Deep Learning Using TensorFlow

Dr. Ash Pahwa

Section 3.2: AdaGrad + RMS Prop + Adam

Outline

- Gradient Descent
- AdaGrad
- RMS Prop
- Adam: Adaptive Moment Estimation

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

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

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: Adjusting the Learning rate (η)
  - AdaGrad
  - RMS Prop
  - Adam

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

Solution#3: Adjusting the Learning Rate η

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: Adjusting the Learning rate (η)
- AdaGrad
- RMS Prop
- Adam

Learning Rate η

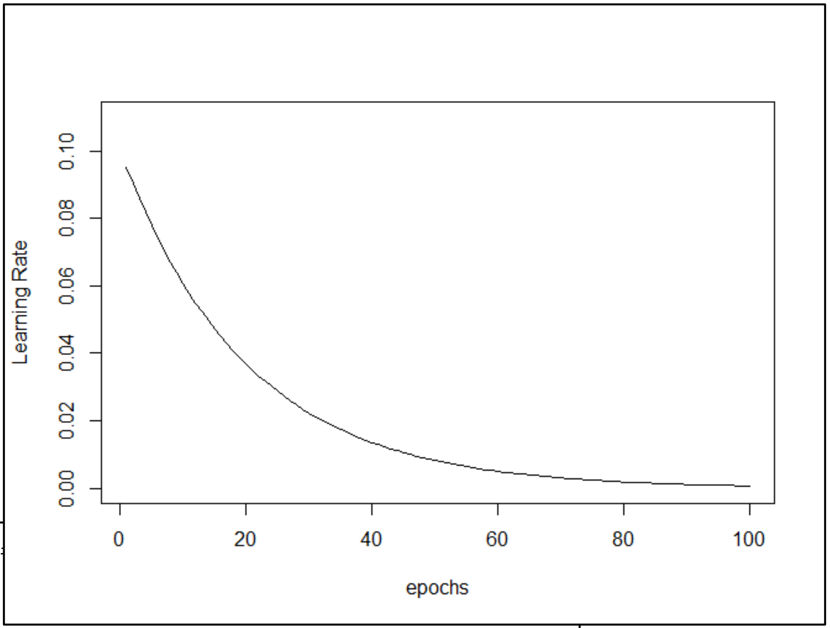
- How to decide the Learning Rate (LR)?
- If LR is small
  - Gradient will gently descend and we will get the optimum value of ‘x’ for which the error is minimum
  - But it may take a long time to converge
- If LR is large
  - NN will converge faster
  - But when we reach the destination “minimum” point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

First Solution
Create: Learning Rate (LR) Schedules

- Start with a large LR
  - 0.1
- Slowly reduce the size of LR
  - 0.01, 0.001, 0.0001
- When we reach close to the minimum point, reduce the LR even smaller
  - 0.00001

Second Solution
Create: Learning Rate (LR) Schedules

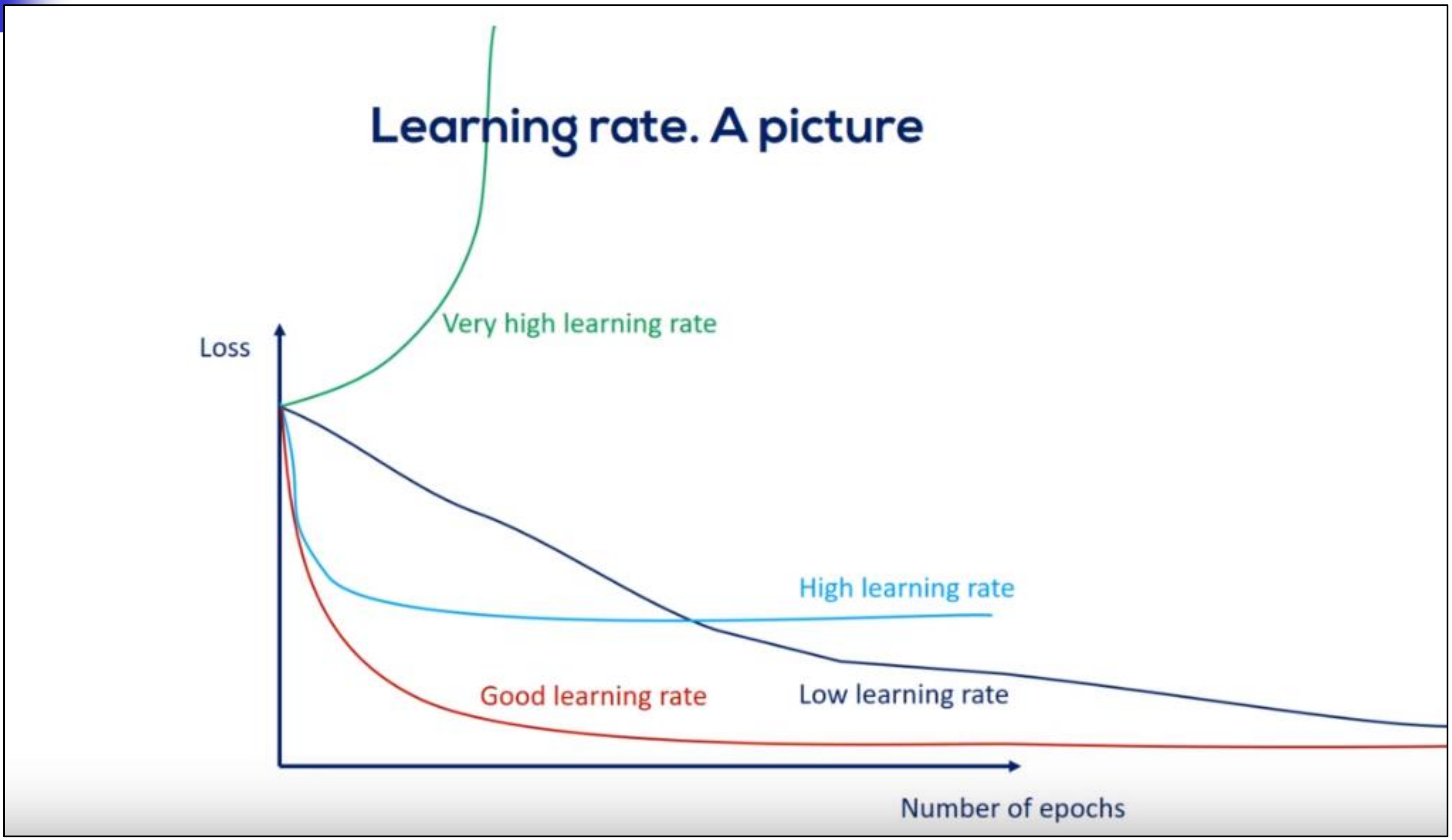
- $\eta_0 = 0.1$
- $\eta_n = \eta_0 e^{-\frac{n}{C}}$
  - $where\ n\ is\ the\ current\ epoch$
  - $And\ C\ is\ some\ constant = 20$
- Since we do not know how many epochs will take to converge
  - This may cause some problem



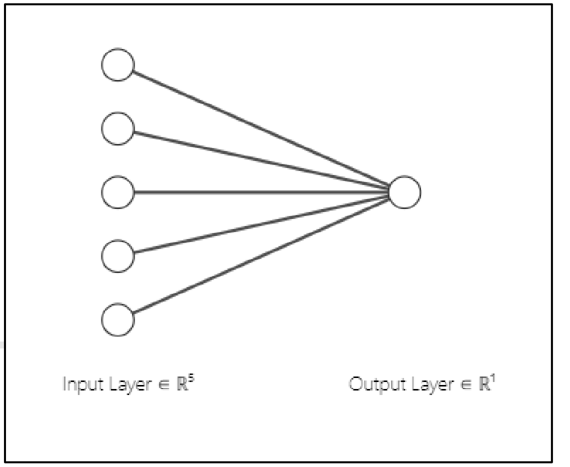

```
> ##############################################
> neta0 = 0.1
> C = 20
> epochs = seq(1,100,1)
> netas = neta0*exp(-epochs/C)
> head(netas)
[1] 0.09512294 0.09048374 0.08607080 0.08187308 0.07788008 0.07408182

> plot(netas,type='l',xlab="epochs", ylab="Learning Rate",ylim=c(0,0.11))
>
```

The Effect of Learning Rate on Conversions



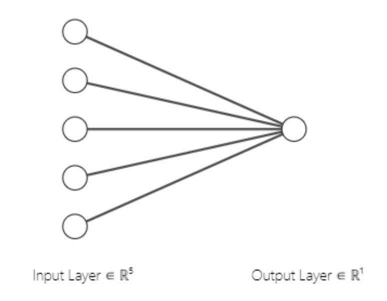
AdaGrad
Adaptive Gradient



- 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

$$1 \le i \le n$$

Where 'n' = number of weights

Learning Rate is Defined for Each Weight

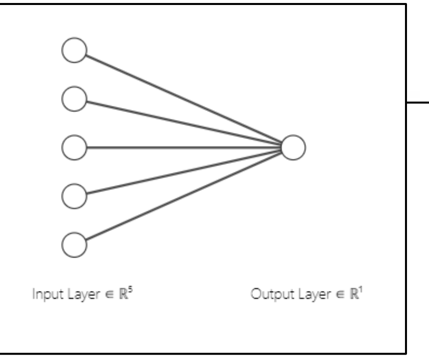
- Adaptive Learning Rate Schedule for each weight
- $1 \leq i \leq n$: where 'n' = number of weights
- Why separate learning rate for each weight?
  - Different weights do not reach their optimal value simultaneously

Difference: AdaGrad and RMS Prop

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

- 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

AdaGrad & RMS Prop

- Adaptive Gradient Algorithm (AdaGrad)
  - Maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems)
- Root Mean Square Propagation (RMS Prop)
  - Also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing)
  - This means the algorithm does well on online and non-stationary problems (e.g. noisy)

AdaGrad & RMS Prop

- The problem with AdaGrad is that eventually the learning rate will become so small that algorithm cannot move forward
- Solution : RMS Prop
  - $G_i^t = \beta * G_i^{t-1} + (1-\beta) * \left(\frac{\partial y}{\partial x_i}|_t\right)^2$
  - We decrease the Learning rate gradually: Assume: $\beta = 0.95$
    - $G_i^0 = 0$
    - $G_i^1 = G_i^0 * 0.95 + 0.05 * \left(\frac{\partial y}{\partial x_i}|_1\right)^2$
    - $G_i^2 = G_i^1 * 0.95 + 0.05 * \left(\frac{\partial y}{\partial x_2}|_2\right)^2$
- Therefore, RMS Prop solves the problem that AdaGrad has of very low learning rate

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.

Benefits of Adam on non-convex Optimization Problems

- Straightforward to Implement
- Computationally Efficient
- Little Memory Requirement
- Invariant to diagonal rescale of the gradients
- Well suited for problems that are large in terms of data and/or parameters
- Appropriate for non-stationary objectives
- Appropriate for problems with very noisy or sparse gradients
- Hyper-parameters have intuitive interpretation and typically require little tuning

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*

Advantages of Adam

- Adam combines the advantages of
  - Momentum
  - and RMS Prop.

- Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMS Prop,
  - Adam also makes use of the average of the second moments of the gradients

Bias Correction

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

Bias Correction

- The Adam algorithm works best when
  - $Expectation[m_t] = Expectation\left[\frac{\partial z}{\partial x}\right]$
  - To derive the expression for bias correction:
  - $\frac{\partial z}{\partial x}$ *is written as* g_t
  - --
  - $m_t = \beta_1 * m_{t-1} + (1 - \beta_1)g_t$
  - $m_0 = 0$
  - $m_1 = \beta * m_0 + (1 - \beta)g_1 \quad ; \qquad \beta_1 = \beta$
  - $m_2 = \beta(1 - \beta)g_1 + (1 - \beta)g_2$
  - $m_3 = \beta^2(1 - \beta)g_1 + \beta(1 - \beta)g_2 + (1 - \beta)g_3$
  - $m_3 = (1 - \beta)\sum_1^3(\beta^{3-i}\, g_i)$
  - $m_t = (1 - \beta)\sum_1^t(\beta^{t-i}\, g_i)$

Bias Correction

- $m_t = (1-\beta)\sum_1^t(\beta^{t-i}\, g_i)$
- $Expectation(m_t) = Expectation\left((1-\beta)\sum_1^t(\beta^{t-i}\, g_i)\right)$
- $E(m_t) = (1-\beta)\, E(g_i)\, \sum_1^t \beta^{t-i}$
- $E(m_t) = E(g_i)(1-\beta)(\beta^{t-1} + \beta^{t-2} + \cdots + \beta^0)$
- Add the geometric series
- $E(m_t) = E(g_i)(1-\beta)\frac{(1-\beta^t)}{(1-\beta)}$
- $E(g_i) = \frac{E(m_t)}{(1-\beta^t)}$
- $E(\widehat{m_t}) = \frac{E(m_t)}{(1-\beta^t)}$
- Similarly $E(\widehat{v_t}) = \frac{E(v_t)}{(1-\beta^t)}$

Adam Bias Correction

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

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

References:
AdaGrad + RMSProp + Adam

References I

L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *COLT*, page 257, 2010.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12:2121–2159, 2011.

E. Hazan, A. Agarwal, and S. Kale. Logarithmic regret algorithms for online convex optimization. *ML*, 69(2-3): 169–192, 2007.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

References:
AdaGrad + RMSProp + Adam

References II

G. Hinton, N. Srivastava, and K. Swersky. Lecture 6d - a separate, adaptive learning rate for each connection. Slides of Lecture Neural Networks for Machine Learning, 2012.

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

M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, 2003.

Which Optimization Method is the Best?

- Adam: Usually works best
- RMS Prop
- AdaGrad
- Stochastic Gradient Descent
- Mini Batch Gradient Descent
- Momentum
- Gradient Descent

Summary

- Gradient Descent
- AdaGrad
- RMS Prop
- Adam: Adaptive Moment Estimation