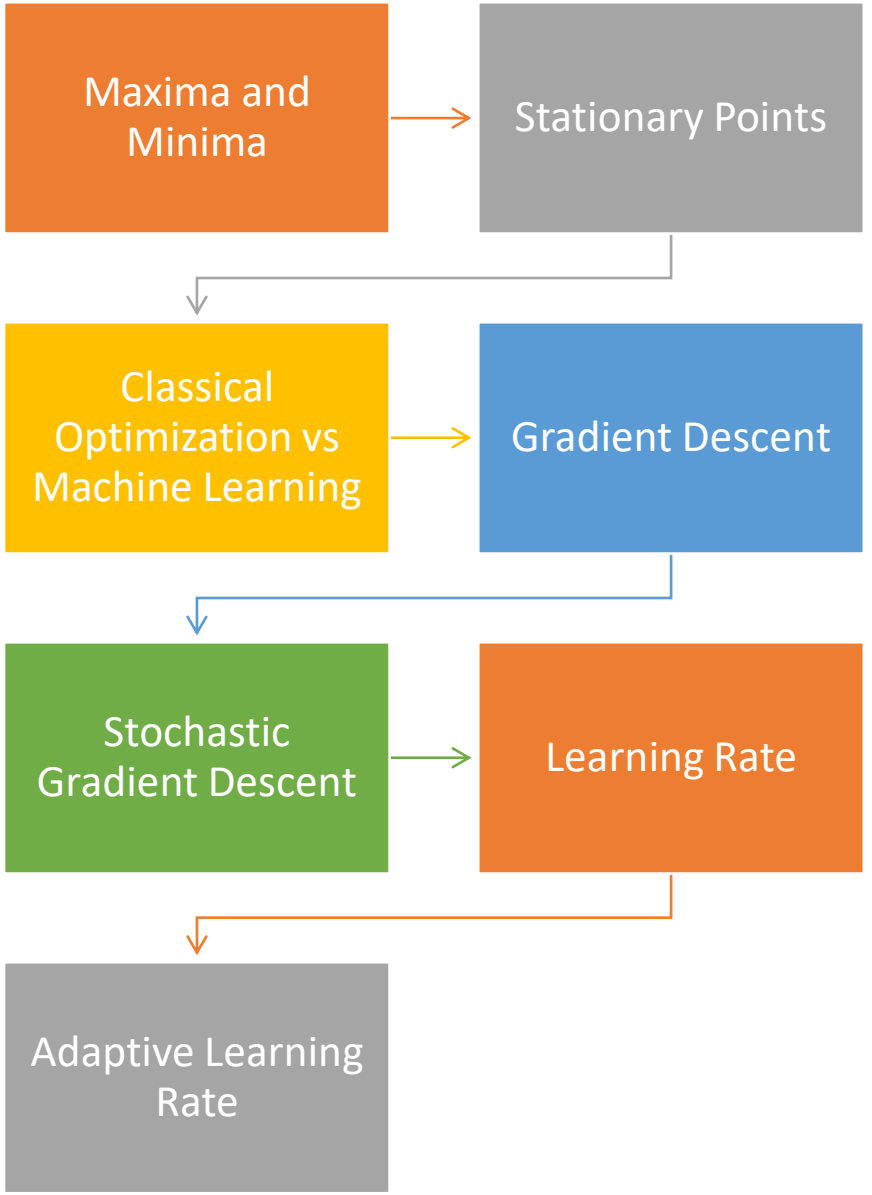
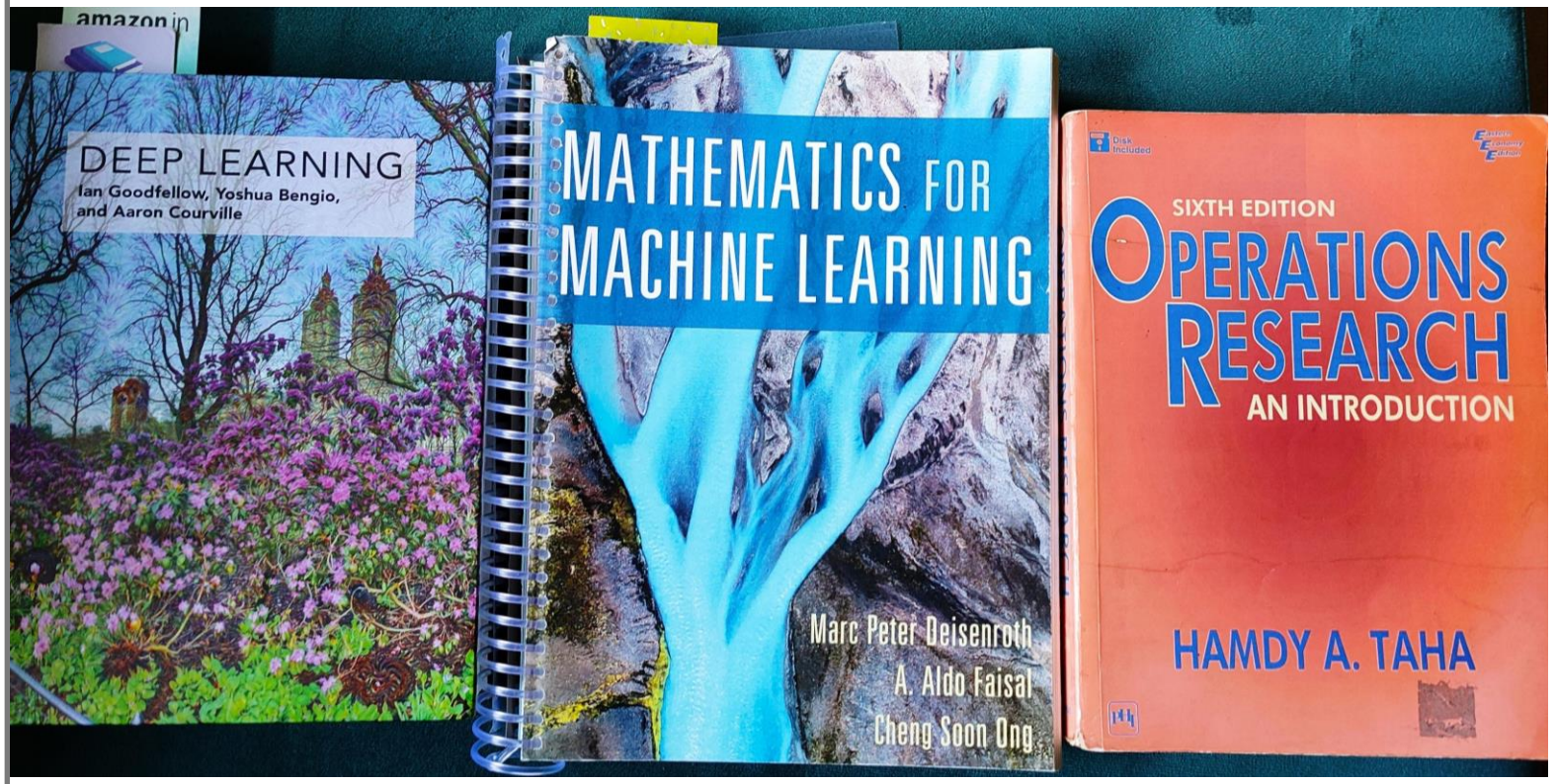
# Learning Rate in Machine Learning

*Dr Pradipta Biswas,* PhD (Cantab)
*Assistant Professor*
*Indian Institute of Science*
*https://cambum.net/*

# Contents


Maxima and Minima
Stationary Points
Classical Optimization vs Machine Learning
Gradient Descent
Stochastic Gradient Descent
Learning Rate
Adaptive Learning Rate

# Acknowledgement



# Prerequisites

- Partial Differential Calculus
- Basics of Gradient Descent
- Backpropagation Algorithm

# Maxima & Minima of a Function

An extreme point of a function f(X) defines either a maximum or a minimum of the function.

Mathematically, a point $X_0 = (x_1,...,x_j,...,x_n)$ is a maximum if

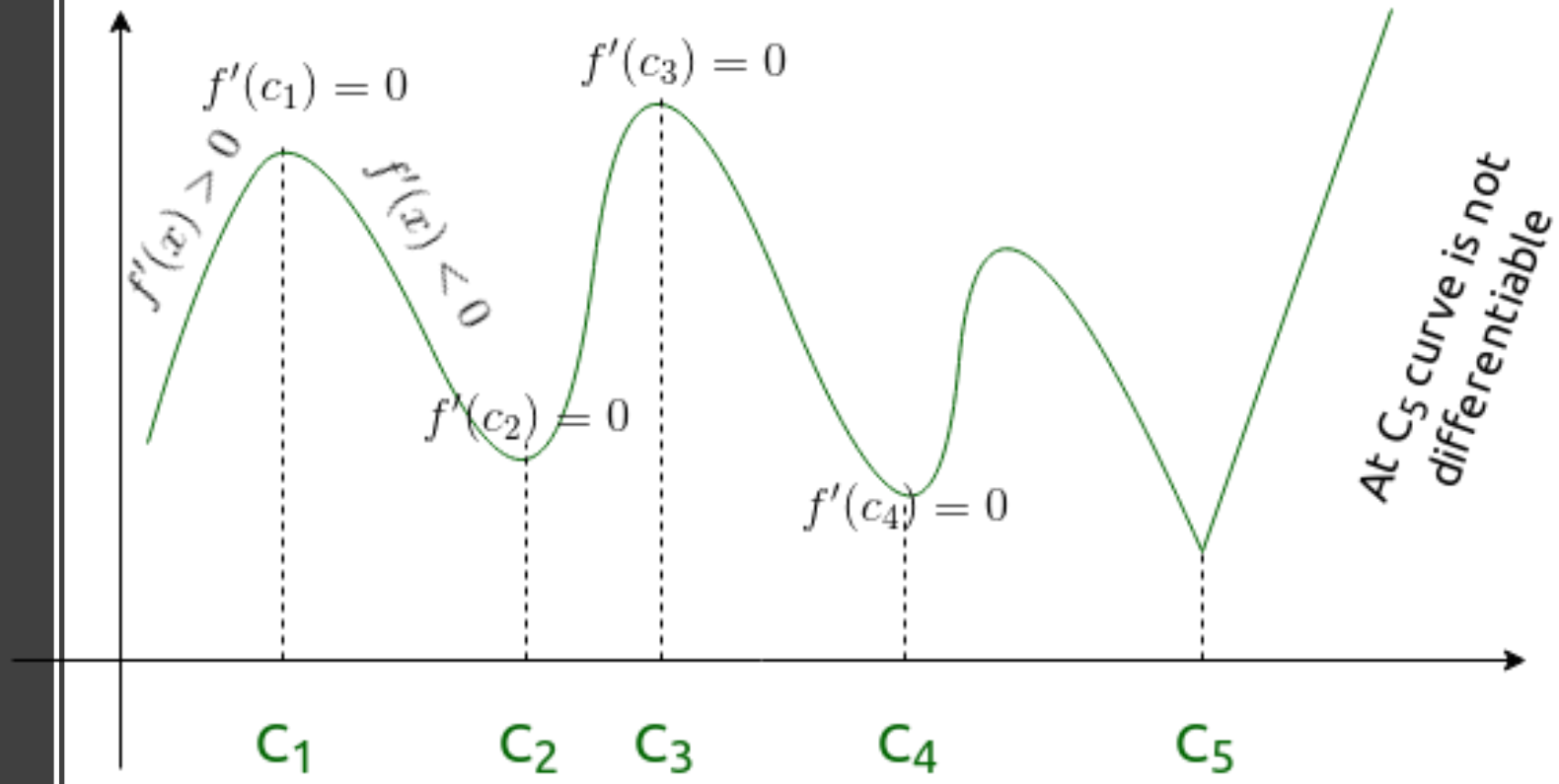
$f(X_0 + h) \leq f(X_0)$

for all $h= (h_1,..., h_j,..., h_n)$ such that $|h|$, is sufficiently small for all j.

$X_0$ is a maximum if the value of f at every point in the neighborhood of $X_0$ does not exceed $f(X_0)$.

In a similar manner, $X_0$ is a minimum if for h as defined

$f(X_0 + h) \geq f(X_0)$



# Stationary Point

1. A necessary condition for $X_0$ to be an extreme point of f(x) is that $\nabla f(X_0)=0$

2. A sufficient condition for a stationary point $X_0$, to be an extremum is for the Hessian matrix H evaluated at $X_0$ to be

(i) Positive definite when $X_0$ is a minimum point.

(ii) Negative definite when $X_0$ is a maximum point.

3. If at a stationary point $y_0$ of f(y), the first (n-1) derivatives vanish and $f^{(n)}(y) \# 0$, then at $y = y_0$, f(y) has

(i) An inflection point if n is odd.

(ii) An extreme point if n is even. This extreme point will be a maximum if $f^{(n)}_{(y0)} <0$ and a minimum if $f^{(n)}_{(y0)} > 0$

# What is Hessian Matrix

- The **Hessian matrix** or **Hessian** is a square matrix of second-order partial derivatives of a scalar-valued function, or scalar field.
- It describes the local curvature of a function of many variables.

$$(\mathbf{H}_f)_{i,j} = \frac{\partial^2 f}{\partial x_i \, \partial x_j}.$$

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \, \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \, \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \, \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \, \partial x_1} & \frac{\partial^2 f}{\partial x_n \, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix},$$

# Example of Stationary Points

Consider the two functions

$$f(y) = y^4 \quad \text{and} \quad g(y) = y^3$$

For $f(y) = y^4$

$$f'(y) = 4y^3 = 0$$

which yields the stationary point $y_0 = 0$. Now

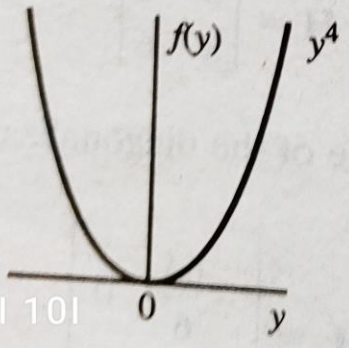
$$f'(0) = f''(0) = f^{(3)}(0) = 0$$

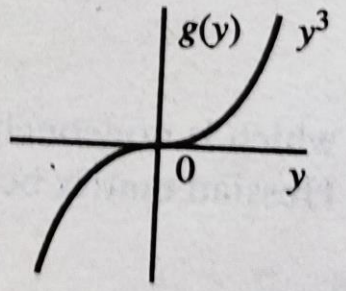
But $f^{(4)}(0) = 24 > 0$; hence, $y_0 = 0$ is a minimum point (see Figure 20-2).

For $g(y) = y^3$,

$$g'(y) = 3y^2 = 0$$

This yields $y_0 = 0$ as a stationary point. Because $g^{(n)}(0)$ is not zero at $n =$ $y_0 = 0$ is an inflection point.



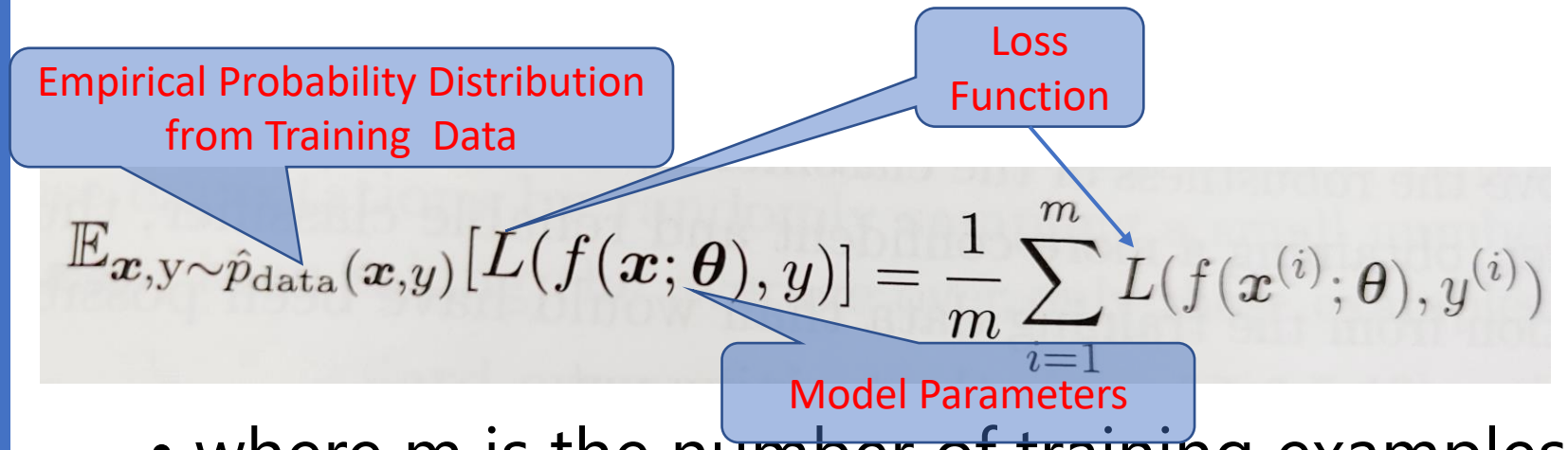
# Classical Optimization vs Machine Learning

Optimization algorithms find maxima or minima of an objective function

ML algorithms also try to minimize loss function between actual and prediction

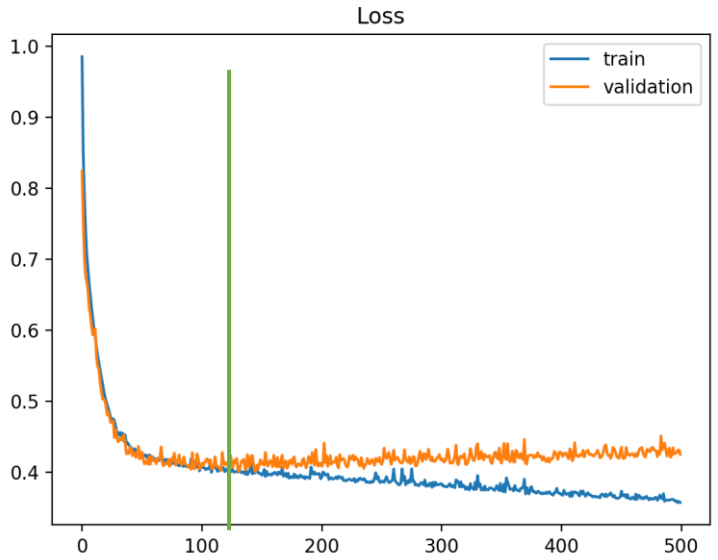
# Classical Optimization vs Machine Learning

- We can convert a machine learning problem into an optimization problem by minimizing the expected loss on the training set.
- This means replacing the true distribution p(x, y) with the empirical distribution p^(x, y) defined by the training set.
- We now minimize the empirical risk




- where m is the number of training examples.

# Differences in Classical Optimization and ML



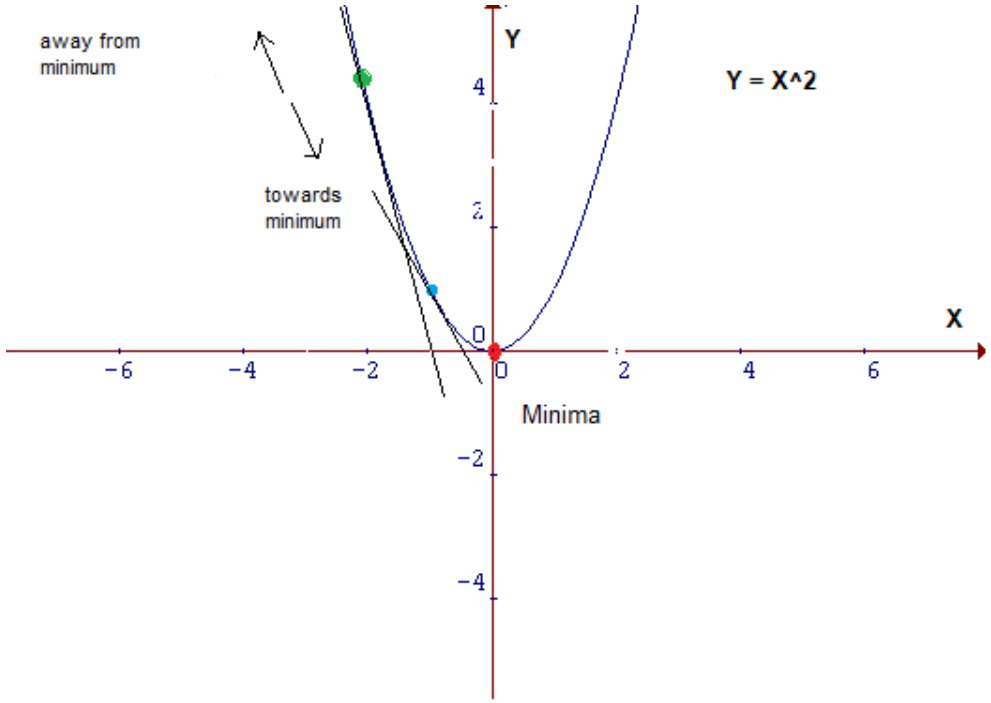

- In optimization problem, we know the objective function while in ML we try to ESTIMATE it from training data set
- We often only reduce the loss function instead of finding minima, which is good enough for practical purposes
- Stopping condition
  - Optimization algorithm stops when gradient becomes very small (may be local minima)
  - ML algorithm stops when overfitting begins to occur (early stopping condition)
- Optimization algorithm works on the objective function but ML algorithm often works on a surrogate loss function (e.g.: Log-Likelihood)

# Gradient Descent

$$\frac{\partial J}{\partial m} = \text{Error} * X * \text{Learning Rate} \qquad \frac{\partial J}{\partial b} = \text{Error} * \text{Learning Rate}$$

$$\text{Since } m = m - \delta m \qquad \text{Since } b = b - \delta b$$

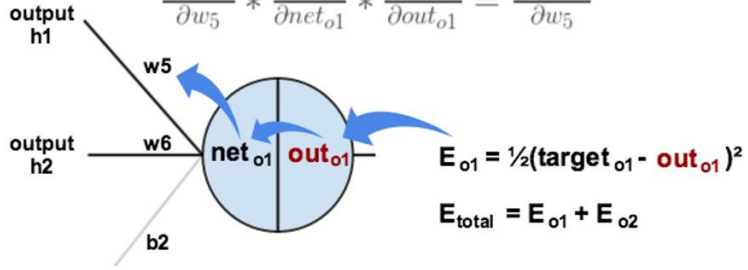
$$m^1 = m^0 - \text{Error} * X * \text{Learning Rate} \qquad b^1 = b^0 - \text{Error} * \text{Learning Rate}$$




## Backward Pass

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$

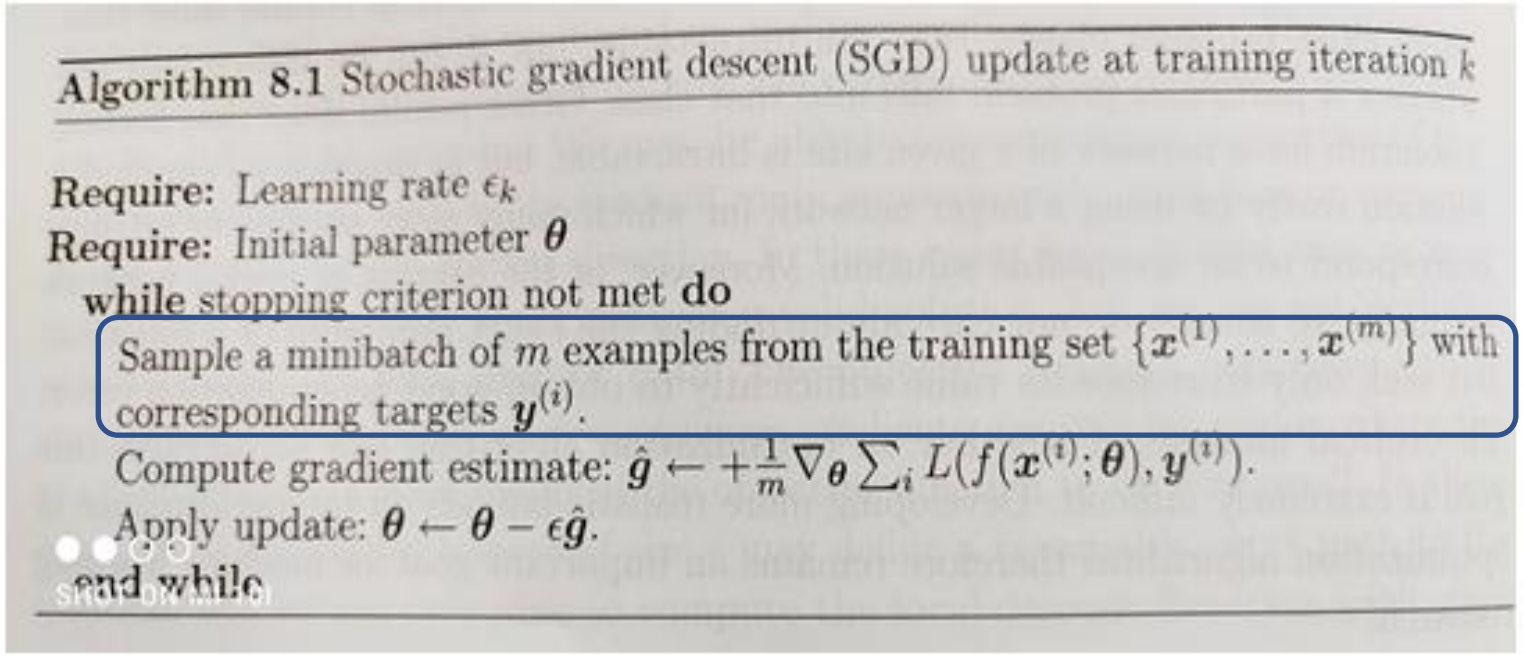



$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

# Stochastic Gradient Descent



**Algorithm 8.1** Stochastic gradient descent (SGD) update at training iteration $k$

**Require:** Learning rate $\epsilon_k$

**Require:** Initial parameter $\boldsymbol{\theta}$

**while** stopping criterion not met **do**

Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$.

**end while**

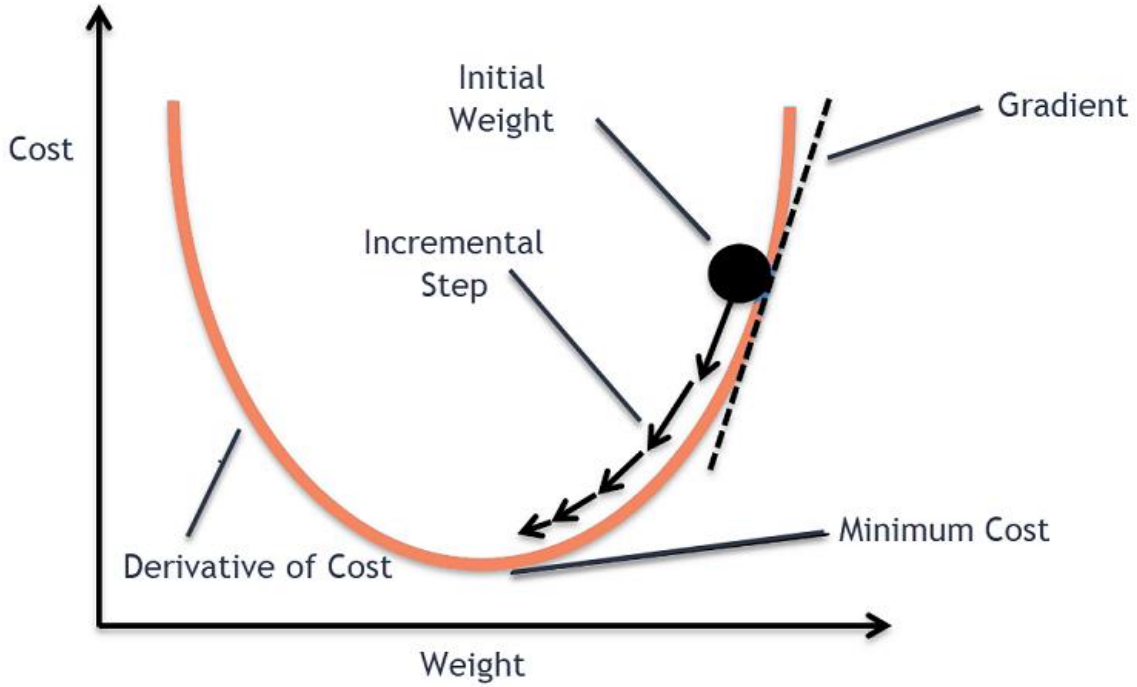
- Take a subset of the full training dataset
  - Batch processing works on the full data set
- It is enough if we can find the right direction of gradient (change in model parameters) from a subset of training example
- For Convergence
  - We require the gradient from the subset of training dataset is an unbiased estimate of true gradient
  - The algorithm makes an empirical estimate of the expected value of gradient

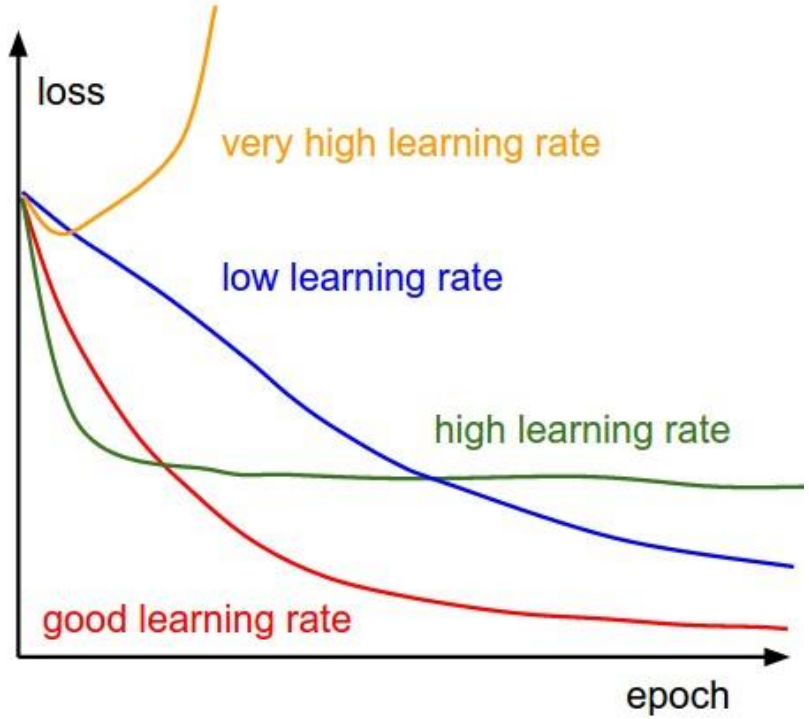
# Learning Rate in Gradient Descent

$$\frac{\partial J}{\partial m} = \text{Error} * X * \text{Learning Rate} \qquad \frac{\partial J}{\partial b} = \text{Error} * \text{Learning Rate}$$

$$Since\ m = m - \delta m \qquad Since\ b = b - \delta b$$

$$m^1 = m^0 - \text{Error} * X * \text{Learning Rate} \qquad b^1 = b^0 - \text{Error} * \text{Learning Rate}$$



$\frac{\partial J}{\partial m}$ = Error * X * Learning Rate

$\frac{\partial J}{\partial b}$ = Error * Learning Rate

Since $m = m - \delta m$

Since $b = b - \delta b$

$m^1 = m^0 -$ Error * X * Learning Rate

$b^1 = b^0 -$ Error * Learning Rate

# Learning Rate in Gradient Descent

- Small Learning Rate slows down convergence
- Large Learning Rate may overshoot minima and can even fail to converge
- How to find best Learning Rate
- Can we Adapt Learning Rate across Iterations
- Can we have different Learning Rates for different Model Parameters



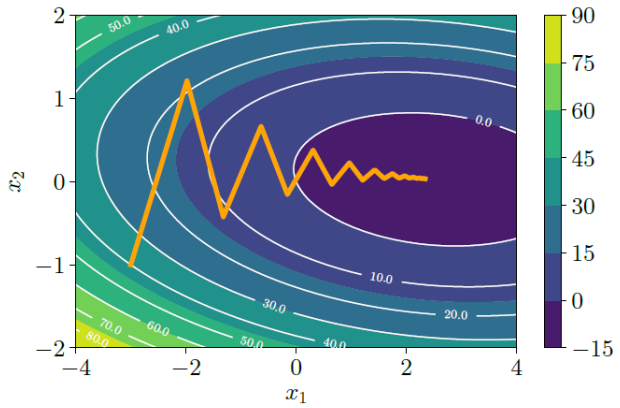
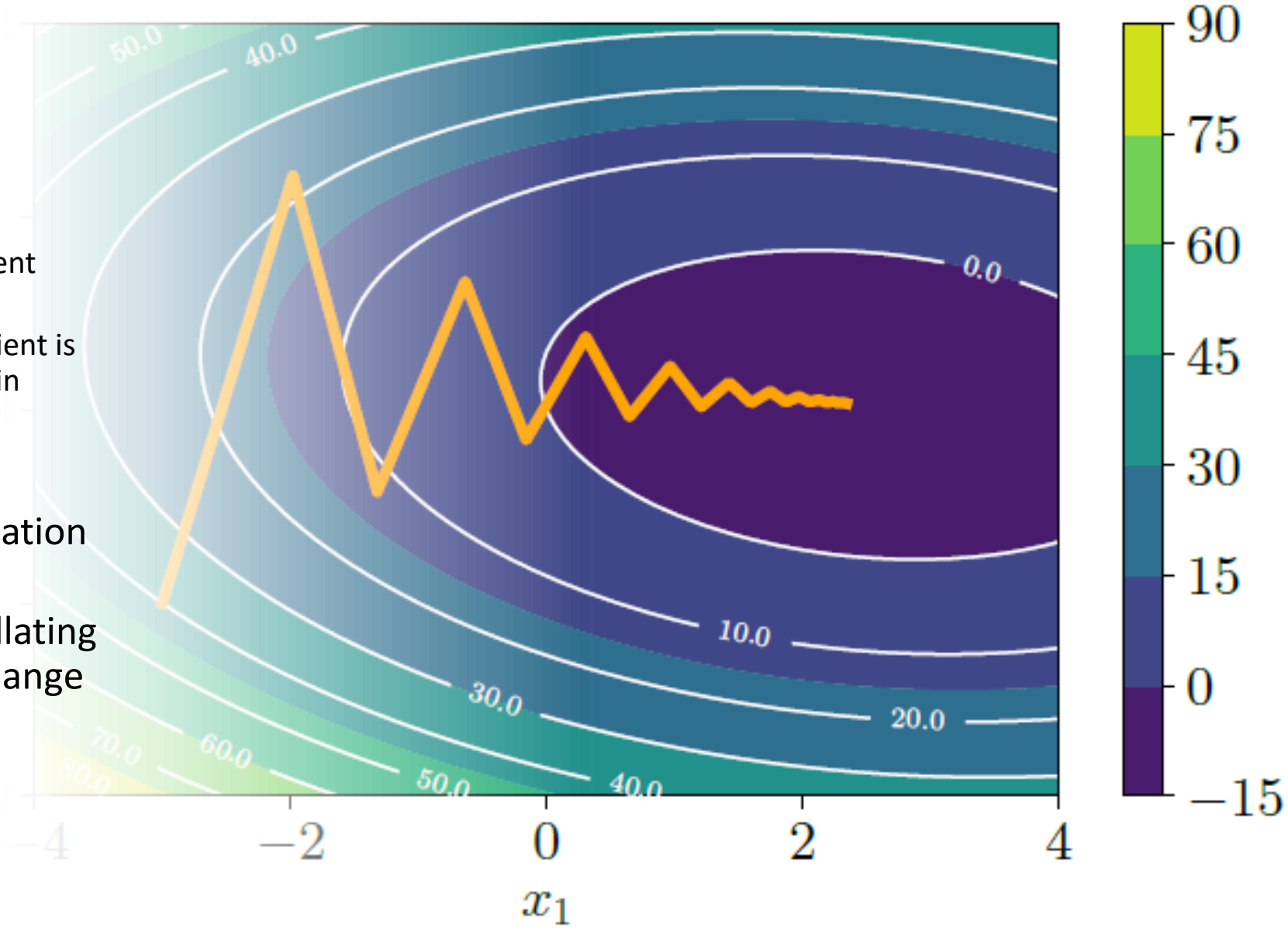



**Figure 7.3** Gradient descent on a two-dimensional quadratic surface (shown as a heatmap). See Example 7.1 for a description.

tion Using Gradient Descent

# Momentum

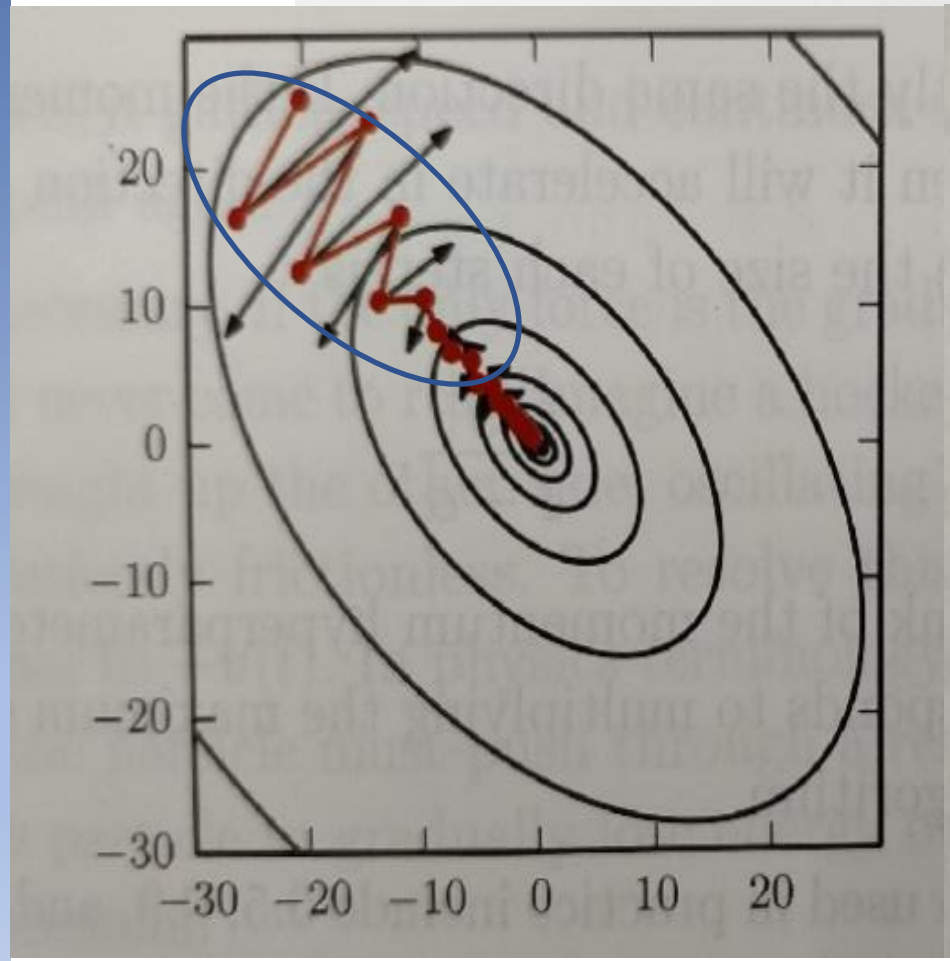
- Remembering past change in gradient
  - Increase Learning Rate if Gradient changes in same direction
  - Decrease Learning Rate if Gradient is changing in opposite direction in subsequent iterations
- Physical Analogy –
  - Damping to and fro oscillation of gradient
  - Adding weight to an oscillating ball resisting direction change

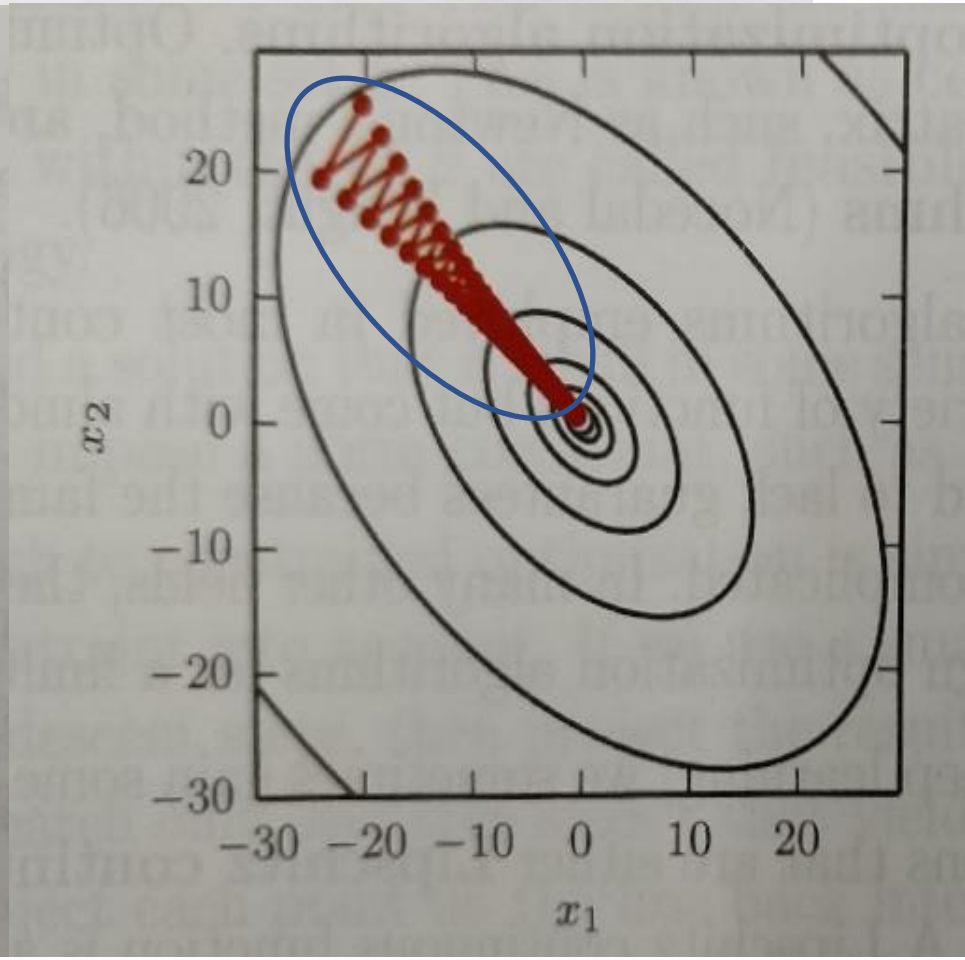


# Momentum

$$v \leftarrow \alpha v - \epsilon \nabla_\theta \left( \frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right),$$

$$\theta \leftarrow \theta + v.$$



# SGD with Momentum

**Algorithm 8.2** Stochastic gradient descent (SGD) with momentum

**Require:** Learning rate $\epsilon$, momentum parameter $\alpha$

**Require:** Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$

**while** stopping criterion not met **do**

Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

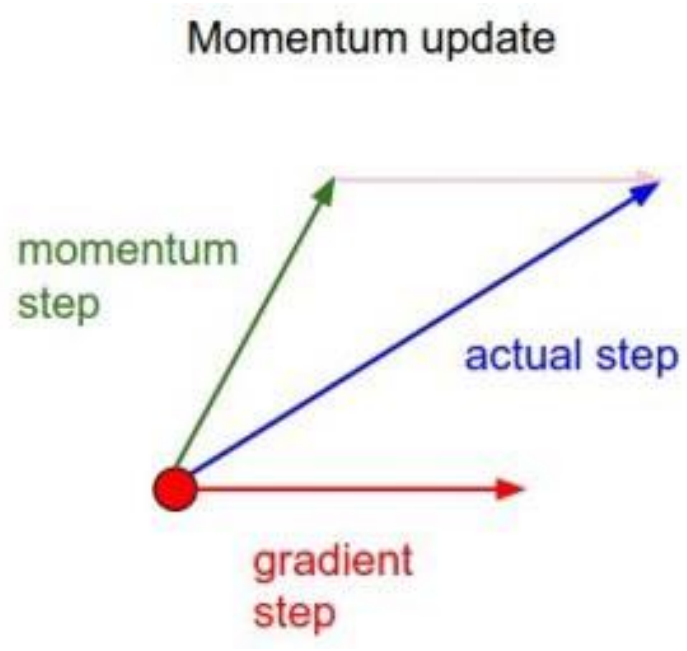
Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.

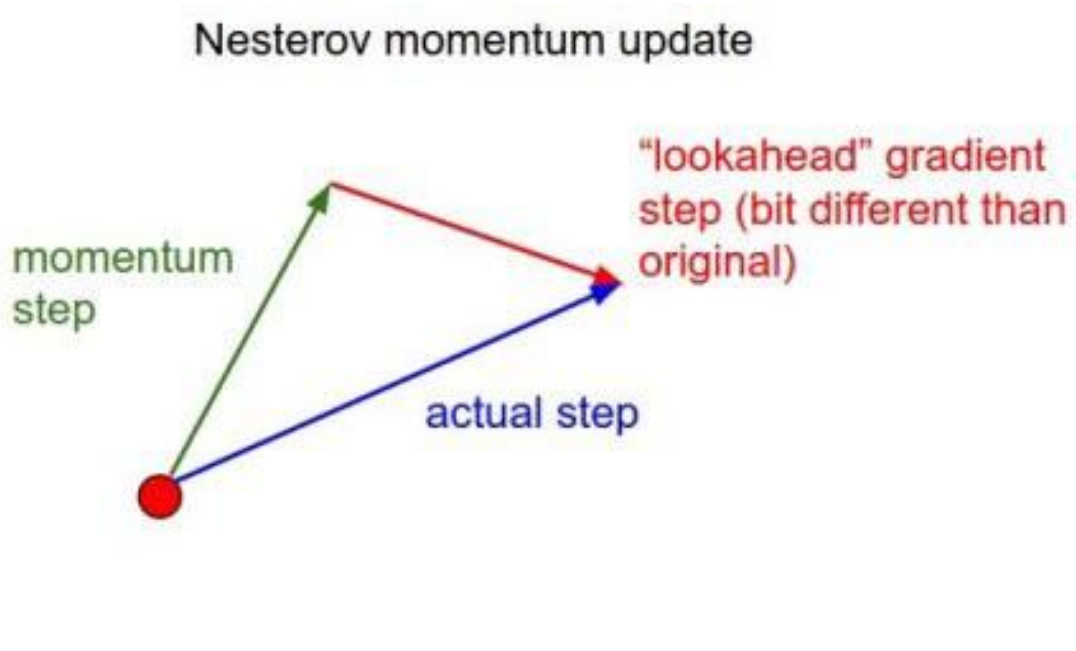
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.

**end while**

# Nesterov Moment Update




$$v \leftarrow \alpha v - \epsilon \nabla_\theta \left( \frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right)$$

$$\theta \leftarrow \theta + v.$$

$$v \leftarrow \alpha v - \epsilon \nabla_\theta \left[ \frac{1}{m} \sum_{i=1}^{m} L\left( \boldsymbol{f}(\boldsymbol{x}^{(i)}; \theta + \alpha v), \boldsymbol{y}^{(i)} \right) \right]$$

$$\theta \leftarrow \theta + v,$$

# Learning Rate in Gradient Descent

- Small Learning Rate slows down convergence
- Large Learning Rate may overshoot minima and can even fail to converge
- How to find best Learning Rate
- Can we Adapt Learning Rate across Iterations
- Can we have different Learning Rates for different Model Parameters

# Adaptive Learning Rate – Basic Algorithm

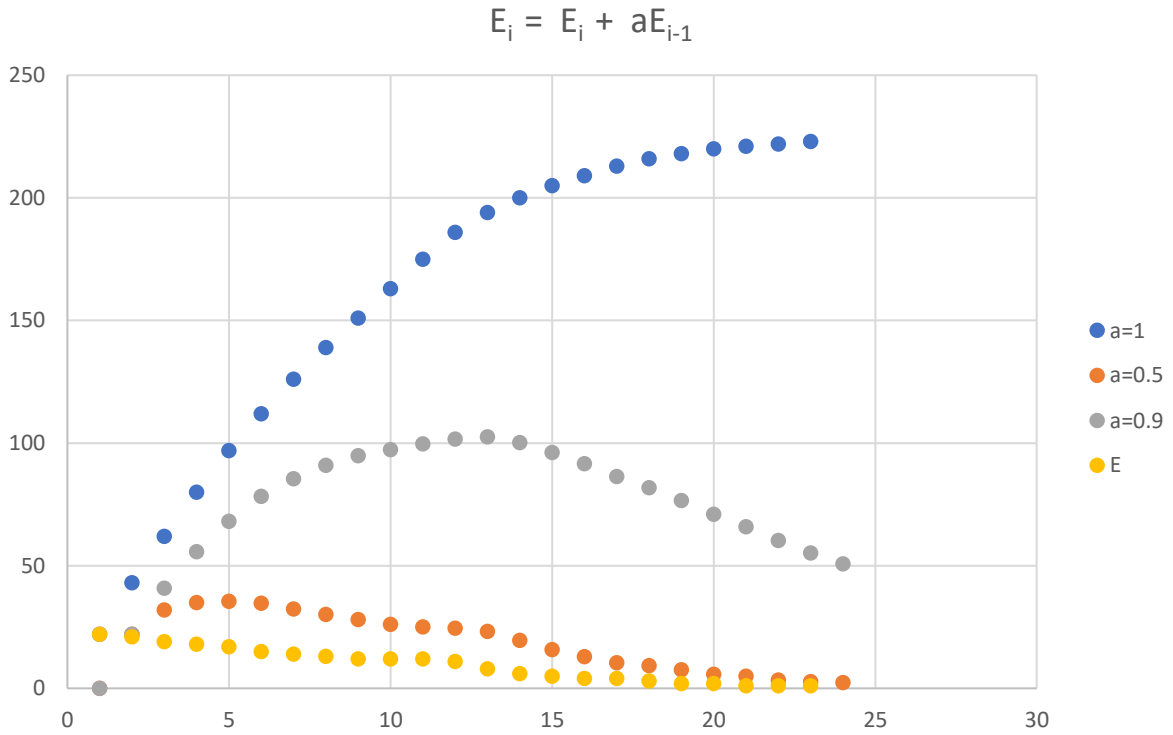
The **delta-bar-delta algorithm** is an early heuristic approach to adapting individual learning rates for model parameters during training.

If the partial derivative of the loss, with respect to a given model parameter, remains the same sign, then the learning rate should increase.

If that partial derivative changes sign, then the learning rate should decrease. Of course, this kind of rule can only be applied to full batch optimization.

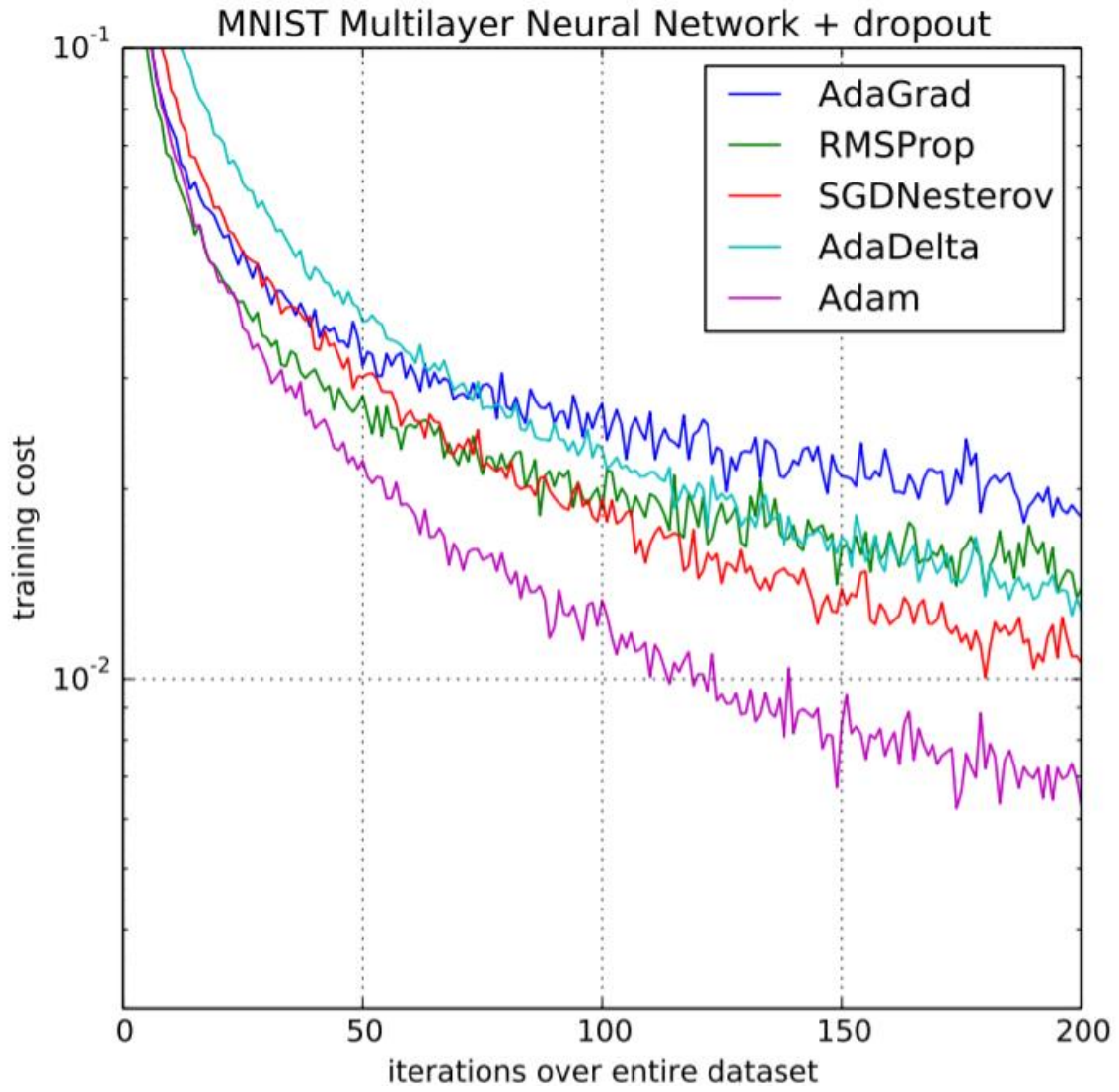
# Adaptive Learning Rate –AdaGrad Algorithm

- The AdaGrad algorithm individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all the historical squared values of the gradient.
- The parameters with the largest partial derivative of the loss have a correspondingly rapid decrease in their learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate.
- The net effect is greater progress in the more gently sloped directions of parameter space.



# Adaptive Learning Rate –Modern Algorithms

- Instead of taking the cumulative sum of squared gradients like in AdaGrad, **RMSProp** takes the exponential moving average.
- Works better than AdaGrad for non-convex functions
- The **Adam Optimizer** inherits the strengths or the positive attributes of the previous two methods and builds upon them to give a more optimized gradient descent.
- Adam uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network.




https://www.geeksforgeeks.org/intuition-of-adam-optimizer/
https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

# Conclusions

Which algorithm should one choose?

There is currently no consensus on this point.

Algorithms with adaptive learning rates (represented by RMSProp and AdaDelta) performed fairly robustly, There is no single best algorithm.

The most popular optimization algorithms actively in use include SGD, SGD with momentum, RMSProp, RMSProp with momentum, AdaDelta, and Adam.

The choice of which algorithm to use, at this point, seems to depend largely on the user's familiarity with the algorithm (for ease of hyperparameter tuning).