Vietnam General Confederation of Labor

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

FINAL REPORT

MACHINE LEARNING

Instructor: **Mr. LE ANH CUONG**

Student: **Tran Dinh Quang Vinh** - *Mssv*: **521H0222**

Class **: 21H50301**

Year **: 25**

HO CHI MINH CITY, 2023

Vietnam General Confederation of Labor

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

FINAL REPORT

MACHINE LEARNING

Instructor: **Mr. LE ANH CUONG**

Student: **Tran Dinh Quang Vinh -** *Mssv*: **521H0333**

Class **: 21H50301**

Year **: 25**

HO CHI MINH CITY, 2023

ACKNOWLEDGEMENT

The first sincere thanks we want to give to Mr. Le Anh Cuong, who enthusiastically taught and worked tirelessly to give us enough tools and skills to complete this report. He played an important role in improving our mathematical logic and knowledge. The second thanks we would like to give to the teachers of the Department of Information Technology of Ton Duc Thang University for giving us the opportunity to do this report. Nevertheless, our report will have some errors, we are open to receive any feedback from teachers so that we can improve our report.

Finally, we wish you good health and success in your noble career.

Ho Chi Minh city, 12/21, 2023

Vinh

Tran Dinh Quang Vinh

THIS PROJECT WAS COMPLETED AT

TON DUC THANG UNIVERSITY

We fully declare that this is our own project and is guided by Mr. Le Anh Cuong; The research contents and results in this topic are honest and have not been published in any form before.

Should any frauds be found, I will take full responsibility for the content of my report. Ton Duc Thang University is not related to copyright and copyright violations caused by me during the implementation process (if any).

Ho Chi Minh city, 12/21, 2023

Vinh

Tran Dinh Quang Vinh

CONFIRMATION AND ASSESSMENT SECTION

Instructor confirmation section

Ho Chi Minh, 12/21, 2023

Evaluation section for grading instructor

Ho Chi Minh, 12/21, 2023

(Sign and write full name)

SUMMARY

Comprehending and Evaluating Machine Learning Optimization Techniques: It is essential to use an appropriate optimization strategy (Optimizer) while training machine learning models. Numerous optimization techniques exist, such as RMSprop, Adam, Mini-batch Gradient Descent, Gradient Descent, and Stochastic Gradient Descent (SGD). Every approach has pros and cons of its own, and it might work well with particular kinds of models. A thorough understanding of these techniques' operation and effects on the model's learning process is necessary to compare them.

Continual Learning and Test Production in Machine Learning Solutions: In the vital subject of machine learning, models are trained to learn from new data indefinitely without losing track of what they have already learned from old data. This is known as continuous learning. When developing machine learning models for real-world issues where data changes over time, this is especially helpful.

Contrarily, test production entails putting the model through its paces and making sure it performs as planned when it is put into use in a real-world setting. Performance testing, accuracy confirmation, and determining whether the model can satisfy particular problem criteria are all included in this.

TABLE OF CONTENT

CHAPTER 1: Learn and compare methods in OPTIMIZER

1.1 What is OPTIMIZER?

An optimizer is an essential part of the model training procedure in machine learning. In order to enable the model to learn from the training data as efficiently as possible, it is used to optimize the loss function by modifying the model's parameters.

Finding the loss function's least or almost-minimum value is the aim of utilizing an optimizer. When training machine learning models, this is especially crucial since it makes the model learn from the data more efficiently, which produces predictions that are more correct.

During training, the optimizer is essential in minimizing the difference between the expected output and the actual target values by iteratively adjusting the model parameters. Through appropriate parameter adjustments, the optimizer directs the model towards a configuration that minimizes loss.

Numerous optimization methods are available, each with unique advantages and disadvantages include:

- Adam
- RMSprop
- Mini-batch Gradient Descent
- Gradient Descent
- Stochastic Gradient Descent, and others.

1.1.1 Adam (Adaptive Moment Estimation)

Adam (Adaptive Moment Estimation) is an optimization technique that is used to train artificial neural networks in deep learning and machine learning. It integrates the concepts of RMSprop and momentum optimization. Adam uses the gradients' first-order moment (mean) and second-order moment (uncentered variance, or standard deviation) to calculate adaptive learning rates for each parameter.

1.1.1.1 Benefits

Adaptive Learning Rates: Adam modifies the learning rates according to each parameter separately. For every parameter, it computes the second-order scaling of the gradients as well as the first-order momentum, enabling the algorithm to dynamically modify the learning rates according to the gradient history.

Efficiency: Adam frequently converges more quickly than conventional optimization techniques and is computationally efficient. Deep neural networks and large-scale machine learning tasks are good fits for it.

Robustness to Sparse Gradients: It is well known that Adam can withstand noisy or sparse gradients. The adaptive learning rate facilitates the handling of gradient variations in several dimensions.

Bias Correction: In order to account for the moment estimates' tendency toward zero, particularly in the early training cycles, Adam uses bias correction. This enhances the estimations' accuracy.

Applicability to a Wide Range of Problems: Adam is a flexible approach that can be used for both convex and non-convex optimization, among other kinds of problems.

1.1.1.2 Drawbacks

Sensitivity to Learning Rate Hyperparameter: Adam's performance may vary depending on the hyperparameter for learning rate that is selected. An excessively high learning rate might cause oscillations or divergence, whereas an excessively low learning rate can impede convergence.

Memory Intensive: Compared to certain other optimization techniques, Adam requires more memory since it keeps track of extra state variables (such as momentum and squared gradients) for every parameter.

Lack of Theoretical Understanding: Theoretical explanations for why Adam performs successfully in practice are still being researched, despite its empirical effectiveness. The inability to establish a solid theoretical basis can make hyperparameter tuning difficult.

Not Always the Best Performer: Even though Adam operates effectively most of the time, it might not be the ideal optimizer for all situations. It's a good idea to experiment with multiple optimizers because different optimization algorithms may be more beneficial for certain challenges.

1.1.2 RMSprop

Root mean squared propagation, or RMSProp, is an optimization machine learning technique that is based on the ideas of gradient descent and RProp and is used to train Artificial Neural Networks (ANNs) at varying adaptive learning rates.

Neural network training in machine learning and deep learning uses the optimization method RMSprop. Its purpose is to mitigate certain drawbacks of the conventional Stochastic Gradient Descent (SGD) optimization technique, specifically those pertaining to the selection of learning rates.

1.1.2.1 Benefits

Adaptive Learning Rate: RMSprop uses the historical gradients to modify the learning rate for each parameter separately. This aids in resolving problems with fixed learning rates that can be either too large, resulting in divergence, or too small, producing sluggish convergence.

Effective in Non-Convex Optimization: RMSprop is frequently useful for highly non-linear and non-convex cost function optimization. It can traverse across areas with different curvatures and handle surfaces with uneven terrain.

Memory Efficiency: Compared to techniques that necessitate keeping all previous gradients, RMSprop is memory-efficient since it maintains an exponentially weighted moving average of squared gradients for each parameter.

Dampening Effect: To keep the learning rates from getting too low, RMSprop adds a dampening component, which is often a tiny constant. This aids in maintaining training process stability.

1.1.2.2 Drawbacks

Noisy Updates: RMSprop incorporates noise into the update process, just like other adaptive learning rate algorithms. The optimization process may oscillate as a result of this noise, leading to less than ideal convergence.

Hyperparameter Sensitivity: The learning rate and decay rate are two hyperparameters that have an impact on how well RMSprop performs. It could take some trial and error to determine the ideal collection of hyperparameters.

Not Always the Best Performer: RMSprop sometimes performs better than other optimization techniques, despite its widespread effectiveness.

The particulars of the problem at hand may influence the optimization method selection, and algorithms like Adam, which take momentum into account, have become more and more popular for particular applications.

Lack of Theoretical Guarantees: RMSprop does not have strong theoretical guarantees about its convergence qualities, in contrast to several other optimization techniques. Although it frequently functions effectively in practice, it can be difficult to comprehend how it behaves in every situation.

1.1.3 Mini-batch Gradient Descent

A variant of the classic gradient descent optimization process that is frequently employed in machine learning is called mini-batch gradient descent. In gradient descent, a cost or loss function is minimized by iteratively updating the model parameters. The dataset is split up into smaller batches for Mini-batch Gradient Descent, and the model parameters are updated for each batch. Mini-batch gradient descent computes the gradient on a tiny random subset of the data, known as a mini-batch, as opposed to computing the gradient of the full dataset (batch gradient descent) or a single data point (stochastic gradient descent) in each iteration.

Additionally, it provides a middle ground between two other well-liked optimization algorithms: Stochastic Gradient Descent (SGD), which processes only some data in each iteration, and Gradient Descent (GD), which processes the complete dataset in each.

1.1.3.1 Benefits

Faster Convergence: When using mini-batch updates instead of pure batch gradient descent, convergence happens more quickly. The efficiency of stochastic updates and the stability of batch updates can be balanced thanks to the mini-batch size.

Smoothing Noise: Mini-batches offer a compromise between batch updates' stability and SGD's stochastic nature. This contributes to more steady convergence by reducing the noise in the gradients.

Memory Efficiency: Because the complete dataset doesn't need to be loaded into memory, mini-batches demand less memory than batch gradient descent. This increases the possibility of managing big datasets that might not fit in memory.

Parallelization: Mini-batches are appropriate for parallel computing systems because they can be processed in parallel. This can greatly accelerate the training process, particularly on multi-core or multi-GPU technology.

1.1.3.2 Drawbacks

Learning Rate Tuning: Mini-batch GD, like SGD, necessitates precise adjustment of the learning rate. Achieving the ideal balance between the ideal learning rate and the mini-batch size is a challenging task.

Noisy Updates: Because each mini-batch contains a restricted number of instances, the updates are still a little loud. The convergence process may exhibit oscillations due to noise in updates.

Not Always Parallelizable: Although mini-batches can be handled in parallel, pure batch gradient descent is easier to parallelize. The degree of parallelization could be constrained by certain dependencies amongst mini-batches.

Sensitivity to Mini-batch Size: One hyperparameter that can impact the algorithm's performance is the selection of the mini-batch size. The features of the dataset and the issue at hand may determine the ideal size.

1.1.4 Gradient descent

One of the most popular optimization techniques for training machine learning models is gradient descent, which reduces the discrepancies between actual and predicted outcomes. Moreover, neural networks are trained by gradient descent as well.

The process of minimizing the cost function that is parameterized by the model's parameters is known as optimization in machine learning. Gradient descent's primary goal is to minimize the convex function through repeated parameter changes. These machine learning models have the potential to be extremely useful tools for a variety of computer science and artificial intelligence applications once they are tuned.

1.1.4.1 Benefits

Versatility:

Application: A variety of machine learning models, such as neural networks, logistic regression, linear regression, and more, can be used with gradient descent.

Optimization: This is an adaptable algorithm for handling differentiable functions in optimization.

Efficiency:

Computationally Efficient: Since the complete dataset need not be imported into memory, it is computationally efficient, particularly when working with huge datasets.

Parallelization: Mini-batch gradient descent is one of the variants that may be parallelized, which makes it appropriate for distributed computing settings.

Convergence: Gradient Descent usually converges to a minimum or near-minimum of the cost curve with appropriate hyperparameter adjustment (learning rate, mini-batch size, etc.).

1.1.4.2 Drawbacks

Sensitivity to Hyperparameters: Selecting the right learning rate is essential. A learning rate that is too high could cause divergence, while one that is too low could cause sluggish convergence.

Noisy Updates: SGD generates noisy updates that could induce oscillations during convergence because it relies on individual example updates.

Local Minima: Gradient Descent may converge to a local minimum as opposed to the global minimum, depending on the form of the cost function. This can be lessened with the use of strategies like learning rate schedules and momentum.

1.1.5 Stochastic Gradient Descent

An optimization technique that is frequently employed in deep learning and machine learning is called stochastic gradient descent (SGD). This variation of the gradient descent optimization approach involves using the gradient of the cost function with respect to a single training example to update the model parameters.

By updating weights for each sample or a small subset of samples (mini-batch), SGD speeds up learning and uses less memory. But the learning process may become unstable as a result of this method.

1.1.5.1 Benefits

Faster Convergence: Compared to batch gradient descent, more frequent updates to the model parameters result in faster convergence, especially for big datasets.

Computational Efficiency: SGD is computationally more efficient when processing a single training sample at a time because it eliminates the need to store and manipulate the full dataset in memory.

Online Learning: SGD is appropriate for dynamic and streaming datasets because it facilitates online learning, where the model may be updated in real-time as new data becomes available.

Memory Efficiency: Uses less memory than batch gradient descent since it only needs to load one training example at a time into memory.

Escape from Local Minima: The processing of individual samples provides noisy updates that can aid in the algorithm's escape from local minima.

1.1.5.2 Drawbacks

Noisy Updates: During the convergence phase, oscillations and instability may arise due to noisy updates to the model parameters.

Learning Rate Tuning: Care must be taken to adjust the learning rate. While a learning rate that is too little can impede convergence, one that is too big can lead to divergence.

Not Easily Parallelizable: SGD's performance on parallel computing systems could be constrained since it is more difficult to parallelize than batch gradient descent or mini-batch gradient descent.

Variance in Convergence: It might be challenging to forecast the precise course of the optimization process because of the variability in the convergence route caused by the stochastic character of the updates.

Sparse Gradients: Gradients in high-dimensional spaces can be sparse, and updating parameters based on a single sample could not give a representative direction.

Possibility of Missing Global Minimum: If the updates are too noisy and the learning rate is not calibrated properly, the algorithm may not be able to attain the genuine global minimum.

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

2.1 What is Continual Learning

Continual Learning is the idea of updating your model as new data becomes available; this makes your model keep up with the current data distributions.

You cannot simply release your revised model into production. To make sure it is safer and superior to the existing production model, it must be tested. The "Testing models in Production" section follows at this point.

2.1.1 Solution to solve a problem.

Step 1: Manual retraining for stateless people

Retraining a model only occurs when two requirements are satisfied: (1) the model's performance has declined to the point that it is no longer beneficial, and (2) your team has the time to update the model.

Stage 2: Automated stateless retraining on a set schedule

This stage often occurs when a domain's primary models have been produced, at which point maintaining and enhancing the current models should take precedence over developing new ones. Remaining in stage 1 is becoming too much of a pain to bear.

During this point, retraining frequency is usually determined by "gut feeling".

The transition point from stage 1 to stage 2 is typically a script that is written that occasionally conducts the stateless retraining. The degree of difficulty in writing this script varies with the number of dependencies that must be synchronized in order to retrain a model.

The high level steps of this script are:

1. Pull data.
2. Downsample or upsample 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: Automated stateful training with a set schedule

You must modify your script and find a means to monitor your data and model lineage in order to accomplish this. An example of a basic model lineage versioning is:

For the same problem, there exist two distinct model architectures: V1 and V2.

Model architecture V1.2 vs. V2.3 indicates that V2 is on its third iteration of a full stateless retraining, whereas V1 is in its second.

V2.3.43 vs. V1.2.12 indicates that 12 stateful trainings were completed on V2.3, whereas 43 were completed on V1.2.

Stage 4: Continual learning

This step replaces the fixed schedule component of earlier stages with a trigger mechanism for retraining. The catalysts may include:

- Time-based
- Performance-based: for example, the last x% of performance has been reached.
- Volume-based: A 5% increase in the total quantity of labeled data
- Drift-based: such as in the event that a "major" shift in the distribution of data is found.

To keep track of the whole picture of how the models are changing, you will probably need to utilize this in conjunction with other versioning strategies like data versioning.

The author claims that since she is not aware of any model store with this kind of model lineage capability, businesses create their own internally.

You will always have many models in production at any given time.

2.2 What is Test Production

In order to thoroughly test your models prior to their widespread release, you must conduct both offline pre-deployment evaluations and production testing. The use of offline evaluations just is insufficient.

Every team should ideally have a well-defined pipeline for evaluating models, including which tests to run, who should execute them, and what criteria must be met in order to advance a model to the next level. These evaluation pipelines should ideally be automated and initiated in response to each new model update. The stage promotions must to be examined in a manner akin to software engineering's evaluation of CI/CD.

2.2.1 Solution to solve a problem.

Data Collection: Gather test results from manufacturing goods. User reviews, technical specs, and any other relevant information may be included in this data.

Data Preprocessing: Data should be preprocessed to manage missing values, eliminate noise, and normalize. Data classification and feature transformation may be examples of this.

Split Data Set: Separate the training and test sets from the data set. The test set aids in assessing the model's performance on fresh data, whilst the training set is utilized to train the model.

Construction Model: To forecast the performance or quality of a product, use a model or collection of models. This could be a statistical model or a machine learning model, based on the particular objective and the type of data.

Model Training: To train the model, use the training data set. In order to maximize performance on the training set, this requires adjusting the model's parameters.

Model Rating: To assess the performance of the model, use the test set. Assess metrics based on the particular objectives of the situation, including sensitivity, prediction accuracy, accuracy, and other metrics.

Optimization and Tuning: Optimize and fine-tune the model as needed to enhance performance. This could entail adjusting parameters, putting other model types to the test, or using other optimization strategies.

Model Deployment: Install the model in the real-world setting. This may entail incorporating the model into the workflow and updating it whenever fresh data becomes accessible.

Continuous Monitoring and Evaluation: Keep an eye on the model's performance in real-world settings and conduct ongoing assessments to make sure it can adapt to changing conditions and data irregularities.

REFERENCES

[1]https://github.com/serodriguez68/designing-ml-systems-summary/blob/main/09-continual-learning-and-test-in-production.md

[2]https://viblo.asia/p/optimizer-hieu-sau-ve-cac-thuat-toan-toi-uu-gdsgdadam-Qbq5QQ9E5D8

[3]https://machinelearningcoban.com/2017/01/16/gradientdescent2/