VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**MACHINE LEARNING**

**FINAL PROJECT**

*Supervisor*: **Assoc. Prof.** **LE ANH CUONG**

*Author*: **PHAN MINH HOANG – 521H0501**

Class **: 21H50302**

Course  **: 25**

**HO CHI MINH CITY, 2023**

VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**MACHINE LEARNING**

**FINAL PROJECT**

Supervisor: **Assoc. Prof.** **LE ANH CUONG**

Author: **PHAN MINH HOANG**

Class **: 21H50302**

Course  **: 25**

**HO CHI MINH CITY, 2023**

ACKNOWLEDGMENTS

Thank you to Assoc. Prof. Le Anh Cuong for guiding me through the knowledge of Discrete Structures course, enabling me to complete this essay

**THESIS WAS COMPLETED**

**AT TON DUC THANG UNIVERSITY**

I hereby certify that this is my thesis project and was conducted under the guidance of Assoc. Prof. Le Anh Cuong. The research contents and results presented in this thesis are honest and have not been published in any form before. The data used for analysis, comments, and evaluations were collected by the author from various sources which are clearly indicated in the reference section

Furthermore, this thesis also includes some comments, evaluations, and data from other authors and organizations which are properly cited and referenced

**If any fraudulent activity is detected, I take full responsibility for the content of my thesis.** Ton Duc Thang University is not liable for any copyright infringement or violation caused by me during the implementation process (if any)

*Ho Chi Minh City, December 23, 2023*

*Author*

* (Signature and full name)*

*Phan Minh Hoang*

LECTURER ENDORSEMENTS AND REVIEWS

**Instructor endorsement**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ho Chi Minh City, day month year

(Signature and full name)

**Teacher evaluation**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ho Chi Minh City, day month year

(Signature and full name)

SUMARY

In this research, we will study the Optimizer methods in training machine learning models. We also focus on learning about Continual Learning and Test Production when building a machine learning solution to solve a certain problem

TABLE OF CONTENTS

[ACKNOWLEDGMENTS i](#_Toc154256592)

[LECTURER ENDORSEMENTS AND REVIEWS iii](#_Toc154256593)

[SUMARY iv](#_Toc154256594)

[TABLE OF CONTENTS 1](#_Toc154256595)

[LIST OF SYMBOLS AND ABBREVIATIONS 3](#_Toc154256596)

[TABLE OF FIGURES 4](#_Toc154256597)

[CHAPTER 1 – OPTIMIZER 5](#_Toc154256598)

[1.1 What is Optimizer? 5](#_Toc154256599)

[1.2 Optimizer methods 6](#_Toc154256600)

[*1.2.1 Gradient Descent* 6](#_Toc154256601)

[*1.2.2 Stochastic Gradient Descent (SGD)* 9](#_Toc154256602)

[*1.2.3 Momentum* 12](#_Toc154256603)

[*1.2.4 AdaGrad, RMSProp, and Adam* 14](#_Toc154256604)

[1.3 Compare all optimizer methods 17](#_Toc154256605)

[CHAPTER 2 – CONTINUAL LEARNING AND TEST PRODUCTION 19](#_Toc154256606)

[2.1 Continual learning 19](#_Toc154256607)

[*2.1.1 Regularization techniques* 20](#_Toc154256608)

[*2.1.2 Memory replay* 20](#_Toc154256609)

[*2.1.3 Regularization techniques* 20](#_Toc154256610)

[*2.1.4 Lifelong learning approaches* 21](#_Toc154256611)

[*2.1.5 Conclusion* 21](#_Toc154256612)

[2.2 Test Production 22](#_Toc154256613)

[*2.2.1 The step of Test Production* 22](#_Toc154256614)

[*2.2.2 Evaluation Test Production* 22](#_Toc154256615)

[References 25](#_Toc154256616)

LIST OF SYMBOLS AND ABBREVIATIONS

**SYMBOLS**

**ABBREVIATIONS**

GD Gradient Descent

SGD Stochastic Gradient Descent

TABLE OF FIGURES

[Figure 1 – Illustration about gradient descent work 6](#_Toc154256628)

[Figure 2 Compare GD and SGD 9](#_Toc154256629)

[Figure 3 Compare ordinary SGD with SGD with momentum 12](#_Toc154256630)

CHAPTER 1 – OPTIMIZER

1.1 What is Optimizer?

An optimizer is the crucial player in a machine learning model's quest for optimal performance. Just like a skilled sculptor shaping a masterpiece, the optimizer molds the model's internal parameters, represented by "weights" and "bias," to best fit the "features" of the input data. This fine-tuning leads to a dramatic improvement in the model's accuracy.

Here are some terms related to Optimizer:

* Epoch: The number of times the algorithm runs over the entire training data.
* Sample: A row of the dataset.
* Batch: The number of samples taken to update the parameters for the model.
* Learning rate: A parameter that helps the model update the weight levels appropriately.
* Cost Function/Loss Function: A function used to calculate the discrepancy between the predicted value and the actual value.
* Weights: These are values that the model learns from the input data. They determine the extent to which each input feature influences the prediction outcome.
* Bias: This is a linear component applied to the input information. Adding it to the result of the weight multiplication with the input results in a linear function passing through the origin. Using “bias” allows the neural network to flexibly shift the linear function.

1.2 Optimizer methods

Just like there's no "one-size-fits-all" recipe for success, there's no single best optimizer for every machine learning task.

*1.2.1 Gradient Descent*

In the vast landscape of Optimizer methods, Gradient Descent (GD) stands as the granddaddy of them all. It's the simplest, most intuitive algorithm, and its influence on the field is undeniable. But what exactly makes GD tick, and how does it contribute to the optimization process?

Imagine a mountainous landscape, where each point represents a different configuration of the model's parameters (weights and bias). The goal is to find the valley, the point where the model performs best. GD takes a straightforward approach: it rolls a ball down this landscape, always moving in the direction that steepest reduces the cost function (the measure of model error).

***The Formula for Gradient Descent:***

where:

* : is the current point.
* : is the learning rate or step size.
* : is the gradient of the function at the current point.
* :​ is the next point.

***Step by step of how gradient descent working:***

* Initialize Weights and Bias values: Start from a random point in the parameter space
* Calculate the Gradient (derivative) of the Loss Function at the randomly chosen point
* Update the parameters by moving in the direction opposite to the gradient. The size of the step (learning rate) determines the learning speed of the algorithm.
* Repeat the above process until a stopping point is reached (usually when the gradient is close to 0 or after a certain number of iterations)

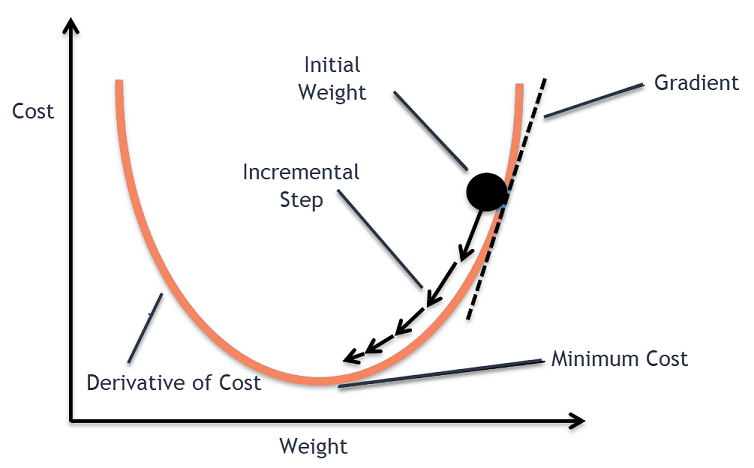


Figure 1 – Illustration about gradient descent work

***Strengths and Weaknesses:***

GD boasts several advantages:

* Simplicity: Easy to understand and implement, making it a great starting point for beginners.
* Stability: Converges reliably to a minimum, even if not always the global minimum.
* Transparency: The downhill analogy provides a clear visual understanding of the optimization process.

However, GD also has limitations:

* Slowness: Can be sluggish, especially for complex problems with many parameters.
* Stuck in Valleys: May get trapped in local minima, missing out on the true optimal solution.
* Sensitive to Learning Rate: Choosing the right learning rate is crucial; too high can lead to instability, while too low can slow down learning.

***GD's Legacy:***

Despite its limitations, GD's influence on Optimizer development is immense. It serves as the foundation for many advanced algorithms, including:

* Stochastic Gradient Descent (SGD): A faster variant that samples data randomly, sacrificing smoothness for speed.
* Momentum: Adds inertia to the downhill roll, helping escape local minima and speeding up convergence.
* Adaptive Optimizers: Dynamically adjust learning rates for different parameters, leading to more efficient optimization.

*1.2.2 Stochastic Gradient Descent (SGD)*

While Gradient Descent (GD) takes a meticulous approach, carefully calculating the gradient over the entire dataset before each update, its sibling, Stochastic Gradient Descent (SGD), prefers a faster, more adventurous path.

Instead of processing the entire dataset at each step, SGD randomly samples small subsets of data, called mini-batches, and uses their gradients to update the parameters. This introduces noise but also leads to significant speed gains, especially for large datasets.

***The Formula for SGD:***

where:

* represents the parameters (weights) of the model.
* is the learning rate.
* is the gradient of the cost function J with respect to the parameters

***Step by step of how stochastic gradient descent working:***

* Starting Point: SGD begins at the same initial point as GD.
* Sampling a Mini-Batch: A random subset of data points is selected from the dataset.
* Calculating the Gradient: The gradient is calculated only for this mini-batch, providing an estimate of the overall direction.
* Taking a Step: Based on this partial gradient and the learning rate, SGD takes a step in the estimated downhill direction.
* Repeat and Resample: The process repeats, but each time with a new mini-batch, leading to a jittery, but ultimately efficient, descent towards the valley.

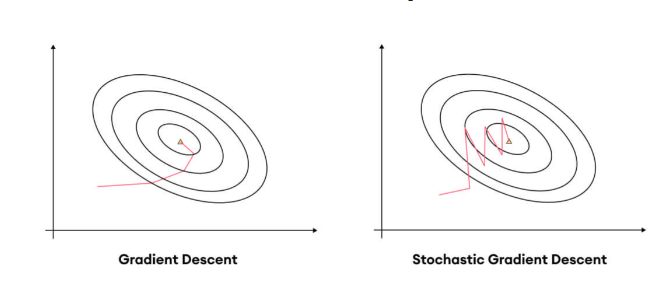


Figure 2 Compare GD and SGD

***Strengths and Weaknesses:***

SGD offers several benefits:

* Speed: Significantly faster than GD, especially for large datasets.
* Online Learning: Can update model parameters as new data arrives, enabling real-time applications.
* Escaping Local Minima: The randomness can sometimes help SGD escape local minima, finding better solutions.

However, it has trade-offs:

* Noisier Convergence: The path is less smooth and may not reach the exact minimum.
* Learning Rate Sensitivity: Choosing the right learning rate is even more crucial in SGD, as too high a rate can lead to instability and divergence.

***Adaptive SGD Variants:***

* Momentum: Adds inertia to the updates, helping smooth out the path and accelerate convergence.
* AdaGrad: Adaptively adjusts learning rates for different parameters, improving performance on sparse data.
* RMSprop: Addresses AdaGrad's potential for excessively diminishing learning rates.
* Adam: Combines momentum with adaptive learning rates, often considered a default choice for many deep learning tasks.

*1.2.3 Momentum*

In the world of optimizers, momentum is like a superpower that propels the optimization process forward, helping escape local minima and reach the true valley of optimal performance. It's a technique often combined with both Gradient Descent (GD) and Stochastic Gradient Descent (SGD) to enhance their efficiency and effectiveness.

Momentum introduces a memory component to the optimization process, incorporating past gradients into the current update. This creates a sense of inertia, making the updates smoother and less prone to getting stuck in local minimal

***The Formula for momentum:***

where:

* is the update to the weights at iteration.
* is the momentum term, typically a value between 0 and 1 that controls the influence of the accumulated previous updates.
* is the learning rate, determining the step size in the direction of the gradient
* is the gradient of the loss function with respect to the weights

***Step by step of how stochastic gradient descent working:***

* Starting Point: The ball starts at an initial point, just like in GD or SGD.
* Calculating Gradient and Momentum: At each step, both the gradient and momentum term are calculated.
* Updating Velocity: The momentum term is updated based on the previous momentum and the current gradient, adding inertia to the process.
* Taking a Step: The velocity is then used to update the ball's position, moving it further towards the valley.

***Strengths and Weaknesses:***

Benefits of Momentum:

* Smoother Updates: Momentum helps reduce oscillations and overshooting, leading to smoother convergence.
* Faster Convergence: The inertia often accelerates convergence, reaching the minimum faster.
* Escaping Local Minima: The momentum can help push the model out of shallow local minima, finding better solutions.

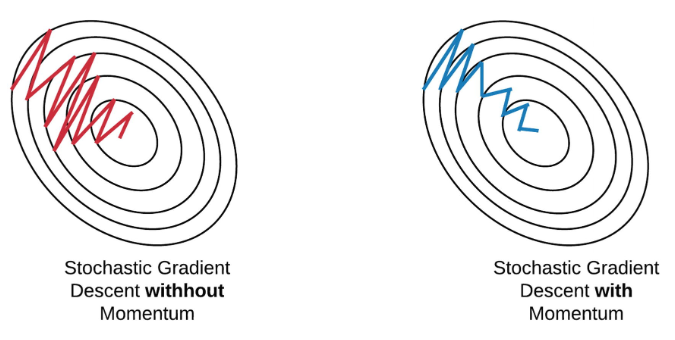


Figure 3 Compare ordinary SGD with SGD with momentum

*1.2.4 AdaGrad, RMSProp, and Adam*

While Gradient Descent, SGD, and Momentum establish a solid foundation for optimization, they have a limitation: they use a single learning rate for all parameters. Adaptive optimizers like AdaGrad, RMSProp, and Adam address this by customizing learning rates for each parameter, leading to more efficient and adaptive learning.

These optimizers individually track past gradients for each parameter, adjusting learning rates accordingly. This allows them to prioritize updates for less frequently changing parameters and slow down updates for frequently changing ones.

*1.2.4.1 AdaGrad*

AdaGrad is a pioneering adaptive optimizer that introduced the concept of parameter-specific learning rates. It addresses a common limitation of traditional optimizers like Gradient Descent and SGD, which use a single learning rate for all parameters. By adapting learning rates individually, AdaGrad can often improve convergence and performance, especially in certain scenarios.

***The Formula for AdaGrad:***

Where:

* is the value of the parameter at the current iteration.
* is the learning rate.
* is the sum of squares of past gradients up to time t
* is a small constant added for numerical stability.
* is the gradient at time t.

*1.2.4.2 RMSProp*

RMSProp builds upon the foundation of adaptive learning rates established by AdaGrad, but it addresses a key limitation: the potential for prematurely diminishing learning rates. By using a decaying average of squared gradients, RMSProp often achieves smoother and more consistent convergence, making it a popular choice in deep learning.

***The Formula for RMSProp:***

Where:

* is the accumulated squared gradient at time (t).
* is the decay rate for the moving average.
* is the gradient at time (t).
* is the parameter being updated.
* is the learning rate.
* is a small constant added for numerical stability.

*1.2.4.3 Adam*

Adam stands as one of the most popular and versatile optimizers in deep learning, often considered the default choice due to its consistent performance and ease of use. It effectively combines the strengths of both Momentum and RMSProp, leading to smoother convergence, better generalization, and reduced hyperparameter sensitivity.

The update rules for the parameters in Adam are as follows:

1. Update biased first moment estimate:

​

1. Update biased second raw moment estimate:
2. Compute bias-corrected first moment estimate:
3. Compute bias-corrected second raw moment estimate:
4. Update parameters:

​​

Where:

* is the parameter vector at timestep t.
* is the gradient at timestep (t).
* and are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively.
* and are bias-corrected estimates of the first and second moment.
* and are exponential decay rates for the moment estimates, typically set to 0.9 and 0.999 respectively.
* is the learning rate.
* is a small constant added for numerical stability, typically set to

1.3 Compare all optimizer methods

|  |  |  |  |
| --- | --- | --- | --- |
| Optimizer | Key Features | Advantages | Common Use Cases |
| ****Gradient Descent (GD)**** | -Uses mini-batches of data to estimate gradients  - Faster than GD for large datasets | - Simple and easy to understand  - Foundation for other optimizers | - Small datasets  - Linear models |
| ****Stochastic Gradient Descent (SGD)**** | -Adds inertia to updates, helping escape local minima  - Often combined with SGD or GD | - Speed  - Online learning | - Large datasets  - Deep learning |
| ****Momentum**** | - Adds inertia to updates, helping escape local minima - Often combined with SGD or GD | - Smoother updates - Faster convergence  - Escaping local minima | - Most deep learning tasks |
| ****AdaGrad**** | - Adapts learning rates for each parameter based on their historical gradients  - Prioritizes less frequently updated parameters | - Improved convergence for sparse data or noisy gradients | - Natural language processing - Recommendation systems  - Online learning |
| ****RMSProp**** | - Decaying average of squared gradients prevents excessively diminishing learning rates  - Smoother convergence than AdaGrad | - Smoother convergence  - Handles noisy gradients | - Deep neural networks - Reinforcement learning  - Generative models |
| ****Adam**** | - Combines momentum and adaptive learning rates like RMSProp  - Often the default choice in deep learning | - Fast convergence - Smooth updates  - Better generalization - Less sensitive to hyperparameters | - Most deep learning tasks  - Wide range of applications |

CHAPTER 2 – CONTINUAL LEARNING AND TEST PRODUCTION

2.1 Continual learning

Continual Learning is a field of machine learning that focuses on building models that can learn and adapt to new data without losing previously learned knowledge. When building a machine learning solution to solve a problem, the model’s ability to learn continuously should be considered. If the model cannot learn continuously, it may lead to the following issues:

* + Overfitting: The model might overfit to old data and be unable to learn from new data.
* Performance degradation: The model’s performance might decrease when exposed to new data.

Here are some techniques that can be used to improve the model’s ability to learn continuously:

* + Regularization Techniques: These techniques are used to prevent the model from overfitting and keep it stable. For example, the L2 regularization method subtracts a small part from the magnitude of the parameters to prevent the model from changing too much when learning new tasks.
  + Memory Replay: These techniques use memory to store examples from previous tasks. When the model is trained for new tasks, examples from memory can be used to update the model.
  + Parameter Distillation: These techniques use old models to train new models. The new models will learn important features from the old models.
  + Lifelong Learning Approaches: These methods are specially designed for continuous learning. These methods often use special architectures and training strategies.

*2.1.1 Regularization techniques*

Regularization techniques work by preventing the model from changing its parameters too much when learning new tasks. This helps the model retain previously learned knowledge and avoid overfitting.

Some popular regularization techniques include:

* L2 Regularization: This is the most common regularization technique. L2 regularization subtracts a small part from the magnitude of the model’s parameters.
* L1 Regularization: This technique is similar to L2 regularization, but it subtracts a small part from the absolute value of the model’s parameters.
* Dropout: This technique randomly omits some connections between neurons in the model. This helps the model avoid over-dependence on specific connections.

*2.1.2 Memory replay*

Memory replay is a technique that uses memory to store examples from previous tasks. When the model is trained for new tasks, examples from memory can be used to update the model.

Memory replay helps the model retain previously learned knowledge by providing the model with examples from previous tasks to learn from.

*2.1.3 Regularization techniques*

Parameter distillation is a technique that uses old models to train new models. The new models will learn important features from the old models.

Parameter distillation helps the model retain previously learned knowledge by transferring knowledge from old models to new models.

*2.1.4 Lifelong learning approaches*

Lifelong learning approaches are specially designed methods for continuous learning. These methods often use special architectures and training strategies.

Some popular lifelong learning approaches include:

* + Incremental Learning: These methods update the model gradually as it encounters new data.
  + Dynamic Architecture Learning: These methods allow the model to change its architecture when learning new tasks.
  + Meta-Learning: These methods allow the model to learn how to learn new tasks more efficiently.

*2.1.5 Conclusion*

The technique used to improve the model’s ability to learn continuously depends on several factors, including:

* Type of Problem: Some problems are more suitable for specific techniques than others.
* Size of the Dataset: Continual Learning (CL) models may require more data than regular machine learning models.
* Computational Capabilities: CL models may require more computational resources than regular machine learning models.

Choosing the appropriate technique will help improve the model’s ability to learn continuously and achieve better performance on specific tasks. It’s important to consider these factors when selecting a technique for Continual Learning.

2.2 Test Production

Test production refers to the process of simulating a real-world production environment for testing and evaluating machine learning models before actual deployment. It's essentially a crucial step to ensure your model functions effectively and handles real-world data and scenarios confidently.

*2.2.1 The step of Test Production*

The steps of Test Production typically include the following:

* Data Preparation: The data used for test production needs to be real-world data, representative of the data that the model will encounter in the production environment. Data can be collected from sources such as production systems, historical data, or synthesized data.
* Creating a Production Environment: A production environment needs to be created to simulate the actual environment in which the model will operate. The environment includes hardware, software, and network.
* Model Training: The model is trained on the data prepared in step 1.
* Model Testing: The model is tested on real-world data or synthesized data. Tests include performance, accuracy, and reliability checks.
* Model Adjustment: The model can be adjusted based on the test results.

The specific steps of Test Production can vary depending on the type of model and production environment. However, the above steps are fundamental and necessary for any Test Production process.

*2.2.2 Evaluation Test Production*

Evaluation in Test Production is an important process to ensure that machine learning models deployed in the production environment perform well and meet user requirements. Test Production evaluation can be performed using quantitative metrics, qualitative indicators, and other evaluation metrics. This process helps to identify any issues or improvements needed before the model is fully deployed. It’s a critical step in maintaining the quality and reliability of machine learning systems in real-world applications.

Evaluation techniques in Test Production:

\_**Quantitative Evaluation**: Quantitative evaluation uses statistical metrics to evaluate the performance of the model. Common statistical metrics include:

* Accuracy: The ratio of correct predictions.
* Precision: The ratio of correct positive predictions among all positive predictions.
* Recall: The ratio of positive predictions that were correctly predicted.
* F1-score: A combination of precision and recall.

\_**Qualitative Evaluation**: Qualitative evaluation uses text descriptions to evaluate the performance of the model. Factors often considered in qualitative evaluation include:

* Reliability of Prediction: How much can the model trust its prediction?
* Explainability of Prediction: Can the model explain why it made that prediction?
* Alignment with Objective: Does the model meet the requirements of the objective?

\_**Other Evaluation Metrics**: In addition to the above metrics, other evaluation metrics can be used to evaluate Test Production:

* Training Time: The time needed to train the model.
* Prediction Time: The time needed for the model to make a prediction.
* Resource Usage: The amount of computational resources needed to run the model.

Some considerations when evaluating in Test Production:

* Frequent Evaluation: The production environment is always changing, so it’s necessary to evaluate the model regularly to ensure it continues to perform well.
* Evaluation by Experts: Evaluation is a complex process that requires the involvement of experts to ensure accurate and reliable results.
* Use Evaluation Results for Improvement: Evaluation results are important information for improving the model. Therefore, evaluation results should be used to adjust the model, ensuring it performs better in the production environment.

# **References**

|  |  |
| --- | --- |
| [1] | Y. B. a. A. C. Ian Goodfellow, Deep Learning Book, MIT Press, 2016. |
| [2] | A. Vidhya, "A Comprehensive Guide on Optimizers in Deep Learning," [Online]. Available: https://www.analyticsvidhya.com/blog/2021/03/optimizers-in-deep-learning. |
| [3] | Musst, "Optimizers in Deep Learning. What is an optimizer?," medium, [Online]. Available: https://medium.com/dataseries/optimizers-in-deep-learning-what-is-an-optimizer-e0c051bc5f54. [Accessed 23 12 2023]. |
| [4] | D. Community, "10 famous Machine Learning Optimizers," [Online]. Available: https://dev.to/srushti1254/10-famous-machine-learning-optimizers-54a8. [Accessed 23 12 2023]. |
| [5] | Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Pascal Vincent, "On the importance of initialization and momentum in deep learning," *International Conference on Machine Learning (ICML),* 2013. |
| [6] | John Duchi, Elad Hazan, and Yoram Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," 2011. |
| [7] | A. Ng, "RMSProp and Adam: Optimizers for Deep Learning," [Online]. Available: https://www.coursera.org/lecture/deep-neural-network/adam-optimization-algorithm-w9VCZ. [Accessed 23 12 2023]. |
| [8] | Optimizely, "What is 'testing in production'?," 20 07 2022. [Online]. Available: https://www.optimizely.com/optimization-glossary/testing-in-production/. |
| [9] | N. T. Đạt, "Tại sao bạn nên kiểm thử trên môi trường Production?," 20 07 2023. [Online]. Available: https://viblo.asia/p/tai-sao-ban-nen-kiem-thu-tren-moi-truong-production-WAyK8oOm5xX. |