

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



FINAL REPORT INTRODUCTION TO MACHINE LEARNING

Question 1

Instructor: **Ph.D LE ANH CUONG**

Student: **Ho Huu An – 521H0489**

Class : **21H50301**

Year : **25**

HO CHI MINH CITY, 2023

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



FINAL REPORT INTRODUCTION TO MACHINE LEARNING

Question 1

Instructor: **Ph.D LE ANH CUONG**

Student: **Ho Huu An – 521H0489**

Class : **21H50301**

Year : **25**

HO CHI MINH CITY, 2023

ACKNOWLEDGEMENT

I would like to express my deep appreciation to Ph.D. Le Anh Cuong from Ton Duc Thang University for his invaluable assistance and guidance. His expertise in the field of machine learning has played a crucial role in shaping the direction of this project.

Ph.D. Le Anh Cuong's unwavering commitment to excellence and his openness to share his knowledge have greatly contributed to the success of this endeavor. His mentorship has not only improved my comprehension of utilizing machine learning algorithms and models but has also motivated me to explore new avenues in logic and machine learning deployment.

I sincerely thank Ph.D. Le Anh Cuong for his continuous encouragement and support, which have been pivotal in the completion of this project. His dedication to creating a conducive learning environment has been a source of inspiration, and I am grateful for the opportunity to work under his guidance.

Ho Chi Minh city, 20th December, 2023

Author

(Sign and write full name)



Ho Huu An

THIS PROJECT WAS COMPLETED AT TON DUC THANG UNIVERSITY

I fully declare that this is my own project and is guided by Mr. Mai Van Manh; The research contents 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 are collected by the author himself from different sources, clearly stated in the reference section.

Besides that, the project also uses a number of comments, assessments as well as data from other authors, other agencies and organizations, with citations and source annotations.

Should any frauds were 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, 20th December, 2023

Author

(Sign and write full name)



Ho Huu An

CONFIRMATION AND ASSESSMENT SECTION

Instructor confirmation section

Ho Chi Minh December, 2023
(Sign and write full name)

Evaluation section for grading instructor

Ho Chi Minh December 2023
(Sign and write full name)

SUMMARY

In machine learning model training, optimizers are algorithms used to adjust the model's parameters iteratively. They play a crucial role in minimizing the loss function. Commonly used optimizers include Stochastic Gradient Descent (SGD), Adam, RMSprop, and Adagrad. SGD is simple and efficient but can be noisy, while Adam combines adaptive learning rates and momentum for fast convergence. RMSprop and Adagrad also adapt the learning rates but in different ways. The choice of optimizer depends on the specific problem and the characteristics of the dataset.

Continual learning refers to the ability of a machine learning system to learn and adapt continuously over time, without forgetting previously learned knowledge. It addresses the challenge of retaining knowledge from previous tasks while accommodating new tasks or data. Techniques such as regularization, replay, and parameter isolation have been proposed to mitigate catastrophic forgetting and enable continual learning.

Test production, on the other hand, is a crucial step in building a machine learning solution for a specific problem. It involves designing and implementing a testing process to evaluate the performance and effectiveness of the trained model. This includes selecting appropriate evaluation metrics, preparing test datasets, and conducting rigorous testing to ensure the model's reliability and generalizability.

INDEX

ACKNOWLEDGEMENT	i
CONFIRMATION AND ASSESSMENT SECTION	iii
SUMMARY	iv
LIST OF TABLES, DRAWINGS, GRAPHICS	3
CHAPTER 1 – OPTIMIZER	4
1.1 Gradient Descent	4
Advantages:.....	7
Disadvantages:	7
1.2 Stochastic Gradient Descent	7
Advantages:.....	9
Disadvantages:	9
1.3 Gradient Descent with Momentum	10
Advantages	11
Disadvantages	11
1.4 Nesterov accelerated gradient (NAG).....	11
1.5 Adagrad	12
Advantages:.....	13
Disadvantages:	13
1.6 RMSprop	14
Advantages:.....	14
Disadvantages:	14
1.7 Adam	14
Steps in the Adam Optimization Algorithm	15
Advantages:.....	15
Disadvantages:	15
Compare Algorithms	16

CHAPTER 2 – CONTINUAL LEARNING AND TEST PRODUCTION.....	18
2.1 Continual learning.....	18
Stateless retraining & Stateful training	19
How often to Update your models	20
2.2 Test production	20
Testing in Production Strategies	21
Shadow Deployment.....	21
A/B Testing.....	21
Bandits	22
Canary Release	23

LIST OF TABLES, DRAWINGS, GRAPHICS

List of figures

Figure 1: Derivative	4
Figure 2: Import Library for Gradient Descent Example Code	5
Figure 3: Functions for Gradient Descent Example Code	6
Figure 4: Test for Gradient Descent Example Code	6
Figure 5: Compare Stochastic Gradient Descent vs Gradient Descent.....	8
Figure 6: Example about Stochastic Gradient Descent.....	9
Figure 7: Compare Gradient Descent with physical phenomena.....	10
Figure 8: Nesterov's idea is accelerated gradient	12
Figure 9: Example AdaGrad	13
Figure 10: Heavy with Friction, where the ball with mass overshoots the local minimum and settles at the flat minimum.....	14
Figure 11: Stateless retraining VS Stateful training.....	19
Figure 12: Train model on data from different time windows in the past and test on data from today	20

CHAPTER 1 – OPTIMIZER

An optimizer is an algorithm used in machine learning that adjusts the parameters of a model to minimize the difference between predicted and actual outcomes, improving the model's accuracy.

1.1 Gradient Descent

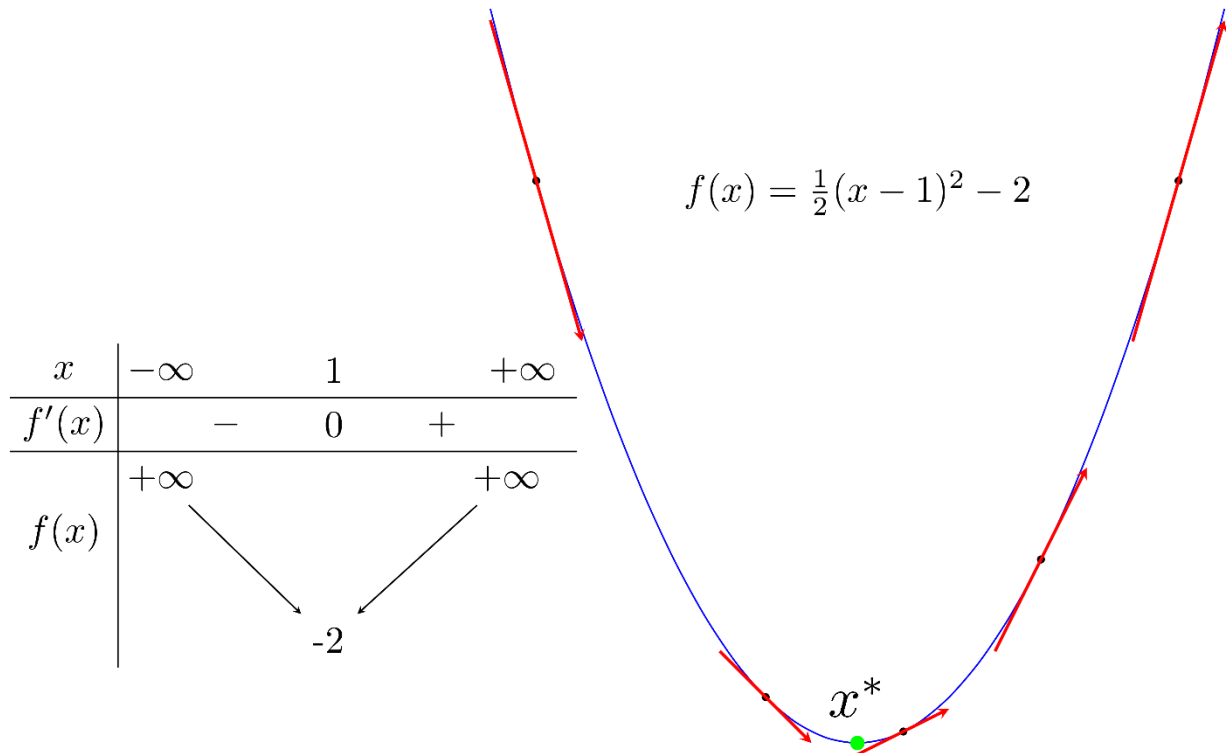


Figure 1: Derivative

Suppose x_t is the point we find after the t^{th} iteration. We need to find an algorithm to bring x_t as close to x^* as possible.

1. If the derivative of the function is at x_t : $f'(x_t) > 0$ then x_t is to the right compared to x^* (and vice versa). To get the next point x_{t+1} closer to x^* , we need to move x_t to the left, that is, to the negative side (and vice versa). In other words, we need to move the opposite sign of the derivative:

$$x_{t+1} = x_t + \Delta$$

2. The farther x_t is from x^* to the right, the larger $f(x_t)$ is than 0 (and vice versa).

So, the amount of movement Δ , most intuitively, is proportional to $-f'(x_t)$.

The two comments above give a simple update:

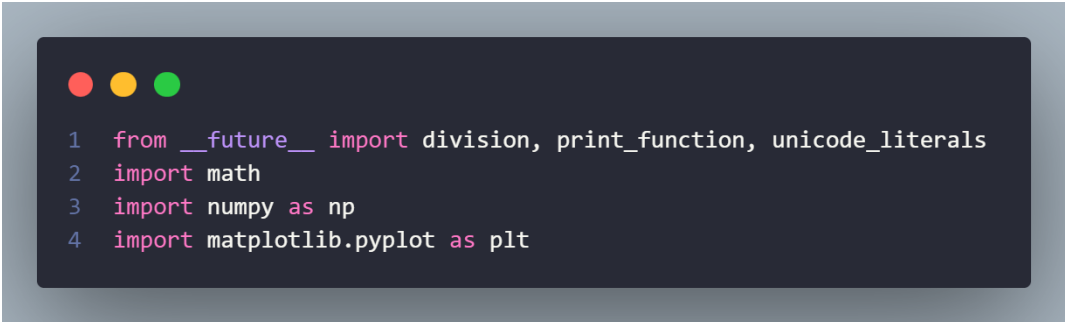
$$x_{t+1} = x_t - \eta f'(x_t)$$

In which η (read as eta) is a number called learning rate. The minus sign shows that we have to go against the derivative

Rule to remember: always go in the opposite direction of the derivative.

Gradient descent depends on many factors: for example, choosing different initial x points will affect the convergence process; or the learning rate is too large or too small, it also has an impact: if the learning rate is too small, the convergence speed is very slow, affecting the training process, and if the learning rate is too large, it will quickly reach the target after a few minutes. However, the algorithm does not converge and loops around the destination because the jump is too large.

Example:



```
1 from __future__ import division, print_function, unicode_literals
2 import math
3 import numpy as np
4 import matplotlib.pyplot as plt
```

Figure 2: Import Library for Gradient Descent Example Code

Consider the function $f(x) = x^2 + 5\sin(x)$ with the derivative $f'(x) = 2x + 5\cos(x)$.

Suppose starting from a certain point x_0 , at the t th loop, we will update as follows:

$$x_{t+1} = x_t - \eta (2x_t + 5\cos(x_t))$$



```

1  def grad(x):
2      return 2*x+ 5*np.cos(x)
3  def cost(x):
4      return x**2 + 5*np.sin(x)
5  def myGD1(eta, x0):
6      x = [x0]
7      for it in range(100):
8          x_new = x[-1] - eta*grad(x[-1])
9          if abs(grad(x_new)) < 1e-3:
10             break
11         x.append(x_new)
12     return (x, it)

```

Figure 3: Functions for Gradient Descent Example Code

‘grad’ to calculate the derivative

‘cost’ to calculate the value of the function. This function is not used in the algorithm but is often used to check whether the calculation of the derivative is correct or to see if the value of the function decreases with each iteration.

‘myGD1’ is the main part that implements the Gradient Descent algorithm mentioned above. The input to this function is the learning rate and the starting point. The algorithm stops when the derivative has a sufficiently small magnitude.



```

1  (x1, it1) = myGD1(.1, -5)
2  (x2, it2) = myGD1(.1, 5)
3  print('Solution x1 = %f, cost = %f, obtained after %d iterations'%(x1[-1], cost(x1[-1]), it1))
4  print('Solution x2 = %f, cost = %f, obtained after %d iterations'%(x2[-1], cost(x2[-1]), it2))

```

Figure 4: Test for Gradient Descent Example Code

Different starting points:

After having the necessary functions, I try to find solutions with different initialization points: $x_0 = -5$ and $x_0 = 5$.

So with different initial points, our algorithm finds nearly the same solution, although with different convergence rates.

Output:

Solution $x_1 = -1.110667$, cost = -3.246394, obtained after 11 iterations

Solution $x_2 = -1.110341$, cost = -3.246394, obtained after 29 iterations

The choice of learning rate is very important in practical problems. Choosing this value depends heavily on each problem and requires some experiments to choose the best value. In addition, depending on some problems, GD can work more effectively by choosing an appropriate learning rate or choosing a different learning rate in each iteration.

Advantages:

Simple gradient descent algorithm that is simple to grasp. By adjusting the weights after each loop, the technique solves the challenge of optimizing the neural network model.

Disadvantages:

The Gradient Descent method has various disadvantages due to its simplicity, such as relying on the initial initial solution and learning rate.

As an illustration, a function with two global minimums will give two distinct final solutions depending on the two original beginning points.

A big learning rate will cause the algorithm to fail to converge and linger about the target due to the large leap; a modest learning rate will effect the training pace.

1.2 Stochastic Gradient Descent

Gradient Descent is a variant of Stochastic. Instead of updating the weight once after each epoch, we will update the weight N times in each epoch using N data points. On the one hand, SGD will drop the pace by one epoch. Looking in the opposite way,

SGD will converge quite fast after only a few epochs. The SGD formula is similar to the GD formula, however it is applied to each data point.

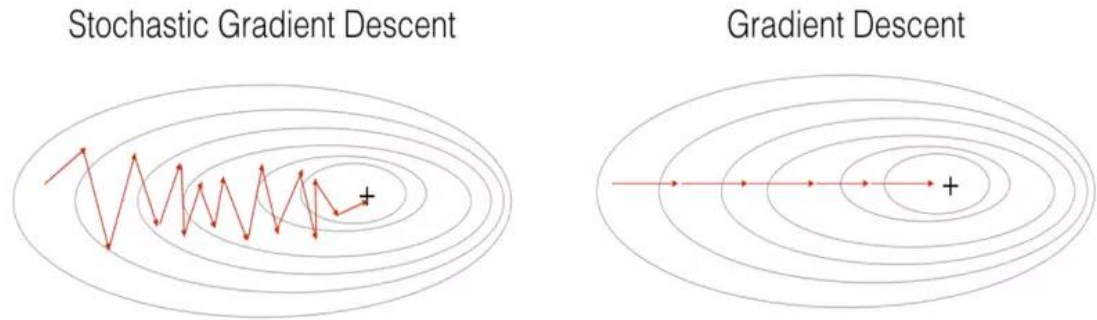


Figure 5: Compare Stochastic Gradient Descent vs Gradient Descent

Looking at the image above, we can observe that SGD has a more zigzag course than GD. It's simple to grasp since one data point cannot represent all of the facts. Because GD has limits for big datasets (several million data points), computing the derivative on the full data set over each loop becomes tedious. Furthermore, education is not suited to online learning. When data is regularly changed (for example, by adding registered users), we must recalculate the derivative on the full data set, resulting in a long computation time and the method no longer being online. As a result, SGD was created to address this issue, because each time new data is supplied, only one data point needs to be modified, making it ideal for online learning.

For example, with 10,000 data points, we may acquire a satisfactory solution within only 3 epochs, but with GD, we must spend up to 90 epochs to reach the same result.

Return back example in Gradient Descent:

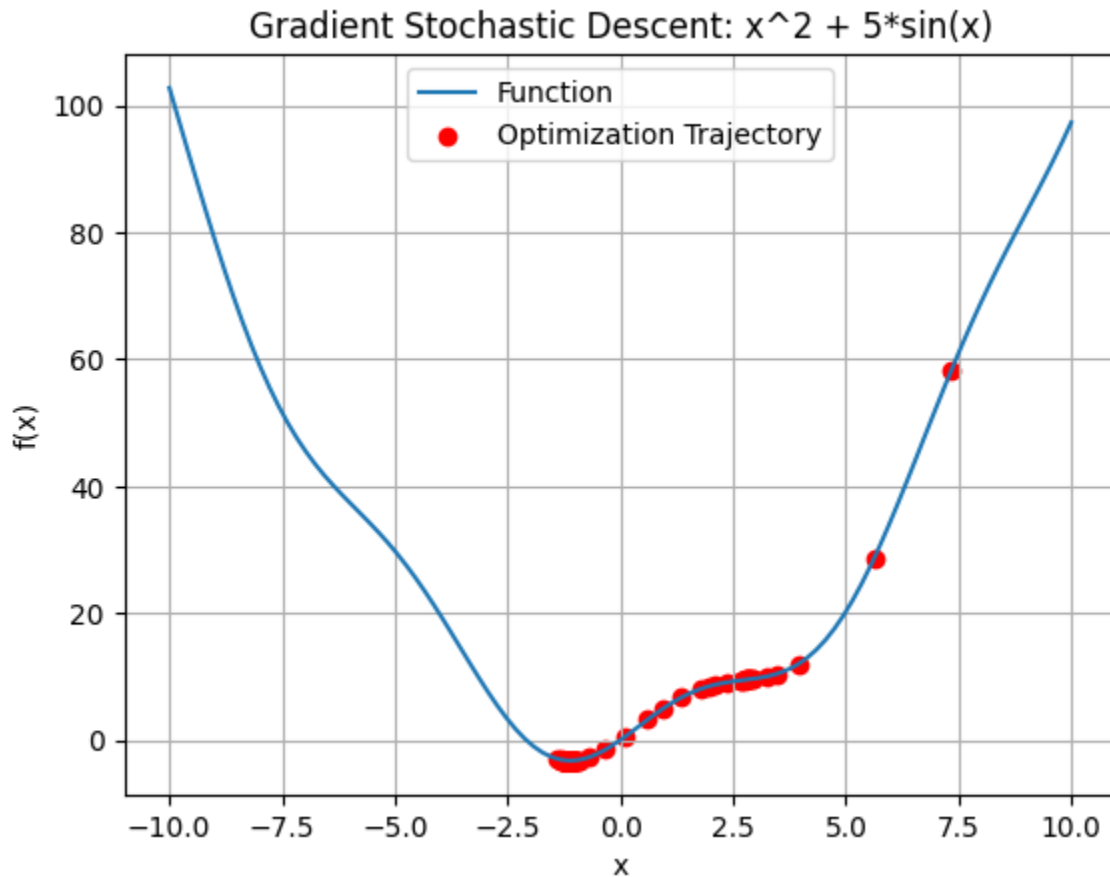


Figure 6: Example about Stochastic Gradient Descent

Advantages:

The algorithm has the ability to handle big databases that Gradient Descent does not. This optimization technique is still widely employed today.

Disadvantages:

The algorithm has not yet solved the two major disadvantages of gradient descent (learning rate, initial data points). Therefore, we have to combine SGD with some other algorithms such as Momentum, AdaGrad, etc.

1.3 Gradient Descent with Momentum

The GD algorithm is often compared to the effect of gravity on a marble placed on a surface shaped like a valley like figure 1a) below. Regardless of whether we place the marble at A or B, the marble will eventually roll down and end up at position C.

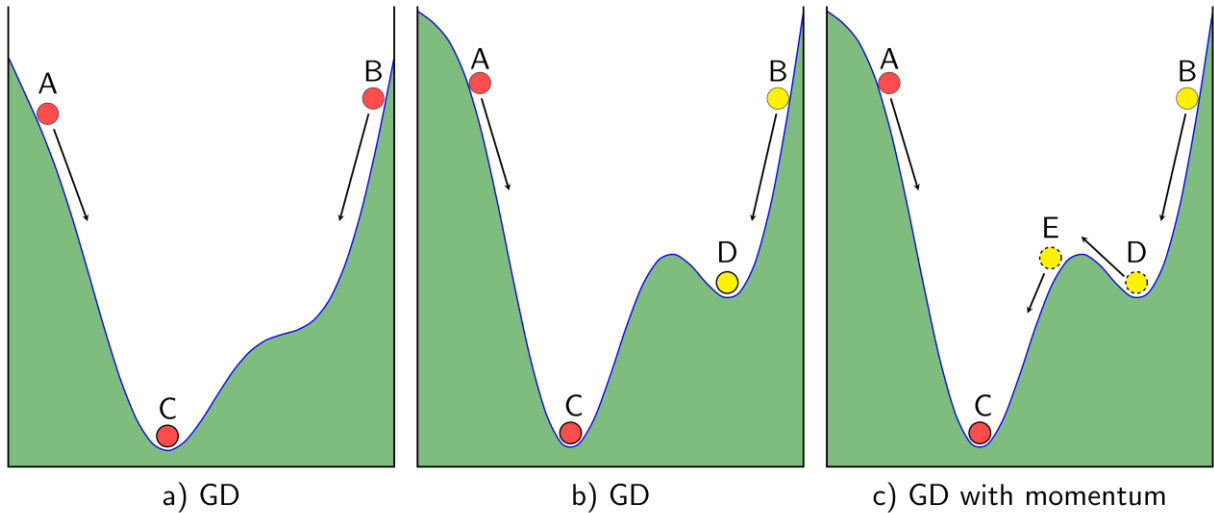


Figure 7: Compare Gradient Descent with physical phenomena

Nevertheless, if the surface contains two valley bottoms, as illustrated in Figure b), the eventual location of the ball will be C or D, depending on whether it is put at A or B. Point D is an unexpected local minimum point.

If we consider more physically, still in Figure b), if the starting velocity of the ball at point B is big enough, the ball can continue to go up the slope to the left of D when it rolls to point D. If the starting velocity is increased, the ball can move uphill to point E and subsequently downhill to point C, as shown in Figure c). This is just what we desire.

Based on this phenomenon, an algorithm was created to overcome the problem of GD's solution falling into an undesirable local minimum point. That algorithm is called Momentum

Express momentum mathematically

In Gradient Descent, we need to calculate the amount of change at time t to update the new position of the solution (ie the marble). If we think of this quantity as the physical velocity v_t , the new position of the marble will be $\theta_{t+1} = \theta_t - v_t$. The minus sign represents having to move against the derivative. Our job now is to calculate the quantity v_t so that it both carries information about the slope (ie the derivative) and also carries information about the momentum, which is the previous velocity v_{t-1} (we consider the initial velocity $v_0=0$). In the simplest way, we can add (weight) these two quantities:

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$

In which γ is usually chosen as a value of about 0.9, v_t is the velocity at the previous time, $\nabla_{\theta} J(\theta)$ is the slope of the previous point. Then the new position of the marble is determined as follows: $\theta = \theta - v_t$

This simple algorithm proves to be very effective in practical problems (in high-dimensional space, the calculation method is completely similar).

Advantages

Gradient Descent, the best solution, addresses the problem by stopping at the local minimum rather than the global minimum.

Disadvantages

Although momentum helps the marble move upward towards the target, it still takes a long time to oscillate back and forth before halting fully, which is explained by the marble's momentum.

1.4 Nesterov accelerated gradient (NAG)

Momentum helps the marble overcome the local minimum slope, however, there is a limitation we can see in the example above: When approaching the destination, momentum still takes a lot of time before stopping. The reason is also because there is momentum. There is another method that further helps overcome this, a method called Nesterov accelerated gradient (NAG), which helps the algorithm converge faster.

The basic idea is to predict future direction, which means looking one step ahead! Specifically, if we use the momentum term γv_{t-1} to update, we can approximate the marble's next position as $\theta - \gamma v_{t-1}$. So, instead of using the gradient of the current point, NAG go one step ahead, use the gradient of the next point. Follow the image below:

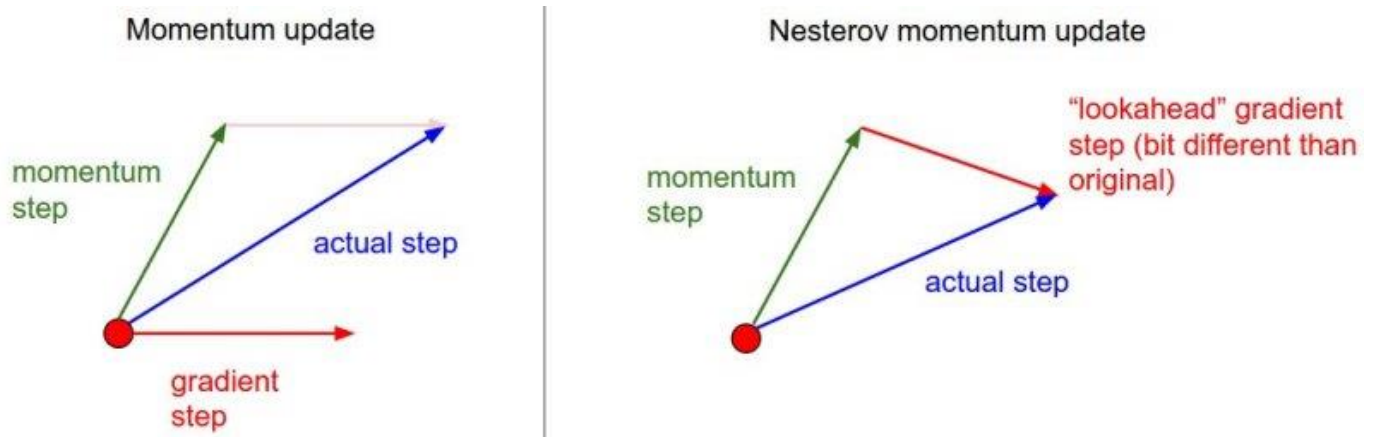


Figure 8: Nesterov's idea is accelerated gradient

- With normal momentum: the amount of change is the sum of two vectors: momentum vector and gradient at the present time.
- With Nesterov momentum: the amount of change is the sum of two vectors: the momentum vector and the gradient at the time that is approximated as the next point

NAG's updated formula is given as follows:

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta - \gamma v_{t-1}) \quad \theta = \theta - v_t$$

1.5 Adagrad

Unlike previous algorithms where the learning rate is almost the same during the training process (learning rate is constant), Adagrad considers learning rate as a parameter. That is, Adagrad will let the learning rate change after each time t .

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

In there :

η : constant

g_t : gradient at time t

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

G: is a diagonal matrix where each element on the diagonal (i,i) is the square of the parameter vector derivative at time t.

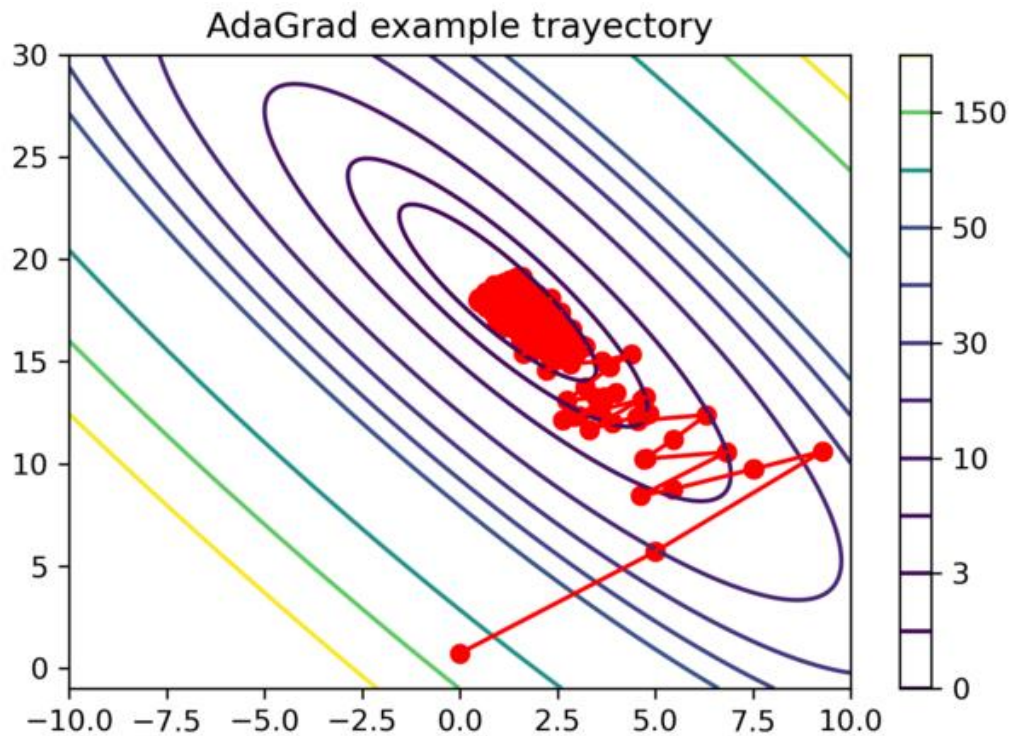


Figure 9: Example AdaGrad

Advantages:

- It removes the need to manually adjust the learning rate.
- When the weights are scaled unequally, convergence is quicker and more reliable than basic SGD.
- It is unaffected by the size of the master step.

Disadvantages:

- The weakness of Adagrad is that the sum of squared variations will grow larger over time until it makes the learning rate extremely small, causing training to freeze.

1.6 RMSprop

RMSprop solves Adagrad's decreasing learning rate problem by dividing the learning rate by the average of the squares of the gradient.

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1 g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Advantages:

The most evident advantage of RMSprop is that it addresses the issue of Adagrad's progressively declining learning speed (the problem of gradually decreasing learning speed over time causes training to slow down, perhaps leading to freezing).

Disadvantages:

The RMSprop method can only provide a local minimum solution, not a global minimum solution like Momentum. As a result, individuals will mix Momentum and RMSprop algorithms to produce an Adam optimal algorithm.

1.7 Adam

Adam is a combination of Momentum and RMSprop. If explained in terms of physical phenomena, Momentum is like a ball rushing downhill, and Adam is like a very heavy ball with friction, so it easily overcomes the local minimum to the global minimum

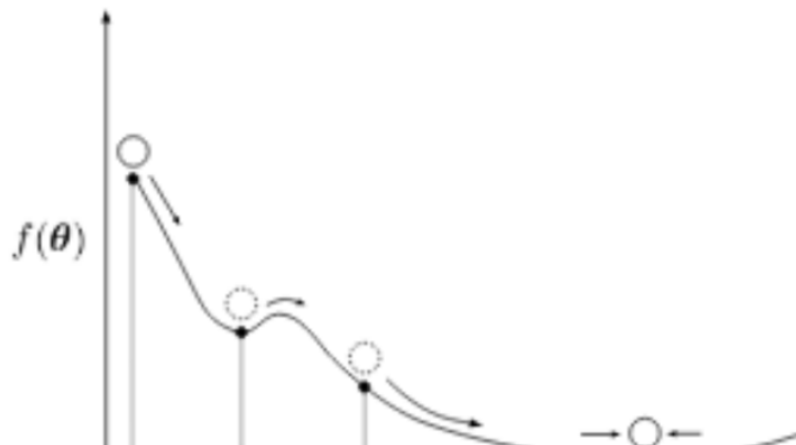


Figure 10: Heavy with Friction, where the ball with mass overshoots the local minimum and settles at the flat minimum

and when it reaches the global minimum it does not. It takes a long time to oscillate back and forth around the target because it has friction so it's easier to stop.

Steps in the Adam Optimization Algorithm

1. Set the moving averages (m and v) for the first and second moments to zero.
2. Determine the gradient of the loss function with respect to the model parameters.
3. Use exponentially decaying averages to update the moving averages. This entails computing m_t and v_t as weighted averages of prior moments and the current gradient.
4. Correct the moving averages for bias, especially in the early iterations.
5. Divide the bias-corrected first moment by the square root of the bias-corrected second moment, with a minor constant (ϵ) added for numerical stability.
6. Apply the estimated modifications to the model parameters.
7. Repeat steps 2-6 for a predetermined number of iterations or until convergence is reached.

Advantages:

- Adaptive Learning Rate: Adam employs individually adaptive learning rates for each parameter, resulting in enhanced convergence and quicker deep learning model training.
- Effective with Sparse Data: Because of its capacity to accommodate changing learning rates for different parameters, Adam works well with sparse data.
- Flexibility and diversity: Adam combines two prominent optimization approaches, momentum and RMSprop, which improves flexibility and efficacy for a wide range of machine learning applications.

Disadvantages:

- Not difficult to implement
- Effective complexity
- Less memory required.
- Suitable for problems with unstable variability and fragmented training data.
- Hyperparameters are varied efficiently and require little adjustment

Compare Algorithms

Algorithms	Character	Advantages	Disadvantages
Gradient Descent	Recalculate the weights using the derivative of the full training data set.	- Simple to apply and comprehend.	- Slow convergence.
Stochastic Gradient Descent	Randomly update the weights depending on each training data point.	- Rapid calculation of large volumes of data.	- Unstable weight update.
Gradient Descent with Momentum	Update the weights based on the derivative of the entire training dataset, with an additional momentum term.	- Helps accelerate convergence by adding a fraction of the previous update vector to the current update. - Reduces oscillations and overshooting in the weight updates. - Can converge faster than standard Gradient Descent.	- Requires tuning the momentum parameter, which can be time-consuming. - Can overshoot the optimal solution if the momentum parameter is set too high.
AdaGrad	For each parameter, adjust the learning rate depending on the gradient history. Appropriate for sparse data optimization methods.	- Works well with limited data.	- When training is progressed, efficiency drops.
RMSprop	Appropriate for problems with scarce data. Divide the learning rate by the magnitude of the determined closest gradient.	- Effective for issues with rapid convergence.	- Need to select an adequate learning rate

Adam	It combines the benefits of RMSprop and Momentum. Use momentum parameters and a learning rate that is time gradient adjustable.	- High performance with large amounts of data.	- It is necessary to pay close attention to the specifications.
-------------	---	--	---

CHAPTER 2 – CONTINUAL LEARNING AND TEST PRODUCTION

2.1 Continual learning

The concept of continuous learning is to update your model as new data becomes available, allowing your model to stay up with the current data distributions.

Once your model is updated, it cannot be blindly released to production. It needs to be tested to ensure that it is safe and that it is better than the current model in production. This is where the next section "Testing Production" comes in.

Continual learning is frequently misunderstood:

- Continual learning does not refer to a special class of ML algorithms that allow for incremental update of the model when every single new datapoint becomes available. Examples of this special class of algorithms are sequential bayesian updating and KNN classifiers. This class of algorithms is small and is sometimes referred to "online learning algorithms".
- Continual learning does not mean starting a retraining job every time a new data sample becomes available. In fact this is dangerous, because it makes neural networks susceptible to catastrophic forgetting.

The primary purpose for this is to assist your model in keeping up with changes in data distribution:

- Use cases in which unexpected and rapid changes can happen.
- Use cases in which it is not possible to get training data for a particular event. An example of this are e-commerce models in Black Friday or some other sale event that has never been tried before. It is very hard to gather

historical data to predict user behaviour in Black Friday, so your model must adapt throughout the day.

- Use cases that are sensitive to the cold start problem. This problem happens when model has to make predictions for a new (or logged out) user that has no historical data (or data is outdated).

Stateless retraining & Stateful training

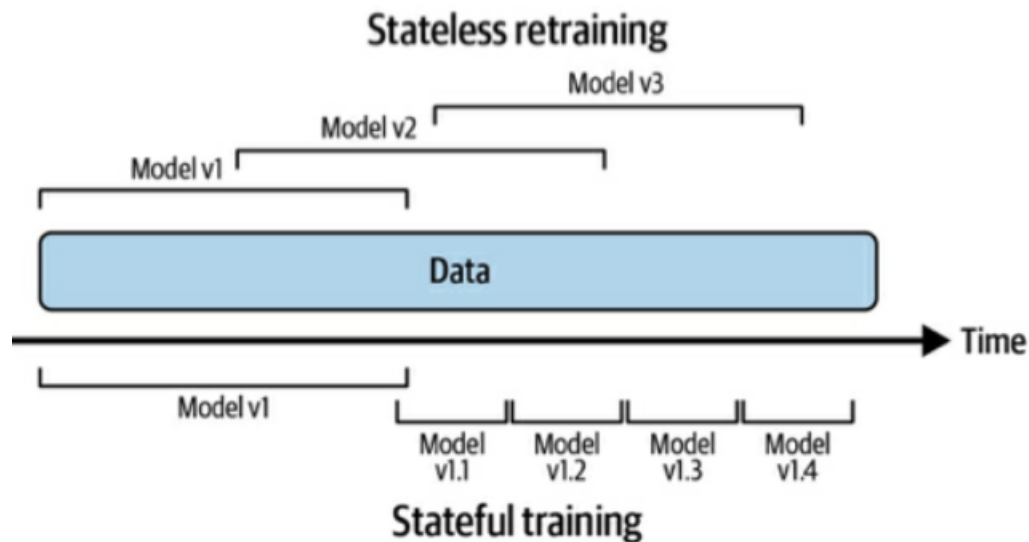


Figure 11: Stateless retraining VS Stateful training

Stateless retraining

Retrain model from scratch each time, using randomly initialised weights and fresher data.

- There might be some overlap with data that had been used for training previous model version.
- Most companies start doing continual learning using stateless retraining.

Stateful training

Initialise model with the weights from the previous training round and continue the training using new unseen data.

- Allows your model to update with significantly less data.

- Allows your model to converge faster and use less compute power.

How often to Update your models

The first need to understand and determine what is the gain we get when we update model with fresh data. The more the gain, the more frequently it should be retrained.

Measuring the value of data freshness: One way to quantify the value of fresher data is to train the same model architecture with data from 3 different periods of time and then testing each model against current labelled data

If you discover that letting the model stale for 3 months causes a 10% difference in the current test data accuracy, and 10% is unacceptable, then you need to retrain in less than 3 months.

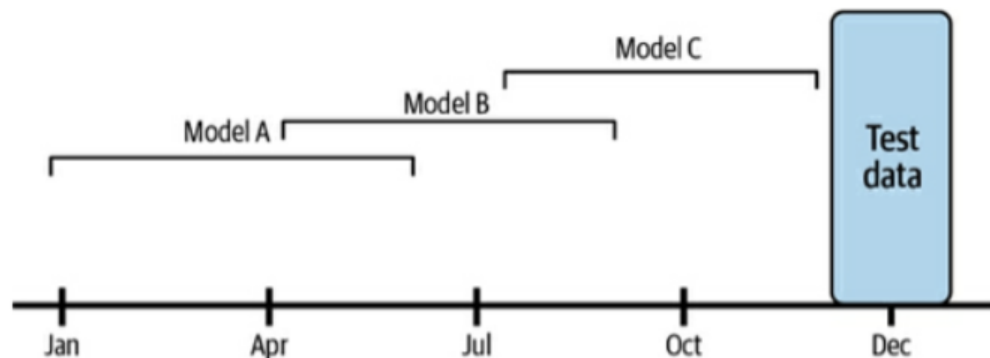


Figure 12: Train model on data from different time windows in the past and test on data from today

2.2 Test production

To sufficiently test your models before making them widely available you need both pre-deployment offline evaluations and testing in production. Offline evaluations alone are not sufficient.

Each team should ideally spell out a clear pipeline for how models are reviewed, including which tests to run, who performs them, and the criteria that apply to advance a model to the next level. It is preferable if these assessment processes are automated

and started whenever a new model update is released. The stage promotions should be assessed in the same way that CI/CD is in software engineering.

Testing in Production Strategies

Shadow Deployment

Intuition: Deploy the challenger model in parallel with the existing champion model. Send every incoming request to both models but only serve the inference of the champion model. Log the predictions for the both models to then compare them.

Advantages:

- This is the safest way to deploy your models. Even if your new model is buggy, predictions will not be served.
- It is conceptually simple.
- Your experiment will gather enough data to achieve statistical significance faster than all other strategies as all models receive full traffic.

Disadvantages:

- This technique can't be used when measuring model performance depends on observing how the user interacts with the predictions.
- This technique is expensive to run because it doubles the number of predictions and therefore the number of compute required.

A/B Testing

Intuition: deploy the challenger model alongside the champion model (model A) and route a percentage of traffic* to the challenger (model B). Predictions from the challenger are shown to the users. Use monitoring and prediction analysis on both models to determine if the performance of the challenger is statistically better than the champion.

Advantages:

- Since predictions are served to users, this technique allows us to fully capture how users are reacting to different models.
- A/B testing is simple to understand and there are a lot of libraries and documentation around it.
- It is cheap to run because there is only one prediction per request.
- We won't need to consider the edge cases that arise from parallelising inference requests for online prediction modes.

Disadvantages:

- It is less safe than shadow deployments. We want some stronger offline evaluation guarantee that your model will not miserably fail since you will be putting real traffic through it.
- We have an inherent choice to make between assuming more risk (routing more traffic to the B model) VS gaining enough samples to make an analysis faster.

Bandits

Intuition: Bandits are an algorithm that keeps track of the current performance of each model variant and makes a dynamic decision on every request on whether to use the model that is most performant so far (i.e. exploit the current knowledge) or try out any of the other models to gain more information about them (i.e. explore in case one of the other models is actually better).

Advantages:

- Bandits need a lot less data than A/B testing to determine which model is better. An example given in the book mentions 630K samples to gain 95% confidence with A/B testing VS 12K with bandits.
- Bandits are more data efficient while simultaneously minimising your opportunity cost. In many cases bandits are considered optimal.

- Compared to A/B testing, bandits are safer because if a model is really bad, the algorithm will select it less often. Also convergence will be faster so you will be able to eliminate the bad challenger quickly.

Disadvantages:

- Compared to all other strategies, bandits are much harder to implement because of the need to propagate the feedback into the algorithm continuously.
- Bandits can only be used on certain use cases

Canary Release

Intuition: deploy challenger and champion side by side but start with the challenger taking no traffic. Slowly move traffic from the champion to the challenger (aka the canary). Monitor the performance metrics of the challenger, if they look good, keep going until all traffic is going to the challenger.

- Canary releases can be paired with A/B testing for rigorous measurement of performance differences.
- Canary releases can also be run in "YOLO mode", in which you eyeball the performance difference.
- Other version of a canary release can be to release the challenger model to a smaller market first and then promote to all markets if everything looks good.
- If the challenger model starts having issues, re-route traffic to the champion.

Advantages

- Easy to understand.
- Simplest of all strategies to implement if you already have some feature flagging infrastructure in you company.

- Since challenger predictions will be served, you can use this with models that require user interaction to capture performance.
- Compared to shadow deployments it is cheaper to run. One inference per request.
- If paired with A/B testing, it allows you to dynamically change the amount of traffic that each model is taking.

Disadvantages:

- It opens the possibility to not be rigorous in determining performance differences.
- If releases are not supervised carefully, accidents can happen. This is arguably the least safe option but it is very easy to rollback.