# Deep Learning Using TensorFlow

Dr. Ash Pahwa

## Section 3.1: EWMA + Momentum

# Outline

- Problems with Gradient Descent Algorithm
- Moving Average
- Weighted Moving Average
- Exponentially Weighted Moving Average (EWMA)
- Momentum based Gradient Descent Algorithm
- Nestirov Gradient Descent Algorithm
- Reducing the Data Points for Approximate Gradient
  - Stochastic Gradient Descent Algorithm
  - Mini Batch Gradient Descent Algorithm

# What is Gradient Descent Algorithm?

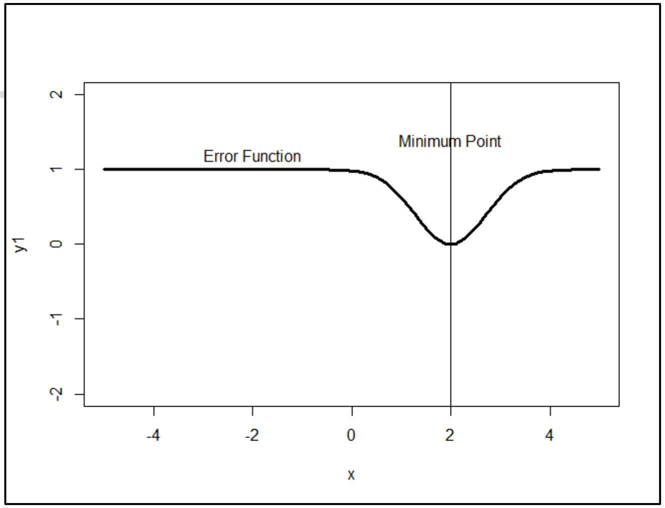
- Suppose a function $y = f(x)$ is given
  - Gradient Descent algorithm allows us to find the values of 'x' where the 'y' value becomes minimum or maximum values
- The Gradient Descent algorithm can be extended to any function with 2 or more variables $'z = f(x, y)'$

- Function y=f(x)
- Gradient Descent Algorithm: Minimum
  - Initialize the value of x
  - Learning rate = η
  - While NOT converged:
    - $x^{t+1} \leftarrow x^t - \eta \frac{\partial y}{\partial x} \|_{x^t}$

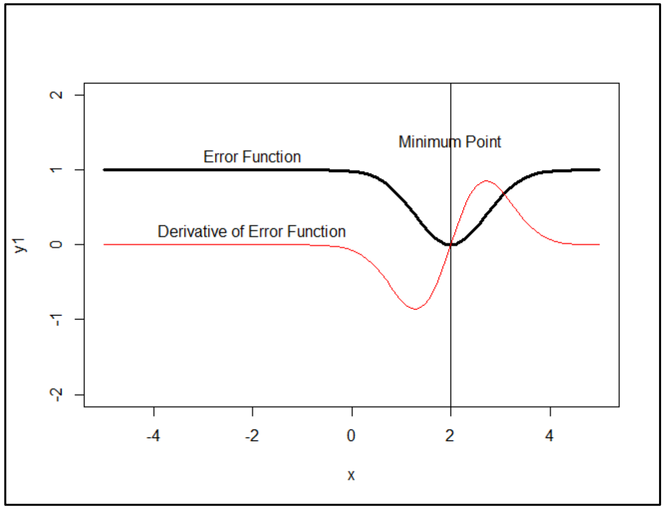
- Function z=f(x,y)
- Gradient Descent Algorithm: Minimum
  - Initialize the value of x and y
  - Learning rate = η
  - While NOT converged:
    - $x^{t+1} \leftarrow x^t - \eta \frac{\partial z}{\partial x} \|_{x^t, y^t}$
    - $y^{t+1} \leftarrow y^t - \eta \frac{\partial z}{\partial y} \|_{x^t, y^t}$

# Example#1
# Error Function

- $Error\ Function:\ y = f(x) = 1 - e^{-(x-2)^2}$
- $\frac{dy}{dx} = 2(x-2)e^{-(x-2)^2}$
- The error function 'y' is almost flat
  - $f(x) = 0{:}\ -\infty < x < 0\ and$
  - $f(x) = 0{:}\quad 4 < x < \infty$

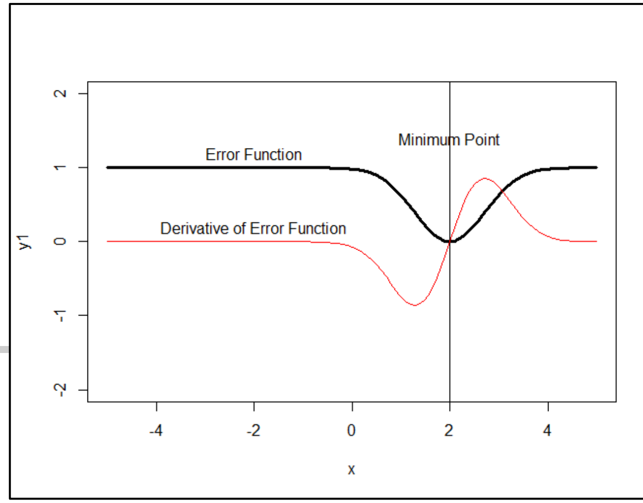



```
> x = seq(-5,5,0.1)
> y1 = 1 - exp(-(x-2)^2)
> plot(x,y1,type='l',ylim=c(-2,2),lwd=3)
> text(-2,1.2,"Error Function")
> text(2,1.4,"Minimum Point")
> abline(v=2)
> dy_dx = function (w1) { 2*(w1-2)*exp(-(w1-2)^2) }
> slope = dy_dx(x)
> points(x,slope,type='l',col='red')
> text(-2,0.2,"Derivative of Error Function")
```

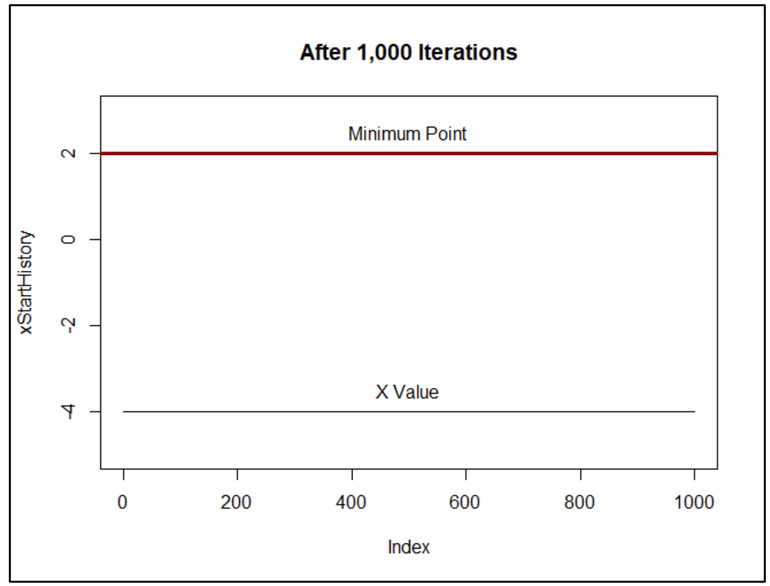


# Example#1
# Gradient Descent Algorithm

- Starting Point = -4
- Number of iteration = 1,000

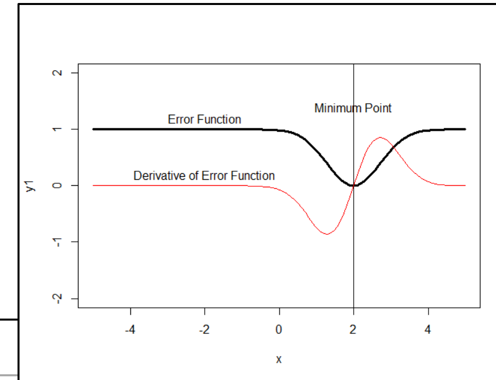


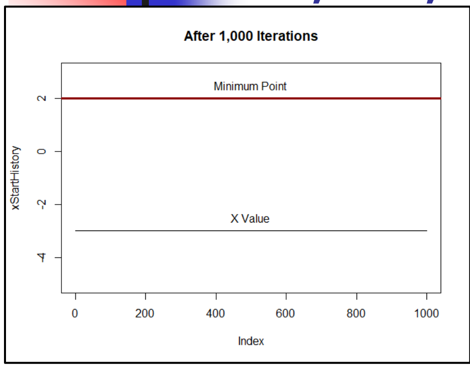

```
> xStart = -4
> learningRate = 0.1
> maxLimit = 1000
> xStartHistory = rep(0,maxLimit)
> for ( i in 1:maxLimit )
+ {
+   xStartHistory[i] = xStart
+   xStart = xStart - learningRate * dy_dx(xStart)
+ }
> plot(xStartHistory,type='l',ylim=c(-5,3),
main="After 1,000 Iterations")
> text(500,-3.5,"X Value")
> abline(h=2, col='dark red', lwd=3)
> text(500,2.5,"Minimum Point")
>
```

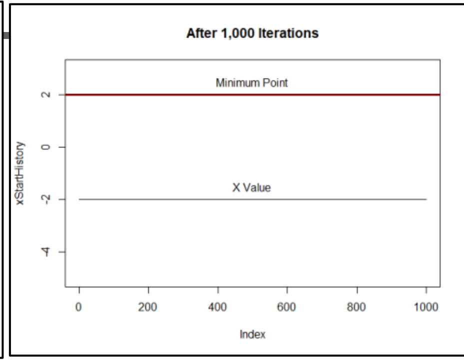


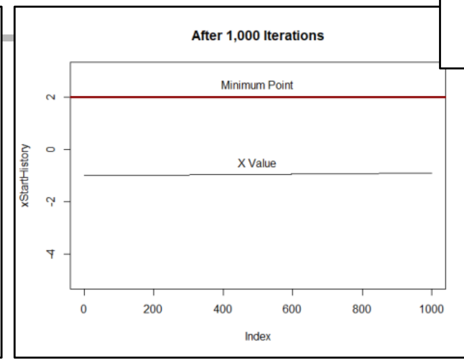

Starting Point = -4, DOES NOT CONVERGE

# Example#1: After 1,000 iteration Starting Point -3, -2, -1, -0.5

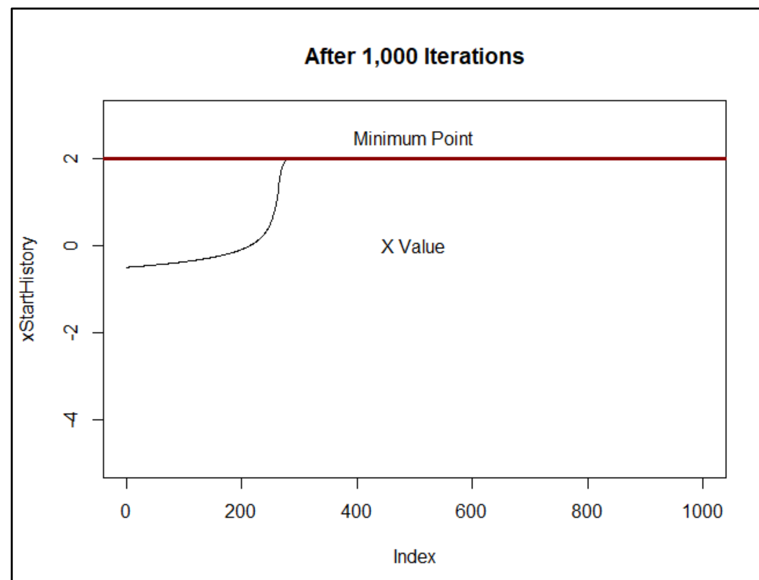



Starting Point = -3, -2, -1: 'x' value DOES NOT CONVERGE



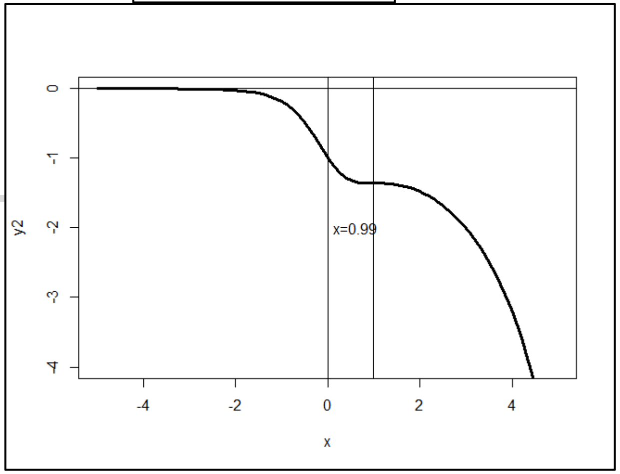

Starting Point = -0.5,

'x' value CONVERGES AFTER 250 ITERATIONS

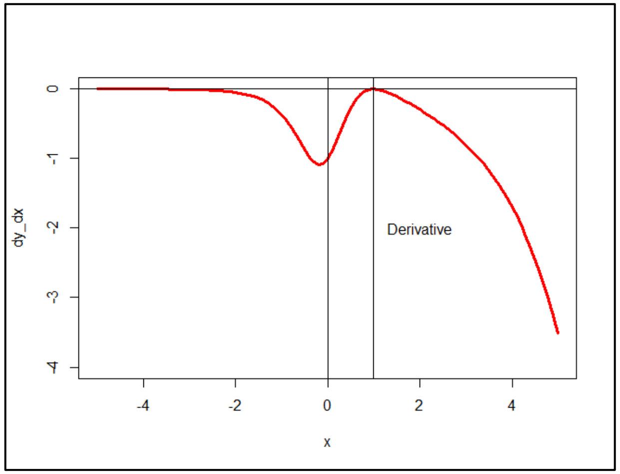
# Example#2
# Error Function

- $Error\ Function = y = -\frac{e^x}{1+x^2}$
- $\frac{dy}{dx} = \frac{(1+x^2)\frac{d}{dx}(e^x) - e^x\frac{d}{dx}(1+x^2)}{(1+x^2)^2}$
- $\frac{dy}{dx} = -\frac{e^x(1-x)^2}{(1+x^2)^2}$
- ------------------------------------------
- Gradient Descent Algorithm
  - Starting point: x = 0
  - Till x < 0.99
    - Descend smoothly using GD
  - After x > 0.99 (as x → 1)
    - Cannot move further because the line is flat
- Solution
  - Exponentially Weighted Moving Average

$$y = -\frac{e^x}{1+x^2}$$

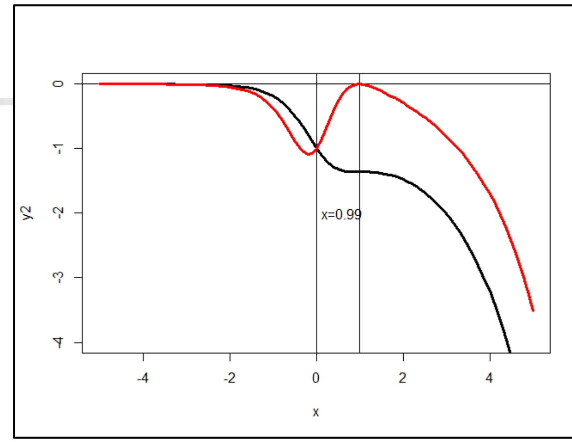


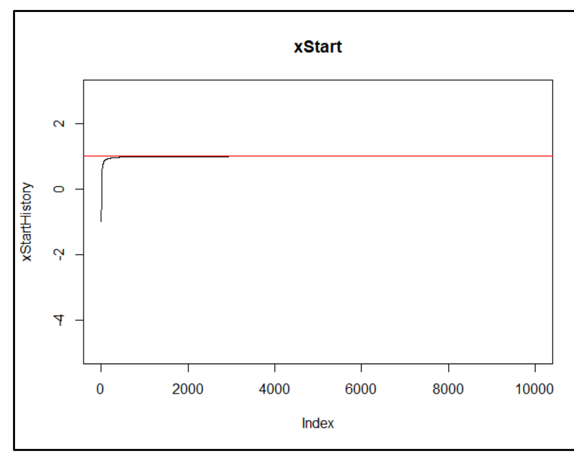

$$\frac{dy}{dx} = -\frac{e^x(1-x)^2}{(1+x^2)^2}$$



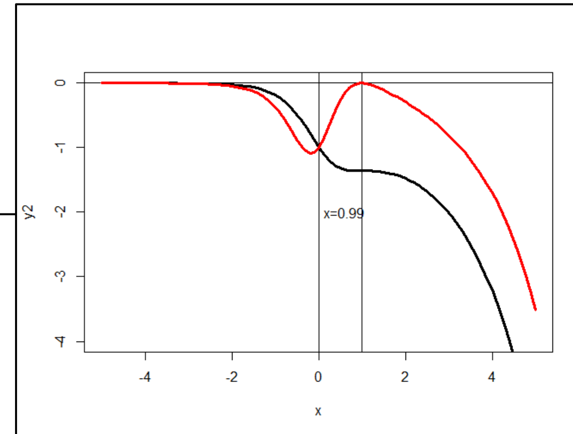
# Example#2
# After 10,000 Iterations
# We cannot move beyond 1

```
> xStart = -1.0
> learningRate = 0.1
> maxLimit = 10000
> xStartHistory = rep(0,maxLimit)
> for ( i in 1:maxLimit )
+ {
+   xStartHistory[i] = xStart
+   xStart = xStart - learningRate * dy_dx(xStart)
+
+   if ( i %% 1000 == 0 ) { cat ("i = ",i, "xstart = ",
xStart, "\n")}
+ }
i =  1000 xstart =  0.9861112
i =  2000 xstart =  0.9928911
i =  3000 xstart =  0.9952164
i =  4000 xstart =  0.9963939
i =  5000 xstart =  0.9971056
i =  6000 xstart =  0.9975825
i =  7000 xstart =  0.9979243
i =  8000 xstart =  0.9981814
i =  9000 xstart =  0.9983817
i =  10000 xstart =  0.9985423
> plot(xStartHistory,type='l',ylim=c(-5,3),
main="xStart")
> abline(h=1,col='red')
>
```

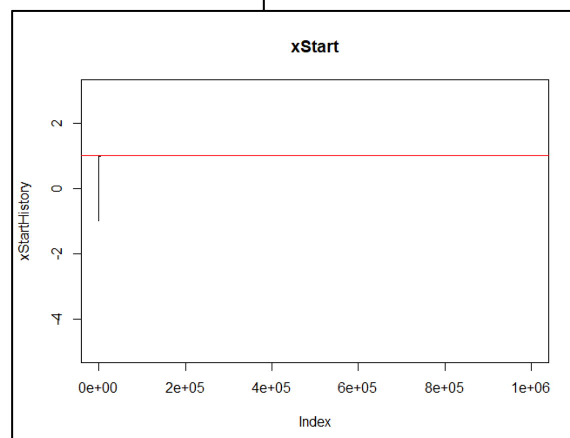


# Example#2
# After 1,000,000 Iterations
# We cannot move beyond 1




```
> ##########################################
> xStart = -1.0
> learningRate = 0.1
> maxLimit = 1000000
> xStartHistory = rep(0,maxLimit)
>
> for ( i in 1:maxLimit )
+ {
+   xStartHistory[i] = xStart
+   xStart = xStart - learningRate * dy_dx(xStart)
+
+   if ( i %% 100000 == 0 ) { cat ("i = ",i, "xstart = ", xStart, "\n")}
+ }
i =  100000 xstart =  0.999853
i =  200000 xstart =  0.9999265
i =  300000 xstart =  0.999951
i =  400000 xstart =  0.9999632
i =  500000 xstart =  0.9999706
i =  600000 xstart =  0.9999755
i =  700000 xstart =  0.999979
i =  800000 xstart =  0.9999816
i =  900000 xstart =  0.9999837
i =  1000000 xstart =  0.9999853
>
> plot(xStartHistory,type='l',ylim=c(-5,3), main="xStart")
> abline(h=1,col='red')
```



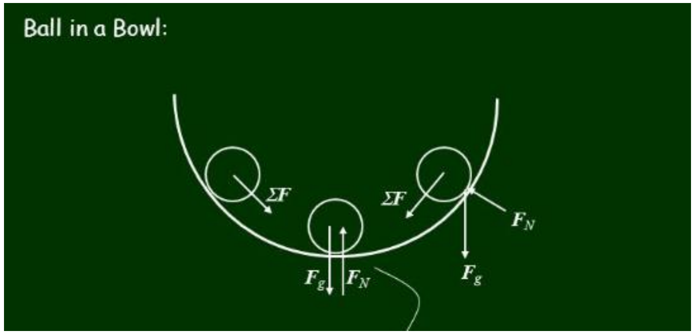
# Problems with Gradient Descent Algorithm

- If the error function has a very small gradient value
  - Starting point is on the flat surface
  - Neural network will take a long time to converge

# First Solution to the Gradient Descent Algorithm Problem

- When the gradient becomes zero
  - Find some way to move forward
- Step size is not only a function of current gradient at time 't'
  - But also previous gradient at time 't-1'
- Solution
  - Momentum
  - Exponentially Weighted Moving Average of Gradient
  - $x_{t+1} = x_t - \eta \frac{\partial z}{\partial x} - \left( previous\ values\ of\ \frac{\partial z}{\partial x} \right)$

# Momentum Based GD




Ball gains momentum while rolling down a slope

- Time t1
  - Ask for direction (Measure Gradient)
  - Take a small step in that direction
- Time t2
  - Ask for direction (Measure Gradient)
  - If the direction computed ay time 't2' is same as the direction at time 't1'
  - Take a bigger step in that direction
- Time t3
  - Ask for direction (Measure Gradient)
  - If the direction computed ay time 't3' is same as the direction at time 't1' and 't2'
  - Take a bigger step in that direction

- Momentum is building as we are moving along
- Speed at which we are moving will increase with time

# Other Solutions to Gradient Descent Algorithm Problem

- Solution#1: Increase the step size
  - Momentum based GD
  - Nesterov GD
- Solution#2: Reducing the data points for approximate gradient
  - Stochastic Gradient Descent
  - Mini Batch Gradient Descent
- Solution#3: Adjust the Learning rate $(\eta)$
  - AdaGrad
  - RMSProp
  - Adam

- Function y=f(x)
- Gradient Descent Algorithm: Minimum
  - Initialize the value of x
  - Learning rate = η
  - While NOT converged:
    - $x^{t+1} \leftarrow x^t - \eta \frac{\partial y}{\partial x} \|_{x^t}$

# Moving Average

Traditional

## * Moving Average
## * Exponentially Weighted Moving Average (EWMA)

- When the data fluctuates
  - We need to compute the average to smooth the data
- Solution
  - Moving Average
  - Weighted Moving Average
  - Exponentially Weighted Moving Average

| | A | B |
|---|---|---|
| 1 | Step# | Derivative |
| 2 | 1 | 0.8550 |
| 3 | 2 | 0.8470 |
| 4 | 3 | 1.0000 |
| 5 | 4 | 0.6350 |
| 6 | 5 | 0.3460 |
| 7 | 6 | 2.1460 |
| 8 | 7 | 1.3280 |
| 9 | 8 | 1.3220 |
| 10 | 9 | 3.1240 |
| 11 | 10 | 1.0120 |
| 12 | 11 | 1.2800 |
| 13 | 12 | 2.4350 |
| 14 | 13 | 1.0160 |
| 15 | 14 | 3.4650 |
| 16 | 15 | 1.1070 |
| 17 | 16 | 1.1720 |
| 18 | 17 | 3.4320 |
| 19 | 18 | 0.8360 |
| 20 | 19 | 0.1420 |
| 21 | 20 | 0.3450 |
| 22 | 21 | 2.6030 |
| 23 | 22 | 0.7390 |
| 24 | 23 | 0.7160 |
| 25 | 24 | 0.8800 |
| 26 | 25 | 1.0080 |
| 27 | 26 | 0.1120 |
| 28 | 27 | 0.3610 |

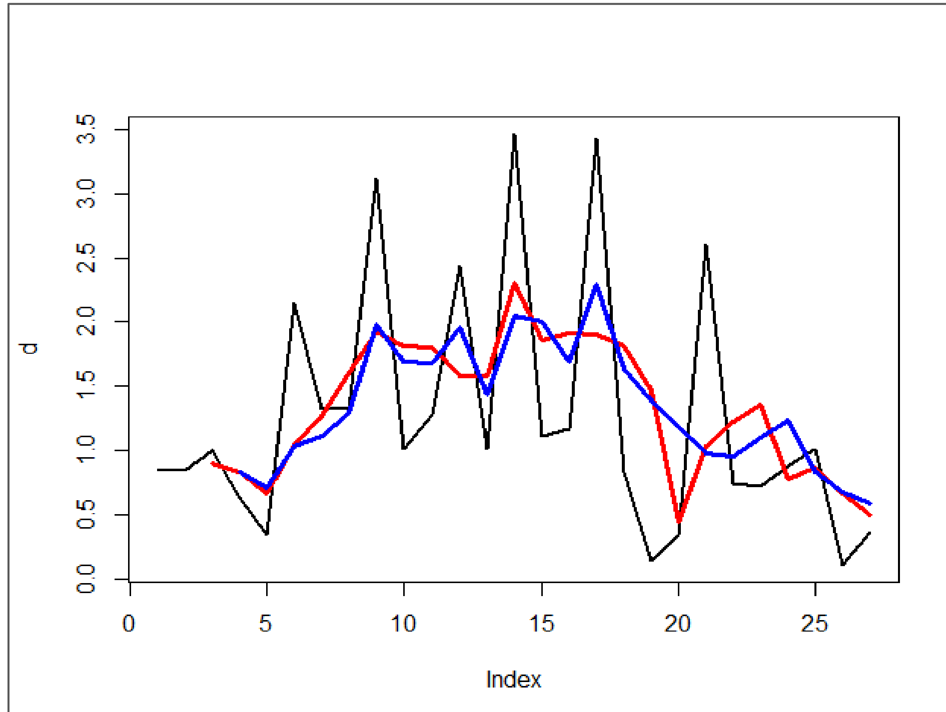
```
> rawData = read.csv("01 RawData+3 StepMA+4 StepMA.csv")
> (d = rawData$Derivative)
 [1] 0.855 0.847 1.000 0.635 0.346 2.146 1.328 1.322 3.124 1.012 1.280
[12] 2.435 1.016 3.465 1.107 1.172 3.432 0.836 0.142 0.345 2.603 0.739
[23] 0.716 0.880 1.008 0.112 0.361
> length(d)
[1] 27
> plot(d,type='l',lwd=2)
>
> ########################################################
> # 3-Step Moving Averge
> #
> (ma.3Step = rollmean(d,3,na.pad = TRUE,align="right"))
 [1]        NA        NA 0.9006667 0.8273333 0.6603333 1.0423333
 [7] 1.2733333 1.5986667 1.9246667 1.8193333 1.8053333 1.5756667
[13] 1.5770000 2.3053333 1.8626667 1.9146667 1.9036667 1.8133333
[19] 1.4700000 0.4410000 1.0300000 1.2290000 1.3526667 0.7783333
[25] 0.8680000 0.6666667 0.4936667
> length(ma.3Step)
[1] 27
> points(ma.3Step,type='l',col='red',lwd=3)
>
> ########################################################
> # 4-Step Moving Averge
> #
> (ma.4Step = rollmean(d,4,na.pad = TRUE,align="right"))
 [1]      NA      NA      NA 0.83425 0.70700 1.03175 1.11375 1.28550
 [9] 1.98000 1.69650 1.68450 1.96275 1.43575 2.04900 2.00575 1.69000
[17] 2.29400 1.63675 1.39550 1.18875 0.98150 0.95725 1.10075 1.23450
[25] 0.83575 0.67900 0.59025
> length(ma.4Step)
[1] 27
> points(ma.4Step,type='l',col='blue',lwd=3)
```

# 3-Step & 4-Step Moving Average

- 4 Step Moving average is smoother than 3-Step Moving Average

- Small 'n'-Step Moving average
  - Driving a small car
  - Fast and picks up speed faster
  - Not very smooth
- Large 'n'-Step Moving average
  - Driving a 18-wheeler Truck
  - Smooth ride
  - Sluggish – slow to pick up speed

```
> #########################################
> # Plot all the data
> #
> plot(d,type='l',lwd=2)
> points(ma.3Step,type='l',col='red',lwd=3)
> points(ma.4Step,type='l',col='blue',lwd=3)
```



# Weighted Moving Average

Moving Average

+

Weighted Average

# Weighted Moving Average

- More recent data has more weight

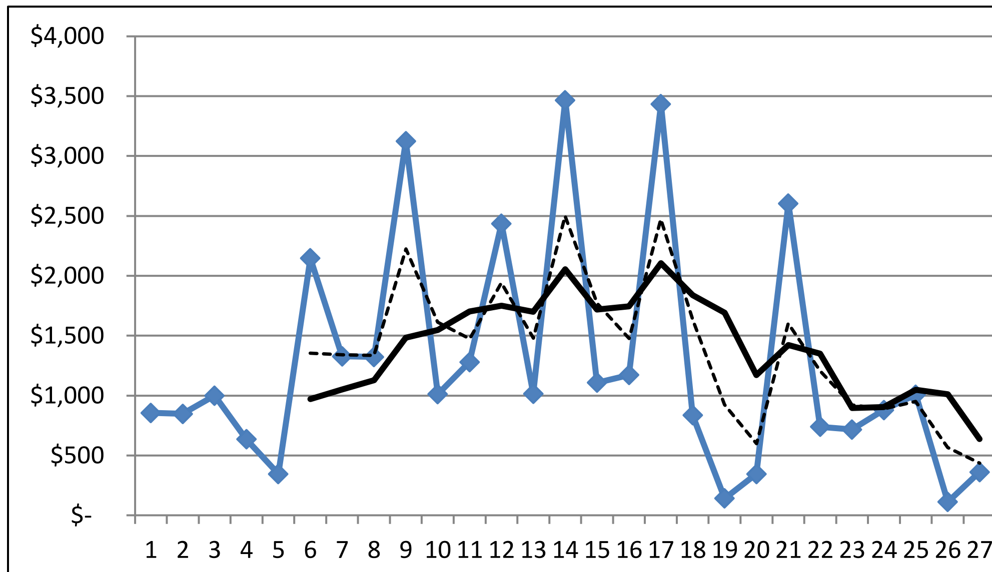
| | A | B | C |
|---|---|---|---|
| 1 | Derivative | Weight | Weighted Derivative |
| 2 | 0.8550 | 0.1875 | 0.1603 |
| 3 | 0.8470 | 0.1875 | 0.1588 |
| 4 | 1.0000 | 0.3750 | 0.3750 |
| 5 | 0.6350 | 0.7500 | 0.4763 |
| 6 | 0.3460 | 1.5000 | 0.5190 |
| 7 | 2.1460 | 3.0000 | 6.4380 |
| 8 | | | |
| 9 | | Sum: | 8.1274 |
| 10 | | Weighted average: | 1.3546 |
| 11 | | Unweighted average: | 0.9715 |
| 12 | | | |

| | A | B | C | D |
|---|---|---|---|---|
| 1 | | Weight | Weight as percentage | |
| 2 | | 0.1875 | 3% | |
| 3 | | 0.1875 | 3% | |
| 4 | | 0.3750 | 6% | |
| 5 | | 0.7500 | 13% | |
| 6 | | 1.5000 | 25% | |
| 7 | | 3.0000 | 50% | |
| 8 | | | | |
| 9 | Total | 6.0000 | 100% | |
| 10 | | | | |

# 6-Step MA vs 6-Step Weighted MA

| Step | Derivative | 6-Step MA | Weights | 6-Step Weighted MA |
|---|---|---|---|---|
| 1 | 0.8550 | | 0.1875 | |
| 2 | 0.8470 | | 0.1875 | |
| 3 | 1.0000 | | 0.3750 | |
| 4 | 0.6350 | | 0.7500 | |
| 5 | 0.3460 | | 1.5000 | |
| 6 | 2.1460 | 0.9715 | 3.0000 | 1.3546 |
| 7 | 1.3280 | 1.0503 | | 1.3412 |
| 8 | 1.3220 | 1.1295 | | 1.3340 |
| 9 | 3.1240 | 1.4835 | | 2.2233 |
| 10 | 1.0120 | 1.5463 | | 1.6131 |
| 11 | 1.2800 | 1.7020 | | 1.4747 |
| 12 | 2.4350 | 1.7502 | | 1.9421 |
| 13 | 1.0160 | 1.6982 | | 1.4789 |
| 14 | 3.4650 | 2.0553 | | 2.5001 |
| 15 | 1.1070 | 1.7192 | | 1.7706 |
| 16 | 1.1720 | 1.7458 | | 1.4755 |
| 17 | 3.4320 | 2.1045 | | 2.4718 |
| 18 | 0.8360 | 1.8380 | | 1.6317 |
| 19 | 0.1420 | 1.6923 | | 0.9251 |
| 20 | 0.3450 | 1.1723 | | 0.5982 |
| 21 | 2.6030 | 1.4217 | | 1.6016 |
| 22 | 0.7390 | 1.3495 | | 1.2056 |
| 23 | 0.7160 | 0.8968 | | 0.9203 |
| 24 | 0.8800 | 0.9042 | | 0.8893 |
| 25 | 1.0080 | 1.0485 | | 0.9518 |
| 26 | 0.1120 | 1.0097 | | 0.5672 |
| 27 | 0.3610 | 0.6360 | | 0.4350 |

| Line Color | Strategy | Conclusion |
|---|---|---|
| Blue Line | Original Derivative | Noisy |
| Black Solid Line | 6-Step Moving Average | Smother |
| Black Dotted Line | 6-Step Weighted Moving Average | Smoother than 6-day moving average |



# Moving Weighted Average

- How to decide the length of moving average?
- How to decide the weights?
- These questions are answered by Exponentially Weighted Moving Average (EWMV)

# Exponentially Weighted Moving Averages (EWMA)

# Exponentially Weighted Moving Average of Gradients

- $update_t = \gamma * update_{t-1} + \eta \frac{\partial z}{\partial x} ||_t$
- $x_{t+1} = x_t - update_t$
- ------------------------------------------------------------
- $update_0 = 0$
- $update_1 = \gamma * update_0 + \eta \frac{\partial z}{\partial x} |_1$
- $update_2 = \gamma * update_1 + \eta \frac{\partial z}{\partial x} |_2 = (\gamma^2 * update_0) + \left(\gamma * \eta \frac{\partial z}{\partial x} |_1\right) + \left(\eta \frac{\partial z}{\partial x} |_2\right)$
- $update_3 = \gamma * update_2 + \eta \frac{\partial z}{\partial x} |_3 = (\gamma^3 * update_0) + \left(\gamma^2 * \eta \frac{\partial z}{\partial x} |_1\right) + \left(\gamma * \eta \frac{\partial z}{\partial x} |_2\right) + \left(\eta \frac{\partial z}{\partial x} |_3\right)$
- $update_4 = \gamma * update_3 + \eta \frac{\partial z}{\partial x} |_4 = (\gamma^4 * update_0) + \left(\gamma^3 * \eta \frac{\partial z}{\partial x} |_1\right) + \left(\gamma^2 * \eta \frac{\partial z}{\partial x} |_2\right) + \left(\gamma * \eta \frac{\partial z}{\partial x} |_3\right) + \left(\eta \frac{\partial z}{\partial x} |_4\right)$
- $\vdots$
- $update_t = \gamma * update_{t-1} + \eta \frac{\partial z}{\partial x} |_t = (\gamma^t * update_0) + \left(\gamma^{t-1} * \eta \frac{\partial z}{\partial x} |_1\right) + \left(\gamma^{t-2} * \eta \frac{\partial z}{\partial x} |_2\right) + \cdots + \left(\eta \frac{\partial z}{\partial x} |_t\right)$
- $update_t = \gamma * update_{t-1} + \eta \frac{\partial z}{\partial x} |_t = \left(\gamma^{t-1} * \eta \frac{\partial z}{\partial x} |_1\right) + \left(\gamma^{t-2} * \eta \frac{\partial z}{\partial x} |_2\right) + \cdots + \left(\eta \frac{\partial z}{\partial x} |_t\right)$

# Contributions Made by Previous Gradient

$\gamma^n$

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | |
| 2 | **Gamma** | **0.9** | **0.8** | **0.7** | **0.6** | **0.5** | **0.4** | **0.3** | **0.2** | **0.1** |
| 3 | 1 | 0.9000 | 0.8000 | 0.7000 | 0.6000 | 0.5000 | 0.4000 | 0.3000 | 0.2000 | 0.1000 |
| 4 | 2 | 0.8100 | 0.6400 | 0.4900 | 0.3600 | 0.2500 | 0.1600 | 0.0900 | 0.0400 | 0.0100 |
| 5 | 3 | 0.7290 | 0.5120 | 0.3430 | 0.2160 | 0.1250 | 0.0640 | 0.0270 | 0.0080 | 0.0010 |
| 6 | 4 | 0.6561 | 0.4096 | 0.2401 | 0.1296 | 0.0625 | 0.0256 | 0.0081 | 0.0016 | 0.0001 |
| 7 | 5 | 0.5905 | 0.3277 | 0.1681 | 0.0778 | 0.0313 | 0.0102 | 0.0024 | 0.0003 | 0.0000 |
| 8 | 6 | 0.5314 | 0.2621 | 0.1176 | 0.0467 | 0.0156 | 0.0041 | 0.0007 | 0.0001 | 0.0000 |
| 9 | 7 | 0.4783 | 0.2097 | 0.0824 | 0.0280 | 0.0078 | 0.0016 | 0.0002 | 0.0000 | 0.0000 |
| 10 | 8 | 0.4305 | 0.1678 | 0.0576 | 0.0168 | 0.0039 | 0.0007 | 0.0001 | 0.0000 | 0.0000 |
| 11 | 9 | 0.3874 | 0.1342 | 0.0404 | 0.0101 | 0.0020 | 0.0003 | 0.0000 | 0.0000 | 0.0000 |
| 12 | 10 | 0.3487 | 0.1074 | 0.0282 | 0.0060 | 0.0010 | 0.0001 | 0.0000 | 0.0000 | 0.0000 |
| 13 | 11 | 0.3138 | 0.0859 | 0.0198 | 0.0036 | 0.0005 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 14 | 12 | 0.2824 | 0.0687 | 0.0138 | 0.0022 | 0.0002 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 15 | 13 | 0.2542 | 0.0550 | 0.0097 | 0.0013 | 0.0001 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 16 | 14 | 0.2288 | 0.0440 | 0.0068 | 0.0008 | 0.0001 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 17 | 15 | 0.2059 | 0.0352 | 0.0047 | 0.0005 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 18 | 16 | 0.1853 | 0.0281 | 0.0033 | 0.0003 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 19 | 17 | 0.1668 | 0.0225 | 0.0023 | 0.0002 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 20 | 18 | 0.1501 | 0.0180 | 0.0016 | 0.0001 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 21 | 19 | 0.1351 | 0.0144 | 0.0011 | 0.0001 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 22 | 20 | 0.1216 | 0.0115 | 0.0008 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 23 | | | | | | | | | | |

# Moving Weighted Average
# Select the value of $\gamma = 0.5$

| | A | F |
|---|---|---|
| 1 | | |
| 2 | **Gamma** | **0.5** |
| 3 | 1 | 0.5000 |
| 4 | 2 | 0.2500 |
| 5 | 3 | 0.1250 |
| 6 | 4 | 0.0625 |
| 7 | 5 | 0.0313 |
| 8 | 6 | 0.0156 |
| 9 | 7 | 0.0078 |
| 10 | 8 | 0.0039 |
| 11 | 9 | 0.0020 |
| 12 | 10 | 0.0010 |
| 13 | 11 | 0.0005 |
| 14 | 12 | 0.0002 |
| 15 | 13 | 0.0001 |
| 16 | 14 | 0.0001 |
| 17 | 15 | 0.0000 |
| 18 | 16 | 0.0000 |
| 19 | 17 | 0.0000 |
| 20 | 18 | 0.0000 |
| 21 | 19 | 0.0000 |
| 22 | 20 | 0.0000 |
| 23 | | |

- How to decide the length of moving average?
  - Window length = 14
- How to decide the weights?
  - Weights are computed automatically

# Solution#1: Momentum Based Gradient Descent Algorithm

Solution#1: Increase the step size

- Momentum based GD
- Nesterov GD

Solution#2: Reducing the data points for approximate gradient

- Stochastic Gradient Descent
- Mini Batch Gradient Descent

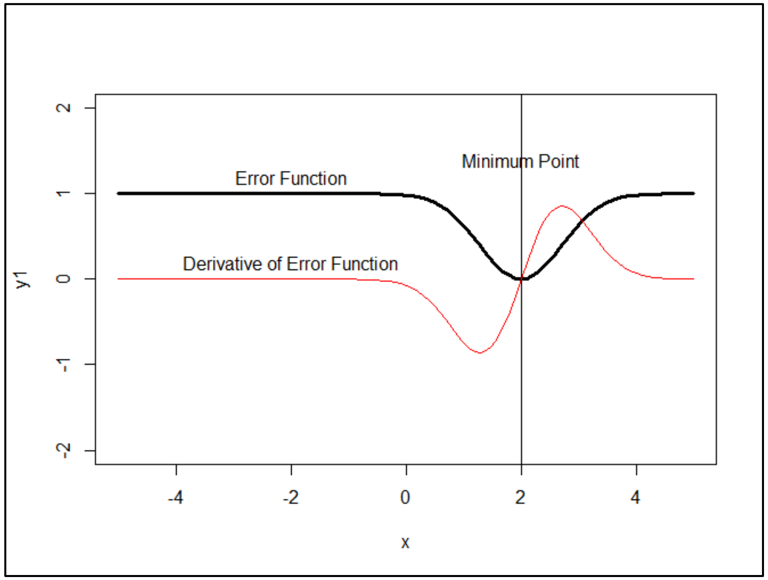
Solution#3: Adjust the Learning rate $(\eta)$

- AdaGrad
- RMSProp
- Adam

# Example#1

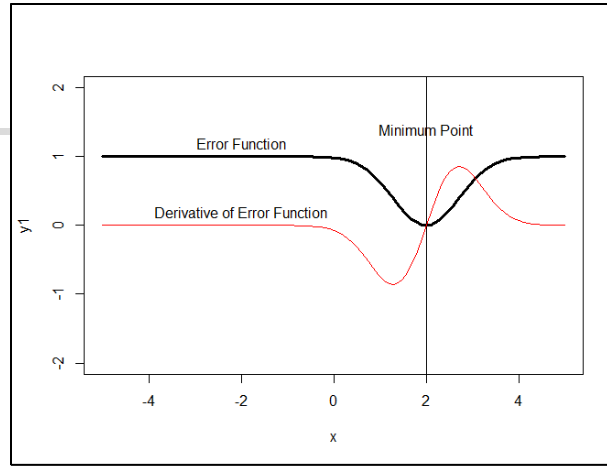
- $Error\ Function\colon\ y = f(x) = 1 - e^{-(x-2)^2}$
- $\frac{dy}{dx} = 2(x-2)e^{-(x-2)^2}$
- The error function 'y' is almost flat
  - $f(x) = 0\colon\ -\infty < x < 0\ and$
  - $f(x) = 0\colon\quad 4 < x < \infty$

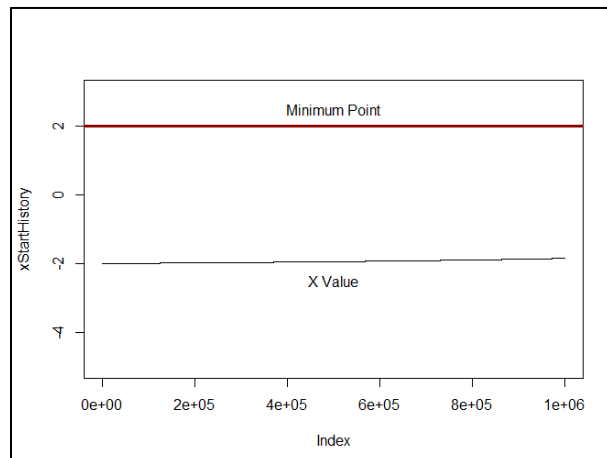
```
> ###########################################
> # Example#1
> #
> x = seq(-5,5,0.1)
> y1 = 1 - exp(-(x-2)^2)
> plot(x,y1,type='l',ylim=c(-2,2),lwd=3)
> text(-2,1.2,"Error Function")
> text(2,1.4,"Minimum Point")
> abline(v=2)
> dy_dx = function (w1) { 2*(w1-2)*exp(-(w1-2)^2)
}
> slope = dy_dx(x)
> points(x,slope,type='l',col='red')
> text(-2,0.2,"Derivative of Error Function")
```



# Example#1: Gradient Descent
# Does not converge after 1,000,000 iterations

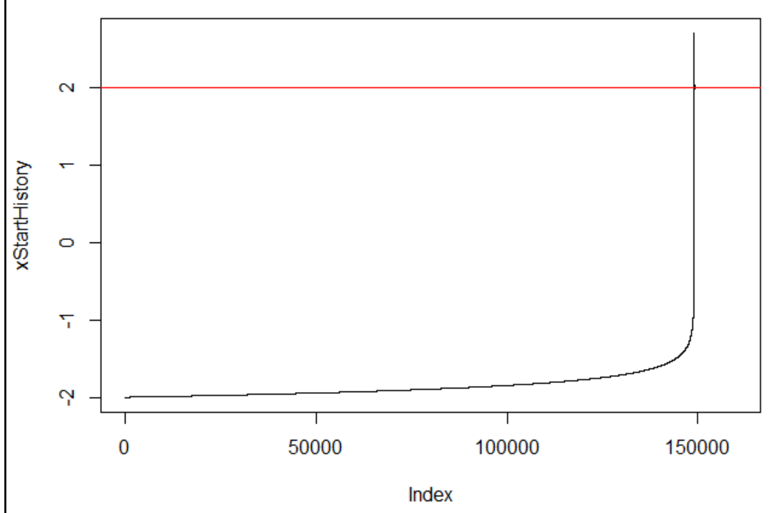
```
> ##########################################
> xStart = -2.0
> learningRate = 0.1
>
> ########################################
> maxLimit = 1000000
> xStartHistory = rep(0,maxLimit)
>
> for ( i in 1:maxLimit )
+ {
+   xStartHistory[i] = xStart
+   xStart = xStart - learningRate * dy_dx(xStart)
+ }
>
> plot(xStartHistory,type='l',ylim=c(-5,3))
> text(500000,-2.5,"X Value")
>
> abline(h=2, col='dark red', lwd=3)
> text(500000,2.5,"Minimum Point")
>
> ##############################################
> >
```



# Example#1: Gradient Descent with Momentum Converges after 150,000 Iterations

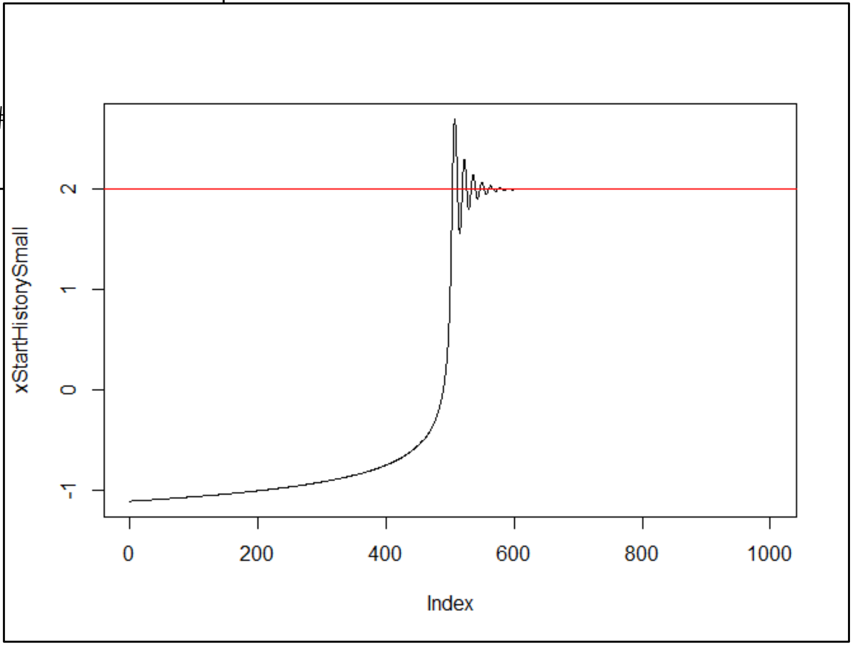
```
###########################################
xStart = -2.0
learningRate = 0.1

########################################
# Momentum
maxLimit = 160000
xStartHistory = rep(0,maxLimit)
gamma = 0.9
update = 0
for ( i in 1:maxLimit ){
  xStartHistory[i] = xStart
  gradient = dy_dx(xStart)

  update = (gamma * update) + (learningRate * gradient)
  xStart = xStart - update
}
plot(xStartHistory,type='l')
abline(h=2,col='red')
```




- $update_t = \gamma * update_{t-1} + \eta \frac{\partial z}{\partial x} ||_t$
- $x_{t+1} = x_t - update_t$
- $update_t = \gamma * update_{t-1} + \eta \frac{\partial z}{\partial x}|_t = \left(\gamma^{t-1} * \eta \frac{\partial z}{\partial x}|_1\right) + \left(\gamma^{t-2} * \eta \frac{\partial z}{\partial x}|_2\right) + \cdots + \left(\eta \frac{\partial z}{\partial x}|_t\right)$

# Example#1:
# Just Before Reaching the Minimum Point Oscillation

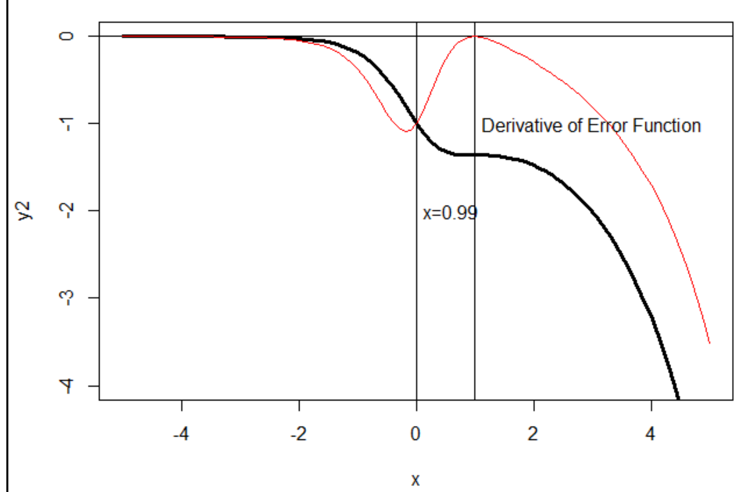
```
> ####################################################
> # Expand when the values are converging
> #
> xStartHistorySmall = xStartHistory[148500:149500]
> plot(xStartHistorySmall,type='l')
> abline(h=2,col='red')
>
> ####################################################
>
```



# Example#2

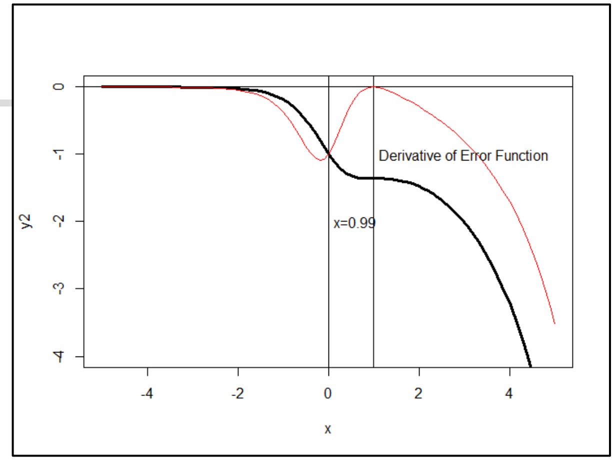
- $Error\ Function = y = -\frac{e^x}{1+x^2}$
- $\frac{dy}{dx} = \frac{(1+x^2)\frac{d}{dx}(e^x) - e^x\frac{d}{dx}(1+x^2)}{(1+x^2)^2}$
- $\frac{dy}{dx} = -\frac{e^x(1-x)^2}{(1+x^2)^2}$

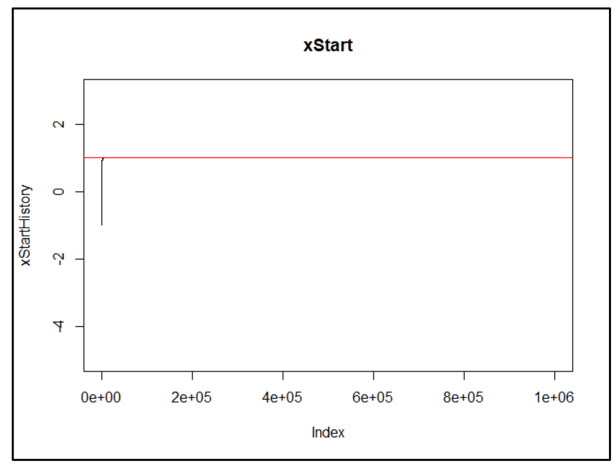
```
>
##################################################
> # Example#2
> #
> x = seq(-5,5,0.1)
> y1 = (exp(x)/(1+x^2))
> y2 = (-1)*y1
> plot(x,y2,type='l',ylim=c(-
4,0),lwd=3,col='black')
> abline(v=0)
> abline(h=0)
> abline(v=0.99)
> text(0.6,-2,"x=0.99")
> dy_dx = function (w1)
+ {
+   dy_dx_n =  exp(w1)*((1-w1)^2)
+   dy_dx_d = (1 + w1^2)^2
+   return( -dy_dx_n/dy_dx_d)
+ }
> slope = dy_dx(x)
> points(x,slope,type='l',col='red')
> text(3,-1,"Derivative of Error Function")
>
```



# Example#2: Gradient Descent
# Does not converge after 1,000,000 iterations

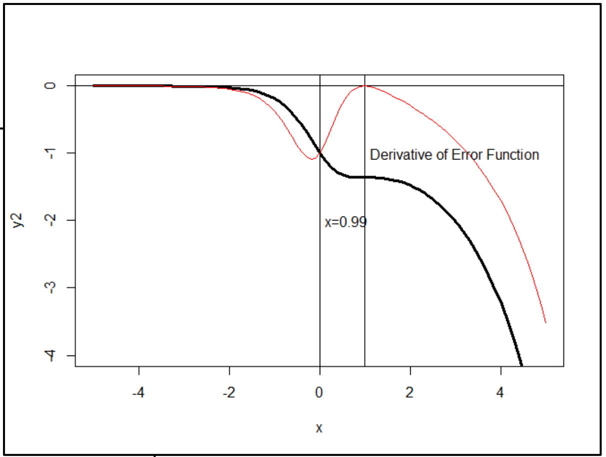
```
##########################################
xStart = -1.0
learningRate = 0.1
maxLimit = 1000000
xStartHistory = rep(0,maxLimit)

for ( i in 1:maxLimit )
{
  xStartHistory[i] = xStart
  xStart = xStart - learningRate * dy_dx(xStart)

  if ( i %% 100000 == 0 ) { cat ("i = ",i, "xstart
= ", xStart, "\n")}
}

plot(xStartHistory,type='l',ylim=c(-5,3),
main="xStart")
abline(h=1,col='red')
```

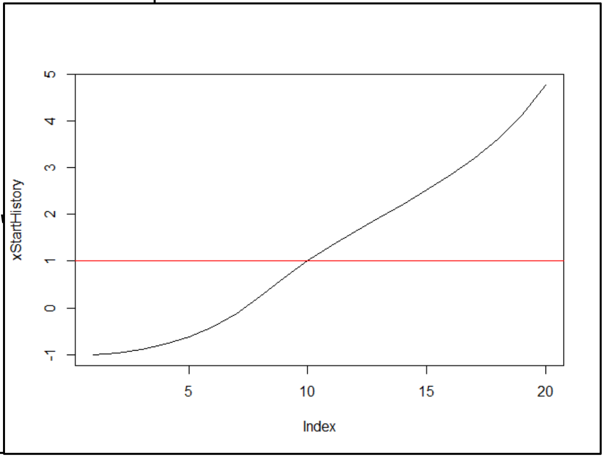


# Example#2: Gradient Descent with Momentum Crosses the line x=1 in 10 iterations

```
xStart = -1.0
learningRate = 0.1
maxLimit = 20
xStartHistory = rep(0,maxLimit)
gamma = 0.9
update = 0
for ( i in 1:maxLimit )
{
  xStartHistory[i] = xStart
  gradient = dy_dx(xStart)

  update = (gamma * update) + (learningRate * gradient)
  xStart = xStart - update

  if ( i %% 1000 == 0 ) { cat ("i = ",i, "xstart = ", xStart, "
}

#plot(xStartHistory,type='l',ylim=c(-5,3), main="xStart")
plot(xStartHistory,type='l')
abline(h=1,col='red')
```




- $update_t = \gamma * update_{t-1} + \eta \frac{\partial z}{\partial x} ||_t$
- $x_{t+1} = x_t - update_t$
- $update_t = \gamma * update_{t-1} + \eta \frac{\partial z}{\partial x} |_t = \left(\gamma^{t-1} * \eta \frac{\partial z}{\partial x} |_1\right) + \left(\gamma^{t-2} * \eta \frac{\partial z}{\partial x} |_2\right) + \cdots + \left(\eta \frac{\partial z}{\partial x} |_t\right)$

# Analysis of Momentum Based Gradient Descent Algorithm

- Pros: The algorithm moves faster on the flat surface
- Cons: But it over shoots when it arrives the minimum point
  - Then it has to be backtrack
  - When the algorithm reaches close to the minimum point, it starts oscillating

# Solution to Momentum Nesterov Momentum

- Before jumping to the next check the gradient at the next step also
- If the next step gradient forces to take a 'U' turn
  - Reduce the size of the step

# Yurii Nesterov

- Nesterov is most famous for his work in convex optimization, including his 2004 book, considered a canonical reference on the subject
- His main novel contribution is an accelerated version of gradient descent that converges considerably faster than ordinary gradient descent (commonly referred as Nesterov momentum or Nesterov accelerated gradient, in short — NAG)

**Yurii Nesterov**

2005 in Oberwolfach

| | |
|---|---|
| **Born** | January 25, 1956 (age 63)<br>Moscow, USSR |
| **Citizenship** | Russia |
| **Alma mater** | Moscow State University (1977) |
| **Awards** | Dantzig Prize, 2000<br>John von Neumann Theory Prize, 2009<br>EURO Gold Medal, 2016 |
| **Scientific career** | |
| **Fields** | Convex optimization,<br>Semidefinite programming,<br>Nonlinear programming,<br>Numerical analysis,<br>Applied mathematics |

# Solution#2: Reducing the Data Points for Approximate Gradient

Mini Batch Gradient Descent

Stochastic Gradient Descent

Solution#1: Increase the step size
- Momentum based GD
- Nesterov GD

Solution#2: Reducing the data points for approximate gradient
- Stochastic Gradient Descent
- Mini Batch Gradient Descent

Solution#3: Adjust the Learning rate $(\eta)$
- AdaGrad
- RMSProp
- Adam

$$\nabla RSS(b,m) = \begin{vmatrix} \frac{\partial RSS(m,b)}{\partial b} \\ \frac{\partial RSS(m,b)}{\partial m} \end{vmatrix} = \begin{vmatrix} -2\sum_{i=1}^{N}(y_i-(mx_i+b)) \\ -2\sum_{i=1}^{N}(y_i-(mx_i+b))x_i \end{vmatrix}$$

# Gradient

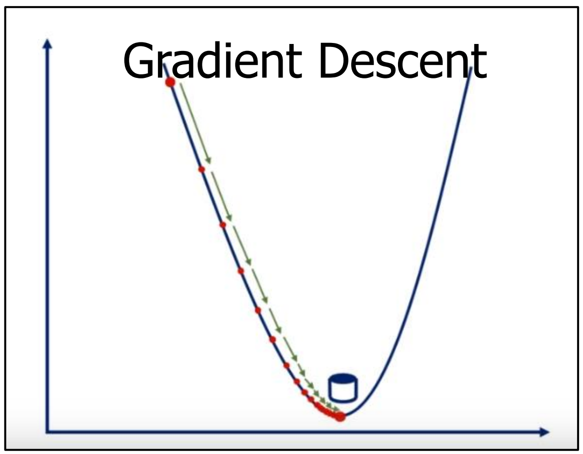
- The gradient descent algorithm uses all the data points to compute gradient
- If the number of data elements are large
  - It takes a lot of time to compute gradient
- Suppose we take less number of elements to compute the gradient
  - Solution will be approximate but faster

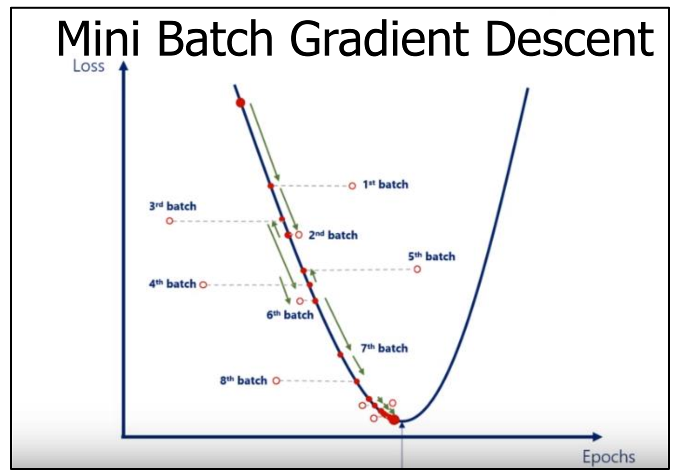
# Types of Gradient Descent

- Mini Batch Gradient Descent
  - Data is divided into small batches
  - Compute gradient using batch data
  - The value of 'x' is incremented
- Stochastic Gradient Descent
  - Randomly select 'n' data points
  - Compute the gradient
  - The value of 'x' is incremented

# Mini Batch Gradient Descent

- Batch Gradient Descent
  - Gradient is not computed using all the data values
  - Gradient is approximate
  - It is possible that the descent may not be smooth


Gradient Descent



Mini Batch Gradient Descent

# Mini Batch Gradient Descent

$$\nabla RSS(b,m) = \begin{vmatrix} \frac{\partial RSS(m,b)}{\partial b} \\ \frac{\partial RSS(m,b)}{\partial m} \end{vmatrix} = \begin{vmatrix} -2\sum_{i=1}^{N}(y_i-(mx_i+b)) \\ -2\sum_{i=1}^{N}(y_i-(mx_i+b))x_i \end{vmatrix}$$

- Gradient Descent
  - Compute gradient using all data elements
  - Update the value of 'x'
  - End of epoch#1
- Mini Batch Gradient Descent: Divide the data into many batches
  - $1^{st}$ Batch
    - Compute gradient using the $1^{st}$ batch of data
    - Gradient may not be precise
    - Update the value of 'x'
  - $2^{nd}$ Batch
    - Compute gradient using the $2^{nd}$ batch of data
    - Update the value of 'x'
  - Continue till all data is used to compute gradient
  - End of epoch#1
  - In a single epoch, 'x' value is incremented many times

# Stochastic Gradient Descent

$$\nabla RSS(b,m) = \begin{vmatrix} \dfrac{\partial RSS(m,b)}{\partial b} \\ \dfrac{\partial RSS(m,b)}{\partial m} \end{vmatrix} = \begin{vmatrix} -2\sum_{i=1}^{N}(y_i-(mx_i+b)) \\ -2\sum_{i=1}^{N}(y_i-(mx_i+b))x_i \end{vmatrix}$$

- Batch size is determined by selecting the data element randomly

# Summary

- Problems with Gradient Descent Algorithm
- Moving Average
- Weighted Moving Average
- Exponentially Weighted Moving Average (EWMA)
- Momentum based Gradient Descent Algorithm
- Nestirov Gradient Descent Algorithm
- Reducing the Data Points for Approximate Gradient
  - Stochastic Gradient Descent Algorithm
  - Mini Batch Gradient Descent Algorithm