DL-DDA - Deep Learning based Dynamic Difficulty Adjustment with UX and Gameplay constraints

Dvir Ben Or
Playtika Research
Herzeliya, Israel
dvirb@playtika.com

Michael Kolomenkin
Playtika Research
Herzeliya, Israel
michaelko@playtika.com

Gil Shabat
Playtika Research
Herzeliya, Israel
gils@playtika.com

***Abstract*—Dynamic difficulty adjustment (DDA) is a process of automatically changing a game difficulty for optimization of the user experience. It is a vital part of almost any modern game. Most existing DDA approaches concentrate on the experience of a player without looking at the rest of the players. We propose a method that automatically optimizes the user experience while taking into consideration other players and the macro constraints imposed by the game. The method is based on a deep neural network architecture that involves a count loss constraint that has zero gradients in most of its support. We suggest a method to optimize this loss function and provide theoretical analysis of its performance. Finally, we provide empirical results of an internal experiment that was done on $200,000$ players and was found to outperform the corresponding manual heuristics crafted by game design experts.**

I. Introduction

Dynamic difficulty adjustment (DDA) is a process of automatically changing a game difficulty for the optimization of the user experience. The difficulty of a game should be just right so that a player does not get bored when the game is too easy and does not get frustrated when the game is too hard. DDA is usually applied to each player based on the player's abilities, skills and observed actions [1].

The inability of games to offer the right difficulty for everyone is considered one of the main reasons for players' discontent [2]. Players need a constant challenge to stay immersed in the game.

Recently, games have been moving from entertainment to other areas, such as healthcare [3] and education [4]. A well-designed game is more than a way to spend free time and to relax. It might be a doctor's or a teacher's tool. Thus, the ability to dynamically adapt the game difficulty for each player is becoming even more important.

DDA is a demanding task. Both industry and academia have been working on it for several decades, but it has not been completely resolved [1].

There are two major challenges in devising a good DDA method. The first one is a precise formulation of the user experience or the user engagement. As stated in the first paragraph, DDA is a process that *optimizes* the user experience. We have to define the user experience first before we can optimize it. The definition should hold the properties of a loss function if we want to utilize modern optimization tools [5] and should be based on the data available in the game.

The second challenge is interpretability and controllability. While DDA processes usually run automatically, games are managed by humans. The option to monitor and control the DDA output is vital for human operators.

In this work, we present a DDA system that addresses the aforementioned challenges. The system focuses on online games with many concurrent users. Specifically, the contribution of our paper is threefold:

1) **UX loss function**. We introduce a novel formulation of user experience. As opposed to previous methods, our formulation leverages not only the experience of the player for whom the difficulty is computed, but also the experience of all other players. It requires that the player's difficulty fits both the style of the player and the style of similar players. The formulation is generic and can be utilized in various games. We successfully employ the formulation as a loss function in a neural network that learns how to define the optimal difficulty based on the user's state.
2) **Completion rate constraint**. We show how to use the completion rate in a neural network. The completion rate is the percentage of players who finish a level or a task. It is often employed as an outside input to control the gameplay.
   Neural networks are the standard of modern machine learning. Thus, the ability to incorporate a mathematical constraint into a neural network is important for the application of the constraint to real life problems. Straightforward usage of the completion rate constraint in a neural network is difficult since the constraint is a piece-wise constant function and its gradient is zero almost everywhere.
   We propose using the completion rate in a variation of the projection gradient descent algorithm [6]. The algorithm projects the parameters of the neural network onto the feasible set defined by the completion rate constraint. We provide an alternation-based iterative procedure for the projection and give theoretical insights for the convergence of this procedure.
3) **A real world DDA system**. Finally, we present a DDA system that was tested in an online game with millions

978-1-6654-3886-5/21/$31.00 ©2021 IEEE

of daily users. The system is based on a deep neural network that optimizes the UX loss function mentioned in Contribution (1) under the gameplay Constraint (2). We show that the system outperforms manually managed DDA methods.

The paper continues as follows: Section II depicts the related work. Section III describes the loss function. Section IV explains how to integrate a common gameplay constraint completion rate into a neural network-based solution. Section V outlines the general architecture and compares the results of our approach with a manual method. Section VI provides theoretical analysis, including convergence analysis, for the method.

II. Related work

DDA has been an important research topic for the last several decades [1]. The research can be roughly divided into three main groups.

The first group searches for the optimal difficulty of the player's physical responses. For example, Stein *et al.* [7] adjusted the difficulty according to the player's EEG response and Wang *et al.* [8] used facial expressions to infer and adapt to the experienced difficulty. While obtaining promising results, those approaches require special environments and are hardly applicable directly to existing games.

The second group concentrates on the game state. The methods of that group usually define an ideal number or order of states in the game and adjust the game parameters so that the order is preserved. For instance, Yannakakis and Hallam [9] define *the appropriate level of challenge* as the variance of steps required for the game engine to "kill" the player in predator-prey games. The higher the variance, the more interesting the game is. The variance is computed over a set of games. When the difficulty is too small, all the games last too long. When the difficulty is high, the games end quickly. When the difficulty is right, some of the games end quickly and some last a long. Xue *et al.* [10] optimize the expected number of rounds in the game, while Sekhavat [11] employs a similar approach, by optimizing the difference between the number of losses and the number of wins of a player in multiple periods. The Hamlet system embedded in the *Half-Life* game engine [12], [13] assumes that the player should move between states according to the flow model. The system modifies the difficulty to increase the chance of relevant transitions, relying on observed statistics. Another approach is to maximize the speed of the player's progress by using a simulation reinforcement learning mechanism [14] and then applying it to real players. The game state approaches strive to achieve uniform movement of players through the game. It is an appropriate choice for some games, but a disadvantage for others, where each player may want to advance at her own rate or where the optimal state flow is difficult to define.

The third group deals with player skills. The general idea is that better players should get harder games. The methods of that group predict a player's abilities and performance and set the difficulty accordingly. For example, a system for Tower defence combines an estimation of player's skills with the enemy's potential [15]. Zook and Riedl [16] developed a method for predicting player's performance in real time. A stroke rehabilitation system uses a partially observable Markov model for estimating the player's abilities [17].

The above approaches are capable of personalizing the user experience, but they are game specific and do not provide a generic UX definition that can be applied to other games.

The approach presented here falls into the third category, but in addition to utilizing the data of a single player, we exploit the data of all concurrent players. We verify that players with similar styles have similar difficulties. Moreover, to the best of our knowledge, our approach is the only one that offers a way to integrate a global gameplay constraint into the UX optimization process.

III. UX loss function

Loss functions are a central part of any optimization system. It determines the errors that the system minimizes. The proposed loss function is called the UX loss function, since it optimizes the quality of the user experience, or, in other words, minimizes UX errors. User experience is complex, since it depends on many factors which are difficult to define [18]. In this paper, the UX loss function is mostly focused on difficulty and can be thought of as a "DDA loss function". Yet, for general and theoretical reasons, we continue with the term "UX" throughout the paper.

A. Terminology and assumptions

The goal is to set the difficulty $\hat{d}_i$ for each player i. We assume that the difficulty for all players is set for the same period of the game. Let's call the period T. The duration of the period can vary. The players do not have to participate in the game simultaneously.

We assume that there exists a one-to-one mapping between the game difficulty and player performance and that the mapping is known. Technically, it means that we know a one-to-one function $Pd = f(d)$ that maps the difficulty user$d$ to a game parameter Pd measurable from the game data. For instance, the difficulty may correspond to the number of objects a player needs to find, the number of levels needed to pass, the strength of the opponent needed to fight or any combination of them. This assumption also means that the actual difficulty d can be computed from the performance as $f^{-1}(Pd)$.

In addition, we assume that players can be clustered into homogeneous groups. The details of the clustering are explained in Section V.

B. Loss definition

The loss function combines two components. The first component ensures that the advancement in the game will fit the player personally. The idea is that if the required performance deviates too much from the actual performance, the player will find the game either too difficult or too easy. This can be linked to a positive experience. The second component is that the requirements of similar players should be similar. The intuition is that a correctly designed user experience should not change much among players that are comparable to each other. This can be associated with game fairness and algorithm stability. Mathematically, the loss function is defined as:

$$\text{UX Loss}(\hat{\mathbf{D}}) = \text{var}(\hat{\mathbf{D}}) + \frac{\alpha}{M}\sum_{i=0}^{M}\left(d_i - \hat{d}_i\right)^2, \tag{1}$$

where d_i denotes the actual difficulty of player i, $\hat{d}_i$ is the difficulty which we aim to find, $\hat{\mathbf{D}}$ denotes the set of required difficulties of all the players in the cluster of player i:

$$\hat{\mathbf{D}} = \{\hat{d}_0, \hat{d}_1, \ldots \hat{d}_M\},$$

where M is the size of the cluster and α is a parameter that controls the relative weight of the two parts of the loss function. In our experiments, we gave equal weights to both parts, i.e. $\alpha = 1$. Further investigation can be done in order to determine the effect of the α on the actual user experience.

C. Loss Optimization

Conceptually, the optimization of Equation 1 can be thought of as a two-step process. The first step is to predict the actual difficulties d_i. This can be done with a neural network. The second step is to optimize the UX loss given the difficulties involved. The loss is a convex function and the optimization can be performed with any gradient based method.

We use a single neural network for the two-step process above. The UX loss is used as the loss function of the network. The difficulties are not predicted explicitly. The network learns to set the required difficulties that minimize the loss given the actual difficulties provided in the training process and given the player behaviour in the period T' before the relevant gaming period T. In our experiments, we set the duration of T' to the duration of T. Formally, the network is defined as:

$$\hat{\mathbf{D}} = N(\mathbf{\Theta}, \mathbf{X}), \tag{2}$$

where $N(\mathbf{\Theta}, \mathbf{X})$ represents the network, $\mathbf{\Theta}$ represents network parameters and $\mathbf{X} \in R^{MxZ}$ represents input features of dimension Z. The input features contain the states and the actions of a player on each day during T'.

Algorithm 1 summarizes the training procedure of the network: The network computes the required difficulties from the input features at inference time. More details about the network appear in Section V.

Algorithm 1 Train NN to minimize UX Loss

Require:
$N(\mathbf{\Theta}; \mathbf{X})$ - neural network, $\mathbf{\Theta}$ - initial network weights, $\mathbf{D} = \{d_0, d_1, \ldots, d_M\}$ - difficulty per user during T, $\mathbf{X}$ - input features during T'.

Ensure:
$\hat{\mathbf{D}}$ - the set of required difficulties that optimizes Equation 1.
$\hat{\mathbf{\Theta}}$ - optimized network weights for the required difficulties.

1: Apply a stochastic gradient descent to train the network.
2: **return** $\mathbf{\Theta}$

IV. Completion rate constraint

In this section, we show how to apply the completion rate constraint together with the neural network in Equation 2.

a) Completion rate: The completion rate is a percentage of users that complete a certain goal or a series of goals in a game. The goal might be a task, a level, a mini sub-game or any game feature. Usually, the completion rate is high for the first levels (easier ones) and gradually decreases with the advancement of the game levels. It can be thought of as a parameter that determines how challenging a game feature is.

It seems natural to use completion rate as an optimization constraint for a DDA process in general and the UX loss function defined in Section III in particular. Indeed, it allows us to optimize user experience while providing additional gameplay constraints.

However, employing the completion rate directly in an optimization is not trivial. The completion rate is a piecewise constant function of the game difficulty and, thus, its gradient is zero everywhere except at a finite number of points. For instance, we assume that when the difficulty is zero, the completion rate is one hundred percent, i.e. all players complete the given task. Increasing difficulty has no impact on the completion rate until the difficulty is high enough for one player to quit before completing it. Then it has no effect again until the following player is unable to complete it, and so on.

Optimization of functions with zero gradients is a complex problem, especially for neural networks. For some problems, they can be solved using reinforcement learning [19]. For others, approximations and surrogate losses are used [20]. Both solutions introduce their own problems.

Instead, we suggest using a variation of projected gradient descent. The idea is that the weights of a neural network can be projected onto the subspace where the completion rate constraint holds. The projection is performed at each iteration of the learning process. The projection itself is also an iterative procedure described below.

b) Problem definition: Let P be the desired completion rate, M be the number of players, d_i is the actual difficulty (performance) of the training set and $\hat{d}_i$ is the desired difficulty (prediction) for player i as defined in Equation 1. Then, the constraint is defined as:

$$\hat{P} \triangleq \frac{1}{M}\sum_{i=1}^{M}\mathbb{1}[d_i \geq \hat{d}_i] = P \tag{3}$$

where $\mathbb{1}[x]$ is an indicator function:

$$\mathbb{1}[x] := \begin{cases} 1 & \text{if } x \text{ is true} \\ 0 & \text{if } x \text{ is false} \end{cases}$$

and $\hat{P}$ is termed the *achieved* completion rate, which is the number of players who were able to complete the challenge (a "count" function) divided by the number of players.

c) Projection: The goal of the projection is to change the weights of the neural network $N(\mathbf{\Theta}, \mathbf{X})$ so that Constraint 3 is satisfied. The projection is performed during training each time the neural network converges. After the projection, the neural network is trained again from the projection point.

The projection works by making the loss function of $N(\mathbf{\Theta}, \mathbf{X})$ roughly proportional to the absolute value of the difference $\hat{P} - P$ in Constraint 3. When the difference is positive, the achieved completion rate is higher than the desired completion rate. Hence, the desired difficulty $\hat{d}_i$ should raise. When the difference is negative, $\hat{d}_i$ should be lowered.

Practically, we saw that good results are obtained when the loss function is equal to the average of the desired difficulties $\hat{d}_i$ in the previous iteration when $\hat{P} - P$ is positive, and to the minus average of $\hat{d}_i$ when it is negative, though other possibilities can be chosen, as long as the shift of the weights is done in the right direction, i.e. to increase or decrease the completion rate.

The average of $\hat{d}_i$ is not guaranteed to be proportional to $||\hat{P}||$. For example, only the required difficulty of a single player may be raised all the time, keeping $\hat{P}$ constant while changing the average of the required difficulty. However, it is hardly possible in practice, since the neural network is already trained to compute all the required difficulties. It is very difficult to change the network weights so that only a few difficulties will be influenced.

Algorithm 2 summarizes the projection method.

Algorithm 2 Projection to optimize completion rate

Require:
η - learning rate, P - desired completion rate, $N(\mathbf{\Theta}; \mathbf{X})$ - neural network, $\mathbf{\Theta}$ - initial network weights, $\mathbf{D} = \{d_0, d_1, \ldots, d_M\}$ - actual difficulty during T
Ensure:
$\hat{\mathbf{\Theta}}$ - optimized network weights that achieve P

repeat
  // *compute model outputs*
  $\hat{\mathbf{D}} = N(\mathbf{\Theta}; \mathbf{X})$
  // *compute hypothesized completion rate*
  $\hat{P} = \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}[d_i \geq \hat{d}_i]$
  if $\hat{P} < P$ **then**
    // *the computed rate is higher than desired*
    $err = \frac{1}{M} \sum_{i=1}^{M} \hat{d}_i$
  else if $\hat{P} > P$ **then**
    // *the computed rate is lower than desired*
    $err = -\frac{1}{M} \sum_{i=1}^{M} \hat{d}_i$
  else
    // *The desired rate P is reached*
    return $\mathbf{\Theta}$
  // *update model parameters*
  $\mathbf{\Theta} \leftarrow \mathbf{\Theta} - \eta \frac{\partial err}{\partial \mathbf{\Theta}}$
until max iterations or convergence
return $\mathbf{\Theta}$

V. System overview

This section describes how the UX loss (Equation 1) and the completion rate constraint (Equation 3) are combined into a DDA system for online games with millions of daily users. The system was used in an internal experiment on a specific feature inside a game and outperformed the corresponding heuristic-based manual difficulty settings.

The DDA system consists of two stages. The first one, called *Clustering*, divides the players into homogeneous groups. The second one creates an iterative mechanism for minimizing the UX loss and applying the completion rate constraint.

a) Clustering: The variance part of Equation 1 represents the user experience more accurately when the players resemble each other. In general, players may differ significantly. There are players who have just installed the game and there are players who have been in the game for several years. The players may have different tastes, preferences and gaming styles. It makes more sense to require similar difficulties for similar users.

We assume that there exists a similarity function $S(p_i, p_j)$ between players i and j. The function defines the distance between the players. The smaller the distance, the more similar the players are to each other. The purpose of the similarity function is to divide players into homogeneous clusters.

We cluster players with a K-Means algorithm, but any other algorithm would do. We define the similarity function as a normalized Euclidean distance between the input features $\mathbf{X}$ from Equation 2. We note that the precise definition of similarity is unimportant as long as the players are divided into smaller groups with comparable properties.

b) Iterative optimization: A single projection of the network weights as described in Section IV is insufficient. When the projection is done, the weights $\mathbf{\Theta}$ are altered and the neural network no longer achieves the minimal error. It has to be retrained. The whole procedure is repeated until convergence.

Algorithm 3 outlines the flow of the whole system.

Figure 1 provides an illustration of the training procedure and the convergence of our approach to a single cluster. The longest part of the training procedure is the first optimization cycle in which the UX loss is minimized. It can be seen from the lower, zoomed-in part of the figure, that the system converges both when the batch values change and when the loss switches from UX to projection.

A. Implementation details

We use a fully connected neural network with 5 hidden layers. The dimension of the input is 40. The input vector has a variety of aggregated player parameters for a two week period before a small mini-game is optimized using this approach.

We used 200 clusters and required a minimum of 5,000 players in each cluster is 5,000. The system runs on an NVIDIA DGX computer. The whole process for a million users takes around half an hour.

Algorithm 3 Full DDA system

Require:
K - Number of clusters, P - Desired completion rate, $N(\mathbf{\Theta};\mathbf{X})$ - neural network architecture, $\mathbf{D} = \{d_0, d_1, \ldots, d_M\}$ - actual difficulty at T

Ensure:
$\{\hat{\mathbf{D}}_k\}_{k=1}^{K}$ - the required difficulty for every player of cluster k

Initialize neural network parameters $\mathbf{\Theta}_k$ with Xavier [21]
for k in range(K) **do**
    assign $\mathbf{X}$ with the k^{th} cluster features data set
    assign $\mathbf{D}$ with the corresponding actual difficulty
    repeat *// alternation cycles*
        // optimize UX loss
        update $\mathbf{\Theta}_k$ by applying $N(\mathbf{\Theta}_k;\mathbf{X})$ to optimize Eq. 1
        // Project weight to ensure completion rate
        update $\mathbf{\Theta}_k$ by applying algorithm 2
    until max iterations or convergence
    compute $\hat{\mathbf{D}}_k = N(\mathbf{\Theta};\mathbf{X})$

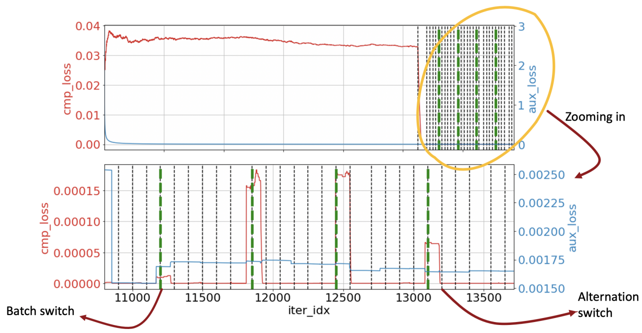

Fig. 1. Training procedure illustration. The horizontal axis represents the model's training steps. The red curve (corresponds to the left vertical axis) displays the l_2 distance between the evaluated completion rate and the desired one, while the blue curve (corresponds to the right vertical axis) describes the loss formulated in Eq. 1. Dashed black vertical lines indicate the switch from UX loss to completion rate projection and vice versa. The green bold vertical lines indicate the replacement of batch samples used for optimization. For convenience, the lower chart provides a closer view of the optimization trajectory described in the upper one, focused on the stage when the optimized objectives converge.

B. Results

We performed an A/B test to verify the validity of our approach. The output was compared to our system with the difficulty levels set by a rule-based method currently used by game operators. The rule-based system is the result of several years of trial and error. It is a collection of *if-else* decisions applied to a variety of game parameters. It incorporates a great deal of knowledge about the game and generally provides satisfactory results.

As shown below, our approach outperformed the rule-based method. In addition to being superior in accuracy, our approach is automatic and saves time for game operators. Each change in the game mechanism requires manual adaptation of the rule-based system. The manual process is time consuming and prone to errors.

The test was carried out in an eight-day mini game (a feature inside one of Playtika's games) where the goal was to optimize the number of points each player had to obtain. About 800,000 players were in the control group and received rule-based difficulties, while about 200,000 players were in the test group receiving machine learning-based difficulties.

Target	Rule based	Our approach
8-10%	12.0%	8.7%

TABLE I

Comparison of the target completion rate with the result achieved by the rule based method and our approach

Table I compares the average completion rate achieved by our approach and the rule-based method with the target range defined by the product team. The result of the rule-based method is fine by the practical standards of the game, but our approach still outperforms it.

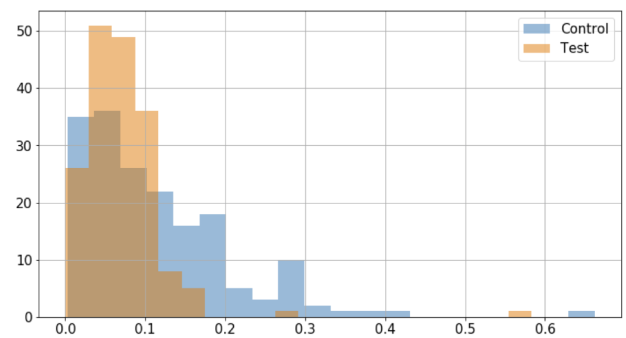

Fig. 2. A histogram of completion rates by clusters. The X-axis is the completion rate, and the Y-axis is the number of clusters that have the completion rate. The blue group is the control group, which is a rule-based method, and the orange group is our system.

The difference between the methods is even more pronounced when we look at the distribution of the completion rates. This is where our approach really shines. Figure 2 shows the histogram of the completion rate by the clusters. Recall that the clusters are homogeneous groups of players computed by the K-Means algorithm. The variance in the completion rate of the control group is much higher than that of the test group.

There are 200 clusters. The number 200 was chosen since it created homogeneous clusters on the one hand and yet contained a relatively large number of players (around 1,000) on the other hand. In the control group, 49 clusters had completion rates of more than 16%. In the test group, only 5 clusters had completion rates of more than 16%.

The implication is that, while the rule-based method achieves a satisfactory average completion rate, it falls short of reaching a sizable proportion of the population. The ability to control the completion rate of every sub group of players is another advantage of our approach.

VI. Theoretical Analysis

This section describes the conditions for the convergence of the algorithm and provides some theoretical insights and, in a sense, the derivation is a bit similar to [22]. The convergence does not assume convexity, but it does assume certain properties of the non-linear projection operators. Those properties depend on the function determined by the structure of the neural net and its loss function. The first projection operator returns the nearest local minimum point.

Definition 1. *Given a training dataset* $\mathbf{X}$*, a neural net* $N(\mathbf{\Theta}, \mathbf{X})$ *with weights* $\mathbf{\Theta}$ *and a set of local minimum* $\mathcal{M}$*, then* $\mathcal{P}_{\mathcal{M}} N(\mathbf{\Theta}, \mathbf{X})$ *returns the closest weights of the nearest local minimum:*

$$\mathcal{P}_{\mathcal{M}} N(\mathbf{\Theta}, \mathbf{X}) = \underset{N(\hat{\mathbf{\Theta}}, \mathbf{X}) \in \mathcal{M}}{\arg\min} \|\Theta - \hat{\Theta}\|_2 \tag{4}$$

The second operator returns the closest set of weights that satisfies the completion rate. Based on Eq. 3, it is possible to define a set of valid solutions.

Definition 2. *Let* C *be the set of all possible weights, given a set of predicted difficulties* $\{d_i\}$*, desired completion rate* P *and tolerance* δ *such that*

$$C = \left\{ \mathbf{\Theta} | \quad |\frac{1}{M} \sum_{i=1}^{M} \mathbb{1}[d_i \geq N(\mathbf{\Theta}, \mathbf{X})] - P| \leq \delta \right\} \tag{5}$$

Definition 3. *Given a training dataset* $\mathbf{X}$*, a neural net* $N(\mathbf{\Theta}, \mathbf{X})$ *with weights* $\mathbf{\Theta}$ *and a set of valid completion points* C *(Definition 2), then* $\mathcal{P}_C N(\mathbf{\Theta}, \mathbf{X})$ *returns the closest weights in* C*:*

$$\mathcal{P}_C N(\mathbf{\Theta}, \mathbf{X}) = \underset{N(\hat{\mathbf{\Theta}}, \mathbf{X}) \in C}{\arg\min} \|\Theta - \hat{\Theta}\|_2 \tag{6}$$

The operators $\mathcal{P}_{\mathcal{M}}$ and $\mathcal{P}_C$ are approximately implemented by Algorithm 1 and Algorithm 2, respectively. The algorithms return a local minimum ($\mathcal{P}_{\mathcal{M}}$) or a completion-valid point ($\mathcal{P}_C$) by the application of a stochastic gradient descent (or other optimizer), but do not guarantee to return the closest point, since it depends on the structure of the neural network, which is typically a high-dimensional non-convex manifold. This is different than the case in [22].

Optimizing the UX loss under the completion rate constraint, can be done by the following alternating scheme, which is approximately implemented by Algorithm 3:

$$\Theta_i^M \leftarrow \mathcal{P}_{\mathcal{M}} \Theta_i^C \tag{7}$$

$$\Theta_{i+1}^C \leftarrow \mathcal{P}_C \Theta_i^M \tag{8}$$

starting from an arbitrary point (random initialization of the weights).

Proposition 1. *Let* Θ_i^C *and* Θ_i^M *(*$i \geq 1$*) be a set of points (weights) obtained by a consecutive application of the alternation scheme (Eqs. 7 and 8) then the series* $\|\Theta_i^C - \Theta_i^M\|$ *converges.*

Proof. Since $i \geq 1$, then according to Eq. 8, $\Theta_i^C \in C$ (Def. 2). By the definition of $\mathcal{P}_{\mathcal{M}}$, Θ_i^M is the closest local minima to Θ_i^C and by the definition of $\mathcal{P}_C$, Θ_{i+1}^C is the closest valid count-constraint point to Θ_i^M. Since $\Theta_i^C \in C$ and $\Theta_{i+1}^C \in C$ is the closest point to Θ_i^M

$$\|\Theta_i^M - \Theta_{i+1}^C\| \leq \|\Theta_i^M - \Theta_i^C\|. \tag{9}$$

By the definition of $\mathcal{P}_{\mathcal{M}}$, $\Theta_{i+1}^M \in \mathcal{M}$ is the closest local minima to Θ_{i+1}^C. Since $\Theta_i^M \in \mathcal{M}$

$$\|\Theta_{i+1}^C - \Theta_{i+1}^M\| \leq \|\Theta_i^M - \Theta_{i+1}^C\| \tag{10}$$

Combining Eqs. 9 and 10 gives

$$\|\Theta_{i+1}^M - \Theta_{i+1}^C\| \leq \|\Theta_i^M - \Theta_i^C\|.$$

Since $\|\Theta_i^M - \Theta_i^C\|$ is monotonically decreasing and bounded and so it converges, which completes the proof. □

Proposition 1 states that the distance between a valid completion rate point and a local minimum point is monotonically decreasing and eventually converges. An interesting observation from the proposition is that it tells us where to look for the next minimum/valid completion rate point, which enables to decrease the step size of the SGD proportionally to the distance between the two points.

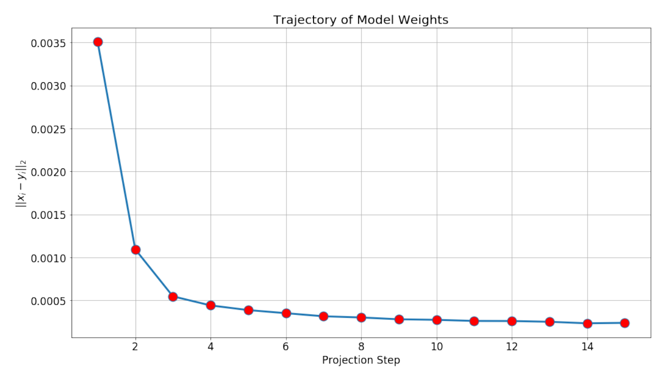

Fig. 3. The distance between the weights of a valid completion rate point to the local minimum followed by the application of $\mathcal{P}_C$ to it

Figure 3 depicts the convergence of Algorithm 3 to illustrate Proposition 1 on real data. The figure shows that the distance between a local minimum and its corresponding completion rate valid point shrinks with each iteration. Interestingly, the algorithm achieves a monotonically decreasing curve even for the last iterations, when the distance is small. This happens even when the projection operators are only approximated and do not satisfy the strict requirements of their definition for finding the closest point.

Additionally, the following observations infer directly from Proposition 1:

- Since the distance between a valid completion point and a local minimum converges, it eventually means (excluding pathological cases of points having exactly the same distance) that the algorithm iterates between one local minimum and one valid completion rate point. Therefore, it converges to a specific local minimum/completion rate point.

- The difference in the model's performance between those two points depends on the distance and the Lipschitz constant of the neural network [23]. So if the distance is small (and hopefully the Lipschitz constant), then stopping at a completion rate point or at a local minimum should not make a big difference.

For more details the reader is referred to [24].

Conclusion

This paper presents a system for dynamic adjustment of game difficulty. The system was tried on an online game with millions of daily users and significantly outperformed the manual heuristics used by the game developers. The system is based on several innovations. First, it presents a formulation of user experience that depends on all the similar players. The formulation exploits more information than the existing methods.

Second, it shows how to incorporate a completion rate constraint into a neural network. The completion rate constraint is important for creating a fun experience. Straightforward application of the constraint to a neural network is difficult, since the gradients of the constraint are piece-wise zero. The paper also presents a theoretical analysis of the convergence of the neural network.

While the system was applied to a specific game, it is very flexible and can easily be adapted to other games. All that one needs in order to employ our system is a definition of difficulty that can be computed from the game data, such as objects collected, monsters slain, etc., and a definition of the similarity between the players. Then the system can learn the appropriate difficulties for any required period of time.

A possible drawback of our approach is that it does not allow us to change the difficulties during the predefined period. If the chosen difficulty is too hard and the player does not advance in the game, it will stay hard. We plan to research how to incorporate real- time-information into our approach.

References

[1] M. Zohaib, "Dynamic Difficulty Adjustment (DDA) in Computer Games: A review," Advances in Human-Computer Interaction, vol. 2018, pp. 1–12, 2018.

[2] R. Koster and W. Wright, A Theory of Fun for Game Design. Arizona: Paraglyph Press, 2004.

[3] A. DeSmet, D. Thompson, T. Baranowski, A. Palmeira, M. Verloigne, and I. De Bourdeaudhuij, "Is Participatory Design Associated with the Effectiveness of Serious Digital Games for Healthy Lifestyle Promotion A Meta-Analysis?" Journal of medical Internet research, vol. 18, no. 4, p. e94, 2016.

[4] T. M. Connolly, E. A. Boyle, E. MacArthur, T. Hainey, and J. M. Boyle, "A systematic literature review of empirical evidence on computer games and serious games," Computers & education, vol. 59, no. 2, pp. 661–686, 2012.

[5] L. Rosasco, E. D. Vito, A. Caponnetto, M. Piana, and A. Verri, "Are Loss Functions All the Same?" Neural Computation, vol. 16, no. 5, pp. 1063–1076, 2004.

[6] A. N. Iusem, "On the convergence properties of the projected gradient method for convex optimization," Computational & Applied Mathematics, vol. 22, no. 1, pp. 37–52, 2003.

[7] A. Stein, Y. Yotam, R. Puzis, G. Shani, and M. Taieb-Maimon, "EEGTriggered Dynamic Difficulty Adjustment for Multiplayer Games," Entertainment computing, vol. 25, pp. 14–25, 2018.

[8] Z.-X. Wang, H.-M. Lee, and W.-H. Lee, "Adjusting the Difficulty of Running Game with Facial Expression Recognition Technology Using Convolutional Neural Network," KoreaScholar - journal of information systems, vol. 31, no. 2, pp. 39–46, 2018.

[9] G. N. Yannakakis and J. Hallam, "Towards Optimizing Entertainment in Computer Games," Applied Artificial Intelligence, vol. 21, no. 10, pp. 933–971, 2007.

[10] S. Xue, M. Wu, J. Kolen, N. Aghdaie, and K. A. Zaman, "Dynamic Difficulty Adjustment for Maximized Engagement in Digital Games," in Proceedings of the 26th International Conference on World Wide Web Companion. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2017, pp. 465–471.

[11] Y. A. Sekhavat, "Mprl: Multiple-Periodic Reinforcement Learning for Difficulty Adjustment in Rehabilitation Games," in 2017 IEEE 5th international conference on serious games and applications for health). Perth, Australia: IEEE, 2017, pp. 1–7.

[12] R. Hunicke and V. Chapman, "AI for dynamic difficulty adjustment in games," Challenges in game artificial intelligence AAAI workshop, vol. 2, 01 2004.

[13] R. Hunicke, "The Case for Dynamic Difficulty Adjustment in Games," in Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology, ser. ACE '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 429–433. [Online]. Available: https://doi.org/10.1145/1178477.1178573

[14] J. Togelius, R. De Nardi, and S. M. Lucas, "Making Racing Fun Through Player Modeling and Track Evolution," Optimizing Player Satisfaction in Computer and Physical Games, vol. 2, no. 1, p. 61, 2006.

[15] R. Sutoyo, D. Winata, K. Oliviani, and D. M. Supriyadi, "Dynamic Difficulty Adjustment in Tower Defence," Procedia Computer Science, vol. 59, pp. 435–444, 2015.

[16] A. E. Zook and M. O. Riedl, "A Temporal Data-Driven Player Model for Dynamic Difficulty Adjustment," in Eighth Artificial Intelligence and Interactive Digital Entertainment Conference. Atlanta, Georgia, USA: AAAI Press, 2012, pp. 20–28.

[17] R. Goetschalckx, "Games with Dynamic Difficulty Adjustment using POMDPs," in ICML Workshop. San Francisco, CA, United States: Morgan Kaufmann Publishers, 2010, pp. 8–14.

[18] J. McCarthy and P. Wright, "Technology as Experience," interactions, vol. 11, no. 5, pp. 42–43, 2004.

[19] S. Xie, Z. Chen, C. Xu, and C. Lu, "Environment upgrade reinforcement learning for non-differentiable multi-stage pipelines," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. New York, NY, USA: IEEE, 2018, pp. 3810–3819.

[20] J. Grabocka, R. Scholz, and L. Schmidt-Thieme, "Learning Surrogate Losses," arXiv preprint arXiv:1905.10108, vol. 1, no. 1, pp. 1–10, 2019.

[21] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in Proceedings of the thirteenth international conference on artificial intelligence and statistics. United States: Journal of Machine Learning Research, 2010, pp. 249–256.

[22] G. Shabat and A. Averbuch, "Interest Zone Matrix Approximation," The Electronic Journal of Linear Algebra, vol. 23, 2012.

[23] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. Pappas, "Efficient and Accurate Estimation of Lipschitz Constants for Deep Neural Networks," in Advances in Neural Information Processing Systems, 2019, pp. 11 427–11 438.

[24] D. B. Or, M. Kolomenkin, and G. Shabat, "Generalized Quantile Loss for Deep Neural Networks," arXiv preprint arXiv:2012.14348, 2020.