TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

DỰ ÁN CUỐI KÌ MÔN NHẬP MÔN HỌC MÁY

DỰ ÁN CUỐI KÌ

Người hướng dẫn: **PGS.TS. LÊ ANH CƯỜNG**
Người thực hiện: **TRƯƠNG CÔNG THÀNH – 521H0302**
Lớp: **21H50302**
Khoá: **25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

DỰ ÁN CUỐI KÌ MÔN NHẬP MÔN HỌC MÁY

DỰ ÁN CUỐI KÌ

Người hướng dẫn: **PGS.TS. LÊ ANH CƯỜNG**
Người thực hiện: **TRƯƠNG CÔNG THÀNH – 521H0302**
Lớp: **21H50302**
Khoá: **25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Lời đầu tiên, em xin trân trọng cảm ơn giảng viên PGS.TS. Lê Anh Cường - người đã trực tiếp chỉ bảo, hướng dẫn em trong quá trình hoàn thành dự án này.

Em cũng xin cảm ơn đến quý thầy, cô giáo trong trương Đại học Tôn Đức Thắng, đặc biệt là các thầy, cô giáo khoa Công nghệ thông tin - người đã truyền lửa và giảng dạy kiến thức cho em suốt thời gian qua.

Mặc dù đã có những sự đầu tư nhất định trong quá trình làm bài song cũng khó có thể tránh khỏi những sai sót, em kính mong nhận được nhận được ý kiến đóng góp của quý thầy cô để dự án được hoàn thiện hơn.

Em xin chân thành cảm ơn.

ĐỒ ÁN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi và được sự hướng dẫn của PGS.TS. Lê Anh Cường. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 18 tháng 12 năm 2023

Tác giả

Trương Công Thành

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

TÓM TẮT

Đây là dự án cuối kì môn Nhập môn học máy học kì I năm 2023.

MỤC LỤC

LỜI CẢM ƠN i

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN iii

TÓM TẮT iv

MỤC LỤC 1

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ 4

QUESTION 1 – OPTIMIZERS IN TRAINING MACHINE LEARNING MODELS 5

1.1 What is optimizer? 5

1.2 Types of optimizers 5

1.2.1 Gradient Descent 5

1.2.1.1 What is gradient descent? 5

1.2.1.2 How does gradient descent work? 5

1.2.1.3 Types of gradient descents 7

1.2.1.4 Challenges with gradient descent 8

1.2.2 Momentum 10

1.2.2.1 What is momentum? 10

1.2.2.2 Theory, methodology, and/or algorithmic discussions 10

1.2.2.3 Application 13

1.2.2.4 Conclusion 13

1.2.3 Adagrad 14

1.2.3.1 What is Adagrad? 14

1.2.3.2 How does Adagrad work? 14

1.2.3.3. Algorithm 14

1.2.3.4 Application 15

1.2.3.5 Conclusion 16

1.2.4 RMSProp 16

1.2.4.1 What is RMSProp? 16

1.2.4.2 How does RMSProp work? 16
1.2.5 Adam 17
1.2.5.1 What is Adam? 17
1.2.5.2 How does Adam work? 17
1.2.6 Adadelta 18
1.2.6.1 What is Adadelta? 18
1.2.6.2 How does Adadelta work? 18
1.3 Comparison 19
QUESTION 2 – CONTINUAL LEARNING AND TEST PRODUCT 21
2.1 Continual Learning 21
2.2 Test Production 21
2.3 The importance of Continual Learning and Test Production 22
2.4 Some examples of the use of Continual Learning and Test Production 22
2.5 Conclusion 22
2.6 Specific problem 23
2.6.1 Application of Continual Learning 23
2.6.2 Application of Test Production 23

DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

CÁC KÝ HIỆU

CÁC CHỮ VIẾT TẮT

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

DANH MỤC HÌNH

DANH MỤC BẢNG

QUESTION 1 – OPTIMIZERS IN TRAINING MACHINE LEARNING MODELS

1.1 What is optimizer?

Optimizers are algorithms that iteratively adjust the parameters of a machine learning model to minimize or maximize a certain objective function. The choice of optimizer can significantly impact the convergence speed, stability, and final performance of the trained model.

1.2 Types of optimizers

There are many optimized methods (optimizer) used in the training of machine modeling. Here are some popular optimization methods.

1.2.1 Gradient Descent

1.2.1.1 What is gradient descent?

Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates. Until the function is close to or equal to zero, the model will continue to adjust its parameters to yield the smallest possible error. Once machine learning models are optimized for accuracy, they can be powerful tools for artificial intelligence (AI) and computer science applications.

1.2.1.2 How does gradient descent work?

Before we dive into gradient descent, it may help to review some concepts from linear regression. You may recall the following formula for the slope of a line, which is y = mx + b, where m represents the slope and b is the intercept on the y-axis.

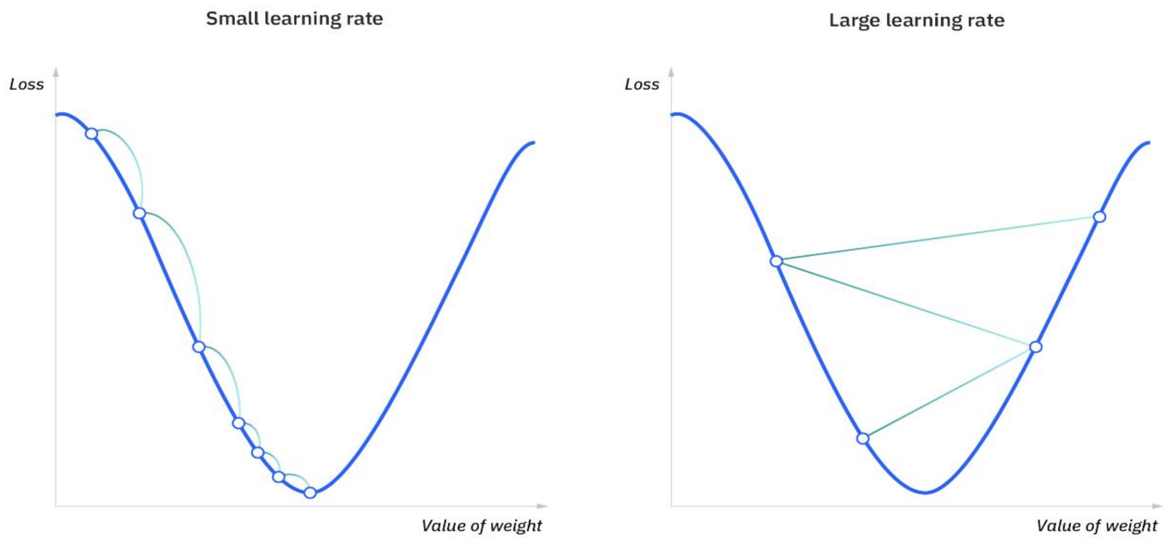
You may also recall plotting a scatterplot in statistics and finding the line of best fit, which required calculating the error between the actual output and the predicted output (y-hat) using the mean squared error formula. The gradient descent algorithm behaves similarly, but it is based on a convex function.

The starting point is just an arbitrary point for us to evaluate the performance. From that starting point, we will find the derivative (or slope), and from there, we can use a tangent line to observe the steepness of the slope. The slope will inform the updates to the parameters—i.e. the weights and bias. The slope at the starting point will be steeper, but as new parameters are generated, the steepness should gradually reduce until it reaches the lowest point on the curve, known as the point of convergence.

Similar to finding the line of best fit in linear regression, the goal of gradient descent is to minimize the cost function, or the error between predicted and actual y. In order to do this, it requires two data points—a direction and a learning rate. These factors determine the partial derivative calculations of future iterations, allowing it to gradually arrive at the local or global minimum (i.e. point of convergence).

- **Learning rate** (also referred to as step size or the alpha) is the size of the steps that are taken to reach the minimum. This is typically a small value, and it is evaluated and updated based on the behavior of the cost function. High learning rates result in larger steps but risks overshooting the minimum. Conversely, a low learning rate has small step sizes. While it has the advantage of more precision, the number of iterations compromises overall efficiency as this takes more time and computations to reach the minimum.

- **The cost (or loss) function** measures the difference, or error, between actual y and predicted y at its current position. This improves the machine learning model's efficacy by providing feedback to the model so that it can adjust the parameters to minimize the error and find the local or global minimum. It continuously iterates, moving along the direction of steepest descent (or the negative gradient) until the cost function is close to or at zero. At this point, the model will stop learning. Additionally, while the terms, cost function and loss function, are considered synonymous, there is a slight difference between them. It’s worth noting that a loss function refers to the error of one training example, while a cost function calculates the average error across an entire training set.



1.2.1.3 Types of gradient descents

There are three types of gradient descent learning algorithms: batch gradient descent, stochastic gradient descent and mini-batch gradient descent.

- Batch gradient descent:

Batch gradient descent sums the error for each point in a training set, updating the model only after all training examples have been evaluated. This process referred to as a training epoch.

While this batching provides computation efficiency, it can still have a long processing time for large training datasets as it still needs to store all of the data into memory. Batch gradient descent also usually produces a stable error gradient and convergence, but sometimes that convergence point isn’t the most ideal, finding the local minimum versus the global one.

- Stochastic gradient descent:

Stochastic gradient descent (SGD) runs a training epoch for each example within the dataset and it updates each training example's parameters one at a time. Since you only need to hold one training example, they are easier to store in memory. While these frequent updates can offer more detail and speed, it can result in losses in computational efficiency when compared to batch gradient descent. Its frequent updates can result in noisy gradients, but this can also be helpful in escaping the local minimum and finding the global one.

- *Mini-batch gradient descent:*

Mini-batch gradient descent combines concepts from both batch gradient descent and stochastic gradient descent. It splits the training dataset into small batch sizes and performs updates on each of those batches. This approach strikes a balance between the computational efficiency of batch gradient descent and the speed of stochastic gradient descent.

1.2.1.4 Challenges with gradient descent

While gradient descent is the most common approach for optimization problems, it does come with its own set of challenges. Some of them include:

- *Local minima and saddle points:*

For convex problems, gradient descent can find the global minimum with ease, but as nonconvex problems emerge, gradient descent can struggle to find the global minimum, where the model achieves the best results.

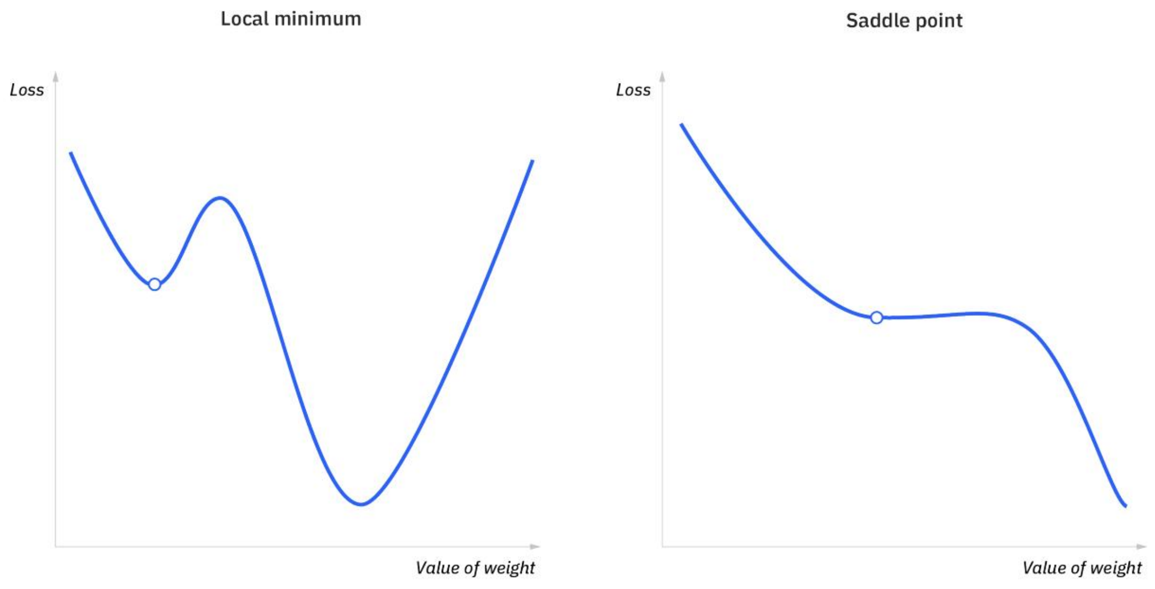
Recall that when the slope of the cost function is at or close to zero, the model stops learning. A few scenarios beyond the global minimum can also yield this slope, which are local minima and saddle points. Local minima mimic the shape of a global minimum, where the slope of the cost function increases on either side of the current point. However, with saddle points, the negative gradient only exists on one side of the point, reaching a local maximum on one side and a local minimum on the other. Its name inspired by that of a horse’s saddle.

Noisy gradients can help the gradient escape local minimums and saddle points.

- Vanishing and Exploding Gradients:

In deeper neural networks, particular recurrent neural networks, we can also encounter two other problems when the model is trained with gradient descent and backpropagation.

- **Vanishing gradients:** This occurs when the gradient is too small. As we move backwards during backpropagation, the gradient continues to become smaller, causing the earlier layers in the network to learn more slowly than later layers. When this happens, the weight parameters update until they become insignificant—i.e. 0—resulting in an algorithm that is no longer learning.
- **Exploding gradients:** This happens when the gradient is too large, creating an unstable model. In this case, the model weights will grow too large, and they will eventually be represented as NaN. One solution to this issue is to leverage a dimensionality reduction technique, which can help to minimize complexity within the model.



1.2.2 Momentum

1.2.2.1 What is momentum?

Momentum is an extension to the gradient descent optimization algorithm that builds inertia in a search direction to overcome local minima and oscillation of noisy gradients. It is based on the same concept of momentum in physics. A classical example of the concept is a ball rolling down a hill that gathers enough momentum to overcome a plateau region and make it to a global minima instead of getting stuck at a local minima. Momentum adds history to the parameter updates of descent problems which significantly accelerates the optimization process. The amount of history included in the update equation is determined via a hyperparameter. This hyperparameter is a value ranging from 0 to 1, where a momentum value of 0 is equivalent to gradient descent without momentum. A higher momentum value means more gradients from the past (history) are considered.

1.2.2.2 Theory, methodology, and/or algorithmic discussions

- Algorithm:

The main idea behind momentum is to compute an exponentially weighted average of the gradients and use that to update the weights. By taking past gradients into account, the steps to gradient descent become smoothed out, which can reduce the amount of oscillations seen in iterations (where the iterative solutions are going between above and below the true curve).

In (stochastic) gradient descent without momentum, the update rule at each iteration is given by:

$\theta_i = \theta_{i-1} - \Upsilon * g_i$

Where:

- θ denotes the parameters to the cost function
- g_i is the gradient indicating which direction to decrease the cost function by

- Υ is the hyperparameter representing the learning rate

In (stochastic) gradient descent with momentum, the update rule at each iteration is given by:

$b_i = \mu * b_{i-1} + g_i$

$\theta_i = \theta_{i-1} - \Upsilon * g_i$

Where:

- θ denotes the parameters to the cost function
- g_i is the gradient indicating which direction to decrease the cost function by
- Υ is the hyperparameter representing the learning rate
- b_i is the modified step direction term (as opposed to just using g_i) that incorporates momentum
- μ is a new hyperparameter that denotes the momentum constant

Momentum can be applied to other gradient descent variations such as batch gradient descent and mini-batch gradient descent. Regardless of the gradient descent variation, the momentum hyperparameter μ must be tuned in addition to the learning rate hyperparameter Υ. μ can be any value between 0 and 1. Generally, the larger the momentum value, the more past gradients must be taken into account, resulting in a smoother update. However, choosing a μ value that is too large will result in over smoothing. It is also possible for a large μ value to overshoot the target. As the μ value gets closer to 0, momentum behaves similarly to steepest descent. A value of 0 is essentially gradient descent without momentum and a momentum value of 0.9 tends to be a good default choice. Finding the optimal μ is a matter of trial and error to determine the best choice for a particular model such that the cost function is minimized.

- Problems with Gradient Descent:

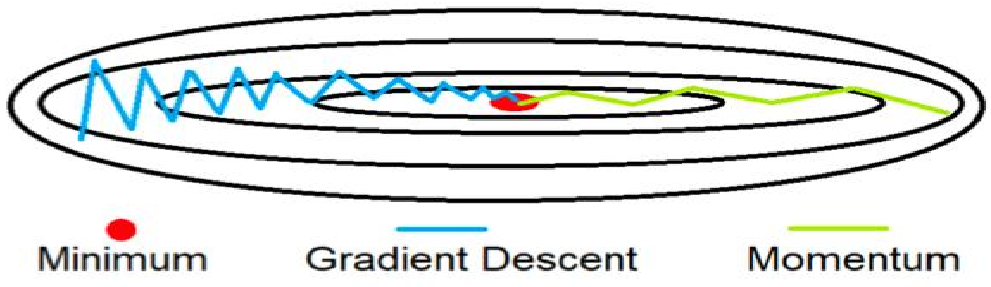
With gradient descent, a weight update at time t is given by the learning rate and gradient at that exact moment. This means that the previous steps are not considered when searching for the next iteration's solution.

This results in two issues:

- Unlike convex functions, a non-convex cost function can have many local minima's meaning the first local minima found is not guaranteed to be the global minima. At the local minima, the gradient of the cost function will be very small resulting in no weight updates. Because of this, gradient descent will get stuck and fail to find the global optimal solution.
- Gradient descent can be noisy with many oscillations which results in a larger number of iterations needed to reach convergence.

Momentum is able to solve both of these issues buy using an exponentially weighted average of the gradients to update the weights at each iteration. This method also prevents gradients of previous iterations to be weighted equally. With an exponential weighted average, recent gradients are given more weight than previous gradients.

The graph to the right shows a comparison of gradient descent with and without momentum. Using momentum, the oscillations are significantly reduced and convergence to the minimum is achieved in fewer iterations.



1.2.2.3 Application

Momentum is widely used in the machine learning community for optimizing non-convex functions such as deep neural networks. Empirically, momentum methods outperform traditional stochastic gradient descent approaches. In deep learning, SGD is widely prevalent and is the underlying basis for many optimizers such as Adam, Adadelta, RMSProp, etc. which already utilize momentum to reduce computation speed. The momentum extension for optimization algorithms is available in many popular machine learning frameworks such as PyTorch, tensor flow, and scikit-learn. Generally, any problem that can be solved with stochastic gradient descent can benefit from the application of momentum. These are often unconstrained optimization problems. Some common SGD applications where momentum may be applied are ridge and logistic regression and support vector machines. Classification problems including those relating to cancer diagnosis and image determination can also have reduced run times when momentum is implemented. In the case of medical diagnoses, this increased computation speed can directly benefit patients through faster diagnosis times and higher accuracy of diagnosis within the neural network.

Momentum is also very useful in objective functions that have a large amount of curvature. These are areas where the gradients changes a lot. Since momentum leverages an exponential weighted average, smaller steps will be taken in these regions that have higher curvature resulting in dampened oscillations.

Overall, momentum may have farther reaching applicability to accelerating second order methods and deeper connections to other methods, such as the ellipsoid algorithm.

1.2.2.4 Conclusion

Momentum improves on gradient descent by reducing oscillatory effects and acting as an accelerator for optimization problem solving. Additionally, it finds the global (and not just local) optimum. Because of these advantages, momentum is commonly used in machine learning and has broad applications to all optimizers through SGD. Though the hyperparameters for momentum must be chosen with care and requires some trial and error, it ultimately addresses common issues seen in gradient descent problems. As deep learning continues to advance, momentum application will allow models and problems to be trained and solved faster than methods without.

1.2.3 Adagrad

1.2.3.1 What is Adagrad?

Adagrad is an optimizer with parameter-specific learning rates, which are adapted relative to how frequently a parameter gets updated during training. The more updates a parameter receives, the smaller the updates.

1.2.3.2 How does Adagrad work?

The objective of AdaGrad is to minimize the expected value of a stochastic objective function, with respect to a set of parameters, given a sequence of realizations of the function. As with other sub-gradient-based methods, it achieves so by updating the parameters in the opposite direction of the sub-gradients. While standard sub-gradient methods use update rules with step-sizes that ignore the information from the past observations, AdaGrad adapts the learning rate for each parameter individually using the sequence of gradient estimates.

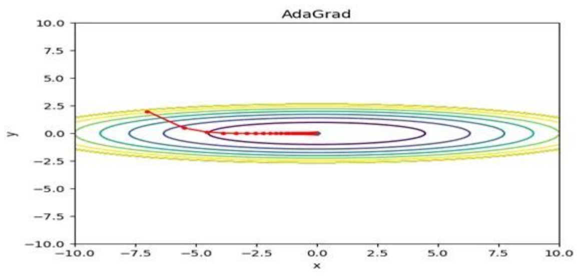
1.2.3.3. Algorithm

The general version of the AdaGrad algorithm is presented in the pseudocode below. The update step within the for loop can be modified with the version that uses the diagonal of G_t.

Algorithm 1: AdaGrad general algorithm

```
$\eta$: Stepsize ;
$f(x)$: Stochastic objective function ;
$x_1$: Initial parameter vector;
for $t = 1$ to $T$ do
    Evaluate $f_t(x_t)$ ;
    Get and save $g_t$ ;
    $G_t \leftarrow \sum_{\tau=1}^{t} g_\tau g_\tau^\top$ ;
    $x_{t+1} \leftarrow x_t - \eta G_t^{-1/2} g_t$ ;
end
return $x_t$
```

Adagrad can be visualized as:



1.2.3.4 Application

The AdaGrad family of algorithms is typically used in machine learning applications. Mainly, it is a good choice for deep learning models with sparse gradients, like recurrent neural networks and transformer models for natural language processing. However, one can apply it to any optimization problem with a differentiable cost function.

Given its popularity and proven performance, different versions of AdaGrad are implemented in the leading deep learning frameworks like TensorFlow and PyTorch. Nevertheless, in practice, AdaGrad tends to be substituted by using the Adam algorithm; since, for a given choice of hyperparameters, Adam is equivalent to AdaGrad.

1.2.3.5 Conclusion

AdaGrad is a family of algorithms for stochastic optimization that uses a Hessian approximation of the cost function for the update rule. It uses that information to adapt different learning rates for the parameters associated with each feature. Their main two advantages are that: it assigns more significant learning rates to parameters related to low-frequency features, and the updates in directions of high curvature tend to be lower than those in low-curvature directions. Therefore, it works especially well for problems where the input features are sparse. AdaGrad is reasonably popular in the machine learning community, and it is implemented in the primary deep learning frameworks. In practice, the Adam algorithm is usually preferred over AdaGrad since, for a given choice of hyperparameters, Adam is equivalent to AdaGrad.

1.2.4 RMSProp

1.2.4.1 What is RMSProp?

RMSProp, root mean square propagation, is an optimization algorithm/method designed for Artificial Neural Network (ANN) training. And it is an unpublished algorithm first proposed in the Coursera course. “Neural Network for Machine Learning” lecture six by Geoff Hinton. RMSProp lies in the realm of adaptive learning rate methods, which have been growing in popularity in recent years because it is the extension of Stochastic Gradient Descent (SGD) algorithm, momentum method, and the foundation of Adam algorithm. One of the applications of RMSProp is the stochastic technology for mini-batch gradient descent.

1.2.4.2 How does RMSProp work?

The gist of RMSprop is to:

- Maintain a moving (discounted) average of the square of gradients
- Divide the gradient by the root of this average

RMSprop’s update rule:

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

1.2.5 Adam

1.2.5.1 What is Adam?

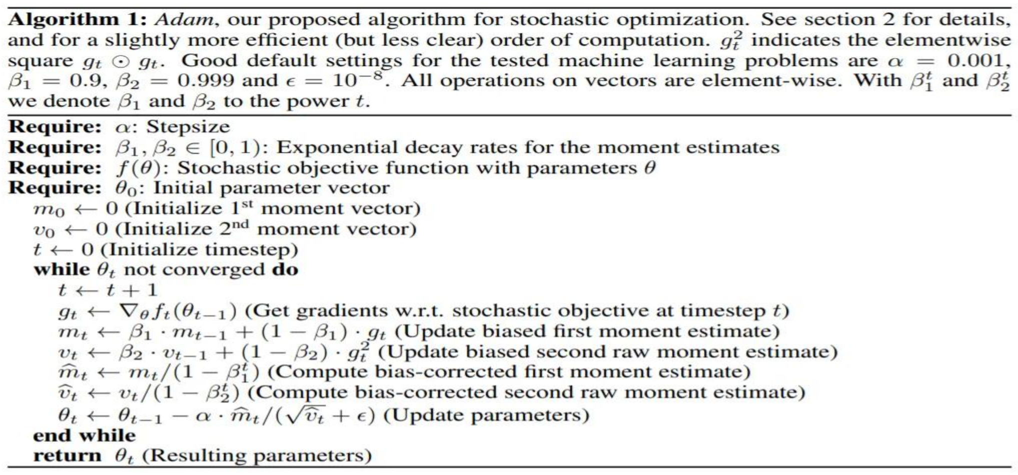
Adam (Kingma & Ba, 2014) is a first-order-gradient-based algorithm of stochastic objective functions, based on adaptive estimates of lower-order moments. Adam is one of the latest state-of-the-art optimization algorithms being used by many practitioners of machine learning. The first moment normalized by the second moment gives the direction of the update.

1.2.5.2 How does Adam work?

Adam’s update rule:

$$\theta_{n+1} \quad = \quad \theta_n - \frac{\alpha}{\sqrt{\hat{v_n} + \epsilon}} \hat{m_n}$$

Adam’s algorithm:



Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

1.2.6 Adadelta

1.2.6.1 What is Adadelta?

Adadelta optimization is a stochastic gradient descent method that is based on adaptive learning rate per dimension to address two drawbacks:

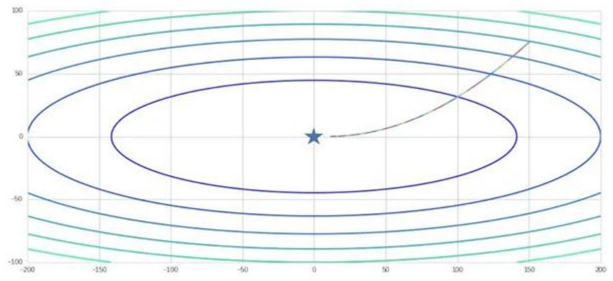
- The continual decay of learning rates throughout training.
- The need for a manually selected global learning rate.

Adadelta is a more robust extension of Adagrad that adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients. This way, Adadelta continues learning even when many updates have been done. Compared to Adagrad, in the original version of Adadelta you don't have to set an initial learning rate. In this version, the initial learning rate can be set, as in most other Keras optimizers.

1.2.6.2 How does Adadelta work?

In a nutshell, Adadelta uses two state variables, to store a leaky average of the second moment of the gradient and Δ x t to store a leaky average of the second moment of the change of parameters in the model itself.

Adadelta can be visualized as:



1.3 Comparison

Below is a comparison table of some popular Optimizer methods:

Algorithm	**Type**	**Pros**	**Cons**
Gradient Descent	Gradient-based	Fast, efficient, easy computation	Can get stuck in a local minima
Momentum	Gradient-based	Overcomes local minima, reduces the oscillations and high variance of the parameters	One more hyper-parameter is added which needs to be selected manually and accurately
			Can be slow at first
Adagrad	Gradient-based	Adapts to varying parameter variance	Computationally expensive as a need to calculate the second order derivative
		Learning rate changes for each training parameter.	Can lead to oscillations
RMSProp	Gradient-based	Adapts to varying parameter variance	Sometimes lead to slow convergence and is also sensitive to the learning rate
		Reducing the training time of the neural network and also prevents overfitting	Can lead to oscillations
Adadelta	Non-gradient-based	Overcomes local minima	Can lead to oscillations

		Now the learning rate does not decay and the training does not stop	Computationally expensive
Adam		The method is too fast and converges rapidly	Can lead to oscillations
		Combines the benefits of Momentum and RMSProp	Computationally costly

Notes:

- *Type:* Gradient-based algorithms use the gradient of the loss function to update the parameters of the model. Non-gradient-based algorithms do not use the gradient of the loss function.
- *Pros:* Fast, efficient, scalable, adaptable
- *Cons:* Can get stuck in local minima, can lead to oscillations

Conclusion:

The choice of optimizer depends on a number of factors, including the size of the dataset, the nature of the loss function, the number of parameters in the model, and the desired performance.

QUESTION 2 – CONTINUAL LEARNING AND TEST PRODUCT

2.1 Continual Learning

Continual Learning is a field of machine learning that focuses on training machine learning models that can learn and adapt to new data continuously. This is in contrast to the traditional approach to machine learning, in which models are trained on a fixed dataset and then used to predict new values.

When data changes over time, traditional machine learning models can start to perform poorly. Continual Learning provides a number of solutions to this problem, such as:

- **Forgetting avoidance:** Machine learning models are designed to avoid forgetting what they have learned in the past. This can be done using techniques such as elastic weight consolidation or distillation.
- **Incremental learning:** Machine learning models are trained on new data gradually, rather than replacing the old model entirely. This helps the model learn a little bit at a time, avoiding being overwhelmed by the new data.
- **Lifelong learning:** Machine learning models are designed to be able to learn and adapt to new data throughout their lifetime. This can be done using techniques such as continual reinforcement learning or continual active learning.

2.2 Test Production

Test Production is a process in which a machine learning solution is deployed into production, but only a portion of the production data is used to evaluate the performance of the model. This helps to reduce the risk of deploying a new machine learning solution.

Test Production provides a number of benefits, such as:

- **Reduced risk:** If the model performs poorly in production, it can be pulled out of production without causing much damage.
- **Increased scalability:** Test Production allows machine learning models to be deployed gradually, helping to increase the scalability of the system.
- **Increased adaptability:** Test Production helps machine learning models to adapt to changes in production data.

2.3 The importance of Continual Learning and Test Production

- *Continual Learning and Test Production* are both important techniques to consider when building a machine learning solution to solve a particular problem.

- *Continual Learning* helps machine learning models to adapt to changing data over time, ensuring that the models always perform effectively.

- *Test Production* helps to reduce the risk of deploying a new machine learning solution, helping to increase the scalability and adaptability of the system.

2.4 Some examples of the use of Continual Learning and Test Production

Continual Learning and Test Production have been used successfully in a number of fields, such as:

- **Speech recognition:** Speech recognition models need to be updated continuously to reflect changes in the way people speak.
- **Computer vision:** Computer vision models need to be updated continuously to reflect changes in the environment.
- **Fraud detection:** Fraud detection models need to be updated continuously to reflect new fraud schemes.

2.5 Conclusion

Continual Learning and Test Production are important techniques to consider when building a machine learning solution to solve a particular problem. By using these techniques, machine learning developers can create machine learning solutions that are more effective, reliable, and scalable.

2.6 Specific problem

Suppose we need to build a machine learning solution to detect fraud in credit card transactions. This solution will use historical transaction data to train a machine learning model that can distinguish between legitimate transactions and fraudulent transactions.

In this case, the transaction data can change over time due to factors such as:

- New fraud schemes are developed.
- User shopping habits change.
- Economic conditions change.

2.6.1 Application of Continual Learning

Continual Learning can help:

- Reduce fraud risk: As transaction data changes over time, new fraud schemes may emerge. Continual Learning helps the machine learning model adapt to these changes, ensuring that the model can always detect fraud effectively.

- Improve reliability: Continual Learning helps the machine learning model avoid being overwhelmed by new data. This helps the model learn more effectively and accurately.

2.6.2 Application of Test Production

Test Production can help:

- Reduce deployment risk: When deploying a new machine learning solution, there may be risks such as the model performing poorly or causing other negative impacts. Test Production helps to mitigate these risks by deploying the new machine learning solution in the production environment, but only using a portion of the production data to evaluate the performance of the model.

- Improve scalability: Test Production helps the machine learning solution to be deployed gradually, helping to improve the scalability of the system.

TÀI LIỆU THAM KHẢO

Tiếng Việt

1. Blog Machine Learning cơ bản
2. Trần Trung Trực, *Optimizer- Hiểu sâu về các thuật toán tối ưu (GD,SGD,Adam,..),* , Viblo

Tiếng Anh

3. TensorFlow
4. Keras
5. Sanket Doshi, *Various Optimization Algorithms For Training Neural Network*, Towards Data Science
6. Jason Huang, *Optimization*, Cornell University
7. Gunand Mayanglambam, *Deep Learning Optimizers,* Towards Data Science
8. Vijay Sharma, *Adagrad and Adadelta Optimizer: In-Depth Explanation*, Medium