2023

On-Line Environment Adaptation for User Performance Optimization

Subharag Sarkar

On-Line Environment Adaptation for User Performance Optimization

by

SUBHARAG SARKAR

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2023

To Thakuma, Maa, Baba, and Dada.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and appreciation to all who have contributed to the completion of my Ph.D. Thesis. The support, guidance, and encouragement have been invaluable throughout this journey.

First and foremost, I am deeply indebted to my supervisor, Dr. Manfred Huber, for his unwavering guidance, patience, and expertise. His insightful feedback and constructive criticism have played a crucial role in shaping my work and pushing me to explore new horizons. Every interaction with him has been a learning experience and I feel extremely lucky to have him as my mentor. Furthermore, I would also like to thank my thesis committee members, Dr. Vassilis Athitsos, Dr. Farhad Kamangar, Professor David Levine, and also Dr. Ramez Elmasri for their insightful and valuable guidance over the years.

I would like to express my gratitude to the Department of Computer Science and Engineering for providing me with a conducive learning environment and access to resources essential for conducting research. Their dedication to fostering academic excellence has been instrumental in my personal and professional growth.

I am grateful to my friends who supported me throughout this endeavor. Their words of encouragement, brainstorming sessions, and willingness to lend a helping hand have been invaluable in overcoming challenges and maintaining motivation.

A special mention goes to my family for their unwavering support, love, and understanding. Their belief in my abilities and constant encouragement have been the driving force behind my accomplishments. I am truly blessed to have them by my side.

Lastly, I would like to acknowledge Mr Satyabrata Ghosh and Mrs Jayita Biswas for their role in helping me reach to this position. The amount of time and effort they have put in to teach me the basics of science and maths in school instilled the confidence in me to pursue my PhD. I will always be indebted to them.

August 07, 2023

ABSTRACT

On-Line Environment Adaptation for User Performance Optimization

Subharag Sarkar, Ph.D.

The University of Texas at Arlington, 2023

Supervising Professor: Manfred Huber

In today's fast-paced and globally connected world, businesses are creating products with more significance to user personalization and customization. This has amplified the importance of capturing and learning user preferences as more information from users can lead to the designing and development of products that will improve user engagement and performance. Numerous algorithms based on collaborative filtering and recommender systems have been used to learn user preferences, but almost all of them require big datasets to train on. This creates a dependency on collecting more and more user information which might lead to ethical considerations and privacy concerns. To solve this dependency, we are proposing a novel architecture that aims to predict a user's preference with minimal interactions and that integrates it in an overall system to optimize user performance. The concept of collaborative filtering in an embedding space is used where based on the information from previous users, the architecture predicts the preference of the new user, with minimal feedback interactions. This is then integrated with a second learning agent that aims at optimizing task settings to optimize a user performance measure while considering the predicted user preferences.

In this dissertation, we have experimented with the structure of the architecture as well as the tools and algorithms that are used to create the architecture. We have then used parts of the architecture in two different domains to experiment and understand the positive impact and usefulness of the research. In the end, we used the complete architecture in a gaming domain and recorded its performance.

Applications in various domains have shown promising results where it has effectively learned on-line about user preferences with minimal interaction with the real user and shown the ability to optimize task contexts for the specific user without the need for a large number of negative user experiences during learning.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

Introduction

In today's world which has become interconnected and technologically advanced, **Personalization** and **Customization** has become an integral aspect of many industries. To make experiences more personalized and enhance user experience and performance, researchers have implemented many different methods to learn more about user habits and preferences. In the realms of e-commerce, entertainment, or even artificial intelligence, acquiring information about user preferences provides an opportunity to customize and personalize experiences for users.

Learning user preferences has allowed businesses to create their products and services to provide an experience to meet the individual's needs. With these collected data on the user's behavior and preferences, either through implicit observation or explicit feedback, the companies gain invaluable insights into customers' desires, allowing them to augment applications to more strongly engage customers, and increase their satisfaction. For example, the largest online video streaming service, Netflix released a large-scale dataset containing 100 million ratings from half a million users. They announced an open competition for the best collaborative filtering algorithm. Matrix Factorization (Koren, Bell, and Volinsky 2009) based on linear algebra and statistical analysis emerged as a state-of-the-art technique. This helped Netflix learn more about its users and provided much better recommendations to them.

When it comes to Learning user preferences, huge amounts of data are collected from user engagement and then used with various collaborative filtering algorithms and recommender systems (Goldberg et al. 1992), (Resnick et al. 1994), (Xiang Wang et al. 2019), (Cui et al. 2020), to learn about the users. All the methods presented above, however, work only when big datasets are available. To reduce the reliance on big datasets, it is imperative to design a system that not only learns a user's preference, thus creating a user behavior model, but also to do that with minimal interaction. This reduces the burden of collecting huge amounts of information for a user.

Learning user preferences has been a crucial technique to implement more personalization into our daily life, but there is another way, personalization and customization among products can be achieved. With the information on user behavior, a unique opportunity arises where companies can develop products with the capability of adaptation. Products that can understand the user's need and adapt themselves to provide exactly what is required.

This simple idea has led us through this research. The Architecture which will be introduced later, has a prime objective of not only understanding the user preferences but also using that knowledge to adapt and improve the domain, to increase the user's productivity. To improve the efficiency of the product we need constant interaction with the user, as the more the user is involved, the more information is collected that will help train the product to adapt itself more to the user's preference. This constant involvement, however, can be arduous and tiresome and in the real world no user will provide a copious amount of information before losing interest completely. To solve this problem it is highly recommended to implement a user behavior model which generates user performance to a certain degree of certainty. The Architecture developed implements one such behavior model which is trained with minimal interaction with the real user. The model is shown to generate information with good accuracy. Now with the help of this user behavior model, we can train the product so that it learns how to adapt itself for that particular user.

The architecture proposed in the dissertation consists of two interacting learning agents and can be divided broadly into two parts:

1.1 Outer Agent

The Outer Agent is the most important part of the architecture. It directly influences the product to create the adaptations necessary to produce a more personalized experience. We have chosen a Reinforcement Learning (RL) agent as the Outer Agent for the Architecture since generally no data regarding the optimal product for the specific new user is available. Reinforcement Learning has an innate characteristic of learning from experiences. It can be very versatile in learning about the user, to find out the likes and dislikes and the changes that can lead to an improved experience for the user. In the Architecture, the Outer Agent works with both the real user and also the second learning agent, the Inner Agent. It generates a certain state in the product and sends it to the real user or the Inner Agent for feedback. With the feedback it receives, it trains itself to understand the nuances of personalization. The aim has been to make the Outer Agent interact more with the Inner Agent rather than the real user. With the Inner agent, the Outer agent can learn really fast and provide much better results to the real user with minimal learning interaction. Figure 1.1 shows how the Outer agent interacts with the real user and Inner Agent and uses the feedback to train itself.

1.2 Inner Agent

The Inner Agent serves a very important task for the Architecture to work. It generates a user behavior model which, with certain accuracy, generates reliable information. To make a more accurate prediction we have used the concepts of Col-

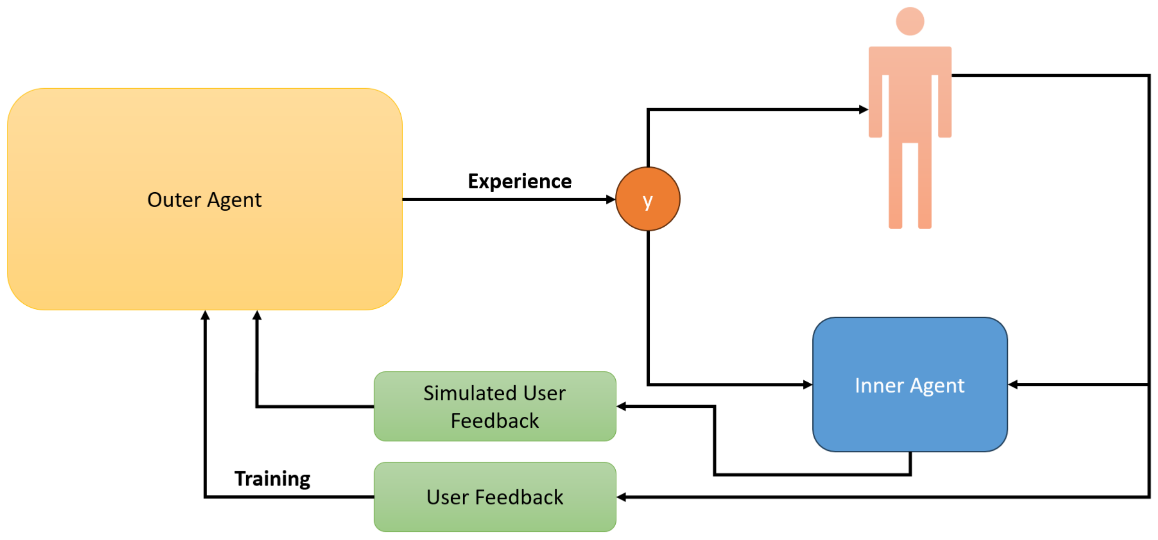


Figure 1.1: Outer Agent

laborative Filtering. A user dataset with information from previous users' experiences and interactions is used to map an embedding space to capture characteristics of user preferences in the target domain. We then use Gaussian Mixture Models to create soft clusters out of this information. When some new user information is available we use the information to find the clusters which are closely associated with new user data and use them to generate predictions for the Inner Agent. To generate the predictions, a Conditional GAN has been trained on the available user dataset. The Inner Agent structure has been successful to generate good user behavior models in all the Experimentations. Figure 1.2 shows how the Inner Agent is provided with Domain information to generate user feedback.

In this dissertation, we evaluate this Architecture in three different domains to see how well it performs.

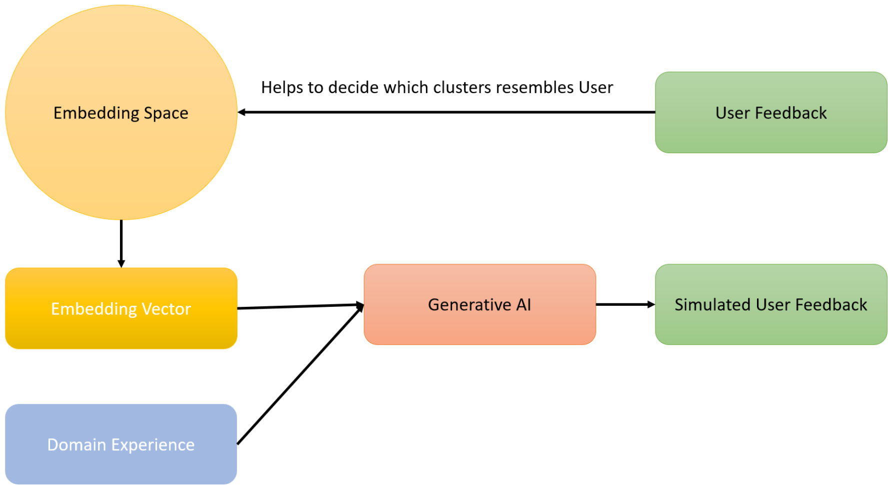


Figure 1.2: Inner Agent

The remainder of this dissertation is structured as follows:

Chapter 2 gives a detailed description of all the background algorithms that have been used to create this architecture.

In Chapter 3 we elaborate on the implementation of the architecture in an E-Learning domain. We create an E-Learning platform based on the Felder Silverman Learning Style Model (FSLSM) and Differentiated Pedagogy (Richard M Felder n.d.; Zaoui Seghroucheni, Mohammed AL Achhab, and El Mohajir 2014) where the Outer Agent takes control of the platform and trains itself to understand the needs of the student and provide them with coursework that will improve their performance. For the Inner Agent, we used a Conditional GAN to generate student feedback after completing coursework. As a proof of concept, the architecture fared well in the E-Learning domain.

In Chapter 4, we implement a part of the Architecture in an image domain. We introduce the concepts and use of Siamese Network (Bromley et al. 1993), Collaborative Filtering, and Gaussian Mixture Models with Conditional GAN (Mirza and Osindero 2014) to create a much more powerful metric to train an Inner Agent that makes accurate predictions. When it is used to predict user likeness, it provided the correct prediction 82% of the time in a domain with significant preference overlaps.

In Chapter 5, we implement the complete Architecture in a gaming domain. We create the Inner Agent which has developed in Chapter 4. The Outer Agent creates the game environment for the user and trains itself to learn how to manipulate the parameters to make the game more engaging for the player. We also introduce and use Facial Emotion Recognition (De Silva, Miyasato, and Nakatsu 1997) to get information about the user while playing the game. This provides extra information for the Outer Agent to be trained on. After experimentation, we get favorable results.

Chapter 6 concludes the dissertation by providing a synopsis of all the work done and the results of the Experimentation.

CHAPTER 2

Learning Model Background

2.1 Introduction

Before we delve into the aspects of the algorithm, this chapter will provide information on all the underlying Machine Learning tools and techniques which have been used to create the algorithm and also for the different experimentation and validations.

2.2 Reinforcement Learning

Reinforcement Learning (RL) (Richard S Sutton 2018), is a dynamic and rapidly evolving branch of Machine Learning where intelligent agents can learn to make decisions and take actions in an environment to maximize a specific notion of cumulative reward. Reinforcement Learning has gained significant attention and achieved remarkable successes in a wide range of domains, including robotics, game playing, autonomous systems, recommendation systems, and more.

Unlike traditional Machine Learning paradigms such as Supervised Learning or Unsupervised Learning, where agents learn from labeled or unlabeled data, Reinforcement Learning emphasizes learning through interaction with the environment. It draws inspiration from the way humans and animals learn through trial and error, exploring their surroundings and adapting their behavior based on feedback and consequences.

The framework of Markov Decision Processes (Puterman 1990) is at the core of Reinforcement Learning. MDPs model sequential decision-making problems where

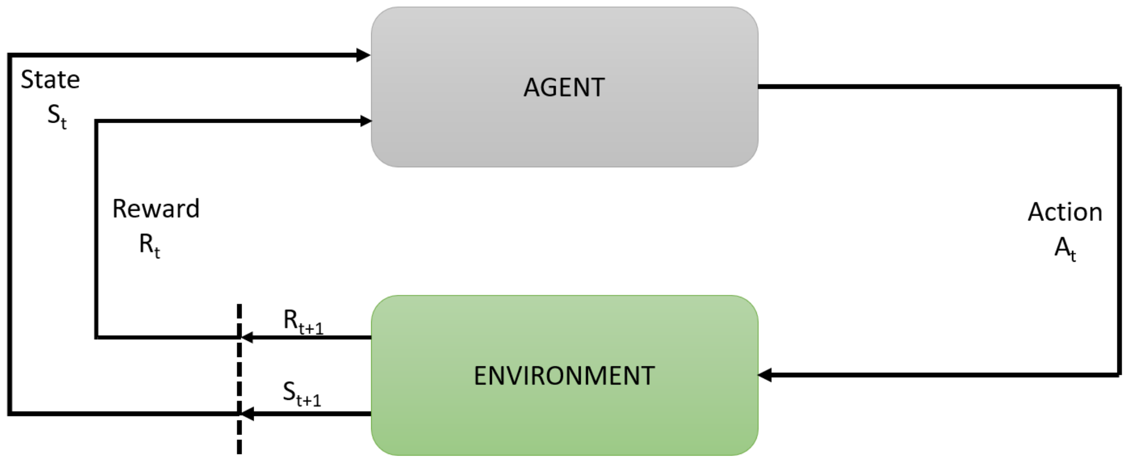


Figure 2.1: General Architecture of Reinforcement Learning

the agent interacts with the environment. An MDP consists of a set of **states**, **actions**, **transition probabilities**, **rewards**, and a **discount factor**. Through the interactions, the agent learns a policy, which is actually a mapping from states to actions, with the aim of maximizing the long-term cumulative reward.

Over the years a number of different methods of Reinforcement Learning algorithms have been introduced.

2.2.1 Value-Based Methods

Value-based methods aim to estimate the value of each state or state-action pair. These methods are based on the concept of a value function, which predicts the expected return an agent will receive from a given state or state-action pair. Notable algorithms within this category are:

2.2.1.1 Q-Learning

Q-Learning (Watkins and Dayan 1992) is an off-policy algorithm that learns the optimal action-value function, also known as the Q-function. It iteratively updates the Q-values based on the agent's experience to converge toward the optimal policy. Q-Learning is known for its ability to handle large state-action spaces and its applicability in environments with unknown dynamics.

2.2.1.2 Deep Q-Networks (DQN)

DQN (Mnih et al. 2015) combines Q-Learning with deep neural networks to approximate the Q-function. It employs a replay buffer and a separate target network to stabilize learning. DQN has been successful in solving complex RL problems, including playing Atari games at a human-level performance.

2.2.2 Policy-Based Methods

Policy-based methods optimize the policy directly while bypassing the estimation of value functions. These algorithms aim to find the policy that maximizes the expected cumulative reward. Notable algorithms within this category include:

2.2.2.1 Policy Gradient Methods

Policy gradient methods (Sutton et al. 1999) optimize the policy parameters by directly estimating the gradient of the expected cumulative reward. By using techniques such as the REINFORCE algorithm (Williams 1992), policy gradient methods can learn both stochastic and deterministic policies.

2.2.2.2 Trust Region Policy Optimization (TRPO)

TRPO (Schulman, Levine, et al. 2015) is a policy optimization algorithm that seeks to improve stability and monotonic improvement. It maintains a trust region constraint on the policy update to ensure that the policy changes are not too large.

TRPO has been successful in learning complex policies with high-dimensional action spaces.

2.2.3 Actor-Critic Methods

Actor-Critic methods (Sutton 1988) combine the benefits of both value-based and policy-based approaches. These algorithms maintain an actor, which learns the policy, and a critic, which estimates the value function. The actor uses the critic's evaluations to update the policy parameters. Notable actor-critic algorithms include:

2.2.3.1 Advantage Actor-Critic (A2C)

A2C (Y. Wu et al. 2017) is a synchronous and online update variant of the actor-critic algorithm. It uses the advantage function to estimate the quality of actions taken relative to the average value function. A2C has been widely used due to its simplicity and parallelizability.

2.2.3.2 Proximal Policy Optimization (PPO)

PPO (Schulman, Wolski, et al. 2017) is an actor-critic algorithm that combines ideas from both policy gradient methods and trusts region methods. PPO uses clipped surrogate objective functions to limit policy changes and ensures stable and efficient learning.

2.3 Siamese Networks

A Siamese Neural Network (Bromley et al. 1993) is an Artificial Neural Network architecture that uses two identical sub-networks to establish an embedding space that represents item similarity. The sub-networks process two separate input data points, typically referred to as the "anchor" and the "comparison" samples. Each input passes through its respective network and generates embedding features. These features are then compared to compute a similarity score or distance metric, indicating the similarity or dissimilarity between the input pairs. To do this, the same weights are used to compute output vectors for two different inputs which can be compared to each other. The networks learn and extract relevant features that are invariant to variations in the input domain. By enforcing this weight sharing, Siamese networks encourage the network to learn discriminative features that are useful for comparing and contrasting input pairs.

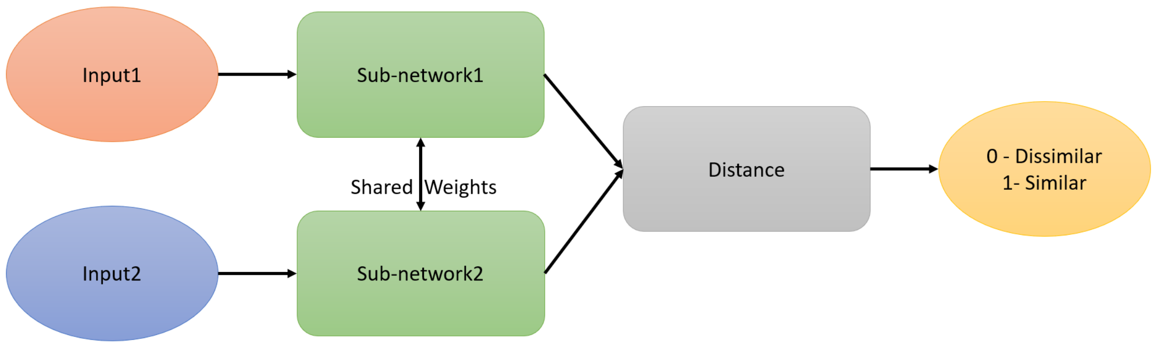


Figure 2.2: General Architecture of Siamese Network

Training the Siamese network involves optimizing the shared weights to minimize a loss function that captures the similarity or dissimilarity between the samples. The loss function aims to maximize the similarity score for similar pairs and minimize it for dissimilar pairs. Commonly used loss functions for Siamese networks include Contrastive loss, Triplet loss, and Binary cross-entropy loss.

2.3.1 Contrastive Loss

Contrastive loss encourages similar pairs to have a small distance or high similarity score, while dissimilar pairs have a large distance or low similarity score. It penalizes pairs that violate these criteria and pushes the network to learn better feature representations.

2.3.2 Triplet Loss

Triplet loss leverages a set of three samples: an anchor, a positive sample (similar to the anchor), and a negative sample (dissimilar to the anchor). It encourages the network to minimize the distance between the anchor and the positive sample while maximizing the distance between the anchor and the negative sample.

2.3.3 Binary Cross-Entropy Loss

Binary cross-entropy loss is commonly used when the Siamese network is trained as a binary classifier. It measures the dissimilarity between the predicted similarity score and the ground truth label (similar or dissimilar).

The Siamese Network was first introduced in (Bromley et al. 1993) where it was used to compare signatures. By learning distinctive signature embeddings, Siamese Network can authenticate signatures and detect forged ones, aiding in fraud prevention and document authentication.

After its introduction, the algorithm has been extensively used in face detection algorithms (H. Wu et al. 2017), with remarkable performances, particularly in scenarios with variations in pose, lighting, and occlusions. By learning discriminative face embeddings, the Siamese Network framework enables accurate and robust face verification and identification.

(Koch, Zemel, Salakhutdinov, et al. 2015) incorporated the concept of Siamese Network and one-shot learning to create a system that learns from a single example of each class. By learning a generalized feature representation space, the Siamese

Network can effectively recognize and match new instances based on their similarity to a few reference samples.

(Sheng and Huber 2019) used the concept of Siamese Network on time series data. The network is trained in a weakly supervised fashion using only information about the similarity between data items to cluster behaviors based on sensor data. Since the network learns to output embedding vectors for the input data that reflect the similarity in terms of Euclidean distance, it is utilized as a weakly supervised technique. It has yielded comparable performance to fully supervised methods with reduced labeling overhead.

2.4 Gaussian Mixture Models

Clustering is an Unsupervised Learning problem aimed at grouping unlabeled data. Several hard clustering algorithms are used most commonly. Their main disadvantage is that they associate each point with only one cluster which will not yield good accuracy with datasets containing points that cater to multiple clusters as in the case with overlapping user preferences. By contrast, a soft clustering technique like Gaussian Mixture Model assumes that the dataset contains multiple Gaussian distributions and that each data point can belong to each distribution with varied degrees of probability. This notion offers a flexible and effective approach to modeling complex data distributions. Each distribution contains the following parameters:

- A mean μ that defines the center of the cluster
- A covariance Σ that defines its width. In a multivariate scenario, it is equivalent to the dimensions of an ellipsoid.
- A mixing probability π which defines how big or small each Gaussian function is.

GMMs represent a probability distribution as a weighted sum of Gaussian components. Each component is characterized by its mean, covariance matrix, and mixing probability or weight. The probability density function (PDF) of a GMM is defined as follows:

P(x) $= \Sigma_i(\pi_i * N(x|\mu_i, \Sigma_i))$

where x represents the observed data, π_i denotes the mixing probability or weight of the i-th Gaussian component, $N(x|\mu_i, \Sigma_i)$ represents the probability density function of the i-th Gaussian component with mean μ_i and covariance matrix Σ_i.

2.4.1 GMM using Gibbs Sampling

Gibbs Sampling (S. Geman and D. Geman 1984) is a Markov Chain Monte Carlo (MCMC) technique that is used to estimate the parameters of GMMs. Gibbs Sampling (Rasmussen 1999) enables us to estimate the latent variables, i.e., the component assignments, by iterative sampling from their conditional distributions. This process allows the model to gradually refine the assignments and learn the underlying structure of the data distribution. By updating the component assignments and GMM parameters iteratively, Gibbs Sampling helps to achieve a better approximation of the true GMM parameters that characterize the observed data. The effectiveness of Gibbs Sampling for estimating GMM parameters depends on the complexity and size of the dataset which led to the introduction of the Expectation-Maximization algorithm.

2.4.2 GMM using Expectation-Maximization

Learning a GMM involves estimating the parameters, i.e., the means, covariance matrices, and mixing probability or weights of the Gaussian components, from the given data. The most commonly used algorithm for GMM learning is the Expectation-

Maximization (EM) algorithm (Xuan, Zhang, and Chai 2001), which iteratively performs two steps:

2.4.2.1 Expectation Step (E-step)

The algorithm calculates the posterior probabilities or responsibilities of each data point belonging to each Gaussian component. It uses the current estimates of the GMM parameters to compute these probabilities.

2.4.2.2 Maximization Step (M-step)

The algorithm updates the parameters of the GMM based on the responsibilities computed in the E-step. It maximizes the log-likelihood of the observed data with respect to the parameters, iteratively improving the GMM fit.

The EM algorithm continues these steps until convergence, achieving a locally optimal solution for the GMM parameters.

2.5 Generative Adversarial Network

Generative Adversarial Networks (GAN) (Goodfellow et al. 2014) have emerged as a groundbreaking framework in the field of machine learning, specifically in the domain of generative modeling. The framework introduces a paradigm where two neural networks engage in a competitive zero-sum game. Here the Generator network is indirectly trained by taking a random noise vector as input and transforming it into a synthetic sample that aligns with the data distribution of the training set. The Discriminator acts as a binary classifier, trying to predict whether the data provided is from the training dataset or from the Generator. The constant struggle of the Generator creating data and trying to pass them as original data and the

Discriminator finding out the real and false data helps the generator to create more realistic responses. Figure 2.3 illustrates the GAN

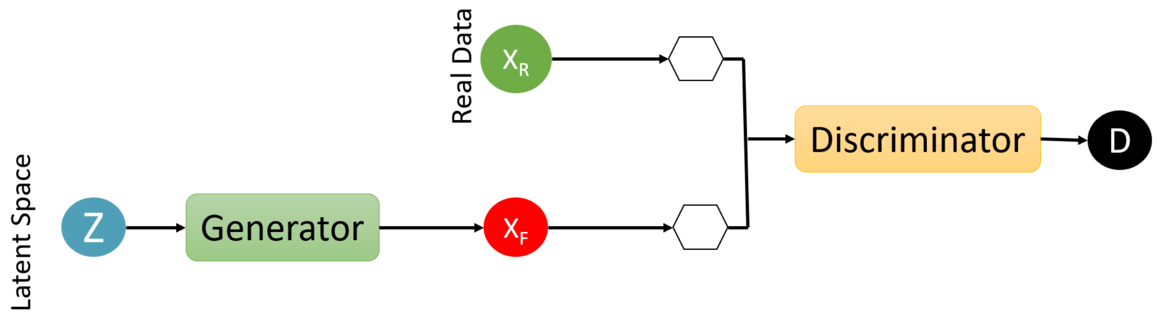


Figure 2.3: General Architecture of a Generative Adversarial Network

The basic GAN framework uses simple noise to generate meaningful data and is not able to differentiate between different classes of data. For example, when given an animal dataset to train, the GAN can learn to create images of different animals. But it is not possible for the framework to generate images of one type of animal. Since we need to generate data based on user type, we need to use a GAN framework that can generate data based on some label that characterizes the type.

2.6 Conditional Generative Adversarial Networks

Conditional GAN (Mirza and Osindero 2014) is a machine learning framework, that has emerged as an extension of Generative Adversarial Networks (GANs) allowing controlled generation of data samples. While traditional GANs generate data solely from random noise, CGANs use the core principle of GANs but add a condition vector to the Generator and Discriminator networks, enabling the targeted generation of class-based data based on specific inputs. The Conditional Input represents the labels for the actual data. This input, when fed to the Generator with the noise vector, trains to generate synthetic data for that label. The general architecture of a Conditional GAN is shown in Figure 2.4

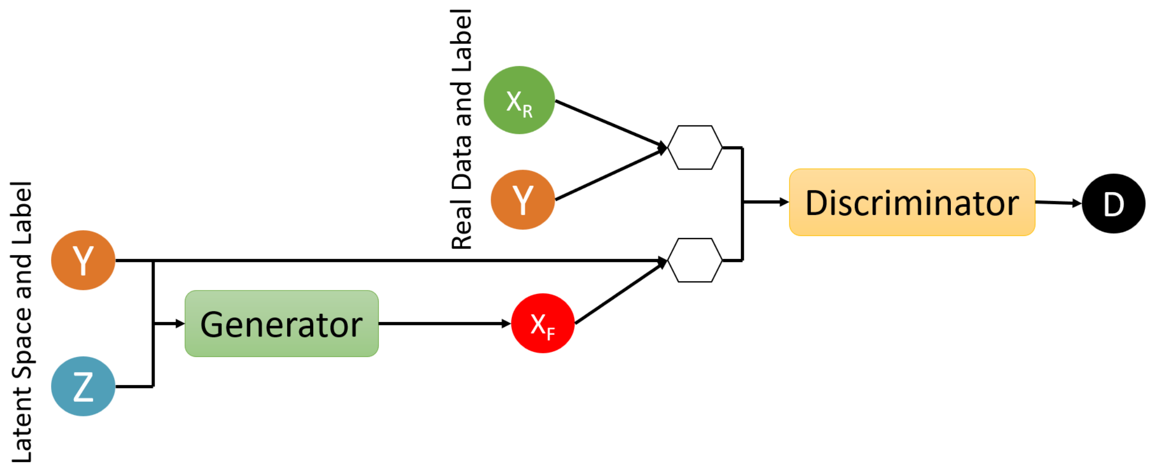


Figure 2.4: General Architecture of a Conditional GAN

The training process of CGAN involves optimizing the parameters of the Generator and Discriminator networks in an adversarial manner, with the inclusion of Conditional Input.

2.6.1 Generator Training

The Generator aims to generate samples that not only fool the Discriminator into classifying them as real but also align with the specified Conditional Input. The Generator receives feedback from the Discriminator and adjusts its parameters to improve the quality and alignment of the generated samples.

2.6.2 Discriminator Training

The Discriminator's objective is to classify real and generated samples accurately while considering the Conditional Input. It learns to distinguish between real samples and samples that deviate from the desired Conditional Input. The Discriminator's parameters are adjusted to improve its ability to discriminate between real and generated samples based on both their realism and class label.

The training process involves alternating updates between the Generator and Discriminator, with the Generator attempting to generate more realistic samples while considering the conditioning information, and the Discriminator aiming to improve its discrimination capabilities.

CHAPTER 3

Personalized Learning Path Generation in E-Learning Systems using Reinforcement Learning and Generative Adversarial Networks

3.1 Introduction

With the increasing prevalence of the Internet as a mode of communication among individuals, and with its mounting importance for education, there has been a growing focus on e-learning platforms and mechanisms. In addition, the acceleration due to the pandemic resulted in increased adoption of online classes by universities. This has increased interest and importance of E-learning and the challenge to increase its efficiency in terms of conveying content effectively to individual learners. While currently underutilized in many platforms, these systems promise to allow students to control aspects of their training according to their own pace and can thus be more adaptive to the individual student's specific learning profile. To facilitate such personalization, adaptive learning systems represent an important component of an e-learning platform. They provide endless possibilities in terms of personalized learning paths based on the needs, prerequisites, and more importantly learning style of the individual learner. Most of the existing e-learning models are interested in building a learner model based on the learner's characteristics. Based on these, the adaptive learning model provides specific learning objects to create a customized learning path that controls the type and order in which content is presented. The quality of the generated learning path here depends heavily on the correct identification of the important aspects of the learner's characteristics which need to be determined either through an expert or an automatic system from interactions of the learner with the e-learning system. In both cases, the number of learner interactions required will generally have a detrimental effect on the learner's experience as during interactions suboptimal learning paths will be presented.

In this research work, we are proposing a model which automatically learns to generate Learning Objects (LO) that are appropriate for a specific learner type based on Reinforcement Learning (RL)(Richard S Sutton 2018). To minimize required learner interactions during system training, the approach integrates a generative learner-type model that adapts to the specific learner and is used to train the RL system. The RL agent takes the Felder Silverman Learning Style Model (FSLSM) (Richard M Felder n.d.) and Differentiated Pedagogy (Zaoui Seghroucheni, M. AL Achhab, and El mohajir 2015) into consideration and aims to select the best version out of multiple versions of the same LO (Zaoui Seghroucheni, Mohammed AL Achhab, and El Mohajir 2014). To reduce learner interactions during training, we build a Conditional Generative Adversarial Network (CGAN) (Mirza and Osindero 2014) which is used to mimic the characteristics of a student to help train the RL agent to provide the best learning path for the learner. The next few sections are organized as follows: We discuss related work which has taken place in this field of research. Then we elaborate on Learning Models. We explain how the Learning Plan Personalization Framework works followed by the experimentations and Results and ending with the Conclusion.

3.2 Related Work

Several approaches for personalizing learning paths and adapting content to a learner's profile have been proposed before. Those works are summarized in two categories based on the way they assess information about the learner's preferences and learning style.

In the first category, the systems use implicit methods based on analysis (Carchiolo et al. 2007), (Nabila Bousbia 2010), and observation (Schiaffino, Garcia, and Amandi 2008), (K. Graf S. n.d.) to identify learning styles. This method is used so that the learner does not have to perform additional tasks such as filling out multiple forms. In addition, this approach is more intuitive for the learner as it does not require the learner to be aware of which content presentation and learning style works best for them. However, this method can also be unreliable because the learner might engage in other activities that might be misleading to the designer.

In the second category, the systems use explicit methods to identify the learning styles, such as e-questionnaires (Tzouveli, Mylonas, and Kollias 2008), (Bontchev and Vassilev 2012), (S. Graf et al. 2007), learners expressing their preference (Guerrero-Roldán and Alfonso n.d.), personal characteristics (Eyssautier et al. n.d.), or using FSLSM (Felder n.d.), (Papanikolaou et al. 2003). These approaches build on the learner's understanding of their strengths and weaknesses and might, particularly in the context of younger learners or learners with cognitive limitations. It also yields answers that do not represent what is most effective for the individual.

In both categories, the information gathered either implicitly or explicitly is then used to propose types of learning objects in the end.

The works presented above do not question the cognitive level of the learning object. The learning objects (LO) are assumed to be valid for all learners. To address this problem, the authors of (Zaoui Seghroucheni, Mohammed AL Achhab, and El Mohajir 2014) proposed a version model of each LO. The authors of (Honey and Mumford 1986) used this proposed model as it solves the non-leading path problem and offers the relevant versions for the learner. They used a Bayesian network with FSLSM (Felder n.d.) and Differentiated Pedagogy (Zaoui Seghroucheni, M. AL Achhab, and El mohajir 2015) to create multiple versions. Our approach takes a similar approach and improves upon the work in (Honey and Mumford 1986) by using a Reinforcement Learning agent to find the learning path. This removes the dependency on e-questionnaires and Bayesian networks and streamlines the learning path agent responsible to generate the correct learning path.

3.3 Learning Model

The system proposed here builds its personalization and learning paths based on the Felder Silverman Learning Style Model (FSLSM) and Differentiated Pedagogy by adapting Learning Objects (LO) and the course content.

3.3.1 Felder Silverman Learning Style Model

There are numerous learning style models presented in the literature. Models from Honey and Mumford (Honey and Mumford 1986), David A. Kolb (Kolb 2014), and Felder and Silverman (Felder 2002) have different classifications and descriptions of the types of learning. The Felder Silverman model provides better detail of the learner's learning style by perceiving the preferences on four learning dimensions, whereas the other models create group styles of learners. Therefore, this work is based on the Felder Silverman Learning Style Model (Richard M Felder n.d.). The Felder Silverman model is also the most appropriate for hypermedia courseware according to (Carver, Howard, and Lane 1999). (Brown 2006), (Popescu, Badica, and Trigano 2008). (Sangineto et al. 2008) also show this model's popularity and justify our choice for this work. Moreover, they show that FSLSM has been widely validated on student data. Although other models might have proper theoretical foundations, FSLSM contains realistic recommendations which personalize learner profiles for teaching (Popescu, Badica, and Trigano 2008).

Dimension	Types	Definition
A/R	Active Reflective	Work on it Thinking about it
S/I	Sensing Intuitive	Learning the Facts Learning the Concepts
V/V	Visual Verbal	Requires Pictures and Videos Requires Lectures and Readings
S/G	Sequential Global	Working Step by Step Finding the Big Picture

Table 3.1: Learning Style Dimensions in FSLSM

3.3.1.1 The Perception of Different Learning Styles

FSLSM differentiates learning styles along four dimensions that broadly characterize the way the learner addresses problems, the type of knowledge the learner acquires, the most effective way information is to be presented, and the way a topic is learned, as shown in the following table:

More specifically, the four dimensions of FSLSM (Centre for Teaching Excellence n.d.) which are characterized by the individual preferences of the learners divide learning styles into categories. Elaborating on the dimensions, they are:

Active/Reflective:

Active: These learners like to work actively with the information provided. They like to talk and try the subject to learn effectively.

Reflective: These learners like to think and grasp the information properly before they act on it.

Sensing/Intuitive:

Sensing: These learners like to work with absolute information. They like all the details and facts and like to work with data that is already proven. They like practical applications.

Intuitive: These learners understand information, which is more abstract, original, and oriented. They are big-picture people who understand general trends.

Visual/Verbal:

Visual: Visual presentations, charts, pictures, and diagrams are used for these learners.

Verbal: These types of learners prefer verbal explanations, both written and spoken.

Sequential/Global:

Sequential: These types of learners prefer linear organized information which is logically sequenced and works in a systematic way.

Global: These learners organize information holistically and randomly without connections. They are scattered and disorganized in their thinking but creative.

3.3.2 Differentiated Pedagogy

As indicated in the FSLSM, students learn in different ways and when creating adaptive learning content, the goal is thus to optimize all learning situations. Therefore, it is imperative to use every educational resource available so that the learner is presented with the most rewarding teaching situations that help improve learning. Differentiated Pedagogy is aimed at enabling this through the personalization of course content and learning objects.

3.3.2.1 Aspects of Differentiation

Philippe Meiriu in his paper (Meiriu 1999) stated that it is essential that a learning space is defined for learning activities to be exercised. The three pillars of learning are the **Teacher**, the **Learner**, and the **Knowledge**. Meiriu states that some learning situations often fail since they attach importance to only the knowledge and teaching and not to the learner, for which the whole platform is built. Differentiated Pedagogy should always consider these three pillars equally because the interaction between them is what success is dependent on.

3.3.2.2 Differentiation of Content

The lesson's content should be differentiated based on the knowledge the learner already possesses. For the basic content, the standards followed by district and state should be covered. There might be learners who are completely unfamiliar with the lesson, while there will be some who have partial knowledge of the lesson, learners with a mistaken understanding of the content, or learners with complete mastery. This differentiation is required to meet the needs of the different learners and can be designed based on the different areas of Bloom's Taxonomy (Bloom et al. 1956). So, learners with no previous knowledge are required to complete tasks on the lower levels of Bloom's Taxonomy: **Knowledge**, **Comprehension**, and **Application**. Partial mastery learners can work on **Application**, **Analysis**, and **Evaluation**. Learners with complete mastery, finally, can finish the **Evaluation** and **Synthesis** areas.

3.3.3 Adaptation of Course

Every type of Learner has a different learning style. The course should be adapted based on the requirement. Figure 3.1 shows an example of the available elements in a course that can be customized to personalize a learning path. Active learners learn by trying out things and working actively, so the coursework for them should be designed in a way that includes a greater number of exercises with assessments presented at the beginning and end of the chapter. Since they are less interested in examples, fewer examples will be fine for them. Reflective learners on the other hand require less emphasis on exercise and self-assessments. They should be initially provided with the learning material and then with examples. Sensing learners learn from examples and concrete data. Hence examples should increase as well as exercises. Intuitive learners like challenges so exercises and self-assessments are more important for them. Since Sequential learners like linear steps, they should be first presented with learning material, then examples, assessments, and then exercises. Global learners, in contrast, need an overview of the coursework. They require many examples after the learning material.

3.3.3.1 The Structure of the Course

To address adaptability, there are generally multiple versions of each Learning Object. The course can thus be represented as a quadruple **Course** = **EXRC**, **EXMP**, **THCON**, **ASMT**:

EXRC = Set of Exercises

EXMP = Set of Examples

THCON = Set of Theoretical Content

ASMT = Set of Assessment

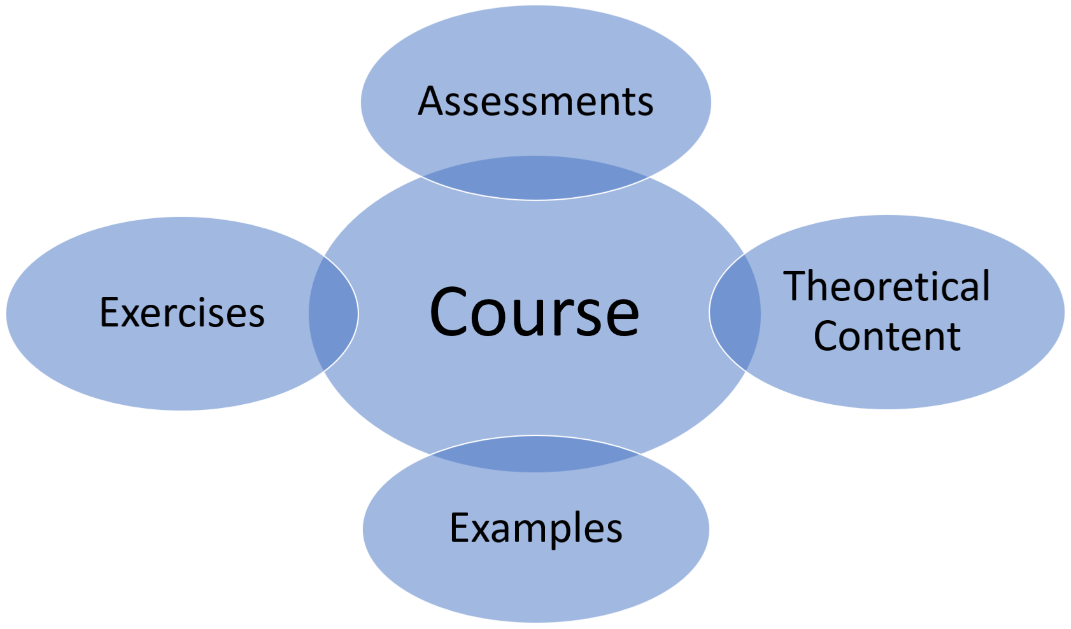


Figure 3.1: Key Elements of a Course

EXRC = $\sum_i^n ER(i)$ where ER=U_i^nV(i)

EXMP = $\sum_i^n EM(j)$ where EM=U_j^nV(j)

THCON = $\sum_i^n TC(k)$ where TC=U_k^nV(k)

ASMT = $\sum_i^n AS(l)$ where AS=U_l^nV(l)

Where V(i), V(j), V(k), and V(l) are the learning object versions available. Finally, the course representation is:

COURSE=$\sum_i^n$ (U_i^nV(i) + U_j^nV(j) + U_k^nV(k) +U_l^nV(l))

3.3.3.2 The Versioning of Learning Objects

According to (CSE n.d.), the reason behind using multiple versions is Differentiated Pedagogy which helps students of different ages, skills, and abilities to achieve academic success by following different routes. It also creates a flexible framework where diversified learning helps the student achieve their goal (Przesmycki 2003), (Wiley 2003).

3.3.4 Learning Objects

Learning Objects (LO) represent the elements used in lesson plans and thus form the entities that a learning path is defined over. There are different definitions of Learning Objects. Figure 3.2 shows a simple Learning Object. Some examples of these are:

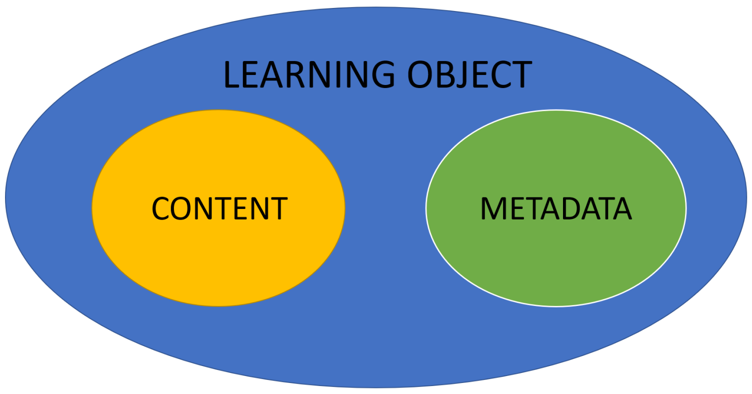


Figure 3.2: A Learning Object

- ”For this standard (Draft Standard for Learning Object Metadata v6.1), a Learning Object is defined as any entity, digital or non-digital, that may be used for learning, education or training” (IEEE Learning Technology Standards Committee 2001).
- ”...Learning Object... [is] ‘Any digital resource that can be reused to support learning.’ This definition includes anything that can be delivered across the network on demand, be it large or small. Examples of smaller reusable digital resources include digital images or photos, live data feeds (like stock tickers), live or prerecorded video or audio snippets, small bits of text, animations, and

smaller web-delivered applications, like a Java calculator. Examples of larger reusable digital resources include entire web pages that combine text, images, and other media or applications to deliver complete experiences, such as a complete instructional event" (Latorre et al. 2009).

- "Learning Objects is a new way of thinking about learning content. Traditionally, content comes in a several-hour chunk. Learning Objects are much smaller units of learning, typically ranging from 2 minutes to 15 minutes" (L'Allier 1997).
- "[A Learning Object] is defined as the smallest independent structural experience that contains an objective, a learning activity, and an assessment."(Mirza and Osindero 2014).

3.3.4.1 Metadata for LOs

Metadata is any type of information that refers to or describes aspects of another information. It is known as "data about data". Metadata is used in information management since it supports administrative operations which include searching, displaying summaries, and configuring interfaces. It creates a level of indirection that allows systems to manage resources without getting into the depth of the information. In the context of e-learning, information about the learning object, such as descriptions, subject classifications, and relations between learning objects can be stored in the metadata. So, when a learning object is searched in the repository, it is the metadata that is used for seamless searching. A learning object is a combination of the metadata and the content. The metadata can contain several basic elements, including the ones shown in Figure 3.3.

Taking the recommendation of Differentiated Pedagogy, and the adaptation of the course model chosen, the following versions are used here to emphasize the

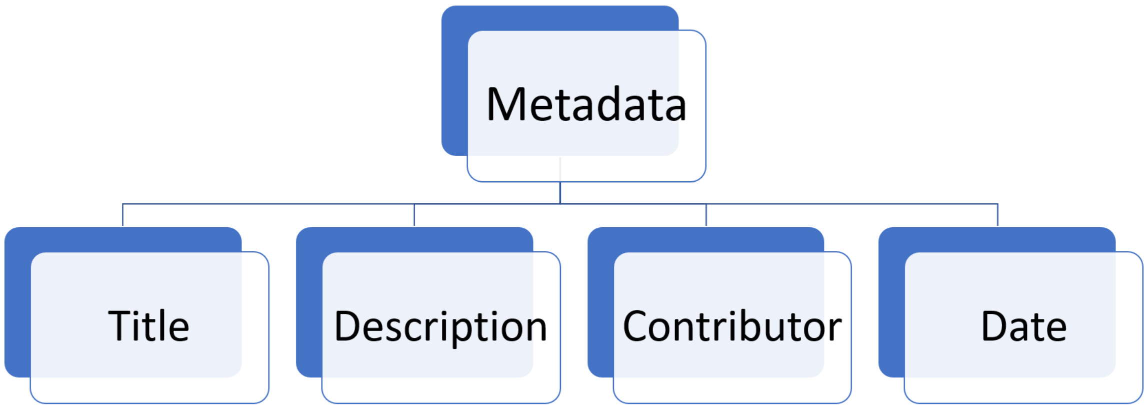


Figure 3.3: Metadata Elements

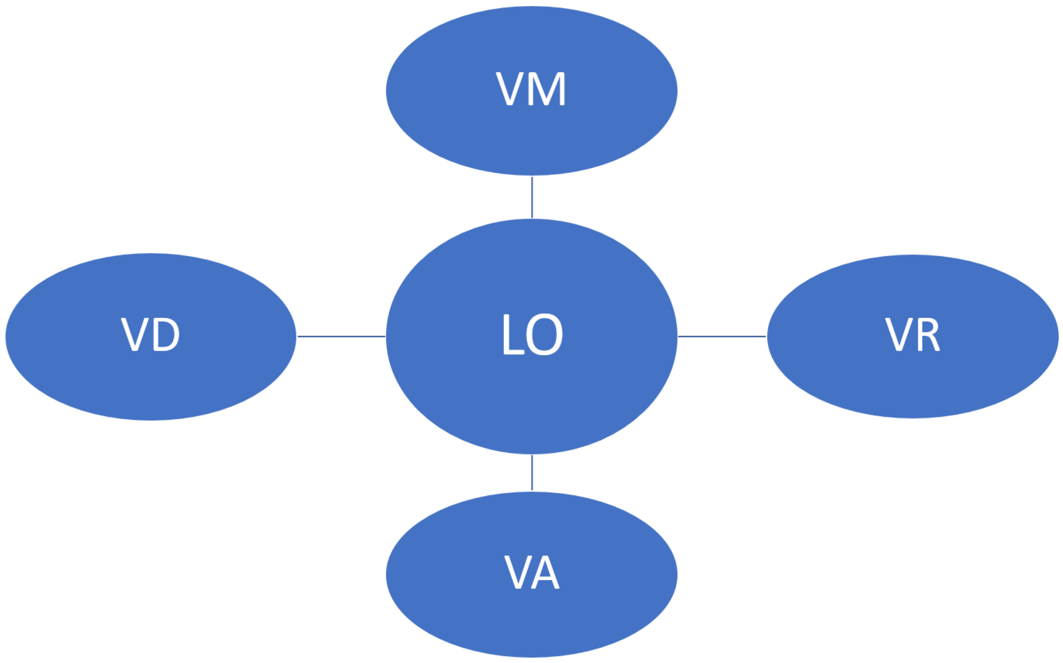


Figure 3.4: The Different Versions of the Same LO

Learning Object. In the example shown in Figure 3.4 (and used in the experiments in this paper), each LO has four versions:

VM: a multimedia version.

VR: a version with a reminder of the previous LO.

VD: a version with a deeper level of knowledge.

VA: a standard version.

To accommodate the new changes in the LO, the metadata is augmented to look like Figure 3.5:

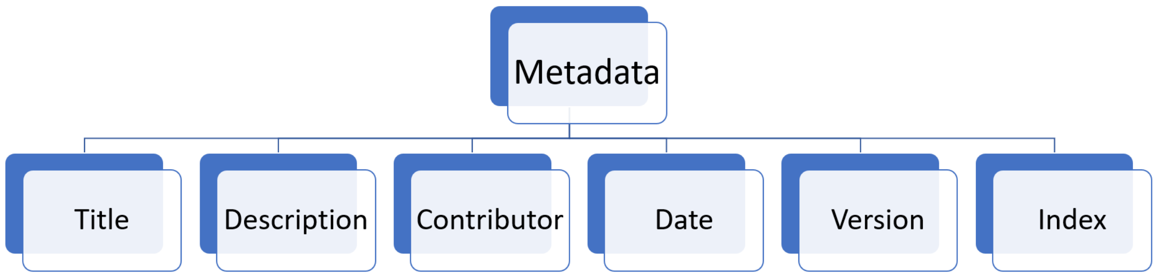


Figure 3.5: Metadata Elements after Adding new Versions

The new changes in the metadata will help to not overload the system as well as to eliminate the versions which are not used.

3.4 Learning Plan Personalization Framework

The goal of the approach presented in this research is to introduce a framework where we utilize the learning models and the adaptive learning objects to create personalized learning paths for each individual student. The architecture tries to create these learning paths true to the student's capability of maximal performance while limiting the sub-optimal learning paths during the training of the architecture. The framework can be divided into two branches:

- **The Inner Agent (Student Simulation Agent):** Machine learning approaches like Conditional GAN are used to create this student simulation agent that will help us to train the architecture with minimal original learner interaction.

- **The Outer Agent (Learning Path Agent):** Reinforcement Learning is used to create this agent that is trained to find the optimal learning path for each student.

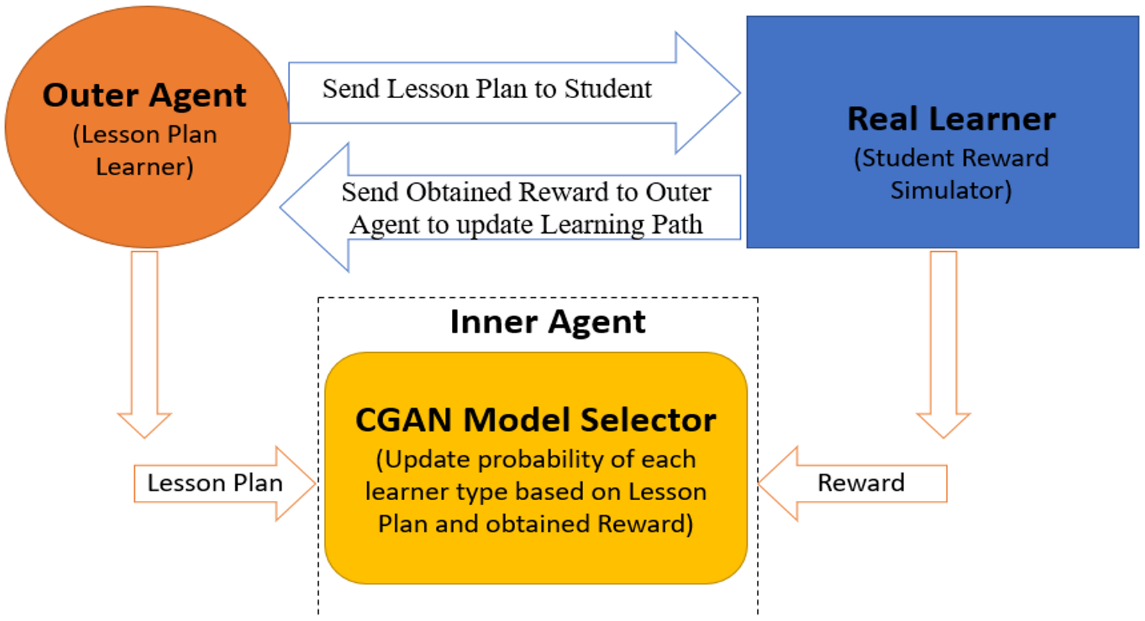

Outer Agent
(Lesson Plan Learner)
Send Lesson Plan to Student
Real Learner
(Student Reward Simulator)
Send Obtained Reward to Outer Agent to update Learning Path
Inner Agent
CGAN Model Selector
(Update probability of each learner type based on Lesson Plan and obtained Reward)
Lesson Plan
Reward

Figure 3.6: Outer Agent Interaction with Original Student

Figure 3.6 shows the overall architecture when interacting with the real student learner while Figure 3.7 shows it when interacting with the student simulator. Here the Outer Agent aims to learn the best learning path for the specific learner while the task of the Inner Agent is to learn to best anticipate and simulate the actions and rewards obtained by the specific learner. As we are taking an implicit learning style inference approach, this agent uses performance feedback from the learner's progress and employs Reinforcement Learning to determine the best learning objects and learning sequence. To reduce the need for frequent interactions by the learner, the framework employs the inner agent as a simulation of the learner's performance.

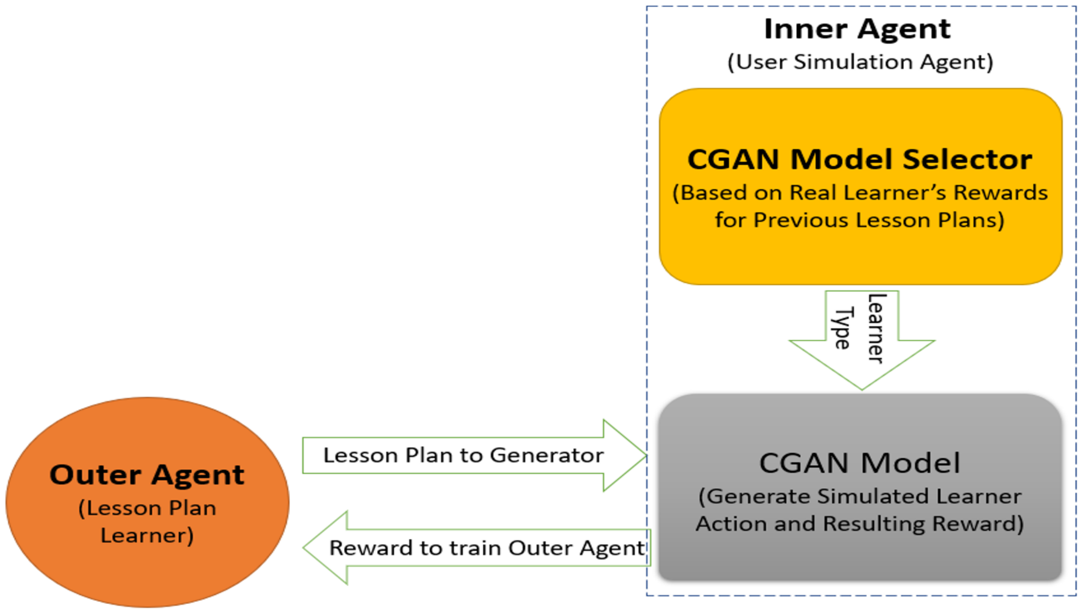

Inner Agent
(User Simulation Agent)
CGAN Model Selector
(Based on Real Learner's Rewards for Previous Lesson Plans)
Learner Type
CGAN Model
(Generate Simulated Learner Action and Resulting Reward)
Outer Agent
(Lesson Plan Learner)
Lesson Plan to Generator
Reward to train Outer Agent

Figure 3.7: Outer Agent Interaction with Inner Agent

For this, a Conditional Generative Adversarial Network (CGAN) is trained using previous experiences with other learners that provide a simulation for learner reactions. During the training of the outer lesson plan agent, the system uses sparse interactions with the actual learner to infer probabilistic learner-type attributes which condition the CGAN student simulation model towards the specific learner, allowing its output to be used to mimic student responses and thus to serve to train the outer agent without actual learner actions. The goal here is to minimize negative experiences by the learner due to inappropriate Learning Object sequences while maintaining the flexibility of the Outer Agent to derive the lesson plan that achieves the highest performance with the specific learner.

3.4.1 Machine Learning Approaches

The two interconnected learning agents in the proposed framework use two different learning approaches. The Outer Agent that learns the lesson plan uses Reinforcement Learning (RL) to optimize the performance of the learner by modifying the learning path. RL agent is used here to avoid the need for hand-coded learning types and corresponding, predetermined learning path choices for those types, thus extending the flexibility of the personalization. The Inner Agent which is aimed at adapting rapidly to provide a simulation of the learner's reactions to different learning objects utilizes a Conditional Generative Adversarial Network (CGAN) due to its ability to learn to generate realistic behavior even in non-deterministic situations where the same response can cause the different performance of the learner at different times.

3.4.1.1 Reinforcement Learning

In our architecture, the Learning Objects can be presented as actions in a learning environment Markov Decision Process where the agent must find the best possible learning path for the student. Using just the performance feedback from the learner, this approach can learn sequences of Learning Objects (i.e. a learning path) that maximize the overall performance of the learner (i.e. perceived reward) without the need for prior knowledge of learner type or promising learning path options.

3.4.1.2 Conditional Generative Adversarial Network

We have extensively used the CGAN architecture to train the Generator for each individual type of student using an initial set of student learners. Once presented with a new learner, the likelihood of each condition label can be inferred from a small number of learner interactions and the resulting performance results, allowing it to provide simulated life-like data that can be used to train the Outer Agent in the Architecture with limited need for actual student interaction.

3.4.2 Student Reward Simulation Model

To train this Architecture we need student participation to determine the best learning path for a specific student as no prior knowledge about the student is assumed to be available. To perform the initial evaluation in this research, a set of synthetic student models are used instead of real participants to reduce the amount of trial and error during development and to obtain more repeatable values instead of relying on interaction with real students for long durations. For this, we have developed a simple student simulator that is based on some assumptions that are informed by FSLSM and generates learner data with particular random distribution which is used instead of feedback from the real students. This not only addresses the repeatability of the experiments but also helps us show the efficiency of the architecture in the context of different learner types. For this, we have created an 8 x 8 table for 8 prototypical types of students from FSLSM and 8 corresponding learning styles, and we design reward distributions for each combination that represent higher performance for more suitable lesson plans and learning Objects. Rewards for each learner type - learning style combination are assumed here to be normally distributed with different means and variances for different combinations, with higher means representing better choices for the specific learner type. This provides a solid starting point to use the student simulator to generate a reward for each coursework provided by the outer agent. Then, that reward is used to train the agent. The reward distribution generated also considers several internal variables that represent real-life conditions that affect learning performance, such as motivation and concentration. In the model, we also consider how previous performance increases or decreases the current performance of the student. With these conditions, we attempt to mimic the real-life performance of a student which helps train our agent.

3.4.3 The Inner Agent

The Inner Agent is an integral part of the framework. The main objective of the Inner Agent is to provide near-accurate simulations of real student data which will help train the outer agent effectively. If the Inner Agent is unable to provide accurate simulations, the Outer Agent would struggle to generate optimal learning paths for the student. Once a new student starts using the system, the Inner Agent attempts to identify the learner-type parameters for the Generators and use it to generate appropriate performance and reward data for the specific, new student in lieu of having to present actual lesson plans to the student. So, when we use these Generators, the generated data should be comparable to the real student which helps us train the architecture. The Inner Agent can be divided into two subparts:

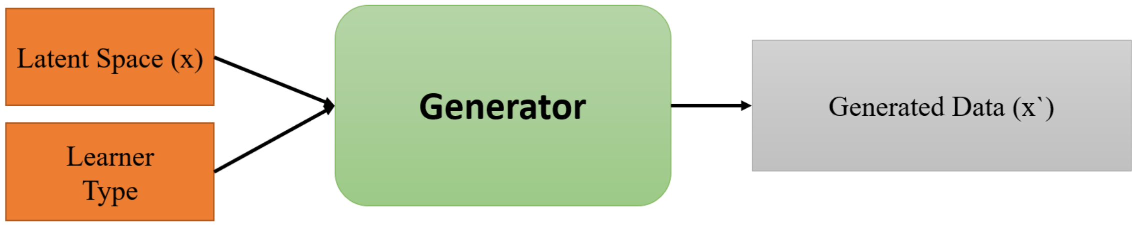


Figure 3.8: Generator Architecture of CGAN

3.4.3.1 Conditional GAN

The Inner Agent is aimed at providing a simulation of the specific student using only a small number of observations of the performance of the student. To

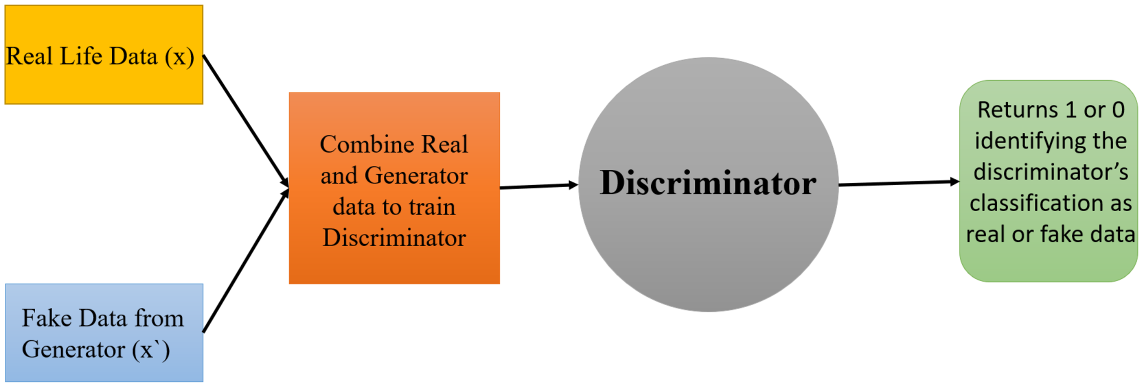

Real Life Data (x)
Fake Data from Generator (x`)
Combine Real and Generator data to train Discriminator
Discriminator
Returns 1 or 0 identifying the discriminator's classification as real or fake data

Figure 3.9: Discriminator Architecture of CGAN

learn realistic data Generators (or simulators), Generative Adversarial Networks have become one of the machine learning tools of choice due to their ability to learn to generate realistic "fake" data in probabilistic domains from an original dataset. We have seen GAN used in numerous image processing problems and the results it has provided in terms of generating realistic images or videos have shown the prowess of the approach. The motivation to use CGAN here is to create Generators for different learner types from the data of a set of individuals. The CGAN with an appropriate set of conditions (learner type) is then used as a stand-in for a specific real student to train the Outer Agent without needing exorbitant amounts of real learner interactions. The Generators here are trained on student data from individuals of each learner type. Figure 3.8 shows the concept of the Conditional Generator with its Conditional **learning style** Input that is ultimately inferred from the limited number of interactions with the specific user. Figure 3.9 shows the basic architecture of the Discriminator that is used to train the Generator in the CGAN architecture. The Generator is one of the pillars of the proposed learning path personalization architecture and it is imperative that the Generator provides valid data for the agent to train itself properly. To address the learning setting for our experiments, we have trained eight Generators, one for each of the student types using a set of synthetic individuals using the probabilistic behavior model described in the previous section. Each Generator has learned the trend of the reward generated by students of each type, hence when the Generator is used, the data generated is comparable to the student feedback. These 8 CGAN Generator models are part of the Inner Agent. This Inner Agent helps train the Outer Agent to derive an optimal learning path for the new student learner.

3.4.3.2 Generator Model Selection

When the framework is used, for the first coursework, the framework randomly chooses and presents it to the student (in our experiments represented by a student reward simulator). This provides performance feedback based on how well the learning path helped teach the student. This feedback is used to update the outer RL agent. In addition, a distribution of rewards is generated from all the Inner Agent Generators for that action, and the probability of the student type is updated based on the likelihood that the actual student feedback could have been generated by each of the Generators. This is done by finding the probability density function of each distribution and choosing the one with the highest softmax value. The Generator with the highest likelihood is the most likely to represent the student type of the student we are working with. We then use this model to train the RL agent for a considerable amount of time before it interacts with the student again. This loop continues until the RL agent learns the optimal learning path for the student. The sooner the correct Generator model is chosen, the faster the optimal learning path is achieved.

3.4.4 The Outer Agent

In the proposed framework, the aim of the Outer Agent is to learn the optimal Learning path and the learning objects which maximize the performance of each individual student. Reinforcement Learning can be a very powerful tool when it comes to learning behaviors. It does not require knowledge of the best behavioral strategy beforehand. Since it generates its own training data to train itself when other information in terms of the state and reward is provided, using the RL algorithm to generate the agent helps us create better personalization. The outer agent uses the RL algorithm to decide which learning path should be provided to the student. The training of the agent has been divided into two parts. The real student interaction and the training using the Generator models.

During the real student interaction phase, for the first iteration, we provide the learning path to the actual student and wait for the student's feedback i.e., the action it chooses, and the reward generated. When we get the feedback, we use this information to train the Outer Agent. We also use this information to update our probabilistic estimate of the student type used to condition the Generators in the Inner Agent. Thus, it helps us to better choose the most appropriate Generator. Figure 3.6 explains the interaction between the Outer Agent and the student.

The Generator interaction phase uses the Generator that best fits the data observed from the real student to train the RL algorithm in the Outer Agent to improve its accuracy. This limits actual student interaction requirements. Once the Outer Agent is trained with the Generator, it again interacts with the real student, providing additional performance data to update the Generator type selection and providing improved lesson plans to the learner. Figure 3.7 shows the interaction between the Inner and Outer agents.

3.4.5 Operation of The Architecture

In the proposed architecture, the Outer Agent (learning the learning path) and the Inner Agent (identifying learner style and generating simulated learner data) tightly interact during training. In the experiments performed, the Student Reward Simulator Model has been used in lieu of real students. This helps us achieve more repeatable results and helps us to test all student learner types. Firstly, the RL agent chooses a learning path (Action) and provides the option to the student as depicted in Figure 3.6. The student (in our experiments represented by a student reward simulator) follows the learning path and provides performance feedback based on how well the learning path helped teach the student. This feedback is used to update the outer RL agent. In addition, a distribution of rewards is generated from all the inner agent Generators for that action, and the probability of the student type is updated based on the likelihood that the actual student feedback could have been generated by each of the Generators. This is done by finding the probability density function of each distribution and choosing the one with the highest softmax value. The Generator with the highest likelihood is the most likely to represent the student type of the student we are working with. We then used this model to train the RL agent for a considerable amount of time before it interacted with the student again as depicted in Figure 3.7. This loop continues until the RL agent learns the optimal learning path for the student.

3.5 Results

We run the Architecture twice to investigate the operation of the Architecture and its ability to adapt to the learner without excessive need for real learner interactions while also limiting negative learner experiences. We run it once solely using real student interactions and once using the student data and the Generators. Figure 3.10 shows the number of real student interactions for the two scenarios for the 8 learner types with blue bars indicating the setting without use of the Generator and orange bars indicating the use of the Generator.

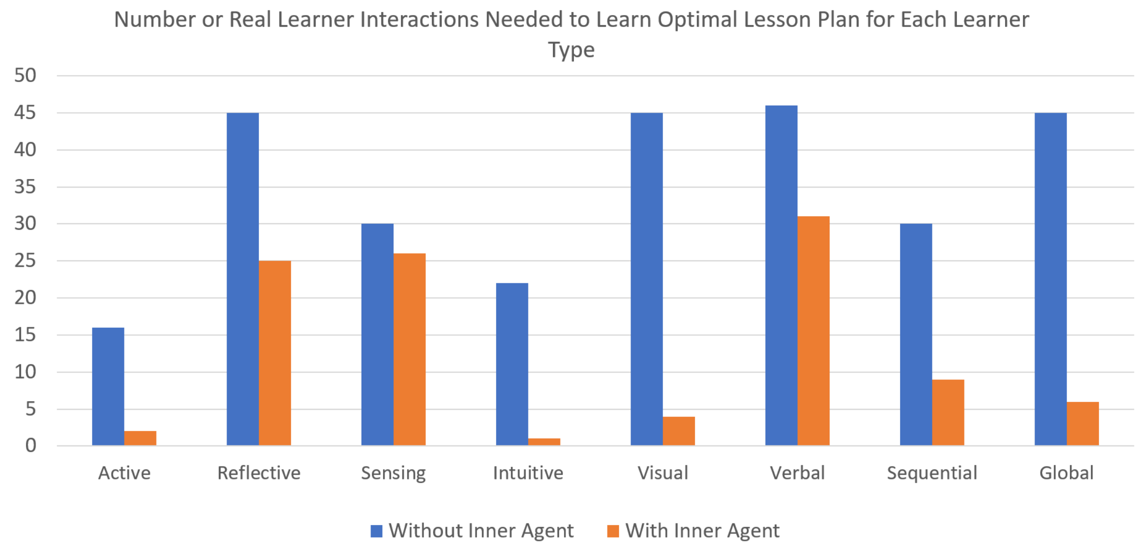


Figure 3.10: Performance of the Architecture with and Without Inner Agent

As we can see in these results When the CGAN-based Inner Agent is used effectively to estimate the learner characteristics, the real student interactions are significantly reduced for all learner types. When the CGAN-based Inner Agent is used, the Generator adds simulated learner reactions to train the Outer Agent. This, in turn, dramatically reduces the negative experiences of the learner with the system during training and helps us achieve the goal of minimal student interaction to train our agents.

3.6 Conclusion

In this research work, we have presented a framework for the adaptation of learning objects and learning paths using Reinforcement learning. The proposed approach is built on the concepts of FSLSM and Differentiated Pedagogy to create coursework that can be used in the adaptive model. The framework is aimed at minimizing the need for explicit student-type modeling while reducing the number of learner interactions needed to learn the best learning path. The model works by generating limited amounts of original data while a student interacts. This data is then used to condition a CGAN-based Generator which helps train the RL agent for a considerable time before it interacts with the student again to generate new original data. Experiments with different learner types showed that the use of the Generator significantly reduces the number of real student interactions needed before the optimal learning path is learned, thus illustrating the architecture's benefit. In the future, we intend to develop a Generator that can be used for all students and not only for a fixed number of types. This will help further generalize the adaptive model.

CHAPTER 4

Capturing Preferences of New Users in Generative Tasks with Minimal Interactions Collaborative Filtering Using Siamese Networks and Soft Clustering

4.1 Introduction

With the popularity of the Internet and the advent of smartphones, there has been a great increase in Internet traffic. There are approximately 5.16 billion Internet users, out of which approximately 4.76 billion are social media users (Petrosyan 2023). Since big social media companies are handling such a huge population, there are pressures to understand, capture and cater to user preferences with the underlying objective of captivating the user and generating revenues. Over the years several Recommender Systems with state-of-the-art collaborative filtering algorithms have been proposed. These algorithms use huge datasets and expensive computations to learn user preferences. However, they become difficult to use if there is very little information about the user. Prediction of user preferences is a challenge. But when the objective is to learn them without requiring the user to provide a profile or a significant number of interactions, it is much harder to achieve.

The novel architecture in this paper is trying to address this problem by aiming to predict a user's preference with minimal interactions. For this, it uses the concept of collaborative filtering in an embedding space where, based on the information from the previous user, the Architecture predicts the preference of the new user, with minimal feedback interactions. A robust Siamese Network (Bromley et al. 1993) is used which takes user data from previous episodes and generates an embedding space for the dataset. After the embedding space is created, a Gaussian Mixture Model generates soft clusters in this space that help predict the preferences of the new user. A Conditional GAN (Mirza and Osindero 2014) is then used, which utilizes the embedding space vector from the soft clusters as the Conditional Input to generate new data. This data is provided to the new user for feedback which the Architecture uses to adjust the clusters and thus the predicted preferences. To test the approach, an image generation domain has been used where the goal is to learn to generate images that match the user's preference based on minimal interactions.

In the next few sections, we will discuss related works in this field. Then we will introduce the underlying learning theories of the different machine-learning tools and the framework is introduced with a discussion on the details of the architecture. Next, we will elaborate on the Experimentation and Results and conclude

4.2 Related Work

Capturing user preferences is a difficult problem as acquiring information directly from the user can be prone to error. Indirect methods when used to reduce error, have been computationally expensive and time-consuming. Still, capturing user preferences accurately helps understand and respond to the needs of the user, which is why enterprises have collected large amounts of information on users. This information in collaboration with AI techniques has generated learning models that have gained a deeper understanding of user interaction. These Recommender Systems have automated the complex generation of recommendations based on the data analysis, but to achieve good accuracy they usually require huge amounts of information and computation.

In one of the first commercial recommendation systems, called Tapestry (Goldberg et al. 1992), the term "collaborative filtering" was introduced. The system recommended documents from newsgroups to users. The aim was to use social collaboration to understand the user requirements and thus save users from large volumes of preference questionnaires or document interactions. Collaborative filtering analyzes all user data to find good user-item pair matches. In contrast, the earlier methodology of content filtering was based on information retrieval. After the early success of collaborative filtering in the Group lens system (Resnick et al. 1994), the problem was mapped to classification, where dimensionality reduction was used to improve the solution.

When Netflix, an online video streaming service released a large-scale dataset containing 100 million ratings from half a million users and announced an open competition for the best collaborative filtering algorithm, Matrix Factorization (Koren, Bell, and Volinsky 2009) based on linear algebra and statistical analysis emerged as a state of the art technique.

Later, a new recommendation framework (Xiang Wang et al. 2019) was proposed which exploits user-item graphs to propagate an embedding. This has led to modeling expressive high-order connections in the graph. Collaborative filtering was also used in IoT scenarios (Cui et al. 2020), where time correlation coefficient and K-means clustering has been used to group similar users for an accurate recommendation.

The works presented above are all based on the notion that big datasets are available that help train the Recommender System. In contrast, our approach is based on the notion of accurately predicting user preference with minimal interaction. Information from previous users is taken into account to develop an embedding space, but the amount of data used is comparably lower. This lessens the dependency on big datasets to train and make accurate predictions.

4.3 Learning framework

4.3.1 Dataset:

To develop and initially evaluate the framework, the Fashion-Mnist dataset (Zalandor 2017) has been used for experimentation. This dataset, examples of which are shown in Figure 4.1, was chosen due to the following properties:

- Each image is 28x28 grayscale, which makes it easier to create an initial neural network for processing.
- The dataset contains images from 10 different fashion items where each item is visually different while sharing lower-level features, facilitating overlapping preferences.

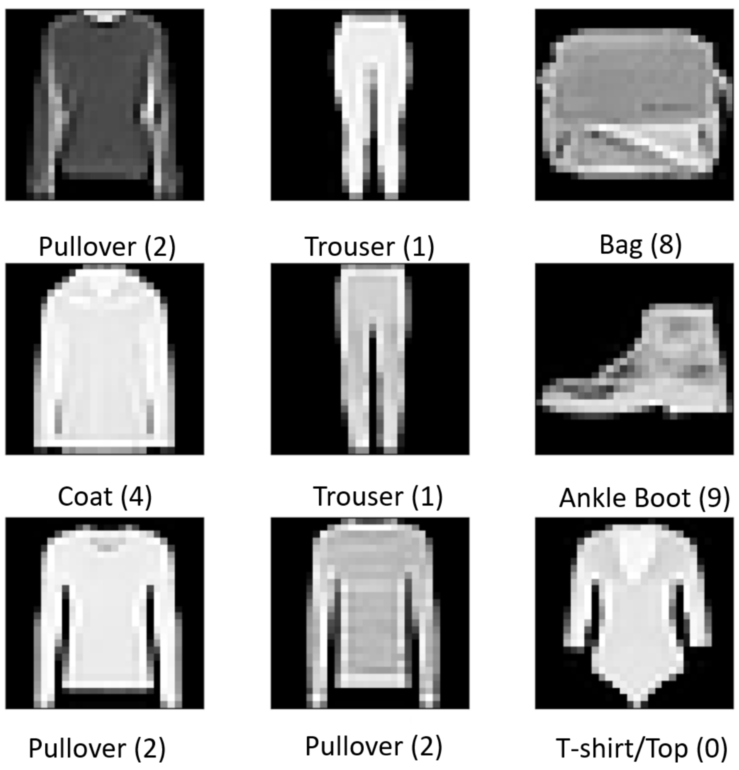


Figure 4.1: Examples from Fashion-Mnist Dataset

4.3.2 Siamese Network

The primary objective of the Siamese Network in this approach is to create an embedding space for the collaborative filter to work on. Figure 4.2 shows the internal structure of the Siamese Network. It includes the following three layers:

- **Input Layer -** Here image pairs of two types are sent as input. A Like-Like pair contains two images liked by the same user. They can be of similar or different classes. A Like-Dislike pair contains two images that can again be of the same class but have to be from two different users.
- **Embedding Layer -** In this layer, we use two similar Branch Networks which share the same weights for updating. The networks take each image from the Input Layer as input and then convert them into embedding vectors which are fed as input to the Decision Layer.
- **Decision Layer -** The first Layer which takes both outputs from the Embedding Layer is the Euclidean distance layer which computes the distance between the two embedding vectors. The images from the same user will generate embedding vectors close to each other whereas images from different users will generate embedding vectors far from each other. The concatenated output goes through a fully connected layer to predict either 1 (Same User) or 0 (Different User).

Based on the true and predicted answer for the pair, the network is trained. Binary Cross Entropy as the loss function and Adam as the optimizer are used for the experiment. Since they predict similarity and dissimilarity, the branch networks are trained with the same weights. Once the Siamese Network is trained, the embedding vectors of the images from previous users are used to generate a distribution in the latent embedding space. These embedding vectors are used to find clusters using

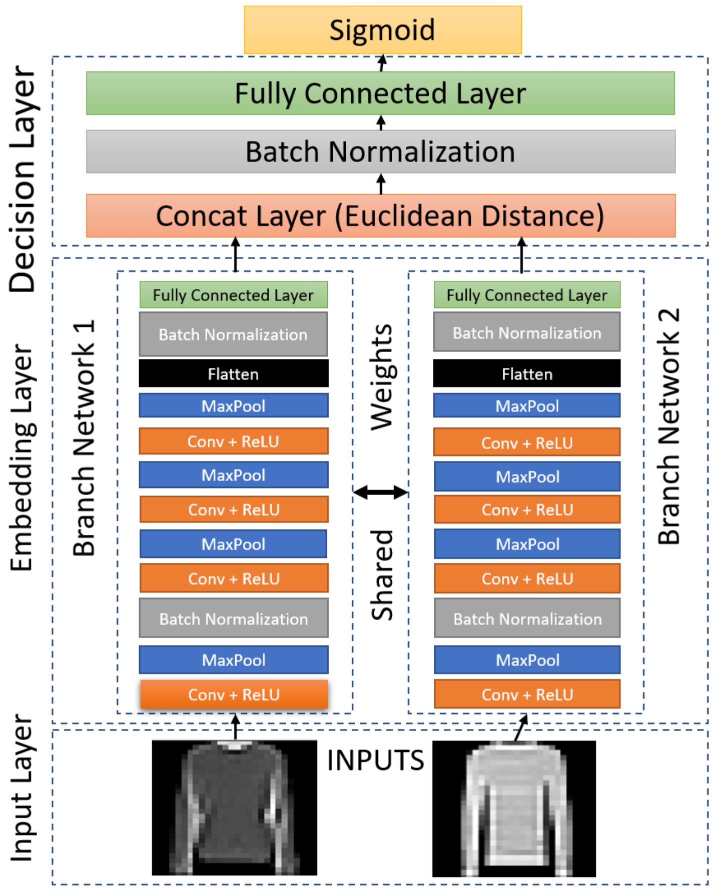
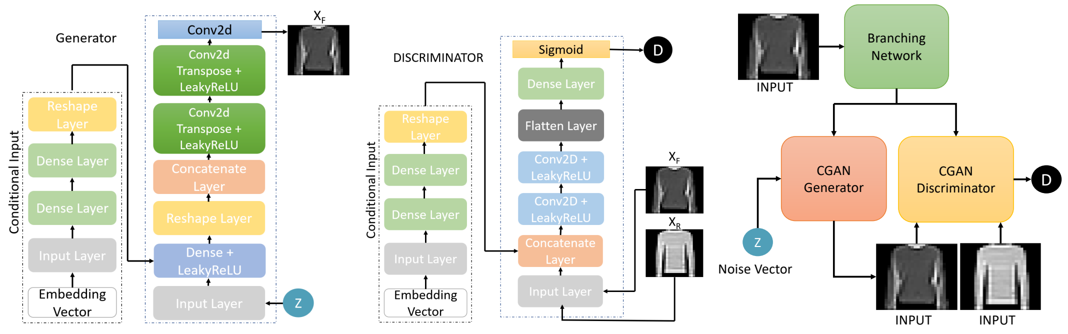


Figure 4.2: Architecture of Siamese Network

Gaussian Mixture Models and they also serve as a Conditional Input to a Conditional Generative Adversarial Network to generate sample images.

4.3.3 Conditional Generative Adversarial Network

The main motivation for using the Conditional GAN is to provide the Architecture the power to create its own data. This ability will aid the Architecture to learn user preferences faster. Here the CGAN is trained with user data collected during experimentation. For the Conditional Input of the CGAN, the **Branching Network** of the Siamese network is used to generate embedding vectors for each image. The architecture is designed as follows:



(a) Generator. (b) Discriminator. (c) CGAN Architecture.

Figure 4.3: Architecture of Conditional GAN.

- **Condition Generation -** We use the **Embedding Layer** of the Siamese Network, where all the user data is sent as input and an embedding vector is generated. This gives us the "Conditional Input" for the CGAN.
- **Generator Computation -** The Conditional Input is concatenated with a noise vector and sent to the generator. The generator uses this input to generate an image of the same dimension as in the original Fashion-Mnist dataset.
- **Discriminator Computation and Training -** The Conditional Input and the image (either from the source dataset or from the Generator) are concatenated and sent to the Discriminator. The Discriminator outputs 1 or 0 depending on whether it predicts that the image is from the source or from the Generator, and based on this the Generator is trained to learn to create images that the Discriminator can not distinguish from the originals.

Figure 4.3(a) and (b) show the Generator and Discriminator network structures, respectively, and Figure 4.3(c) shows their integration into the CGAN architecture.

4.3.4 Gaussian Mixture Model for New User Preferences

The Branching Network from Siamese Network takes all the images from previous users and generates embedding vectors which create the embedding space. These unlabeled vectors in the embedding space will be used to bootstrap and structure the user preference for new users. When the model works on predicting the preference of a new user, the new user might prefer images from different clusters (previous user preferences) or sub-cluster combinations. The goal is to capture these by building a Gaussian Mixture Model for the new user's likes and dislikes, capturing them in the form of soft clusters in the embedding space. This implicitly assumes that the structure of the embedding space captures basic commonalities related to human tastes. Note that we are not assuming that resulting user preferences fall into a type that is already present among previous users (as is done generally in collaborative filtering) but only that the embedding space captures basic underlying data properties in light of human preferences. The training algorithm works in a semi-supervised fashion, treating user feedback as preference labels while assigning no labels to other users' data. The presence of liked and disliked examples leads to a small modification where each Gaussian also receives a cluster label, assigning it to either the liked or disliked class for the user. The complete algorithm operates in the following way:

- **Initialization -** For initialization, K-means Clustering is a good choice. The value of the cluster number k is defined by the elbow method. These clusters provide a starting point for the Gaussian Mixture model to train itself. The mixing probability or weights π is initialized according to the data distribution, and as initially, no data for the new user is available, membership in the liked or disliked distribution for each Gaussian is initialized randomly.
- **Expectation-Maximization -** The EM algorithm finds the maximum likelihood of a vector among all the clusters. It is divided into two steps:

  E-Step - In the E-Step the posterior probabilities (Expectation value) of each data point are computed with the given π, μ, and Σ. For data points that received feedback from the new user, posterior probabilities for all Gaussians that have the opposite liked or disliked label as the feedback indicates are set to 0.

M-Step- In the M-Step, π, μ, and Σ are updated by maximizing the log-likelihood with respect to the parameters. Once data points with user feedback are available, cluster labels for all Gaussians are updated according to the likelihood of the cluster generating liked or disliked data points from the set that the user provided feedback on.

The E-M step continues until the algorithm converges. The architecture is shown in Figure 4.4

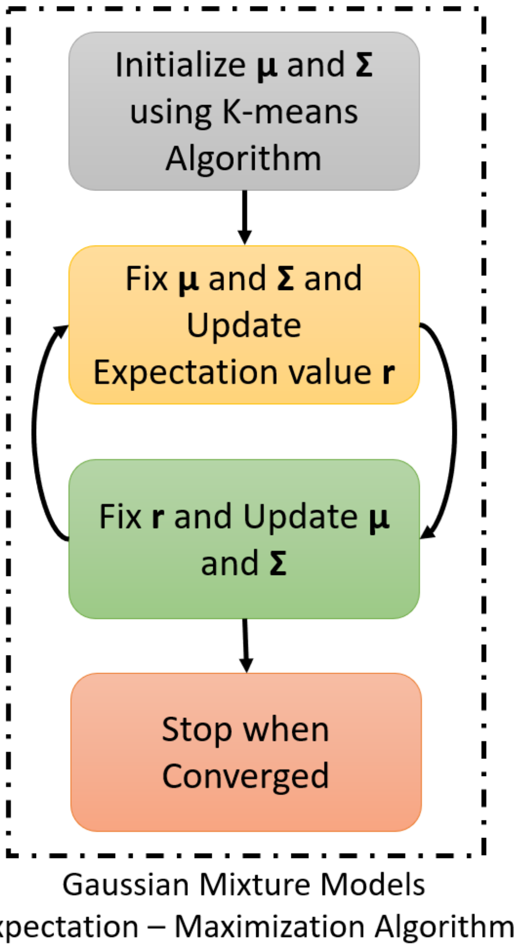
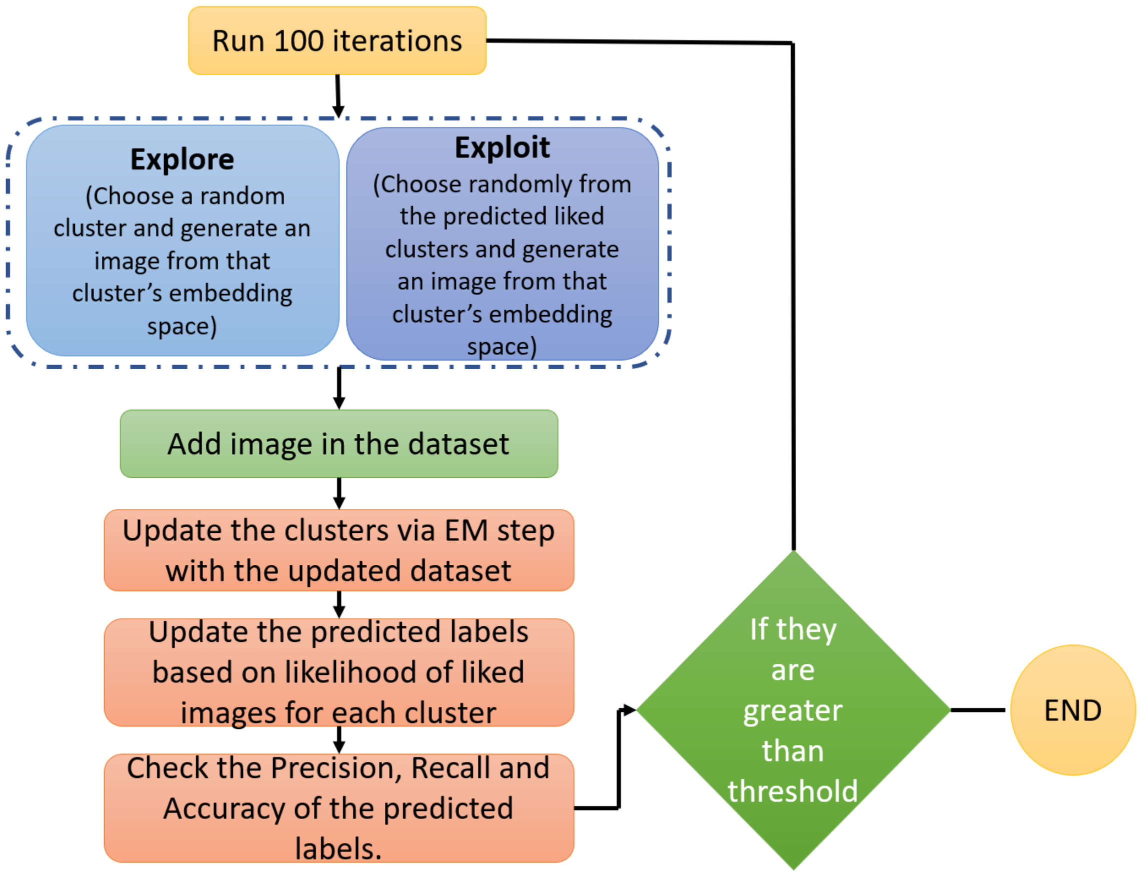

Initialize **μ** and **Σ** using K-means Algorithm
Fix **μ** and **Σ** and Update Expectation value **r**
Fix **r** and Update **μ** and **Σ**
Stop when Converged
Gaussian Mixture Models
Expectation – Maximization Algorithm

Figure 4.4: Architecture of Gaussian Mixture Model

4.3.5 Prediction Model

The Prediction Model interacts directly with new users and based on their feedback predicts the images a user might like. The following steps are used for the model to work.



Run 100 iterations
Explore
(Choose a random cluster and generate an image from that cluster's embedding space)
Exploit
(Choose randomly from the predicted liked clusters and generate an image from that cluster's embedding space)
Add image in the dataset
Update the clusters via EM step with the updated dataset
Update the predicted labels based on likelihood of liked images for each cluster
Check the Precision, Recall and Accuracy of the predicted labels.
If they are greater than threshold
END

Figure 4.5: Architecture of Prediction Model

- Initially there is no user information, so a cluster from the embedding space is chosen randomly and the corresponding μ and Σ are used to pick a vector from that cluster.

- This vector is then used as the Conditional Input for the CGAN. The generated image is shown to the user and the feedback is recorded as "Liked" or "Disliked". The embedding vector is also added to the dataset.
- The EM algorithm is called to update the Gaussian Mixture Model based on the latest dataset and user feedback. During the EM step, the new images for the new user are updated differently. If the user has liked or disliked the image, its Expectation value for all the predefined clusters which the model thinks has the opposite label is set to 0. This process moves the clusters every time, customizing the prediction for the new user.
- After the EM step, the predicted labels are updated. For each cluster, we find the probability density values of the liked images (li) and disliked images (di), and then use the formula $(li - di)/(li + di)$. If the value is greater than 0, then the cluster is liked, otherwise, it is disliked.
- The model runs for at most 100 iterations. Initially, it explores 90% of the time with a decay of 20% every 20 iterations. This initially promotes exploration and slowly moves towards exploitation. At each iteration, Precision, Recall, and Accuracy are evaluated and training is stopped if a threshold is crossed. For the experiment, the threshold for (Precision, Recall, and Accuracy) is (80%,50%,50%). We use a higher Precision value than Recall since we want the model to learn to present liked images early but are not as interested in covering all interests of the user.

The Model is shown in Figure 4.5

4.4 Experiments and Results

In this section, each of the components, as well as the complete approach, is tested and evaluated.

4.4.1 Siamese Network

The Siamese Network was trained with user-consistent and user-inconsistent pairs. The network ran for 250 epochs and achieved a testing accuracy of 97% and validation accuracy of 95%. The training loss is 0.11 and the validation loss is 0.21. The training curves are shown in Figure 4.6.

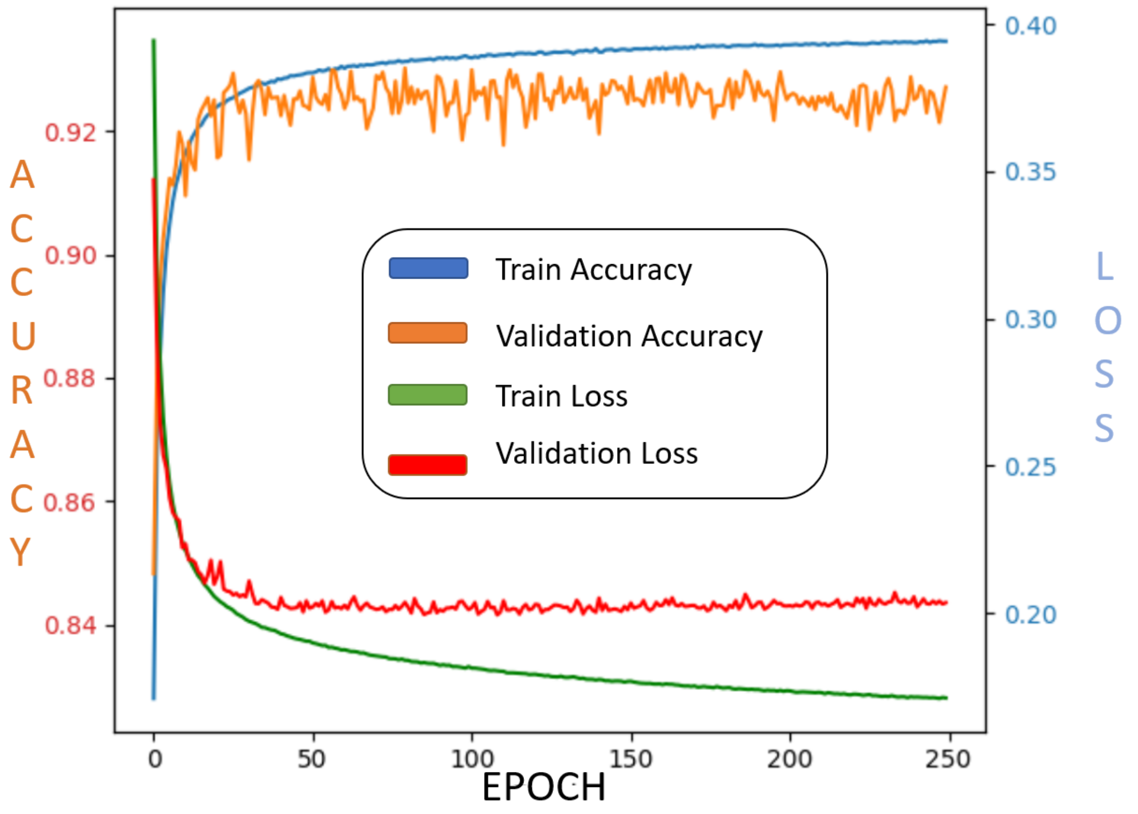


Figure 4.6: Accuracy of Siamese Network

These results show that the Siamese network is able to learn consistent embedding of the distinguishing characteristics of existing user data with high precision.

4.4.2 Gaussian Mixture Model

Using the embedding space and the existing user data without labels, a Gaussian Mixture Model with 18 clusters (chosen by the elbow method) is trained and then customized using new user feedback. The clusters are shown in Figure 4.7, where (a) displays the clusters before training with the new user's feedback, and (b) shows the clusters after training, where red clusters are Disliked and green clusters are Liked.

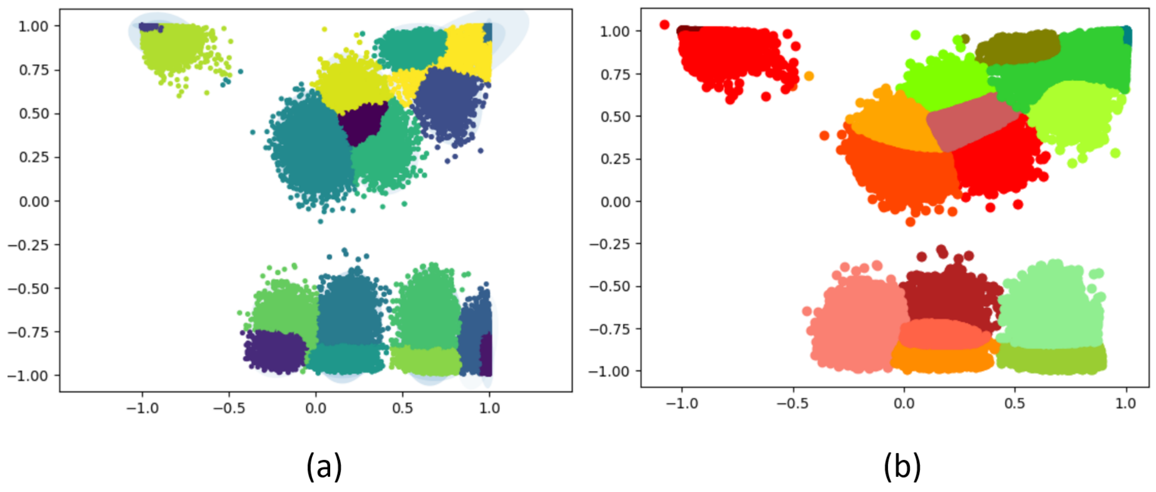


Figure 4.7: Soft Clusters in Embedding Space Before (a) and After (b) User Interaction

This figure shows that the system is able to form distinct clusters that align with the users, indicating that it can identify relevant characteristics within the embedding space.

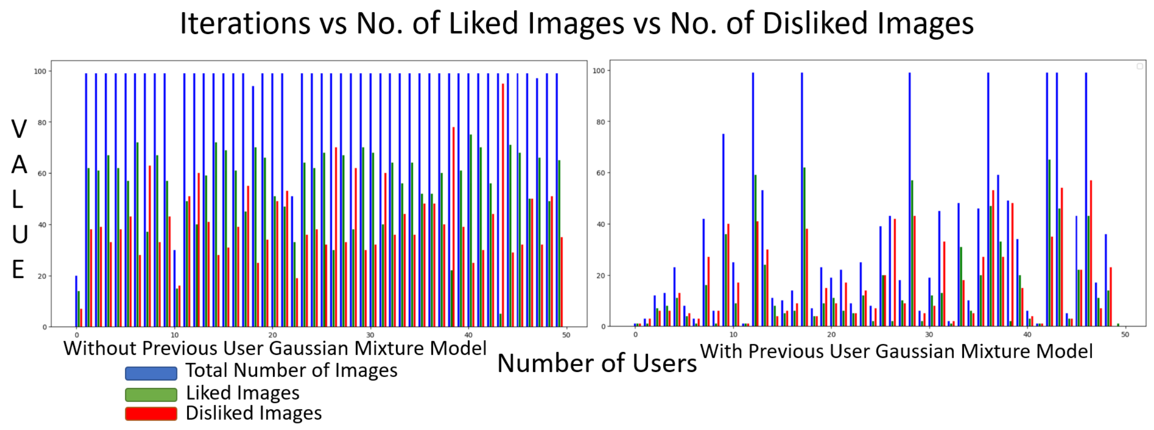


Figure 4.8: Iterations Required to Train User Model for Each of the 50 Users Without Previous User Gaussian Mixture Model (left) and With Previous User Gaussian Mixture Model for Collaborative Filtering (right).

4.4.3 Prediction Model

Once the Gaussian Mixture is pre-trained, the prediction and feedback process is started with a set of 50 new users with randomly initialized preferences defined as subsets of the Fashion-Mnist categories. To compare the efficiency of the proposed architecture, another experiment is set up with the same number of clusters but where the Gaussian Mixtures do not utilize previous users' data but only the new user's data points and feedback. This corresponds to the situation where no collaborative filtering is used but preferences are learned in a supervised way using the pre-trained embedding space. Figure 4.8 shows the number of iterations the models required in each condition to reach the prediction thresholds for each of the new users and how many Liked and Disliked images each user has seen during the training process. The use of the collaborative filtering approach by pre-trained Gaussian Mixtures (shown in the right figure) significantly reduces the need for user feedback compared to the baseline (shown on the left) and thus increases the efficiency of the architecture. Using this, the system is able to learn preference distributions relatively fast, with many users having to rate less than 20 images. However, the results also show that there still is significant variance among users, indicating the potential for improvement in the way data points are generated from the Mixture distribution. Focusing more on liked clusters should here increase precision at the cost of recall with fewer disliked images, while focusing longer on exploration should yield higher recall but more disliked images. The model with no pre-trained Gaussian Mixture (shown in the right figure), required significantly more interactions with only 3 instances where fewer than 40 images were shown. For the rest of the users, the model completed its run of displaying all 100 images and never reached the desired thresholds.

To check how efficient the proposed user preference learning approach is in predicting user preferences, we generated 100 images from each liked and disliked cluster according to the trained user-specific Gaussian Mixture model's predicted preference for the user. These images were then tested against the actual preference of the user and precision and recall values were computed. Precision shows how well the model can generate items matching each user's preference while Recall measures how comprehensively the user's preferences are captured. Figure 4.9 shows the precision and recall for each of the 50 new users for the proposed model with the pre-trained Gaussian Mixture from previous users' data.

The model achieved an average precision of 81.9% and an average recall of 71.3% across all 50 users. By contrast, the model without the pre-trained mixture only achieved an average precision of 36.6% and an average recall of 37.2%.

Figure 4.10 similarly shows the model accuracy value for each user using the pre-trained Mixtures.

The average accuracy here is 82.67%, which contrasts with only 60.18% without the collaborative pre-training. These results again demonstrate the ability of the proposed architecture to learn effective user preferences with limited interactions and illustrate the importance of the approach's ability to utilize previous users' data.

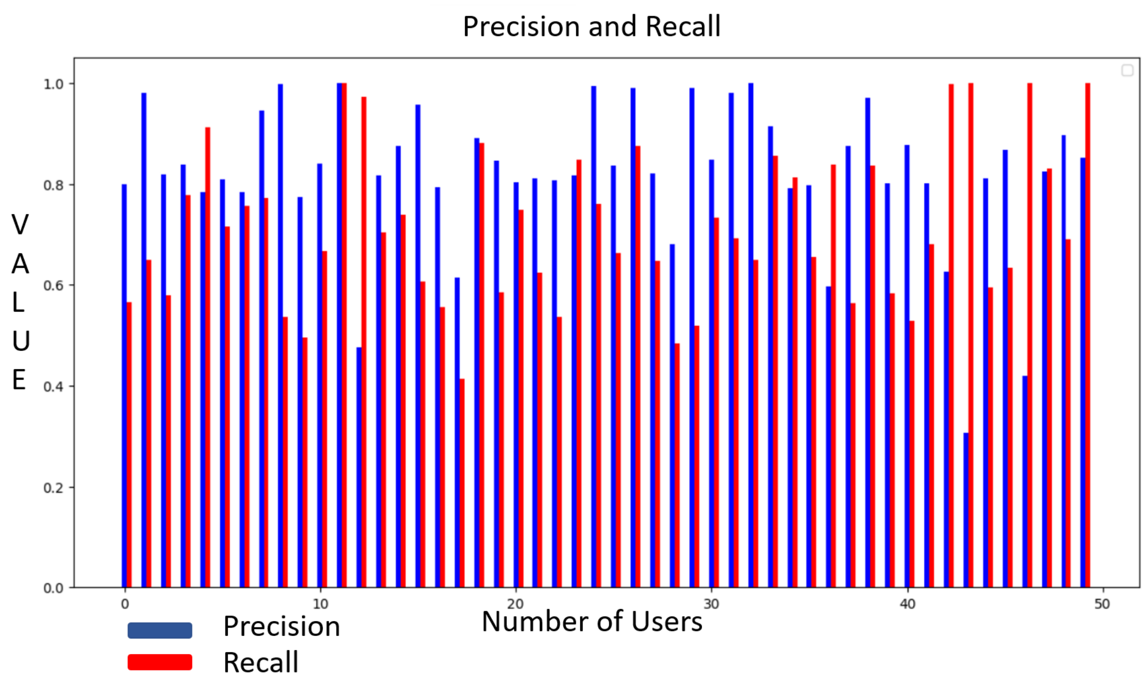


Figure 4.9: Precision and Recall of Prediction Model

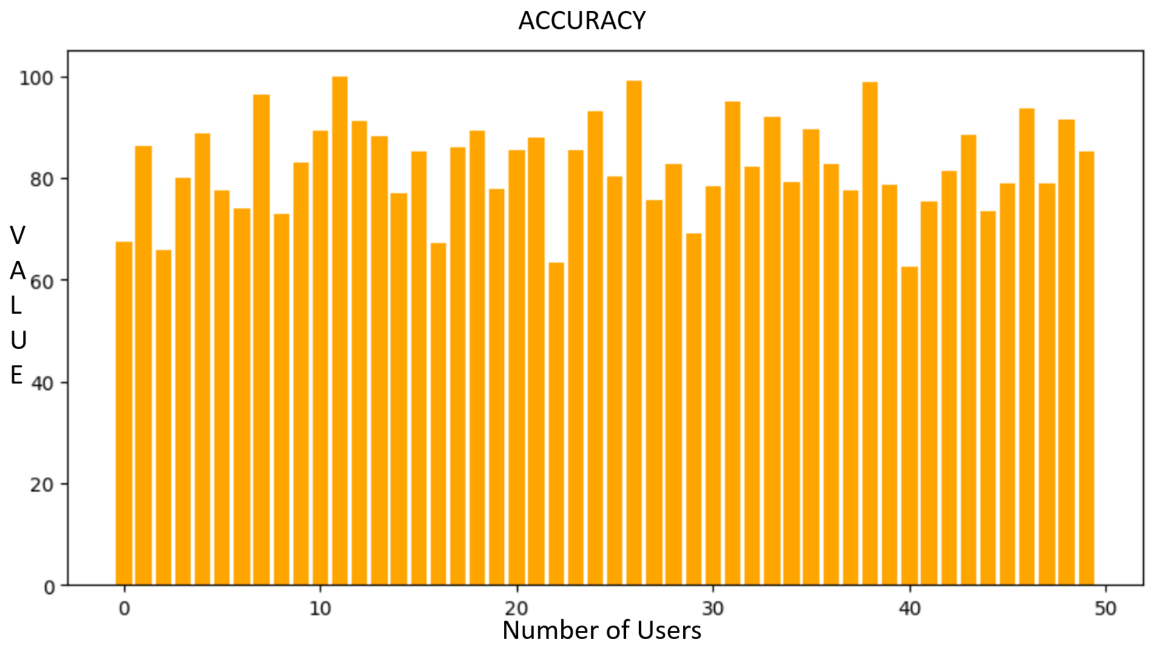


Figure 4.10: Accuracy of Predictive Model

4.5 Conclusion and Future Work

This research work presents a novel framework that uses the concept of collaborative filtering in conjunction with a Siamese Network, Gaussian Mixture Models, and CGAN to create a predictive model. This predictive model generates an accurate preference prediction for new users with limited amounts of user-specific data. For this, it establishes a user preference distribution as a Gaussian Mixture Model on an embedding space learned by a Siamese Network using other users' data. Results in an image generation domain show that the system is capable to embed important general preference characteristics. It successfully uses these to build a user preference distribution of unknown preferences from limited feedback and generate training data efficiently using a Conditional GAN network. While the results presented here are promising, the used exploration and exploitation policy is very simple and could be improved upon in terms of reducing false-positive exposures using other exploration strategies. We can also use exploration strategies that are more targeted to a specific task to further improve the experience of users with the system.

CHAPTER 5

Adaptive Task Presentation to Improve User Performance in a Gaming Domain

5.1 Introduction

Learning user behavior models in the gaming domain has always been a fascinating field of research. Understanding how the player will behave provides a lot of information that can be effectively used to make decisions on game designs. This can lead to more captivating and interesting games, that can keep the players engaged as long as possible. Various game design methods are implemented, such as procedural generation which generates game levels on the go based on some rules and regulations decided by the game designer. But what happens if the vision of the game designer does not align with the player? This can lead to bad game design resulting in a loss of interest among players. So what if the rules and regulations for the game design are influenced by the user behavior model? It will lead to developing game levels that are personalized to the specific player's interest. It can not only learn user behavior but also adapt itself to generate game levels that will be interesting for the user, thus leading to increase interest from the player.

The goal of this research is to introduce a novel architecture that can not only learn about user behavior but also learn and generate game levels with the aim of increasing user interest in the game. For this, an RL agent (Richard S Sutton 2018) (**Outer Agent**) has been used which will learn to generate Game levels and will be trained based on how the user performed while playing the game level. Facial Emotion (De Silva, Miyasato, and Nakatsu 1997) is captured and used as a parameter of user performance. The concepts of Embedding space, Collaborative filtering (Goldberg et al. 1992), and Conditional GAN (Mirza and Osindero 2014) are used to create an **Inner Agent** which learns about the user behavior, including their emotional actions and helps to train the Outer Agent to create more meaningful and interesting game levels. One of the main goals of the architecture is to learn about the user behavior with minimal interactions, where the Outer Agent is trained more on the Inner Agent and gets periodic exposure to the real user to validate its training.

The next sections will elaborate on related works, then provide an overview of the tools used in the learning framework and how the whole architecture is connected. This is followed by the experimentation carried out and the results received, ending with conclusions.

5.2 Related Work

Researchers have always been interested in predicting players' gameplay. It provides valuable information about how the player uses the information present in the game to develop a strategy to increase their chances of winning. (Erev and Roth 1998) showed how simple learning models with motivation by the psychology of learning can model player interaction with the goal to predict accurate gameplay. Though the experimentation is done on simple games, (Mahlmann et al. 2010) shows the implication of predicting player behavior in video games. They chose a major commercial title Tomb Raider: Underworld (TRU), where a large-scale set of in-game player behavior data is used to train various supervised learning algorithms that will predict when the player might stop playing, and also the time required to finish the game. (Harrison and Roberts 2011) presented a data-driven solution to design user behavior models. The models were tested on the achievement data from the game World of Warcraft.

Though the aforementioned research has been successful in predicting user behavior models, they were not directly used to aid game design or build adaptive games. (S. C. Bakkes, Spronck, and Lankveld 2012) shows the potential of using this information to create interesting and effective AI and also creating an opportunity for game developers to personalize the player's gameplay as a whole, by providing new user-driven game mechanics. To show how well player models can be used to create personalized games, (S. Bakkes, Tan, and Pisan 2012) provides a literature overview on the concept of personalizing games.

In all the work presented above we see that to generate a good user behavior model, a huge amount of user information is required and there is an opportunity to use this information to create adaptable and personalized games. The novel Architecture introduced in this research aims to predict player behavior with minimal interaction with the player. This reduces the dependence on huge datasets. The Architecture also uses this knowledge to train a Reinforcement Learning agent that strives to create personalized and adaptable gameplay for the individual player.

5.3 Game and Learning Framework

Before we explain the learning framework and its training, let us elaborate on the Game domain which has been developed for the research work. The Game has been developed from the tutorial posted on youtube (Russ 2020). It is a simple 2d platformer game, where the player moves around the environment to reach the door (the end point of the level). In between it has to climb blocks, dodge enemies and lava, and also collect coins. Once it reaches the door the game moves to the next game level. Figure 5.1 shows the game in the tutorial. The reward is based on the number of coins collected and whether the player reached the endpoint. One problem with the game is that if the player dies (by colliding with the enemy or lava), it restarts from the beginning of the game level. Since this can lead to more frustration among

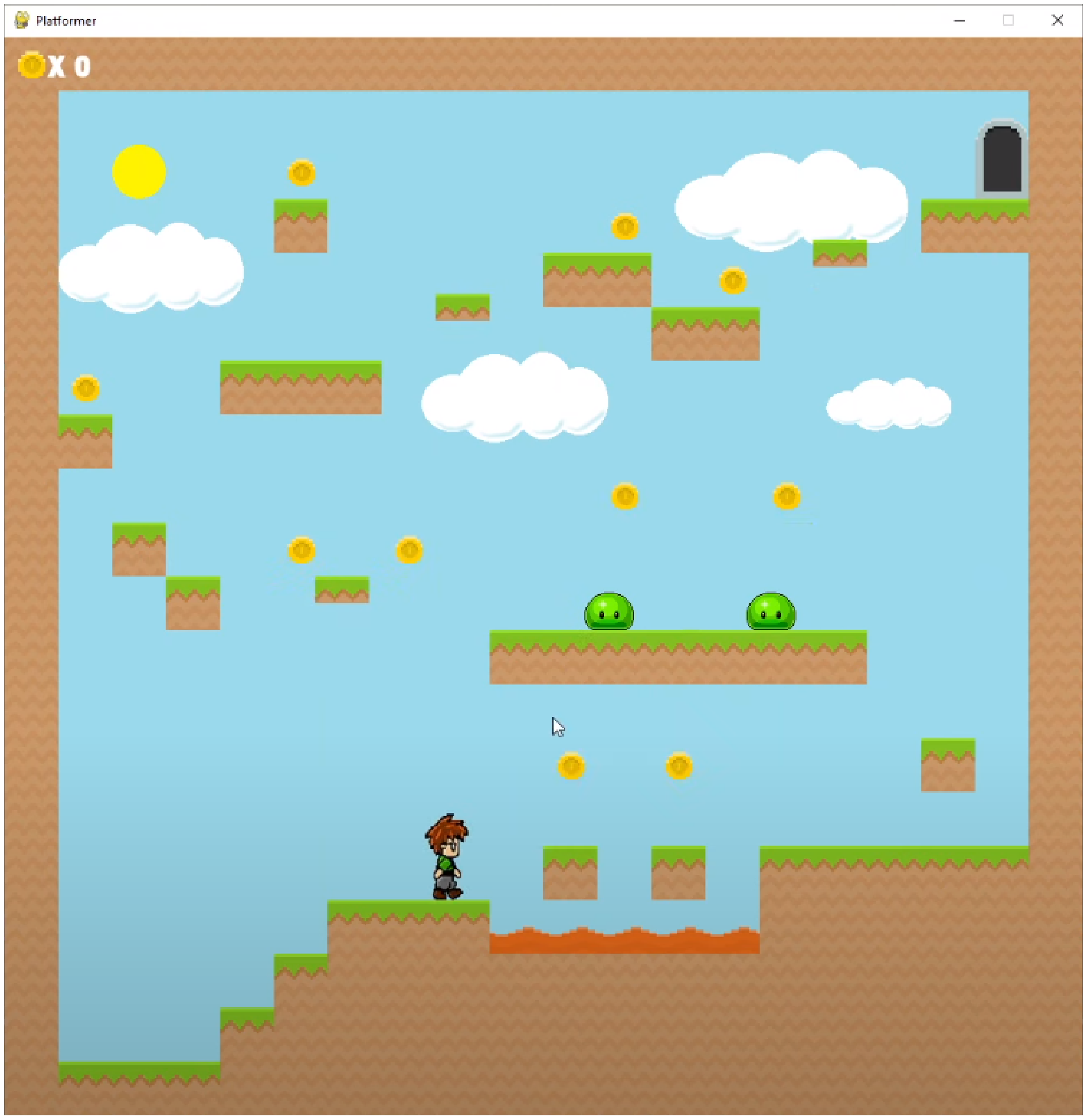


Figure 5.1: Game Level

players, we introduced a 3-life option at the game level. So now when they start playing the game, they start with three lives. Every time they die, they lose one life. When there is no more life left, the game ends. Figure 5.2 shows the new addition to the game. To add a little more difficulty to the game we time-bounded the game

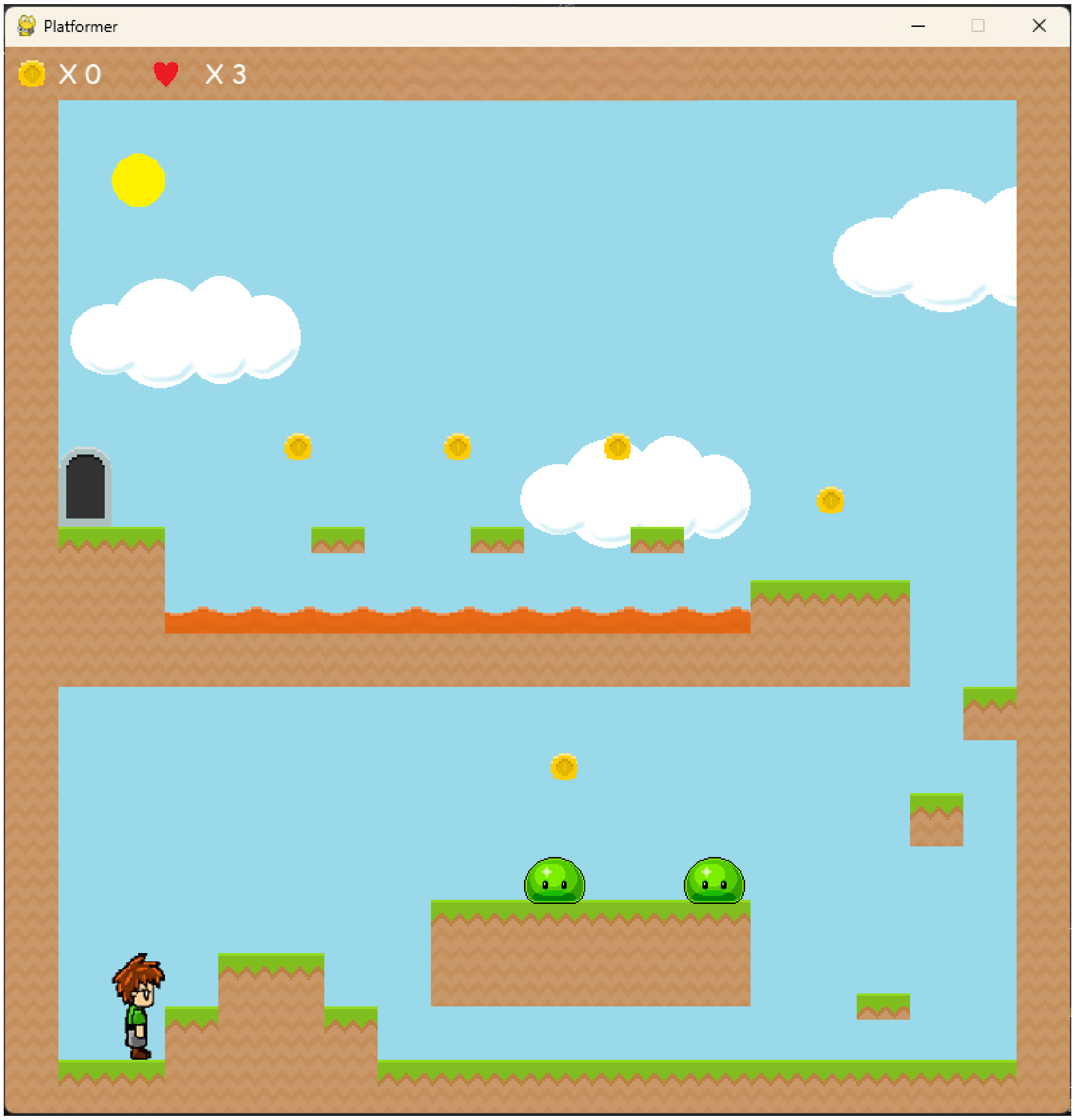


Figure 5.2: Game Level with extra lives

levels. Each game level will contain an internal timer and the player has to complete the level before the timer ends. This new addition will display the proficiency of each player under stress and provide valuable information about the player. Figure 5.3

displays the final form of the game. When the player interacts with the Game level,

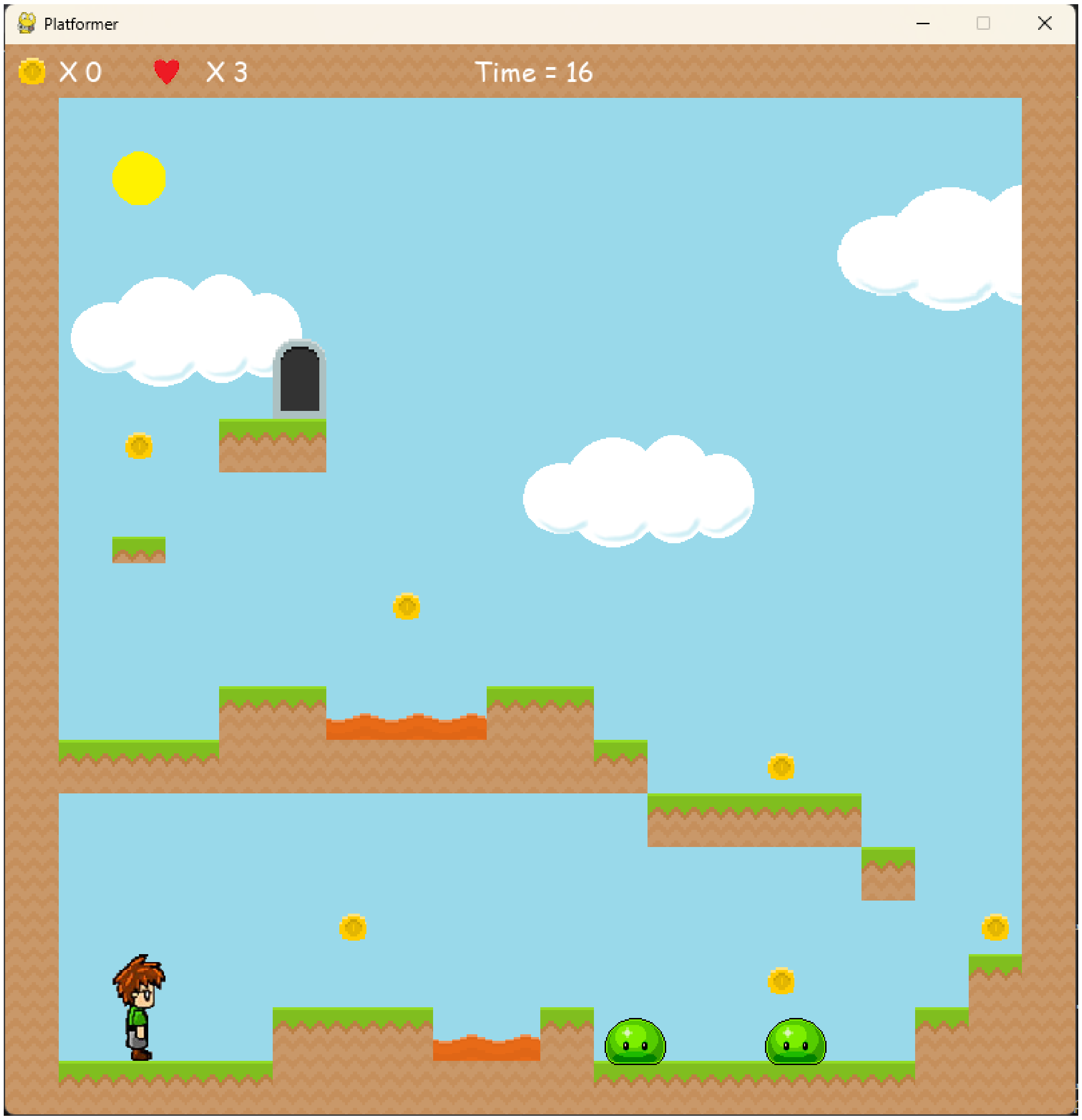


Figure 5.3: Final Game level

the performance is registered. The number of total jumps tried, the number of total jumps made, the number of total enemies averted, the total number of coins collected, and if the player finished the level or not are recorded as player performance.

Integrating this into the learning architecture presented in this dissertation, the Outer Agent is responsible for creating specific game levels by generating the Game states. It interacts with both the player and the Inner Agent to train itself. The Outer Agent receives the current game state and player Emotion Information and then generates the game state corresponding to the next game level to present to the player. Initially, it here largely explores the space of possible game states but over time increasingly exploits the learned knowledge. Based on the reward the Outer Agent receives, which is extracted from the performance data captured from the player and environment, the Outer Agent is trained using Q-Learning. In the beginning, after the real user has interacted with the Outer Agent, the user performance is used to generate an embedding vector which helps locate the clusters that are more likely to provide a better prediction for the new player. Once that is learned, the Inner Agent produces performance predictions and with it significantly larger amounts of data to train the Outer Agent for some time before it interacts with the real user again. The goal here is to minimize user frustration resulting from negative game experiences due to inappropriately easy or hard game levels by reducing the number of real user interactions required during training. The complete framework is depicted in Figure 5.4

5.4 Learning Framework Implementation

5.4.1 Game State

The Game environment state is a vector of size 10. It contains all the important parameters required to generate the Game level. Each parameter, directly and indirectly, influences the performance of the player. The information in each data point is as follows:

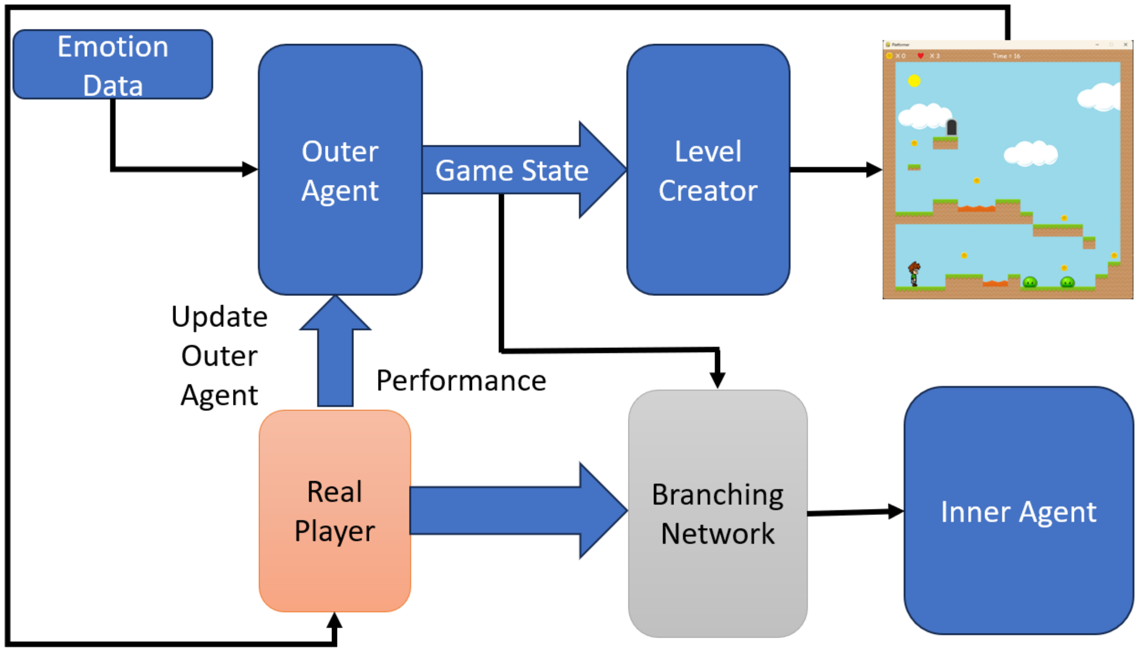


Figure 5.4: Learning Framework

- **Trajectory 1:** This feature informs the availability of the first goal trajectory at the game level. It is always set to 1.
- **Time 1:** This feature stores the time which according to the game is required to complete the first trajectory.
- **Trajectory 2:** This feature informs about the availability of a second goal trajectory at the game level. It can store values 0 or 1.
- **Time 2:** This feature stores the time which according to the game is required to complete the second trajectory.
- **Side Trajectory:** This feature informs the availability of a side trajectory, i.e. a trajectory that does not lead to the goal but instead to a dead end. It is available only if there is no second trajectory. It can store values 0 or 1.
- **Time 3:** This feature stores the time which according to the game is required to traverse the side trajectory.

- **Jumps:** This feature stores the number of jumps required to traverse the game level.
- **Enemy:** This feature stores the number of enemies in the game level.
- **Coins:** This feature stores the number of coins to collect in the game level.
- **Heart:** This feature stores the number of hearts to collect in the game level.

Fig 5.5 shows the Game Environment State for each Game level.

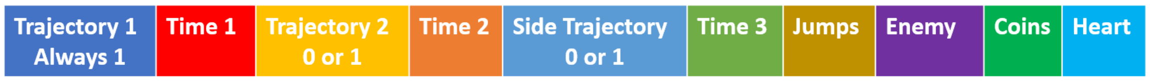


Figure 5.5: Game State

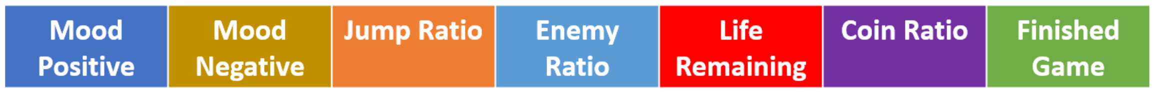
5.4.2 User Performance Dataset

The User Performance dataset contains performance vectors of the player for a particular game level and also the Emotional information captured using FER while the performance takes place. This dataset provides invaluable information on the player's emotional state as well as their proficiency in the game. The User Performance is here captured in terms of the following features:

- **Mood Positive:** This feature stores information on whether the user is in a positive mood. The "Mood Positive" feature and "Mood Negative" feature total a value of 1.
- **Mood Negative:** This feature stores information on whether the user is in a negative mood. The calculation of the player's emotional state will be explained in detail in the next section.
- **Jump Ratio:** This feature stores the ratio of successful jumps out of the total number of tries.

- **Enemy Ratio:** This feature stores the ratio of successfully evading the enemy out of the total number of tries.
- **Life Remaining:** This feature stores the number of lives available after the game level.
- **Coin Ratio:** This feature stores the ratio of successfully collecting the coins out of the total number available.
- **Finished Game:** This feature stores 1 if the player won and 0 if the player lost.

Figure 5.6 shows the User Performance Dataset for each gameplay.



Mood Positive	Mood Negative	Jump Ratio	Enemy Ratio	Life Remaining	Coin Ratio	Finished Game

Figure 5.6: User Performance Vector

5.4.3 Facial Emotion Recognition

Behind every performance and action, there are two aspects always in play: The **physical aspect**, where the user is following the rules and taking the actions to register their performance, and the **emotional aspect**, which is dependent on what the player is feeling and how the player is processing the impact of the game on them. It is easier to register and store the physical aspect of performance, but to effectively understand the player's performance it is imperative to take the emotional aspect into consideration as well. Reading and storing the emotional aspect directly through direct questioning can be very invasive and uncomfortable and would not provide valuable information in the end. Hence there is a requirement for a non-invasive method, which will record the information with a certain amount of accuracy.

Since the beginning of recorded history, it has been observed that facial expressions of human beings have been a good way of predicting emotions. Most people consciously or subconsciously, express their emotions through their facial expressions. Hence, if facial expressions can be effectively predicted by an algorithm, emotional information can be estimated and stored in a non-invasive manner. There has been research on Facial Emotion Recognition using Multimodal information, (De Silva, Miyasato, and Nakatsu 1997) where both audio and video are used to effectively predict human emotion. To remove the dependency on the Multi-modal aspect, The Facial Emotion Recognition (**FER**) (Shenk et al. 2021) library in Python is used for detecting Facial emotions.

It is an open-source Python library built and maintained by Justin Shenk and used for sentiment analysis of images and videos. The library uses **MTCNN** (Multi Cascade Convolutional Network) (Sun, Xiaogang Wang, and Tang 2013). The MTCNN framework can constructively do both face detection and face alignment. It uses a process of three stages of Convolutional Networks that recognize faces and landmark locations like eyes, nose, and mouth. The FER library detects emotion in the following way:

- **MTCNN:** It is a parameter of the constructor. When it is set to **True**, the library uses the MTCNN algorithm to detect faces. When set to "False", the library uses the default OpenCV Haar cascade classifier (Soo 2014).
- **Detect emotion():** This function in the library classifies the detection of emotion and registers the output of 7 categories, namely 'Anger', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral', All the emotions are calculated and the output is put on a scale of 0 to 1.

For our research, we have clubbed the seven emotions into two. 'Anger', 'Disgust', 'Fear', and 'Sad' has been clubbed together into one classifier called 'Negative', whereas 'Happy', 'Surprise', and 'Neutral' are clubbed together into 'Positive'. This reduces the number of features from seven to two but still keeps their relevance in the output. Figure 5.7 shows the new classifiers.

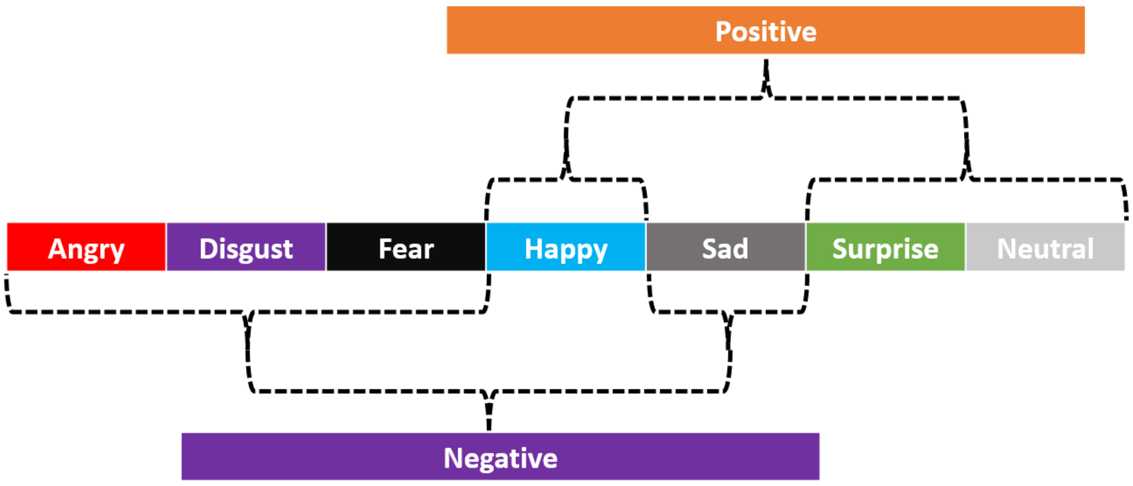


Figure 5.7: Output of Facial Emotion Recognition

5.4.4 Inner Agent

The goal of the Inner Agent is to rapidly build a predictive, generative model of the specific user playing the game. It is based on the approach in the previous chapter, using a Siamese Network to build a user-type embedding, GMM to characterize the user in this embedding space, and a Conditional GAN to predict user performance data for new game levels.

5.4.4.1 Siamese Network

The Siamese Network (Bromley et al. 1993) has emerged as a powerful tool for similarity learning tasks. With their ability to learn discriminative features and handle limited labeled data, Siamese networks can be used to generate embedding spaces for the dataset. This embedding space stores relevant information about old and new users which helps in predicting user preferences. The Network can be divided into three components:

Input Layer:

For the input of the Siamese Network, we concatenate the game environment and the corresponding user performance. This concatenation not only provides information about the game level but also the inherent performance of the user for the game level, making the dataset rich in information. Once this concatenation is achieved, We generate user-consistent pairs and user-inconsistent pairs for the Siamese Network to train on. Figure 5.8 shows the dimension of the input data.

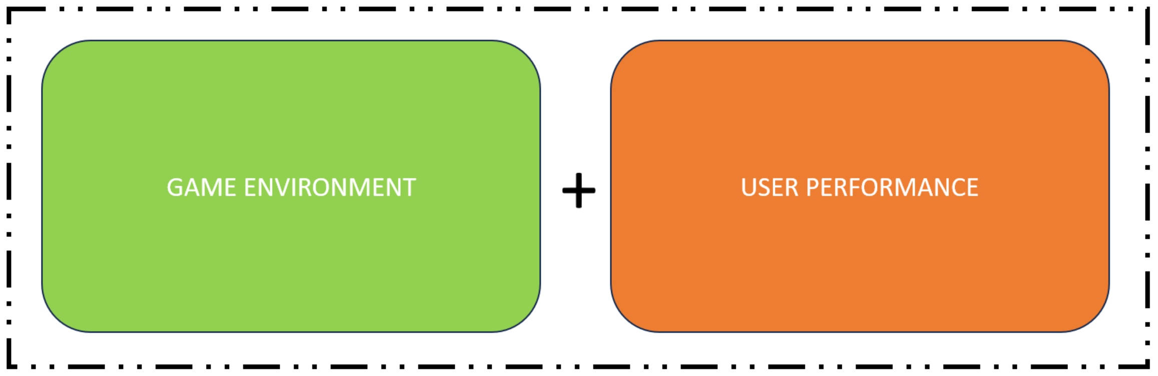


Figure 5.8: Input of Siamese Network

Branching Network:

The Branching Networks are two identical neural networks that receive each of the data from either the user-consistent pair or user-inconsistent pair and output an embedding vector which the Decision Component uses to find out whether the data belongs to the same user or a different user. The Branching networks are trained with the same weights since sharing the same weights ensures consistency in the embedding space and means that there are fewer parameters to learn during training and they can produce good results with a relatively small amount of training data. Figure 5.9 shows the Branching network. The Siamese network uses two of these Branching Networks.

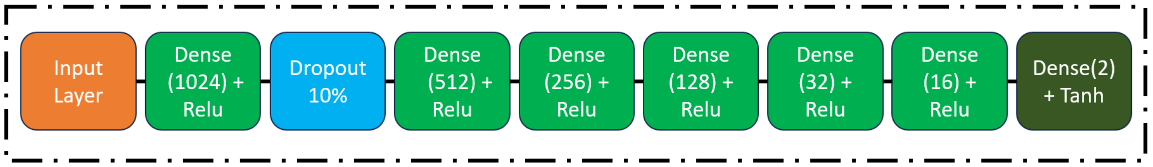


Figure 5.9: Branching Network

Decision Component:

The third component of the Siamese Network is the Decision Module. The Decision Component, shown in Figure 5.10 receives the output of the two identical Branching Networks and outputs either 0 (dissimilar) or 1 (similar). Firstly, the

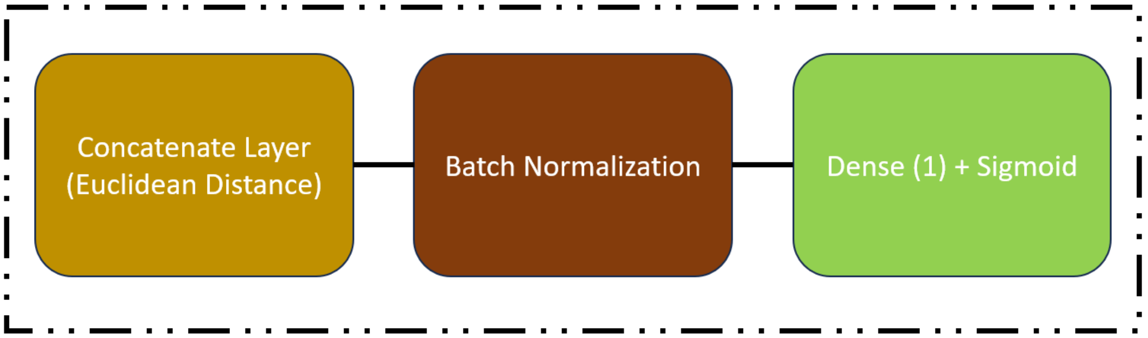


Figure 5.10: Decision Component

outputs of the branching network go through a Lambda Layer, where the Euclidean

Distance of the embedding vectors is measured. The output goes through Batch Normalization, after which, it goes through a fully connected dense layer with sigmoid activation to predict whether the pair of inputs is similar or dissimilar. To train the Siamese Network, Binary Cross entropy is used as the loss function and RMSprop is used as the Optimizer. Figure 5.11 shows the complete Siamese Network.

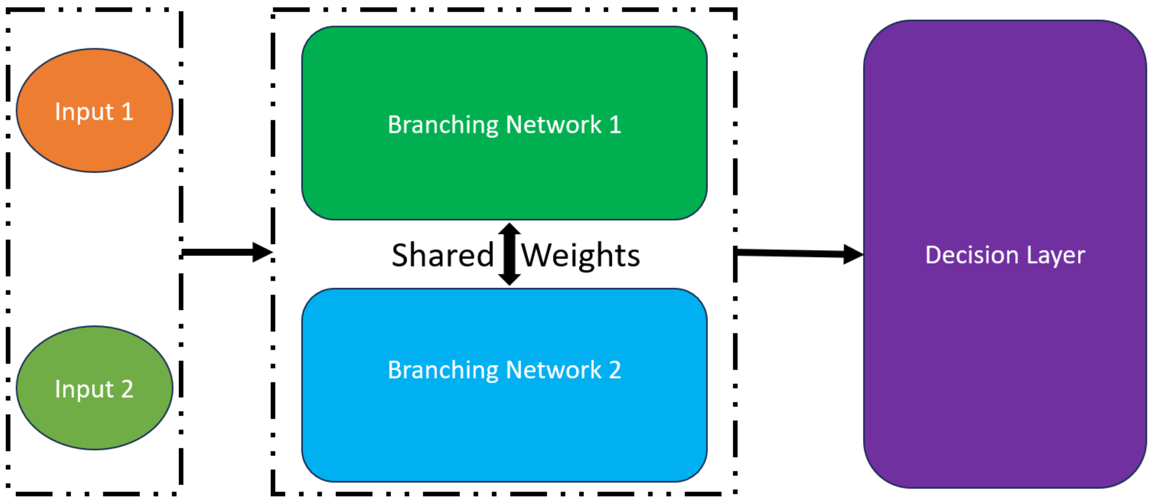


Figure 5.11: Siamese Network

Once the Siamese Network is trained, the data (Game Environment + User Performance) is provided as input to the Branching Network which generates the corresponding embedding vectors. These embedding vectors are further used in the Conditional GAN and Gaussian Mixture Models.

5.4.4.2 Conditional GAN

The Conditional GAN (Mirza and Osindero 2014) is one of the most important tools in the Architecture. It comprises the **Inner Agent** aspect of the Architecture in conjunction with the Gaussian Mixture Model. The objective of the CGAN is to generate User Performance predictions for the new user on unseen game levels. This will enable us to train our Reinforcement Learning agent without frequent interaction with the real user. The three important parts of the CGAN are:

Conditional Input:

Generative Adversarial Networks (Goodfellow et al. 2014) are really good at generating information but when it comes to generating data based on class labels, a GAN cannot make that distinction. Hence CGAN becomes a more powerful alternative to GAN. The Conditional Input is the distinguishing factor. With the help of Conditional Input, the CGAN can generate class-based data. For our research, the Conditional Input for the CGAN is the concatenation of the Game Environment, Emotional data, and Embedding vector from the Siamese network as shown in Figure 5.12.

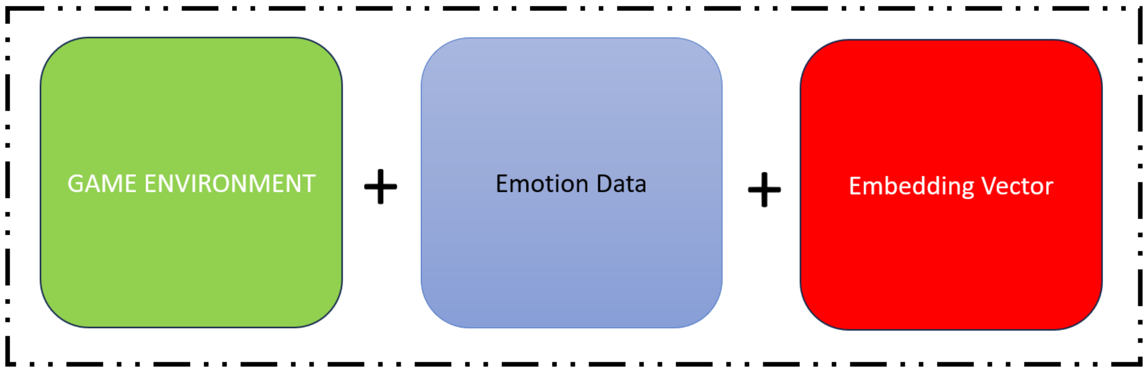


Figure 5.12: Conditional Input

This particular combination provides a proper class label characterizing the specific user and game environment and can be used to generate data (User performance) with the CGAN.

Discriminator:

The job of the Discriminator is to predict whether the data it received is from the data source, or from the Generator. It will receive a batch of performance data from both the source and the Generator, together with the corresponding Conditional Input, and based on this the Generator is trained to try to distinguish real and simulated user performance data. Figure 5.13 shows the Discriminator architecture. The architecture is very similar to the branching networks used in the Siamese Network. The only difference is that in the Branching network the output is a 2d vector whereas in the Discriminator, it is a single output stating 0 for false and 1 for true.

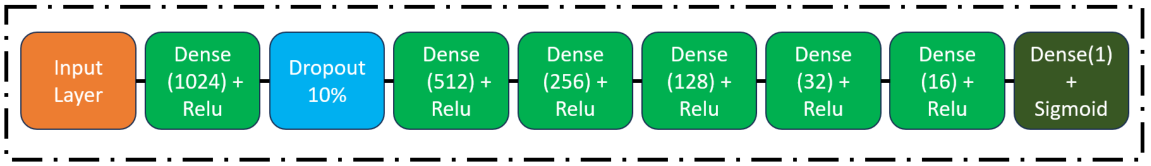


Figure 5.13: Discriminator

Generator:

The Generator network's main goal is to trick the Discriminator. The Generator receives the Conditional Input and a Noise Vector as input and it generates a piece of information similar to the original performance data. This information is fed to the Discriminator and based on the prediction, the Generator is trained. Figure 5.14 shows the Generator architecture.

This constant battle between the Generator and Discriminator in a zero-sum game trains the Generator to generate user performance data closely resembling the actual user performance for the particular Conditional Input. Figure 5.15 shows the workflow of the complete CGAN architecture.

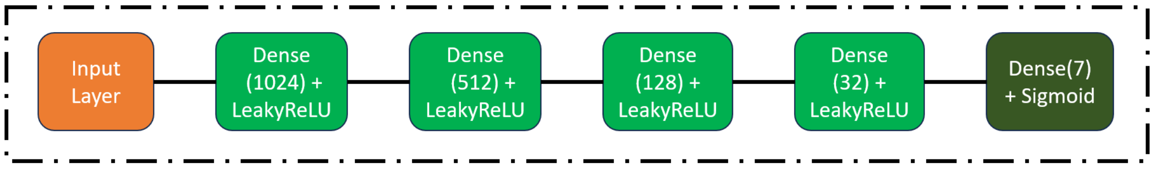


Figure 5.14: Generator

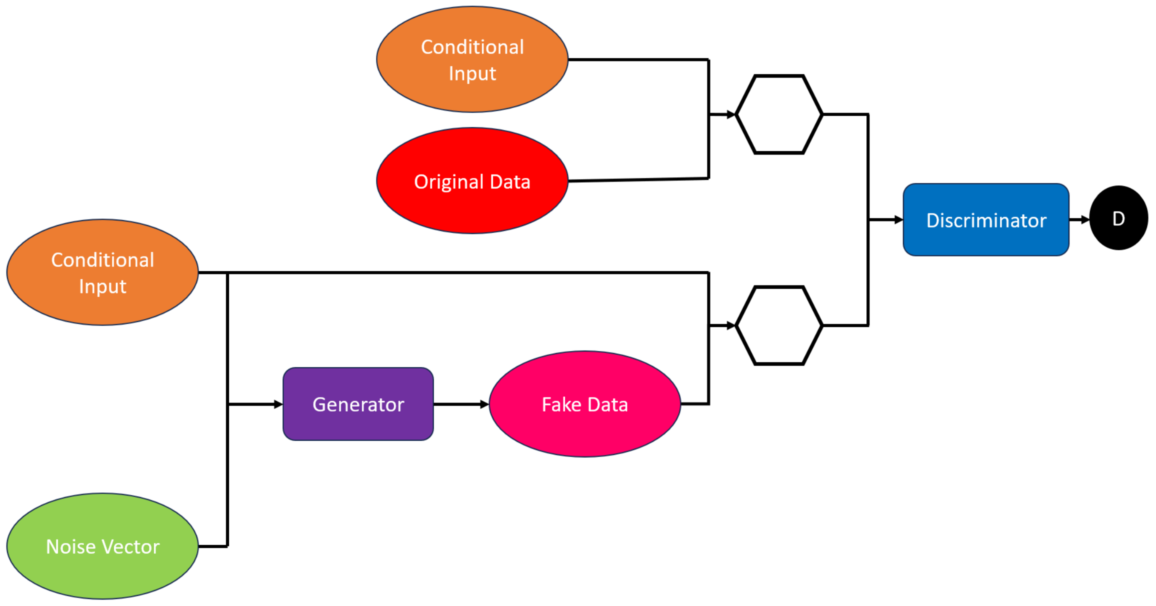


Figure 5.15: Conditional GAN

5.4.4.3 Gaussian Mixture Model

After the Siamese Network is trained, the Branching Network is used to generate embedding vectors for all the previous user data that is available. These embedding vectors will create the embedding space. Once the unlabeled embedding vectors from previous users are ready, they will be used to bootstrap and structure the user preference for new users. This is based on the notion that the new user might have similar characteristics with multiple previous users and hence might belong to different clusters or sub-cluster combinations within the embedding space formed by previous users' data. To capture this nuance, a Gaussian Mixture Model is used. The model builds based on whether embedding vectors generate correct or incorrect user performances for the new user and captures them in the form of soft clusters in the embedding space, implicitly assuming that the embedding space structure captures the basic commonalities related to human performance variations. The complete algorithm operates in the following way:

- **Initialization -** Before training the Gaussian Mixture Model, first we use the silhouette score method to determine the correct cluster number k. Once the cluster number is available, we initialize the embedding space using K-means Clustering. The data distribution is used to initialize the mixing probability or weights π. In the beginning, since no data for the new user is available, membership in the matching or non-matching distribution for each Gaussian is initialized randomly.
- **Expectation-Maximization -** Here the EM algorithm is applied to find the maximum likelihood of a vector among all the clusters. It is divided into two steps:

  E-Step - In the E-Step the Expectation value or the posterior probabilities of each data point are computed with the initial π, μ, and Σ. For the data points that received feedback from the new user, the posterior probabilities for all the Gaussians that have the opposite match label as the feedback indicates are set to 0.

  M-Step - In the M-Step, we update the π, μ, and Σ values by maximizing the log-likelihood with respect to the parameters. Once all the data points with user feedback are available, the cluster labels for all the Gaussians are updated according to the likelihood of the cluster generating user-matching or non-matching data points from the set that the user provided feedback on.

The E-M step will continue until the algorithm converges. In the end, it will display all the clusters which align with new user performance and also the clusters which do not. The architecture is shown in Figure 5.16.

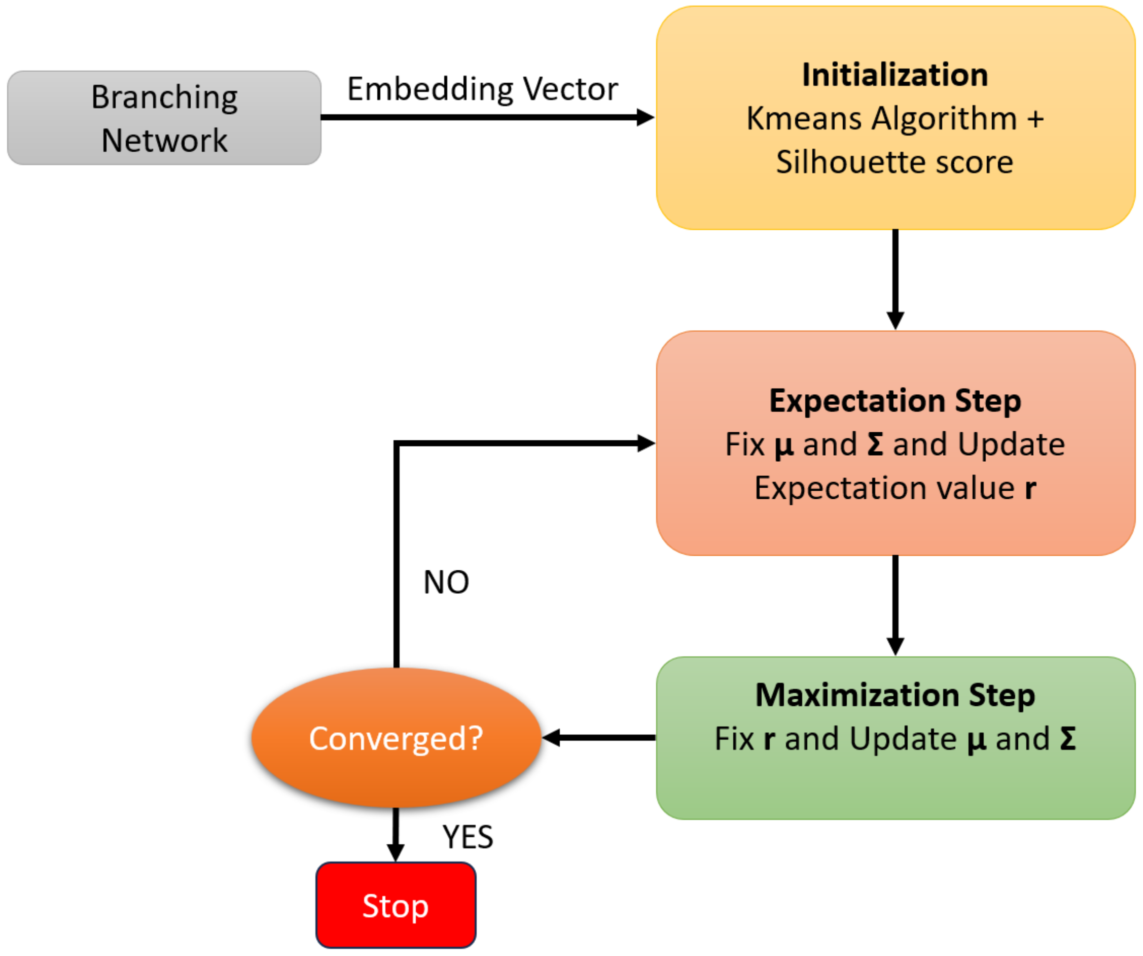

Branching Network
Embedding Vector
Initialization
Kmeans Algorithm + Silhouette score
Expectation Step
Fix **μ** and **Σ** and Update Expectation value **r**
NO
Maximization Step
Fix **r** and Update **μ** and **Σ**
Converged?
YES
Stop

Figure 5.16: Architecture of Gaussian Mixture Model

5.4.5 Outer Agent

5.4.5.1 Reinforcement Learning

For the Outer Agent, we use a Reinforcement Learning Agent. The RL agent is shown to be a robust agent which learns from its own experiences. It creates a balance between **Exploration**, where it explores the environment using random actions, effectively generating random game levels, and **Exploitation** where the agent uses the acquired knowledge to choose the best action in that particular state, generating what it thinks is the best next game level for the user given the previous level and current emotional state. Since the RL agent learns from its own experiences, training it only on real user information can require numerous gameplays which is often infeasible and usually frustrating due to the number of non-user-optimized levels that will be presented. To solve this problem, the RL agent gets trained with both the user performance and also the simulated performance generated by the **Inner Agent**. 10 actions have been chosen for the agent where each action provides a game state to be generated. The states available to the RL agent are the concatenation of the last action + Emotion data from the gameplay. Q learning is used to train the RL agent.

5.5 Training of Learning Framework

Now that all the underlying dependencies have been explained, let us elaborate on how the architecture works in conjunction with all the parts.

5.5.1 Pre-processing

Before the architecture starts working and interacting with users, we can preprocess some of the information. Firstly, the Siamese Network is trained with available user-consistent and user-inconsistent pairs. Once the Siamese Network is trained, the

Branching network is used to generate all the embedding vectors. Using these embedding vectors the Gaussian Mixture Model is trained. Similarly, the embedding vectors in conjunction with the game environment are also used to train the CGAN. From the previous user dataset, we also compute the covariance of all the performances and find the average of that data as a measure of the natural variability of the average user, which will later be used to determine whether simulated performance data from the Inner Agent is sufficiently similar to real user performance in the same environment to be considered matching or if it is sufficiently dissimilar to be considered non-matching. Once all the training is completed, The Branching Network, The Generator, the Gaussian Mixture Models, and the average of all covariance data is ready for the Architecture. Figure 5.17 displays the operation during pre-processing of the Inner Agent.

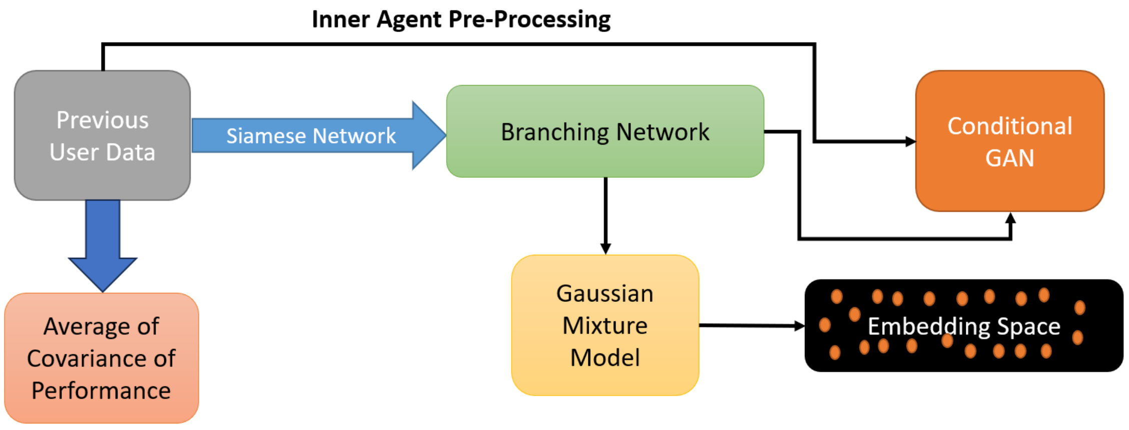


Figure 5.17: Pre-Processing of Inner Agent

5.5.2 Interaction with Real Player

After the pre-processing of the Inner Agent is completed, the Architecture can start interacting with real users. The Architecture interacts with the user in each iteration once with an upper limit of 100 iterations for training. In the beginning, the Emotion data of the user is captured and sent to the Outer Agent (Reinforcement Learning Agent). With the help of this information, the Outer Agent chooses a Game state which is then used to generate a Game environment. Since the Outer Agent has no prior information, a random action will be chosen. The new user is provided with the Game environment. The user plays the game and the performance is recorded and stored, including the Facial Emotions data. Once the information is ready, we first train the Outer Agent.

5.5.2.1 Training Outer Agent

We use the User performance to train the Outer agent using Q-Learning. Once the training is completed, we generate the embedding vector and sent all the information required to update the labels for the Clusters.

5.5.2.2 Update Cluster Labels

We have the original user performance and the new embedding vector. This embedding vector is used to generate 10 simulated performances for the same environment and Emotion data using the CGAN Generator and an average is calculated. We then subtract the original performance from the average simulated performance and divide the difference by the average of all covariance. Once it is done, we calculate the **L2 norm** value and call it **normdiff**. This value shows how close the simulated performance is to the real performance in terms of the number of standard deviations and thus corresponds to a likelihood that the simulated performance could occur from the real performance distribution of the user.

Next, we randomly generate 10 embedding vectors for each cluster. We use each of the vectors to generate simulated performance. Then we calculate the **L2**

norm value and call it **diff**. So if(**diff** <= **2*normdiff**), then the embedding vector is labeled as **positive**, i.e. it represents a predicted performance that matches the user performance. if(**diff** > **2*normdiff**), then the embedding vector is labeled as **negative**, i.e. it is unlikely to reflect the performance of the actual user. After all the embedding vectors from the cluster are calculated, if there are more positive points than negative, the cluster is **matching**, or else it is **non-matching**. This is done for all the clusters. We also store all these embedding vectors as labeled data and their positive and negative classifications as Outcome, to help update the GMM.

5.5.2.3 Updating the GMM

After updating the label, we update the GMM by running the E-M step first over the embedding dataset. Once it is trained, we use the E-M step again for the labeled data but it will be updated in a special way. For each of the labeled data, if the outcome is positive, the expectation value for the non-matching clusters is changed to 0. If the outcome is negative, the expectation value for the matching clusters is changed to 0. Once all the updates are completed, we check if the GMM requires retraining.

5.5.2.4 Retraining the GMM

Out of the labeled data, for each matching cluster, if there are more negative vectors than positive ones, we retrain the GMM with one extra cluster. The notion is that if there are more negative examples in a matching cluster there is a chance of a hidden cluster that will be available after retraining.

After Completing all these steps, the Outer Agent interacts exclusively with the Inner Agent for a considerable amount of time. Figure 5.18 shows the workflow of the Architecture when interacting with the User.

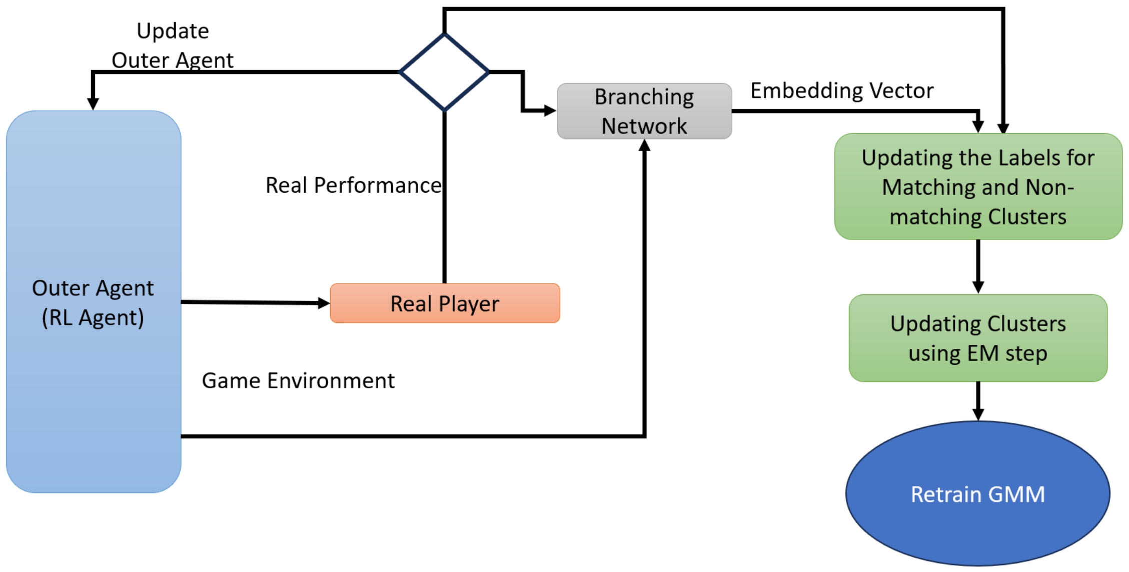


Figure 5.18: Interaction with Real Player

5.5.3 Interaction with Inner Agent

Once the Outer Agent has interacted with the Real user and trained itself with the user performance, it interacts with the Inner Agent **n** number of times. The value of **n** is set to 100. The Emotion data from the previous encounter is recorded and sent to The Outer Agent. The Outer Agent returns a Game state. A matching cluster is randomly chosen, and from that matching cluster, an embedding vector is selected. The Game state, Emotion data, and the Embedding vector are sent to the CGAN Generator as Conditional Input and simulated performance is generated. We extract the Emotion data which will be used in the next iteration. Based on the performance, whether the Inner Agent has won or lost, the Outer Agent is updated. Figure 5.19 shows the workflow where the Outer Agent trains with the Inner Agent.

After training with the Inner Agent, the Outer Agent will interact directly with the User and this complete loop will continue with Evaluation in regular intervals to check if the Outer Agent is successful in generating good Game states for the user.

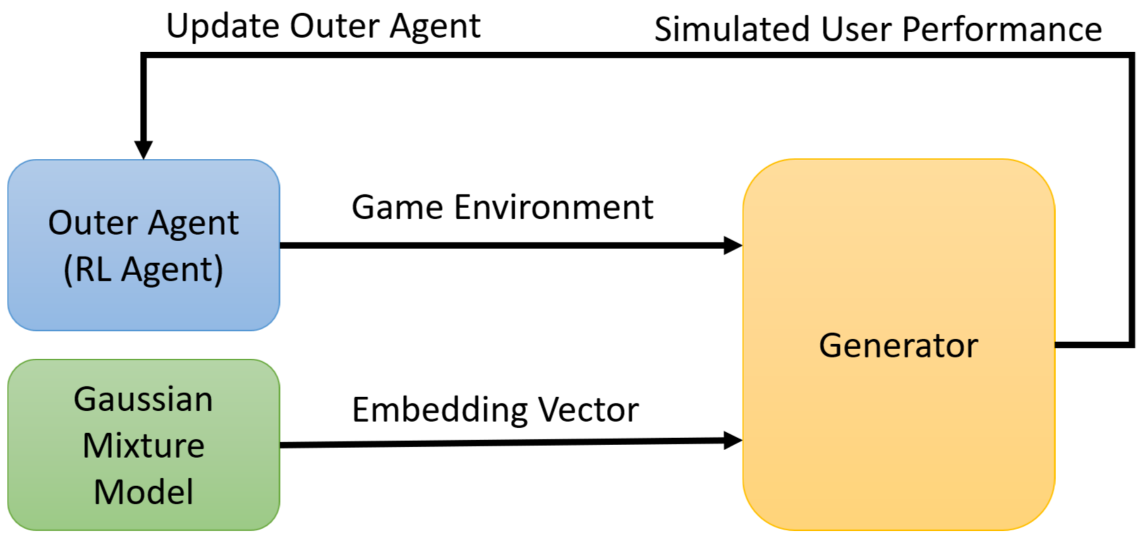


Figure 5.19: Interaction with Inner Agent

5.5.4 Evaluations

After every 3 interactions with the real user, we evaluate the Outer Agent to find out if it has learned a good policy that generates personalized Game states for the user. In the beginning, we handpick one Game state which is used to record the user performance. After that, the Outer Agent chooses the next 10 Game states based on the performance. After all the performances are recorded the average reward is calculated and stored with the Interaction number with the real player. This information is used in the Results.

5.6 Experimentation and Results

In this section, each of the components, as well as the complete approach, is tested and evaluated.

5.6.1 Inner Agent

5.6.1.1 Siamese Network

The Siamese Network was trained with both user-consistent and user-inconsistent pairs. The network ran for 500 epochs and achieved a training accuracy of 75% and validation accuracy of 76%. The training loss and validation loss is 0.4. The training curves are shown in Figure 5.20.

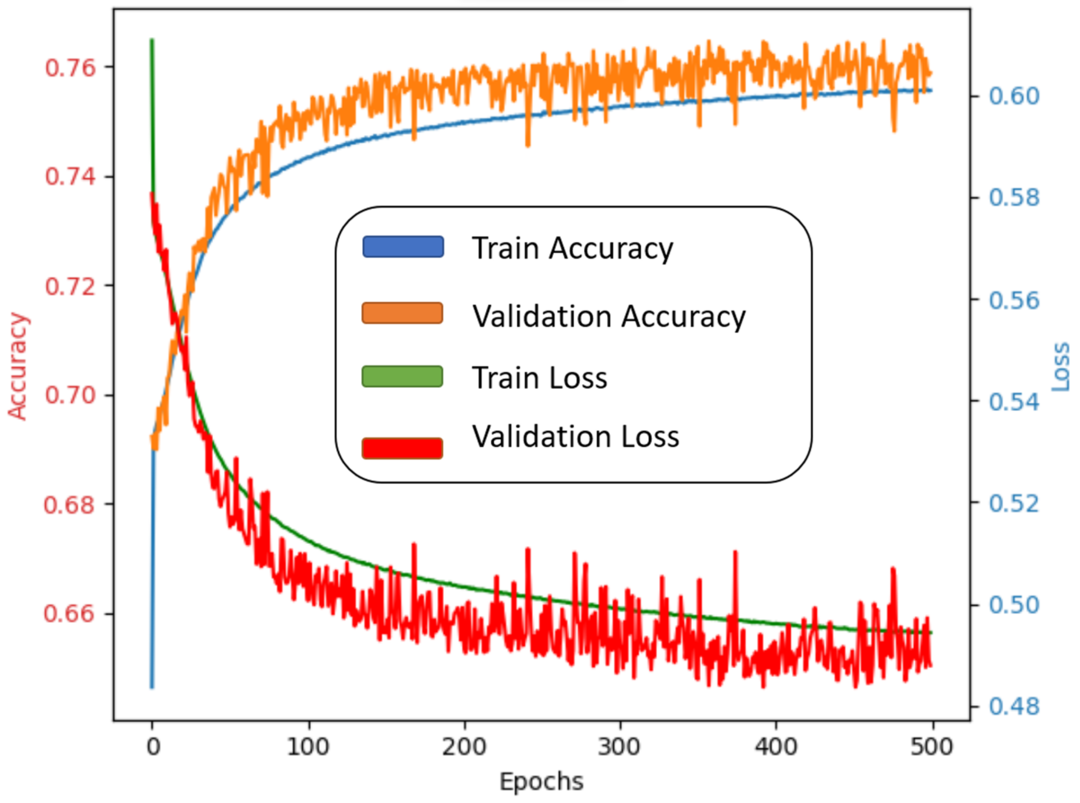


Figure 5.20: Accuracy of Siamese Network

These results show that the Siamese network is able to learn consistent embedding of the distinguishing characteristics of existing user data with good precision.

5.6.1.2 Gaussian Mixture Model

Using the embedding space and the existing user data without labels, a Gaussian Mixture Model with 4 clusters (chosen by the Silhouette score method) is trained and then customized using new user feedback. The clusters are shown in Figure 5.21. The Top left picture displays the embedding space before training and the remaining three pictures show the embedding space after training for three different users. As it can be seen to predict the user effectively, the Inner Agent found some hidden clusters which are now being displayed. The clusters in the shade of red do not align with user performance and the clusters in the shade of green align with user performance.

5.6.2 Inner Agent

To check the effectiveness of the Inner Agent in the architecture, we have to find out how many interactions the Outer Agent requires to learn the largest three performances of each agent. For this, we run two experiments. In the first experiment, the Outer Agent interacts only with the Real player. The Inner Agent is not used. In the second experiment, the Outer Agent interacts both with the real player and the Inner Agent. In both experiments, the Outer Agent interacts with the player 100 times, and in the second experiment, the Outer Agent interacts with Inner Agent 100 times after each interaction with the real player. Once the experiments are run, we find out how many interactions with the player are required to register the 3 largest performances by the Outer Agent. Figure 5.22 shows the two charts. When the Inner Agent is used, for many of the players, the Outer Agent required less interaction

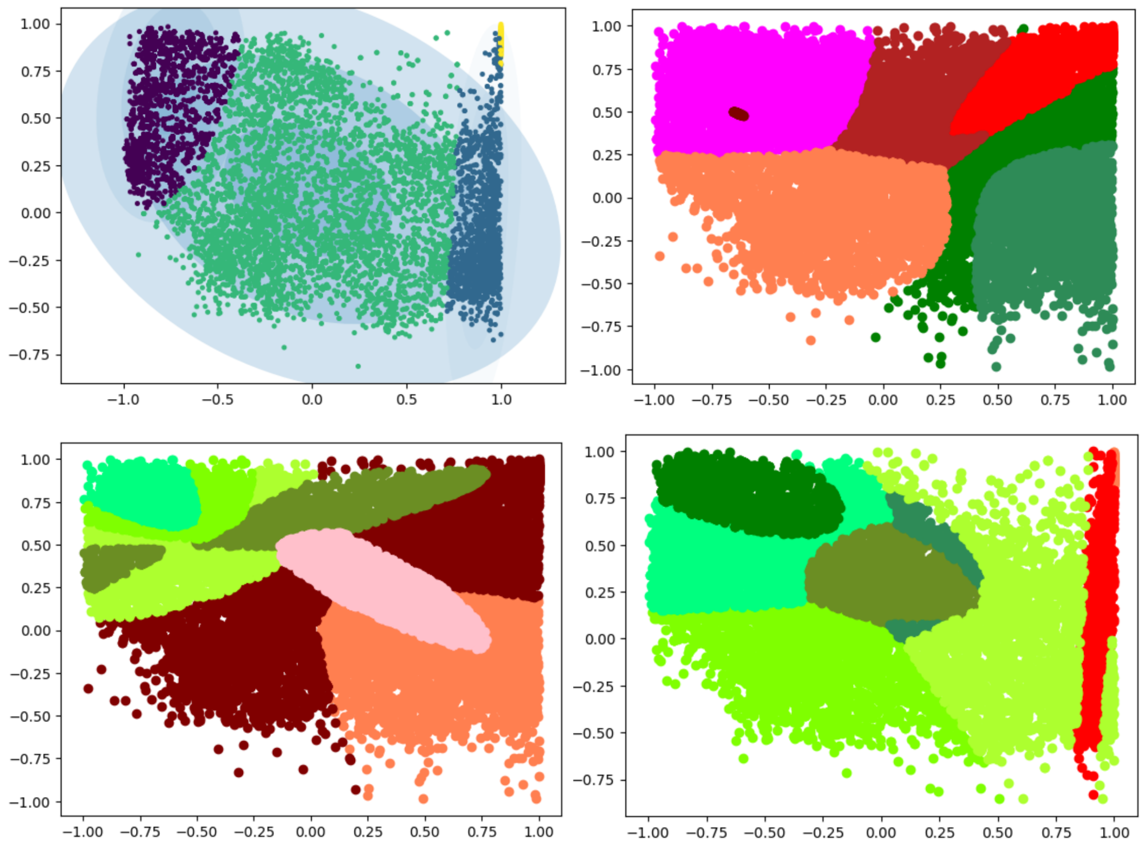


Figure 5.21: Soft Clusters in Embedding Space Before Training (Top Left) and After Training for three different Users

with the player to achieve the 3 largest performance. In comparison, the experiment without Inner Agent required more interactions. However, looking at the information also suggest that the Outer Agent has not been trained sufficiently in both cases, leading to instabilities in the top rewards and large differences in the points in time when the highest utilities are achieved. Figure 5.23 shows the number of interactions required by the Outer Agent to achieve the highest performance in both experiments. As can be seen, when the Inner Agent is used, many of the players require fewer interactions. However, again the Outer Agent does not seem to have fully converged, making it harder to interpret the results and suggesting that some changes to the Outer Agent might be required to stabilize its performance.

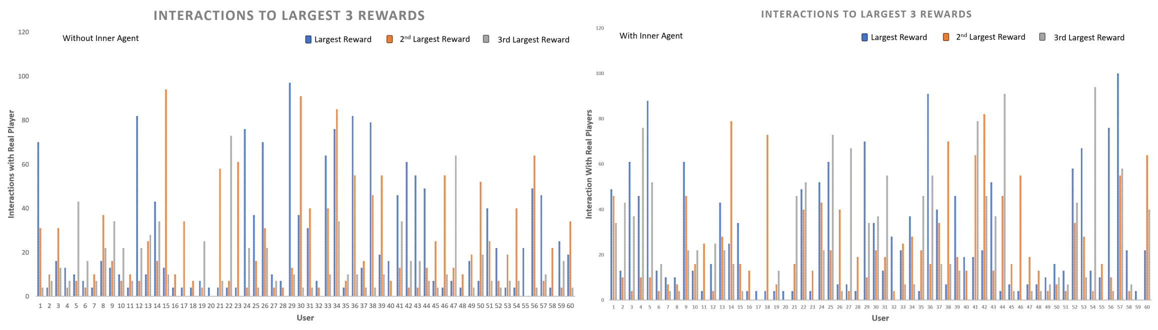


Figure 5.22: Interactions required for Largest 3 rewards without Inner Agent (Left) and With Inner Agent (Right)

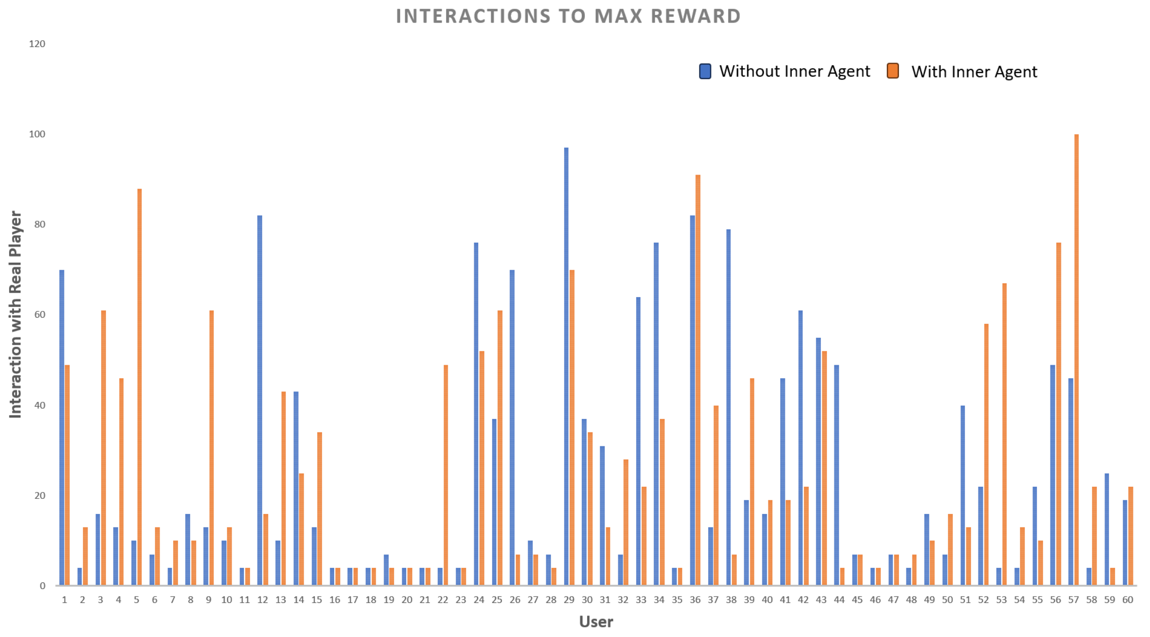


Figure 5.23: Iterations required for the Largest reward with and without Inner Agent

5.6.3 Outer Agent

Once the Outer Agent is trained for the user, We use it to see if provides better results. First, we hand-pick 5 Game states. The user plays each of them 10 times and the performance is averaged. In the end, we get 5 averaged reward values. Next, we choose the Outer Agent to create 5 Game states. The user plays each state 10 times and the performance is averaged. we then compare the reward from the performance based on Outer Agent with the rewards for the hand-picked Game states. As we can see in Figure 5.24, for most of the Users, the respective Outer Agents have learned well and produced higher rewards than when using the hand-picked game states.

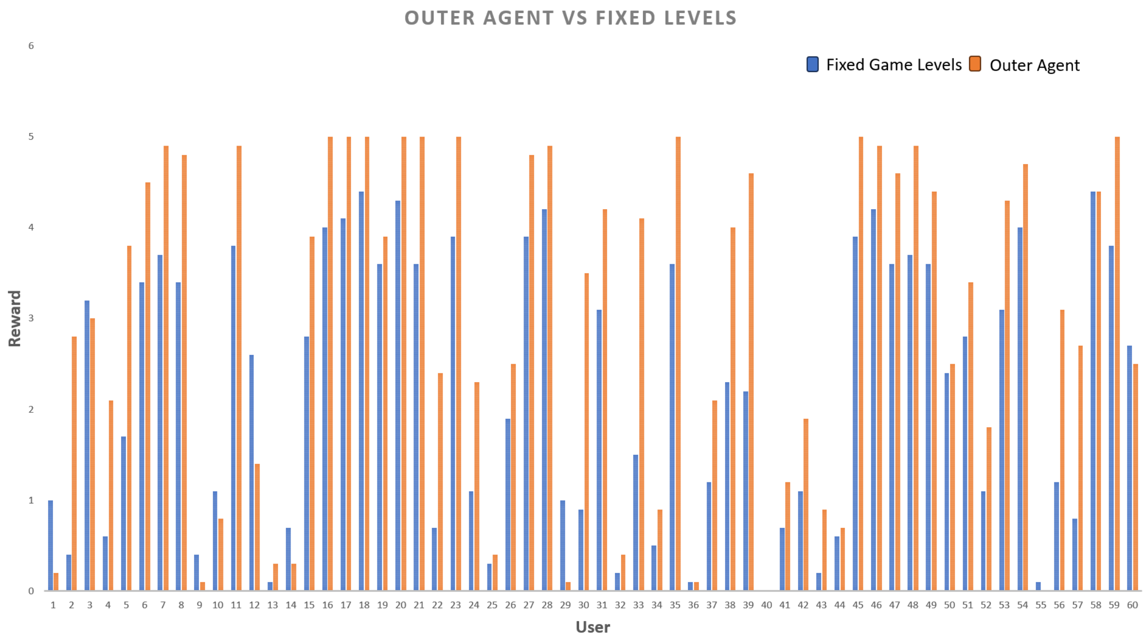


Figure 5.24: Average Performance of User in Fixed Game Levels vs Levels Generated by the Outer Agent

5.7 Conclusion

This chapter introduced an extended system architecture that has learned to adapt the domain Environment based on user preference, with minimal interaction. An Outer Agent based on Reinforcement Learning is trained to manipulate and adapt the Game environment based on user preference by providing Game states. Emotion data is also extracted from the user which helps in the training of the Outer Agent. The Outer Agent interacts with the real user at intervals while getting trained on simulated performance provided by the Inner Agent. The Inner Agent utilizes a Siamese Network which develops Embedding Vectors. A Gaussian Mixture Model is used to create soft clusters over the embedding vectors to capture the behavior of the specific user. The Embedding vectors in addition to the Game state and Emotion data are used as Conditional Input to train a CGAN that will generate simulated performances. Experimentation has shown promising results where the Outer Agent has successfully created Game states which improve user performance.

CHAPTER 6

Conclusion

Personalization without the need for massive amounts of data from a specific user is an important capability to build systems that can improve the performance of a user by adapting the task environment. This dissertation introduces an architecture to achieve this through the interaction of an Inner Agent which attempts to learn a model of the user bootstrapped by other user's data, and an Outer Agent, which attempts to learn the environment adaptation based on the user feedback using Reinforcement Learning.

The dissertation then used the Architecture in the context of various domain problems and displayed its effectiveness in solving them. In Chapter 1, we introduce the problem, the effect it has, and how the Architecture will be solving the problem.

In Chapter 2 we provide a summarised introduction to all the various Supervised, Unsupervised, and Reinforcement Learning tools used to create this Architecture.

In Chapter 3 for proof of concept, we create the Architecture to work with the E-learning domain. The task to solve is to understand the student's learning preferences and provide them with the correct course material version to improve their learning. The architecture was successful in predicting the student's learning preferences and created an E-learning system to provide the students with correct coursework based on FSLSM and Differentiated Pedagogy.

In Chapter 4, we work on creating an efficient and accurate Inner Agent. We create a predictive model which is tasked to learn the user preference and display images that the user will like and avoid disliked images. To solve this problem, the concept of collaborative filtering, Siamese Network, and Gaussian Mixture Models are included. The Inner Agent learns to map the new users as a soft cluster that can contain the preferences of multiple previous users and based on that information, it generates images the user might like or dislike. The Inner Agent has been successful in learning about individual users' likenesses with minimal interaction.

Finally, in Chapter 5, the new Inner Agent is introduced in the Architecture and it is tasked to train an Outer Agent which based on the player's preference generates interesting Game states. To solve this problem Facial Emotion Recognition algorithm is used which generates Emotion data to train the Outer agent. The Outer agent interacts with the real player and trains itself. After this, the inner Agent learns about the real player and generates a behavior model mimicking the real player. This behavior model is then used extensively to train the Outer Agent with intervals where the Outer Agent interacts with the real player. The architecture effectively learns about the player's preferences and uses that knowledge to train the Outer Agent to generate game levels specially created for the player.

Through the experiments in these domains, we have shown how effectively the Architecture can learn and perform in various domains. This displays the power of Reinforcement Learning, Conditional GANs, Gaussian Mixture Models, and Siamese Networks when used in unison to learn to adapt the environment to user preference and also to generate user behavior models.

Bibliography

Bakkes, Sander, Chek Tien Tan, and Yusuf Pisan (2012). “Personalised gaming: a motivation and overview of literature”. In: *Proceedings of the 8th Australasian Conference on Interactive Entertainment: Playing the System*, pp. 1–10.

Bakkes, Sander CJ, Pieter HM Spronck, and Giel van Lankveld (2012). “Player behavioural modelling for video games”. In: *Entertainment Computing* 3.3, pp. 71–79.

Bloom, Benjamin S et al. (1956). “Handbook I: cognitive domain”. In: *New York: David McKay.*

Bontchev, Boyan and Dessislava Vassilev (Mar. 14, 2012). “Courseware Adaptation to Learning Styles and Knowledge Level”. In: ed. by Sergio Kofuji. DOI: 10.5772/29340. URL: http://www.intechopen.com/books/e-learning-engineering-on-job-training-and-interactive-teaching/courseware-adaptation-to-learning-styles-and-knowledge-level (visited on 09/22/2021).

Bromley, Jane et al. (1993). “Signature verification using a” siamese” time delay neural network”. In: *Advances in neural information processing systems* 6.

Brown, John Seely (2006). “New learning environments for the 21st century: Exploring the edge”. In: *Change: The magazine of higher learning* 38.5, pp. 18–24.

Carchiolo, Vincenza et al. (2007). “An Architecture to Support Adaptive E-Learning”. In: p. 13.

Carver, Curtis A, Richard A Howard, and William D Lane (1999). “Enhancing student learning through hypermedia courseware and incorporation of student learning styles”. In: *IEEE transactions on Education* 42.1, pp. 33–38.

Centre for Teaching Excellence, University of Waterloo (n.d.). *Self-Directed Learning: A Four-Step Process.*

CSE 2001 dans Caron, 2003 (n.d.). *Conseil Suprieur de lducation,*

Cui, Zhihua et al. (2020). "Personalized Recommendation System Based on Collaborative Filtering for IoT Scenarios". In: *IEEE Transactions on Services Computing* 13.4, pp. 685–695. DOI: 10.1109/TSC.2020.2964552.

De Silva, Liyanage C, Tsutomu Miyasato, and Ryohei Nakatsu (1997). "Facial emotion recognition using multi-modal information". In: *Proceedings of ICICS, 1997 International Conference on Information, Communications and Signal Processing. Theme: Trends in Information Systems Engineering and Wireless Multimedia Communications (Cat.* Vol. 1. IEEE, pp. 397–401.

Erev, Ido and Alvin E Roth (1998). "Predicting how people play games: Reinforcement learning in experimental games with unique, mixed strategy equilibria". In: *American economic review*, pp. 848–881.

Eyssautier, Carole et al. (n.d.). "A model of learners profiles management process". In: (), p. 9.

Felder, Richard M (2002). "Learning and teaching styles in engineering education". In:

— (n.d.). "LEARNING AND TEACHING STYLES IN ENGINEERING EDUCATION". In: (), p. 11.

Geman, Stuart and Donald Geman (1984). "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-6.6, pp. 721–741. DOI: 10.1109/TPAMI.1984.4767596.

Goldberg, David et al. (1992). "Using collaborative filtering to weave an information tapestry". In: *Communications of the ACM* 35.12, pp. 61–70.

Goodfellow, Ian J. et al. (June 10, 2014). *Generative Adversarial Networks*. arXiv: `1406.2661[cs,stat]`. URL: `http://arxiv.org/abs/1406.2661` (visited on 02/10/2023).

Graf Sabine, Kinshuk (n.d.). "Providing Adaptive Courses in Learning Management Systems with respect to Learning Styles*". In: (), p. 8.

Graf, Sabine et al. (Sept. 2007). "In-Depth Analysis of the Felder-Silverman Learning Style Dimensions". In: *Journal of Research on Technology in Education* 40.1, pp. 79–93. ISSN: 1539-1523, 1945-0818. DOI: `10.1080/15391523.2007.10782498`. URL: `http://www.tandfonline.com/doi/abs/10.1080/15391523.2007.10782498` (visited on 09/22/2021).

Guerrero-Roldán, Ana-Elena and Julià Minguillón Alfonso (n.d.). "Adaptive learning paths for improving lifelong learning experiences". In: (), p. 8.

Harrison, Brent and David L Roberts (2011). "Using sequential observations to model and predict player behavior". In: *Proceedings of the 6th International Conference on Foundations of Digital Games*, pp. 91–98.

Honey, Peter and Alan Mumford (1986). *Using your learning styles*. Chartered Institute of Personnel and Development.

Koch, Gregory, Richard Zemel, Ruslan Salakhutdinov, et al. (2015). "Siamese neural networks for one-shot image recognition". In: *ICML deep learning workshop*. Vol. 2. Lille.

Kolb, David A (2014). *Experiential learning: Experience as the source of learning and development*. FT press.

Koren, Yehuda, Robert Bell, and Chris Volinsky (2009). "Matrix factorization techniques for recommender systems". In: *Computer* 42.8, pp. 30–37.

L'Allier, James J (1997). "Frame of reference: NETg's map to the products, their structure and core beliefs". In: *Research and Development–NETg. Retrieved June* 25, p. 2013.

Latorre, Miguel et al. (2009). "An experience on Learning Objects Reutilization based on Educational Resources Developed". In: *2009 Annual Conference & Exposition*, pp. 14–191.

Mahlmann, Tobias et al. (2010). "Predicting player behavior in Tomb Raider: Underworld". In: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pp. 178–185. DOI: 10.1109/ITW.2010.5593355.

Meiriu, Philippe (1999). *Apprendre... oui, mais comment, Paris, ESF éditeur.*

Mirza, Mehdi and Simon Osindero (Nov. 6, 2014). "Conditional Generative Adversarial Nets". In: arXiv:1411.1784. arXiv. arXiv: 1411.1784[cs,stat]. URL: http://arxiv.org/abs/1411.1784 (visited on 02/10/2023).

Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pp. 529–533.

Nabila Bousbia Issam Rebaï, Jean-marc Labat (Dec. 15, 2010). "Analysing the relationship between learning styles and navigation behaviour in web-based educational system". In: *Knowledge Management & E-Learning: An International Journal*, pp. 400–421. ISSN: 20737904. DOI: 10.34105/j.kmel.2010.02.029. URL: http://www.kmel-journal.org/ojs/index.php/online-publication/article/view/92 (visited on 09/22/2021).

Papanikolaou, Kyparisia A. et al. (Aug. 1, 2003). "Personalizing the Interaction in a Web-based Educational Hypermedia System: the case of INSPIRE". In: *User Modeling and User-Adapted Interaction* 13.3, pp. 213–267. ISSN: 1573-1391. DOI: 10.1023/A:1024746731130. URL: https://doi.org/10.1023/A:1024746731130 (visited on 07/05/2023).

Petrosyan, Ani (2023). *Worldwide digital population 2023.* URL: `https://tinyurl.com/4fht5679`.

Popescu, Elvira, Costin Badica, and Philippe Trigano (2008). “Description and organization of instructional resources in an adaptive educational system focused on learning styles”. In: *Advances in Intelligent and Distributed Computing.* Springer, pp. 177–186.

Przesmycki, Halina (2003). “la pdagogie diffrencie, 2004”. In: *Hachette les ingalits et la diffrenciation de l’enseignement, Facult de psychologie et des sciences de l’ducation, Universit de Genve.*

Puterman, Martin L (1990). “Markov decision processes”. In: *Handbooks in operations research and management science* 2, pp. 331–434.

Rasmussen, Carl (1999). “The infinite Gaussian mixture model”. In: *Advances in neural information processing systems* 12.

Resnick, Paul et al. (1994). “Grouplens: An open architecture for collaborative filtering of netnews”. In: *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pp. 175–186.

Richard M Felder, L.K. Silverman (n.d.). “LEARNING AND TEACHING STYLES IN ENGINEERING EDUCATION”. In: (), p. 11.

Richard S Sutton, Andrew G Barto (2018). *Reinforcement Learning: An introduction. MIT press.*

Russ, Coding with (Dec. 2020). URL: `https://www.youtube.com/watch?v=0ngc4EVqRjo&amp;list=PLjcN1EyupaQnHM1I9SmiXfbT6aG4ezUvu`.

Sangineto, Enver et al. (2008). “Adaptive course generation through learning styles representation”. In: *Universal Access in the Information Society* 7, pp. 1–23.

Schiaffino, Silvia, Patricio Garcia, and Analia Amandi (Dec. 2008). “eTeacher: Providing personalized assistance to e-learning students”. In: *Computers & Ed-*

ucation 51.4, pp. 1744–1754. ISSN: 03601315. DOI: 10.1016/j.compedu.2008.05.008. URL: https://linkinghub.elsevier.com/retrieve/pii/S036013150800078X (visited on 09/22/2021).

Schulman, John, Sergey Levine, et al. (2015). “Trust region policy optimization”. In: *International conference on machine learning*. PMLR, pp. 1889–1897.

Schulman, John, Filip Wolski, et al. (2017). “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347*.

Sheng, Taoran and Manfred Huber (2019). “Siamese networks for weakly supervised human activity recognition”. In: *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, pp. 4069–4075.

Shenk, Justin et al. (Sept. 2021). *justinshenk/fer: Zenodo*. Version zenodo. DOI: 10.5281/zenodo.5362356. URL: https://doi.org/10.5281/zenodo.5362356.

Soo, Sander (2014). “Object detection using Haar-cascade Classifier”. In: *Institute of Computer Science, University of Tartu* 2.3, pp. 1–12.

Sun, Yi, Xiaogang Wang, and Xiaoou Tang (2013). “Deep convolutional network cascade for facial point detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3476–3483.

Sutton, Richard S (1988). “Learning to predict by the methods of temporal differences”. In: *Machine learning* 3, pp. 9–44.

Sutton, Richard S et al. (1999). “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in neural information processing systems* 12.

Tzouveli, Paraskevi, Phivos Mylonas, and Stefanos Kollias (Aug. 2008). “An intelligent e-learning system based on learner profiling and learning resources adaptation”. In: *Computers & Education* 51.1, pp. 224–238. ISSN: 03601315. DOI:

10.1016/j.compedu.2007.05.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S0360131507000504 (visited on 09/22/2021).

Wang, Xiang et al. (2019). "Neural Graph Collaborative Filtering". In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR'19. Paris, France: Association for Computing Machinery, pp. 165–174. ISBN: 9781450361729. DOI: 10.1145/3331184.3331267. URL: https://doi.org/10.1145/3331184.3331267.

Watkins, Christopher JCH and Peter Dayan (1992). "Q-learning". In: *Machine learning* 8, pp. 279–292.

Wiley, David (2003). *Learning objects: Difficulties and opportunities.*

Williams, Ronald J (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8, pp. 229–256.

Wu, Haoran et al. (2017). "Face recognition based on convolution siamese networks". In: *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. IEEE, pp. 1–5.

Wu, Yuhuai et al. (2017). "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation". In: *Advances in neural information processing systems* 30.

Xuan, Guorong, Wei Zhang, and Peiqi Chai (2001). "EM algorithms of Gaussian mixture model and hidden Markov model". In: *Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205)*. Vol. 1, 145–148 vol.1. DOI: 10.1109/ICIP.2001.958974.

Zalandor (2017). *Fashion-Mnist Dataset*. URL: https://github.com/zalandoresearch/fashion-mnist.

Zaoui Seghroucheni, Yassine, M. AL Achhab, and B. E. El mohajir (May 12, 2015). "A Bayesian Network Based on the Differentiated Pedagogy to Generate Learning

Object According to FSLSM". In: *International Journal of Recent Contributions from Engineering, Science & IT (iJES)* 3.2, p. 21. ISSN: 2197-8581. DOI: 10.3991/ijes.v3i2.4363. URL: https://online-journals.org/index.php/i-jes/article/view/4363 (visited on 09/22/2021).

Zaoui Seghroucheni, Yassine, Mohammed AL Achhab, and Badr Eddin El Mohajir (Sept. 14, 2014). "An Approach to Create Multiple Versions of the Same Learning Object". In: *International Journal of Emerging Technologies in Learning (iJET)* 9.5, p. 17. ISSN: 1863-0383. DOI: 10.3991/ijet.v9i5.3762. URL: https://online-journals.org/index.php/i-jet/article/view/3762 (visited on 09/22/2021).

BIOGRAPHICAL STATEMENT

Subharag Sarkar was born in Batanager, West Bengal, India, in October 1992. He received his BTech degree from West Bengal University of Technology, India, in 2015, and his Ph.D. degree from the University of Texas at Arlington in 2023, respectively all in Computer Science and Engineering. From 2016 to 2023, he was with the University of Texas at Arlington as Teaching Assistant and then as Assistant Instructor. His current research interest is in the area of predicting User preferences for varied domains. He is a member of several IEEE societies.