t) \tag{4}$$

where $p_t = P(y_{j+1} = k \mid \mathbf{h}_j)$ is the softmax probability of the target class. The hyperparameter γ was set to two. The total loss is

$$L_{total} = L_t + L_m. \tag{5}$$

Experiments

Evaluation Metrics

This study's goal was to develop a model that can accurately predict which type of event will happen and when. We report the predictive performance using the following metrics: mean absolute error (MAE), multi-class precision, recall, and F-1. MAE is used to evaluate the performance of predicting event timings. It is defined as the absolute difference of the estimated and actual timing of the next event. Other metrics are used to evaluate event marker prediction performances.

Models

The following models and their variants have been tested in the experiment.

- **Linear and Logistic Regression**: The baseline model consists of a linear regression model to predict the timing and a logistic regression model to predict the type of the next event.
- **LSTM**: The LSTM model takes sequences of events and predicts the timestamp and softmax probability of the next upcoming event. The loss is a sum of the mean squared error for timing prediction and the cross-entropy loss for the event marker prediction.

Model	MAE (s)	Acc.	Precision	Recall	F-1
Lin. Log. Reg.	11.238	0.721	0.328	0.303	0.222
LSTM	44.766	0.7	0.18	0.252	0.152
LSTM + RMTPP	3.99	0.702	0.136	0.254	0.152
Lin. Log. Reg. + FL	24.852	**0.745**	0.361	**0.361**	**0.358**
LSTM + FL	45.954	0.73	0.375	0.323	0.289
LSTM + RMTPP + FL	**1.43**	0.738	**0.415**	0.346	0.318

Table 2: Experiment result. Precision, recall, and F-1 are macro indices. 'FL' stands for Focal Loss.

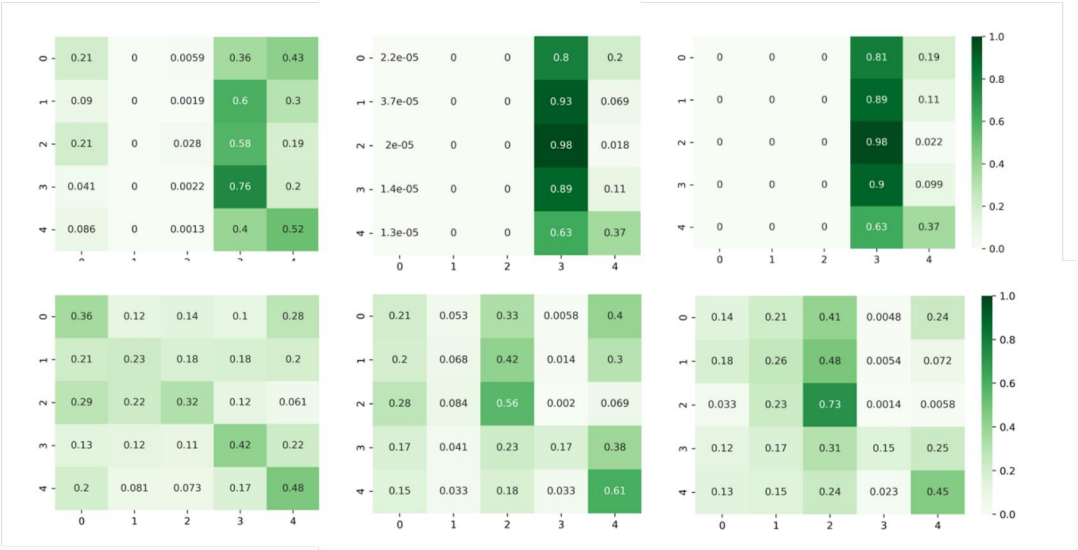

Figure 4: Confusion matrix of the classification results. The top row is from the models trained with cross-entropy loss and the bottom rows are from the models trained with Focal Loss. The left, middle, and right column is from the logistic regression, the LSTM, and the LSTM+RMTPP model, respectively.

- **RMTPP with LSTM**: This model was described above.

The models were all trained for 10 epochs after hyperparameter tuning. Using the validation set, hyperparameters were swept across the learning rates of [0.001, 0.004, 0.0001, 0.00004] and the hidden size of [4, 8, 16, 32] to finally select the learning rate of 0.00004 and the hidden size of 16. The Adam Optimizer was used with zero weight decay for all the models. The training task was to predict the type of the next event among the five possible categories, paired with its timing. All models were trained using a mini-batch size of 128 and a Back Propagation Through Time window size (b) of 32.

Results

The overall performance of the different methods is presented in Table 2.

The proposed model accurately predicted timing. Using the temporal point process, it could predict the next events within 1.43 seconds of error on average. The regression and basic LSTM model, which tried to predict the timing directly rather than the intensity of timing, showed worse performance than the best model. This could be because the linear and logistic regression models could not capture the dependence between events. Also, LSTM models are often applied to datasets with fixed intervals in a way that teaches temporal sequences implicitly rather than explicitly treating them as one of the prediction goals.

Many of the challenges during training had to do with overfitting and class imbalance in the marker prediction. All models suffered from overfitting to one of the classes. Nonetheless, several techniques were explored to counter the overfitting. First, as mentioned previously, Focal Loss helped to balance biased predictions (Figure 4). A dropout layer was also applied to the final marker prediction layer for regularization. Lower learning rates also helped prevent the model from "collapsing" into the most frequent class value in the early stage of training. With higher learning rates, the model quickly learned to return the single most or second frequent class value and could not escape from the local optimum.

Other than the aforementioned successful techniques, other techniques were applied, but empirical performance gains were not observed. Undersampling common classes and oversampling rarer classes were considered, but because the timing model had to be trained simultaneously, this prevented the adjustment of the sampling rates. It would have changed the frequency of events, leading to worse performance of the timing prediction layer. Reducing hidden size of the model was also considered. The final hidden size of 16 was selected as a result of validation process, where the smallest hidden size of 4 was chosen and was gradually increased to reach 16. Using the hidden size of 32 for the LSTM and feedforward layers, or stacking the LSTM layers twice, all led to worse performance. Other regularization techniques were attempted such as weight decay and auxiliary targets (Lipton et al. 2015), but these were unsuccessful as well. Initially, it was expected that a weight decay (or L2 regularization) would allow for an increase in the hidden size, but even with different decay rates, significant gains from the regularization was not observed.

Conclusion

This paper proposes a model that can predict the timing and type of the next upcoming event in an online MOBA game called League of Legends. By exploiting LSTM model and temporal point process, the model can accurately predict the timing of the next event and capture its type.

Despite best efforts, this study involves several limitations. First, the proposed model simplified the dynamic interaction between many entities in the game, including players, monsters, and towers. Often, localized fightings take place in LoL and individual players have different preferences, which can all lead to different patterns in the event sequence. Second, due to the limitation in temporal point process model, the model in this study can only predict the timing of the immediate upcoming event. This is helpful for live broadcasting and player-assisting softwares, but the model can be more valuable if it could predict sequences of events over the longer term.

Therefore, there are several areas where future studies can be conducted. First, the model can be improved in its ability to predict performance on event types, especially the "CHAMPION_KILL" type, as champion kills are of most interest to audience and game players. Second, as mentioned above, the model can be improved to predict events over a longer term. Finally, events could perhaps be predicted by player, event type, or lane.

Contribution

This study is a solo project of Eun Jee Sung (@ejsung). The dataset was collected using the official APIs provided by Riot Games and the baseline code adapted from https://shinminyong.tistory.com/30. Other codes including the preprocessing pipeline, baseline and proposed models, and inference pipeline were developed with PyTorch myself.

References

Bray, A.; and Schoenberg, F. P. 2013. Assessment of point process models for earthquake forecasting. *Statistical science* 510–520.

Cai, R.; Bai, X.; Wang, Z.; Shi, Y.; Sondhi, P.; and Wang, H. 2018. Modeling sequential online interactive behaviors with temporal point process. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 873–882.

Du, N.; Dai, H.; Trivedi, R.; Upadhyay, U.; Gomez-Rodriguez, M.; and Song, L. 2016. 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*, 1555–1564.

Katona, A.; Spick, R.; Hodge, V. J.; Demediuk, S.; Block, F.; Drachen, A.; and Walker, J. A. 2019. Time to die: Death prediction in dota 2 using deep learning. In *2019 IEEE Conference on Games (CoG)*, 1–8. IEEE.

Kim, D.-H.; Lee, C.; and Chung, K.-S. 2020. A confidence-calibrated moba game winner predictor. In *2020 IEEE Conference on Games (CoG)*, 622–625. IEEE.

Lin, T.; Goyal, P.; Girshick, R. B.; He, K.; and Dollár, P. 2017. Focal Loss for Dense Object Detection. *CoRR* abs/1708.02002. URL http://arxiv.org/abs/1708.02002.

Lipton, Z. C.; Kale, D. C.; Elkan, C.; and Wetzel, R. 2015. Learning to diagnose with LSTM recurrent neural networks. *arXiv preprint arXiv:1511.03677* .

Lohokare, A.; Shah, A.; and Zyda, M. 2020. Deep Learning Bot for League of Legends. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, 322–324.

Vassøy, B.; Ruocco, M.; de Souza da Silva, E.; and Aune, E. 2019. Time is of the essence: a joint hierarchical rnn and point process model for time and item predictions. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 591–599.

Yang, Z.; Wang, Y.; Li, P.; Lin, S.; Shi, S.; and Huang, S. 2020. Predicting Events in MOBA Games: Dataset, Attribution, and Evaluation. *CoRR* abs/2012.09424. URL https://arxiv.org/abs/2012.09424.