VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

NGUYEN THI YEN NHI – 520H036

FINAL REPORT INTRODUCTION TO MACHINE LEARNING

HO CHI MINH CITY, 2023

VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

NGUYEN THI YEN NHI – 520H036

FINAL REPORT INTRODUCTION TO MACHINE LEARNING

Instructor
PGS.TS LE ANH CUONG

HO CHI MINH CITY, 2023

THANK YOU

I would like to express my sincere thanks to PGS.TS Le Anh Cuong and the Faculty of Information Technology for supporting and helping me a lot in the process of completing the final report. During task performance. has certain advantages and disadvantages. Fortunately, those difficulties were promptly supported by the knowledge that the Teacher imparted through lessons in class for us to recognize and overcome. The report during the writing process will have shortcomings.

We look forward to receiving your contributions so we can identify and make changes for improvement.

We would like to send our sincere thanks to the teachers and the Faculty of Information Technology!

Ho Chi Minh City, day ... month ... year 2023
Author
(Sign and write your full name)

Nguyen Thi Yen Nhi

WORK IS COMPLETED AT TON ĐUC THANG UNIVERSITY

I hereby declare that this is my own research project and is under the scientific guidance of PGS.TS Le Anh Cuong. The research content and results in this topic are honest and have not been published in any form before. The data in the tables for analysis, comments, and evaluation were collected by the author from different sources and clearly stated in the reference section.

In addition, the Project also uses a number of comments, assessments as well as data from other authors and other organizations, all with citations and source notes.

If any fraud is detected, I will take full responsibility for the content of my Project. Ton Duc Thang University is not involved in copyright violations caused by me during the implementation process (if any).

Ho Chi Minh City, day ... month ... year 2023

Author

(Sign and write your full name)

Nguyen Thi Yen Nhi

TABLE OF CONTENTS

LIST OF DRAWINGS

LIST OF ABBREVIATIONS

GD	Gradient descent
SGD	Stochastic gradient descent
NAG	Nesterov accelerated gradient

CHAPTER 1. OPTIMIZER IN TRAINING MACHINE LEARNING

1.1 What is Optimizer?

The optimization algorithm is the basis for building a neural network model with the purpose of "learning" the features (or patterns) of the input data, from which a suitable pair of weights and biases can be found to optimize the model.

When training a machine learning model, you must adapt every epoch's weight and minimize the loss function. An optimizer is an algorithm or function that adapts the neural network's attributes, like learning rate and weights. Hence, it assists in improving the accuracy and reduces the total loss. But it is a daunting task to choose the appropriate weights for the model.

1.2 Optimizer method in training machine learning models

1.2.1 Gradient descent (GD)

This is the most basic optimizer that directly uses the derivative of the loss function and the learning rate to reduce the loss and achieve the minimum. This approach is also applied in backpropagation in neural networks where the update parameters are shared among different layers depending on when the minimum loss is achieved. The GD optimization algorithm uses calculus to vary the values and consistently achieve a local minimum.

The formula of the gradient descent algorithm is as follows:

$$a\_new = x - alpha * f'(a)$$

- Advantage:
  - GD algorithm is one of the easy-to-understand.
  - The algorithm has solved the problem of optimizing the neural network model by updating the weights after each loop.
- Disadvantage:

- Because it is simple, the gradient descent algorithm has many limitations such as depending on the initial initial solution and learning rate.
- A function with 2 global minimums will depend on the initial initialization point and will produce 2 different final solutions.
- Too high a learning rate will cause the algorithm to fail to converge.
- Small learning rate affects training speed.

1.2.2 Stochastic gradient descent (SGD)

In this algorithm, at one time, we only calculate the derivative of the loss function based on just one data point and then update θ based on this derivative. This is done for each point across the entire data, then the above process is repeated. This very simple algorithm actually works very well.

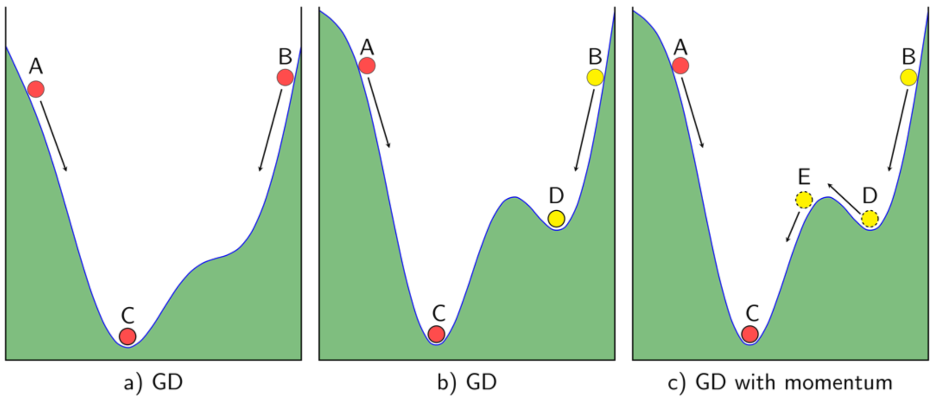


Figure 1.2.2: Compare gradients with physical phenomena

- Advantage:
  - The algorithm can handle large databases that GD cannot. This optimization algorithm is still commonly used today.
- Disadvantage:
  - The algorithm has not yet solved the two major disadvantages of gradient descent (learning rate, initial data points). So we have to combine SGD with some other algorithms such as: Momentum, AdaGrad,..

1.2.3 Momentum

Momentum is a gradient descent optimization method that adds a percentage of the previous update vector to the current update vector to speed up the learning process. Essentially, momentum is a method of smoothing out model parameter updates and allowing the optimizer to continue advancing in the same direction as before, minimizing oscillations and speeding up convergence. Momentum can be more accurately described as an exponentially weighted moving average of previous gradients. Instead of updating the parameters with the current slope, the optimizer uses an exponentially weighted moving average of the previous slope. The exponentially weighted moving average serves as a memory for the optimizer, allowing it to remember the direction it has moved and continue to follow that path even if the current slope points in the same direction. another direction.

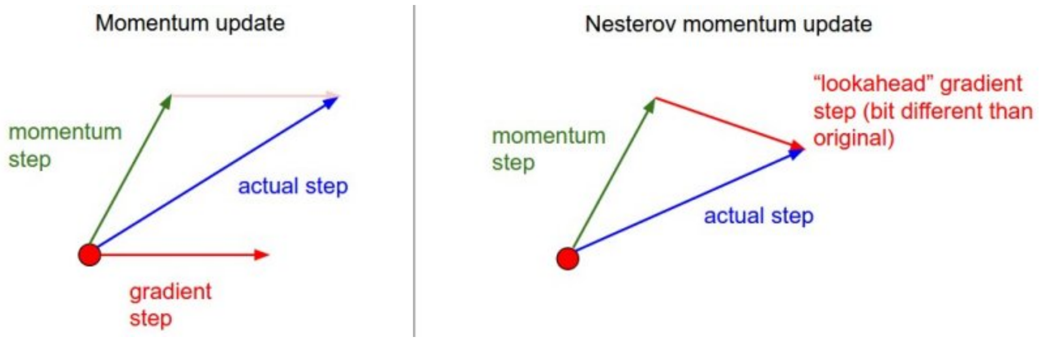


Figure 1.2.3: Nesterov's idea is accelerated gradient

- Advantage:
  - The optimization algorithm solves the problem: Gradient Descent does not reach the global minimum point but only stops at the local minimum.
- Disadvantage:
  - Although momentum helps the marble climb uphill to reach the destination, when it gets close to the destination, it still takes a lot of time to oscillate back and forth before stopping completely, this is explained because the marble has momentum.

1.2.4 Adagrad

Unlike previous algorithms where the learning rate is almost the same during the training process (learning rate is constant), Adagrad considers learning rate as a parameter. That is, Adagrad will let the learning rate change after each time t.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

n: constant

gt: gradient at time t

ϵ: error avoidance factor (divided by sample equals 0)

G: is a diagonal matrix where each element on the diagonal (i, i) is the square of the parameter vector derivative at time t.

- Advantage:

  - An obvious benefit of Adagrad is to avoid adjusting the learning rate manually, just set the default learning rate to 0.01 and the algorithm will automatically adjust.

- Disadvantage:

  - The weakness of Adagrad is that the sum of squared variations will grow larger over time until it makes the learning rate extremely small, causing training to freeze.

  - Can lead to excessive learning rate reduction, slowing down the learning process.

1.2.5 RMSprop

RMSprop solves Adagrad's decreasing learning rate problem by dividing the learning rate by the average of the squares of the gradient.

- Advantage:

  - RMSprop solves the problem of Adagrad's gradually decreasing learning speed (the problem of gradually decreasing learning speed over time will cause training to slow down, possibly leading to freezing).

- Disadvantage:

- The RMSprop algorithm can only give a solution that is local minimum but not global minimum like Momentum. Therefore, people will combine both Momentum algorithms with RMSprop to create an Adam optimal algorithm. We will present it in the following section.

1.2.6 Adam

Adam is an optimization algorithm that can be used instead of the classic stochastic gradient descent process to iteratively update network weights based on training data.

Adam is combining the advantages of two other extensions of stochastic gradient descent.

Special:

- The adaptive gradient algorithm (AdaGrad) maintains a per-parameter learning rate that improves performance on problems with sparse gradients.

- Root mean square propagation (RMSProp) also maintains a per-parameter learning rate that is adjusted based on the average of the recent magnitude of the gradient for the weights.

- Advantage:

  - Adam is relatively easy to configure in that the default configuration parameters work well on most problems.

  - Adam uses an adaptive learning rate technique that can adjust the learning rate depending on the magnitude of the gradient.

- Disadvantage:

  - Many hyperparameters need to be configured, and require memory as it needs to store additional information such as a history of the gradient.

  - Not the best choice for every problem.

1.2.7 Adadelta

Adadelta can be considered a further extension of gradient descent that builds upon AdaGrad and RMSProp and changes the calculation of the custom step size so that the units are consistent and in turn no longer requires an initial learning rate hyperparameter. The Adadelta algorithm is one of several optimization methods that are commonly used in conjunction with various learning algorithms, including supervised and unsupervised methods. The use of Adadelta has been shown to be effective in a variety of applications, including image and speech recognition, natural language processing, and computer vision.

- Advantage:
  - Has faster convergence speed and works well on sparse data.
  - Adadelta does not require the user to select a learning rate value. The algorithm automatically adjusts the learning rate based on historical information of previous gradients.
  - Adadelta often works effectively on large and complex problems, helping to optimize machine learning models in a stable manner.
  - Adadelta has the ability to reduce the risk of overshooting, a problem that frequently appears when using conventional gradient descent algorithms.
- Disadvantage:
  - Adadelta needs to store a state for each parameter of the model, which can significantly increase the amount of memory needed when the number of parameters is large.
  - Depends on hyperparameters.

CHAPTER 2. CONTINUAL LEARNING

2.1 What is continual learning?

Continual Learning, also known as Lifelong learning, is built on the idea of learning continuously about the external world in order to enable the autonomous, incremental development of ever more complex skills and knowledge.

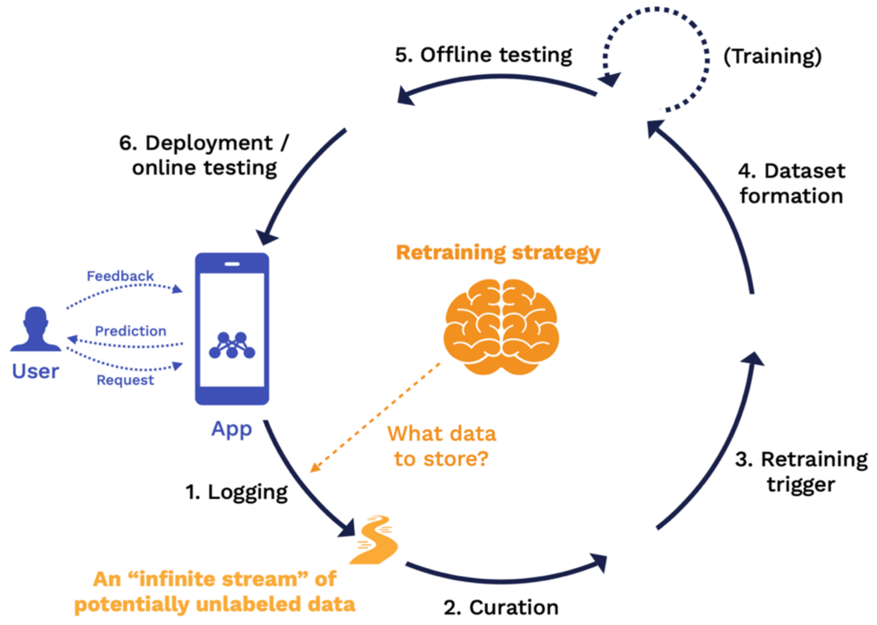
A Continual learning system can be defined as an adaptive algorithm capable of learning from a continuous stream of information, with such information becoming progressively available over time and where the number of tasks to be learned (e.g. membership classes in a classification task) are not predefined. Critically, the accommodation of new information should occur without catastrophic forgetting or interference.

Hence, in the Continual Learning scenario, a learning model is required to incrementally build and dynamically update internal representations as the distribution of tasks dynamically changes across its lifetime. Ideally, part of such internal representations will be general and invariant enough to be reusable across similar tasks, while another part should preserve and encode task-specific representations.

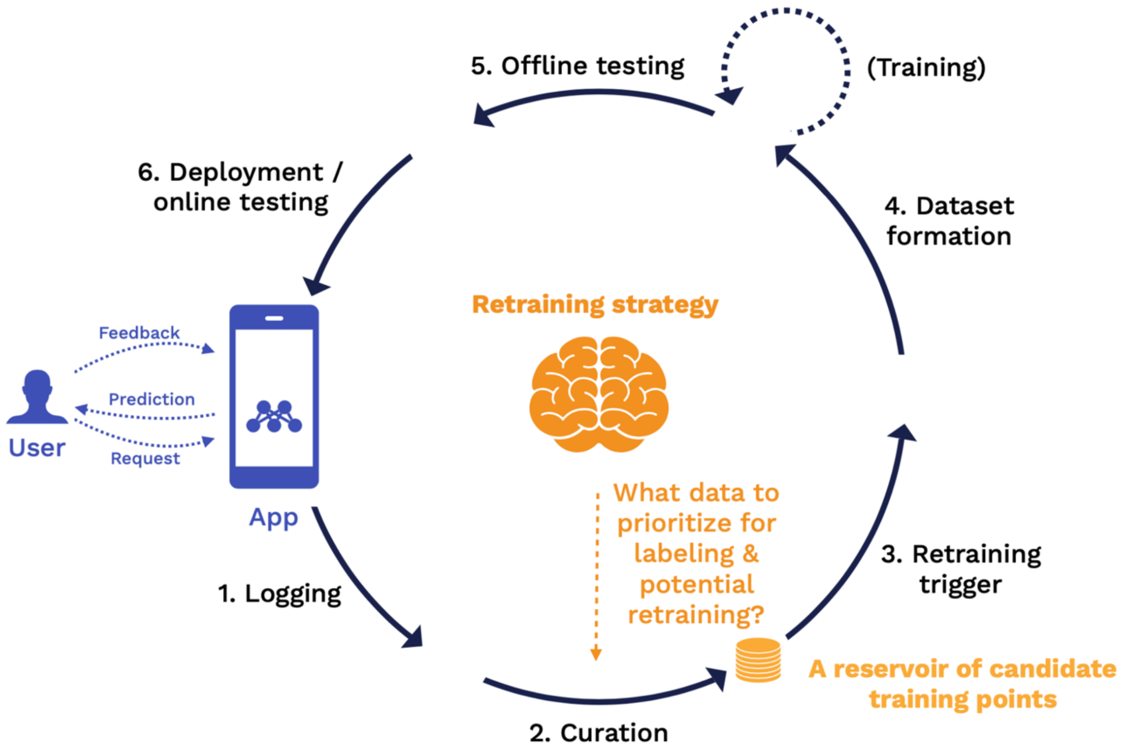
2.2 What does continual learning look like?

The continual learning loop starts with logging, which is how we get all the data into the loop. Then we have data curation, triggers for the retraining process, dataset formation to pick the data to retrain on, the training process itself, and offline testing to validate whether the retrained model is good enough to go into production. After the model is deployed, we have online testing, and that brings the next version of the model into production, where we can start the loop all over.

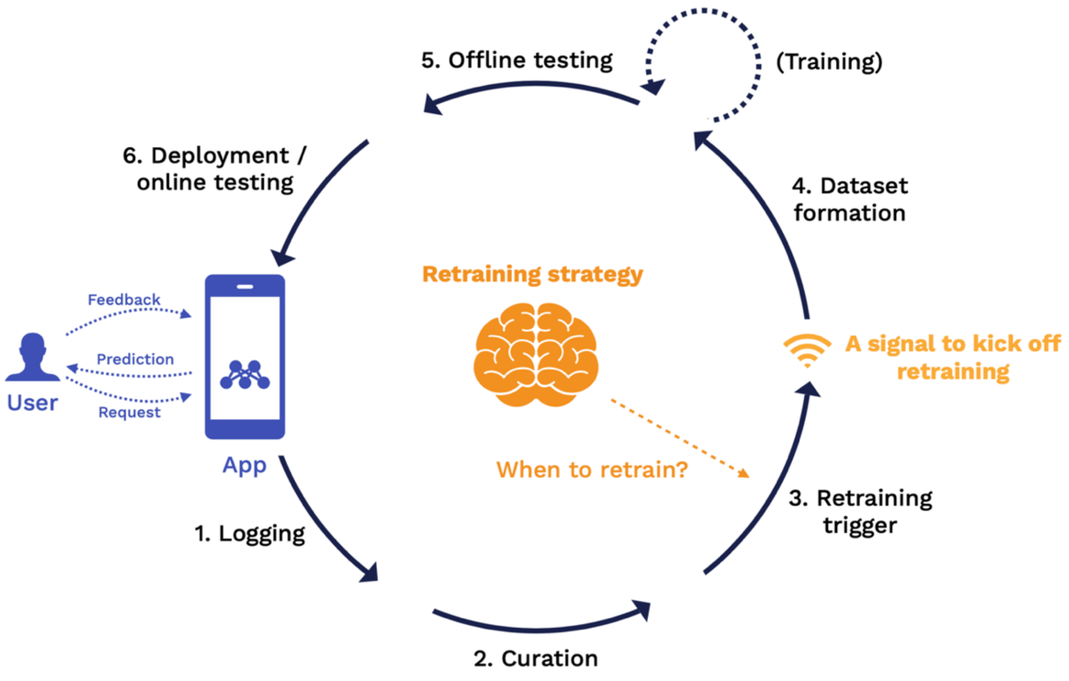
Each of these stages passes the output to the next step. Output is defined by a set of rules. These rules combine to form our retraining strategy. Let's discuss what the retraining strategy looks like for each stage:



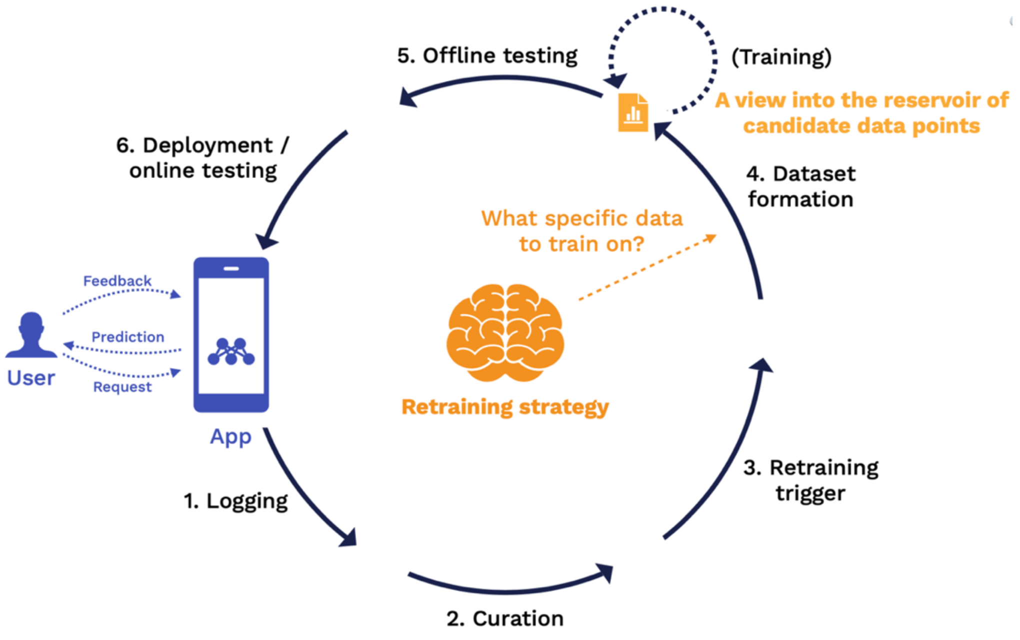
At the logging stage, the key question answered by the retraining strategy is what data should we store? At the end of this stage, we have an "infinite stream" of potentially unlabeled data coming from production and can be used for downstream analysis.



At the curation stage, the key rules we need to define are what data from that infinite stream will we prioritize for labeling and potential retraining? At the end of this stage, we have a reservoir of candidate training points that have labels and are fully ready to be fed back into a training process.

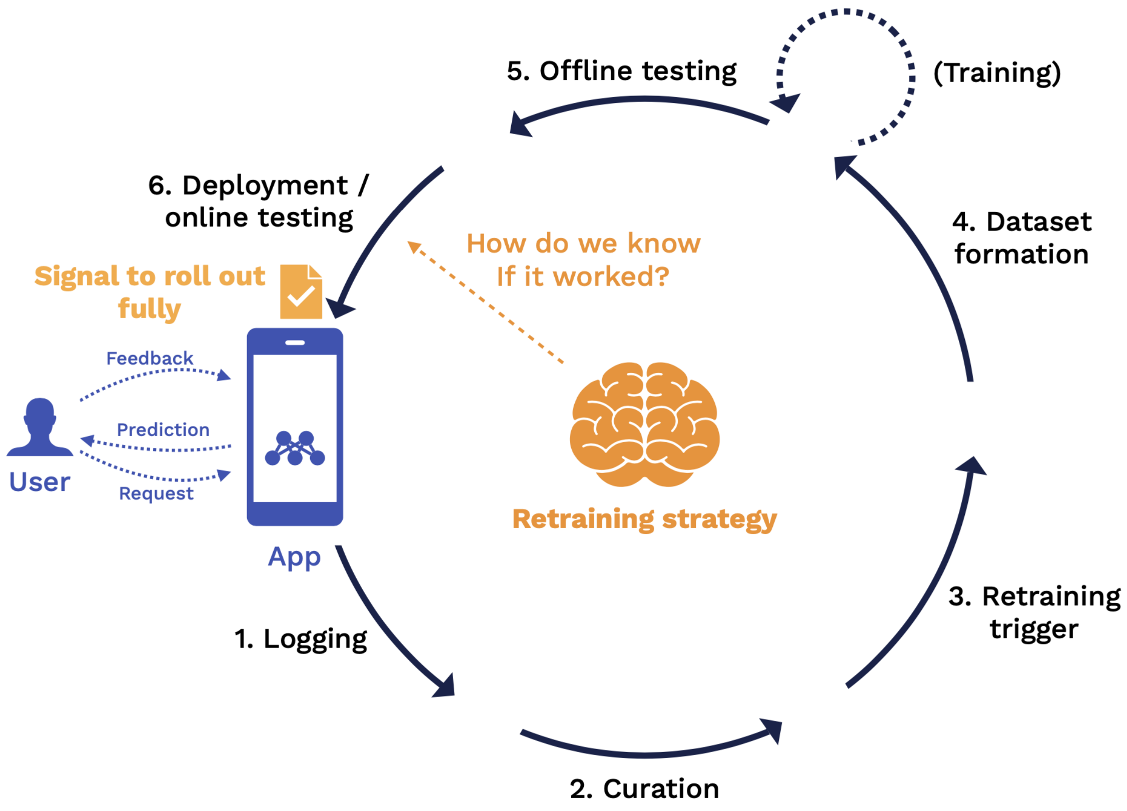


At the retraining trigger stage, the key question is when should we retrain? The output of this stage is a signal to kick off a retraining job.



At the dataset formation stage, the key rules we need to define are from this entire reservoir of data, what specific subset of that data are we using to train on for this particular training job? The output of this stage is a view into that reservoir or training data that specifies the exact data points to be used for the training job.

At the offline testing stage, the key rule we need to define is what "good enough" looks like for all stakeholders. The output of this stage is equivalent to a "pull request" report card for your model with a clear sign-off process. Once you are signed off, the new model will roll out into production.



The output of this stage is a signal to roll this model out fully to all of your users.

CHAPTER 3. TEST PRODUCTION

Test Production is a process of testing a machine learning model on a new data set, helping you evaluate the model's performance on real data. Once you have built a machine learning model, you need to test whether your model works properly on new data. Before deploying Test Production, you need to ensure that your model is fully trained and achieves good results on the training data set. You also need to ensure that your model has been fully tested and evaluated on the evaluation data set.

- Prepare Data:
  - Identify and ensure that the data input to the model in the production environment is similar to the data used during training.
  - Check the completeness and quality of the data.
- Check Model:
  - Ensure that the trained model achieves good performance on the test dataset.
  - Check if the model is suitable for the production environment.
- Model Optimization:
  - Optimize model size to ensure high performance without increasing complexity or excessive resource requirements.
- Testing Against Real Data:
  - Test models with real data from production environments to ensure performance and accuracy.
- Code Synchronization Management:
  - Use source code management tools to monitor and maintain model source code.
- Version Control:
  - Make sure you have a version management process for your model and related components.
- Security and Privacy:
  - Identify and implement security and privacy measures for models and data.
- Continuous Monitoring:
  - Set up a continuous monitoring system to monitor model performance and detect problems quickly.

- Construction of Incident System:

  - Develop troubleshooting systems to handle errors and incidents automatically.
- Documentation and Communication:

  - Develop detailed documentation on how to deploy and manage the model.

  - Communicate closely with the development team and management team to ensure common understanding.
- Team and End User Training:

  - Train operations teams on how to deploy, monitor, and manage the model.

  - Train end users on how to interact with the new machine learning solution.
- System Performance Test:

  - Test the performance of the entire system upon deployment, including the model, database, and other components.

REFERENCES

https://www.upgrad.com/blog/types-of-optimizers-in-deep-learning/

https://serpdotai.gitbook.io/the-hitchhikers-guide-to-machine-learning-algorithms/chapters/rmsprop