**Vietnam General Confederation of Labor**

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**FINAL EXAM REPORT**

**MACHINE LEARNING**

*Instructor*: **MR. LE ANH CUONG**

*Student*: **NGUYEN LE PHUOC TIEN - 521H0514**

*Class* :**21H50302**

*Year* **: 2023-2024**

**HO CHI MINH CITY, 2023**

Vietnam General Confederation of Labor

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**FINAL EXAM REPORT**

**MACHINE LEARNING**

*Instructor*: **MR. LE ANH CUONG**

*Student*: **NGUYEN LE PHUOC TIEN - 521H0514**

*Class* :**21H50302**

*Year* **: 2023-2024**

**HO CHI MINH CITY, 2023**

ACKNOWLEDGEMENT

*I would like to thank Ton Duc Thang University, Faculty of Information Technology, for including this subject in the training chart and for creating the best learning conditions for me as well as all students. students of the faculty. Our report is a product of what I have learned in the past half semester. We would like to sincerely thank Mr. Le Anh Cuong for his dedication to help me understand this subject, thereby laying a good premise for future steps.*

*Ho Chi Minh city, 14th December, 2023*

*Author*

*(Sign and write full name)*

*Nguyen Le Phuoc Tien*

CONFIRMATION AND ASSESSMENT SECTION

**Instructor confirmation section**

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

*Ho Chi Minh January, 2022*

*(Sign and write full name)*

**Evaluation section for grading instructor**

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

*Ho Chi Minh January 2022*

*(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 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 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, 14th December, 2023*

*Author*

*(Sign and write full name)*

Nguyen Le Phuoc Tien

SUMMARY

Improving model training requires an understanding of machine learning optimizers. Several well-known algorithms, including Adam, Stochastic Gradient Descent, and Gradient Descent, modify model parameters to reduce mistakes. A number of criteria need to be taken into account when selecting an optimizer, including memory requirements, convergence speed, and dataset size. The secret to determining which optimizer and hyperparameter combination is ideal for your particular machine learning assignment is to experiment with them.

INDEX

[SUMMARY iv](#_Toc17557)

[Chapter 1: Learn and compare Optimizer methods in training machine learning models 2](#_Toc9786)

[1) Learn about Optimizer methods 2](#_Toc32753)

[1.1 Gradient Descent (GD): 2](#_Toc19480)

[1.2 Momentum 9](#_Toc13038)

[1.3 RMSprop 12](#_Toc22265)

[1.4 Adam 14](#_Toc29156)

[2) Compare the performance of methods 17](#_Toc6641)

[2.1 Compare using table 18](#_Toc19850)

[2.2 Compare graphically on simple and complex problem types 19](#_Toc11503)

[2.3 Conclude 20](#_Toc26598)

[Chapter 2: Learn about Continuous Learning and Test Production when building a machine learning solution to solve a problem 21](#_Toc5247)

[1) Continuous Learning 21](#_Toc31589)

[1.1 Concept of Continuous Learning 21](#_Toc12561)

[1.2 Mechanism of Continuous Learning 22](#_Toc32680)

[1.3 Application of Continuous Learning 22](#_Toc30974)

[1.4 Notes on Continuous Learning 23](#_Toc474)

[2) Test Production 23](#_Toc23580)

[2.1 Test Production process 23](#_Toc13446)

[2.2 Role of Test Production 23](#_Toc25771)

[2.3 Example of applying Test Production 24](#_Toc6979)

**Chapter 1: Learn and compare Optimizer methods in training machine learning models**

Optimizers, basically, are the basis for building a neural network model with the purpose of 'learning' the features (or patterns) of the input data, from which one can find a solution. Weights and bias pairs are suitable for optimizing parameters in the model by adjusting them based on training data so that the model can learn and make better predictions.

1. **Learn about Optimizer methods**

**1.1 Gradient Descent (GD):**

**1.1.1 Definition of Gradient Descent**

This is a basic optimization method in machine learning and optimization, often used to find the optimal value of a loss function by adjusting parameters iteratively through each loss function. training data to minimize the cost function.

**1.1.2 How Gradient Descent works**



- The above equation describes the function of the gradient descent algorithm, in which:

. b: is the next value (new value) of the “parameter”.

. a: is the current value of the “parameter”.

. The minus sign refers to the minimization part of the gradient descent algorithm.

. : is the waiting coefficient, which determines the size of the move.

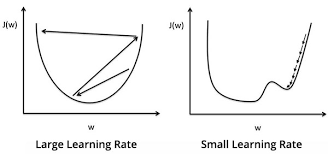
. : slope, represents the fastest increasing direction and magnitude of the function at a time.

- This formula updates the value of the parameter based on the gradient of the loss function according to each parameter. By moving against the sign of the gradient, the algorithm will find a direction that minimizes the loss function and gets closer to the optimal point.

- **Note:** in each iteration, the parameters are updated according to this formula to converge to the optimal point or near the optimal point of the loss function. However, choosing the learning rate is an important factor, which needs to be adjusted appropriately to ensure the algorithm converges effectively.

**1.1.3 Learning rate of Gradient Descent**

In order for the Gradient Descent algorithm to reach a local minimum, it is necessary to set the learning rate to an appropriate value (neither too low nor too high). This is really important because if the learning rate is set to too large a value, the algorithm may not reach the local minimum because the value will fluctuate back and forth. If the learning rate is set to too small a value, the gradient descent will eventually reach the local minimum but it will take a long time and can easily get stuck at the local minimum. We can check whether the model's learning speed is good or not by plotting it on a graph.



**1.1.4 How to solve gradient descent challenges**

- One of the best ways to ensure the gradient descent algorithm runs properly is to plot the cost function as the optimization runs. Set the number of iterations on the x-axis and the price of the cost function graph when run on the y-axis. This will help to see the value of the cost function after each iteration of gradient descent and provisioning way to easily see how consistent the model's learning rate is. The cost function decreases after each iteration, proving that the Gradient Descent algorithm works normally.

- There are some algorithms that can automatically tell you whether the gradient descent has converged or not, but a threshold for convergence must be predetermined, which is also quite difficult to estimate. Therefore, simple plotting is the preferred test of convergence.

- Another advantage of tracking gradient descent through a graph is that it allows us to easily detect if it is not behaving properly, such as if the cost function is increasing. Most of the reason the cost function is increasing when using gradient descent is that the learning rate is too high.

**1.1.5 Classification and variations of Gradient Descent**

1. **Batch Gradient Descent (BGD)**

**a.a) Introduction to Batch Gradient Descent**

calculate the gradient of the loss function over the entire dataset, then use this gradient to update the model parameters. This process starts by calculating the derivative of the loss function with respect to each parameter based on the entire training data. Then, through the update formula, the parameters are adjusted by subtracting a learning rate multiplied by the gradient, which moves the model toward the direction that minimizes the loss function.

**a.b) How Batch Gradient Descent works**

***Gradient calculation:*** First, we calculate the gradient of the loss function according to each parameter in the model. This is often done using backpropagation in neural networks, in which the gradient is backpropagated from the output layer to the input layer.

***Update weights:*** After calculating the gradient, we update the model's weights by moving in the opposite direction of the gradient. Updating the weights is usually done using a simple formula:

**newWeights = oldWeights - learningRate \* gradient.**

Learning rate is an important parameter, determining the level of weight updates after each step.

***Repeating the process:*** The process of calculating the gradient and updating the weights is repeated until a stopping condition is reached. The stopping condition can be a maximum number of iterations, achieving sufficiently good convergence, or when a sufficiently small threshold for the gradient is achieved.

**a.c) Advantages of Batch Gradient Descent:**

***Accurate convergence:*** BGD has the ability to accurately converge to the global minimum point of the loss function, between local minimum points.

***Stable convergence:*** BGD often converges more stably than gradient-based optimization methods such as SGD and Mini-Batch Gradient Descent. This has special significance when the loss function is not flat and has many minima.

**a.d) Disadvantages of Batch Gradient Descent:**

***Computational efficiency:*** BGD requires calculating gradients for the entire training set during each update, which can be time-consuming and computationally resource-intensive, especially when working with large datasets.

***Difficulty with memory:*** Because BGD calculates gradients for the entire training set, it needs to store the entire dataset in memory, which can make it difficult to work with very large datasets.

1. **Stochastic Gradient Descent (SGD)**

**b.a) Introduction to Stochastic Gradient Descent**   
- Stochastic Gradient Descent (SGD) is an optimization method in machine learning used to update model parameters based on the gradient of each data point randomly from training data set. Unlike Batch Gradient Descent (BGD), which calculates the gradient on the entire data, SGD only calculates the gradient based on one data point at each update. This makes the parameter update process faster because only the derivative needs to be calculated for one data point. However, frequent updates are computationally more expensive than the batch gradient descent method. Additionally, the frequency of those updates can lead to noise gradients, which can cause error rates to spike instead of slowly decreasing.

- Stochastic Gradient Descent is often more suitable when applied to large data because it can handle large datasets without having to store all the data in memory and can converge faster in some cases.

**b.b) Stochastic Gradient Descent works**

***Randomly select a data point:*** First, we select a random data point from the training set. This random selection helps avoid falling into local minima and helps the learning model generalize better.

***Calculating the gradient:*** Next, we calculate the gradient of the loss function at the selected data point. The gradient represents the fastest decreasing direction of the loss function. Usually, the gradient is calculated using the backpropagation method in the neural network.

***Update weights:*** After calculating the gradient, we update the model's weights by moving in the opposite direction of the gradient. The weight update formula can be determined by multiplying the gradient by a learning rate coefficient and subtracting this result from the current weights.

**b.c) Advantages of Stochastic Gradient Descent**

***Computational efficiency:*** SGD only needs to calculate the gradient for a single data point per update, so it is often faster than other optimization methods when working with large data sets.

***Ability to avoid falling into local minimum points:*** By randomly selecting data points, SGD has the ability to avoid falling into local minimum points and can better find global minimum points.

**b.d) Disadvantages of Stochastic Gradient Descent**

***Unstable:*** SGD can produce large and unstable weight update steps due to the random variation of the gradient from each data point. This can slow convergence and require careful adjustment of the learning rate.

***Difficult to find the ideal learning speed:*** The learning rate in textbooks needs to be adjusted carefully. If the learning rate is too large, SGD may not converge or "miss" the minimum point. On the contrary, if the learning rate is too small, the SGD may converge very slowly.

1. **Mini-batch gradient Descent**

**c.a) Introduction to Mini-batch gradient Descent**

Mini-batch gradient descent is the method applied because it is a combination of SGD and BGD concepts. It simply splits the training dataset into small batches and performs updates for each batch. This helps combine the benefits of both methods, reducing the slowness of BGD and increasing stability compared to SGD. Mini-batch GD is often widely used in large machine learning models because it helps optimize calculations on large data effectively, while improving the convergence speed of the algorithm compared to BGD. This is the required algorithm when training neural networks and is the most common type of gradient descent in deep learning.

**c.b) How Mini-batch gradient Descent works**

***Divide the training set into mini-batches:*** First, we divide the training set into small mini-batches. Each mini-batch contains a small number of data points, typically between 10 and 1000 data points, depending on the training set size.

***Calculate the gradient for each mini-batch:*** Next, we calculate the gradient of the loss function corresponding to each mini-batch. This gradient is calculated using the backpropagation method in the neural network, just like in BGD.

***Update weights:*** After calculating the gradient for each mini-batch, we update the model's weights by moving in the opposite direction of the gradient. The formula for updating weights in MBGD is similar to that in BGD:

**newWeights = oldWeights - learningRate \* gradient.**

However, the gradient in MBGD is the average of the gradients calculated from the mini-batches.

***Repeat process:*** The process of calculating the gradient and updating the weights is repeated until the stopping criterion is reached. The stopping criterion can be a maximum number of iterations, achieving sufficiently good convergence, or when a sufficiently small threshold for the gradient is achieved.

**c.c) Advantages of Mini-batch gradient Descent**

***Reduced calculation time:*** Compared to BGD, MBGD only calculates the gradient for a sub-mini-batch instead of the entire training set, which helps reduce calculation time significantly. This is especially useful when working with large data sets.

***Faster Convergence:*** MBGD has the ability to converge faster than BGD because the gradient is calculated based on some random data points. This avoids getting stuck at a local minimum and allows finding a global minimum.

**c.d) Disadvantages of Mini-batch gradient Descent**

***Accurate convergence is not guaranteed:*** Because the gradient is calculated based on a number of random data points, MBGD does not guarantee accurate convergence to the global minimum point like BGD. However, it usually gives good enough results in practice.

***Setting up the mini-batch size:*** Choosing the right mini-batch size is challenging. If the mini-batch size is too small, the gradient may fluctuate and lead to instability during the annealing process. On the contrary, if the mini-batch size is too large, the weight update process can become slow and inefficient.

**1.1.6 Advantages of Gradient Descent**

- As a classic and powerful optimization method, Gradient Descent is widely applied in many Machine Learning algorithms and optimization problems.

- Gradient Descent can converge to the global optimal point (if any) of the loss function.

- Can be applied on large data and large parameter space.

**1.1.7 Disadvantages of Gradient Descent**

- Choosing a poor learning rate can lead to the algorithm not converging or taking a long time to converge.

- For complex loss functions, Gradient Descent can get stuck at the local minimum point instead of finding the global optimal point.

- For BGD, calculating the derivative on the entire data can be expensive and slow on large data sets.

- Poor initial parameter initialization can affect the convergence process.

- Sometimes, the complex characteristics of the loss function can make Gradient Descent ineffective or non-converging.

**1.2 Momentum**

**1.2.1 Definition of Momentum**

Momentum is a technique used in the process of optimizing neural network training. It is designed to help speed up the convergence process and avoid falling into local minima in the optimization space.

**1.2.2 Differences of Momentum compared to Gradient Descent**

- X considers the process of updating weights in the neural network through a gradient optimization algorithm such as Gradient Descent. In Gradient Descent, we update the weights by moving them in the direction opposite to the gradient of the loss function to minimize the loss function.

- When applying momentum to the optimization process, we integrate a previous training component to help adjust the weight updates. Instead of just using the current gradient, momentum retains a portion of the previous gradient and combines it with the current gradient to calculate the update steps.

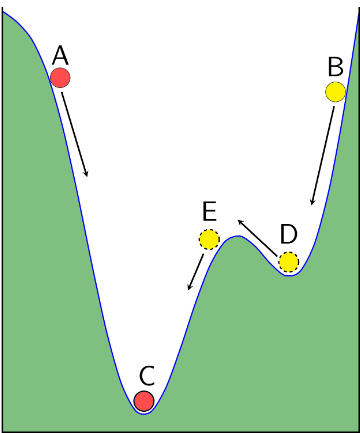
**1.2.3 Momentum formula**

The mathematical formula to calculate weight updates with momentum in the Gradient Descent algorithm is determined as follows:

**(t) = β \* v(t-1) + (1 - β) \* ∇J(w(t-1))   
w(t) = w(t-1) - α \* v(t)**

***In there:***

* v(t) is the training vector at time t.
* β is the momentum coefficient (usually in the range [0, 1]).
* ∇J(w(t-1)) is the gradient of the loss function J at time t-1.
* w(t) is the weight vector at time t.
* α is the learning rate, which determines the size of the update steps.



**1.2.4 Some notes about Momentum**

- Almost like running down a slope, momentum helps the model "remember" the previous direction of movement and continue moving in that direction for a short time. If the previous and current gradients have similar directions, the accumulation of the previous gradient will increase which speeds up the convergence process. Conversely, if the previous and current gradients have different directions, accumulation will guide the weight update process in the new direction.

- Momentum helps avoid falling into local minima by allowing the model to overcome weaknesses and continue searching for the global optimum. It also helps minimize unnecessary fluctuations during optimization and speeds up convergence.

- The important point when using momentum is to choose a reasonable value for the momentum coefficient β. If β is too small, the cumulative momentum will not be strong enough to have a significant impact. On the contrary, if β is too large, the accumulated momentum can unbalance the optimization process and lead to unwanted fluctuations.

**1.2.5 Advantages of Momentum**

- Momentum accumulates previous training, helping to reduce spikes and fluctuations during optimization. This can help speed up the convergence process and reduce training time.

- With loss functions that have saddle points or narrow gaps in the optimal space, Momentum can help overcome saddle points and find the local optimum better.

- Momentum can help adjust the learning rate automatically based on the magnitude of the gradient, automatically increasing the learning rate for large gradient cases and decreasing the learning rate for small gradient cases. Therefore the momentum is stable and adjusts the learning rate well.

**1.2.6 Disadvantages of Momentum**

- Requires parameter adjustment: To achieve the best optimal performance, Momentum needs to have its momentum parameter adjusted. If the parameter is not chosen carefully, it can lead to overaccumulation of prior training or not enough accumulation of prior training.

- On complex curved surfaces, Momentum can get stuck in local regions or cannot find a global optimum. This can happen when training a deep neural network with many saddle points.

- Momentum may require time for parameter tuning and may require a large number of iterations to achieve optimal results. Therefore, it can take a long time to adjust.

**1.3 RMSprop**

**1.3.1 Introduction to RMSprop**

RMSprop (Root Mean Square Propagation) is an optimization algorithm used in the neural network training process. It is designed to minimize the impact of large changes in the gradient and make convergence faster. RMSprop is an optimization method that solves this problem by adjusting the learning rate based on the moving average of the square of the gradient. Specifically, RMSprop uses a new parameter called "moving average of squared gradients" to track gradient fluctuations over time.

**1.3.2 How RMSprop works**

***Initializing weights and gradients:*** First, the model's weights are initialized randomly and the initial gradient is calculated using the backpropagation method on an amount of training data.

***Parameter initialization:*** RMSprop uses a new parameter called "moving average of squared gradients" to track gradient fluctuations over time. Initially, this parameter is initialized to 0.

***Update weights:*** RMSprop updates the weights using the formula:

**newWeights = oldWeights - learningRate \* (gradient / sqrt(averageSquaredGradients + epsilon))**

In there:

- newWeights : is the updated weight.

- oldWeights : is the current weight.

- learningRate : is the learning rate , which determines the degree of change in weights.

- gradient : is the gradient of the loss function with respect to weight.

- averageSquaredGradients: is the moving average of the squared gradient calculated according to the formula:

**averageSquaredGradients = decayRate \* averageSquaredGradients +**

**(1 - decayRate) \* (gradient \*\* 2)**

In there :

- decayRate is a parameter between 0 and 1, determining the level of forgetting of old information and the importance of new information. Typically, decay\_rate is set near 1, for example 0.9 or 0.99.

- epsilon is a very small value (usually 1e-8) added to the denominator to avoid division by 0.

**=>** The weight update formula in RMSprop allows adjusting the learning rate based on the average of the squares of the gradient. If the gradient is large, the denominator will increase, reducing the learning rate. On the contrary, if the gradient is small, the learning rate will increase.

***Iterative process:*** The process of calculating the gradient and updating the weights is repeated until a stopping criterion is reached, such as reaching a maximum number of iterations or achieving sufficiently good convergence.

**1.3.3 Advantages of RMSprop**

***Automatic learning rate adjustment:*** RMSprop automatically adjusts the learning rate based on the average of the squares of the gradient, helping to speed up the convergence process and avoid 'exploding' or 'vanishing' gradients.

***Efficiency in optimizing non-stationary objectives:*** RMSprop often performs better than other optimization algorithms when the objectives are unstable (non-stationary objectives), that is, the objectives change over time.

***Minimize gradient fluctuations:*** RMSprop helps minimize the impact of large changes in gradients by using a moving average of the squares of the gradient. This helps reduce training fluctuations and achieve more stability.

**1.3.4 Disadvantages of RMSprop**

***Requires parameter selection:*** RMSprop has a parameter called decay\_rate that needs to be chosen carefully. If decay\_rate is set too high, the average\_squared\_gradients parameter will forget information too quickly, leading to instability. Conversely, if the decay\_rate is too low, the convergence process may be slow and ineffective.

***Does not guarantee exact convergence:*** Similar to other optimization methods, RMSprop also does not guarantee exact convergence to the global minimum point. This depends on the initial initialization, learning rate, and structure of the model.

**1.4 Adam**

**1.4.1 Introduction to Adam**

Adam (Adaptive Moment Estimation) is an optimization algorithm commonly used in machine learning to update weights during neural network training. Adam combines the advantages of two other algorithms, RMSprop and Momentum, to help speed up the convergence process and improve model performance.

**1.4.2 How Adam works**

***Initializing weights and gradients:*** First, the model's weights are initialized randomly and the initial gradient is calculated using the backpropagation method on a batch of training data.

***Initialize parameters:*** Adam uses several parameters to adjust the weight update process. Main parameters include learning rate, beta\_1, beta\_2 and epsilon.

a. Learning rate: determines the weight update rate. It is an important parameter in Adam and needs to be adjusted carefully. If the learning rate is too high, the convergence process may not be stable. Conversely, if the learning rate is too low, convergence may occur too slowly.

b. Beta\_1 and beta\_2: these two parameters are used to adjust the weight of the gradient and the square of the gradient during the update process. Normally, beta\_1 has a value close to 1 and beta\_2 has a value close to 0.9.

c. Epsilon: a very small value added to the denominator to avoid division by zero during calculation.

***Update weights:*** Adam updates the weights with the following steps:

a. Compute the gradient of the weighted loss function using the backpropagation method.

b. Calculate the moving average of the gradient according to the formula:

**m = beta\_1 \* m + (1 - beta\_1) \* gradient**

Where , m is the moving average of the gradient. The beta\_1 parameter helps adjust the significance of the current and previous gradients. It is similar to momentum in the Gradient Descent algorithm with Momentum.

c. Calculate the moving average of the squared gradient according to the formula:

**v = beta\_2 \* v + (1 - beta\_2) \* (gradient \*\* 2)**

Where, v is the moving average of the squared gradient. The beta\_2 parameter helps adjust the significance of the square of the current gradient and the square of the previous gradient. This is part of RMSprop.

d. Adjusting the moving average of the gradient:

**m\_hat = m / (1 - beta\_1^t)**

Where, t is the current update (usually starts at 1 and increases after each update).

e. Calibrate the moving average of the squared gradient:

**v\_hat = v / (1 - beta\_2^t)**

f. Update weights according to the formula:

**new\_weights = old\_weights - learning\_rate \***

**(m\_hat / (sqrt(v\_hat) + epsilon))**

***Repeat the process:*** Repeat steps 3a-3f for all batches of training data until the stopping criterion is reached (e.g. number of iterations, accuracy reaches a certain threshold, etc.).

**1.4.3 Note about Adam**

- Adam combines both the ability to adjust the significance of the current gradient and the previous gradient (thanks to beta\_1) along with the ability to adjust the importance of the square of the current gradient and the square of the previous gradient (thanks to beta\_2). This helps minimize fluctuations and speed up the convergence process. It also helps combat problems like lost gradients or scattered gradients.

- Although Adam is a powerful and popular optimization algorithm, there are still some points to note. First, the learning rate needs to be carefully adjusted to ensure stable convergence. Second, Adam may require more memory than other optimization algorithms due to storing the moving average of the gradient and the square of the gradient. Therefore, in some cases, other algorithms such as SGD (Stochastic Gradient Descent) can be used more effectively.

**1.4.4 Advantages of Adam**

***High performance:*** Adam combines the advantages of RMSprop and Momentum, which accelerates convergence and improves model performance. Usually Adam gives better results than other optimization algorithms, especially in deep and complex models.

***Flexible update ability:*** Adam has the ability to adjust the learning rate automatically for each individual weight. This helps the algorithm adapt to each weight and optimize performance based on the structure of the model.

***Combats gradient loss and distributed gradients:*** Adam uses gradient moving averages and gradient squares, which help minimize fluctuations and more stabilize the weight update process. This helps combat problems such as lost gradients or scattered gradients.

***Does not require too many adjustment parameters:*** Adam only requires some basic parameters such as learning rate, beta\_1 and beta\_2. These parameters often have default values that are easy to use and can work well in many situations.

**1.4.5 Adam's disadvantages**

***Larger memory requirements:*** Adam stores the moving average of the gradient and the square of the gradient for each weight. This requires larger memory compared to other optimization algorithms such as SGD. In some cases, especially when working with large and complex models, using Adam can cause memory problems.

***Convergence speed is not guaranteed:*** Although Adam usually gives good results, however, convergence speed is not guaranteed like in the Gradient Descent algorithm with Momentum. In some situations, Adam may converge more slowly or unstable.

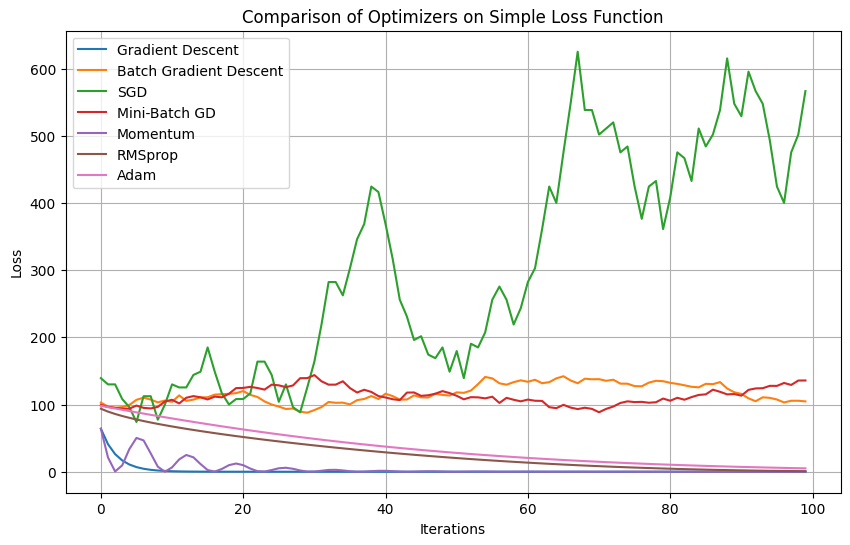
***Difficult to control and adjust:*** Adam has several adjustable parameters such as learning rate, beta\_1 and beta\_2. Adjusting these parameters correctly can significantly affect the performance of the algorithm. This requires knowledge and experience to fine-tune these parameters effectively.

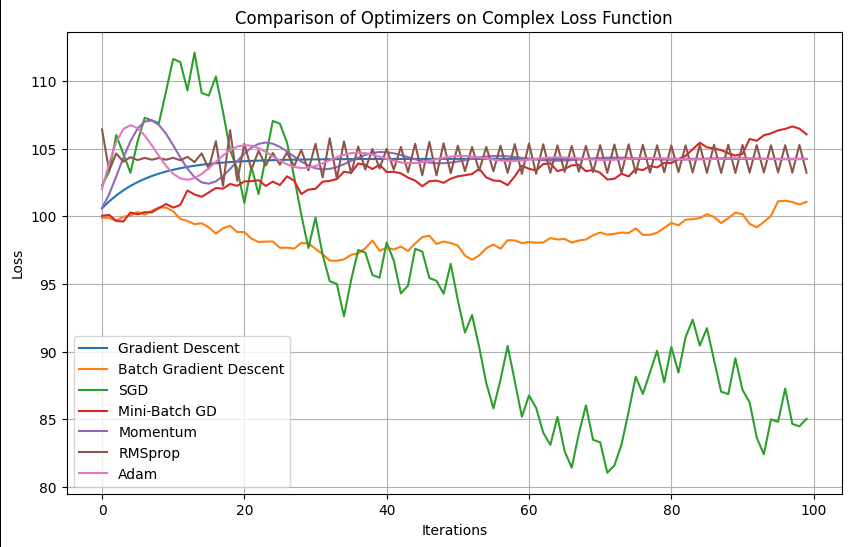
1. **Compare the performance of methods**

**2.1 Compare using table**

| **Method** | **Efficiency** | **Convergence speed** |
| --- | --- | --- |
| Gradient Descent (GD) | Good at small and simple problems. | Convergence is slow with large data sets and can get stuck in local minima. |
| Batch Gradient Descent (BGD) | Converges faster than GD with small data sets. | Requires large memory when processing large data sets. |
| Stochastic Gradient Descent (SGD) | - Effective with large and uneven data sets.  - Easily escape local minima. | - Fast convergence, but exact convergence is not guaranteed with each update.  - More oscillation and instability during convergence. |
| Mini-batch Gradient Descent | combines calculations on a small number of data points to achieve better efficiency and convergence speed than BGD and SGD alone. | Effective with large and heterogeneous data sets. |
| Momentum | Speed up the convergence process and avoid falling into local minima. | Need to tweak parameters to achieve best performance. |
| RMSprop | Minimize distributed gradient and gradient loss problems. | Larger memory requirements compared to GD and SGD. |
| Adam | Combine the advantages of Momentum and RMSprop, to achieve better optimization performance, overcome flat areas, minimize fluctuations, and ensure fast and stable convergence speed. | Larger memory requirements compared to GD and SGD. |

**2.2 Compare graphically on simple and complex problem types**





**2.3 Conclude**

***- Gradient Descent (GD):*** Effective for small and simple problems. However, slow convergence with large datasets and can get stuck in local minima.

***- Batch Gradient Descent (BGD):*** Converges faster than GD with small datasets. Requires large memory when processing large datasets.

***- Stochastic Gradient Descent (SGD):*** Effective with large and uneven datasets.

* Easily escapes local minima.
* Fast convergence, but exact convergence is not guaranteed with each update.
* More oscillation and instability during convergence.

***- Mini-batch Gradient Descent:*** Combines calculations on a small number of data points to achieve better efficiency and convergence speed than BGD and SGD alone. Effective with large and heterogeneous datasets.

***- Momentum:*** Speeds up the convergence process and avoids falling into local minima. Parameter tweaking needed for best performance.

***- RMSprop:*** Minimizes distributed gradient and gradient loss problems. Larger memory requirements compared to GD and SGD.

***- Adam:*** Combines the advantages of Momentum and RMSprop, achieving better optimization performance, overcoming flat areas, minimizing fluctuations, and ensuring fast and stable convergence speed. Larger memory requirements compared to GD and SGD.

=> Each optimization algorithm has its own strengths and limitations. Adam usually performs best across diverse problem types due to its ability to combine the advantages of Momentum and RMSprop. However, choosing the right algorithm depends on the data characteristics and tuning hyperparameters for optimal performance.

**Chapter 2: Learn about Continuous Learning and Test Production when building a machine learning solution to solve a problem**

1. **Continuous Learning**

**1.1 Concept of Continuous Learning**

- Continuous Learning is an area in machine learning where the learning model must continuously learn and update its knowledge when encountering new data, while retaining previously learned knowledge. This is especially important in machine learning systems such as autonomous artificial intelligence or reinforcement learning robotics, where the environment and data are constantly changing.

- In a Continuous Learning environment, the model faces two main challenges: forgetting old knowledge and interfering with previously learned knowledge (also known as the phenomenon of "catastrophic forgetting"). To solve this problem, Continuous Learning methods focus on developing mechanisms to retain old knowledge and control the learning of new knowledge.

**1.2 Mechanism of Continuous Learning**

There are many proposed mechanisms to solve the Continuous Learning problem, below are some popular methods:

**1.2.1 Replay-based methods**

This method reuses old data samples to recall previously learned knowledge. Data samples are stored in memory and reused to train the model during new learning. However, storage and memory management can become complicated as the amount of data increases.

**1.2.2 Regularization-based methods**

This method adds regularization coefficients to the loss function to minimize forgetting and collisions. These regularization coefficients help retain prior knowledge by evaluating the importance of the weights in the model.

**1.2.3 Dynamic architecture methods**

This method creates models that have the ability to change their architecture to accommodate new data. Instead of fixing the model architecture, these methods allow the model to grow or shrink in size as new data is encountered.

**1.3 Application of Continuous Learning**

Continuous Learning can be applied in many situations in building machine learning solutions. Here are some examples of how to apply Continuous Learning in different fields:

**1.3.1 Natural language processing (NLP)**

In the field of NLP, Continuous Learning can be used to train a natural language processing model to understand and produce different types of text. The model can learn from data sets containing semantic, grammatical, or cultural information from different sources and retain the learned knowledge when encountering new data.

**1.3.2 Image recognition**

In the field of image recognition, Continuous Learning can be used to train models to recognize and classify objects from different data sets. The model can learn from datasets containing images of different objects such as cars, dogs, or trees, and maintain the learned knowledge as new objects are added.

**1.3.3 Enhanced self-learning system**

In a self-reinforcement learning system, Continuous Learning can be used to develop robots or other automated systems capable of learning and adapting to changing environments. The model can learn from interactions with the environment and maintain learned knowledge when faced with new situations.

**1.4 Notes on Continuous Learning**

Applying Continuous Learning in machine learning solutions also poses some challenges. These challenges include memory management, trade-offs between learning new knowledge and retaining old knowledge, and dealing with forgetting and conflict. To address these challenges, it is necessary to develop efficient and stable Continuous Learning methods and mechanisms.

1. **Test Production**

**2.1 Test Production process**

The Test Production process is an important step in building a machine learning solution, where trained models are applied to new data to make predictions and test their performance. During this process, new data is fed into the model and the predicted output is compared with the actual value to evaluate the model's accuracy and performance.

**2.2 Role of Test Production**

Test Production plays an important role in building machine learning solutions because it provides evaluation information about model performance on new data. The main roles of Test Production include:

**2.2.1 Performance assessment**

Test Production allows evaluating the model's performance on new data it has never encountered before. By comparing the predicted output with the actual value, one can evaluate the accuracy, sensitivity and specificity of the model. This result helps define the model's ability to solve problems and make decisions.

**2.2.2 Check generality**

Test Production helps check the generality of the model. If the model only performs well on the training data on which it was trained, but cannot be applied well on new data, overfitting can occur. Test Production helps detect model non-generality and make necessary adjustments to improve general performance.

**2.2.3 Model optimization**

Test Production provides important feedback for model optimization. When the model fails to achieve expected performance on new data, information from Test Production helps detect errors and tune the model to improve performance. This may include fine-tuning hyperparameters, changing model architecture, or collecting additional data.

**2.3 Example of applying Test Production**

- Suppose we are building a machine learning model to identify spam emails. The Test Production process plays an important role in evaluating model performance.

- First, we train the model on a dataset of labeled emails (spam or not). After the training process, we will use the trained model to predict the labels of new emails that the model has not yet encountered.

- Next, we compare the model's predicted output with the actual labels of new emails during Test Production. If the model predicts the correct label (spam or not spam), we record a correct result. Conversely, if the model predicts the wrong label, we record an incorrect result.

- Based on the results of the Test Production process, we can calculate measures to evaluate model performance such as accuracy, sensitivity (recall), specificity (precision) and F1- score. score. These metrics help evaluate the overall performance of the model in identifying spam emails.

- If Test Production results show that the model has unsatisfactory performance, we can make adjustments to improve the model, for example increasing the training set size, tweaking the model's hyperparameters model or change the model architecture.

- In short, Test Production plays an important role in evaluating performance and improving machine learning models. It helps ensure the generality of the model on new data and supports the model optimization process to achieve the best performance in solving machine learning problems.