Predicting Player Churn with Echo State Networks

Rafet Sifa
Fraunhofer IAIS
St. Augustin, Germany
Rafet.Sifa@iais.fraunhofer.de

***Abstract*—We introduce the idea of utilizing a recurrent neural network based representation learning approach to extract and model the complex and sequentially dependent player behavior in games. Our approach is based on the dynamical systems of Echo State Networks, which are very simple to evaluate yet powerful temporal representation learners. We empirically evaluate our approach by illustrating a case study for predicting player churn.**

I. Introduction

Previous game analytics research has shown that the behavior of digital game players from macro and micro perspectives as well as the resulting decisions they make change as a function of time [1]–[5]. When predicting future activities of players, analysts and marketing experts nowadays are in need of scalable and reliable models that can extract and model the temporal player behavior for building informed decision-making frameworks. Two commonly observed challenges of the existing solutions relate to the scope with respect to the temporal dependency or the complexity of the modeling approaches. Namely, they either do not provide full-fledged modeling of the temporal aspects (for instance relying on the markovian assumption [6], [7] or censored temporal data representations [8] that do not consider the variable length of sequential behavioral data) or are very complex for the existing datasets (e.g. in terms of the necessary number of parameters to be trained to avoid overfitting) or require expensive computation infrastructures (such as a dedicated computer cluster).

Taking a look at predicting player departure, i.e. churn, in games, we note that it has been gaining popularity in the last years mostly due to the steadily increasing number of newly released games as well as the financial challenges related to acquiring new players [9]. This has become more popular for non-contractual settings (such as freemium games) as the optional micro-transactional payments usually only occur for advancing the gameplay quickly or unlocking special or cosmetic items. Similarly, for advertisement-driven games keeping the players in the game increases the chances of clicks as well. Being able to accurately predict players that might depart beforehand gives the developers and market managers a competitive advantage, as necessary actions (such as giveaways in terms of in-game items or currencies) can be taken to keep such players from quitting interacting with the system. Given that, introduced in [10], the two popular established definitions of churn (i.e. hard vs soft churn) were based on formulating the prediction setting as a binary classification problem that was based on an observation period with fixed length and predicted if the user would be likely to churn or retain in a subsequent period of time. Over the last years, a variety of different supervised methods has been proposed to predict churn in the context of freemium games. Examples of such methods include simple linear classifiers such as Logistic Regression [10]–[12], tree based classifiers such as Decision Trees and Random Forests [10]–[14], markovian models such as Hidden Markov Models [6] as well as feed-forward and recurrent neural network based classifiers [11]–[13]. Additionally, predicting different customer loyalty indicators (such as retention, purchase decisions, lifetime value and disengagement) in freemium as well as retail games, has also been tackled following similar settings [5], [7], [15]–[18]. We note that the already utilized methods rely heavily on extensive feature extraction steps, requiring the analysts to engineer a set of informative and predictive features as well as to evaluate their prediction performance before rolling them out in a productive system [10]–[12].

Although not as widespread as directly building behavioral predictors with engineered features as explained above, there has been research work about building automated feature extractors (a.k.a methods for learning representations) to find important features for the tackled problem [8], [19]–[21]. For instance [19] proposed to learn spatio-temporal features by factorizing a tensor containing the collection of waypoint graphs encoding their movements in an open-world game. The features were defined as low dimensional representations containing affinities among global factors. Similarly [8] predicted retention by learning temporal behavioral features by factorizing a tensor containing temporal features players. [21] consider a combined approach for learning features from aggregated historic data and sequential data, where features from the latter are extracted using Long Short-Term Memory (LSTM) networks, to predict churn in a freemium game.

The main contribution of this work is based on introducing the idea of utilizing Echo State Networks (ESNs), which are straightforward-to-implement recurrent neural networks with random connectivity, to extract sequential representations from behavioral datasets with minimal effort for a variety of downstream Game Analytics tasks. We note three main advantages of utilizing ESNs for tasks involving sequential data. First of all, unlike matrix and tensor factorization models, which treat temporal aspects as another data dimension [8], through their capability of having a dynamic memory, ESNs provide a more concise representation to model the sequential dependencies in the dataset and can work with sequential datasets of varied sizes. The latter is very crucial when, for instance, extracting temporal player representations. Secondly,

978-1-6654-3886-5/21/$31.00

unlike Hidden Markov Models or temporal difference based supervised models, they do not necessarily rely on the markovian assumption and can learn longer dependencies [22]. Finally, ESNs are in general exceedingly easy-to-implement and, unlike their counterparts, usually contain a reduced number of (hyper-) parameters to be adjusted according to the solved task. These two points make them good candidates for fast-prototyping (as baseline/main temporal feature extractors) as well as in thin-data scenarios, where the available data to train models is scarce (for instance in early development phases).

With this early research work, we intend to propose a new area of investigation and show a use-case of how ESNs can be utilized to predict player churn in the context of freemium games. We will continue this work by shortly introducing the main idea behind ESNs in Sec. II. Following that, in Sec. III we will show how unadjusted ESNs can be used to extract fixed dimensional data representations, which will be later used in Sec. IV to extract temporal player representations for predicting churn using a dataset from a casual freemium mobile game called *Dodge the Mud* [11]. Finally, we will summarize this work in Sec. V and present our future research directions.

II. Introduction to Echo State Networks (ESNs)

Belonging to the family of Reservoir Computing, ESNs are special types of recurrent neural networks that rely on preserving the patterns in the analyzed dataset by considering a particular connection and training mechanism, that can be based on multiple regression. ESNs and their variants have been successfully applied as standalone prediction models and feature extractors to numerous challenging problems e.g. from natural language processing [23]–[25], financial data analysis [26], healthcare analytics [27], telecommunication [28] and optimization [29]. Unlike the conventional and nowadays popular backpropagation methods, ESNs keep randomly initialized input and recurrent (a.k.a reservoir) weights, while training the output (also called readout) weights is done by mapping the network states to the target variables [22] (see Figure 1 for a pictorial illustration). Assuming we are given a sequential dataset containing T number of observations defined as input/output tuples as $\mathcal{d} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_T, y_T)\}$, where $\boldsymbol{x}_i \in \mathbb{R}^n$ is an input vector and $y_i \in \mathbb{R}$ is an output value. Given that $\boldsymbol{x}_t$ is the input datapoint at time t and g is the number of recurrent neurons in the reservoir, the activations of the network's hidden state at time t are grouped in $\boldsymbol{h}_t \in \mathbb{R}^g$ and defined as

$$\boldsymbol{h}_t = f^r(\boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{h}_{t-1}), \tag{1}$$

where f^r is the reservoir activation function (typically defined as the sigmoid or the $tanh$ function), $\boldsymbol{A} \in \mathbb{R}^{g \times n}$ is the input weight matrix and $\boldsymbol{B} \in \mathbb{R}^{g \times g}$ is the reservoir weight matrix. Following that the output of the network at time t is defined as $\hat{y}_t = f^o(\boldsymbol{c}^T \boldsymbol{h}_t)$, where f^o is the activation function for the output unit (usually defined as the unity function $f^o(q) = q$) and $\boldsymbol{c} \in \mathbb{R}^g$ contains the output weights.

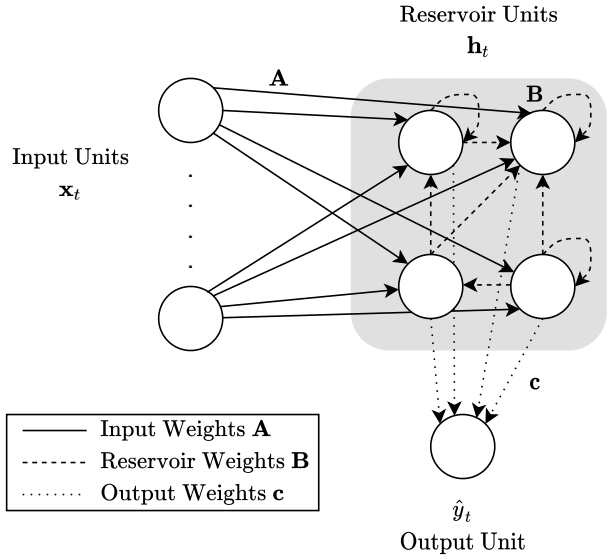

Fig. 1: A pictorial illustration of the architecture of an Echo State Network (ESN) for datasets constructed as $\mathcal{d} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_T, y_T)\}$, where $\boldsymbol{x}_i \in \mathbb{R}^n$ is an input vector and $y_i \in \mathbb{R}$ is an output value. For a given nonlinear reservoir activation function f^r as well as the randomly initialized input- and reservoir weight matrices resp. $\boldsymbol{A} \in \mathbb{R}^{g \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{g \times g}$, the hidden state of the network at time t is calculated as $\boldsymbol{h}_t = f^r(\boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{h}_{t-1})$. Following that, only the output (sometimes called readout) weights (in this example the weight vector $\boldsymbol{c}$) are learned to facilitate mapping the hidden states to the output values so that (for a given activation function of the output units f^o) the output at time t is evaluated as $\hat{y}_t = f^o(\boldsymbol{c}^T \boldsymbol{h}_t)$. In this work we focus on obtaining temporal representations of each data entry by considering their corresponding final hidden states.

The training procedure starts by 1-initializing $\boldsymbol{A}$ and $\boldsymbol{B}$ randomly (the spectral radius of $\boldsymbol{B}$ is usually selected to be less than unity in the *hopes of* reaching a certainty condition for the so-called *Echo State Property* [22]), 2- creating a row-wise matrix of the hidden states $\boldsymbol{G} \in \mathbb{R}^{T \times g}$ s.t. each ith row is defined as $\boldsymbol{g}_{i:} = \boldsymbol{h}_i$ (computed as in (1)), 3- defining a vector $\boldsymbol{y} \in \mathbb{R}^T$ containing all the output values in $\mathcal{d}$. For $f^o(q) = q$ the training is usually done by minimizing the sum of the squared differences of the produced output values and the actual ones observed from the dataset, which can be obtained analytically considering the linear regression solution $\boldsymbol{c} = \left(\boldsymbol{G}^T\boldsymbol{G}\right)^{-1}\boldsymbol{G}^T\boldsymbol{y}$. We note that for datasets with vector-valued outputs, the network output is defined as a vector and this requires solving a multiple regression problem instead.

III. Extracting Representations with ESNs

Having introduced the idea of ESNs, we will now turn our attention to a methodology to extract fixed dimensional representations from datasets with sequential dependencies considering the first two steps of the training process described above. It is worth noting that, numerous downstream tasks in

game analytics such as time series clustering for profiling and predicting player churn based on temporal data operate on a set of sequential datasets to encode dependencies among them and can majorly benefit from having vector representation of each dataset encoding its characteristics as each entry might have a different cardinality value.

Formally, given $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$, where each entry is defined as $d_i = \{\boldsymbol{x}_{i1}, \boldsymbol{x}_{i2}, \ldots, \boldsymbol{x}_{iT_i}\}$ with length T_i and $\boldsymbol{x}_{ij} \in \mathbb{R}^n \; \forall \; i, j$, our goal is for each $d_i \in \mathcal{D}$ to find a vector representation $\boldsymbol{z}_i \in \mathbb{R}^g$. ESNs have been successfully used to encode sequential datasets of different lengths into a fixed dimensional space (see examples from [23], [25]). Instead of training different networks for different datasets, the main idea here is to consider a *single* network with fixed parameters that can be used as a feature extractor for each set of sequential observations. To that end, we represent each unit in the dataset by considering the last hidden state of the network. That is, given that $\boldsymbol{h}_{i0}$ and $\boldsymbol{h}_{iT_i}$ stands resp. for the initial and the last hidden state of the ith entry $d_i \in \mathcal{D}$, our approach represents d_i using $\boldsymbol{z}_i = \boldsymbol{h}_{iT_i}$ by starting with $\boldsymbol{h}_{i0} = \mathbf{0}$ and iteratively evaluating (1) until hitting the final element of d_i. The whole approach can be summarized as follows:

- Let $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$, where each entry is defined as $d_i = \{\boldsymbol{x}_{i1}, \boldsymbol{x}_{i2}, \ldots, \boldsymbol{x}_{iT_i}\}$
- Let $\mathcal{Z}$ contain the *fixed* dimensional representations of each element of $\mathcal{D}$ s.t. $z_i \in \mathcal{Z}$ corresponds to d_i
- Initialize the weight matrices $\boldsymbol{A}$ and $\boldsymbol{B}$
- For each $d_i \in \mathcal{D}$:
  - Set $\boldsymbol{h}_0 = \mathbf{0}$
  - For $t \in [1, 2, \ldots, T_i]$:
    $$\boldsymbol{h}_t = f^r(\boldsymbol{A}\boldsymbol{x}_{it} + \boldsymbol{B}\boldsymbol{h}_{t-1})$$
  - Set $\boldsymbol{z}_i = \boldsymbol{h}_{T_i}$ and add $\boldsymbol{z}_i$ in $\mathcal{Z}$
- Use the representations in $\mathcal{Z}$ in a downstream analytics task such as clustering or classification

IV. Predicting Churn in a Freemium Game

As a case study to show the effectiveness of our approach for predicting churn, we considered an offline evaluation [10] that is based on the publicly available behavioral dataset from [11] concerning the mobile game called *Dodge the Mod*. The dataset contains more than 150k sessions of $25,954$ players for an observation period spanning more than a year. To have comparable results to the ones from [11], we considered an observation period of 5 days and a prediction (churning) period of 10 days.

A. Data Preprocessing and Methods

To learn temporal representations and/or predict churn we benchmarked different approaches including k-Nearest Neighbors (kNN), Logistic Regression (LR), Decision Trees (DTs), Random Forests (RFs), LSTMs (with fully connected final layer) as well as Tensor Factorization (TF) and ESN based models coupled with LR (similar to [8], [23]). We also considered an $80\%/20\%$ training/test split of the data, where the former was used to train the models and the latter to evaluate their generalization. We used 5-fold cross-validation on the training split for hyperparameter tuning by maximizing the F1-Score (shortly F1) and also reported the area under the Receiver Operating Characteristic (ROC) curve (shortly AUC).

TABLE I: A summary of the utilized data representations extracted from the observation period for predicting churn.

Type	Content
Hand Engineered Features	session count, consecutive session ratio, active duration, normalized best score index, best score value, mean score value, standard deviation of the score value, worst score value, ratio of the difference between the best score and the mean score divided by the mean score, normalized differences between the best score and the mean score divided by the session count
Binned Features	individually summed scores values for 10-minute intervals
Pure Temporal Observations	normalized cumulative time from the first session, normalized numerical score and the combination of both values

To show the fact that ESNs can indeed be used by analysts to learn temporal features with minimum overhead, we benchmarked our approach against models trained on the 10 hand-engineered features as well as the binned score representations from [11] (see Table I for an overview of the utilized data representations). For the former, to encode playtime related information we considered the number of sessions (i.e. *play count*), the total time spent in the game (i.e. *active duration*) as well as the ratio of consecutive plays (by considering the ratio of sessions with a difference less than 1500 seconds). Following that we considered the average and the standard deviation of the score values, the index of the best score (normalized by the session count), the values of the worst and the best scores, the (Laplace-normalized) ratio of the difference between the best score and the mean score divided by the mean score (a.k.a. *bestSubMeanRatio* [11]) as well as the normalized differences between the best score and the mean score divided by the session count (a.k.a. *bestSubMeanCount* [11]). Note that, for reproducibility of the results, we will list the used hyper-parameters of the models. Similar to [10], [11] we trained our DTs, with *gini* index as the heterogeneity measure and tuned the maximum depth of the trees for the range $[3, 4, \ldots, 7]$. Considering the same setting for each of their decision trees, for the RF we also tuned the number of trees within $\{41, 51, 61, 71\}$. For the kNN classifier, we considered the Euclidean distance with the parameter for the number of similar points for inference (i.e. k) being in $\{1, 11, 21, 31, 41\}$. For all the runs of LR (including the ones for ESNs and TF) we considered $l2$ regularization with unity weights and using the BFGS-Algorithm for the optimization.

Similarly, as utilized in [11], we considered binning the scores in 10 minutes intervals to consider them as inputs to

the LSTMs to have a $\boldsymbol{x}_i \in \mathbb{R}^{720}$ for each ith player. We used the Adam optimizer with a batch size of 64, while fixing the activation functions of the fully connected layer and the output layer resp. to be the rectifying linear units and the softmax function. As reported in [11] to have comparable results to ESNs, we evaluated the sizes of the states to be in $\{100, 150, 200\}$. In order to evaluate whether TF can extract useful patterns in this context as well, we matricized each binned score vector such that $\boldsymbol{X}_i \in \mathbb{R}^{120\times 6}$, where each row corresponds to the hours and columns to correspond to the 10 minutes chunks. For TF we considered the Tucker 2 decomposition described in [8] using the algorithm from [30], that constrains the basis matrices to be column orthogonal. The TF algorithm factorizes a bipartite tensor (a collection of bipartite matrices of the same size) such that each matrix in our case is described as $\boldsymbol{X}_i \approx \boldsymbol{L}\boldsymbol{W}_i\boldsymbol{J}^T$, where $\boldsymbol{L} \in \mathbb{R}^{120\times r}$ and $\boldsymbol{J} \in \mathbb{R}^{6\times v}$ are the basis matrices describing the global patterns in the dataset (for the hours and the 10 minutes chunks resp.) and $\boldsymbol{W}_i \in \mathbb{R}^{r\times v}$ are the player individual coefficients. As proposed in [8], we will train our TF model on the training set and train a classifier (here we used LR) that takes the vectorized coefficient matrices as input (i.e. for each ith player the input vector is $vec(\boldsymbol{W}_i)$) and whether the player churned or not is defined as the label to be learned. For unseen datapoints, the coefficients are generated from $\boldsymbol{L}$ and $\boldsymbol{J}$ (see [8]). For the parameter search, we considered $r \in \{10, 20, 30\}$ and $v \in \{2, 3, 4\}$ (note the dimensionality of each player matrix for the latter).

Finally, following a similar approach to [8], we consider single ESNs with fixed input and reservoir weight vectors $\boldsymbol{A}$ and $\boldsymbol{B}$ that are randomly generated using the uniform distribution between -0.5 and 0.5 (i.e. $a_{ij}, b_{lm} \sim \mathcal{U}(-0.5, 0.5)\ \forall\, i, j, l, m$), where each d_i in this case corresponds to the temporal observations of the ith player. The activation function for the reservoir neurons is selected to be the $tanh$ function. We only tuned the size of the hidden states to be in $\{100, 150, 200\}$ (as in the case for LSTMs) and the spectral radius to be in $\{.85, .9, .95, 1, 1.05\}$. To assure the latter, we multiplied $\boldsymbol{B}$ by the desired radius divided by the maximum absolute eigenvector of $\boldsymbol{B}$. To make use of the recurrent connections in (1) for the players with single sessions, we duplicated their single session information and added it to the dataset s.t. the cardinality is equal to two. Following that to keep the data processing to the minimum we created three types of datasets for the ESNs, where for each player we had:

- **ESN-T**: Containing the cumulative time from the first session (normalized by the number of seconds in 5 days);
- **ESN-S**: Containing the observed numerical score (normalized by the value 1000);
- **ESN-TS**: Containing the combination of both the normalized time and score values to evaluate whether both observation types have an impact on the churn behavior.

We emphasize the fact that, unlike the previous two data representations, in this case, each observation has a variable length and containing the normalized pure observations.

TABLE II: Predicting churn in a freemium casual mobile game using different representation and prediction models presented in terms of the F1-Score (shortly F1) and the area under the ROC curve (shortly AUC). (a) shows prediction results on an unseen test set followed by a cross validation based model selection using three different representations: hand-engineered features (for DT, RF, LR, kNN), binned score representations (for LSTM and TF) and the pure observations (for ESNs) that are based on time (*-T), score (*-S) and both (*-TS) and only normalized the data by static values. (b) shows the how different DT-, RF- and kNN-based classifiers predict churn based on representations extracted by ESN-TS. See the text for the abbreviations and a more in-depth explanation.

(a) cross validation results

Method	F1	AUC
DT	96.35	79.88
RF	**96.37**	**80.45**
LR	**96.37**	79.17
k**NN**	96.21	77.99
LSTM	96.25	78.25
TF+LR	96.19	74.00
ESN-T+LR	96.07	78.85
ESN-S+LR	96.34	**79.99**
ESN-TS+LR	**96.41**	**80.86**

(b) further ESN evaluations

Method	F1	AUC
DT-1	**96.35**	**78.23**
DT-2	96.32	77.54
RF-1	**96.38**	**80.85**
RF-2	**96.37**	**80.75**
k**NN (1)**	93.78	58.32
k**NN (11)**	96.25	73.09
k**NN (21)**	96.29	76.21
k**NN (31)**	96.27	77.80
k**NN (41)**	96.27	78.00

B. Prediction Results

Having followed the above steps for building the prediction models, we show the results in Table IIa. Taking a look at the results of the predictors that are trained and evaluated with the hand-engineered features, we note that while kNN performed the poorest, we obtained similar AUC values with minor discrepancies for DT, RF and LR as in [11] with different order probably due to having different hyperparameter values. Considering the results with binned representation, we note that LSTMs outperformed TF based models both for F1 and AUC, yet the previous results were overall better. Considering the prediction results with ESNs we note that the results where ESN representations only learned on time were the lowest overall, yet the solely score based results (i.e. ESN-S) were better among the best in terms of AUC and comparable to the results from the models trained with hand-engineered features. Having the temporal as well as score information as input gave the best results in terms of F1 and AUC. We note that the results for LSTMs might increase with a larger hyperparameter search especially on dedicated training hardware, yet using optimized libraries for their training in a CPU machine, the run time for training and inference of an LSTM with the hidden space of size 150 was 563.68 times higher compared to extracting features from the corresponding ESN (with $g = 150$) with time and score information and training a LR model for classification (6.98 vs 3934.45 seconds for a single run).

We also performed a further evaluation to test the predictive power of ESN-TS for different classifiers (see the results from Table IIb). Namely, we input the hidden states extracted by

the ESN-TS (whose hyperparameters were tuned based on LR) and evaluated the performance of predicting churn for kNN classifiers with $k \in [1, 11, 21, 31, 41]$ (results referred to as kNN(k)), two DTs with maximum depths of 3 and 4 (resp. referred to as DT-1 and DT-2) and two RF models with maximum depths of 4 with 51 and 71 random trees (resp. RF-1 and RF-2). In terms of F1 score, the prediction results for kNN improved and the AUC scores seem to be comparable with the previous results. Compared to the prediction performance of ESN-TS+LR, DT and RF based results are similar in terms of the F1 score and the RF results are comparable to the AUC score showing that ESN based features can indeed be combined with decision tree based classifiers to predict churn.

V. Conclusion and Future Work

In this work took a look at an alternative approach towards alleviating some of the commonly faced challenges of modeling temporal behavior. Our approach was based on the idea of utilizing unadjusted Echo State Networks, that do not require an expensive training process, for extracting temporal features, but rather requires the evaluation of a dynamical systems equation that is capable of capturing the dynamical aspects of players' behavior and can be easily implemented without necessarily requiring dedicated hardware infrastructures. We evaluated the proposed approach by presenting a brief case study on predicting player churn. Our results indicated that ESN-based features can better predict player churn compared to hand-engineered features and learned features that are based on binned data.

Our future work involves evaluating the above framework for predicting future behavior in different games. We note that finding the right architecture for the weights of ESNs is an open question in the Reservoir Computing research. Given that it would also be interesting to evaluate the performance of different ESN architectures (e.g. Deep ESNs [27]) for behavior prediction tasks. Additionally, it will be interesting to evaluate the performance of ESN based representations for different applications such as behavioral profiling, outlier detection and temporal recommender systems.

Acknowledgment

Part of this research is supported by the Competence Center for Machine Learning Rhine Ruhr (ML2R) which is funded by the Federal Ministry of Education and Research of Germany (grant no. 01—S18038B). We would like to thank the anonymous reviewers for their insightful comments.

References

[1] D. Vihanga, M. Barlow, E. Lakshika, and K. Kasmarik, "Weekly seasonal player population patterns in online games: A time series clustering approach," in *Proc. IEEE CoG*, 2019.

[2] M. Viljanen, A. Airola, T. Pahikkala, and J. Heikkonen, "User activity decay in mobile games determined by simple differential equations?" in *Proc. IEEE CIG*, 2016.

[3] A. Saas, A. Guitart, and A. Perianez, "Discovering playing patterns: Time series clustering of free-to-play game data," in *Proc. IEEE CIG*, 2016.

[4] R. Sifa, C. Bauckhage, and A. Drachen, "The playtime principle: Large-scale cross-games interest modeling," in *Proc. IEEE CIG*, 2014.

[5] R. Sifa, F. Hadiji, J. Runge, A. Drachen, K. Kersting, and C. Bauckhage, "Predicting purchase decisions in mobile free-to-play games," in *Proc. AAAI AIIDE*, 2015.

[6] P. Rothenbuehler, J. Runge, F. Garcin, and B. Faltings, "Hidden markov models for churn prediction," in *Proc. SAI IntelliSys*, 2015.

[7] M. Tamassia, W. Raffe, R. Sifa, A. Drachen, F. Zambetta, and M. Hitchens, "Predicting player churn in destiny: A hidden markov models approach to predicting player departure in a major online game," in *Proc. IEEE CIG*, 2016.

[8] R. Sifa, M. Fedell, N. Franklin, D. Klabjan, S. Ram, A. Venugopal, S. Demediuk, and A. Drachen, "Retention prediction in sandbox games with bipartite tensor factorization," in *SAI CC*. Springer, 2020.

[9] Newzoo, "The mobile games market: Casual games sector report," Tech. Rep., 2016.

[10] F. Hadiji, R. Sifa, A. Drachen, C. Thurau, K. Kersting, and C. Bauckhage, "Predicting player churn in the wild," in *Proc. IEEE CIG*, 2014.

[11] S. Kim, D. Choi, E. Lee, and W. Rhee, "Churn prediction of mobile and online casual games using play log data," *PloS one*, vol. 12, 2017.

[12] J. Runge, P. Gao, F. Garcin, and B. Faltings, "Churn prediction for high-value players in casual social games," in *2014 IEEE conference on Computational Intelligence and Games*. IEEE, 2014, pp. 1–8.

[13] K. Rothmeier, N. Pflanzl, J. Hüllmann, and M. Preuss, "Prediction of player churn and disengagement based on user activity data of a freemium online strategy game," *IEEE Transactions on Games*, 2020.

[14] Á. Periáñez, A. Saas, A. Guitart, and C. Magne, "Churn prediction in mobile social games: Towards a complete assessment using survival ensembles," in *Proc. IEEE DSAA*, 2016.

[15] H. Xie, S. Devlin, D. Kudenko, and P. Cowling, "Predicting player disengagement and first purchase with event-frequency based data representation," in *Proc. IEEE CIG*, 2015.

[16] R. Sifa, J. Runge, C. Bauckhage, and D. Klapper, "Customer lifetime value prediction in non-contractual freemium settings: Chasing high-value users using deep neural networks and smote," in *Proc. HICSS*, 2018.

[17] A. Drachen, M. Pastor, A. Liu, D. J. Fontaine, Y. Chang, J. Runge, R. Sifa, and D. Klabjan, "To be or not to be... social: Incorporating simple social features in mobile game customer lifetime value predictions," in *Proc. ACM ACSWM-IE*, 2018.

[18] P. Burelli, "Predicting customer lifetime value in free-to-play games," in *Data analytics applications in gaming and entertainment*. Auerbach Publications, 2019.

[19] R. Sifa, S. Srikanth, A. Drachen, C. Ojeda, and C. Bauckhage, "Predicting retention in sandbox games with tensor factorization-based representation learning," in *Proc. IEEE CIG*, 2016.

[20] X. Liu, M. Xie, X. Wen, R. Chen, Y. Ge, N. Duffield, and N. Wang, "A semi-supervised and inductive embedding model for churn prediction of large-scale mobile games," in *Proc. IEEE ICDM*, 2018.

[21] J. T. Kristensen and P. Burelli, "Combining sequential and aggregated data for churn prediction in casual freemium games," in *Proc. IEEE CoG*, 2019.

[22] H. Jäger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, 2004.

[23] R. Ramamurthy, R. Stenzel, R. Sifa, A. Ladi, and C. Bauckhage, "Echo state networks for named entity recognition," in *Proc. ICANN*. Springer, 2019.

[24] A. Garg, Y. Cao, and Q. Ge, "Echo state neural machine translation," *arXiv preprint arXiv:2002.11847*, 2020.

[25] J. Wieting and D. Kiela, "No training required: Exploring random encoders for sentence classification," in *Proc. ICLR*, 2019.

[26] X. Lin, Z. Yang, and Y. Song, "Short-term stock price prediction based on echo state networks," *Expert systems with applications*, vol. 36, 2009.

[27] C. Gallicchio, A. Micheli, and L. Pedrelli, "Deep echo state networks for diagnosis of parkinson's disease," *ArXiv:1802.06708*, 2018.

[28] A. Rodan and H. Faris, "Echo state network with svm-readout for customer churn prediction," in *2015 IEEE Jordan Conference on AEECT*. IEEE, 2015.

[29] R. Sifa, D. Paurat, D. Trabold, and C. Bauckhage, "Simple recurrent neural networks for support vector machine training," in *Proc. ICANN*. Springer, 2018.

[30] R. Sifa, R. Yawar, R. Ramamurthy, C. Bauckhage, and K. Kersting, "Matrix-and tensor factorization for game content recommendation," *Springer KI*, vol. 34, no. 1, 2020.