Vietnam General Confederation of Labor
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

FINAL REPORT

MACHINE LEARNING

Instructor: **Mr. LÊ ANH CƯỜNG**

Student: **PHAN THÀNH ĐẠT - 521H0218**

Class :**21H50302**

Year **: 2023-2024**

HO CHI MINH CITY, 2023

Vietnam General Confederation of Labor
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

FINAL REPORT

MACHINE LEARNING

Instructor: **Mr. LÊ ANH CƯỜNG**

Student: **PHAN THÀNH ĐẠT - 521H0218**

Class :**21H50302**

Year **: 2023-2024**

HO CHI MINH CITY, 2023

ACKNOWLEDGEMENT

I would like to express my deep gratitude to Mr. Le Anh Cuong for his valuable support in the past project. The teacher's knowledge and guidance have played an important role in our work process. What he shared and guided us helped us overcome challenges, expand our knowledge and approach the project more confidently. His dedication and help not only helped us understand the project better, but also encouraged and inspired us.

Ho Chi Minh city, 22th December, 2023

Author

(Sign and write full name)

Phan Thành Đạt

CONFIRMATION AND ASSESSMENT SECTION

Instructor confirmation section

_____

_____

_____

_____

_____

_____

_____

Ho Chi Minh 22 December, 2023

(Sign and write full name)

Evaluation section for grading instructor

_____

_____

_____

_____

_____

_____

_____

Ho Chi Minh, 22 December 2023

(Sign and write full name)

THE PROJECT IS COMPLETED
AT TON DUC THANG UNIVERSITY

Our team would like to assure that this is our own research project and is under the scientific guidance of Lê Anh Cường Teacher. 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 report 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, our team will take full responsibility for the content of our IT Project Report 2. Ton Duc Thang University is not involved in copyright violations caused by us during the implementation process (if any).

Ho Chi Minh city, 22th December, 2023

Author

(Sign and write full name)

Phan Thành Đạt

SUMMARY

Improving model training requires understanding machine learning optimizers. Several well-known algorithms such as Adam, Stochastic Gradient Descent and Gradient Descent reduce errors by modifying model parameters

INDEX

CHAPTER 1 – Optimizer

1.1 Introduction

Optimization algorithms are important in the construction of neural network models because they allow them to learn the features of incoming data and discover optimum weight and bias settings. This learning process, on the other hand, requires systematic algorithms that iteratively adjust weights and bias. At this stage, optimizer algorithms intervene, changing these variables to minimize the loss function and quantify model performance. Gradient descent is a method for determining the best parameters by computing gradients that guide updates.

Iterative optimization improves the model's ability to capture data patterns by adjusting parameters based on forecast discrepancies. Several optimizer algorithms, including as SGD, Adam, RMSprop, and Adagrad, use unique tactics to increase convergence speed and stability (adaptive learning rates, momentum).

1.2 Optimizer's optimization algorithms

1.2.1 Gradient Descent (GD)

A common problem in Machine Learning and Optimization Mathematics is to find the lowest (or biggest) value of a function. Determining the global minimum of loss functions is usually difficult or impossible in Machine Learning. The emphasis is instead on discovering local minimum points and analyzing them as potential solutions.

Local minimums satisfy the derivative equation = 0. Finding all of these points and choosing the one with the lowest function value is ideal, but due to derivative complexity, a large number of data points, or their sheer size, solving = 0 is usually impossible.

The most often used method is GD (Gradient Descent) and its derivatives. It involves starting near a solution and iterating until the derivative approaches zero.

1.2.1.1 Gradient for 1-variable function

Assume we have an objective function f(x) that depends on only one variable x to learn how Gradient Descent works for a function of one variable. The aim is to determine the value at which the function f(x) has the smallest value.

There are 4 steps:

Step 1: Set the starting value of x: Begin with a random or pre-determined value of x.

Step 2: Determine the derivative of the target function at x: Determine the derivative of the function f(x) at x. This derivative indicates the function's direction and slope at point x.

Step 3:Move x in the opposite direction of the derivative: Move against the derivative of the target function to update the x value. This may be accomplished using the following revised formula::

$$x_{new} = x_{old} - \eta \cdot f'(x_{old})$$

In there:

+ x_{new} is the x value after update
+ x_{old} is the x value before the update
+ η is the learning rate, the learning rate determines the degree of movement at each update step.
+ $f'(x_{old})$ is the derivative of f(x) at x_{old}

Step 4: ***Repeat the process:*** Continue updating x by computing the derivative at the current position and pushing x in the opposite direction as the derivative until the halting condition or a near value is obtained. true for the function f(x)'s lowest value

1.2.1.2 Gradient descent for multi-variable function:

For a multi-variable function f(x1,x2,x3,...,xn), the gradient descent strategy comprises iteratively updating the variables x1,x2,x3,...,xn in the direction of the function's negative gradient with respect to these variables.

The gradient descent approach for a multivariable function:

1. Initialize variables: Choose initial values for $x_1,x_2,x_3,\ldots,x_n$

2. Set hyperparameters: Define the learning rate *(α)*, which determines the step size in each iteration.

3. Iterate until convergence or a stopping criterion:

Compute the gradient: Calculate the partial derivatives of the function with respect to each variable.

Update the variables: Update each variable using the gradient information:

$$x_i = x_i - \alpha \frac{\partial f}{\partial x_i}$$

Continue in this manner until convergence is reached, which is the point at which there is very little or no change in the function value or variables over a certain threshold.

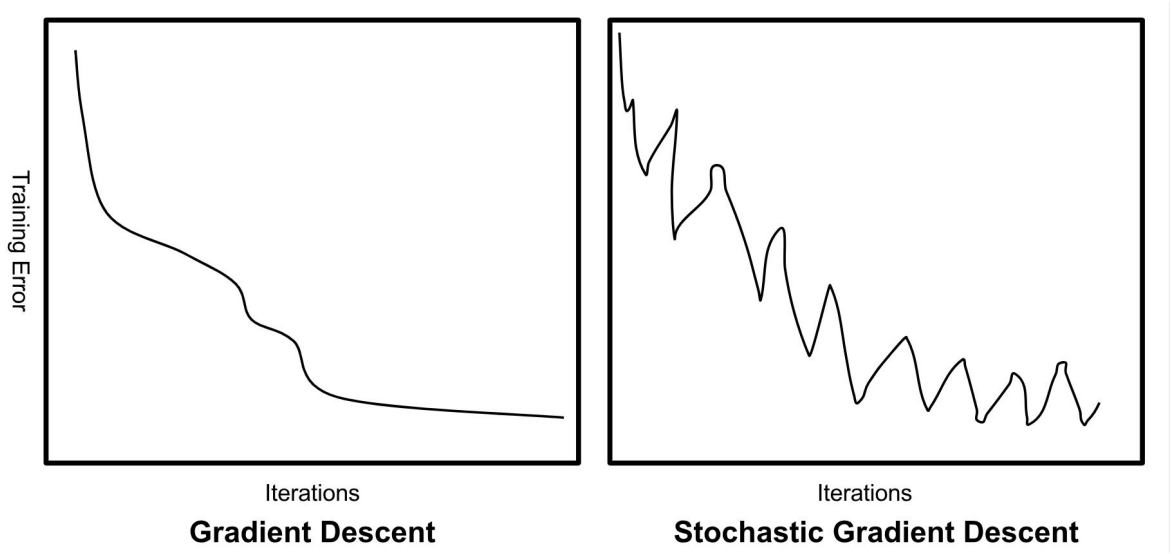
The gradient multiplied by the learning rate is subtracted from the current value of each variable x_i to find the formula for updating each variable xi in the direction of the negative gradient.

The method seeks to discover the local minimum of the function by going in the opposite direction (subtracting the gradient), which gives the path of the steepest climb indicated by the gradient.

It's vital to remember that selecting the right learning rate is essential; a high learning rate might lead to overshooting and divergence, while a modest learning rate could cause the convergence to happen extremely slowly.

1.2.2 Stochastic Gradient Descent (SGD)

Stochastic gradient descent (SGD) is an improvement on stochastic gradient descent. Unlike the standard method of updating the weights once per epoch, SGD updates the weights N times in an epoch, corresponding to N data points in the set. Although this method seems to be less efficient in each epoch, SGD converges quickly and the convergence target is usually achieved after a few epochs. Its formula is similar to GD, but operates on individual data points during updates.



SGD follows a less stable path than the smoother GD, which is understandable since one data point cannot fully represent the entire data set. Despite this property, why do we prefer SGD to GD?

The point is that GD is struggling with huge databases (e.g., millions of data) due to the expensive computational cost of derivatives for the entire dataset in each iteration. Furthermore, it is not suitable for online learning.

In online learning, continuous data updates (e.g. adding new user data) make it impractical to recompute the complete derivative of GD. SGD's solution lies in

updating weights only based on new data points, making it ideal for online learning. It adapts well to dynamic data without high computational requirements, making it a reasonable choice for large datasets and dynamic online learning settings.

1.2.2.1 Algorithm

The stochastic gradient descent (SGD) algorithm is an optimization technique that can be used to determine the minimum of a function by modifying the parameters toward the negative gradient of the function.

The basic steps of the SGD algorithm are as follows:

1. Initialization: Select initial values for the model parameters (weights).

2. Choose learning rate: Determine the value of the learning rate (α), which dictates the step size in each update.

3. Iterative parameter updates:

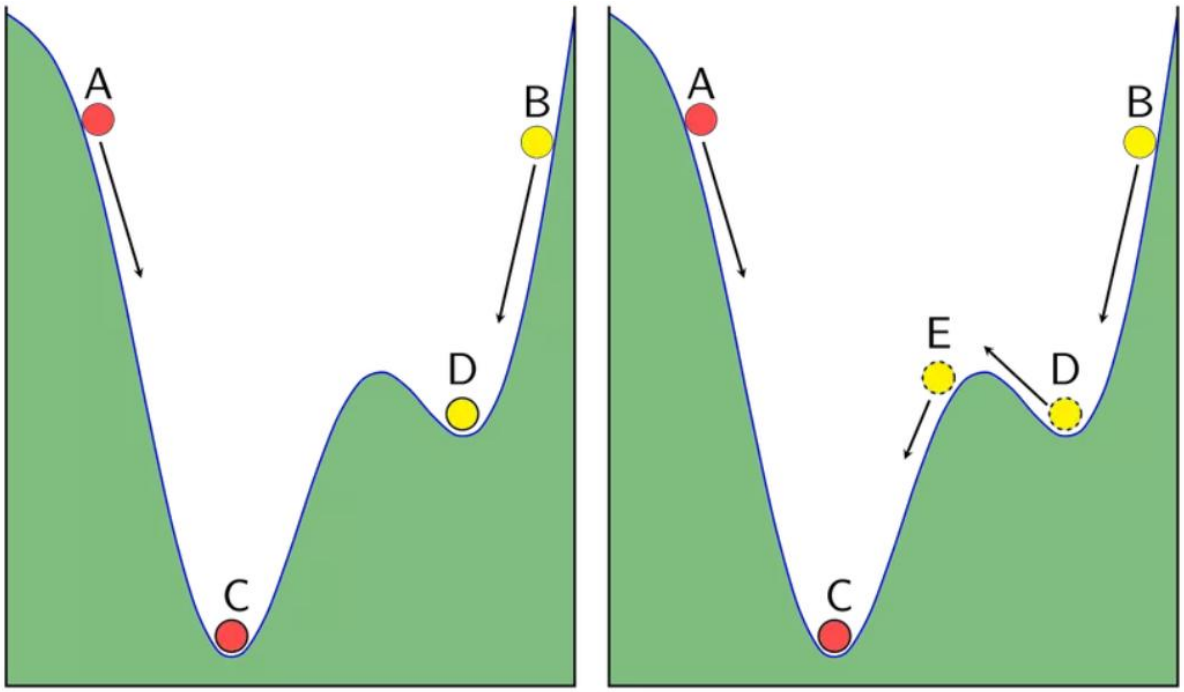
- Randomly select a data point from the training dataset.
- Compute the partial derivatives (gradient) of the loss function at this data point.
- Update the parameters based on this gradient using the formula:

$$weight_{new} = weight_{old} - \alpha \; x \; gradient.$$

- Repeat this process until a stopping criterion is met (e.g., a certain number of iterations or desired accuracy).

1.2.3 Momentum

An additional momentum term was added during the update to the optimization algorithm variant, called momentum-based gradient descent. This term is a moving average of previous gradients, set by a hyperparameter called Beta. Solve the problem of slow convergence and oscillation and speed up optimization by collecting updates in the fastest descending direction. This approach improves the training of machine learning models for better performance.



b) GD c) GD with momentum

To understand the momentum gradient, consider this physics illustration: Imagine that points A and B are marbles that you release. The goal is to prevent the marble at B from stopping at D (local minimum) when it reaches point C at A, and then continuing to C (global minimum). This can be done using momentum, which is similar to the initial velocity of the marble in the momentum algorithm. It enhances the optimization by integrating the previous momentum as well as the current gradient to avoid local minima and accelerate convergence to the desired point.

1.2.3.1 Algorithm

Input:

- Loss jaw: J
- Initial set of parameters θ_0
- Learning rate: α
- Momentum coefficient β
- Number of iterations (epochs) T

Step 1: Initialize

Initialize v_0 as a zero vector of the same size as θ_0

Step 2: Training

Let t = 0 to T-1:

Calculate the gradient of the loss $\nabla J(\theta_t)$ function

Momentum update:

$$v_{t+1} = \beta * v_t + (1 - \beta) * \nabla J(\theta_t)$$

Update parameters:

$$\theta_{t+1} = \theta_t - \alpha * v_{t+1}$$

Output:

The parameter set is trained with θ_T the optimized value of the loss function.

1.2.4 Adagrad

Unlike the uniform scaling used in classic methods, Adagrad (a popular machine learning algorithm) scales based on the gradient history to adjust the learning rate based on the model parameters.

1.2.4.1 Key Characteristics

- *Adaptive Learning Rates:* Adagrad adjusts learning rates per parameter using specific gradient history, optimizing performance.

- *Gradient Squares Accumulation:* Adagrad stores squared gradients per parameter, essential for rate calibration throughout training.

- *Parameter-Specific Scaling:* It scales rates based on parameter update frequencies, allocating larger adjustments to less frequently updated parameters.

- *Update Mechanism:* Adagrad divides the initial rate by the root of squared gradient sums per parameter, adjusting rates based on historical gradients.

Adagrad's adaptability suits models with varied parameter sensitivities but tends to reduce rates aggressively over time. This limitation has led to the development of alternatives like RMSprop and Adam.

1.2.4.2 Fomula

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

$\theta_{t+1,i}$: is the update value of parameter *i* at time *t + 1*

$\theta_{t,i}$: is the current value of parameter *i* at time *t*

η : denotes the learning rate

$G_{t,ii}$:is a diagonal matrix where each element on the diagonal *(i,i)* is the square of the parameter vector derivative at time *t*.

$g_{t,i}$: signifies the gradient of the loss function concerning parameter *i* at time *t*

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

1.2.5 RMSprop

An adjustable learning rate method called root mean square propagation (RMSprop) is used to train neural networks. It attempts to solve the problem of diminishing learning rate in Adagrad.

1.2.5.1 How does it work?

Although using a different approach, RMSprop computes a weighted average of squared gradients in a similar way to Adagrad. It uses this weighted average to update the neural network weights for each parameter, allowing each parameter to use a different learning rate based on its gradient history.

1.2.5.2 Formula

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_{t,i}$$

$\theta_{t+1,i}$:is the updated value of the parameter *i* at time *t + 1*

$\theta_{t,i}$: is the current value of the parameter *i* at time *t*

η : represents the learning rate.

$E[g^2]_t$: is the time-weighted average of the squared gradient up to time *t*

$g_{t,i}$: represents the gradient of the loss function for parameter *i* at time *t*

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

1.2.6 Adam

Adam transformed neural network training by combining Momentum and RMSprop, overcoming the limitations of previous methods and improving convergence and stability.

To minimize fluctuations during convergence, momentum acts as a "speed", pushing the model toward local regions and towards the global optimum.

RMSprop promotes efficient learning by dynamically adjusting the learning rate of each parameter based on the gradient history without potentially disruptive mutations to training.

Thanks to the integration of Momentum and RMSprop, Adam's optimization approach is flexible and versatile. Its fast convergence, local region navigation, and per-parameter learning rate optimization make it a popular choice for training large-scale, complex neural networks.

1.2.6.1 How does it work?

Adam computes the slope and its square for each model parameter, sizes it using a weighted average similar to RMSprop, and updates the model weights based on the given average.

1.2.6.2 Formula

1. Compute the first moment (mean) of the gradient:

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

2. Compute the second moment (variance) of the gradient:

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

3. Bias correction for the first moment:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

4. Bias correction for the second moment:

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

5. Update the parameters:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon} \cdot \hat{m}_t$$

m_t and v_t are the first and second moment estimates of the gradient g_t.

$\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected first and second moments.

θ_{t+1} is the updated value of the parameter at time step *t + 1*

η denotes the learning rate.

β_1 and β_2 are the decay rates for moment estimates (commonly 0.9 and 0.999).

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

1.3 Pros and Cons

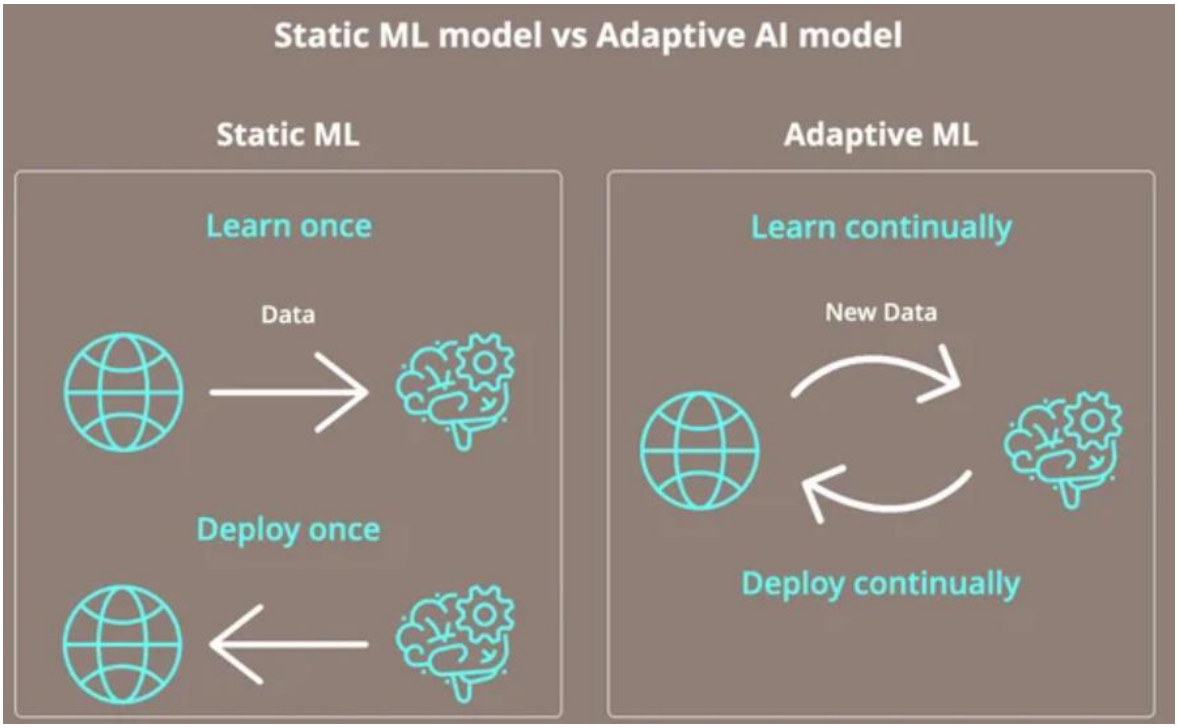
Algorithm	Advantages	Disadvantages
Gradient Descent (GD)	Simple implementation, finds global minima	Slow convergence, susceptible to local minima
Stochastic Gradient Descent (SGD)	Faster due to individual data updates	High variance, instability, difficulty in convergence
Momentum	Reduces oscillations, helps overcome local minima	May overshoot, prone to oscillations
Adagrad	Adapts learning rates for each parameter	Rapidly decreasing learning rates, premature convergence
RMSprop	Effective with sparse gradients, unstable data	Requires additional hyperparameter tuning
Adam	High performance, adaptability, stability	Hyperparameter sensitivity, potential for overfitting

CHAPTER 2 – Continual Learning and Test Production

2.1. Continual Learning

2.1.1 Introduction

In machine learning, continuous learning solves the problem of training models to continue learning, retaining prior knowledge while adapting to new tasks and inputs. For adaptive continuous learning, methods such as regularization, memory enhancement networks, and model architecture design are crucial.



In traditional machine learning, models are trained for a specific task using fixed data. But models are difficult to adjust as needs and data change. Models that use continuous learning can continue to learn without losing previous information.

However, catastrophic forgetting continues to severely hinder this ongoing learning process.

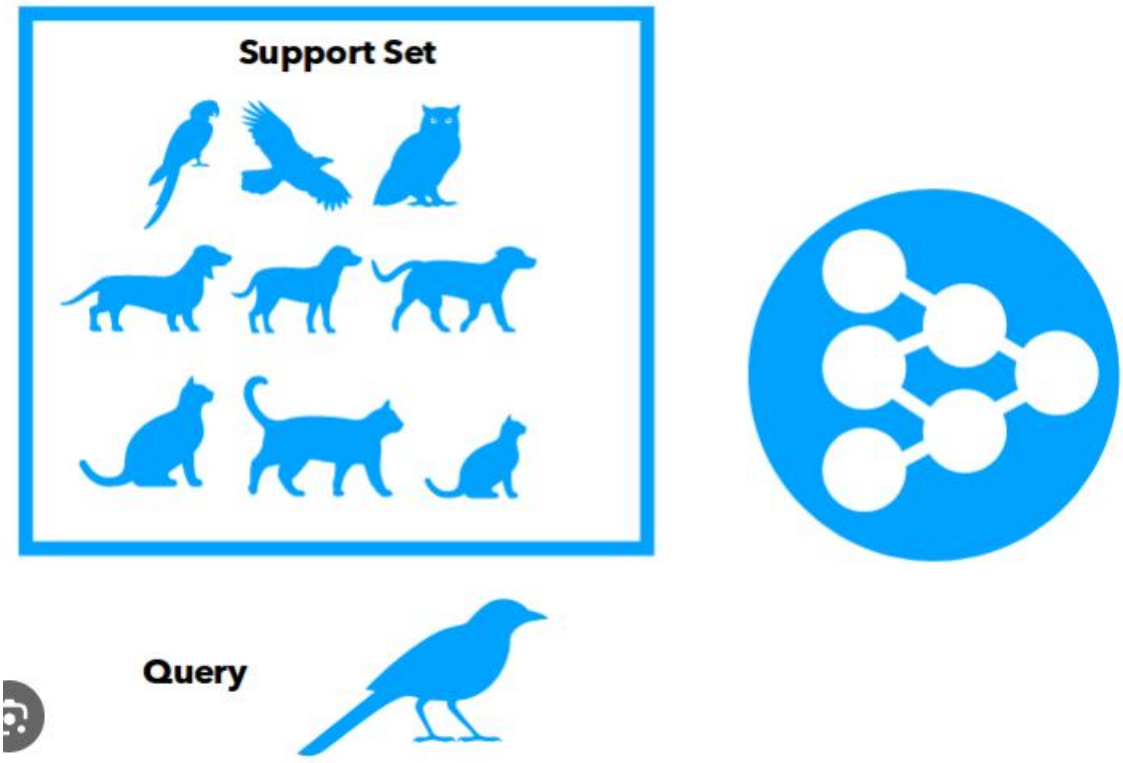
2.1.2 What is Catastrophic Forgetting?

This is bad. When a machine learning model learns new information, it may forget all or most of what it learned previously. This phenomenon is called forgetting. When the same task or data, new material, is taught repeatedly, the model's overall performance on the previous task may be affected by overwriting or loss of the ability to recall previously acquired patterns or information. This effect makes it difficult for the model to learn new information while remembering known information, which can cause problems in situations involving continuous learning.

For example:

Let's say you are training an AI model to recognize different animal species. Initially, the model was taught to recognize different animal species, such as dogs, cats, and birds, from a predefined data set. However, if you want to expand its recognition to include other species, such as deer, the model will need to be retrained with new data.

This is where catastrophic forgetting can occur: Once a model learns to detect deer, its initial recognition of dogs, cats, and birds can be severely compromised. tell me. The model can gather information about deer, but its previous knowledge about other species is lost or negatively affected. As a result, the model's overall ability to identify different animal species decreased.



2.1.3 Why we need Continual Learning?

1. Flexibility: Thanks to continuous learning, the model can continuously learn and adapt to new inputs without losing previous information.

2. Efficiency and Savings: By retaining learned knowledge and assimilating new information, time and resources can be saved.

3. Lifelong learning: It allows the model to learn over time just like humans.

4.Advanced Generalization: This approach makes AI systems more flexible and flexible by allowing the model to learn from a variety of data sources and adapt to different tasks.

6. **Avoid catastrophic losses:** Continuous learning ensures a thorough and continuous learning process by preventing the pattern of losing previous knowledge while acquiring new information.

2.1.4 Continual Learning Challenges

Evaluation Challenge:

- Applying continuous learning increases the likelihood of large model errors due to frequent updates. Additionally, it exposes the model to coordinated adversary attacks, which is why thorough testing before deployment is important.

Data Scaling Challenge:

- Managing global statistics complicates stateful training, requiring both incoming and outgoing data to be considered. A typical technique is to calculate or estimate these statistics incrementally as new data becomes available, eliminating the need to load the entire data set during training.

Algorithmic Challenge:

- Some algorithms, such as tree-based, dimensionality reduction, and matrix-based models, are difficult to update frequently (e.g., hourly). In contrast to weight-based models or neural networks, these algorithms cannot be trained incrementally using new data; instead, they require a complete data set for training. This limitation is serious when frequent upgrades are required.

2.1.5 The Stages of Continual Learning

Stage 1: Manual, stateless retraining

Models only get retrained when two conditions are met:

(1) The model's performance has degraded so much that it is now doing more harm than good

(2) Your team has time to update it

Stage 2: Fixed schedule automated stateless retraining

The high level steps of this script are:

(1) Pull data

(2) Downsample or unsample data if necessary

(3) Extract features

(4) Process and/or annotate labels to create training data

(5) Kick off the training process

(6) Evaluate the new model

(7) Deploy it

Stage 3: Fixed schedule automated stateful training

To achieve this you need to reconfigure your script and a way to track your data and model lineage.

Stage 4: Continual Learning

In this stage the fixed schedule part of previous stages gets replaced by some re-training trigger mechanism. The triggers can be:

- Time-based
- Performance-based

- Volume-based
- Driff-based

2.2. Test Production

2.2.1 Introduction

In machine learning, "production testing" is the process of evaluating and testing processes, models, and algorithms in real-world environments. During this phase, the performance of the model trained in a real-world environment is evaluated and its effectiveness, scalability, and reliability are verified before being put into practical use.

2.2.2 Why we need to test production?

- Validate your real-world performance by validating machine learning models through production testing to ensure correct operation during real user interactions.

- Identify specific issues: This process quickly resolves data changes or unusual user behavior by identifying specific issues during live deployment.

- Ensure continuous improvement through ongoing monitoring and improvement so that the model remains effective over time.

- Assess scalability: By assessing scalability and resource handling, testing ensures accuracy even as workloads grow or data volumes fluctuate.

- Integrated real-time feedback: By integrating real-time user feedback, the predictive power of the model is improved, resulting in better performance.

- Preemptive issue resolution helps reduce risk, enabling reliable performance in real-world scenarios by proactively identifying and resolving any issues through production testing before full deployment.

2.1.3 Pros and Cons

Pros	Cons
Evaluate real-world performance	Risk to the production environment if errors occur during testing
Detect specific problems	Cost and time-intensive compared to isolated testing
Continuous improvement	Difficulty in reproducing the production environment for testing
	Potential security vulnerabilities if not implemented or managed well

2.1.4 Method to perform.

There are several widely used techniques for machine learning production testing:

A/B testing: is the process of comparing the performance of two iterations of a model (usually the current version and an upgraded version). Groups of users are formed, each using a separate instance to evaluate real-world performance.

Canary testing: involves keeping existing versions of the model intact in most of the production environment while introducing new versions in a small part of it. This technique helps evaluate the performance of new releases before full deployment.

Green deployment: Green construction and green environment are comparable. The current version of the model is maintained in the green environment, while new

versions are installed in the green environment. The new version will be fully turned green after verification.

Chaos testing: is a technique for assessing system resiliency by simulating unusual or error-prone events in a real-world environment.

Shadow Testing: Deploy new versions alongside existing versions while ensuring production users are not affected by shadow testing. The behavior and results of the new and current models are compared.