Deep Learning Using R & TensorFlow

Dr. Ash Pahwa

OC R User's Group

March 26, 2019

April 29, 2019

Bio: Dr. Ash Pahwa

- Ph.D. Computer Science
- Website: www.AshPahwa.com
- Affiliation
  - California Institute of Technology, Pasadena
  - UC Irvine
  - UCLA
  - UCSD
  - Chapman University: Adjunct
- Field of Expertise
  - Machine Learning, Deep Learning, Digital Image Processing, Database Management, CD-ROM/DVD
- Worked for
  - General Electric, AT&T Bell Laboratories, Oracle, UC Santa Barbara

Caltech Course

Caltech

CALIFORNIA INSTITUTE OF TECHNOLOGY

Center for Technology & Management Education

Division of Engineering & Applied Science

Deep Learning with TensorFlow

SCHEDULE

Classes are held Saturdays, 8:00 AM to 5:00 PM, on the Caltech campus in Pasadena, California.

	Spring 2019	Add to Cart
Deep Learning with TensorFlow	April 6, 13, 20	Register

INSTRUCTOR

Ash Pahwa, PhD

Outline: Part 1
OC R User's Group: March 26, 2019

- What is Deep Learning
- Deep Learning Applications
- Tools for Deep Learning
- Misunderstanding about TensorFlow
- Availability of GPU
- TensorFlow Architecture
- R Code with TensorFlow

Outline: Part 2
OC R User’s Group: April 29, 2019

- Title: Deep Learning Optimization Techniques
  - Gradient Descent
  - Momentum
  - Nesterov Momentum
  - AdaGrad
  - RMSProp
  - Adam: Adaptive Moments

What is Optimization Problem?

- Regression
  - Solution: Minimize the cost function
    - Closed form solution: Matrix Approach
    - Iterative Approach: Gradient Descent Algorithm

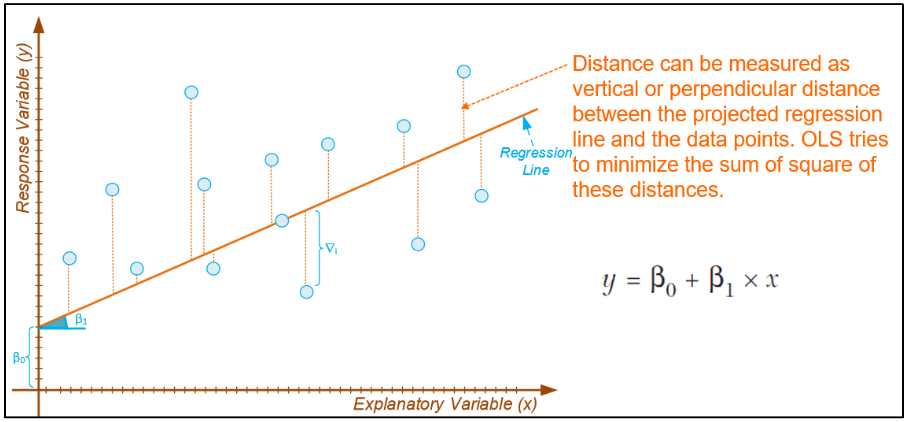
- Neural Networks: Deep Learning
  - Solution: Minimize the cost function
    - Iterative Approach: Gradient Descent Algorithm

Regression

$Cost\ Function = (Computed\ Output\ - Observed\ Output)^2$

Minimize Cost Function

Closed Form Solution: Matrix Approach



- $Y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \dots & \dots \\ 1 & x_n \end{bmatrix} \quad A = \begin{bmatrix} b \\ m \end{bmatrix} \quad E = \begin{bmatrix} e_1 \\ e_2 \\ \dots \\ e_n \end{bmatrix}$
- Matrix Equation: $Y = XA + E$
- $E = Y - XA$

Closed Form Solution
Matrix Approach

Used by
- R
- Python
- Excel
- All other statistical software

- $RSS = (\boldsymbol{Y} - \boldsymbol{XA})^{\boldsymbol{T}}(\boldsymbol{Y} - \boldsymbol{XA})$
- $RSS = \boldsymbol{Y^TY} - \boldsymbol{2Y^TXA} + \boldsymbol{AX^TXA}$
- $Set\ \frac{\partial RSS}{\partial A} = 0,$
  - $to\ compute\ the\ value\ of\ A\ for\ minimum\ RSS$
- $\boldsymbol{A = (X^TX)^{-1}X^TY}$

Example

X	Y
0	1
1	3
2	7
3	13
4	21

- $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$
- $A^{-1} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$

- Solution for Least RSS = $A = (X^T X)^{-1} X^T Y$

- $Y = \begin{bmatrix} 1 \\ 3 \\ 7 \\ 13 \\ 21 \end{bmatrix}$ $X = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}$

- $X^T X = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 10 \\ 10 & 30 \end{bmatrix}$
- $(X^T X)^{-1} = \frac{1}{50}\begin{bmatrix} 30 & -10 \\ -10 & 5 \end{bmatrix} = \begin{bmatrix} 0.6 & -0.2 \\ -0.2 & 0.1 \end{bmatrix}$
- $X^T Y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 7 \\ 13 \\ 21 \end{bmatrix} = \begin{bmatrix} 45 \\ 140 \end{bmatrix}$
- $A = (X^T X)^{-1} X^T Y = \begin{bmatrix} 0.6 & -0.2 \\ -0.2 & 0.1 \end{bmatrix} \begin{bmatrix} 45 \\ 140 \end{bmatrix} = \begin{bmatrix} -1 \\ 5 \end{bmatrix}$

Regression Equation
$y = 5x - 1$

Who Invented Gradient Descent Algorithm?

- Gradient Descent algorithm was invented by **Cauchy** in 1847
- Méthode générale pour la résolution **des** systèmes d'équations simultanées. pp. 536–538

Augustin-Louis Cauchy

Cauchy around 1840. Lithography by Zéphirin Belliard after a painting by Jean Roller.

Born	21 August 1789 Paris, France
Died	23 May 1857 (aged 67) Sceaux, France
Nationality	French
Alma mater	École Nationale des Ponts et Chaussées
Known for	See list
Spouse(s)	Aloise de Bure
Children	Marie Françoise Alicia, Marie Mathilde

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
- 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:
    - $x^{t+1} \leftarrow x^t - \eta \frac{\partial z}{\partial x} \|_{x^t, y^t}$
    - $y^{t+1} \leftarrow y^t - \eta \frac{\partial z}{\partial y} \|_{x^t, y^t}$

Partial Derivatives of the RSS w.r.t. Intercept and Slope

- $Residuals\ Sum\ of\ Squares = (RSS) = \sum_{i=1}^{N}(y_i - (mx_i + b))^2$
- To find the minimum point of this function,
  - we will take the partial derivative of RSS with respect to 'm' and 'b' and set that to zero.

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

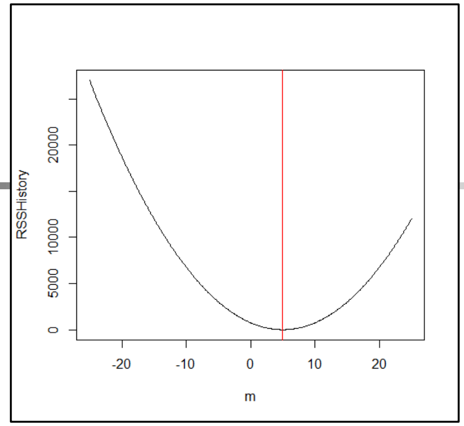
- $RSS = \sum_{i=1}^{N}(y_i - (mx_i + b))^2$
- $\frac{\partial RSS(m,b)}{\partial m} = \sum_{i=1}^{N}\frac{\partial}{\partial m}(y_i - (mx_i + b))^2$
- $\frac{\partial RSS(m,b)}{\partial m} = -2\sum_{i=1}^{N}(y_i-(mx_i + b))x_i$

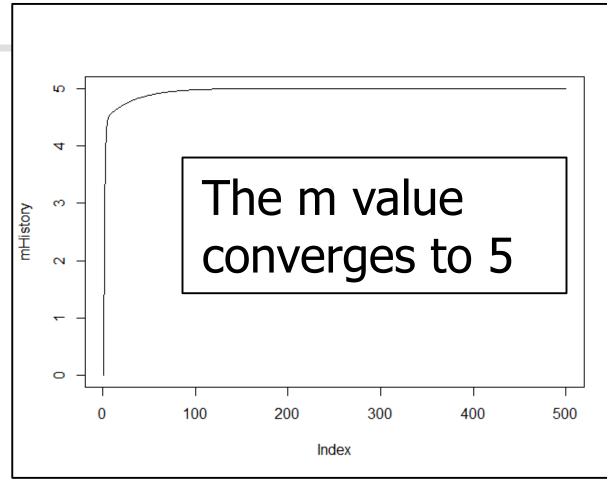
$$\nabla RSS(b,m) = \begin{vmatrix} \frac{\partial RSS(m,b)}{\partial b} \\ \frac{\partial RSS(m,b)}{\partial m} \end{vmatrix} = \begin{vmatrix} -2\sum_{i=1}^{N}(y_i-(mx_i+b)) \\ -2\sum_{i=1}^{N}(y_i-(mx_i+b))x_i \end{vmatrix} = 0$$

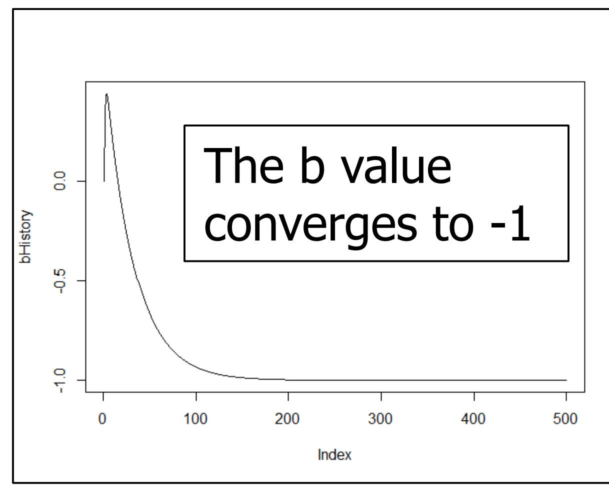
Regression
Gradient Descent Algorithm Approach

X	Y
0	1
1	3
2	7
3	13
4	21

Regression Equation
$y = 5x - 1$







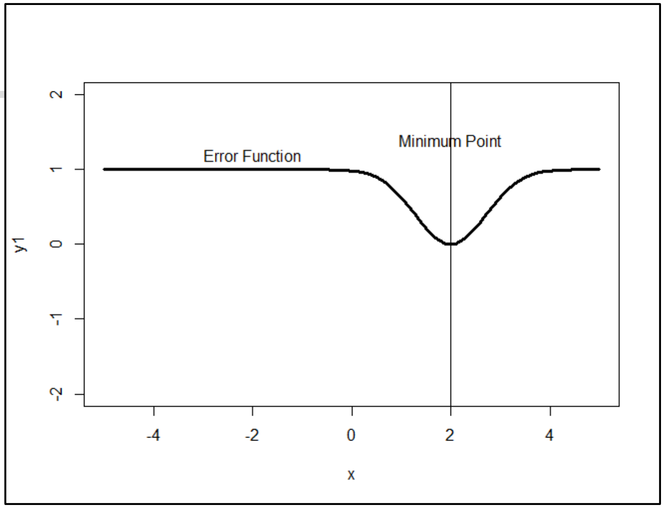

```
> dRSS_dm = function (m,b) {-2*sum((y-m*x-b)*x)   }
> dRSS_db = function (m,b) { -2*sum(y-m*x-b) }
> mStart = bStart = 0
> learningRate = 0.01;  maxLimit = 500
> mHistory = bHistory = rep(0,maxLimit)
> for ( i in 1:maxLimit )
+ {
+   mHistory[i] = mStart
+   bHistory[i] = bStart
+
+   dW = dRSS_dm(mStart,bStart)
+   db = dRSS_db(mStart,bStart)
+
+   mStart = mStart - learningRate * dW
+   bStart = bStart - learningRate * db
+ }
> plot(mHistory,type='l')
> plot(bHistory,type='l')
```

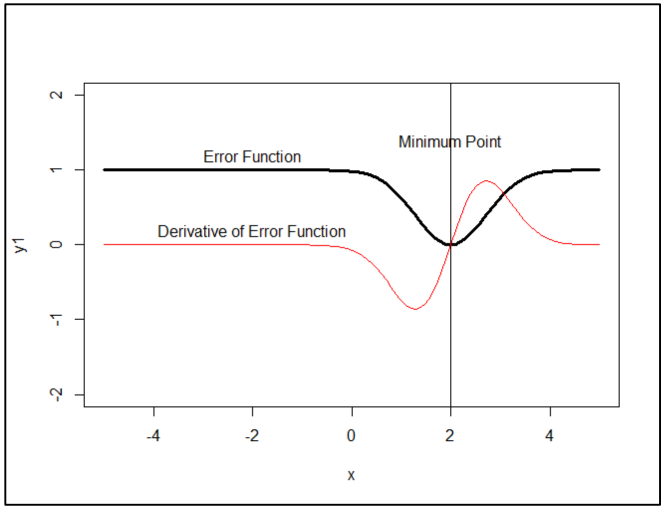
Problems with Gradient Descent Algorithm

- When the gradient of the error function is low (flat)
  - It takes a long time for the algorithm to converge (reach the minimum point)
- If there are multiple local minima, then
  - There is no guarantee that the procedure will find the global minimum

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
  - $f(x) = 0: -\infty < x < 0\ and$
  - $f(x) = 0: \quad 4 < x < \infty$



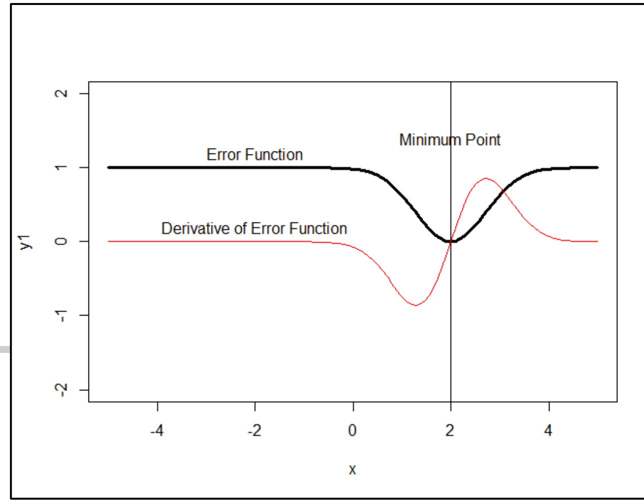


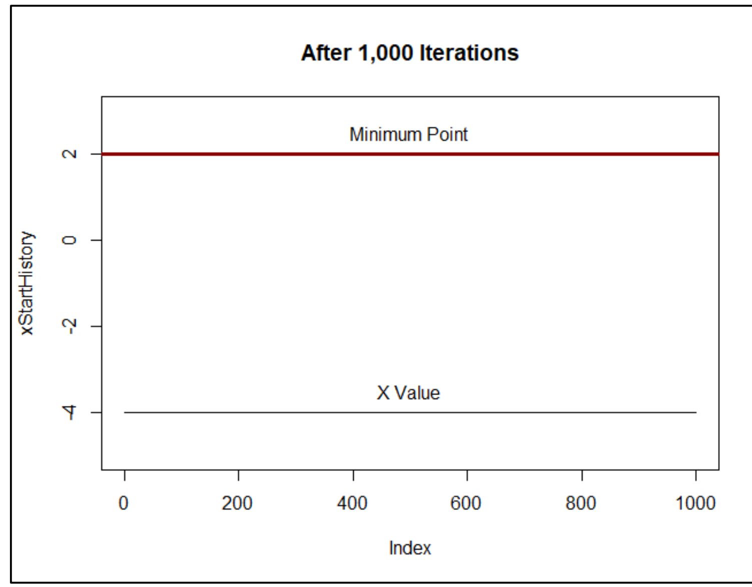

```
> x = seq(-5,5,0.1)
> y1 = 1 - exp(-(x-2)^2)
> plot(x,y1,type='l',ylim=c(-2,2),lwd=3)
> text(-2,1.2,"Error Function")
> text(2,1.4,"Minimum Point")
> abline(v=2)
> dy_dx = function (w1) { 2*(w1-2)*exp(-(w1-2)^2) }
> slope = dy_dx(x)
> points(x,slope,type='l',col='red')
> text(-2,0.2,"Derivative of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

Gradient Descent Algorithm

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

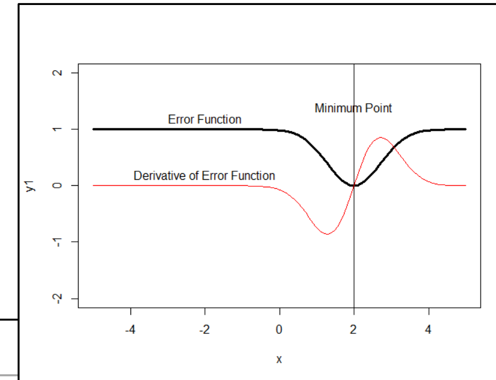
```
> xStart = -4
> learningRate = 0.1
> maxLimit = 1000
> xStartHistory = rep(0,maxLimit)
> for ( i in 1:maxLimit )
+ {
+   xStartHistory[i] = xStart
+   xStart = xStart - learningRate * dy_dx(xStart)
+ }
> plot(xStartHistory,type='l',ylim=c(-5,3),
main="After 1,000 Iterations")
> text(500,-3.5,"X Value")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

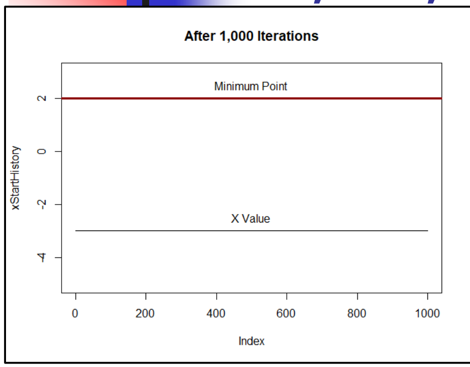


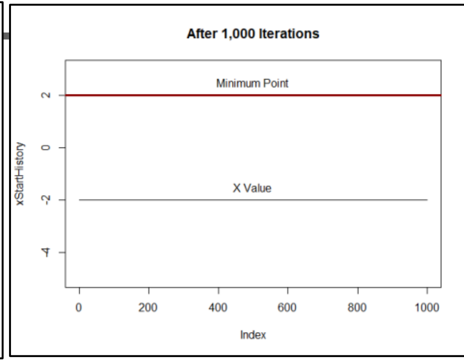


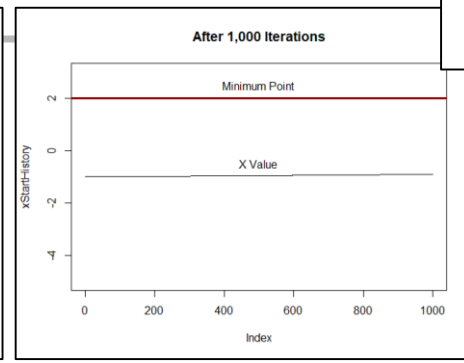
Starting Point = -4, DOES NOT CONVERGE

After 1,000 iteration Starting Point -3, -2, -1, -0.5

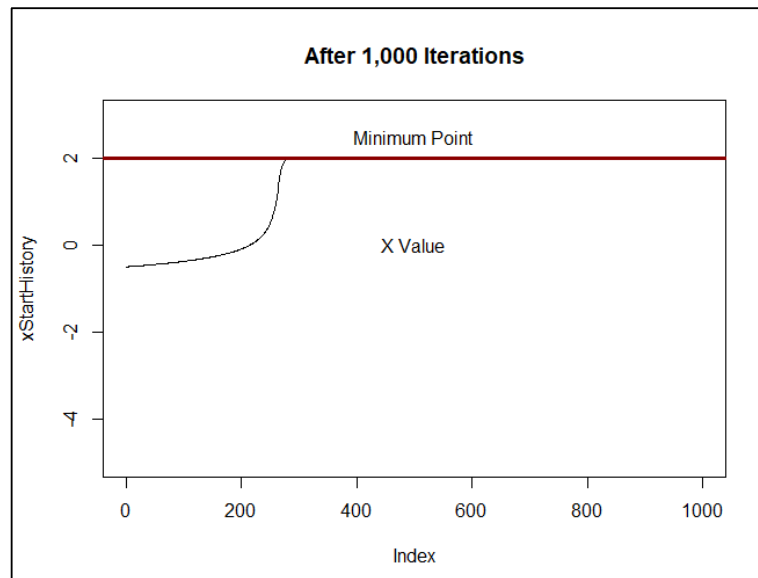








Starting Point = -3, -2, -1: 'x' value DOES NOT CONVERGE

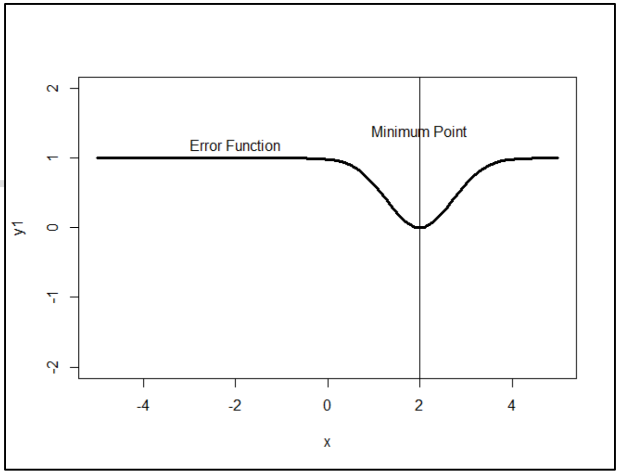


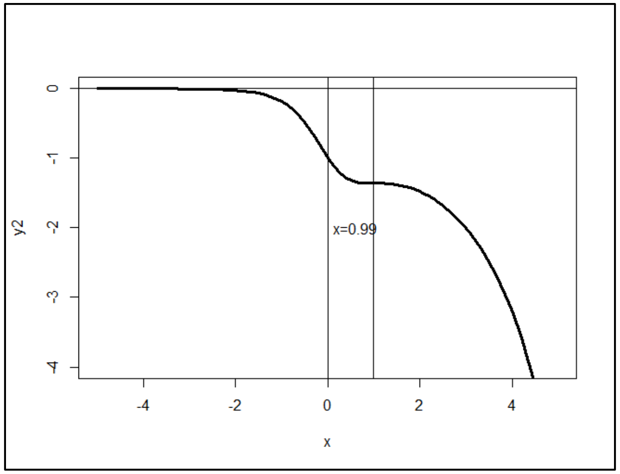
Starting Point = -0.5,

'x' value CONVERGES AFTER 250 ITERATIONS

Problems with Gradient Descent Algorithm

- If the error function has very small gradient
  - Starting point is on the flat surface
  - Gradient Descent Algorithm will take a long time to converge





Solutions to Gradient Descent Algorithm Problem

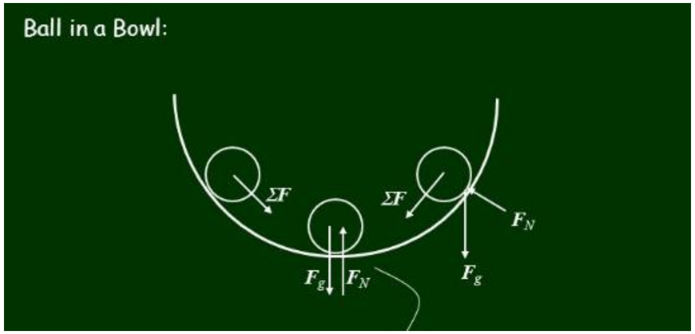
- Solution#1: Increase the step size
  - Momentum based GD
  - Nesterov GD
- Solution#2: Reduce the data points for approximate gradient
  - Stochastic Gradient Descent
  - Mini Batch Gradient Descent
- Solution#3: Adjust the Learning rate (η)
  - AdaGrad
  - RMSProp
  - Adam

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

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
  - $x_{t+1} = x_t - \eta \frac{\partial z}{\partial x} - (previous\ values\ of\ \frac{\partial z}{\partial x})$

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

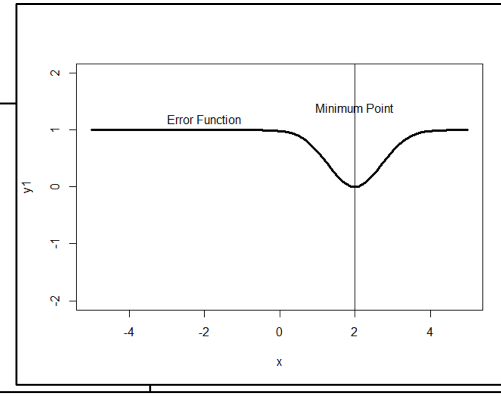
- Momentum is building as we are moving along
- Speed at which we are moving will increase with time

Exponentially Weighted Moving Averages (EWMA)

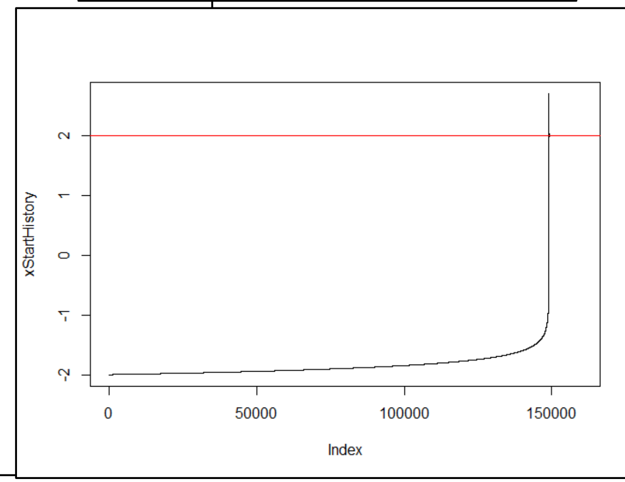
Exponentially Weighted Moving Average of Gradients

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

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



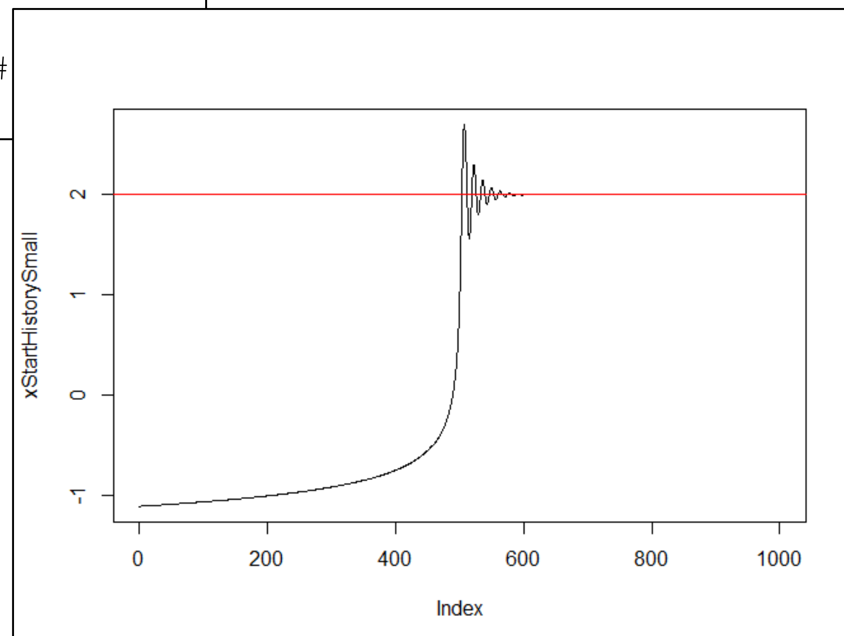

```
##############################################
xStart = -2.0
learningRate = 0.1

###########################################
# 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')
>
> ##################################################
>
```



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

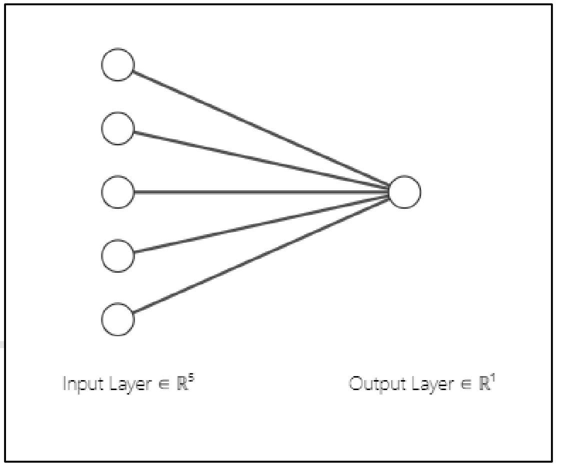
Born	January 25, 1956 (age 63) Moscow, USSR
Citizenship	Russia
Alma mater	Moscow State University (1977)
Awards	Dantzig Prize, 2000 John von Neumann Theory Prize, 2009 EURO Gold Medal, 2016
Scientific career	
Fields	Convex optimization, Semidefinite programming, Nonlinear programming, Numerical analysis,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

Learning Rate η

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

- How to decide the Learning Rate (LR)?
- If LR is small
  - Gradient will gently descend and we will get the optimum value of 'x' for which the error is minimum
  - But it may take a long time to converge
- If LR is large
  - NN will converge faster
  - But when we reach the destination "minimum" point of the error function we will see the 'x' values oscillation

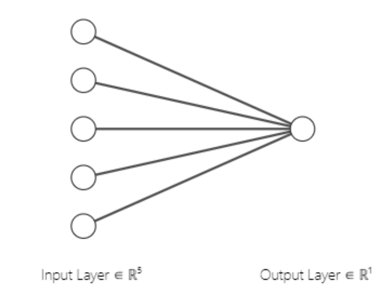
AdaGrad
Adaptive Gradient



- AdaGrad
  - It dynamically varies the learning rate
    - At each update
    - For each weight individually
- Learning rate decreases as the algorithm moves forward
- Every variable will have a separate learning rate
- In the above example
  - 5 weights from input to output layers
  - 5 separate learning rates which are decreasing as algorithm moves forward

AdaGrad
Adaptive Gradient Algorithm

- Gradient Descent
  - While NOT converged:
    - $x^{t+1} \leftarrow x^t - \eta \frac{\partial y}{\partial x}|_{x^t}$
- AdaGrad: Update rule is for individual weight
    - $x_i^{t+1} = x_i^t - \frac{\eta}{\sqrt{G_i^t + \varepsilon}} * \frac{\partial y}{\partial x_i}|_t$
    - $G_i^t = G_i^{t-1} + \left(\frac{\partial y}{\partial x_i}|_t\right)^2$
    - $G_o^i = 0$
    - As we move ahead, the G value will increase because we are adding the square of the derivative which will always remain positive
    - As the value of G increases, the value of learning rate η will decrease.
    - The value of epsilon (an arbitrary small number) is added to the denominator to avoid a situation of dividing by zero



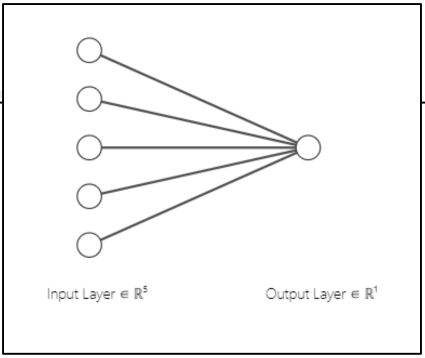
Adaptive Learning Rate Schedule for each weight
$$1 \leq i \leq n$$
Where 'n' = number of weights

Difference: AdaGrad and RMSProp

- $$x_i^{t+1} = x_i^t - \frac{\eta}{\sqrt{G_i^t + \varepsilon}} * \frac{\partial y}{\partial x_i}\Big|_t$$
- AdaGrad
  - $$G_i^t = G_i^{t-1} + \left(\frac{\partial y}{\partial x_i}\Big|_t\right)^2$$
- RMS Prop
  - $$G_i^t = \beta * G_i^{t-1} + (1 - \beta) * \left(\frac{\partial y}{\partial x_i}\Big|_t\right)^2$$

- AdaGrad
  - Since the 'G' value will increase continuously
    - Learning rate will decrease continuously
    - Eventually learning rate will be very small
- RMS Prop
  - Since the 'G' value will NOT increase continuously
    - It will adjust to the data and decrease or increase the learning rate accordingly

Root Mean Square Propagation "RMS Prop" Algorithm



- AdaGrad
  - $x_i^{t+1} = x_i^t - \frac{\eta}{\sqrt{G_i^t + \varepsilon}} * \frac{\partial y}{\partial x_i}|_t$
  - $G_i^t = G_i^{t-1} + \left(\frac{\partial y}{\partial x_i}|_t\right)^2$
  - $G_o^i = 0$
- RMS Prop
  - $x_i^{t+1} = x_i^t - \frac{\eta}{\sqrt{G_i^t + \varepsilon}} * \frac{\partial y}{\partial x_i}|_t$
  - $G_i^t = \beta * G_i^{t-1} + (1 - \beta) * \left(\frac{\partial y}{\partial x_i}|_t\right)^2$
  - Where β is another hyper parameter: typical value = 0.9

Adaptive Learning Rate Schedule for each weight
$1 \leq i \leq n$
Where 'n' = number of weights

What is Adam? Adaptive Moment Estimation

- Adam = RMS Prop + Momentum
- Adam takes the positive points of
  - Momentum &
  - RMS Prop algorithm
- Authors: Diederik Kingma and Jimmy Bai

D. P. Kingma and J. L. Bai. Adam: a method for stochastic optimization. *ICLR*, 2015.

Adam
Adaptive Moment Estimation

- Adaptive Moment Estimation (Adam) combines ideas from both RMSProp and Momentum
- It computes adaptive learning rates for each parameter and works as follows
  - $m_t = \beta_1 * m_{t-1} + (1 - \beta_1)\frac{\partial z}{\partial x}|_t$
  - $v_t = \beta_2 * v_{t-1} + (1 - \beta_2)\left(\frac{\partial z}{\partial x}|_t\right)^2$

$$\boxed{\begin{array}{c} m_0 = 0 \\ v_0 = 0 \end{array}}$$

- Lastly, the parameters are updated using the information from the calculated averages
  - $w_t = w_{t-1} - \eta \frac{m_t}{\sqrt{v_t} + \varepsilon}$

- $m_t: Exponentially\ weighted\ average\ of\ past\ gradients$
- $v_t: Exponentially\ weighted\ average\ of\ the\ past\ squares\ of\ gradients$

Bias Correction

- The Adam works best when the
- $\text{Expectations}(m_t) = \text{Expectation}(\text{distribution of derivative})$
- $Expectation(v_t) = \text{Expectation}(\text{distribution of square of derivative})$
- After every update we adjust the value of $m_t \ and \ v_t$

- Adam
  - $m_t = \beta_1 * m_{t-1} + (1-\beta_1)\frac{\partial z}{\partial x}|_t$
  - $v_t = \beta_2 * v_{t-1} + (1-\beta_2)\left(\frac{\partial z}{\partial x}|_t\right)^2$
  - $w_t = w_{t-1} - \eta\frac{m_t}{\sqrt{v_t}+\varepsilon}$
- Adam Bias Correction
  - $m_t = \beta_1 * m_{t-1} + (1-\beta_1)\frac{\partial z}{\partial x}|_t$
  - $v_t = \beta_2 * v_{t-1} + (1-\beta_2)\left(\frac{\partial z}{\partial x}|_t\right)^2$
  - $\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$ $\widehat{v_t} = \frac{v_t}{1-\beta_2^t}$
  - $w_t = w_{t-1} - \eta\frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}+\varepsilon}$

Adam Algorithm Implementation in R

Adam

Adaptive Moment Estimation

Define Adam Hyper Parameters

```
> #######################################
> # ADAM Hyper Parameters
> # learning Rate + epochs + epsilon
> #
> learningRate <- 0.01
> epsilon <- 1e-8
>
> ##################################
> # EWMA: Exponentially Weighted Moving Average
> # Hyper parameters
> #
> beta1 <- 0.9
> beta2 <- 0.999
>
> ######################################
> # Initial value of 'm' and 'v' is zero
> m <- 0
> v <- 0
>
> epochs <- 1000
> l <- nrow(Y)
> costHistory <- array(dim=c(epochs,1))
>
```

$m_0 = 0$
$v_0 = 0$

TensorFlow default parameter values

- $\beta_1 : Hyper\ parameter = 0.9$
  - Exponential decay rate for the moment estimate
- $\beta_2 : Hyper\ parameter = 0.999$
  - Exponential decay rate for the second moment estimates
- $\eta : Learning\ Rate = 0.01$
- $\varepsilon : Small\ value\ to\ avoid\ dividing\ by\ zero = 1e - 08$

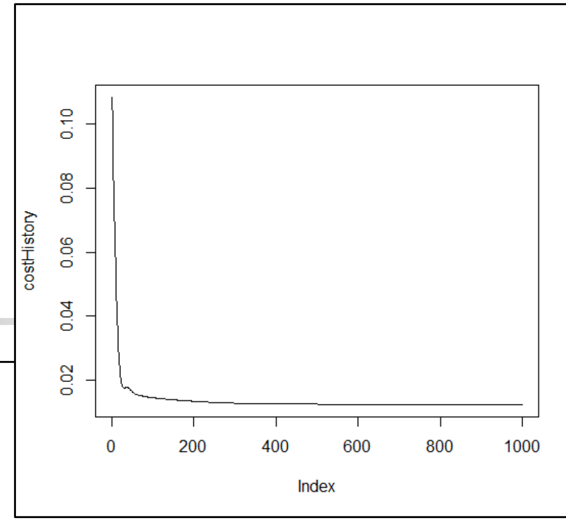
Adam Implementation

```
> for(i in 1:epochs) {
+     # 1. Computed output: h = x values * weight Matrix
+     h = X %*% t(W)
+
+     # 2. Loss = Computed output - observed output
+     loss = h - Y
+
+     error = (h - Y)^2
+     costHistory[i] = sum(error)/(2*nrow(Y))
+
+     # 3. gradient = loss * X
+     gradient = (t(loss) %*% X)/l
+
+     # 4. calculate new W weight values
+     m = beta1*m + (1 - beta1)*gradient
+     v = beta2*v + (1 - beta2)*(gradient^2)
+
+     # 5. corrected values Bias Correcttion
+     m_hat = m/(1 - beta1^i)
+     v_hat = v/(1 - beta2^i)
+
+     # 6. Update the weights
+     W = W - learningRate*(m_hat/(sqrt(v_hat) + epsilon))
+ }
```

$m_0 = 0$
$v_0 = 0$

- Adam
  - $m_t = \beta_1 * m_{t-1} + (1-\beta_1)\frac{\partial z}{\partial x}|_t$
  - $v_t = \beta_2 * v_{t-1} + (1-\beta_2)\left(\frac{\partial z}{\partial x}|_t\right)^2$
  - $w_t = w_{t-1} - \eta\frac{m_t}{\sqrt{v_t}+\varepsilon}$
- Adam Bias Correction
  - $\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$ $\qquad \widehat{v_t} = \frac{v_t}{1-\beta_2^t}$
  - $w_t = w_{t-1} - \eta\frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}+\varepsilon}$

Results




```
> print(W)
            [,1]      [,2]      [,3]      [,4]
[1,] -0.0152272 0.696738 0.4110761 0.2406706
>
> plot(costHistory,type='l')
```

$Regression\ Equation$: R 'lm' function
$strength = -0.01533 + 0.69688 * cement + 0.41118 * slag + 0.24075 * ash$

$Regression\ Equation$: R Matrix Approach
$strength = -0.01533 + 0.69688 * cement + 0.41118 * slag + 0.24075 * ash$

$Regression\ Equation$: R Adam
$strength = -0.01522 + 0.6967 * cement + 0.41107 * slag + 0.2406 * ash$

Summary: Part 2
OC R User’s Group: April 29, 2019

- Title: Deep Learning Optimization Techniques
  - Gradient Descent
  - Momentum
  - Nesterov Momentum
  - AdaGrad
  - RMSProp
  - Adam: Adaptive Moments