Deep Learning Using TensorFlow

Dr. Ash Pahwa

Section 3.3: Implementation of Adam in R

Outline

- Regression: lm
- Regression: Matrix Approach
- Regression: Adam: Adaptive Moment Estimation

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
- $\text{Expectation}(m_t) = \text{Expectation}(\textit{distribution of derivative})$
- $Expectation(v_t) = \text{Expectation}(\textit{distribution of square of derivative})$
- 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
  - $m_t = \beta_1 * m_{t-1} + (1-\beta_1)\frac{\partial z}{\partial x}|_t$
  - $v_t = \beta_2 * v_{t-1} + (1-\beta_2)\left(\frac{\partial z}{\partial x}|_t\right)^2$
  - $\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$ $\widehat{v_t} = \frac{v_t}{1-\beta_2^t}$
  - $w_t = w_{t-1} - \eta \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}+\varepsilon}$

TensorFlow Default Parameter Values

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
- $\eta: Learning\ Rate = 0.001$
- $\varepsilon: Small\ value\ to\ avoid\ dividing\ by\ zero = 1e - 08$

Regression: lm Implementation in R

Data Set: Concrete

```
> ###################################################
> # Data set
> # Read data
> #
> concrete <- read.csv("concrete.csv")
> head(concrete)
  cement  slag   ash water superplastic coarseagg fineagg age strength
1  141.3 212.0   0.0 203.5          0.0     971.8   748.5  28    29.89
2  168.9  42.2 124.3 158.3         10.8    1080.8   796.2  14    23.51
3  250.0   0.0  95.7 187.4          5.5     956.9   861.2  28    29.22
4  266.0 114.0   0.0 228.0          0.0     932.0   670.0  28    45.85
5  154.8 183.4   0.0 193.3          9.1    1047.4   696.7  28    18.29
6  255.0   0.0   0.0 192.0          0.0     889.8   945.0  90    21.86
> class(concrete)
[1] "data.frame"
>
```

Normalization: Scale the Data Between 0-1

```
> ####################################################
> # custom normalization function
> #
> normalize <- function(x) {
+   return((x - min(x)) / (max(x) - min(x)))
+ }
> # apply normalization to entire data frame
> concrete_norm <- as.data.frame(lapply(concrete, normalize))
```

```
> head(concrete_norm)
      cement      slag       ash     water superplastic coarseagg   fineagg        age  strength
1 0.08972603 0.5898720 0.0000000 0.6525559    0.0000000 0.4965116 0.3876066 0.07417582 0.3433412
2 0.15273973 0.1174179 0.6211894 0.2915335    0.3354037 0.8133721 0.5072755 0.03571429 0.2638595
3 0.33789954 0.0000000 0.4782609 0.5239617    0.1708075 0.4531977 0.6703462 0.07417582 0.3349944
4 0.37442922 0.3171953 0.0000000 0.8482428    0.0000000 0.3808140 0.1906673 0.07417582 0.5421702
5 0.12054795 0.5102949 0.0000000 0.5710863    0.2826087 0.7162791 0.2576518 0.07417582 0.1988290
6 0.34931507 0.0000000 0.0000000 0.5607029    0.0000000 0.2581395 0.8805820 0.24450549 0.2433038
> class(concrete_norm)
[1] "data.frame"
> dim(concrete_norm)
[1] 1030    9
>
```

Split Dataset into Train + Test
Select Predictor and Response Variables

```
> ######################################################
> # Split data into Train + Test
> #
> concrete_train <- concrete_norm[1:773, ]
> concrete_test <- concrete_norm[774:1030, ]
```

```
> concrete_train_x <- concrete_train[,1:3]
> concrete_train_y <- concrete_train[9]

> head(concrete_train_x)
      cement      slag       ash
1 0.08972603 0.5898720 0.0000000
2 0.15273973 0.1174179 0.6211894
3 0.33789954 0.0000000 0.4782609
4 0.37442922 0.3171953 0.0000000
5 0.12054795 0.5102949 0.0000000
6 0.34931507 0.0000000 0.0000000
> class(concrete_train_x)
[1] "data.frame"
> head(concrete_train_y)
   strength
1 0.3433412
2 0.2638595
3 0.3349944
4 0.5421702
5 0.1988290
6 0.2433038
```

$Regression\ Equation$:
$strength = \beta_0 + \beta_1 * cement + \beta_2 * slag + \beta_3 * ash$

Linear Regression Using 'lm' Function

$$\mathit{Regression\ Equation}:$$
$$\mathit{strength} = -0.01533 + 0.69688 * \mathit{cement} + 0.41118 * \mathit{slag} + 0.24075 * \mathit{ash}$$

```
> ########################################################
> # 1. Linear Regression using the 'lm' function
> d = cbind(concrete_train_x,concrete_train_y)
> class(d)
[1] "data.frame"
> #head(d)
> linearModel = lm (strength ~ cement + slag + ash, data = d)
> summary(linearModel)

Call:
lm(formula = strength ~ cement + slag + ash, data = d)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.48947 -0.11429 -0.00482  0.11255  0.50911 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -0.01533    0.02043   -0.75    0.453    
cement       0.69688    0.02945   23.67   <2e-16 ***
slag         0.41118    0.02837   14.49   <2e-16 ***
ash          0.24075    0.02227   10.81   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Regression: Matrix Approach Implementation in R

* Predictor Var Matrix: Add '1' in the First Column
* Convert Data into Matrix

```
> ########################################################
> # Extra Data Manipulation in Self Equation
> # Add 1 in the first column of 'concrete_train_x' data
> # Convert both 'concrete_train_x' and 'concrete_train_y' into matrix
> #
> concrete_train_x_m <- matrix(as.numeric(unlist(concrete_train_x)),ncol=3)
> concrete_train_x_m <- cbind(rep(1,nrow(concrete_train_x)),concrete_train_x_m)
> concrete_train_y_m <- matrix(as.numeric(unlist(concrete_train_y)),ncol=1)
> head(concrete_train_x_m)
     [,1]       [,2]      [,3]      [,4]
[1,]    1 0.08972603 0.5898720 0.0000000
[2,]    1 0.15273973 0.1174179 0.6211894
[3,]    1 0.33789954 0.0000000 0.4782609
[4,]    1 0.37442922 0.3171953 0.0000000
[5,]    1 0.12054795 0.5102949 0.0000000
[6,]    1 0.34931507 0.0000000 0.0000000
> dim(concrete_train_x_m)
[1] 773   4
> class(concrete_train_x_m)
[1] "matrix"
```

```
> head(concrete_train_y_m)
          [,1]
[1,] 0.3433412
[2,] 0.2638595
[3,] 0.3349944
[4,] 0.5421702
[5,] 0.1988290
[6,] 0.2433038
> dim(concrete_train_y_m)
[1] 773   1
> class(concrete_train_y_m)
[1] "matrix"
>
```

Regression: Using Matrix

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

$A = (X^T X)^{-1} X^T Y$

```
> ###########################################
> # 2. Linear Regression using Matrix approach
> #
> # z1 = Inverse of (t(x) * x)
> # z2 = t(x) * y
> # Answer = z1 * z2
> #
> z1 = t(concrete_train_x_m) %*% concrete_train_x_m
> z1
         [,1]      [,2]      [,3]      [,4]
[1,] 773.0000 313.45936 159.15526 203.06197
[2,] 313.4594 171.26421  51.08163  60.01172
[3,] 159.1553  51.08163  77.97358  23.04099
[4,] 203.0620  60.01172  23.04099 131.29828
> # comput the inverse of z
> det(z1)
[1] 71191603
> (invz1 = solve(z1))
            [,1]        [,2]        [,3]        [,4]
[1,]  0.01657913 -0.02084729 -0.01626535 -0.01325790
[2,] -0.02084729  0.03443838  0.01594181  0.01370368
[3,] -0.01626535  0.01594181  0.03195799  0.01226089
[4,] -0.01325790  0.01370368  0.01226089  0.01970545
```

```
> ################
> z2 = t(concrete_train_x_m) %*%
concrete_train_y_m
> z2
          [,1]
[1,] 320.92151
[2,] 149.99661
[3,]  70.76601
[4,]  79.79216
> ######################
> answer = invz1 %*% z2
> answer
            [,1]
[1,] -0.01533269
[2,]  0.69688220
[3,]  0.41118041
[4,]  0.24075367
>
```

Adam Algorithm Implementation in R

Adam

Adaptive Moment Estimation

Create X and Y Values Initialize Weight Vector

```
> ####################################################
> # 3. Regression Using Adam Algorithm
> #
> # W is the weight vector
> # Initialize the weight vector
> #
> W <- array(c(0.,0.,0.,0.),dim=c(1,4))
> X = concrete_train_x_m
> Y = concrete_train_y_m
>
```

Define Adam Hyper Parameters

$$m_0 = 0$$
$$v_0 = 0$$

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

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

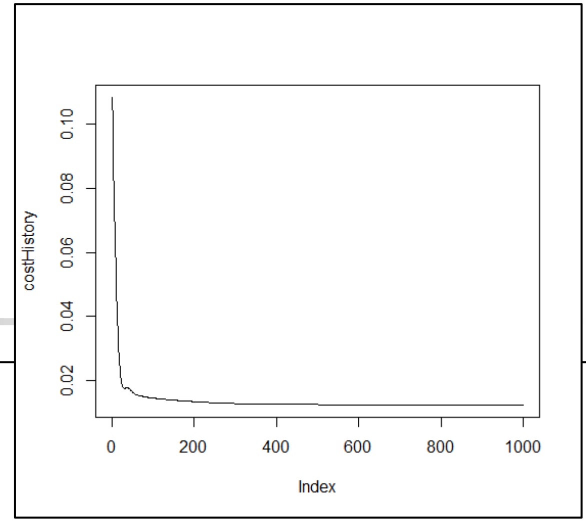
```
> for(i in 1:epochs) {
+     # 1. Computed output: h = x values * weight Matrix
+     h = X %*% t(W)
+
+     # 2. Loss = Computed output - observed output
+     loss = h - Y
+
+     error = (h - Y)^2
+     costHistory[i] = sum(error)/(2*nrow(Y))
+
+     # 3. gradient = loss * X
+     gradient = (t(loss) %*% 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 Correction
+     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
  - $\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$ $\widehat{v_t} = \frac{v_t}{1-\beta_2^t}$
  - $w_t = w_{t-1} - \eta\frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}+\varepsilon}$

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

- Regression: lm
- Regression: Matrix Approach
- Regression: Adam: Adaptive Moment Estimation