[機器學習ML NOTE]SGD, Momentum, AdaGrad, Adam Optimizer

GGWithRabitLIFE Follow
Aug 5, 2018 · 9 min read

> **我在練習實作mnist手寫辨識的時候，發現學習優化器(Optimizer)有許多種，因此去讀了一下各種不同優化器的比較，做個筆記，順便練習用tensorflow在簡單的方程式中把每種優化器的表現給呈現出來**

SGD-準確率梯度下降法 (stochastic gradient decent)

SGD 也就是最單純的gradient decent 方法，找出參數的梯度(利用微分的方法)，往梯度的方向去更新參數(weight)，即：

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

SGD Weight update equation

W 為權重(weight)參數，L 為損失函數(loss function)， η 是學習率(learning rate)， $\partial L/\partial W$ 是損失函數對參數的梯度(微分)

Momentum

Momentum 是「運動量」的意思，此優化器為模擬物理動量的概念，在同方向的維度上學習速度會變快，方向改變的時候學習速度會變慢。

> "一顆球從山上滾下來，在下坡的時候速度越來越快，遇到上坡，方向改變，速度下降"

$$V_t \leftarrow \beta V_{t-1} - \eta \frac{\partial L}{\partial W}$$

$$W \leftarrow W + V_t$$

Momentum Weight update equation

這裡多了一個 Vt 的參數，可以將他想像成「方向速度」 ，會跟上一次的更新有關，如果上一次的梯度跟這次同方向的話，$|Vt|$(速度)會越來越大(代表梯度增強)，W參數的更新梯度便會越來越快，如果方向不同，$|Vt|$便會比上次更小(梯度減弱)，W參數的更新梯度便會變小， β 可以想像成空氣阻力或是地面摩擦力，通常設定成0.9

AdaGrad

對於Optimizer來說，learning rate(學習率) η 相當的重要，太小會花費太多時間學習，太大有可能會造成overfitting，無法正確學習，前面幾種Optimizer的學習率 η，都為固定值，而AdaGrad就是會依照梯度去調整learning rate η 的優化器，Ada對我來說就是Adaptive的意思

$$W \leftarrow W - \eta \frac{1}{\sqrt{n+\epsilon}} \frac{\partial L}{\partial W}$$

$$n = \sum_{r=1}^{t} (\frac{\partial L_r}{\partial W_r})^2$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{\sum_{r=1}^{t} (\frac{\partial L_r}{\partial W_r})^2 + \epsilon}} \frac{\partial L}{\partial W}$$

AdaGrad Weight update equation

在AdaGrad Optimizer 中，η 乘上 $1/\sqrt{(n+\epsilon)}$ 再做參數更新，出現了一個n的參數，n為前面所有梯度值的平方和，利用前面學習的梯度值**平方和**來調整learning rate ，ϵ 為平滑值，加上 ϵ 的原因是為了不讓分母為0，ϵ 一般值為1e-8

- 前期梯度較小的時候，n較小，能夠放大學習率
- 後期梯度較大的時候，n較大，能夠約束學習率，但分母上梯度平方的累加會越來越大，會使梯度趨近於0，訓練便會結束，為了防止這個情況，後面有開發出 `RMSprop Optimizer` ，主要就是把n變成RMS(均方根)，這我在這邊就不多做說明了

Adam

Adam Optimizer 其實可以說就是把前面介紹的Momentum 跟 AdaGrad這二種Optimizer做結合，

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) \frac{\partial L_t}{\partial W_t}$$

$$v_t = \beta_1 v_{t-1} + (1-\beta_2) (\frac{\partial L_t}{\partial W_t})^2$$

像Momentum一樣保持了過去梯度的指數衰減平均值，像Adam一樣存了過去梯度的平方衰減平均值

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

對mt跟vt做偏離校正

$$W \leftarrow W - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Adam Weight update equation

Adam 保留了 Momentum 對過去梯度的方向做梯度速度調整與Adam對過去梯度的平方值做learning rate的調整，再加上Adam有做參數的”**偏離校正**”，使得每一次的學習率都會有個確定的範圍，會讓參數的更新較為平穩。

> Adam為目前較常使用的Optimizer

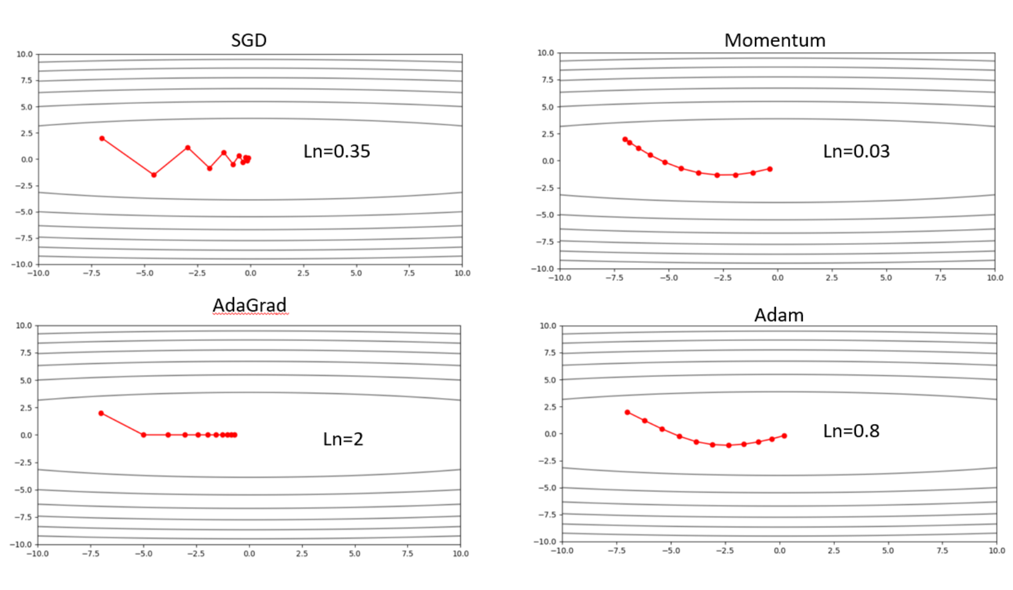
Using easy equation to implement each optimizer

我用Tensorflow去實作以下簡單的方程式在不同Optimizer所呈現的學習情況

$$f(x,y) = 0.5x^2 + 2.5y^2$$

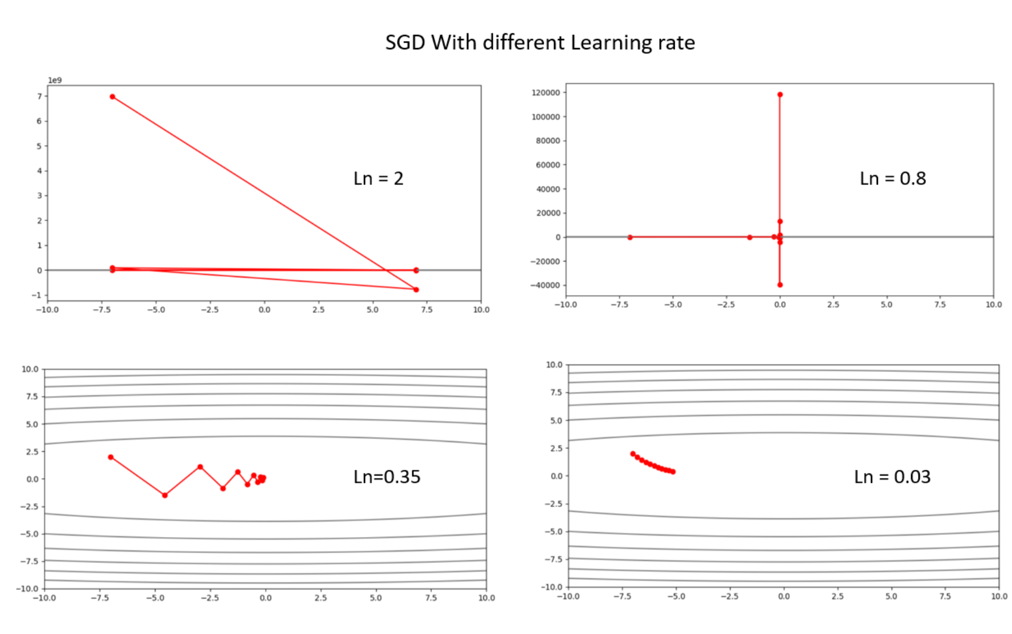
實作此簡易方程式，最小值為(x,y)=(0,0)

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

#--------Build tensorflow for equation 0.5*x^2+2.5*y^2
x = tf.Variable(-7.00000)
y = tf.Variable(2.00000)
a = tf.constant(0.5000)
b = tf.constant(2.50000)
mul1= tf.multiply(a,tf.square(x))
mul2= tf.multiply(b,tf.square(y))
output = tf.add(mul1,mul2)

#-------Chosing different optimizer and different learning rate
#train_step = tf.train.GradientDescentOptimizer(learning_rate =
0.35).minimize(output)
#train_step = tf.train.MomentumOptimizer(learning_rate = 0.03,
momentum=0.9).minimize(output)
#train_step = tf.train.AdagradOptimizer(learning_rate =
2).minimize(output)
#train_step = tf.train.AdamOptimizer(learning_rate =
0.8).minimize(output)

#-------Session start
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

#-------only do 10 step training
epochs = 10
epoch = 0
array_x = -7
xtt=[-7.0]
ytt=[2.0]
while epoch < epochs:
    print ("epoch of triaining", epoch)
    sess.run(train_step)
    if epoch % 1 == 0:
        array_x = x.eval(sess)
        array_y = y.eval(sess)
        xtt.append(array_x)
        ytt.append(array_y)
    epoch+=1
print(epoch)
print(xtt)
print(ytt)

#---------Plot Contour to display learning status for each optimizer
def obj_fun(x,y):
    z = (1/20)*(x**2)+1*(y**2)
    return z
delta = 0.1
x = np.arange(-10.0, 10.0, 0.01)
y = np.arange(-10.0, 10.0, 0.01)
X, Y = np.meshgrid(x, y)
Z = obj_fun(X,Y)
plt.figure(figsize = (10,5))
CS = plt.contour(X,Y,Z, colors = 'gray')
plt.plot(xtt, ytt, c='r')
for xt, yt in zip(xtt,ytt):
    plt.scatter(xt, yt , c='r')
plt.show()
```

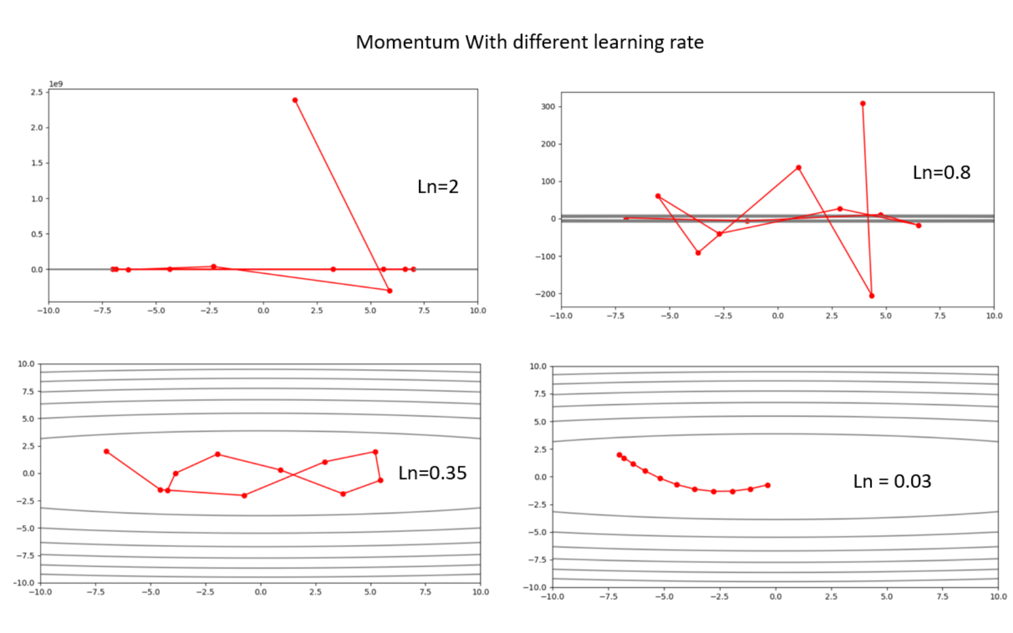


在Code上面需要自行修改要用那個Optimizer，而且我在使用每個Optimizer的時候，發現調整learning rate 非常重要，以下我也整理了每個Optimizer的調整狀況

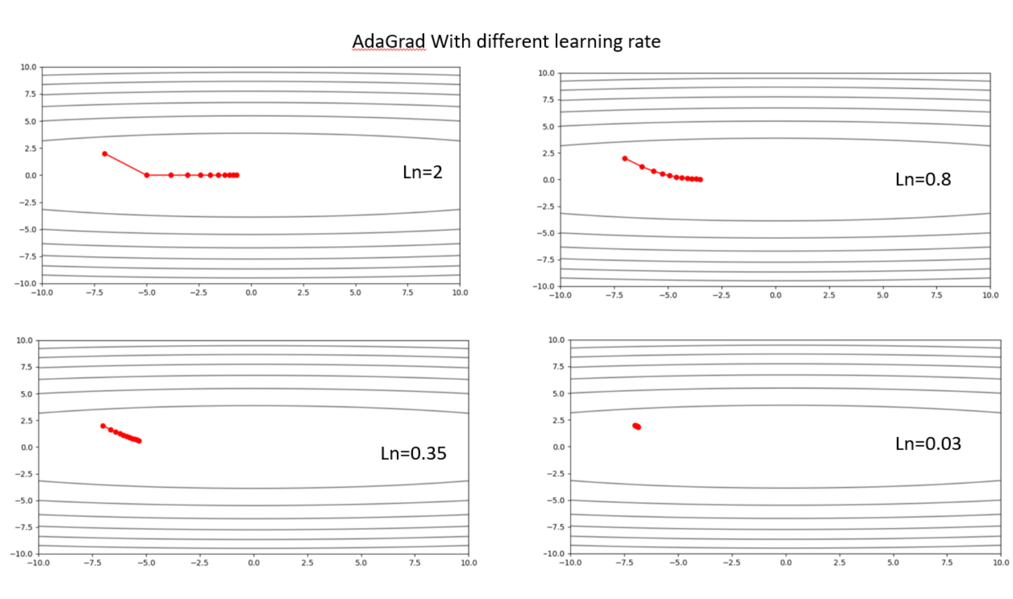
- SGD => 如果Ln 調整不好的話就會造成嚴重震盪造成參數調整出錯，從不同ln中看得出來SGD容易造成震盪，如果設定太小的話收斂會更慢



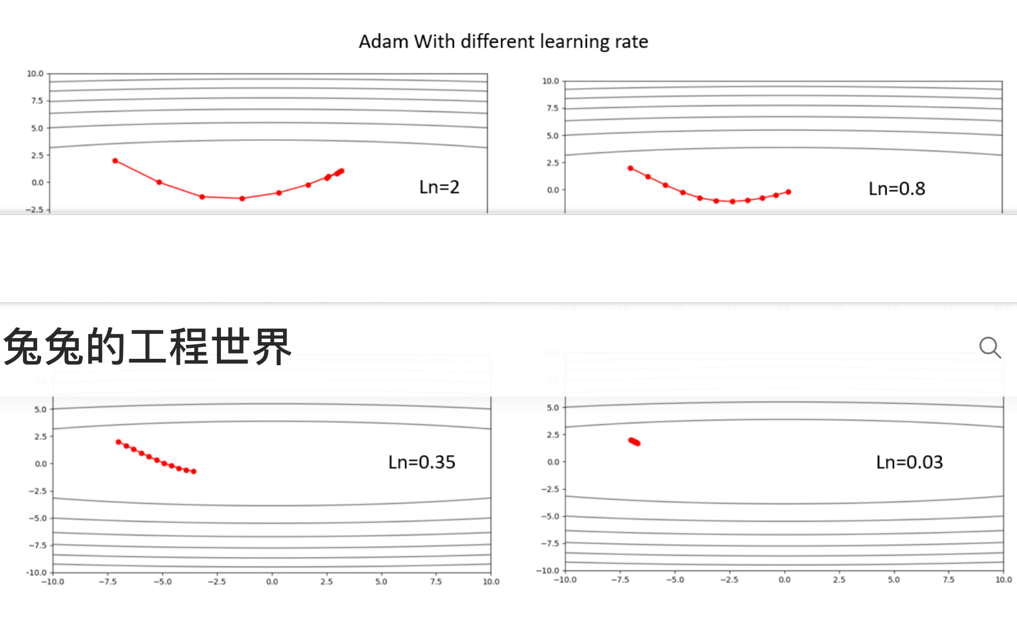
- Momentum =>Momentum也容易造成震盪，但並沒有SGD這麼嚴重，所以看得出來Momentum變得更smooth些，減少了「起伏」的程度



- AdaGrad =>看得出來 AdaGrad少了震盪的問題，如果設定ln太小的話收斂會變得超級慢。



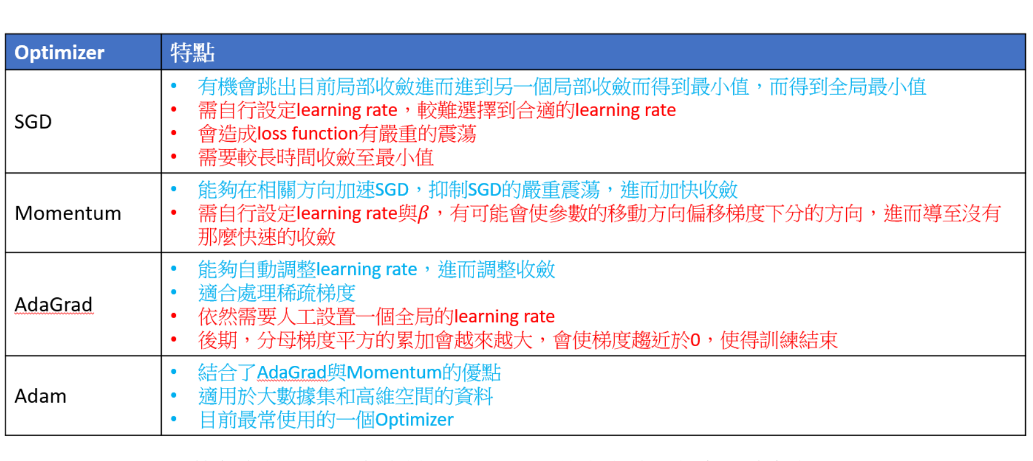
- Adam => 保留了AdaGrad的情況，相對於AdaGrad收斂得更快



The Medium App
An app designed for readers.
OPEN IN APP
雞雞與兔兔的工程世界 Sign in Get started
329

- 以上是用簡單的方程式對不同的優化器做出來的實驗跟觀察，至於要用那個優化器，還是得看不同的訓練資料去調整，不同的優化器各有特色，也有擅長與不擅長解決的問題，不過從上面的資料來看，**調整每個優化器的Ln是ML很重要的一個環節**。

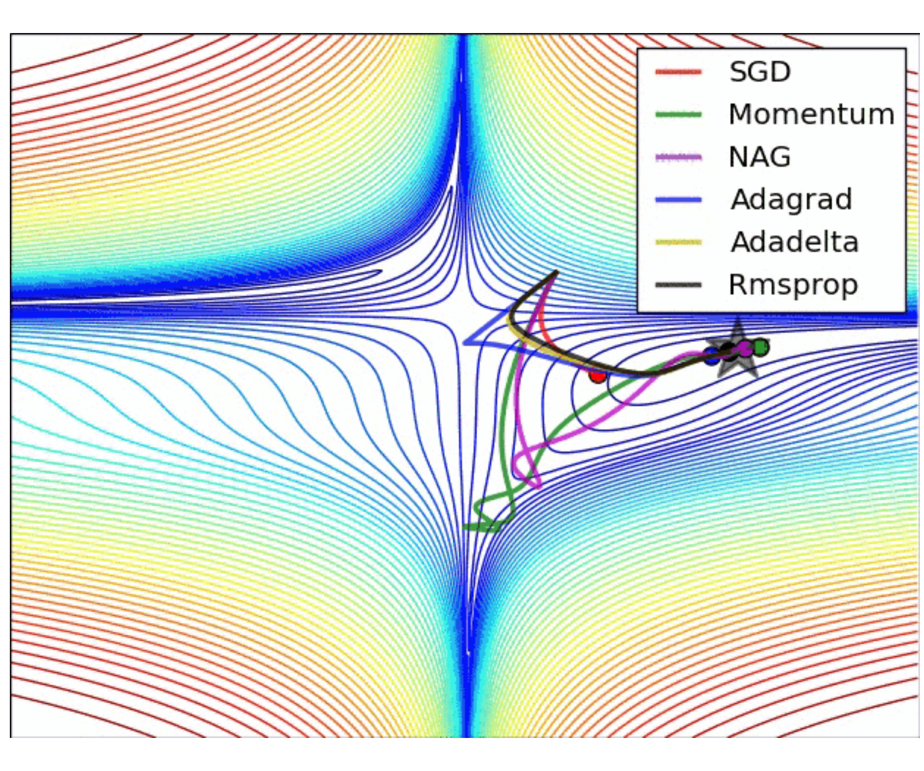
統整各個Optimizer

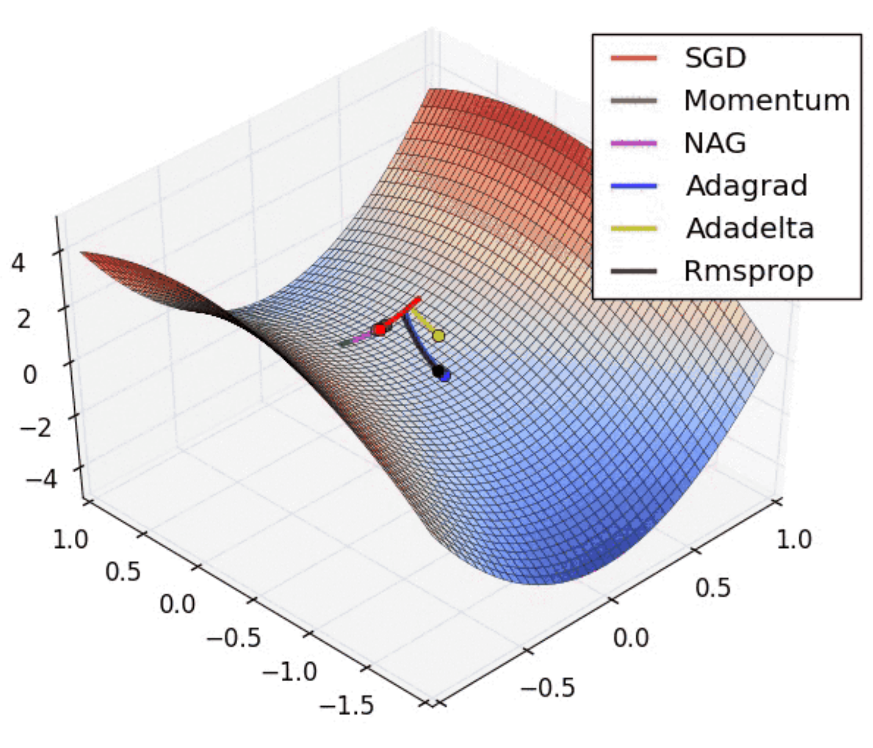


Optimizer	特點
SGD	• 有機會跳出目前局部收斂進而進到另一個局部收斂而得到最小值，而得到全局最小值 • 需自行設定learning rate，較難選擇到合適的learning rate • 會造成loss function有嚴重的震蕩 • 需要較長時間收斂至最小值
Momentum	• 能夠在相關方向加速SGD，抑制SGD的嚴重震蕩，進而加快收斂 • 需自行設定learning rate與β，有可能會使參數的移動方向偏移梯度下分的方向，進而導至沒有那麼快速的收斂
AdaGrad	• 能夠自動調整learning rate，進而調整收斂 • 適合處理稀疏梯度 • 依然需要人工設置一個全局的learning rate • 後期，分母梯度平方的累加會越來越大，會使梯度趨近於0，使得訓練結束
Adam	• 結合了AdaGrad與Momentum的優點 • 適用於大數據集和高維空間的資料 • 目前最常使用的一個Optimizer

藍色為優點，紅色為缺點，Adam目前大多適用於各種資料訓練

最後附上每個介紹Optimizer都會放的動畫圖，因為真的太厲害，而且簡單易懂，此二張動畫圖來自 Alec Radford 大神。





參考資料

Ruder’s Blog => http://ruder.io/optimizing-gradient-descent/

Deep Learning 書籍=>http://www.books.com.tw/products/0010761759

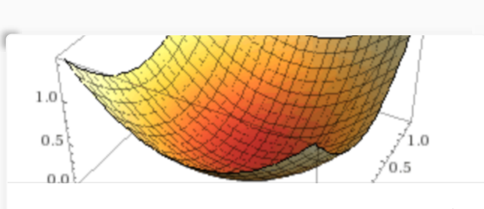
Joe’s Blog=>https://blog.csdn.net/u010089444/article/details/76725843

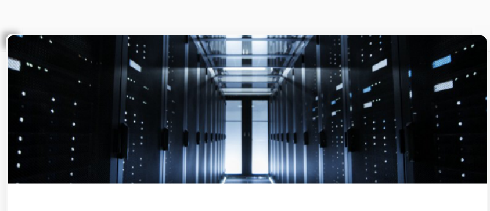
Machine Learning
Deep Learning
Gradient Descent

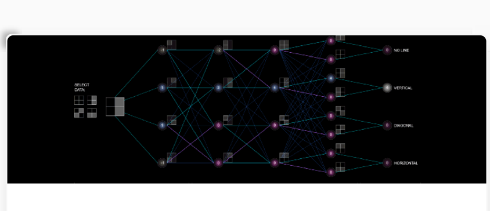
329 claps
2

GGWithRabitLIFE Follow
Share things to you, Machine learning, Life, Love

雞雞與兔兔的工程世界 Follow
Share things to you, Machine learning, Life, Love







Also tagged Gradient Descent
An Introduction to Gradient Descent
Yang S
May 13 · 8 min read
27

Related reads
Big Data in Healthcare
Gwynn Group
Aug 23, 2018 · 6 min re
101

Related reads
Toy Neural Network Classifies Orientation of Line
Monica Zhou
Jul 25, 2018 · 7 min reac
85

Responses

Show all responses