**VIETNAM GENERAL CONFEDERATION OF LABOR**

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**FINAL REPORT**

**INTRODUCTION TO MACHINE LEARNING**

*Instructor*: **Assoc.Prof.PhD. LE ANH CUONG**

*Student*: **VO KIEN NAM - 521H0506**

*Class*: **21H50302**

**HO CHI MINH CITY, 2023**

**VIETNAM GENERAL CONFEDERATION OF LABOR**

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**FINAL REPORT**

**INTRODUCTION TO MACHINE LEARNING**

*Instructor*: **Assoc.Prof.PhD. LE ANH CUONG**

*Student*:**VO KIEN NAM - 521H0506**

*Class*: **21H50302**

**HO CHI MINH CITY, 2023**

ACKNOWLEDGEMENT

To complete this essay, besides our own efforts, I have received a lot of help in terms of knowledge, experience, and skills from the school and teachers. First and foremost, we would like to express my special gratitude to Assoc.Prof.PhD Le Anh Cuong - the lecturer of Introduction to Machine Learning course who has taught us valuable knowledge of the subject. That knowledge is the foundation for me to continue learning and effectively apply it to this essay. Additionally, I would like to thank the teacher for allowing us to complete this essay, which has helped us to further develop my understanding of the subject. Thank you for your guidance and support in helping me to complete this essay to the best of my ability. Moreover, I would also like to express our gratitude to the school and the teachers who have compiled the Introduction to Machine Learning materials, providing me with useful resources for research and essay writing. Thank you sincerely!

**COMPLETION OF THESIS**

**AT TON DUC THANG UNIVERSITY**

I here by certify that this thesis is my/our own work and was conducted under the guidance of Assoc.Prof.PhD Le Anh Cuong. The research and results presented in this thesis are truthful and have not been published previously in any form. The data presented in tables and figures used for analysis, comments, and evaluations were collected by the author from various sources and are clearly cited in the reference section.

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

If any misconduct is detected, I fully take responsibility for the content of my thesis. Ton Duc Thang University is not liable for any copyright infringement that may occur during the thesis completion process.

*Ho Chi Minh City, December 19, 2023*

*Author*

*(signature and full name)*

ACKNOWLEDGEMENT AND EVALUATION SECTION BY INSTRUCTOR

**Instructor's Acknowledgement Section**

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

Ho Chi Minh City, 2023

(signature and full name)

**Instructor's Evaluation Section**

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

Ho Chi Minh City, 2023

(signature and full name)

SUMMARY

This essay thoroughly explores machine learning optimization algorithms, focusing on gradient descent techniques like Gradient Descent, SGD, MB-SGD, Momentum, NAG, AdaGrad, AdaDelta, RMSprop, and Adam. The first chapter details their workings, importance of the learning rate, and pros and cons.

The second chapter shifts to Continual Learning, emphasizing its significance and covering stages from manual retraining to advanced continual learning. Challenges, including data access and algorithmic issues, are discussed. The chapter also delves into testing models in production, detailing strategies like Shadow Deployment, A/B Testing, Canary Release, Interleaving Experiments, and Bandits for robust deployment.

In summary, the essay provides a comprehensive exploration of optimization algorithms and delves into Continual Learning challenges and testing strategies for effective machine learning model deployment.

**TABLE OF CONTENTS**

[ACKNOWLEDGEMENT i](#_Toc154351392)

[ACKNOWLEDGEMENT AND EVALUATION SECTION BY INSTRUCTOR iii](#_Toc154351393)

[SUMMARY v](#_Toc154351394)

[**TABLE OF CONTENTS** 1](#_Toc154351395)

[**CHAPTER 1: OPTIMIZATION ALGORITHMS IN MACHINE LEARNING** 3](#_Toc154351396)

[What are Optimizers in Machine Learning (Deep Learning) ? 3](#_Toc154351397)

[1.1. Gradient Descent 3](#_Toc154351398)

[1.2. Stochastic Gradient Descent (SGD) 4](#_Toc154351399)

[1.3. Stochastic Gradient Descent with momentum 4](#_Toc154351400)

[1.4. Adagrad 4](#_Toc154351401)

[1.5. RMS Prop (Root Mean Square) 4](#_Toc154351402)

[1.6. Adam 4](#_Toc154351403)

[**CHAPTER 2: CONTINUAL LEARNING AND TEST IN PRODUCTION** 5](#_Toc154351404)

[**2.1 Continual Learning** 5](#_Toc154351405)

[***2.1.1 Why Continual Learning?*** 6](#_Toc154351406)

[***2.1.2 Concept: Stateless retraining VS Stateful training*** 6](#_Toc154351407)

[2.1.2.1 Stateless retraining 7](#_Toc154351408)

[2.1.2.2 Stateful training (aka fine-tuning, incremental learning) 7](#_Toc154351409)

[***2.1.3 Concept: feature reuse through log and wait*** 8](#_Toc154351410)

[***2.1.4 Continual Learning Challenges*** 8](#_Toc154351411)

[2.1.4.1 Fresh data access challenge 9](#_Toc154351412)

[2.1.4.2 Evaluation Challenge 10](#_Toc154351413)

[2.1.4.3 Data scaling challenge 11](#_Toc154351414)

[2.1.4.4 Algorithm challenge 11](#_Toc154351415)

[***2.1.5 The Four Stages of Continual Learning*** 12](#_Toc154351416)

[2.1.5.1 Stage 1: Manual, stateless retraining 12](#_Toc154351417)

[2.1.5.2 Stage 2: Fixed schedule automated stateless retraining 12](#_Toc154351418)

[2.1.5.3 Stage 3: Fixed schedule automated stateful training 13](#_Toc154351419)

[2.1.5.4 Stage 4: Continual learning 14](#_Toc154351420)

[**REFERENCES** 16](#_Toc154351421)

**CHAPTER 1: OPTIMIZATION ALGORITHMS IN MACHINE LEARNING**

# What are Optimizers in Machine Learning (Deep Learning) ?

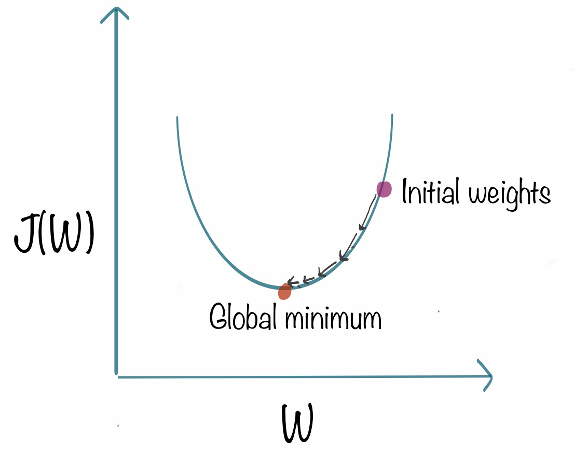
+ Optimizers are algorithms that adjust the model’s parameters during training to minimize a loss function and maximize accuracy. They enable neural networks to learn from data by iteratively updating weights and biases

+ Commonly used and well-known optimizers are **Gradient Descent**, **Stochastic Gradient Descent (SGD), RMSprop and Adam.** Each of them has specific update rules, learning rates and momentum to find optimal model parameters for improving performance.

# Gradient Descent

- Imagine there is a valley, and the height of this valley can be thought as magnitude for loss function J(w,b). The deeper we go of the valley, the lower the value of loss function. The same happens to Gradient descent, we start at a position of the valley with arbitrary initial coefficients (w = 0 and b = 0,usually) and then we gradually go down by using an optimizer to update our coefficients. We will go at one direction as long as the next step the deeper than the one we are in, we will only stop when there is no deeper way around us to go, that is where we got to the minimum of loss function - best performance.

- Here is how we could possibly imagine about gradient descent, normally this bowl would be in 3D as there are weight, bias and value of loss function



- Mathematical Expression

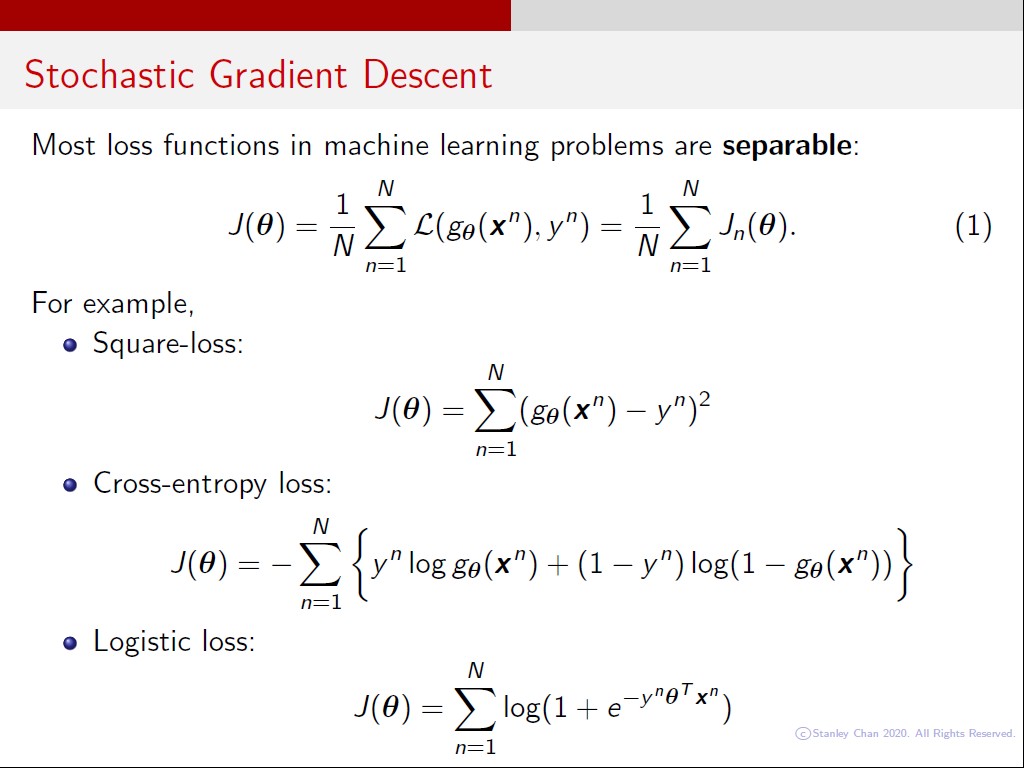
A math equations on a transparent background

Description automatically generated

- If data is huge it would be computationally expensive to perform gradient descent and for those non-convex functions

# Stochastic Gradient Descent (SGD)

- This is a better version of Gradient Descent, SGD could tackle the problem of massive data. “Stochastic” means random or arbitrary, and that is also the way the algorithm works, it randomly select the batches of data - only taking a few samples from dataset.



- Similar to GD, first we select initial w,b and learning rate alpha, then randomly shuffle the data at each iteration to reach an approximate minimum.

- Downside: Since we are not using the whole dataset but the batches of it for each iteration, the path taken by the algorithm is full of noise as compared to the gradient descent algorithm; It is still time consuming to compute but not as much as Gradient Descent.

# Stochastic Gradient Descent with momentum

**-** The momentum part helps faster convergence of the loss function by multplying to the derivatives part. Stochastic gradient descent oscillates between either direction of the gradient and updates the weights accordingly. However, adding a fraction of the previous update to the current update will make the process a bit faster.

- It use a algorithm called exponentially weighted moving average, where the momentum v of t has larger weight than v of (t-n) (0 <= n < t)

A graph and a diagram

Description automatically generated with medium confidence

- Using momentum for SGD can even make it convergence faster, But if combining with choosing the right learning rate, we would faster the process but it would be easier to land on local minima, it is like a trough on the way we are going down hill, and this make the predicting accuracy bad.

# Adagrad

**-** The momentum part helps faster convergence of the loss function by multplying to the derivatives part. Stochastic gradient descent oscillates between either direction of the gradient and updates the weights accordingly. However, adding a fraction of the previous update to the current update will make the process a bit faster.

- It use a algorithm called exponentially weighted moving average, where the momentum v of t has larger weight than v of (t-n) (0 <= n < t)

A graph and a diagram

Description automatically generated with medium confidence

- Using momentum for SGD can even make it convergence faster, But if combining with choosing the right learning rate, we would faster the process but it would be easier to land on local minima, it is like a trough on the way we are going down hill, and this make the predicting accuracy bad.

- Adagrad uses different learning rate for each iteration, the change in learning rate depends upon the difference in the parameters during training. The more the parameters get changed, the more minor the learning rate changes.

IMG_256

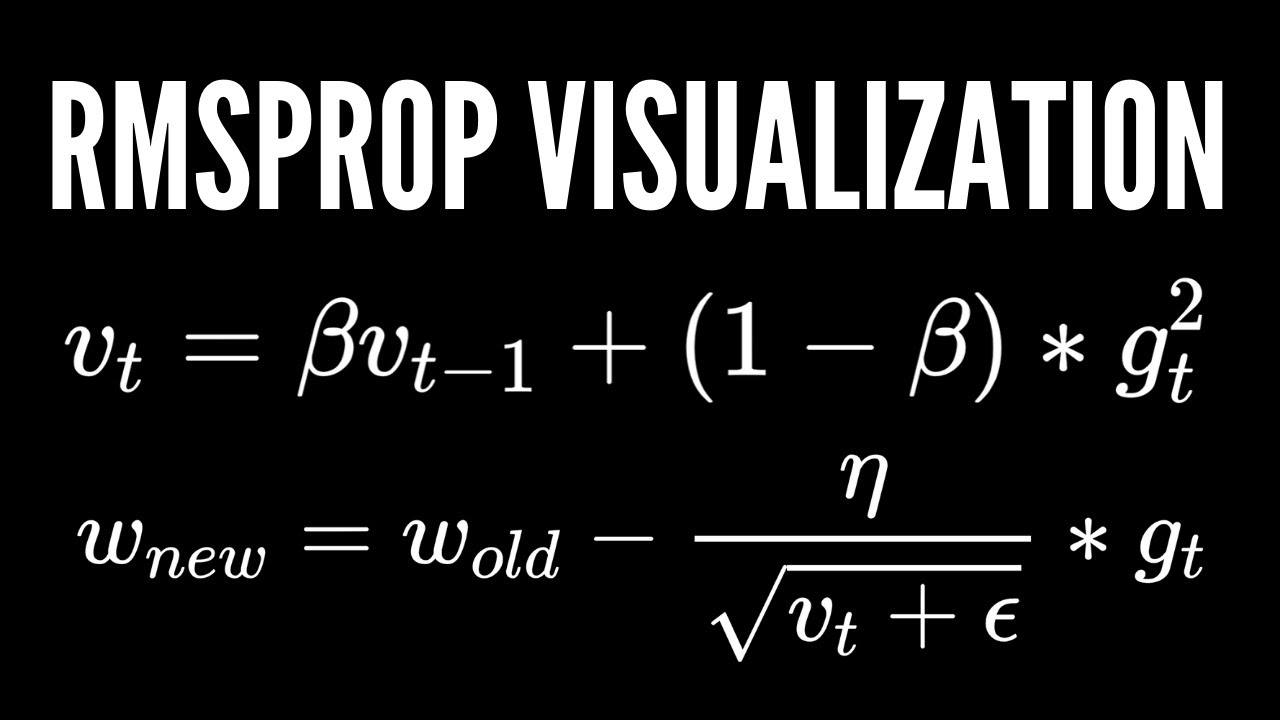
IMG_256

- The benefit of using Adagrad is that it abolishes the need to modify the learning rate manually. It is more reliable than gradient descent algorithms and their variants, and it reaches convergence at a higher speed.

- Downside: There might be a point where the learning rate got so small and it negatively affects the learning process

# RMS Prop (Root Mean Square)

- RMS prop involves dividing the learning rate by an exponentially decaying average of squared gradients. Geoffrey Hinton recommends setting y to 0.9 and a commonly suggested default value for the learning rate is 0.001



# Adam

- Adam optimizer, short for Adaptive Moment Estimation optimizer, is an optimization algorithm commonly used in deep learning. It is an extension of the stochastic gradient descent (SGD) algorithm and is designed to update the weights of a neural network during training.

-The name “Adam” is derived from “adaptive moment estimation,” highlighting its ability to adaptively adjust the learning rate for each network weight individually. Unlike SGD, which maintains a single learning rate throughout training

- The creators of Adam optimizer incorporated the beneficial features of other optimization algorithms such as AdaGrad and RMSProp

IMG_256

+ The above formula represents the working of adam optimizer. Here B1 and B2 represent the decay rate of the average of the gradients.

**CHAPTER 2: CONTINUAL LEARNING AND TEST IN PRODUCTION**

This section addresses two significant and interconnected subjects: Continual Learning and Testing models in Production. The objective of exploring these topics concurrently is to acquire the skills needed to automate, ensure safety, and enhance the efficiency of updating models in a production setting.

**2.1 Continual Learning**

Continual Learning involves the concept of regularly updating your model as new data becomes accessible, allowing it to stay aligned with current data distributions.

However, once the model undergoes an update, it is not advisable to deploy it to production without thorough testing. It is crucial to conduct testing to verify the safety and superiority of the updated model compared to the existing one in production. This is the focus of the subsequent section, titled "Testing models in Production".

Continual learning is frequently misunderstood:

* Continual learning does NOT exclusively pertain to a specific category of ML algorithms designed for incremental model updates with each new datapoint, such as sequential Bayesian updating and KNN classifiers. These algorithms, often termed "online learning algorithms," represent a limited subset.
* The concept of Continual learning is applicable to any supervised ML algorithm, not confined to a particular class.
* Continual learning does NOT involve initiating a retraining process every time a new data sample becomes available. In fact, this practice is risky, as it can render neural networks susceptible to catastrophic forgetting.
* Many companies that implement continual learning update their models in micro-batches, typically at intervals like every 512 or 1024 examples. The optimal number of examples varies depending on the specific task.

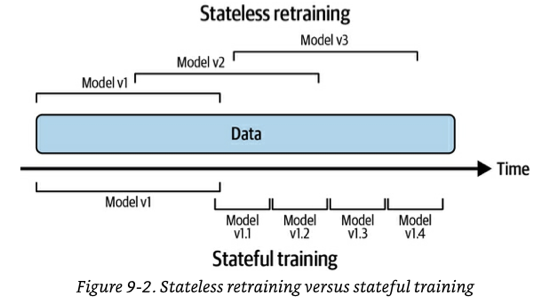
While Continual learning may initially appear to be a task for data scientists, it frequently demands substantial **infrastructure work** to be effectively implemented.

***2.1.1 Why Continual Learning?***

The fundamental purpose is to assist your model in **staying aligned with shifts in data distribution**. Several critical use cases underscore the need for swift adaptation to changing distributions, including:

* **Use Cases with Unpredictable and Rapid Changes**: Industries like ride-sharing face scenarios where unexpected and rapid changes occur. For instance, a concert in an unforeseen location on a random Monday could challenge the effectiveness of the "Monday pricing ML model".
* **Use Cases Lacking Training Data for Specific Events**: Certain situations, such as Black Friday or novel sale events in e-commerce, pose challenges in acquiring sufficient historical data for training. Adapting the model throughout the event becomes essential to predict user behavior.
* **Use Cases Prone to the Cold Start Problem**: The cold start problem arises when a model needs to make predictions for a new or logged-out user without any historical or outdated data. Adapting the model promptly upon receiving data from such users is crucial for providing relevant recommendations.

***2.1.2 Concept: Stateless retraining VS Stateful training***



2.1.2.1 Stateless retraining

Initiate a complete retraining of your model on every occasion, utilizing freshly initialized weights and updated data.

* There could be instances of data overlap with what was previously used to train the prior version of the model.
* Many companies commence continual learning through a stateless retraining approach.

2.1.2.2 Stateful training (aka fine-tuning, incremental learning)

Initialize your model with weights from the previous training round and proceed with training on new, unseen data.

* This approach enables the model to update with considerably less data.
* Convergence is faster, and less compute power is required, with some companies reporting a 45% reduction.
* Theoretically, it may eliminate the need to store data once it's been used for training, addressing privacy concerns.
* However, in practice, many companies tend to retain data even when unnecessary.
* Periodic **stateless retraining** with a large dataset is necessary to recalibrate the model.
* Once the infrastructure is properly configured, switching from stateless retraining to stateful training is a simple process.
* **Model iteration versus data iteration**: Stateful training is mainly employed to integrate new data **into an existing fixed model architecture** (i.e., data iteration). If you want to modify your model's features or architecture, a preliminary stateless retraining is required.
* Some research exists on techniques like Net2Net knowledge transfer and model surgery for transferring weights between different model architectures. However, these methods have seen little to no adoption in industry so far.

***2.1.3 Concept: feature reuse through log and wait***

Features are computed for inference, and certain companies opt to store these calculated features for each data sample. This practice, known as **log and wait**, allows for the reuse of features in continual learning training, leading to computational savings. Additionally, it serves the purpose of aiding feature monitoring. While not widely adopted as of January 2023, log and wait is gradually gaining popularity.

***2.1.4 Continual Learning Challenges***

Despite its successful application in the industry, continual learning poses three significant challenges that companies must address.

2.1.4.1 Fresh data access challenge

To update your model every hour, obtaining **high-quality labeled** training data on an hourly basis is essential. The more frequent the update schedule, the more crucial this challenge becomes.

**Problem: Speed of data deposit into data warehouses**

Numerous companies retrieve their training data from data warehouses such as Snowflake or BigQuery. However, data from various sources is fed into the warehouse through distinct mechanisms and at varying rates.

For instance, data in the warehouse may come directly from real-time transport (events), while other portions may be sourced from daily or weekly ETLs that transfer data from other origins.

A common solution to address this challenge involves extracting data directly from real-time transport for training before it gets deposited in the warehouse. This approach is particularly effective when the real-time transport is linked to a feature store. However, there are hurdles to implementing this strategy:

* Some data, especially from external vendor systems beyond your control, may not flow through events. To ensure freshness in such cases, finding a method to capture changes on those systems using events, web-hooks, or polling APIs becomes necessary.
* In certain companies, batched ETLs perform extensive processing and data joining within the data warehouse to enhance utility. Shifting to a full real-time transport strategy requires devising a way to replicate the same processing on a continuous stream of data.

**Problem: Speed of labelling**

The rate at which new data can be labeled often serves as a bottleneck. Tasks with inherent labels and short feedback loops are the most suitable for continual learning, as a **shorter feedback loop** allows for faster labeling.

If obtaining natural labels within the required timeframe is challenging, alternative approaches such as weak supervision or semi-supervision techniques can be considered, albeit with the trade-off of potentially noisier labels. As a last resort, recurrent and rapid crowdsourcing for label annotation may be an option.

Another factor influencing labeling speed is the **label computation strategy**:

* Batch label computation involves periodic processing of data deposited into the data warehouse. The labeling speed is dependent on both the speed of data deposition and the frequency of label computation jobs.
* Similar to the aforementioned solution, a common method to expedite labeling is to compute labels directly from real-time transport (events). However, this streaming computation comes with its own set of challenges.

2.1.4.2 Evaluation Challenge

Incorporating continual learning as a practice introduces the risk of significant model failures. The higher the frequency of model updates, the greater the chances of encountering failures.

Furthermore, continual learning creates opportunities for coordinated adversarial attacks aimed at poisoning the models.

Consequently, rigorous testing of models before their deployment to a broader audience becomes crucial.

* Testing is a time-consuming process, serving as a potential constraint on achieving the fastest model update frequency.
* For instance, a new model designed for fraud detection may require approximately two weeks to accumulate sufficient traffic for a confident evaluation.

2.1.4.3 Data scaling challenge

Feature calculation typically involves scaling, which, in turn, necessitates access to global data statistics such as minimum, maximum, average, and variance.

In the case of stateful training, global statistics must account for both the previous data used to train the model and the new data being employed for updates. Managing global statistics in this context can be challenging.

A common approach to address this challenge is to calculate or approximate these statistics incrementally as new data is observed, as opposed to loading the entire dataset at training time for computation.

* An illustrative example of this technique is “Optimal Quantile Approximation in Streams”.
* Sklearn’s StandardScaler offers a *partial\_fit* method that enables a feature scaler to be used with running statistics. However, the built-in methods are slow and have limitations in supporting a broad range of running statistics.

2.1.4.4 Algorithm challenge

This issue arises when certain types of algorithms are employed and there is a need for rapid updates, such as every hour.

The algorithms in question are those that, by design, depend on having access to the complete dataset for training. Examples include matrix-based, dimensionality reduction-based, and tree-based models. Unlike neural networks or other weight-based models that can be incrementally trained with new data, these types of models require the entire dataset.

* For instance, PCA dimensionality reduction cannot be performed incrementally; the full dataset is necessary.

This challenge becomes particularly pronounced when there is a need for very fast updates, and waiting for the algorithm to process the full dataset is not feasible.

While there are some variations of these models designed for incremental training, their adoption is not widespread. Hoeffding Trees and sub-variants are examples of such algorithms.

***2.1.5 The Four Stages of Continual Learning***

Typically, companies progress through four stages when transitioning to continual learning.

2.1.5.1 Stage 1: Manual, stateless retraining

Retraining of models occurs only under two conditions: (1) when the model's performance has significantly deteriorated, reaching a point where it becomes counterproductive, and (2) when the team has the available time and resources to carry out the update.

2.1.5.2 Stage 2: Fixed schedule automated stateless retraining

This stage typically occurs when the primary models in a domain have already been established, shifting the focus from creating new models to maintaining and enhancing existing ones. Remaining in stage 1 becomes too burdensome to overlook.

During this stage, the retraining frequency is often determined by intuition or a “gut feeling”.

The transition from stage 1 to stage 2 is commonly marked by the development of a script that someone creates to execute stateless retraining at regular intervals. Writing this script's complexity varies based on the number of dependencies that need coordination for model retraining.

The fundamental steps of this script include:

1. Retrieve data.
2. Perform downsampling or upsampling if required.
3. Extract features.
4. Process and/or annotate labels to generate training data.
5. Initiate the training process.
6. Assess the new model.
7. Deploy the updated model.

To implement this script, two additional components of infrastructure are necessary:

1. A scheduler.
2. A model store to automatically version and store all the artifacts required to reproduce the model. Established model stores include AWS SageMaker and Databrick's MLFlow.

2.1.5.3 Stage 3: Fixed schedule automated stateful training

To accomplish this, you must modify your script and establish a method for tracking data and model lineage. A straightforward model lineage versioning example includes:

* V1 vs V2, representing two distinct model architectures addressing the same problem.
* V1.2 vs V2.3 signifies that model architecture V1 is in its second iteration of a full stateless retraining, while V2 is in its third.
* V1.2.12 vs V2.3.43 indicates that there have been 12 stateful trainings performed on V1.2 and 43 on V2.3.
* It's likely that you'll need to combine this approach with other versioning techniques, such as data versioning, to maintain a comprehensive understanding of how the models are evolving.
* The author notes that there is no known model store with this type of model lineage capability, prompting companies to develop their own in-house solutions.
* At any given point, multiple models will be operational in production concurrently, facilitated by the arrangements outlined in Testing Models in Production.

2.1.5.4 Stage 4: Continual learning

In this stage, the **fixed schedule** component from previous stages is replaced by a **retraining trigger mechanism**. The triggers can be:

* **Time-based**.
* **Performance-based**, where retraining is triggered if performance falls below a certain threshold (e.g., below x% accuracy). However, measuring accuracy directly in production may not always be feasible, requiring the use of a weaker proxy.
* **Volume-based**, where retraining is prompted by a 5% increase in the total amount of labeled data.
* **Drift-based**, where retraining is initiated when a “major” shift in data distribution is detected. The challenge with drift-based triggers, as mentioned in the preceding chapter, is determining when such data distribution shifts become problematic for the model's performance degradation.

**REFERENCES**

1. *Optimization algorithms in neural networks*. (2022, June 7). The AI Dream. <https://www.theaidream.com/post/optimization-algorithms-in-neural-networks>
2. Kumar, S. (2021, December 14). Overview of various Optimizers in Neural Networks - Towards Data Science. *Medium*. <https://towardsdatascience.com/overview-of-various-optimizers-in-neural-networks-17c1be2df6d5>
3. Doshi, S. (2021, December 7). Various optimization algorithms for training neural network. *Medium*. <https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>
4. Bushaev, V. (2018, June 21). Stochastic Gradient Descent with momentum - Towards Data Science. *Medium*. <https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d>
5. Ruder, S. (2020, March 20). *An overview of gradient descent optimization algorithms*. ruder.io. <https://ruder.io/optimizing-gradient-descent/>
6. Huyen, C. (2022, Chapter 9). *Designing machine learning systems: An Iterative Process for Production-Ready Applications*. O’Reilly Media.