VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**

**TRAN HUU NHAN- 521H0507**

# FINAL REPORT INTRODUCTION TO MACHINE LEARNING

**HO CHI MINH CITY, YEAR 2023**

VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**

**TRAN HUU NHAN- 521H0507**

# FINAL REPORT

# INTRODUCTION TO MACHINE LEARNING

Advised by

**Assoc. Prof. Le Anh Cuong**

**HO CHI MINH CITY, YEAR 2023**

# ACKNOWLEDGMENT

I would like to express my sincere gratitude for your invaluable guidance and mentorship throughout the Introduction to Machine learning course. Your expertise and passion for the subject matter have inspired and enriched my understanding.

*Ho Chi Minh City, day 23 month 12 year 2023*

*Author*

*(Signature and full name)*

Nhan

Tran Huu Nhan

# DECLARATION OF AUTHORSHIP

I hereby declare that this thesis was carried out by myself under the guidance and supervision of Assoc. Prof. Le Anh Cuong and that the work and the results contained in it are original and have not been submitted anywhere for any previous purposes. The data and figures presented in this thesis are for analysis, comments, and evaluations from various resources by my own work and have been duly acknowledged in the reference part.

In addition, other comments, reviews and data used by other authors, and organizations have been acknowledged, and explicitly cited.

**I will take full responsibility for any fraud detected in my thesis**. Ton Duc Thang University is unrelated to any copyright infringement caused on my work (if any).

*Ho Chi Minh City, day 23 month 12 year 2023*

*Author*

*(Signature and full name)*

*Nhan*

*Tran Huu Nhan*

# CONTENTS

# LIST OF FIGURES

# CHAPTER 1. OPTIMIZATION STRATEGIES: A COMPARATIVE STUDY IN MACHINE LEARNING MODEL TRAINING

## 1.1 Introduction

Optimization is a pivotal process in Machine Learning, where models are iteratively trained to achieve maximum or minimum function evaluation, thereby enhancing results. The primary objective of Machine Learning models is to minimize the loss function. This involves calculating the error after passing an input through the model and subsequently updating the weights. The optimizer plays a crucial role in this process by determining how to adjust the parameters to approach the minima.

The aim of optimizing strategies in machine learning models to create more accurate models with lower error rates. This is accomplished by continuously comparing the results in each iteration and adjusting the hyperparameters at each step until optimal results are achieved. Essentially, optimization entails finding the optimal parameters for the model, leading to a significant reduction in the error function.

Within this essay, two prominent optimization algorithms are examined: Gradient Descent, Adam. These algorithms are widely employed in Machine Learning models and possess a profound impact on the achievement of optimization goals, guiding the search towards the minimum of the loss function. By comprehending and applying these algorithms, researchers can effectively optimize their models, resulting in improved accuracy and more precise predictions.

## 1.2 Gradient Descent

### *1.2.1 The cost function*

It is a function used to gauge the accuracy of a model’s predictions against actual results, providing a single numerical value that represents the discrepancy.

Initially, a hypothesis is formulated using certain parameters. The cost function is then computed based on this hypothesis. The objective is to minimize the value of the cost function. To achieve this, the parameters are adjusted using the Gradient Descent algorithm applied to the data at hand. This process can be mathematically represented.

**Mathematical representation:**

| | |
|---|---|
| **Hypothesis**: | $h_\theta(x) = \theta_0 + \theta_1 x$ |
| **Parameters**: | $\theta_0, \theta_1$ |
| **Cost Function** | $J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$ |
| **Goal**: | minimize $J(\theta_0, \theta_1)$ |

### *1.2.2 Gradient Descent*

Gradient descent (GD) is a fundamental optimization algorithm in machine learning that aims to minimize the cost function. It operates by iteratively adjusting model parameters in the direction of the negative gradient, which represents the steepest ascent. The objective is to find the optimal set of parameters that minimizes the discrepancy between the model's predicted output and the actual output, thereby enhancing the model's performance.

The algorithm calculates the gradient of the cost function to determine the direction and magnitude of the steepest ascent. However, since the goal is to minimize the cost function, the algorithm moves in the opposite direction of the gradient, known as the negative gradient direction.

The iterative update of the model's parameters in the negative gradient direction allows gradient descent to gradually converge towards the optimal set of parameters that yields the lowest cost. The learning rate, a hyperparameter, influences the step size taken in each iteration, thereby affecting the speed and stability of convergence.

Gradient descent is applicable to various machine learning algorithms, including linear regression, logistic regression, neural networks, and support vector machines. It provides a general framework for optimizing models by iteratively refining their parameters based on the cost function. This iterative refinement is crucial in machine learning, as it enables models to learn from data and improve their performance.

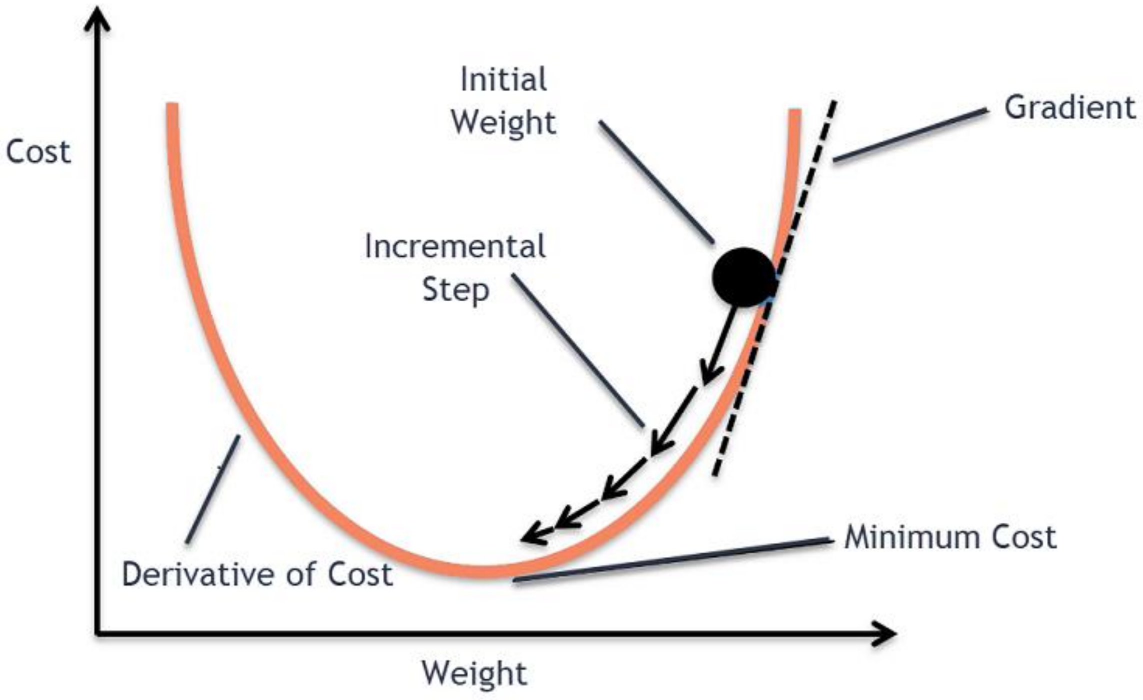


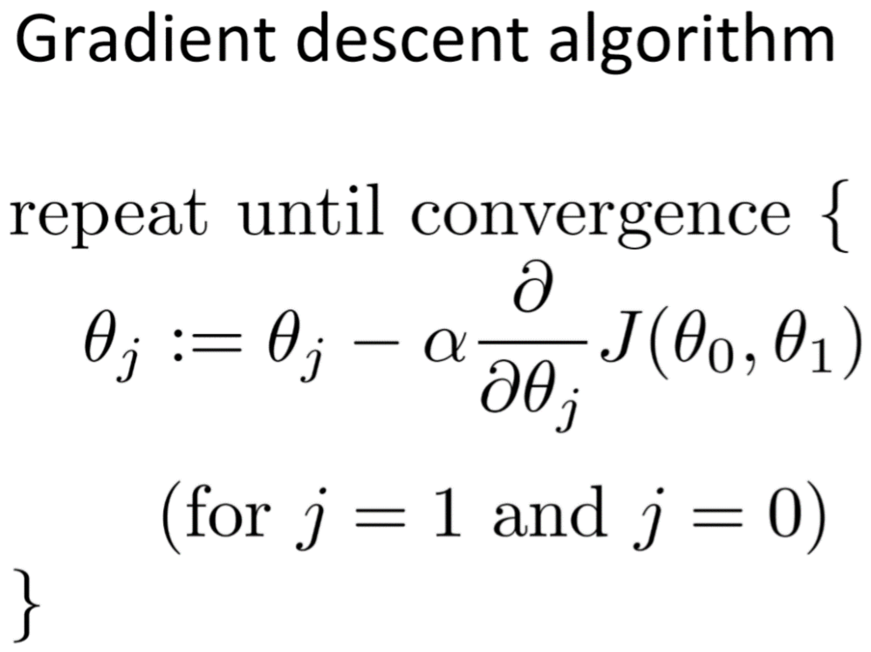
Figure 1 Gradient Descent Algorithm (Source: Clairvoyant)

### *1.2.3 Gradient Descent Algorithm*

The objective of the gradient descent algorithm is to find the minimum of a given function, often referred to as the cost function. To accomplish this, it carries out two iterative steps:

1. It calculates the gradient, which is the first derivative of the function at a specific point, representing the slope.
2. It then takes a step in the direction that is opposite to the gradient, which is the direction where the slope decreases. The size of this step is determined by multiplying the gradient at that point by a factor known as alpha.

In essence, the algorithm continually adjusts its position by following the path of steepest descent, with the aim of finding the lowest point of the function. The alpha factor controls the size of these adjustments.



Gradient descent algorithm

```
repeat until convergence {
    $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$
        (for $j = 1$ and $j = 0$)
}
```

Figure 2 Gradient descent algorithm (Source: Coursera)

1. **Initialization**: Start with an initial set of parameters for the model.
2. **Compute the Cost Function**: Measure how well the model fits the training data using the cost function, which is defined based on the difference between the predicted and actual values.
3. **Calculate the Gradient**: Compute the gradient of the cost function with respect to each parameter. The gradient points in the direction of the steepest ascent.
4. **Update the Parameters**: Adjust the model's parameters in small steps in the opposite direction of the gradient, which is the direction of the steepest descent. The size of the step is controlled by the learning rate.

5. **Repeat**: Continue the process of computing the cost function, calculating the gradient, and updating the parameters until the cost function converges to a minimum.
6. **Check Convergence**: Once the cost function has reached a minimum, the model has found the optimal set of parameters.
7. **Consider Variations**: Depending on the size of the dataset and the specific problem, different variations of gradient descent such as batch gradient descent, stochastic gradient descent, and mini-batch gradient descent can be used, each with its own advantages and limitations.
8. **Optimize Performance**: For efficient implementation and good performance in machine learning tasks, the choice of the learning rate and the number of iterations can significantly impact the performance of the algorithm.

### *1.2.4 Type of Gradient Descent*

#### 1.2.4.1 Batch Gradient Descent

Definition: This method updates the model's parameters using the gradient calculated from the entire training set. It computes the average gradient of the cost function across all training examples and adjusts the parameters in the opposite direction.

**Pros:**

- Guarantees convergence to the global minimum.
- Can leverage matrix computation for speed.

**Cons:**

- Can be computationally expensive and slow for large datasets.
- Requires running through the entire dataset before making a single update.

### 1.2.4.2 Stochastic Gradient Descent

**Definition**: Stochastic gradient descent updates the model's parameters using the gradient calculated from a single training example at a time. It randomly picks a training example, computes the gradient of the cost function for that example, and adjusts the parameters in the opposite direction.

**Pros:**

- Computationally efficient and can converge faster than batch gradient descent.
- Due to its randomness, it can escape local minima.

**Cons:**

- Can be noisy and may not necessarily converge to the global minimum.
- Frequent parameter updates can lead to significant oscillations.

### 1.2.4.3 Mini-Batch Gradient Descent

**Definition**: This method updates the model's parameters using the gradient calculated from a small subset of the training set, known as a mini batch. It computes the average gradient of the cost function for the mini-batch and adjusts the parameters in the opposite direction.

**Pros:**

- Combines the advantages of both batch and stochastic gradient descent.
- Computationally efficient and less noisy than stochastic gradient descent.

**Cons:**

- Requires choosing an appropriate mini-batch size.
- Can get stuck in local minima, depending on the learning rate and mini-batch size.

### *1.2.5 Challenges and Considerations in Gradient Descent Optimization*

1. **Local Optima**: Gradient descent may settle at local optima, not the global one, particularly if the cost function is complex with multiple peaks and valleys.
2. **Learning Rate Selection**: The choice of learning rate greatly affects gradient descent's performance. Too high, it may miss the minimum; too low, convergence may be slow.
3. **Overfitting**: If the model is overly complex or the learning rate is high, gradient descent may overfit the training data, leading to poor performance on new data.
4. **Convergence Rate**: For large datasets or high-dimensional spaces, gradient descent's convergence can be slow, making it computationally costly.
5. **Saddle Points**: In high-dimensional spaces, the cost function's gradient can have saddle points, potentially causing gradient descent to plateau instead of finding a minimum.

## 1.3 AdaGrad

### *1.3.1 Adaptive learning rate algorithm*

To get precise and effective outcomes in the realm of deep learning and machine learning, it's essential to optimize a model's parameters. Several optimization algorithms have been created to improve training and speed convergence.

Adaptive learning rates are necessary because conventional optimization techniques, such as Gradient Descent, use a fixed learning rate for all parameters during training. However, this uniform learning rate may not be optimal for all parameters, leading to convergence issues. Some parameters may require more frequent updates to converge faster, while others may need smaller adjustments to avoid overshooting the desired value.

### *1.3.2 AdaGrad algorithm*

**AdaGrad** is one of the earliest adaptive learning rate algorithms, which has been introduced in 2011 by Duchi et al. It main objective is to hasten convergence for sparse gradient parameters. Each parameter's previous gradient information is tracked by the algorithm, which then modifies the learning rate as necessary.

To overcome this limitation, algorithms with adaptive learning rates have been developed. These approaches allow the algorithm to navigate the optimization landscape more effectively by adjusting the learning rate for each parameter based on their past gradients.

### 1.3.2.1 How AdaGrad work

AdaGrad's key idea is to adjust the learning rate for each parameter based on the cumulative squared gradients observed during training.

Formula for AdaGrad

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} \cdot g_{t,i}$$

Where:

- $\theta_{t+1}$the value of the parameter i at the t+1 iteration.
- $\eta$ the learning rate, which determines the step size in parameter updates.
- $G_{t,i}$ is the accumulated sum of squared gradients for parameter i up to iteration t
- $g_{t,i}$ is the gradient of the loss function with respect to parameter i at iteration t.$\varepsilon$ is a small constant to avoid division by zero.

The algorithm proceeds as follows:

**Step 1: Variable Initialization**

Start by setting the parameters θ and a small constant ϵ (to prevent division by zero). Also, initialize the sum of squared gradients variable G, which should be the same shape as θ, with zeros.

**Step 2: Gradient Calculation**

Compute the gradient of the loss function with respect to each parameter, denoted as $\nabla\theta J(\theta)$.

**Step 3: Squared Gradients Accumulation**

Update the sum of squared gradients G for each parameter i:

$G[i] \mathrel{+}= (\nabla\theta J(\theta[i]))^2$.

**Step 4: Parameter Update**

Update each parameter using the adaptive learning rate:

$\theta[i] \mathrel{-}= (\eta / (\sqrt{(G[i])} + \epsilon)) * \nabla\theta J(\theta[i])$.

In these equations, $\eta$ represents the learning rate, and $\nabla\theta J(\theta[i])$ is the gradient of the loss function with respect to the parameter $\theta[i]$.

### *1.3.3 Pros and cons*

**Pros:**

- AdaGrad is suitable for parameters with sparse gradients.
- The algorithm adjusts the learning rate for each parameter based on the sum of squared gradients observed during training.

**Cons:**

- A major drawback of AdaGrad is the accumulation of squared gradients in the denominator. Since each added term is positive, the accumulated sum continues to grow during training, causing the learning rate to shrink and become infinitesimally small.
- AdaGrad requires a manually selected global learning rate.
- AdaGrad may not perform well with large datasets or high-dimensional spaces.

## 1.4 Root Mean Square Propagation (RMSProp)

### *1.4.1 RMSProp Algorithm*

RMSProp is an adaptive learning rate optimization algorithm that enhances convergence speed by maintaining an exponentially weighted average of the squared gradients from previous iterations. This average is used to adjust the learning rate by dividing it, resulting in faster convergence.

In RMSProp, the algorithm keeps track of the historical information about gradients through an exponentially weighted moving average. By considering the squared gradients, RMSProp emphasizes the importance of parameters with larger gradients and dampens the impact of those with smaller gradients. This adaptive learning rate scheme allows the algorithm to effectively navigate complex optimization landscapes and converge more quickly.

By dividing the learning rate by the square root of the averaged squared gradients, **RMSProp** ensures that the learning rate diminishes for parameters with larger gradients while maintaining a relatively larger learning rate for parameters with smaller gradients. This adaptive adjustment enables the algorithm to converge faster and improve the efficiency of model training.

Mathematical formula:

$$s_{dW} = \beta s_{dW} + (1-\beta)\left(\frac{\partial J}{\partial W}\right)^2$$

$$W = W - \alpha \frac{\frac{\partial J}{\partial W}}{\sqrt{s_{dW}^{corrected}} + \epsilon}$$

where

- $s_{dW}$ the exponentially weighted average of past squares of gradients
- $\frac{\partial J}{\partial W}$ cost gradient with respect to current layer weight tensor
- W: tensor weight.
- $\beta$ hyperparameter to be tuned.

- $\alpha$ the learning rate.
- $\epsilon$ very small value to avoid dividing by zero.

### *1.4.2 Pros and cons*

**Pros**

- Adaptive Learning Rates: RMSProp adjusts the learning rate for each parameter, enabling it to handle data with different scales and loss functions with varying curvatures.
- Faster Convergence: Compared to Stochastic Gradient Descent (SGD) with momentum, RMSProp has the potential to converge faster, especially in situations with noisy or sparse gradients.
- Robustness: RMSProp is a robust optimizer that incorporates pseudo-curvature information, enhancing its stability and performance.
- Handling Stochastic Objectives: RMSProp is well-suited for dealing with stochastic objectives, making it applicable to scenarios involving mini-batch learning.
- Fewer Hyperparameters: Unlike some other optimization algorithms, RMSProp has fewer hyperparameters, making it easier to tune and implement in practical applications.

**Cons of RMSProp:**

- Setting the learning rate too high can lead to unstable convergence or overshooting the optimal solution, while setting it too low may result in slow convergence or getting stuck in suboptimal solutions.
- Tuning the decay rate is crucial for balancing the exploration-exploitation trade-off.
- Properly adjusting these hyperparameters is essential to ensure the effectiveness of RMSProp in achieving optimal performance.

## 1.5 Adam

### *1.5.1 Introduction*

The Adam optimizer, also known as “Adaptive Moment Estimation,” is a repetitive optimization method utilized to reduce the loss function while training neural networks. It can be viewed as a fusion of RMSprop and Stochastic Gradient Descent with momentum. Created by Diederik P. Kingma and Jimmy Ba in 2014, Adam has emerged as a preferred option for numerous practitioners in the field of machine learning.

### *1.5.2 Adam algorithm*

Adam optimization is a method that integrates the advantages of two other optimization techniques - Momentum and RMSProp. The Momentum technique utilizes the preceding gradient to stabilize variations during the optimization process, whereas RMSProp adjusts the learning rate in accordance with the size of the recent gradients. Adam optimization enhances these concepts by calculating an exponential moving average of the gradients and their squares, which allows for the adaptive adjustment of learning rates.

### *1.5.3 How Adam algorithm work*

Adam, which stands for Adaptive Moment Estimation, is a method that adjusts the learning rate for each parameter individually. It uses estimates of the first and second moments of the gradient to modify the learning rate for each weight in the neural network. In practical terms, Adam optimization maintains a running average of both the gradients and their squares for each model parameter. These averages are then used to determine the update for each parameter during the training process. The update rule for Adam optimization can be expressed as follows:

1. Compute the first moment (m):

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

2. Compute the second moment (v):

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

3. Compute bias-corrected first moment estimate

$$\widehat{m_t} = \frac{m_t}{1 - \beta_1^t}$$

4. Compute bias-corrected second raw moment estimate:

$$\widehat{v_t} = \frac{v_t}{1 - \beta_2^t}$$

5. Update the weights:

$$\theta_t = \theta_{t-1} - \alpha \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon}$$

Where:

- $m_t$and $v_t$ are the first and second moments at time $t$
- $\widehat{m_t}$ and $\widehat{v_t}$ are the bias-corrected first and second moments at time $t$
- $\theta_t$ is the weight at time $t$
- $\alpha$ is the learning rate.
- $\beta_1$ and $\beta_2$ are parameters that determine the weight of the first and second moments.
- $\epsilon$ is a small positive number to avoid division by 0.

Advantages/Pros

- Adaptive Learning Rates: Unlike other optimization algorithms with fixed learning rates, Adam adjusts the learning rates for each parameter based on their gradient history. This adaptive feature enables faster and more accurate convergence, particularly in high-dimensional parameter spaces.

- Momentum: Adam utilizes momentum to stabilize fluctuations during the optimization process. This helps the optimizer avoid getting trapped in local minima and saddle points, leading to improved performance.
- Bias Correction: Adam incorporates bias correction for the first and second moment estimates. This correction ensures that the estimates are unbiased representations of the true values, enhancing the accuracy of the optimization process.
- Robustness: Adam optimization exhibits robustness, meaning it performs well across a wide range of deep learning architectures. It is relatively insensitive to the choice of hyperparameters, making it a versatile and reliable optimization algorithm.

Disadvantages:

- Sensitivity to learning rate: Adam can be sensitive to the choice of learning rate. Setting the learning rate too high can lead to oscillations or divergence, while setting it too low can result in slow convergence. Finding the appropriate learning rate for a specific problem can be challenging.
- Memory requirements: Adam requires additional memory to store the exponentially decaying average of past squared gradients and the exponentially decaying average of past gradients. This increased memory usage can become a limitation when working with large-scale models or limited computational resources.
- Lack of theoretical guarantees: Unlike some other optimization algorithms, such as stochastic gradient descent (SGD), Adam does not have strong theoretical convergence guarantees. While it often performs well in practice, the lack of theoretical guarantees can make it more difficult to reason about the algorithm's behavior in certain scenarios.

- Dependency on hyperparameters: Adam has several hyperparameters that need to be tuned, such as the learning rate, momentum, and decay rates for the moving averages. Finding the optimal values for these hyperparameters can require extensive experimentation and may be problem specific.
- Performance on non-stationary objectives: Adam may not perform as well on non-stationary objectives or in situations where the objective function changes over time. The adaptive learning rates and momentum in Adam may hinder its ability to adapt quickly to sudden changes in the optimization landscape.

# CHAPTER 2. CONTINUAL LEARNING AND TEST PRODUCTION: BUILDING MACHINE LEARNING SOLUTIONS FOR PROBLEM-SOLVING

## 2.1 Continual learning

### *2.1.1 introduction*

Continuous learning, a concept within machine learning, allows models to learn from new data streams iteratively, thereby adapting to dynamic environments. Unlike traditional approaches that rely on static datasets, continuous learning models update their parameters in response to changing data distributions.

### *2.1.2 Types of Continuous Machine Learning Approaches*

Continuous machine learning encompasses various approaches, including:

- Incremental learning: Models update their parameters using new data instances as they arrive, allowing them to adapt to changing distributions.
- Transfer learning: Knowledge gained from one task or domain is transferred to another, enabling models to leverage prior learning and accelerate new learning.
- Lifelong learning: Models learn continuously throughout their lifetime, accumulating knowledge and adapting to new contexts.

### *2.1.3 The Continuous Learning Process*

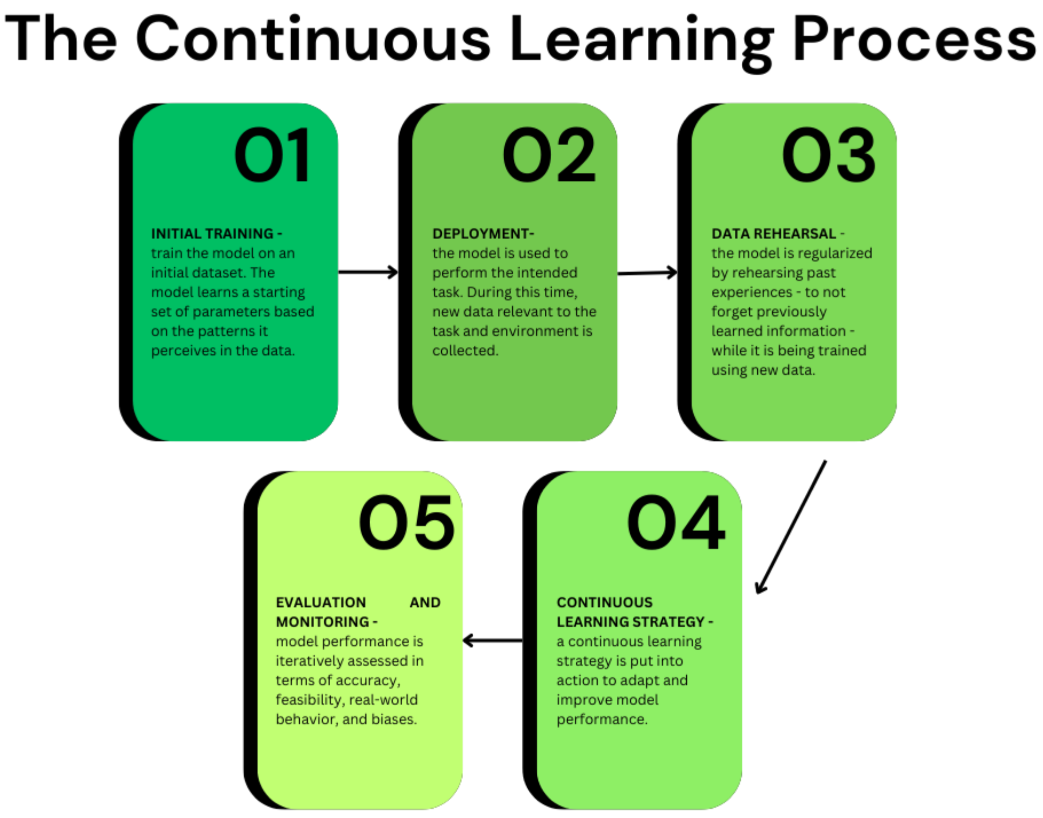

The Continuous Learning Process
01
INITIAL TRAINING - train the model on an initial dataset. The model learns a starting set of parameters based on the patterns it perceives in the data.
02
DEPLOYMENT- the model is used to perform the intended task. During this time, new data relevant to the task and environment is collected.
03
DATA REHEARSAL - the model is regularized by rehearsing past experiences - to not forget previously learned information - while it is being trained using new data.
04
CONTINUOUS LEARNING STRATEGY - a continuous learning strategy is put into action to adapt and improve model performance.
05
EVALUATION AND MONITORING - model performance is iteratively assessed in terms of accuracy, feasibility, real-world behavior, and biases.

Figure 3 The Continuous Learning Process

Figure 4 Flowchart of the Continuous Learning Process to Machine Learning

1. The continuous learning process builds upon traditional machine learning principles and involves additional steps:
2. Pre-processing: Data is prepared and transformed to ensure compatibility with the continuous learning approach.
3. Model selection: Suitable models are chosen based on the specific requirements of the task at hand.
4. Hyperparameter tuning: Parameters of the selected model are optimized to achieve desired performance.
5. Training: Models are trained on initial data and updated iteratively as new data streams become available.
6. Deployment: The trained models are deployed for use in real-world applications.

7. Monitoring: Models are continuously monitored for performance evaluation and adaptation.

### *2.1.4 Advantages of Continuous Learning*

Generalization: Continuous learning models are more robust and accurate in the face of new data.

Retention of information: Models retain previous knowledge gained in past iterations, allowing them to accumulate information over time.

Adaptability: Models adapt to new knowledge, such as concept drift and emerging trends, improving predictive capabilities in the long run.

### *2.1.5 Limitations of Continuous Learning*

While continuous learning brings significant benefits, it also has limitations that should be considered:

Cost: Continuous learning approaches can be computationally complex, requiring additional data, human, and computing resources.

Model management: Continuous learning may generate many models, making it challenging to identify the best-performing ones.

Data drift: Models may lose predictive capabilities if the feature distribution changes abruptly, necessitating careful monitoring and intervention.

### *2.1.6 Applications of Continuous Learning*

Continuous learning finds applications in domains where a constant stream of new data is present:

Computer vision: Continuous learning is applied to train algorithms for image classification and facial recognition.

Cybersecurity: Continuous learning aids in detecting security threats, such as phishing and network intrusion.

Healthcare: Continuous learning enhances diagnostic workflows in fields like oncology and radiology, adapting to evolving disease patterns.

Robotics: Continuous learning enables robots to optimize their actions in changing environments based on past and new experiences.

## 2.2 Test Production

### *2.2.1 Test production introduction*

Test production refers to the process of creating and implementing tests or evaluations to assess the performance and effectiveness of a machine learning model or solution. It involves designing test cases, generating test data, and evaluating the model's predictions against the ground truth or expected outcomes.

### *2.2.2 The Crucial Role of Test Production in Constructing Machine Learning Solutions*

When constructing a machine learning solution, test production plays a crucial role in several aspects:

1. Performance Evaluation: Test production helps measure the performance of the model by assessing its accuracy, precision, recall, F1-score, or other evaluation metrics. It allows researchers and practitioners to quantitatively assess the model's ability to generalize and make accurate predictions on unseen data.
2. Model Selection and Comparison: By conducting rigorous testing, different machine learning models or algorithms can be compared and evaluated based on their performance. Test production aids in selecting the best model or technique that suits the problem at hand.
3. Hyperparameter Tuning: Machine learning models often have hyperparameters that need to be tuned for optimal performance. Test production can be used to evaluate the model's performance across different hyperparameter settings, helping in the process of finding the best configuration.
4. Robustness and Generalization: Test production allows for the assessment of a model's robustness and generalization capabilities. By

evaluating the model's performance on diverse and representative test datasets, researchers can assess how well the model can handle variations, noise, or unseen data.

5. Deployment and Monitoring: Test production is also crucial in the deployment phase of a machine learning solution. It helps ensure that the model performs as expected in a real-world scenario. Furthermore, ongoing testing and monitoring can help identify any degradation in performance over time, prompting necessary adjustments or retraining.

## REFERENCES

[1] Article What is Continuous Learning? Revolutionizing Machine Learning & Adaptability | DataCamp