Họ tên: Lý Mạnh Phi

MSSV: 521H0286

**FINAL PROJECT**

**MACHINE LEARNING**

1. **Study and compare Optimizer methods in training machine learning models.**
   1. **What is optimizer methods?**

In machine learning, an optimizer is a crucial component of the training process for a model. The optimizer is responsible for adjusting the parameters of a model to minimize the error or loss function. The goal of training a machine learning model is to find the set of parameters that results in the best possible performance on the task at hand.

Optimizers use optimization algorithms to update the model parameters iteratively during the training process. The choice of optimizer and its hyperparameters can significantly impact the training speed and the quality of the learned model. Some commonly optimizer methods in machine learning include:

1. **Gradient Descent (GD)**

**What is gradient descent?**

* An optimization algorithm used to train machine learning models.
* Finds the local minimum of a function by iteratively adjusting its parameters.
* Used in various algorithms like linear regression and neural networks.

**Key concepts:**

* Gradient: Measures how much a function's output changes with respect to its inputs. Think of it as the steepness of a hill.
* Cost function: A function that measures the error of a model. Gradient descent aims to minimize this function.
* Learning rate: Determines the size of steps taken towards the minimum. Setting it correctly is crucial for convergence.

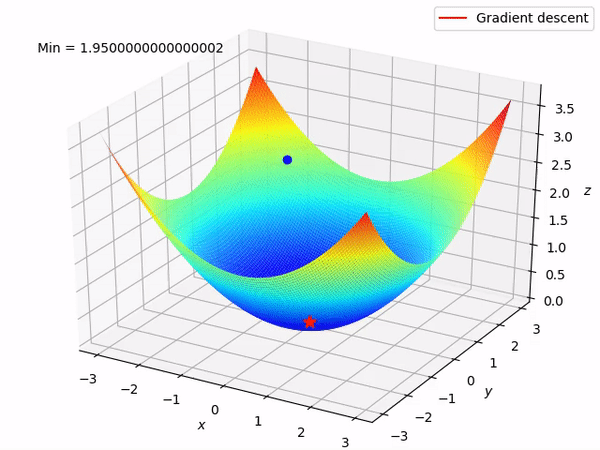
**Types of gradient descent:**

* Batch gradient descent: Updates model parameters after all training examples are processed. Stable but computationally expensive.
* Stochastic gradient descent: Updates parameters after each training example. Faster but can lead to noisy gradients.
* Mini-batch gradient descent: Combines aspects of batch and stochastic approaches. Updates parameters after processing small batches of data. Most common type in deep learning.

**Challenges and solutions:**

* Convergence: Monitoring the cost function over iterations helps in identifying convergence and potential issues like high learning rate.
* Choosing the right learning rate: Experimenting with different values and plotting the learning curve is a good approach.

**Formula**: *x*new​=*x*old​−learning\_rate×gradient(*x*old​)

Example: Imagine trying to descend a hill slowly to find the lowest point. The algorithm iteratively updates the parameters in the direction opposite to the gradient.

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

**Explanation**: SGD is a variation of Gradient Descent where, instead of updating parameters after each epoch, updates are performed for each data point in each epoch. It's particularly useful for large datasets and online learning.

* Modifies gradient descent to work with large datasets efficiently.
* Updates parameters using a single training example or a small batch at each iteration.

**Key Features of SGD:**

* Minibatches: Updates parameters using a subset of training examples (minibatch) in each iteration.
* Momentum: Accumulates past gradients to accelerate convergence and reduce oscillations.
* Random Start Values: Helps avoid local minima by starting from different points in the parameter space.

**Applications:**

* Machine learning: finding model parameters that minimize cost function (e.g., linear regression, logistic regression, neural networks).
* Optimization problems in various domains.

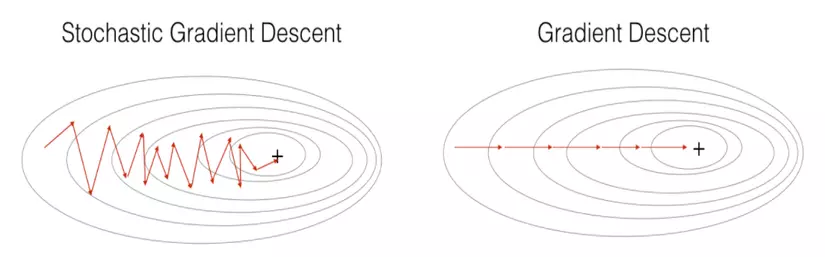
**Additional Insights:**

* SGD often gets stuck in local minima or saddle points, especially for nonconvex functions.
* Learning rate adjustments and random restarts can help mitigate this.
* Advanced techniques like adaptive learning rates and momentum can further improve performance.
* Gradient descent is widely used in machine learning libraries like Keras and TensorFlow.

**Considerations:**

* SGD's convergence speed can be slow, especially with noisy data or large datasets.
* It can be sensitive to hyperparameter choices (learning rate, batch size, momentum).
* Careful tuning and monitoring are often required.

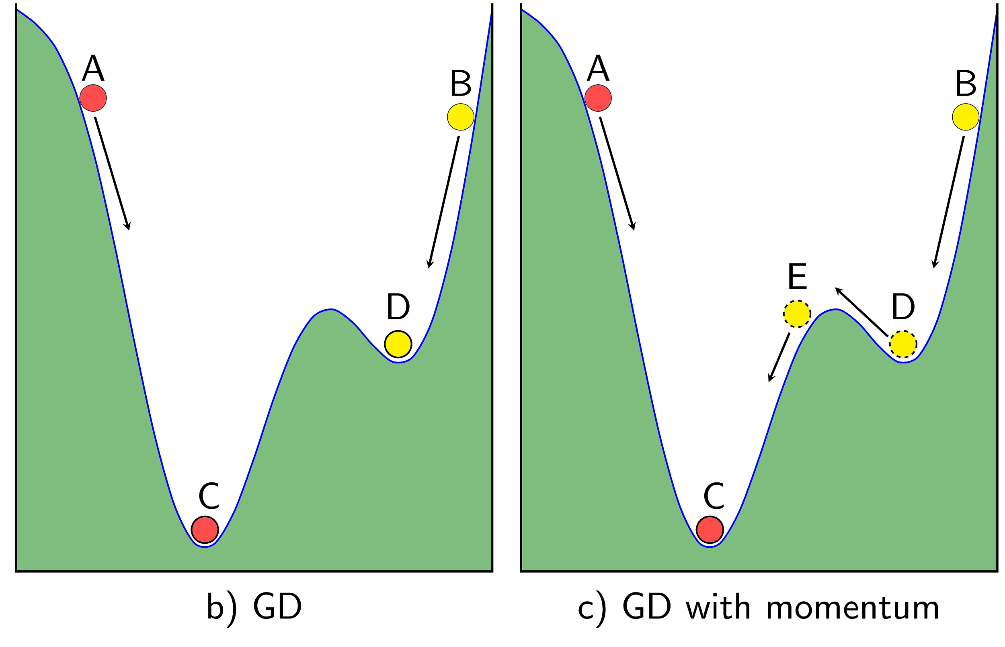
**Formula**: Similar to GD but updates are performed for each data point.

**Example**: Think of randomly looking at small patches of grass to make predictions about the entire field.

1. **Momentum**

**Explanation**: Momentum addresses the limitation of getting stuck in local minima by introducing a "momentum" term. It helps the optimization process overcome small bumps and accelerates convergence.

**Formula**: *x*new​=*x*old −(momentum×previous\_update+learning\_rate×gradient)

**Example**: Similar to rolling a ball down a hill, gaining momentum as it goes.

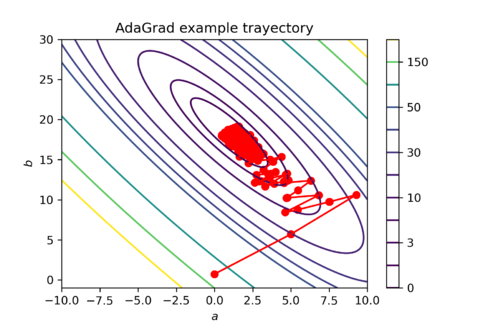
1. **Adagrad**

**Explanation**: Adagrad adapts the learning rate for each parameter based on the historical gradients. It adjusts the learning rate individually for each parameter to account for varying importance.

**Key features:**

* Adaptive: Adagrad automatically adjusts the learning rate for each parameter, which can be beneficial for problems with non-uniform data or a large number of parameters.
* Simple: Adagrad is relatively simple to implement and requires few hyperparameters.
* Effective: Adagrad has been shown to be effective for a variety of machine learning problems, including image classification, natural language processing, and reinforcement learning.

**Formula**: 

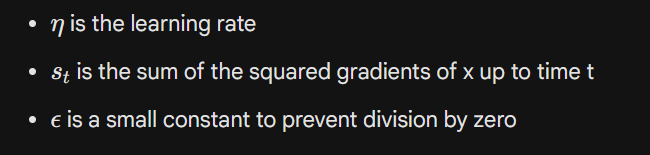
**Example**: Like adjusting driving speed based on the conditions of the road.

Consider the following quadratic function: f(x) = x^2

* The gradient of this function is 2x.

If we use Adagrad to minimize this function, the update rule for the parameter x is as follows: x\_t = x\_{t-1} - \eta \frac{2x\_{t-1}}{\sqrt{s\_t + \epsilon}}

Where:



For this example, the update rule becomes:

x\_t = x\_{t-1} - \frac{2\eta x\_{t-1}}{\sqrt{2x\_{t-1}^2 + \epsilon}}

As the gradient of x increases, the denominator of the update rule increases. This means that the learning rate decreases, which slows down the update of x.

1. **RMSprop**

**Explanation**: RMSprop is a modification of Adagrad that uses a moving average of squared gradients to adapt the learning rates. It helps address the diminishing learning rate issue.

* RMSProp is a coordinate-wise adaptive learning rate algorithm that uses exponential moving average to adjust the preconditioner.
* RMSProp can be considered as a modified version of Adagrad, in which the preconditioner is updated using the sum of squared gradients instead of the cumulative sum of squared gradients.
* RMSProp has several advantages over Adagrad, including:
* Does not suffer from learning rate decay over time.
* Can be more effective in non-convex problems.
* However, RMSProp also has some disadvantages, including:
* Can be sensitive to the choice of the gamma parameter.
* Can lead to oscillations during convergence.

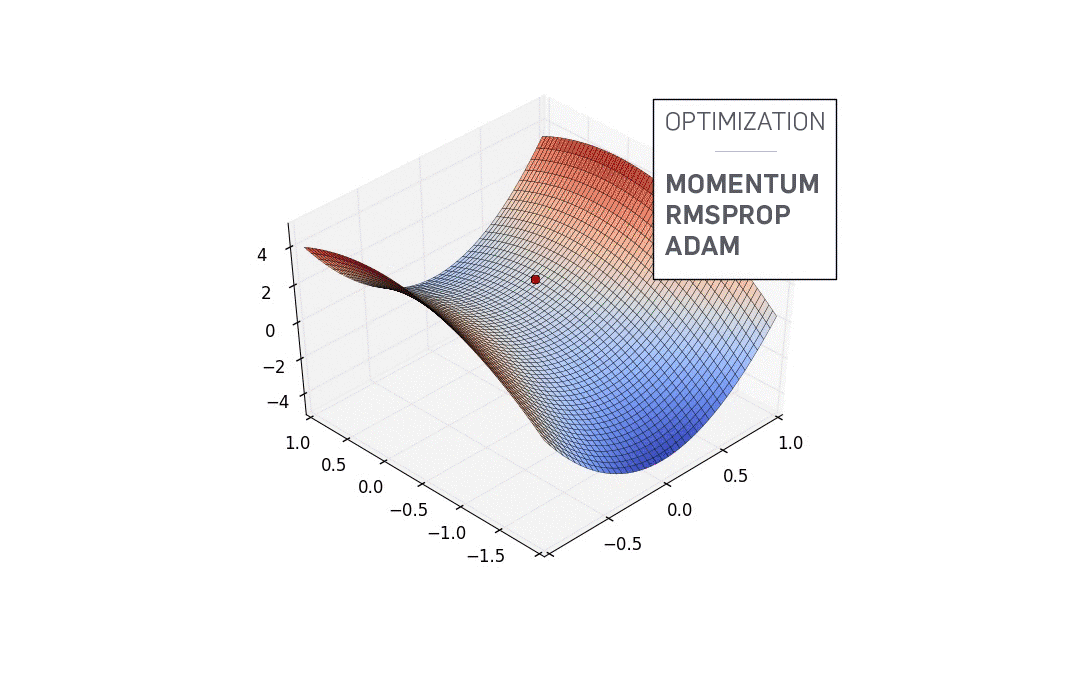
**Formula**: Similar to Adagrad but with a weighted average of squared gradients.

**Comparison with Adagrad:**

RMSProp and Adagrad are both coordinate-wise adaptive learning rate algorithms that use the square of the gradient to adjust the coordinate-wise learning rate. However, there are some important differences between the two methods:

* RMSProp uses an exponential moving average to update the preconditioner, while Adagrad accumulates the cumulative sum of squared gradients. This means that RMSProp does not suffer from learning rate decay over time, while Adagrad can decay.
* RMSProp can be sensitive to the choice of the gamma parameter. If gamma is too small, then RMSProp may not converge. If gamma is too large, then RMSProp may lead to oscillations during convergence.

**Example**: Adjusting driving speed with more consideration for recent changes in road conditions.

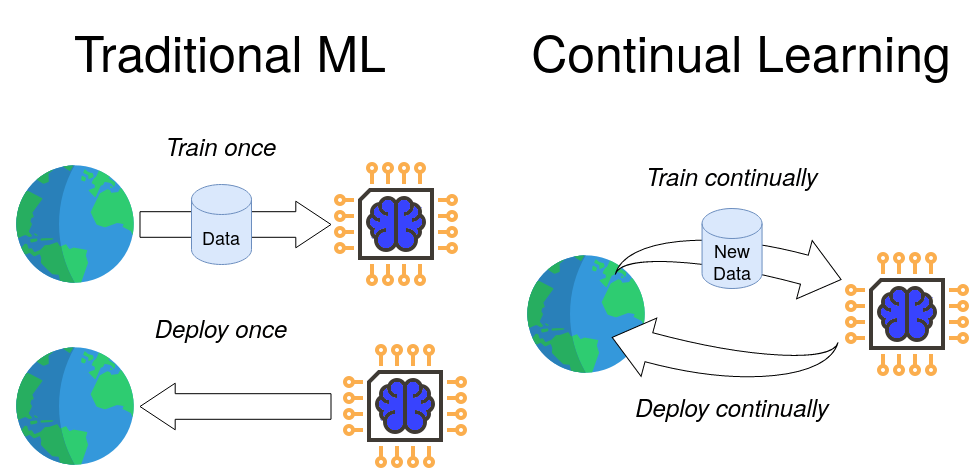


1. **Adam**
   * **Explanation**: Adam combines the ideas of Momentum and RMSprop. It has adaptive learning rates for each parameter and incorporates momentum to efficiently converge to optimal solutions.
   * **Formula**: Combines the update rules of Momentum and RMSprop.
   * **Example**: Walking with a good balance of maintaining a steady pace and adjusting to the terrain for smoother progress.
   1. **Comparison**

| **Optimization Algorithm** | **Description** | **Advantages** | **Disadvantages** | **Recommended Use Cases** | **Considerations** |
| --- | --- | --- | --- | --- | --- |
| **Gradient Descent (GD)** | Basic optimization algorithm to adjust model parameters. | - Simple and easy to understand. | - May converge slowly. | Small to medium-sized datasets with smooth loss surfaces. | Suitable for simple models or when training time is not critical. |
| **Stochastic GD (SGD)** | Variation of GD; updates parameters for each data point. | - Suitable for large datasets. | - Noisy updates can result in oscillations. | Large datasets where computing the gradient on the entire dataset is expensive. | Effective for online learning scenarios. |
| **Momentum** | Introduces momentum to overcome local minima. | - Accelerates convergence. | - May overshoot and oscillate. | Dealing with sparse data or noisy gradients. | Useful for a wide range of scenarios. |
| **Adagrad** | Adapts learning rates individually for each parameter. | - Automatic adjustment of learning rates. | - Accumulates squared gradients, leading to diminishing learning rate. | Initially designed for sparse data; scenarios with varying importance of parameters. | Be cautious with accumulating squared gradients. |
| **RMSprop** | Modification of Adagrad with a moving average of gradients. | - Addresses Adagrad's diminishing learning rate issue. | - Still accumulates historical information. | Dealing with non-stationary or noisy problems. | Suitable for scenarios where dataset characteristics change over time. |
| **Adam** | Combines ideas from Momentum and RMSprop. | - Adaptive learning rates, efficient convergence. | - May require tuning of additional hyperparameters. | Broad range of problems; widely used as a default. | Often effective, may require hyperparameter tuning. |

1. **Research Continual Learning and Test Production when building a machine learning solution to solve a certain problem.**

**2.1) Continual Learning**

**Continual Learning** (also known **as Incremental Learning, Life-long Learning**) is a concept to learn a model for a large number of tasks sequentially without forgetting knowledge obtained from the preceding tasks, where the data in the old tasks are not available anymore during training new ones.

**Key Aspects:**

* Incremental Training: Continual learning involves updating a model with new data while preserving knowledge learned from previous data. The model evolves over time.
* Memory and Forgetting: Managing the memory of the model is crucial. Strategies to prevent catastrophic forgetting, where the model loses performance on previous tasks when learning new ones, are essential.
* Task Boundaries: Detecting and adapting to task boundaries is crucial. Proper handling of transitions between different tasks or concepts is required to maintain model performance.

**Challenges:**

* Catastrophic Forgetting: The model may forget information from earlier tasks when learning new tasks.
* Bias Accumulation: Continual learning might lead to biased models over time if not carefully managed.

**Applications:**

* Online Learning: Useful for systems where new data arrives continuously (e.g., streaming data, online platforms).
* Adaptive Systems: Ideal for scenarios where the underlying data distribution changes over time.

**2.2) Test Production**

Test Production refers to the process of generating test datasets to evaluate the performance of a machine learning model. It involves creating representative datasets that help assess the model's generalization and robustness.

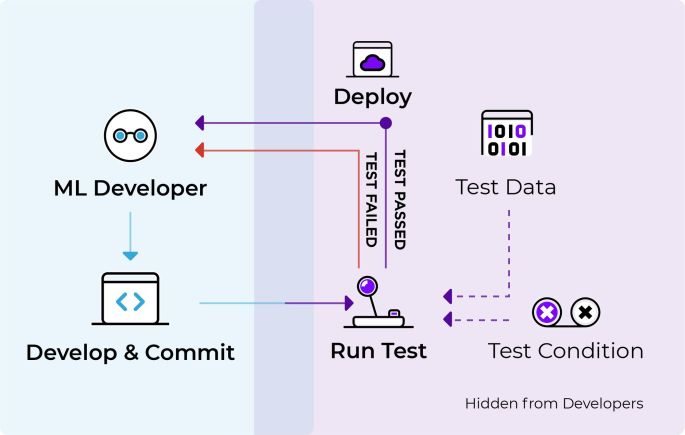
**Key Aspects:**

* Diversity: Test datasets should cover a diverse range of scenarios, ensuring that the model has generalized well across various conditions.
* Edge Cases: Including edge cases or scenarios that deviate from the norm helps identify potential vulnerabilities in the model.
* Dynamic Testing: Test production may involve dynamic adaptation, especially in continual learning scenarios, where the model evolves over time.

**Challenges:**

* Representativeness: Ensuring that the test datasets are representative of the real-world scenarios the model will encounter.
* Ethical Considerations: Handling sensitive or biased content in test datasets requires careful consideration.

**Applications:**

* Model Evaluation: Used to assess the performance of a model on various tasks, ensuring it meets the desired standards.
* Benchmarking: Comparing models based on standardized test datasets helps in benchmarking their performance.