Player Modeling using HOSVD towards Dynamic Difficulty Adjustment in Videogames

Kostas Anagnostou[*] and ManolisMaragoudakis[+]

*Blitz Game Studios, United Kingdom
IonianUniversity, Department of Informatics, Corfu, Greece, kostasan@ionio.gr
[+]University of the Aegean, Department of Information and Communication Systems Engineering, Samos, Greece
mmarag@aegean.gr

Abstract. In this work, we propose and evaluate a Higher Order Singular Value Decomposition (HOSVD) of a tensor as a means to classify player behavior and adjust game difficulty dynamically. Applying this method to player data collected during a plethora of game sessions resulted in a reduction of the dimensionality of the classification problem and a robust classification of player behavior. Simultaneously HOSVD was able to perform significant data compression without significant reduction as regards to the accuracy of the classification outcome and furthermore, was able to alleviate the data sparseness problem common within data collected from game sessions.

Keywords: Clustering algorithms, games, user modeling, Tensor Analysis

1 Introduction

Designing a successful videogame depends among others on achieving a delicate balance between difficulty and user skill. Csikszentmihalyi's influential work on Flow [1] dictates that in order to accomplish the optimal user experience, any interactive process (such as videogames) must tread constantly between two states, one of frustration and the other of boredom, never reaching any of them. Frustration in games can be the result of game challenges that are too hard for the user to achieve, while on the other hand boredom can result from game tasks that are too easy. Both states will undermine the player experience and are undesirable.

In order to avoid those two states, ideally, a videogame must constantly match the difficulty of the game challenges to the player capabilities. This entails providing the player with increasingly harder tasks, as her ability to solve those increases. Considering the huge player base, and the great variation in gameplaying experience, player capability and free time to devote to games, designing a game to satisfy many is no easy task.In an attempt to match game difficulty to player abilities, game designers have traditionally provided a “difficulty” setting. The player has to choose a priori between 3 or 4 difficulty levels (easy-medium-hard) before the game starts, with no real feeling of how well the difficulty level suits her abilities. Based on the chosen difficulty level, the game adjusts the number of enemies, enemy AI, player aim precision etc. A problem with this approach is that the user might misjudge the suitability of the selected difficulty level and her game playing experience might be thwarted by a too easy or too difficult game.Ideally, the game system should be able to monitor user performance and based on such data to accommodate for her playing abilities by adjusting the difficulty of the game to match those abilities, in order to maintain a high level of player engagement throughout. Recently, an increasing number of games, such as Microsoft's Halo Wars and SiN Episodes, are opting for a Dynamic Difficulty Adjustment (DDA) system, where the game changes the difficulty based on the runtime player performance [2][3]. DDA during game-playing can be numerical and/or structural. Numerical adjustment amounts to altering parameters such as the number of enemies, enemies' shield and weapon power and accuracy, using statistical models of player performance, in order to make the game easier or harder according to player skill [4][5]. According to Booth [6] Valve utilized such an approach be dynamically altering the location and frequency of spawn points for enemies and items in Left for Dead based on player performance. Structural adjustment on the other hand amounts to changing the structure of a game level dynamically according to player performance, using procedural level generation. Procedural level generation can be offline, i.e. before the game starts, such as in the case of Pedersen et al [7] who derive a statistical model of player challenge and frustration and by using evolutionary algorithms generate levels that match a particular level of player challenge. There applications on the other hand, such as Polymorph, which adjust the structure of the level during gameplay by generating sections of a level ahead of the player's movement, allowing a level to change in difficulty from start to finish in response to changes in the player's performance [8]. Yet, the basis of any approach for dynamic difficult adjustment, be it numerical or structural, offline or in the runtime, is the successful specification of the model of player behavior and performance from data gathered during gameplay. Any dynamic adjustment will made be according to this model.

Towards the extraction of the player model, in this paper we extend the work in [9] by proposing the use of higher order singular value decomposition (HOSVD) of a tensor [10] for classification of a player, in order to maintain a high level of engagement. The suggested algorithm uses HOSVD to compute a small set of basis matrices that span the dominant subspace for each class of players. The basis matrices are then used to describe unknown players/games. Furthermore, the proposed methodology is able to exploit sparse data, such as the task at hand, where numerous games are difficult to be collected. This algorithm is closely related to SIMCA [11] and PCA. Experimental results show interesting outcomes that deal with both increased classification accuracy and data compression.

Throughout recent years, tensor methods have been successfully applied to problems in pattern recognition and other areas. Tensors are known to be multidimensional or multimode matrices. In a plethora of real world pattern recognition applications, the data have a multidimensional structure and it is then somewhat unnatural to organize them as matrices or vectors. As an example, consider a video segment. Each image is a two-dimensional matrix of integers and together with images from different time frames the data constitute a third order tensor (three-dimensional data matrix). While most Machine Learning algorithms require data to be reformed as vectors, in many circumstances, it is beneficial to exploit the collected data without destroying its inherent multidimensional structure.

2 Methodology

The video game used for data collection was based on the Space Debris prototype developed at Ionian University. It is a top-down scrolling space action game in which the player has to defend Earth against waves of incoming alien invaders. The user can utilize a variety of weapons as well as the environment to eliminate the advancing menace.In this prototype, the action takes place within the confines of a single screen in which alien ships scroll downwards. There are two kinds of enemy spaceships, the cargo which is slow and has a stronger shield that can withstand more laser blasts and the fighter which is fast and easier to destroy. The cargo ship will maintain its course and will not attack the player except when she is directly in front of it. On the other hand the fighter ship has a more aggressive behavior and will alter its course in order to hunt and attack the player. At the end of each level, the user has to confront a large boss space ship which is equipped with strong shields and has more lethal weapons.

The player's ship has a shield that can absorb enemy blasts. When this reaches zero, the player loses a life. When all three available lives have been lost, the game ends. The player wins only if she destroys the boss ship at the end of the level.

The game environment is littered with floating asteroids which can be used as a weapon against enemy ships. In their default state the asteroids do no interact (i.e. collide) with any of the game spaceships. In order to do so, an asteroid has to be "activated", i.e. hit by a player weapon. The asteroids never destroy the player's ship on collision. Also floating are shield and life power-ups which the user can use to replenish her ship's shield and increase lives.

The following list enumerates the types of weapons available to the player

- A laser cannon which can used to shoot alien ships. The laser canon is weak and about 2-3 successful shots are required to destroy an enemy ship (except for the boss ship which requires many more). The laser can also be used to "activate" an asteroid and deflect it towards an enemy ship.
- An asteroid-grab ray, which can capture any asteroid in the player's ship area, activate and hurl it, slingshot-like, towards an enemy ship.
- A blast wave which centers on the player's ship position and expands outwards in a circular fashion, activating all asteroids in the expanding area. The activated asteroids will destroy any enemy ship they collide with.
- A freeze wave, which centers on the player's ship position and expands outwards in a circular fashion, immobilizing all asteroids in the expanding area. An immobilized asteroid can act as a shield to enemy fire and can be activated and set in motion at any time, and guided towards enemy ships

The game design specifies the use of two main weapon mechanics: direct enemy ship destruction using the laser and indirect destruction through an activated asteroid. Table 1 summarizes the balance of the two mechanics (laser vs activated asteroid) in terms of enemy destruction effectiveness.

Table 1. Weaponbalance

Weapon	**CargoShip**	**FighterShip**
Laser	3 successful shots per ship	2 successful shots per ship
Activated Asteroid	Instant death, many ships	Instant death, many ships

This dichotomy between ease of use (laser) and effectiveness (activated asteroids) creates a choice for players to either use the fast but ineffective laser to kill enemies or to be more tactical and efficient by directing asteroids towards them. The first weapon mechanic is easier to use but not as efficient because each enemy ship's shield can withstand many laser hits. The second weapon mechanic is slightly harder to use, but highly efficient since the asteroid will destroy any enemy ship on its trajectory that happens to collide with.Based on the choice of game tactic, we initially assume two player types: The action-player which favors weapon simplicity over efficiency, and the tactical-player which favors weapon efficiency over ease of use.For those reasons we chose to use, as an input device, a videogame controller, and more specifically the standard Xbox360 controller which can be connected to a PC via a USB port, instead of a keyboard. The Xbox controller, as well as other videogame controllers, features 4 symmetrical buttons (labeled A, B, X and Y) which we use to map each of the 4 weapons, reducing the effect of weapon choice bias.

The video game was developed using XNA Game Studio 3.1, on the Windows XP platform. During gameplay, we recorded automatically, with a new code module developed especially for the game, a large set of parameters that describe the way each player approaches the game, be it in an action or more tactical style.

Data collection consisted of a single session in which 210 students from the Ionian University and the University of the Aegean played the game 4 times each (840 records in total). In addition to the player behavior data collected for each player we recorded the player gender, and whether or not he/she is an experienced game player. The sample population consisted of 150 males and 60 females. 66,67% of the male users were regular videogame players, and 26,6% of males had some game playing experience while only 0,66% had no experience at all. 50% of the female users had some game playing experience and 40% in the sample set of females had no experience at all 10% of female users declared themselves as experienced players.

For each game we collected a record of data illustrated in Table 2.

Table 2. Data Collectedforeach Game

Parameter	Description
Time	Time required to finish the current game
Accuracy	Percentage of player laser blasts that found a target
LasersAvoided	Number of enemy lasers the user avoided
Lasers	Number of laser blasts fired by the user
BlastWaves	Number of blast waves used
Freezers	Number of asteroid freezers used
Grabbers	Number of asteroid grabbers used
KeysPerSecond	User key press rate
Carriers	Number of enemy carriers destroyed
Fighters	Number of enemy fighters destroyed
KilledByLaser	Number of enemies killed by laser
KilledByAsteroid	Number of enemies killed by asteroid
LasersHitAsteroid	Number of user laser blasts that hit an asteroid
LasersHitEnemy	Number of user laser blasts that hit an enemy
LasersMissed	Number of user laser blasts that missed
ShieldsUsed	Number of shield upgrades used
LivesUsed	Number of life upgrades used
BossTime	Time required to destroy the Boss ship
Result	Whether the player won the game or not

3 User Modeling with HOSVD

3.1 Introduction to Tensors

Informally speaking, an N^th order tensor is an object with N indices. The “dimensions” of a tensor will be also referred to as modes. Vectors and matrices can be considered as first and second order tensors, respectively. Within the framework of this article, the order is defined as N = 3. Therefore, for reasons of simplicity, most of the theoretical issues concerning tensors shall be stated for third order tensors noted as:

$\mathrm{T} \in \Re^{I\times J\times K}$, where *I,J,K* are integers greater than zero. The dimensionality of the vector space $\Re^{I\times J\times K}$ is I*J*K. The aforementioned notation can be straightforwardly generalized to include further dimensions. It is clear that a tensor of order 0 is a scalar, while a tensor of order 1 is a vector and a tensor of order 2 is a matrix. Most of the notation used is borrowed from data mining and especially from data warehousing literature, therefore, readers familiar to such concepts could comprehend more effortlessly since the definitions used follow many of the data cubes notions. Treating tensors as multidimensional arrays is a convenient way of processing high-dimensional data, since we are also interested in extracting arrays of lower dimensionality. A vector can be extracted along any mode, forming a fiber. Furthermore, when a single plane from an order-r tensor on any mode is extracted, a matrix known as a slice is produced. Slices and fibers of a third order tensor are illustrated in Fig. 1 and Fig. 2, respectively.

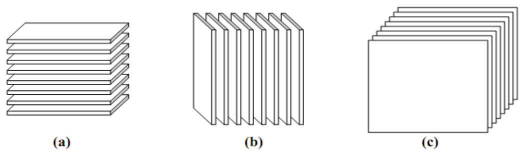

Fig.1. Slices of the (a) first (horizontal), (b) second (latteral) and (c) third (frontal) modes.

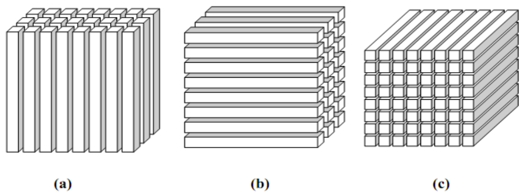

Fig.2. Fibers of the (a) first (column fibers), (b) second (row fibers) and (c) third (tube fibers) modes.

3.2 Proposed algorithm

For reasons of computational complexity, we have emphasized throughout the whole design and experimental phase to 3rd order tensors, since considering a tensor that contained at once all the 19 game input parameters described in Table 2 is unrealistic.

The choice of which three parameters to choose each time was based solely on the criterion of the lowest error rate in terms of classification accuracy. Figure 3 depicts tensor decomposition example for three of the most significant input features, as it turned out from the experimental section, namely *Lasers*, *BlastWaves* and *EnemiesKilledbyLasers*. Features that contained numeric values were discretized into 20 bins, so that each game that was encoded as a point in a $\Re^{20\times 20\times 20}$ space. The choice of the discretization intervals was done according to the unsupervised method of *Tree-based density estimation* [12]. Furthermore, each game was manually annotated, through external observation during game playing, as belonging to the *tactical* or the *action* user attitude (the class), so it is straightforward to believe that games of the training set constitute 2 well separated clusters, otherwise any other classification algorithm would perform poorly. The most representative vectors of each cluster span a subspace in $\Re^{20\times 20\times 20}$. The idea behind the proposed algorithm is to construct a set of orthogonal vectors that span the dominant elements of each cluster. An unlabeled, unseen game can be classified by estimating the approximation error for each of the 2 clusters. Basis vectors are approximated using HOSVD.

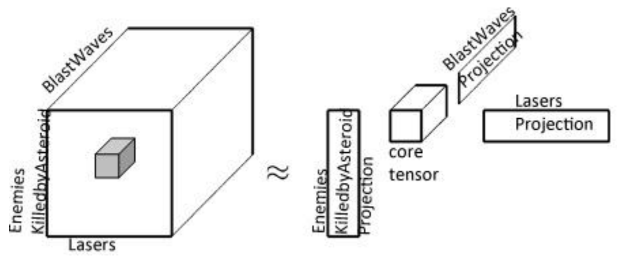

Fig.3. Illustration of a third order HOSVD decomposition considering 3 of the 19 game features. The core tensor contains concept strengths, while the projection matrices represent mode-to-concept similarities.

Classification Algorithm

The following algorithm describes the training phase:

```
for each class
    for d=1 to 3
        Temp=$A^{n1xn2xn3}$ $(A^{n1xn2xn3})^T$ /*the covariance matrix of mode-d matricization*/
        Set $U^{n1xn2xn3}$ to be the r leading left eigenvectors of Temp
    end for
    $C = A \times_1 U_1^T \times_2 U_2^T \times_3 U_3^T$ // calculate the core tensor
end for
```

Upon completion of the training phase, suppose a new game *G* is encountered. The testing phase deals with identifying which set of orthogonal projections per class describes it best. This is actually a minimization problem:

$$\min \left\| G - \sum_{i=1}^{l} x_i^c A(:,:,i) \times_1 U_1 \times_2 U_2 \right\|, \text{ where:}$$

- c denotes the class.
- *l* denotes the projected training examples per class and
- x_i^c are unknown scalars to be determined.

Due to the fact that matrices are all orthogonal, the solution is given by:

$M(c) = 1 - \sum_{i=1}^{l} \langle G, A(:,:,i) \times_1 U_1 \times_2 U_2 \rangle^2$. The game *G* is credited to the class for which M(c) is the lowest.

Classification by compression using HOSVD

The methodology of HOSVD can be additionally applied to compress the training set prior to the calculation of the basis matrices for the different classes. In this section, some mathematical formulations are provided, which are in a close analogy with low rank matrix approximations. The advantage is based on the fact that games from different users (action or tactical) are all projected onto one common subspace. Hence, an unknown game is only projected once, which is a substantial decrease in term of computational cost both in terms of computations and memory allocation. Furthermore, compression can also be applied to the class dimension, reducing noisy instances and creating a more compact database if one collects a plethora of gameplay data.In the training phase, a tensor D is built by using all games from the training set. They are organized in such a way that every slice contains games from the same class. The HOSVD approach decomposes D from $\Re^{20\times20\times20} = \Re^{4000}$ to $\Re^{r}$, *(r<<4000)* and the amount of game instances in each separate class *c*. Let D have the HOSVD $D \approx F \times_1 U_r \times_2 U_c$ where $U_r = U(:,1:r)$ $U_c = U(:,1:c)$ and $F = U(1:r,1:c,:) \times_3 U_3$. Under the assumption that both *r* and *c* are significantly smaller than the original values, high compression rates could be achieved without compromising the classification robustness. In the next section, we tabulate experiments that point towards this belief.

4 Experimental Results

The evaluation approach dealt with identifying which triplet of all attributes described in Table III best describes user models according to their behavioral patterns. The test set consisted of 840 games played by 210 different players, 150 male and 60 female. Each game was manually annotated by the authors according to whether they belonged to the tactical or the action class. In this section, we present the classification outcome for the four best attribute triplets in relation to classification error rate, as well as results from the compression phase, concluding with some performance metrics in terms of computational cost. Up to 16 basis matrices for each class were used in the classification and 10 least square problems were solved. In each experiment, the same number of basis matrices was kept for all classes. Figures 4 to 5 depict the error rate of the two best 3-mode tensors from the available set of game input features.As can be observed the best result is provided by the tensor which models ***Lasers***, ***BlastWaves*** and ***EnemiesKilledbyAsteroid*** attributes. The error rate is approximately 4,5% which is reached at around 12 basis matrices. In Table 4, we provide classification results from the complete dataset by using the K-Nearest-Neighbor and Support Vector Machines algorithms, two well-known approaches that have been previously used in many data mining tasks with successful outcomes.Furthemore, their error rate is almost identical to that of 3-mode tensor modeling, a fact that supports our claim that even low-mode tensors can reveal useful patterns from sparse data. Another triplet of significant results is that of Lasers-Grabbers-EnemiesMissed. The error rate is slightly higher than that of the previous case and again it is achieved when the number of basis matrices increases from 11 to 13. The same principle applies to the other two attribute triplets.

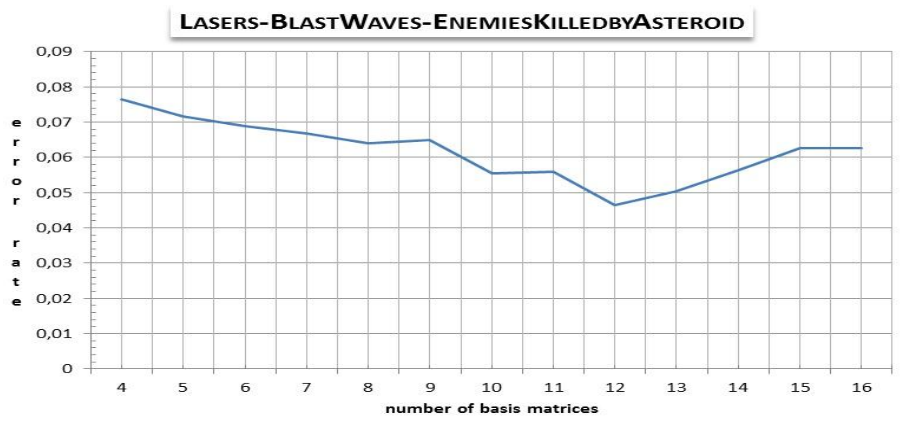

Fig.4. Classification outcome in terms of error rate for given 3-mode tensor.

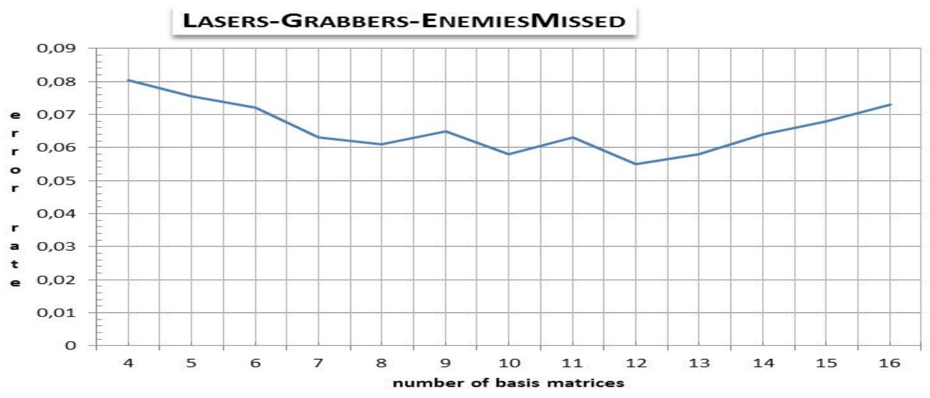

Fig.5. Classification outcome in terms of error rate for given 3-mode tensor**.**

Table 2. Classificationoutcomesfromthe full feature set using KNN and SVM classifiers.

	K-Nearest Neighbor	Support Vector Machines
Error Rate	4,25%	3,97%

From the above figures, we can conclude that a plethora of attributes, initially considered as important, such as the time required to kill the Boss (BossTime), the number of cargo (Carriers) and fighter (Fighter) ships the player has destroyed etc., do not seem to play an important role in deciding whether a player's style is tactical or action-oriented. Furthermore, the Laser attribute is very important since it is present in the top-3 of the 3-mode tensors as regards to error rate.

5 Conclusions

Modeling player behavior can be a means to understand her game-playing skills and preferences. This, in turn, can provide game developers with the tools to adjust the game challenges in order to better match the player capabilities, potentially resulting in higher engagement and user-satisfaction.In our experiments, using this robust 3-mode tensor, game modeling experts could extract useful patterns and verify the actual parameters that define the problem. Using this knowledge, one can straightforwardly classify (we remind readers that the classifier is very fast in the test phase) new players according to the predefined behavior easily, based on a tutorial or introductory playing session, considering only the reduced set of attributes, a considerably faster process.

6 References

1. M. Csikszentmihalyi, Flow: The Psychology of Optimal Experience. Harper & Row Publishers Inc., New York, NY, USA, 1990
2. Hunicke, R. 2005. The case for dynamic difficulty adjustment in games. In Proceedings of the 2005 ACM SIGCHI international Conference on Advances in Computer Entertainment Technology (Valencia, Spain, June 15 - 17, 2005). ACE '05, vol. 265. ACM, New York, NY, 429-433.
3. Ernest Adams, The Designer's Notebook: Difficulty Modes and Dynamic Difficulty Adjustment, Gamasutra, May 14,2008, http://www.gamasutra.com/view/feature/3660/the_designers_notebook.php?page=1
4. G.N. Yannakakis and M. Maragoudakis, "Player Modeling Impact on Player's Entertainment in Computer Games," User Modeling 2005, 2005, pp. 74-78
5. Kazemi, D. 2008. Metrics and Dynamic Difficulty in Ritual's SiN Episodes. OrbusGameWorks.com. http://orbusgameworks.com/blog/article/70/metricsanddynamic-difficulty-in-rituals-sin-episodes-part-1
6. Booth, M. 2009. The AI Systems of Left 4 Dead. Keynote, Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE '09). Stanford, CA. October 14 – 16, 2009.
7. Pedersen, C., Togelius, J., and Yannakakis, G. 2009. Modeling Payer Experience in Super Mario Bros. Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games (Politecnico di Milano, Milano, Italy, September 07-10, 2009).
8. M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin, "Polymorph: dynamic difficulty adjustment through level generation," Proceedings of the 2010 Workshop on Procedural Content Generation in Games, Monterey, California: ACM, 2010, pp. 1-4.
9. K. Anagnostou and M. Maragoudakis, "Data Mining for Player Modeling in Videogames," Informatics, Panhellenic Conference on, IEEE press, 2009.
10. L. De Lathauwer, B. De Moor, J. Vandewalle, A multilinear singular value decomposition, SIAM J. Matrix Anal. Appl. 21 (4) (2000), 1253–1278.
11. S. Wold, Pattern recognition by means of disjoint principal components models, Pattern Recognition 8 (1975) 127–139.
12. Gabi Schmidberger and Eibe Frank, Unsupervised Discretization Using Tree-Based Density Estimation ,Knowledge Discovery in Databases: PKDD 2005, Lecture Notes in Computer Science, 2005, Volume 3721/2005, 240-251.