A Student Performance Prediction Using RNNs Models with variety of optimization Techniques in Deep Learning

Abdelmajid El Hajoui*, Otmane Yazidi Alaoui, Omar El Kharki, Miriam Wahbi, Hakim Boulaassal, and Mustapha Maatouk

Laboratoire de Recherche et Developpement en GeoScience Appliquées, FSTT, Abdelmalek Essaadi University, Tetouan, Morocco
Email: elhajoui.abdelmajid@etu.uae.ac.ma (A.E.H.); o.yalaoui@uae.ac.ma (O.Y.A.); elkharki@gmail.com (O.E.K.); mwahbi@uae.ac.ma (M.W.); h.boulaassal@uae.ac.ma (H.B.); mmaatouk@uae.ac.ma(M.M.)
*Corresponding author
Manuscript received January 28, 2025; revised February 25, 2025; accepted March 19, 2025; published June 20, 2025

***Abstract*—The goal of learning analytics is to assess students' performance over time. While virtual learning environments enable educators to intervene quickly, distance can make it challenging to evaluate students' success. Many studies have developed prediction models using data from Massive Open Online Courses (MOOCs), but these models were limited to classifying students into binary groups based on the courses they had completed. The paper tackles an important gap in predicting student performance by introducing a daily multi-class model based on Recurrent Neural Networks (RNNs), specifically leveraging Gated Recurrent Units (GRUs) and Long Short-Term Memory (LSTM) networks. To validate the GRU model, it is compared against two baseline models: Artificial Neural Networks (ANNs) and LSTM networks. The results show that the GRU model achieves an impressive accuracy of nearly 90%, outperforming the LSTM model, which reaches 88% accuracy. This highlights the potential of GRUs to better capture temporal dependencies and patterns in student performance data, making them a strong candidate for educational forecasting. The study suggests that GRU-based models could serve as a powerful tool for educators and institutions to predict and address student performance issues proactively. This demonstrates how early student performance in MOOCs can be predicted using the design and latent dependency maintenance capabilities of the GRU time series model. Along with this, the paper evaluates the accuracy, loss, and training time of several stochastic gradient descent algorithms, such as Adam, AdaGrad, RMSProp, and Stochastic Gradient Descent (SGD) with momentum. It also assesses the loss that each algorithm must endure in order to produce an optimal solution. When comparing all optimizers utilizing the pre-trained GRU model, the highest accuracies of 97.58% and 97.66% are obtained by Adam and SGD with momentum, respectively.**

***Keywords*—Recurrent Neural Network (RNN), Gated Recurrent Unit (GRU), Artificial Neural Network- Long Short-Term Memory (ANN-LSTM), Massive Open Online Course (MOOC), students performance, Stochastic Gradient Descent (SGD), RMSprop, Adam, Adagrad**

I. Introduction

The COVID-19 pandemic has underscored the importance of Virtual Learning Environments (VLEs), as students worldwide were compelled to transition to remote learning. This shift has accelerated the adoption and reliance on digital educational platforms. At the same time, advancements in science and technology have driven significant progress in educational tools over the past few decades, enabling more interactive, accessible, and personalized learning experiences. Together, these developments have reshaped the educational landscape, emphasizing the critical role of technology in supporting modern learning environments. Massive Open Online Courses (MOOCs) and similar VLEs have gained traction, especially during the pandemic. These platforms offer a variety of resources, including lecture videos, online assessments, discussion forums, and live video sessions over the internet. Early insights into student performance within VLEs at the beginning of courses facilitate the assessment of learning analytics objectives. As a result, researchers have developed predictive models to forecast student outcomes in MOOCs, with a primary focus on binary classifications such as pass/fail or dropout scenarios. However, there has been limited exploration of models that predict performance using multi-classification approaches. Addressing this gap, the authors in reference [1] propose a novel method that combines Variable Reduction with an Optimized Deep Recurrent Neural Network (VR-ODRNN) to analyze student performance and predict course outcomes. The VR-ODRNN model is specifically designed to process student data and accurately predict course grades, offering a more nuanced and detailed approach compared to traditional binary classification models. This innovative framework represents a significant step forward in understanding and forecasting student performance in MOOCs. Input data is first normalized using min-max normalization to ensure consistency. For feature selection, the Coot Optimization Algorithm (COA) is employed to identify an optimal subset of variables. Student performance is then predicted using a Deep Recurrent Neural Network (DRNN), with its hyperparameters optimized through the Dwarf Mongoose Gannet Optimization Algorithm (DMGOA). The proposed VR-ODRNN model is evaluated using a student performance dataset from the Kaggle repository. Experimental results demonstrate that the VR-ODRNN model achieves a high accuracy. Several studies have investigated the use of various machine learning techniques to predict students' academic performance. As highlighted in [2], Principal Component Analysis (PCA) was first utilized to reduce the dimensionality of the dataset, improving its visualization. Following this, K-Means clustering was applied to categorize students according to their learning patterns. The resulting clusters were then used to train classification models, allowing for customized predictions for each group of students. This approach was validated using the Open University Learning Analytics Dataset (OULAD) and data from an undergraduate science course at a North American University (NAU) [2].

As a result, there is a need for further research to improve the performance of multi-class prediction models [3]. While traditional artificial intelligence techniques are frequently

employed to predict student achievement in online higher education, more advanced methods such as deep learning remain underutilized [4]. The main objective of this study is to develop a multi-class, course-agnostic, day-wise predictive model using activity clickstream data and demographic information. This model aims to classify students' performance in MOOC environments as early as possible while maintaining satisfactory accuracy. Such a model can support higher education decision-making processes, promote sustainable education in MOOC settings, and enable instructors to take timely and appropriate actions to support student success.

Deep Neural Networks (DNNs) have established themselves as highly effective tools across a variety of domains, particularly for processing and analyzing large-scale datasets. By training on vast amounts of data, DNNs can uncover complex patterns and relationships between variables. To enhance their performance, optimization algorithms such as Stochastic Gradient Descent (SGD), RMSprop, Adam, and Adagrad are commonly employed in various applications. However, despite their strengths, these methods often encounter difficulties with generalization, especially when dealing with large datasets, which can hinder their ability to deliver consistent and reliable results.

The success of this study hinges on achieving the following objectives:

1) To develop an innovative deep learning model that makes daily predictions about student performance in a multi-class setting.
2) To identify which of the three recurrent deep learning models LSTM, and GRU can improve the precision of student performance forecasts.
3) To evaluate the accuracy of the proposed model by comparing it with relevant state-of-the-art models.
4) To assess different stochastic gradient optimization techniques for deep learning and identify a method that combines the rapid convergence of adaptive techniques with the robust generalization performance of traditional Stochastic Gradient Descent (SGD). Additionally, this study proposes optimization strategies that incorporate parameter adjustments to enhance performance.

The reste of the article is organized as follows: Section II reviews the relevant literature on the use of deep learning models for predicting student performance. Section III describes the methodology employed in this study. Section IV presents the results and provides a discussion. Finally, Section V offers conclusions and perspectives based on the findings.

II. Related Works

Massive Open Online Courses, commonly known as MOOCs, are freely accessible online courses available to everyone. They are structured around specific learning objectives across various fields of study and typically run over a set period within a digital learning environment that encourages interaction among peers and instructors. MOOCs foster the development of a learning community. Many research studies have employed performance prediction models in MOOC courses, with a significant number utilizing the OULAD dataset, which is widely recognized in the MOOC domain. Throughout the years 2021, 2022, 2023, and 2024, numerous prediction models trained using the OULAD dataset have been published [5].

RNNs (Recurrent Neural Networks) are supervised neural networks that preserve the temporal dimension of time series data by using a recurrent hidden state, which is updated by sequential information obtained from the input time series data. The output of a sequence depends on this hidden state, meaning the current output of a sequence is linked to the previous output (recurrent neural network. They rely on the repeated use of classifiers, which avoids the need for a large number of historical classifiers [6].

Recurrent Neural Networks (RNNs) can be used to predict student performance by analyzing sequential data, such as time-series data of student activities, grades, or engagement metrics. RNNs are particularly well-suited for this task because they can capture temporal dependencies and patterns in the data, which are often critical for understanding student behavior and performance over time [6].

In 2021, Adnan et al. utilized a Deep Feed Forward Neural Network (DFFNN) to predict students' final performance status, categorizing them as fail, pass, withdrawn, or distinct. They observed varying levels of accuracy based on the input data used. Specifically, when only demographic data was considered, the average accuracy was 43% by the end of the course. However, this accuracy increased to 63% when demographic and clickstream data were combined, and further improved to 71% when assessment data was also included. Notably, incorporating all features from the OULAD dataset in the input data led to a significant accuracy of 72%. Additionally, the researchers employed binaryization to facilitate multi-classification grouping. Remarkably, after conversion, they achieved an accuracy of 90% at the conclusion of the courses [4] .

Another study, referenced as [7], focused on utilizing demographic information and aggregated clickstream data from two STEM courses (CCC_2014B and CCC_2014J), as well as two social science courses (AAA_2013J and AAA_2014J).

The study aimed to predict whether a student would persist or drop out in the following week. To achieve this, a supervised machine learning model was developed using an expectation-maximization approach. The model's prediction accuracy for the current week was subsequently enhanced by incorporating the resulting probabilities. Additionally, Reference [8] utilized the Synthetic Minority Over-Sampling (SMOTE) technique. By the end of the courses, the average accuracy across all selected courses for all weeks approached 88%.

In 2022, as described in [9], a prediction framework was developed. This framework incorporates six classical machine learning algorithms: SVC (R), SVC (L), Naïve Bayes, KNN (U), KNN (D), and Softmax. The predictor aimed to classify outcomes as either "qualified" or "unqualified." The framework utilized the "DDD" module (course) along with clickstream data across 12 activities in its implementation.

In Reference [10], researchers created a basic prediction model using a Bayesian Network (BN) and compared it with ensemble models, including Naïve BN, Multi-Layer Perceptron (MLP), and Gradient Boosting Decision Trees, to predict final performance. The performance value, based on

student assessment results, was categorized into four groups: fail, pass, good, and distinction.

Adnan et al. created several machine learning models within a multi-class classification framework to forecast student performance across four categories: distinct, pass, fail, and withdrawn. These models featured a single deep learning classifier, specifically a Feed Forward Neural Network (FFNN), and were assessed using criteria such as Gini and entropy. The evaluation involved various classifiers, including Random Forest, AdaBoost, Extra Tree classifier, K-Nearest Neighbor (KNN), Decision Tree, Support Vector Machine (SVM), Gradient Boosting, Logistic Regression, Gaussian NB, and Bernoulli NB, as detailed in [3].

In 2023, a team proposed the ANN-LSTM model, which integrates an artificial neural network with long short-term memory, to predict students' performance. They investigated and compared this model with two baseline models, the Gated Recurrent Unit (GRU) and the Recurrent Neural Network (RNN), to evaluate its effectiveness in this specific context. Additionally, they compared the accuracy of ANN-LSTM with that of the most advanced models available. The results demonstrated that the ANN-LSTM model outperformed the baseline models. By the end of the third month of the course, ANN-LSTM achieved an accuracy of nearly 70%, surpassing the accuracy of the RNN and GRU models, which were 53% and 57%, respectively [11].

In [12] , the study explores a wide range of machine learning algorithms using two data representations. It demonstrates that algorithms relying solely on assignment information and a representation based on mean information entropy (MIL) can outperform single instance learning representations by more than 20% in terms of accuracy. This suggests that using an appropriate representation to mitigate data sparsity highlights the importance of a factor—such as completed assignments—that has not been extensively utilized to date for forecasting students' academic achievement. Furthermore, a comparison with earlier research on the same dataset and problem shows that predictive models based on MIL, which use only assignment data, produce competitive outcomes compared to earlier studies that incorporate other predictor variables to predict student performance.

Several machine learning techniques were employed in this study to forecast students performance. The analysis was conducted using educational data from the Open University (OU), based on performance, engagement, and demographic criteria. Through experimental analysis, it was found that among all the compared methods on the OU dataset, the k-NN approach performed best in some circumstances, while the ANN approach excelled in others [13].

In [14] , the momentum method is used to develop a dynamic adjustment scheme that incorporates both stale penalty and stale compensation. Stale compensation aims to mitigate the adverse effects of stale gradients, while stale penalty reduces the reliance on these gradients. This research introduces a dynamic asynchronous stochastic gradient descent method (DASGD) based on this adjustment approach. DASGD dynamically modifies the compensation and penalty factors using stale size. Under certain mild assumptions, the study demonstrates that DASGD is convergent. Additionally, the paper proposes evaluating DASGD on a real distributed training cluster using the Cifar10 and ImageNet datasets. Experimental results confirm the superior performance of DASGD compared to four state-of-the-art baselines. Specifically, DASGD achieves nearly the same test accuracy as SGD on Cifar10 and ImageNet, while using only around 27.6% and 40.8% of the training time required by SGD, respectively.

In [15] , the authors developed an ensemble method based on stochastic gradient descent with warm restarts (SGDRE) to address issues related to generalization, dataset size, and time complexity. This method leverages the generalization capabilities of ensemble techniques and the SGD with warm restarts mechanism, which generates a diverse set of classifiers required for the ensemble in a single training process. This approach takes the same amount of time to train as a single CNN classification model. The SGDRE algorithm was evaluated using a 10-fold cross-validation technique after being trained on a publicly available dataset of pediatric chest X-ray images. According to the experimental data, SGDRE significantly outperforms the two baseline approaches under comparison. The proposed method proved to be a highly competitive classification technique, achieving a test accuracy of 96.26% and an AUC of 95.15%.

III. Materials and Methods

This section will describe the operation of the system, utilizing the most advanced deep learning and artificial intelligence techniques that are relevant to and compatible with predicting student performance.

A. Dataset Collection

One of the publicly available learning analytics datasets is the Open University Learning Analytics Dataset (OULAD), which includes information on 32,593 students who enrolled in open online courses. The data for this study was provided by the Open University, one of the largest distance learning organizations in the UK [11]. The dataset contains over 10 million student interaction records, encompassing clickstream data, course information, assessment results, and demographic details. The OULAD dataset covers seven online courses offered over four semesters, catering to students of all ages and educational backgrounds [16].

The Open University, UK, provided the Open University Learning Analytics Dataset (OULAD), which is freely accessible. The dataset is organized into seven tables, each containing student-centered data such as demographics, interactions with the Virtual Learning Environment (VLE), exam results, course registration, and course offerings. These tables are interconnected through key identifiers. Clickstream data, which represents the total number of clicks and reflects students' daily activities and interactions within the VLE, is stored in the student VLE table. The student-module presentation dataset triplet contains the assessment results of the students. OULAD was created for the academic years 2013 and 2014, encompassing 32,593 registered students across 7 courses and 22 module presentations. The dataset has been certified by the Open Data Institute (http://theodi.org/) and is publicly available at https://analyse.kmi.open.ac.uk/open_dataset [16].

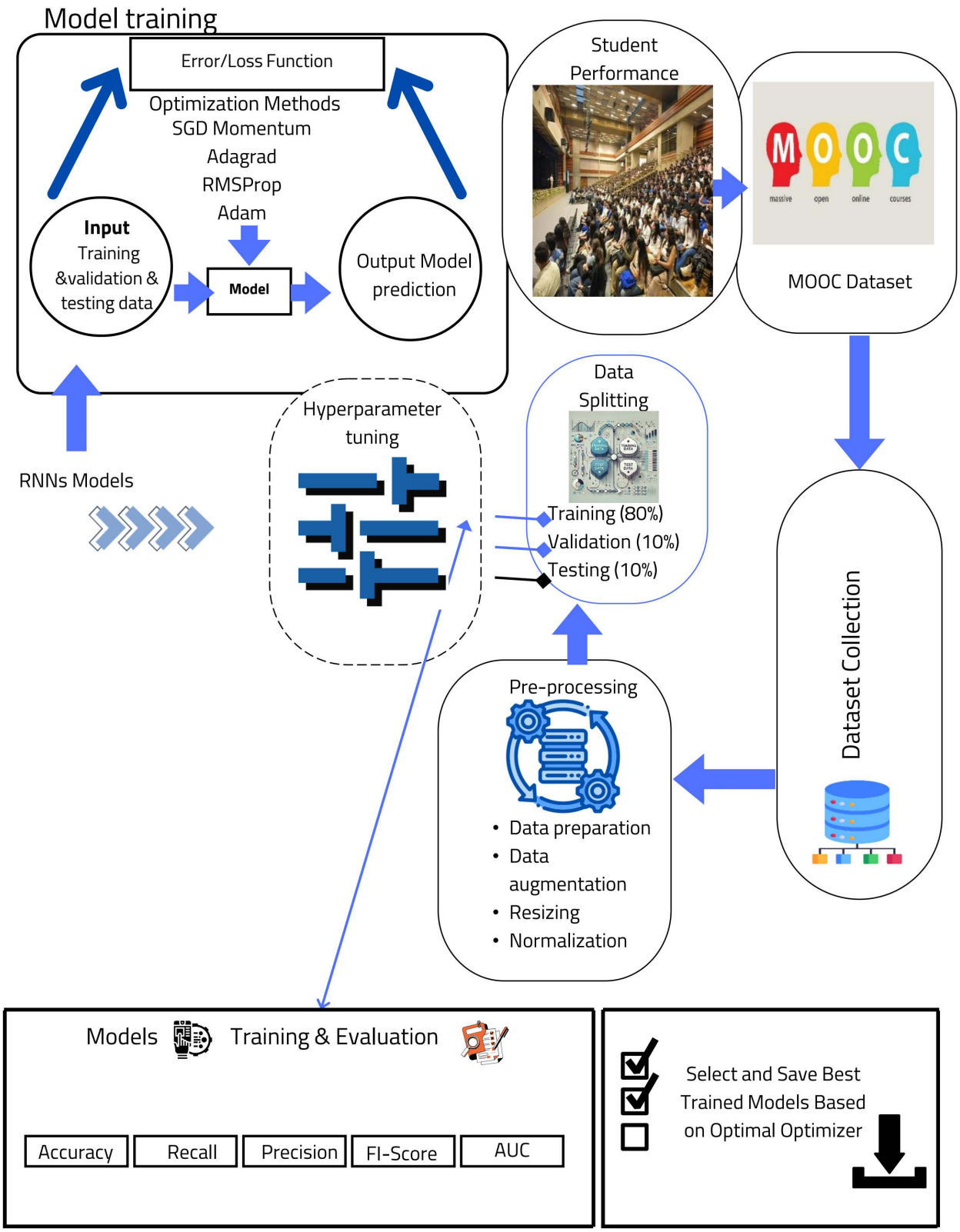

Fig. 1. The proposed system to predict student performance.

B. *Background on GRU and LSTM Model*

Recurrent Neural Networks (RNNs) are supervised neural networks designed to preserve the temporal dimension of sequential data. They achieve this by employing a recurrent cache that is updated with sequential information obtained from temporally organized input data series. The current output of a sequence depends on this hidden state, and as illustrated in Fig. 1, a sequence's previous output (a neural recurrent reservoir) is linked to its current output. RNNs rely on the repeated use of classifiers, eliminating the need for a large number of historical classifiers. Unlike traditional neural networks, where cached neurons are not connected, RNNs feature interconnected cached neurons. This characteristic makes RNNs a more effective method for solving traffic flow prediction problems), as shown in Fig. 2 [17].

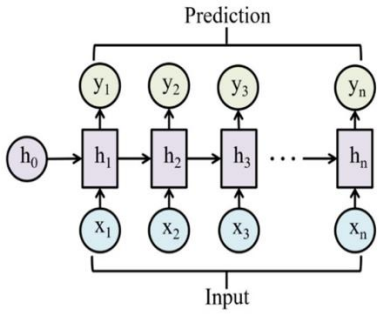

Fig. 2. The structure of RNN [17].

LSTM networks was develped to solve the vanishing gradient issue that basic RNNs have. The use of gating techniques to regulate the information flow across the network is the main breakthrough of LSTM [18].

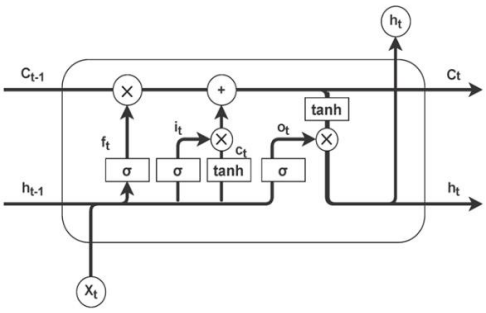

Fig. 3. The structure of LSTM [18].

Two well-known models that RNNs use to solve the traffic flow prediction problem are the Long Short-Term Memory model (LSTM) and the Gated Recurrent Unit model (GRU) [19].

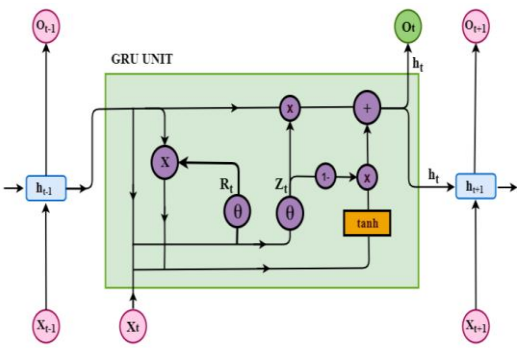

Fig. 4. The structure of GRU [20].

Cho *et al*. in [18] developed the Gated Recurrent Unit (GRU) cell, which features a more compact structure compared to the Long Short-Term Memory (LSTM) model. Unlike LSTM's multi-gate structure), as shown in Fig. 3, GRU has fewer control gates and a single cell state, resulting in lower computational complexity and a lower cutoff point. Additionally, GRU exhibits a similar ability to LSTM in addressing common RNN issues such as gradient explosion and vanishing. The two control gates in the GRU, as shown in Fig. 4 are:

1) The reset gate that determines if we need to combine the current state with the previous state.
2) The update portal, which determines how much state-related data were provided at the previous moment [20].

B. *Optimisers Algorithms*

Optimizers are essential algorithms in deep learning designed to adjust a model's parameters during training to minimize a specified loss function. These tools enhance the learning process of neural networks by iteratively updating weights and biases based on feedback from the data. Popular optimizers such as Stochastic Gradient Descent (SGD), Adam, and RMSprop each employ unique update rules, learning rates, and momentum strategies. Their primary goal is to identify and converge toward optimal model parameters, ultimately improving the model's performance [21].

Optimizer algorithms are optimization techniques designed to improve the performance of deep learning models. These algorithms, often referred to as optimizers, play a critical role in determining both the accuracy and efficiency of the training process. However, before delving into their impact, it is important to first grasp the fundamental concept of what an optimizer is and how it functions [21].

During the training of a deep learning model, optimizers adjust the weights at each epoch to minimize the loss function. At its core, an optimizer is a function or algorithm designed to fine-tune a neural network's parameters, such as weights and learning rates, with the aim of reducing the overall loss and enhancing the model's accuracy. Selecting the right weights for the model is a complex task, especially since deep learning models often involve millions of parameters. This underscores the significance of choosing an appropriate optimization algorithm tailored to your specific application. As a result, a solid understanding of these machine learning algorithms is crucial for data scientists before they dive deeper into the field [21].

You can employ various optimizers in a machine learning model to fine-tune weights and learning rates. However, selecting the most suitable optimizer depends heavily on the specific application. As a beginner, one might consider experimenting with different optimizers and selecting the one that delivers the best results. While this trial-and-error approach may work initially, it becomes increasingly impractical when working with hundreds of gigabytes of data, as even a single training epoch can consume a substantial amount of time. Randomly selecting an optimizer can end up being a costly gamble with your time—a realization that becomes clearer as you advance in your machine learning journey.

This guide will explore a variety of deep learning optimizers, such as Gradient Descent, Stochastic Gradient Descent (SGD), Stochastic Gradient Descent with Momentum, Mini-Batch Gradient Descent, Adagrad, RMSprop, AdaDelta, and Adam. By the end of the article, you will gain the ability to compare these optimizers and understand the underlying principles and procedures they rely on. This paper will help you make informed decisions when selecting the right optimizer for your specific deep learning tasks.

C. *Stochastic Gradient Descent Optimizer for Deep Learning*

At the end of the previous section, you discovered why gradient descent may not be the most suitable choice for very large datasets. To tackle the challenges associated with massive datasets, Stochastic Gradient Descent (SGD) emerges as a widely used optimizer in deep learning. The term "stochastic" refers to the randomness inherent in the algorithm's approach. Unlike traditional gradient descent,

which processes the entire dataset in each iteration, SGD randomly selects small batches of data. This means only a subset of samples is used at a time, making the optimization process more efficient and computationally manageable for training deep learning models [22].

$$\omega = \omega - \eta \nabla Qi(\omega) \tag{1}$$

The procedure for Stochastic Gradient Descent (SGD) begins, as shown in Fig. 5 by initializing the parameters and setting the learning rate. Next, the data is randomly shuffled at each iteration to ensure randomness in the selection of data samples. By processing small batches or individual data points at a time, the algorithm iteratively updates the parameters to move toward an approximate minimum of the loss function. This approach allows SGD to efficiently handle large datasets and converge faster compared to traditional gradient descent, albeit with some noise in the optimization process due to the randomness introduced [22].

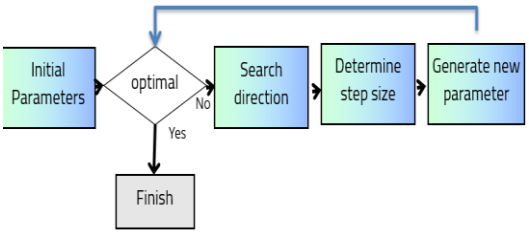

Fig. 5. The diagram of SGD.

Since SGD processes only small batches of data (or individual samples) in each iteration rather than the entire dataset, the optimization path it follows tends to be noisier compared to the gradient descent algorithm. As a result, SGD typically requires a higher number of iterations to converge to the local minimum. While this increases the overall computation time, the computational cost per iteration remains significantly lower than that of gradient descent. Therefore, even with more iterations, the total computational expense of SGD is still less than that of batch gradient descent. In conclusion, if you are working with extremely large datasets and computational efficiency is a critical factor, stochastic gradient descent is generally a better choice than batch gradient descent.

Algorithm 1 : *SGD algorithm*

Inputs: Initial vector θ, Learning rate η,
Maximum number of iterations max_iter

Begin:
$k \leftarrow 0$
While (not converged and $k \leq$ max_iter) do:
Shuffle the dataset
For each observation i in the dataset do:
Compute the gradient $\nabla f(\theta)$, for observation i
Compute the step: step $\leftarrow \eta \times \nabla f(\theta)$
Update θ: $\theta \leftarrow \theta +$ step
$k \leftarrow k + 1$
End While
End

D. *Stochastic Gradient Descent with Momentum (SGDM)*

As discussed earlier, stochastic gradient descent (SGD) follows a much noisier path compared to the gradient descent algorithm when optimizing deep learning models. This noise results in a higher number of iterations being required to reach the optimal minimum, which in turn slows down the computation time. To address this issue, we use Stochastic Gradient Descent with Momentum. This enhanced algorithm incorporates the concept of momentum, which helps accelerate convergence by smoothing the update path. Momentum achieves this by accumulating a fraction of the past gradients, effectively reducing oscillations and allowing the optimizer to move more consistently toward the minimum. This makes the training process faster and more stable compared to standard SGD.

Momentum accelerates the convergence of the loss function by addressing the oscillations commonly seen in stochastic gradient descent (SGD). In standard SGD, the updates to the weights can oscillate significantly, as the algorithm follows the gradient's direction in each iteration. By incorporating momentum, a fraction of the previous update is added to the current update, which helps smooth out these oscillations and allows the optimizer to move more consistently toward the minimum. This results in faster convergence.

However, when using momentum, it's important to remember that the learning rate should be reduced if a high momentum term is applied. This is because a high momentum term can cause the updates to overshoot the optimal minimum if the learning rate is too large. Balancing the learning rate and momentum is key to achieving stable and efficient training.

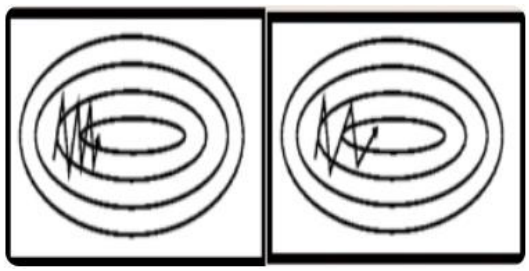

Fig. 6. Comparison of SGD algorithms without and with momentum.

In the Fig. 6, the left side illustrates the convergence path of the standard stochastic gradient descent (SGD) algorithm, while the right side demonstrates SGD with momentum. By comparing the two, it's clear that the addition of momentum results in a smoother and more direct path toward convergence, significantly reducing the time required to reach the optimal minimum.

You might consider using a large momentum and learning rate to further speed up the process. However, it's important to note that increasing the momentum too much can lead to overshooting the optimal minimum. This can cause the algorithm to miss the best solution, resulting in poor accuracy and even increased oscillations. Therefore, finding the right balance between momentum and learning rate is crucial to ensure efficient and stable convergence without compromising the model's performance [23].

In this variation of gradient descent, the loss function is computed using a subset of the training data rather than the

entire dataset. Fewer rounds are required because we are using a chunk of data rather than the entire dataset. Because of this, the mini-batch gradient descent algorithm outperforms the batch and stochastic gradient descent techniques. Compared to the previous gradient descent variations, this algorithm is more reliable and efficient. The procedure is more efficient to implement since the algorithm uses batching, which eliminates the need to put all of the training data into memory. Additionally, the mini-batch gradient descent algorithm's cost function is smoother than the stochastic gradient descent technique's but noisier than the batch gradient descent algorithm. Mini-batch gradient descent is therefore perfect since it offers a decent trade-off between accuracy and speed.

Despite all that, the mini-batch gradient descent algorithm has some downsides too. It needs a hyperparameter that is "mini-batch-size", which needs to be tuned to achieve the required accuracy. Although, the batch size of 32 is considered to be appropriate for almost every case. Also, in some cases, it results in poor final accuracy. Due to this, there needs a rise to look for other alternatives too.

Algorithm 2 : *SGD with Momentum*

Inputs: learning rate η***, momentum coefficient*** β***, initial parameters*** θ***, Initial velocity*** m

Begin:

while stopping criteria is not met do:

compose a minibatch $\{\theta^{(i)}, y^{(i)}\}$ ***from the dataset***

compute the gradient estimate:

$$g \leftarrow \frac{1}{n}\nabla_\theta \sum_i J(f(\theta^{(i)};\theta), y^{(i)})$$

update the velocity:

$$m \leftarrow \beta m - \eta . g$$

apply the update to parameters:

$$\theta \leftarrow \theta + m$$

End While

Return θ

End

E. Adaptive Gradient Descent Deep Learning Optimizer (Adagrad)

Despite its advantages, the mini-batch gradient descent algorithm also has some drawbacks. One key limitation is the need to tune the **"mini-batch size"** hyperparameter to achieve the desired accuracy. While a batch size of 32 is generally considered suitable for most cases, this may not always hold true, requiring experimentation to find the optimal size. Additionally, in certain scenarios, mini-batch gradient descent can lead to suboptimal final accuracy, which can be a significant drawback depending on the application. These limitations highlight the importance of exploring other optimization alternatives to address specific challenges and improve performance [24].

$$w_t = w_{t-1} - \eta'_t \frac{\partial L}{\partial \omega(\vdash -1)} \tag{2}$$

$$\eta'_t = \frac{\eta}{\sqrt{\alpha_t + \epsilon}} \tag{3}$$

The Adagrad optimizer offers a significant advantage by eliminating the need to manually adjust the learning rate. It is more robust and reliable compared to standard gradient descent and its variants, and it often achieves convergence faster. However, Adagrad has a notable drawback: it reduces the learning rate aggressively and monotonically over time. This occurs because the squared gradients in the denominator of the update rule continuously accumulate, causing the denominator to grow larger with each iteration. As a result, the learning rate can become excessively small, potentially reaching a point where the model can no longer learn effectively. This stagnation in learning compromises the model's accuracy, making it less suitable for tasks requiring prolonged training or fine-tuning.

Algorithm 3 : Adagrad

Algorithm 3 : *AdaGrad algorithm*

Inputs: learning rate η ***, small constant*** $\beta(10^{-7})$

initial parameters θ

Initialize:

Gradient accumulation variable $r \leftarrow 0$

Begin:

while stopping criteria is not met do:

compose a minibatch $\{\theta^{(i)}, y^{(i)}\}$ ***from the dataset***

apply an intermediate update:

$$\tilde{\theta} \leftarrow \theta + \eta . m$$

compute the gradient estimate:

$$g \leftarrow \frac{1}{n}\nabla_\theta \sum_i J(f(\theta^{(i)};\theta), y^{(i)})$$

accumulate the squares of gradients:

$$r \leftarrow r + g . g$$

calculate the update:

$$m \leftarrow -\frac{\eta}{\sqrt{r+\beta}} \odot g$$

apply the update to parameters:

$$\theta \leftarrow \theta + m$$

End While

Return θ

End

F. Root Mean Square Deep Learning Optimizer (RMSProp)

RMSprop is a widely popular optimizer among deep learning practitioners, despite not being formally published in a research paper. It is well-regarded in the community and is considered an extension of the RMSprop algorithm. RMSprop addresses the issue of varying gradients, where some gradients may be very small while others are large, making it challenging to define a single global learning rate. RMSprop solves this by adapting the step size individually for each weight based on the sign of the gradients. It compares the signs of two consecutive gradients: if they are the same, it increases the step size slightly, indicating that the optimization is moving in the right direction. If the signs are opposite, it decreases the step size to avoid overshooting. After adjusting the step size, it is clipped to stay within a reasonable range, and the weight update is performed. RMSprop builds on this idea but uses a different approach to adapt the learning rate, making it more efficient and robust

for training deep neural networks. It is particularly effective in handling non-stationary objectives and noisy gradients, which are common in deep learning tasks [25].

The issue with RMSprop is that it struggles with large datasets and mini-batch updates, limiting its effectiveness in such scenarios. The goal of combining the robustness of RMSprop with the efficiency of mini-batch updates led to the development of improved optimization techniques. RMSprop itself is an enhancement over the AdaGrad optimizer, addressing AdaGrad's key limitation: the monotonically decreasing learning rate. While AdaGrad accumulates squared gradients, causing the learning rate to shrink excessively over time, RMSprop introduces a decay factor to control this accumulation. This adjustment prevents the learning rate from becoming too small, allowing for more stable and efficient training. However, further advancements were still needed to fully address the challenges of large-scale datasets and mini-batch optimization [25].

The RMSprop algorithm primarily aims to accelerate the optimization process by reducing the number of function evaluations required to reach the local minimum. It achieves this by maintaining a moving average of squared gradients for each weight. During each update, the gradient is divided by the square root of this mean square, which helps normalize the step size. This approach effectively adapts the learning rate for each parameter, addressing the issue of varying gradient magnitudes. By doing so, RMSprop ensures more stable and efficient convergence, particularly in scenarios with noisy or non-stationary gradients. This makes it a popular choice for optimizing deep learning models.

$$\vartheta(w,t) \coloneqq \gamma\vartheta(w,t-1) + (1-\gamma)\left(\nabla Q_i(\omega)\right)^2 \quad (4)$$

where gamma is the forgetting factor. Weights are updated by the below formula

$$\omega \coloneqq w - \frac{\eta}{\sqrt{v(w,t)}}\nabla Q_i(w) \quad (5)$$

In simpler terms, RMSprop works by penalizing updates to parameters that cause significant oscillations in the cost function. For example, imagine you built a model to classify different types of fish, and the model heavily relies on the feature "color" to make predictions. This over-reliance on a single feature leads to many errors. RMSprop addresses this by penalizing the parameter associated with "color," encouraging the model to consider other features as well. This prevents the algorithm from adapting too quickly to changes in the "color" parameter compared to others, ensuring a more balanced learning process.

Compared to earlier gradient descent algorithms, RMSprop offers several advantages. It converges faster and requires less hyperparameter tuning, making it more user-friendly and efficient. By adaptively adjusting the learning rate for each parameter, RMSprop provides a more stable and robust optimization process, especially in scenarios with noisy or imbalanced data [25].

The main drawback of RMSprop is that it requires the learning rate to be set manually, and the default value may not suit every application. This means users often need to experiment to find the optimal learning rate, which can be time-consuming. While RMSprop adapts the learning rate for each parameter, the initial value remains crucial. If not set properly, it can lead to slow convergence or poor performance, highlighting the need for more advanced optimizers with less manual tuning [25].

Algorithm 4 : RMSprop

Inputs: learning rate η ***, small constant*** $\beta(10^{-7})$
initial parameters θ
Initialize:
Gradient accumulation variable $r \leftarrow 0$
Begin:
while stopping criteria is not met do:
compose a minibatch $\{\theta^{(i)}, y^{(i)}\}$ ***from the dataset***
apply an intermediate update: $\tilde{\theta} \leftarrow \theta + \eta.m$
compute the gradient estimate: $g \leftarrow \frac{1}{n}\nabla_\theta \sum_i J(f(\theta^{(i)};\theta), y^{(i)})$
accumulate the squares of gradients: $r \leftarrow \beta r + (1-\beta)g.g$
calculate the update: $m \leftarrow -\frac{\eta}{\sqrt{r+\beta}} \odot g$
apply the update to parameters: $\theta \leftarrow \theta + m$
End While
Return θ
End

C. Adam Optimizer in Deep Learning

The Adam optimizer, short for Adaptive Moment Estimation, is a widely-used optimization algorithm in deep learning. It builds upon the stochastic gradient descent (SGD) approach by dynamically adjusting the weights of a neural network during training. Adam combines the benefits of momentum-based methods and adaptive learning rates, using estimates of both the first and second moments of gradients to update parameters efficiently. This results in faster convergence, improved stability, and better performance across a wide range of deep learning tasks. Its adaptability and robustness have made Adam one of the most popular optimizers in the field.

The Adam optimizer adaptively adjusts the learning rate for each individual weight in a neural network. Unlike Stochastic Gradient Descent (SGD), which uses a fixed, global learning rate for all parameters, Adam dynamically computes unique learning rates for every parameter during training. This is achieved by maintaining and utilizing two key pieces of information:

1) The first moment (mean) of past gradients, which provides momentum-like acceleration.
2) The previous gradients' second instant (uncentered variance), which aids in adaptively scaling the learning rate.

By leveraging these moments, Adam ensures efficient and stable optimization, making it a powerful and widely-used optimizer in deep learning.

In order to create a very efficient optimizer, the Adam optimizer's developers integrated the advantages of several optimization algorithms, including RMSProp and AdaGrad.

Similar to RMSProp, Adam takes into account the gradients' second moment, which aids in adjusting the learning rate for every parameter. Adam, in contrast to RMSProp, determines the gradients' uncentered variance (without deducting the mean). Adam can dynamically modify learning rates and attain faster convergence using this method with the addition of momentum (initial instant of gradients). By combining these characteristics, Adam offers a reliable and effective optimization technique that performs better on deep learning tasks than many conventional algorithms.

The Adam optimizer creates an adaptive learning rate that effectively traverses the optimization landscape during training by taking into account both the gradients' first moment (mean) and second moment (uncentered variance). This flexibility allows for quicker convergence and improves the neural network's overall performance.

In conclusion, by dynamically modifying learning rates for specific weights, the Adam optimizer provides a sophisticated optimization technique that builds upon Stochastic Gradient Descent (SGD). By utilizing adaptive learning rates and momentum, it combines the advantages of RMSProp and AdaGrad to deliver effective and efficient network weight updates. Because of this, Adam is a strong and popular deep learning optimizer that strikes a balance between performance, stability, and speed.

Because of its many advantages, the Adam optimizer is widely utilized. As a default optimization approach, it is suggested and modified as a benchmark for deep learning research. In addition, the technique requires less tuning than any other optimization algorithm, is easy to build, and runs faster and uses less memory [26].

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\delta L}{\delta \omega_t}\right] \tag{6}$$

$$\vartheta_t = \beta_2 \vartheta_{t-1} + (1-\beta_2)\left[\frac{\delta L}{\delta \omega_t}\right]^2 \tag{7}$$

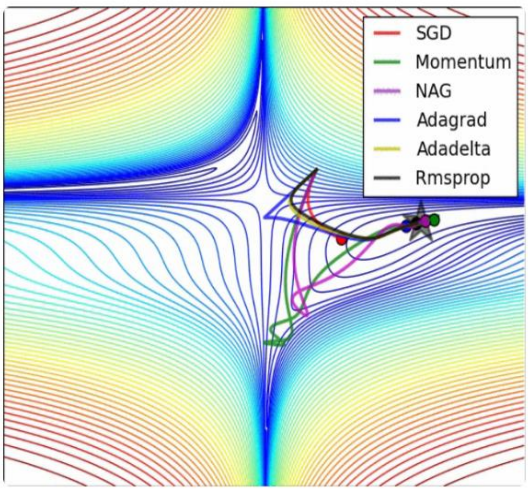

Fig. 7. The variety of SGD optimizers.

While the Adam optimizer combines the strengths of various algorithms and is often considered one of the best optimizers available, it is not universally suitable for every application. Adam prioritizes faster computation and efficient convergence, but this can sometimes come at the cost of generalization performance. Algorithms like Stochastic Gradient Descent (SGD), though slower, often generalize better because they focus more on individual data points during training. This makes SGD a better choice for tasks where generalization is critical, even if it requires more computation time [27].

From the above, we have represented several types of SGD algorithms to see the differences between them as shown in Fig. 7 and to see the functionality of each one.

Algorithm 5 : Adam

Inputs: Learning rate η
Decay rates β_1 *and* β_2 **, *Small constant*** α (10^{-7}),
Initialize:
1st moment vector $m_0 \leftarrow 0$
2nd moment vector $v_0 \leftarrow 0$
Time step $t \leftarrow 0$
Gradient accumulation variable $r \leftarrow 0$
Begin:
While θ_t ***not converged do:***
$t \leftarrow t+1$
Compute gradients for step t:
$g_t \leftarrow \nabla_\theta J_t(\theta_{t-1})$
Update biased 1st moment estimate:
$m_t \leftarrow \beta_1 m_{t-1} - (1-\beta_1)\, g_t$
Update biased 2nd raw moment estimate:
$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) {g_t}^2$
Accumulate the squares of gradients:
$r \leftarrow \beta r + (1-\beta) g.g$
Compute bias-corrected 1st moment estimate:
$\widehat{m_t} \leftarrow m_t/(1-\beta_1^t)$
Compute bias-corrected 2nd moment 4 estimate:
$\widehat{v_t} \leftarrow v_t/(1-\beta_2^t)$
Update parameters:
$\theta_t \leftarrow \theta_{t-1} + \eta\, \widehat{m_t}/(\sqrt{\widehat{v_t}} + \alpha)$
End While
Return θ_t
End

IV. Experimental Results

A. Evaluation Metrics

The suggested method's performance was evaluated using several metrics derived from the tested sample photographs' categorization as shown in Fig. 8. These measurements were computed using a contingency table called the confusion matrix. The percentage of correctly classified image samples, regardless of their class labels, is displayed by the accuracy statistic. Specificity is the percentage of correctly classified negative class samples, while sensitivity is the percentage of correctly classified positive class samples in binary classification, which is relevant to our study [19].

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{8}$$

$$Sensitivity = \frac{TP}{TP+FN} \tag{9}$$

$$Specificity = \frac{TN}{TN+FN} \tag{10}$$

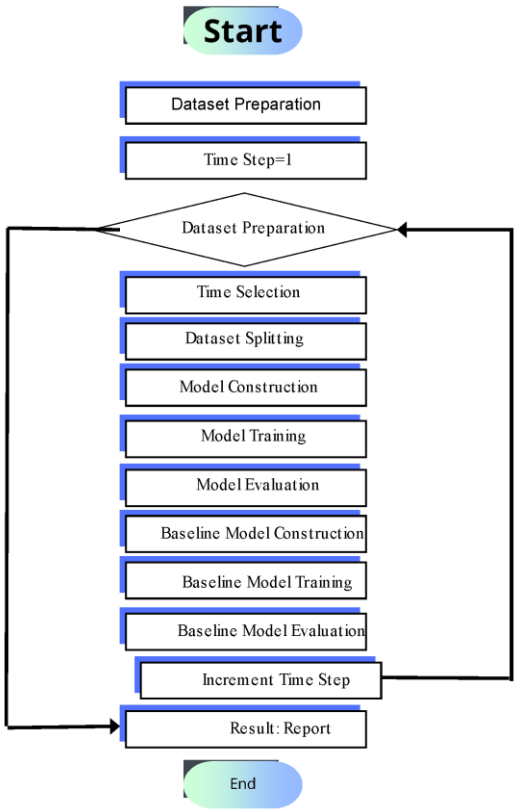

Fig. 8. The experiments flowchart.

B. *Algorithms Best Parameters*

Several variables, including learning rate, batch size, optimizer selection, loss function, activation functions, and the number of epochs, are considered optimal parameters for GRU algorithms. The adjustment of these parameters depends on the specific network architecture as shown in Fig. 9, the characteristics of the dataset, and the desired task. The ideal parameters identified for our pre-trained GRU model are presented in Table 1 [20].

Table 1. The best hyper-parameters used for the TL models in training phase of different optimizers

Network	Learning rate	Batch Size	Optimiser	Loss Function	Epochs
RNNs model	1.00e10-7	10	SGD momentum	Mean-Squared-Error	140
	1.00e10-7	10	Adagrad	Mean-Squared-Error	140
	1.00e10-7	10	RMSProp	Mean-Squared-Error	140
	1.00e10-7	10	Adam	Mean-Squared-error	140

C. *Training and Testing Results*

Figs. 10, 11, and 12 show that the GRU model using the RMSprop optimizer, starting from epoch 30, yields satisfactory results, as the training and validation curves converge. However, between epochs 80 and 140, these curves begin to diverge, indicating a decline in performance. On the other hand, for the Adagrad and Adam optimizers, the training and validation curves gradually converge from epoch 30 to epoch 140, suggesting that these optimizers deliver better results for the GRU model. Among them, the Adam optimizer stands out as the most effective for this model.

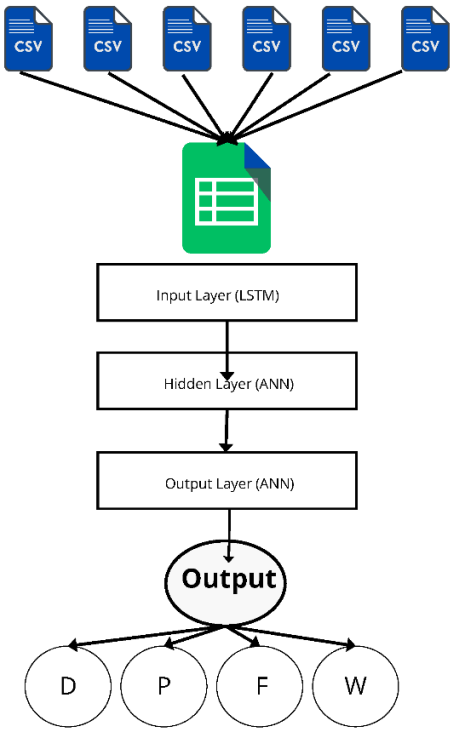

Fig. 9. Architecture of the proposed model.

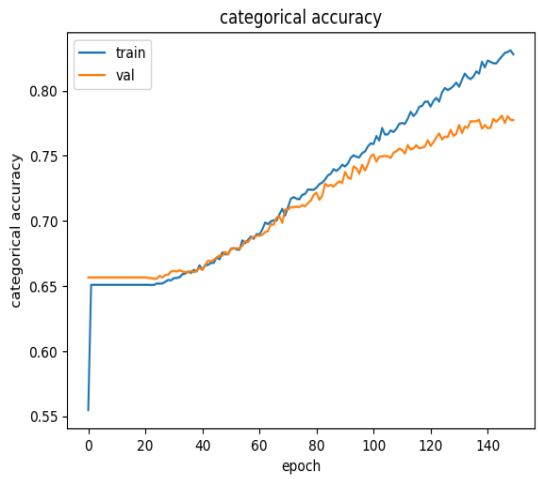

Fig. 10. The accuracy obtained by training and evaluating the GRU model based on RMSProp optimizer.

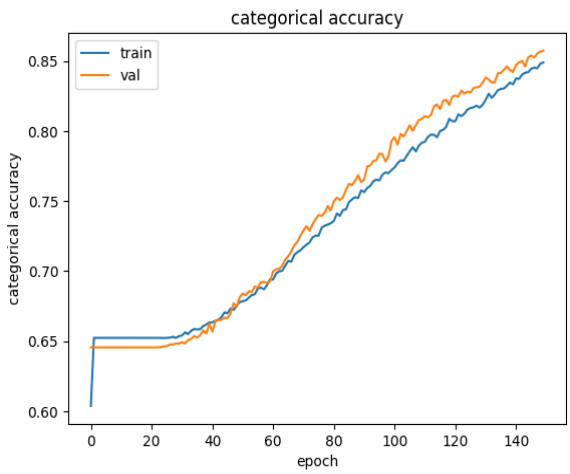

Fig. 11. The accuracy obtained by training and evaluating the GRU model based on Adagrad optimizer.

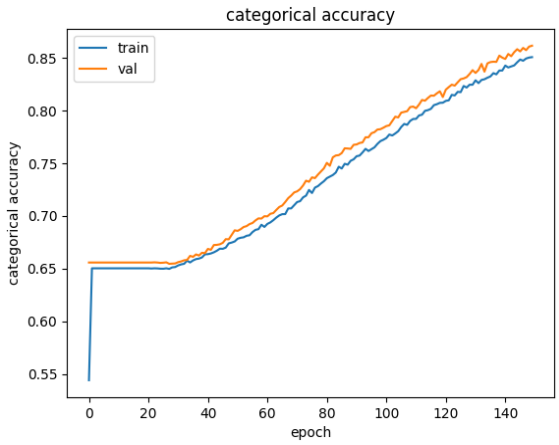

Fig. 12. The accuracy obtained by training and evaluating the GRU model based on Adam optimizer.

Figs. 13, 14, and 15 show that the LSTM model using the RMSprop optimizer yields good results from epoch 0 to epoch 30. However, after epoch 30, the curves begin to diverge, indicating that this optimizer is not the most suitable choice for LSTM models in this specific case and with the data used. On the other hand, for the Adagrad and Adam optimizers, the training and validation curves converge progressively from epoch 30 to epoch 140, suggesting that both optimizers deliver better results for the LSTM model. Among them, the Adam optimizer remains the best choice for this model, as well as for the GRU model.

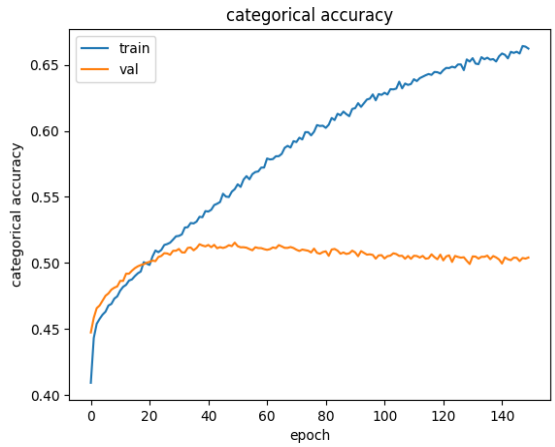

Fig. 13. The accuracy obtained by training and evaluating the LSTM model based on RMSProp optimizer.

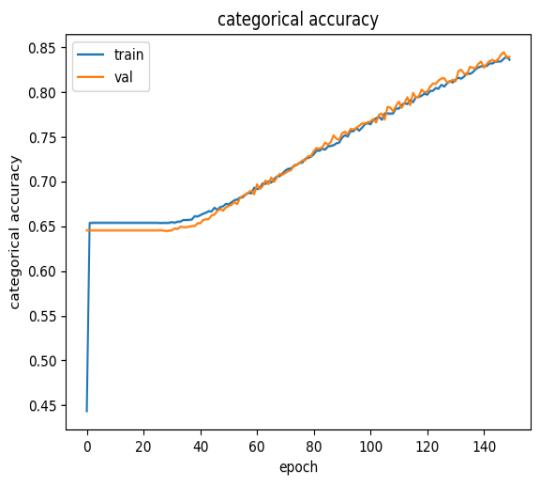

Fig. 14. The accuracy obtained by training and evaluating the LSTM model based on Adagrad optimizer.

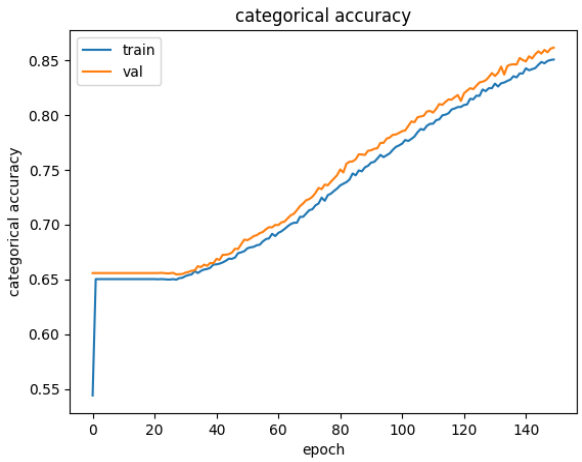

Fig. 15. The accuracy obtained by training and evaluating the LSTM model based on Adam optimizer.

D. *Discussion*

Initially, a day-wise deep learning model known as the GRU (Gated Recurrent Unit) model was developed to forecast students' performance across multiple classes. This model was specifically designed to predict the performance category of students in a multi-class format. The GRU model achieved an accuracy of 88% on the validation dataset, which was comparable to its performance on the training dataset. Notably, it provided the most precise predictions regarding students' performance compared to other evaluated models, such as the LSTM model. The primary objective of the study was to assess the GRU model against other state-of-the-art models.

On the one hand results indicated that the GRU model consistently outperformed its counterparts. Specifically, when incorporating clickstream and demographic data from the final day of courses, the GRU model demonstrated superior accuracy compared to the LSTM model. While the GRU achieved an accuracy of 88%, the LSTM model lagged behind at 63%. This performance gap can be attributed to the GRU model's ability to effectively retain long-term relationships between parameters.

On the other hand, according to the figure we notice that the Adam optimizer is always given better results by comparing with the other optimizers either RMSProp or Adagrad.

GRU model with adam optimizer is always better than LSTM model with Adam optimizer. This explains that GRU has fewer control ports and a state of the cell, which accounts for its lower computational complexity and lower cutoff point than LSTM.

V. Conclusion and Perspectives

The use of deep learning to predict student performance represents a significant advancement in the field of education. By leveraging advanced neural network architectures, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformer-based models, deep learning can effectively analyze complex and diverse datasets, including academic records, learning behaviors, and engagement metrics. These models excel at identifying hidden patterns and correlations that traditional statistical

methods often overlook, enabling more accurate and personalized predictions.

The The application of various optimization algorithms, such as Stochastic Gradient Descent (SGD), plays a critical role in training these models. SGD, along with its variants like SGD with momentum, Adagrad, RMSProp, and Adam, allows for efficient adjustment of model parameters to minimize the loss function and improve prediction accuracy. Each of these algorithms offers specific advantages. This paper proposed an application of RNNs models with different optimization algorithms and obtained a precision of more than 90%.

However, challenges remain, such as the need for high-quality, well-annotated data, the interpretability of deep learning models, and the risks of bias in training data. To ensure responsible implementation, it is essential to employ robust preprocessing techniques, improve model transparency, and address ethical considerations, application of other optimization method that I did not have time to apply like ADadelta algorithm and others, that's our future work.

In conclusion, deep learning, combined with optimization algorithms like SGD and its variants, offers powerful tools for predicting student performance. This approach has the potential to transform educational systems by enabling data-driven decision-making and improving learning outcomes. With ongoing advancements in research and technology, the integration of deep learning into education will open new possibilities for teaching and learning.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Abdelmajid El Hajoui conducted the research, analyzed the data; and wrote the paper; Otmane Yazidi Alaoui check the database and provide programming guidance; Omar El Kharki provides feedback to properly apply the RNNS to our database; Mustapha Maatouk provide important references and check the writing; Miriam Wahbi corrects the English writing and give advice on content; Hakim Boulaassal correct the English writing give advice on content. All authors had approved the final version.

References

[1] I. Katib and M. Ragab, "Harnessing variable reduction approach with deep recurrent neural network for student's academic performance analysis," *Alexandria Engineering Journal*, vol. 118, pp. 393–405, 2025. doi: 10.1016/j.aej.2025.01.082

[2] G. Al-Tameemi, J. Xue, I. H. Ali, and S. Ajit, "A hybrid machine learning approach for predicting student performance using multi-class educational datasets," *Procedia Computer Science*, vol. 238, pp. 888–895, 2024. doi: 10.1016/j.procs.2024.06.108

[3] M. Adnan, A. A. S. Alarood, M. I. Uddin, and I. Ur Rehman, "Utilizing grid search cross-validation with adaptive boosting for augmenting performance of machine learning models," *PeerJ Computer Science*, vol. 8, p. e803, 2022. doi: 10.7717/peerj-cs.803

[4] F. A. Al-azazi and M. Ghurab, "ANN-LSTM: A deep learning model for early student performance prediction in MOOC," *Heliyon*, vol. 9, no. 4, p. e15382, 2023. doi: 10.1016/j.heliyon.2023.e15382

[5] X. Chen, H. Xie, D. Zou, G. Cheng, X. Tao, and F. Lee Wang, "Perceived MOOC satisfaction: A review mining approach using machine learning and fine-tuned BERTs," *Computers and Education: Artificial Intelligence*, vol. 8, 100366, 2025. doi: 10.1016/j.caeai.2025.100366

[6] I. D. Mienye, T. G. Swart, and G. Obaido, "Recurrent neural networks: A comprehensive review of architectures, variants, and applications," *Information*, vol. 15, no. 9, p. 517, 2024. doi: 10.3390/info15090517

[7] B. Pei and W. Xing, "An interpretable pipeline for identifying at-risk students," *Journal of Educational Computing Research*, vol. 60, no. 2, pp. 380–405, 2022. doi: 10.1177/07356331211038168

[8] Y. Bee Wah *et al.*, "Machine learning and synthetic minority oversampling techniques for imbalanced data: Improving machine failure prediction," *Computers, Materials & Continua*, vol. 75, no. 3, pp. 4821–4841, 2023. doi: 10.32604/cmc.2023.034470

[9] F. Qiu *et al.*, "Predicting students' performance in e-learning using learning process and behaviour data," *Sci Rep*, vol. 12, no. 1, p. 453, 2022. doi: 10.1038/s41598-021-03867-8

[10] J. Hao, J. Gan, and L. Zhu, "MOOC performance prediction and personal performance improvement via Bayesian network," *Educ Inf Technol*, vol. 27, no. 5, pp. 7303–7326, 2022. doi: 10.1007/s10639-022-10926-8

[11] F. Ouyang, L. Zheng, and P. Jiao, "Artificial intelligence in online higher education: A systematic review of empirical research from 2011 to 2020," *Educ Inf Technol*, vol. 27, no. 6, pp. 7893–7925, 2022. doi: 10.1007/s10639-022-10925-9

[12] A. Esteban, C. Romero, and A. Zafra, "Assignments as influential factor to improve the prediction of student performance in online courses," *Applied Sciences*, vol. 11, no. 21, p. 10145, 2021. doi: 10.3390/app112110145

[13] B. K. Verma, D. N. Srivastava, and H. K. Singh, "Prediction of students' performance in e-learning environment using data mining/machine learning techniques," *J. Univ. Shanghai Sci. Technol*, vol. 23, no. 05, pp. 596–593, 2021. doi: 10.51201/JUSST/21/05179

[14] G. Vrbančič and V. Podgorelec, "Efficient ensemble for image-based identification of Pneumonia utilizing deep CNN and SGD with warm restarts," *Expert Systems with Applications*, vol. 187, 115834, 2022. doi: 10.1016/j.eswa.2021.115834

[15] T. Tan, H. Xie, Y. Xia, X. Shi, and M. Shang, "Asynchronous SGD with stale gradient dynamic adjustment for deep learning training," *Information Sciences*, vol. 681, 121220, 2024. doi: 10.1016/j.ins.2024.121220

[16] open_dataset. [Online]. Available: https://analyse.kmi.open.ac.uk/open_dataset

[17] A. Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network," *Physica D: Nonlinear Phenomena*, vol. 404, 132306, 2020. doi: 10.1016/j.physd.2019.132306

[18] J. Wang, J. Yan, C. Li, R. X. Gao, and R. Zhao, "Deep heterogeneous GRU model for predictive analytics in smart manufacturing: Application to tool wear prediction," *Computers in Industry*, vol. 111, pp. 1–14, 2019. doi: 10.1016/j.compind.2019.06.001

[19] B. C. Mateus, M. Mendes, J. T. Farinha, R. Assis, and A. M. Cardoso, "Comparing LSTM and GRU models to predict the condition of a pulp paper press," *Energies*, vol. 14, no. 21, p. 6958, 2021. doi: 10.3390/en14216958

[20] N. Zafar, I. U. Haq, J.-R. Chughtai, and O. Shafiq, "Applying hybrid LSTM-Gru model based on heterogeneous data sources for traffic speed prediction in urban areas," *Sensors*, vol. 22, no. 9, p. 3348, 2022. doi: 10.3390/s22093348

[21] Y. Tian, Y. Zhang, and H. Zhang, "Recent advances in stochastic gradient descent in deep learning," *Mathematics*, vol. 11, no. 3, p. 682, 2023. doi: 10.3390/math11030682

[22] I. Dagal, K. Tanriöven, A. Nayir, and B. Akın, "Adaptive Stochastic Gradient Descent (SGD) for erratic datasets," *Future Generation Computer Systems*, vol. 166, 107682, 2025. doi: 10.1016/j.future.2024.107682

[23] X. Zhou, Z. You, W. Sun, D. Zhao, and S. Yan, "Fractional-order stochastic gradient descent method with momentum and energy for deep neural networks," *Neural Networks*, vol. 181, 106810, 2025. doi: 10.1016/j.neunet.2024.106810

[24] S. Bhakta, U. Nandi, C. Changdar, B. Paul, T. Si, and R. K. Pal, "aMacP: An adaptive optimization algorithm for deep neural network," *Neurocomputing*, vol. 620, 129242, 2025. doi: 10.1016/j.neucom.2024.129242

[25] K. Bouanane, B. Dokkar, M. Allaoui, B. Meddour, M. L. Kherfi, and R. Hedjam, "Behaviors of first-order optimizers in the context of sparse data and sparse models: A comparative study," *Digital Signal Processing*, vol. 153, 104637, 2024. doi: 10.1016/j.dsp.2024.104637

[26] J. J. Jeong and G. Koo, "AdaLo: Adaptive learning rate optimizer with loss for classification," *Information Sciences*, vol. 690, 121607, 2025. doi: 10.1016/j.ins.2024.121607

[27] B. Barati, M. Erfaninejad, and H. Khanbabaei, "Evaluation of effect of optimizers and loss functions on prediction accuracy of brain tumor

type using a Light neural network," *Biomedical Signal Processing and Control*, vol. 103, 107409, 2025. doi: 10.1016/j.bspc.2024.107409

Copyright © 2025 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited (CC BY 4.0).