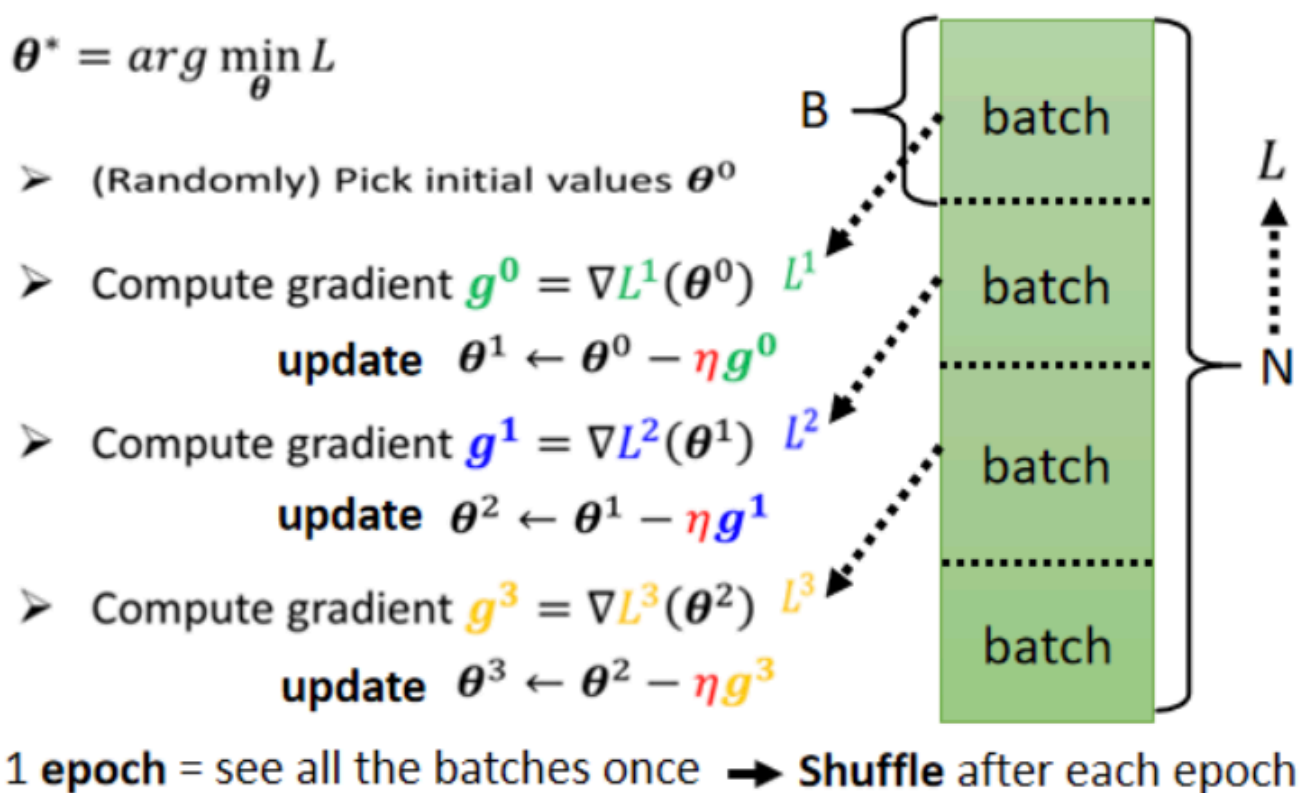


task05

Batch and Momentum



是把所有的 Data 分成一个一个的 Batch, 也称Mini Batch。

把所有的Batch看一遍, 叫做一个epoch

shuffle有很多不同做法, 常见的一个为在每一个epoch开始之前, 会分一次batch, 然后每一个epoch的batch都不一样。重新再分batch。

Small Batch v.s. Large Batch

Larger batch size does not require longer time to compute gradient

即比较大的 Batch Size,你要算 Loss,再进而算 Gradient,所需要的时间,不一定比小的 Batch Size 要花的时间长

MNIST 为大型手写数字数据库, 每张图含有0-9的数字。类似机器学习中的helloworld

Smaller batch requires longer time for one epoch




GPU 虽然有并行运算的能力,但它并行运算能力终究有个极限,所以Batch Size 真的很大的时候,时间还是会增加的。

但是因为有并行运算的能力,因此实际上,当 **Batch Size** 小的时候,跑完一个 **Epoch**,花的时间是比大的 **Batch Size** 还要多的

如果一个 Local Minima 它在一个峡谷里面,它是坏的 Minima,然后它在一个平原上,它是好的 Minima。

大的 **Batch Size**,会让我们倾向於走到峡谷里面,而小的 **Batch Size**,倾向於让我们走到盆地里面

但这只是一个解释,那也不是每个人都相信这个解释,那这个其实还是一个尚待研究的问题

	Small	Large
Speed for one update (no parallel)	Faster	Slower
Speed for one update (with parallel)	Same	Same (not too large)
Time for one epoch	Slower	Faster 
Gradient	Noisy	Stable
Optimization	Better 	Worse
Generalization	Better 	Worse

Momentum

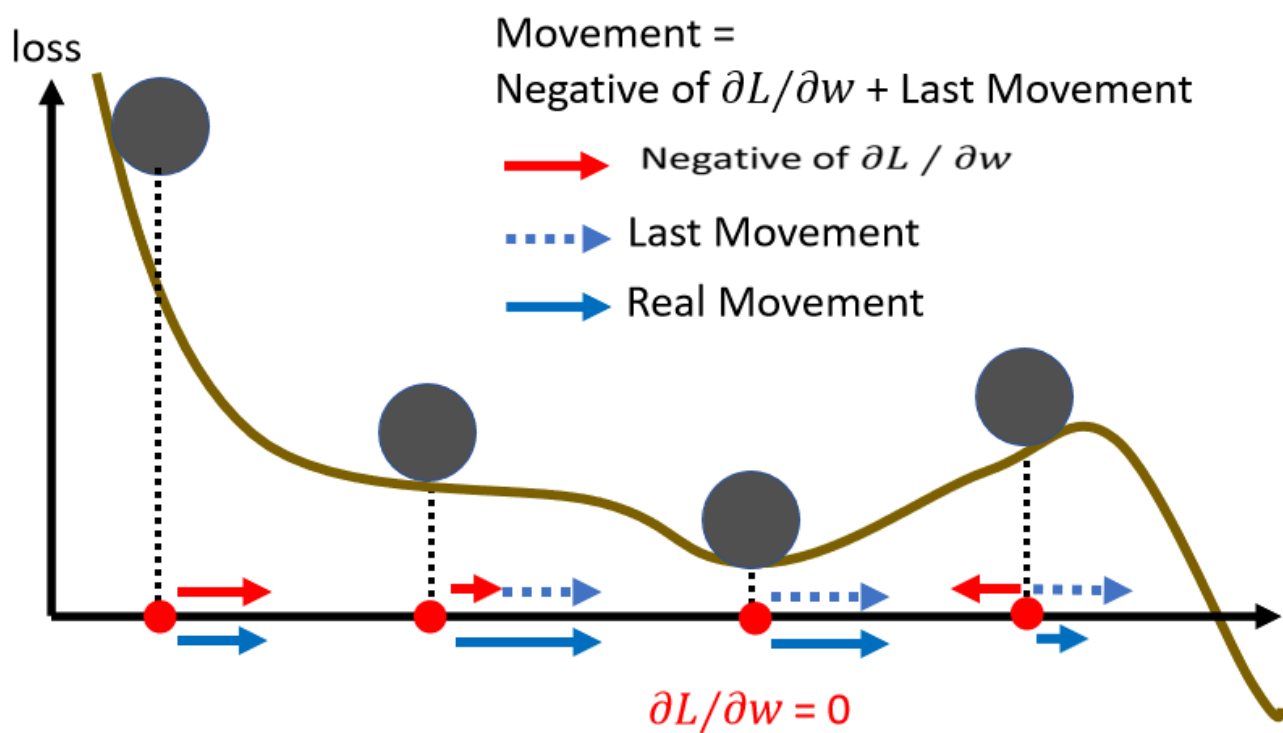
对抗Saddle Point或local Minima的技术。

在物理世界里,假设 Error Surface 就是真正的斜坡,而参数是一个球,把球从斜坡上滚下来,如果今天是 Gradient Descent,它走到 Local Minima 就停住了,走到 Saddle Point 就停住了

但是在物理的世界里,一个球如果从高处滚下来,从高处滚下来就算滚到 Saddle Point,如果有惯性,它从左边滚下来,因为惯性的关系它还是会继续往右走,甚至它走到一个 Local Minima,如果今天它的动量够大的话,它还是会继续往右走,甚至翻过这个小坡然后继续往右走。

所以今天在物理的世界里,一个球从高处滚下来的时候,它并不会被 Saddle Point,或 Local Minima卡住,不一定会被 Saddle Point,或 Local Minima 卡住,有没有办法运用这样子的概念,到 Gradient Descent 里面呢,那这个就是我们等一下要讲的 Momentum 这个技术。

加上 Momentum 以后,每一次移动参数时,不是只往 Gradient 的反方向来移动参数,是 **Gradient** 的反方向,加上前一步移动的方向,两者加起来的结果,去调整去到参数,



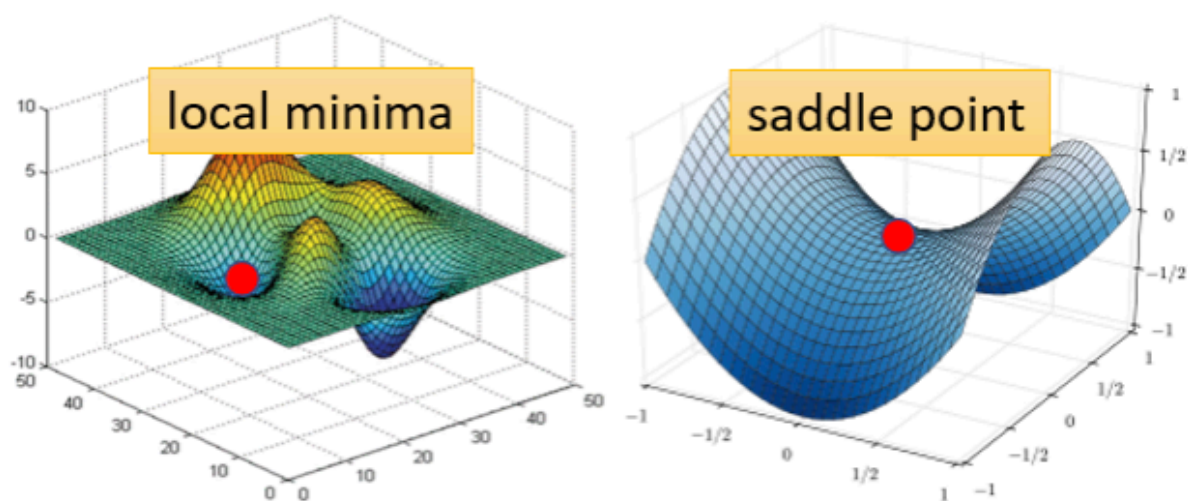
When gradient is small

Critical Point

训练失败的原因：

随着参数不断的更新，但training 的loss不会再下降，可以猜到这个地方参数对loss的微分为0，gradient descent就没有办法更新，loss就不再下降。

讲到gradient为0的时候，可能最先想到的是local minima或 local maxima，但是也可能是saddle point。



那怎么知道是卡在local minima 还是saddle point呢？

- 因为如果是卡在local minima，那可能没有路可以走了。
- 但saddle point旁边是还有路可以走的，只要逃离saddle point，就可能使loss 更低

$L(\theta)$ around $\theta = \theta'$ can be approximated below

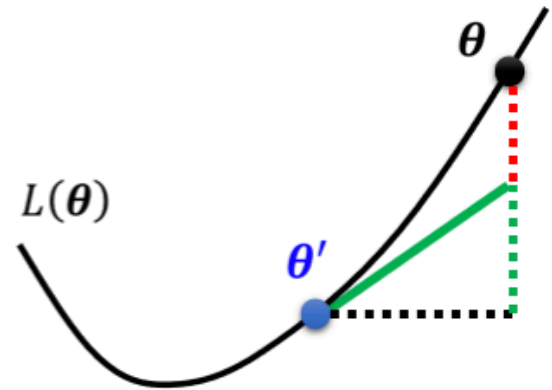
$$L(\theta) \approx L(\theta') + (\theta - \theta')^T \mathbf{g} + \frac{1}{2} (\theta - \theta')^T \mathbf{H} (\theta - \theta')$$

Gradient \mathbf{g} is a vector

$$\mathbf{g} = \nabla L(\theta') \quad g_i = \frac{\partial L(\theta')}{\partial \theta_i}$$

Hessian \mathbf{H} is a matrix

$$H_{ij} = \frac{\partial^2}{\partial \theta_i \partial \theta_j} L(\theta')$$



线性代数理论上,如果对所有的 v 而言, $v^T H v$ 都大于零,那这种矩阵叫做**positive definite** 正定矩阵,positive definite的矩阵,它所有的eigen value特征值都是正的

At critical point:

$$\text{Hessian} \quad L(\theta) \approx L(\theta') + \frac{1}{2} (\theta - \theta')^T \mathbf{H} (\theta - \theta')$$

For all v

$$v^T \mathbf{H} v > 0 \implies \text{Around } \theta': L(\theta) > L(\theta') \implies \text{Local minima}$$

$$= \mathbf{H} \text{ is positive definite} = \text{All eigen values are positive.} \uparrow$$

For all v

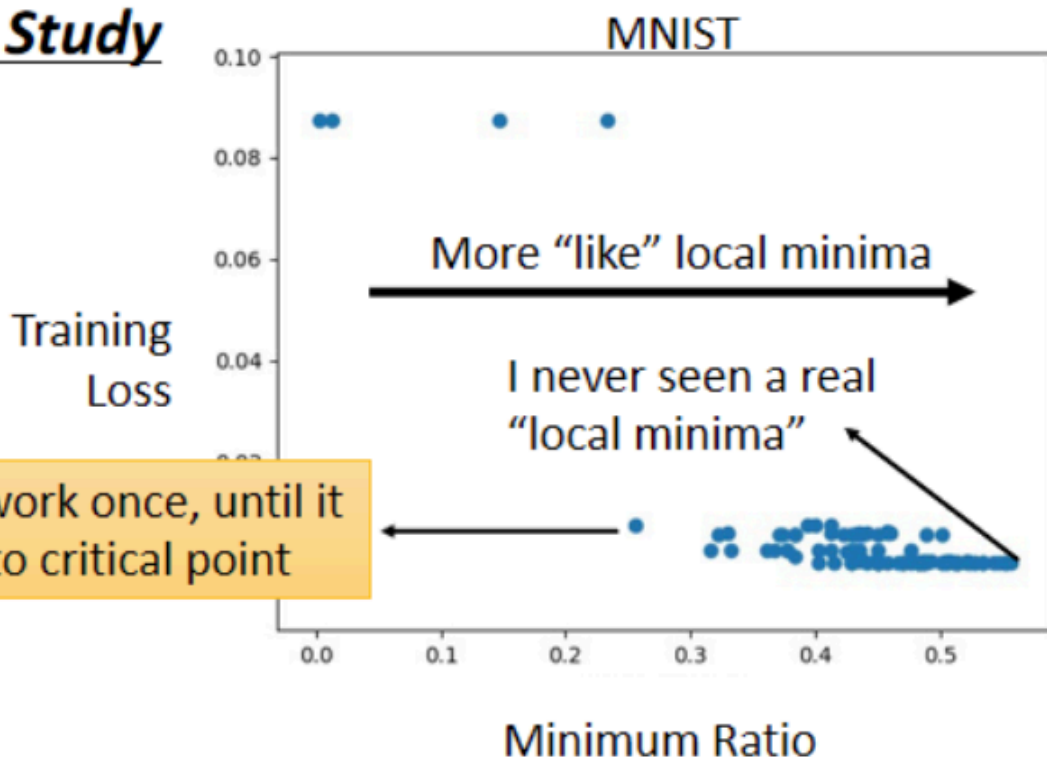
$$v^T \mathbf{H} v < 0 \implies \text{Around } \theta': L(\theta) < L(\theta') \implies \text{Local maxima}$$

$$= \mathbf{H} \text{ is negative definite} = \text{All eigen values are negative.} \uparrow$$

$$\text{Sometimes } v^T \mathbf{H} v > 0, \text{ sometimes } v^T \mathbf{H} v < 0 \implies \text{Saddle point}$$

Some eigen values are positive, and some are negative. \uparrow

Empirical Study

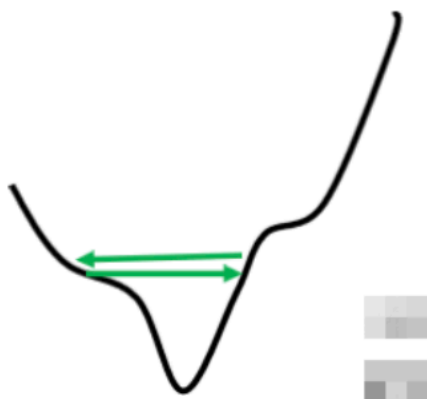


$$\text{Minimum ratio} = \frac{\text{Number of Positive Eigen values}}{\text{Number of Eigen values}}$$

从经验上看起来,其实local minima并没有那么常见,多数的时候,你觉得你train到一个地方,你gradient真的很小,然后所以你的参数不再update了,往往是因为你卡在了一个saddle point。

Tips for training: Adaptive Learning Rate

当loss不再下降时,不一定是卡到了critical point、saddle point、local minima。它的梯度gradient仍然很大,只是loss不见得再小了。gradient可能在error surface山谷的两个谷壁间,不断的来回的震荡。



更感觉走到一个critical point,其实是困难的一件事,多数时候training,在还没有走到critical point的时候,就已经停止了,那这并不代表说,critical point不是一个问题,只是想要说,当你用gradient descend,来做optimization的时候,你真正应该要怪罪的对象,往往不是critical point,而是其他的原因,

Training can be difficult even without critical points

learning rate设太大了会导致在山壁的两端不断震荡。但设太小,学习率实在太小,无法使得训练有效前进

Different parameters needs different learning rate

如果在某一个方向上,我们的gradient的值很小,非常的平坦,那我们会希望learning rate调大一点,如果在某一个方向上非常的陡峭,坡度很大,那我们其实期待,learning rate可以设得小一点

我们要改一下,gradient descend原来的式子:

$$\theta_i^{t+1} \leftarrow \theta_i^t - \eta g_i^t$$

但我们需要有随着参数变化的learning rate, 将 η 改成 $\frac{\eta}{\sigma_i^t}$

$$\theta_i^{t+1} \leftarrow \theta_i^t - \frac{\eta}{\sigma_i^t} g_i^t$$

不同的参数我们要给它不同的 σ ,同时它也是iteration dependent的,不同的iteration我们也会有不同的 σ

Root mean square

这个 σ 有什么样的方式,可以把它计算出来呢,一个常见的类型是算,gradient的Root Mean Square

Root Mean Square

$$\theta_i^{t+1} \leftarrow \theta_i^t - \boxed{\frac{\eta}{\sigma_i^t}} g_i^t$$

$$\theta_i^1 \leftarrow \theta_i^0 - \frac{\eta}{\sigma_i^0} g_i^0 \quad \sigma_i^0 = \sqrt{(g_i^0)^2} = |g_i^0|$$

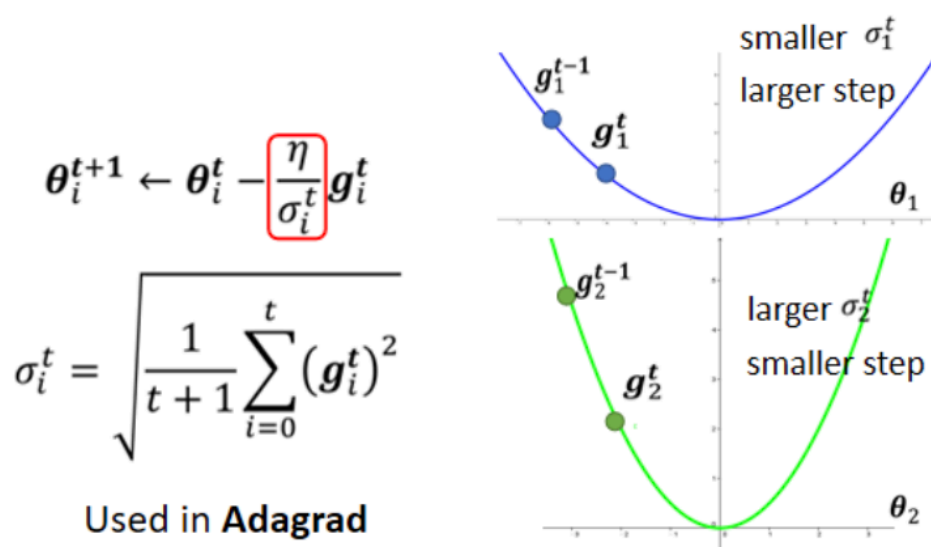
$$\theta_i^2 \leftarrow \theta_i^1 - \frac{\eta}{\sigma_i^1} g_i^1 \quad \sigma_i^1 = \sqrt{\frac{1}{2} [(g_i^0)^2 + (g_i^1)^2]}$$

$$\theta_i^3 \leftarrow \theta_i^2 - \frac{\eta}{\sigma_i^2} g_i^2 \quad \sigma_i^2 = \sqrt{\frac{1}{3} [(g_i^0)^2 + (g_i^1)^2 + (g_i^2)^2]}$$

$$\vdots$$

$$\theta_i^{t+1} \leftarrow \theta_i^t - \frac{\eta}{\sigma_i^t} g_i^t \quad \sigma_i^t = \sqrt{\frac{1}{t+1} \sum_{i=0}^t (g_i^t)^2}$$

Adagrad



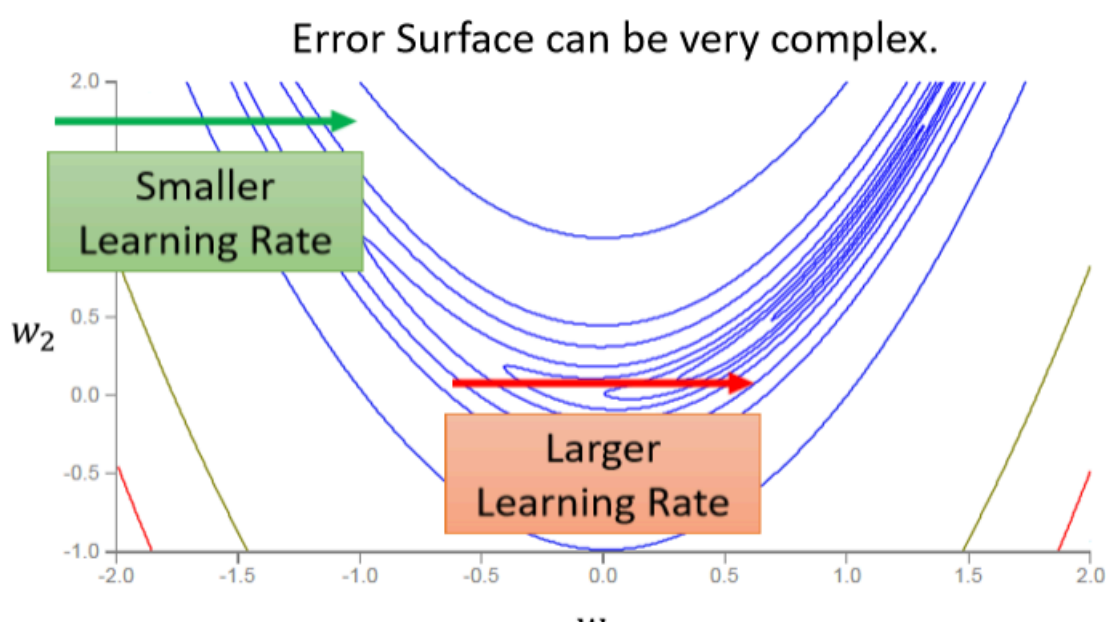
你可以想像说,现在有两个参数:一个叫 θ_1 一个叫 θ_2 θ_1 坡度小 θ_2 坡度大

- θ_1 因为它坡度小,所以你在 θ_1 这个参数上面,算出来的gradient值都比较小
- 因为gradient算出来的值比较小,然后这个 σ 是gradient的平方和取平均再开根号
- 所以算出来的 σ 就小, σ 小 learning rate就大

反过来说 θ_2 是一个比较陡峭的参数,在 θ_2 这个方向上loss的变化比较大,所以算出来的gradient都比较大,你的 σ 就比较大,你在update的时候 你的step,你的参数update的量就比较小

所以有了 σ 这一项以后,你就可以随著gradient的不同,每一个参数的gradient的不同,来自动的调整learning rate的大小,那这个并不是,你今天会用的最终极的版本,

RMSProp



如果考虑横轴,考虑左右横的水平线的方向的话,在绿色箭头这个地方坡度比较陡峭,所以需要比较小的learning rate,

但是走到了中间这一段，到了红色箭头的时候呢，坡度又变得平滑了起来，平滑了起来就需要比较大的**learning rate**，所以就算是同一个参数同一个方向，我们也期待说，**learning rate**是可以动态的调整的，于是就有了一个新的招数，这个招数叫做RMS Prop

RMS Prop这个方法有点传奇，它传奇的地方在于它找不到论文，非常多年前应该是将近十年前，Hinton在Coursera上，开过deep learning的课程，那个时候他在他的课程里面，讲了RMS Prop这个方法，然后这个方法没有论文，所以你要cite的话，你要cite那个影片的连接，这是个传奇的方法叫做RMS Prop

$$\begin{aligned}
 &\text{RMSProp} & \theta_i^{t+1} &\leftarrow \theta_i^t - \frac{\eta}{\sigma_i^t} g_i^t \\
 &\theta_i^1 \leftarrow \theta_i^0 - \frac{\eta}{\sigma_i^0} g_i^0 & \sigma_i^0 &= \sqrt{(g_i^0)^2} & 0 < \alpha < 1 \\
 &\theta_i^2 \leftarrow \theta_i^1 - \frac{\eta}{\sigma_i^1} g_i^1 & \sigma_i^1 &= \sqrt{\alpha(\sigma_i^0)^2 + (1 - \alpha)(g_i^1)^2} \\
 &\theta_i^3 \leftarrow \theta_i^2 - \frac{\eta}{\sigma_i^2} g_i^2 & \sigma_i^2 &= \sqrt{\alpha(\sigma_i^1)^2 + (1 - \alpha)(g_i^2)^2} \\
 &\vdots \\
 &\theta_i^{t+1} \leftarrow \theta_i^t - \frac{\eta}{\sigma_i^t} g_i^t & \sigma_i^t &= \sqrt{\alpha(\sigma_i^{t-1})^2 + (1 - \alpha)(g_i^t)^2}
 \end{aligned}$$

Adam

今天最常用的optimization的策略，有人又叫做optimizer，就是Adam

Adam: RMSProp + Momentum

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t .

Require: α : Stepsize

Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates

Require: $f(\theta)$: Stochastic objective function with parameters θ

Require: θ_0 : Initial parameter vector

$m_0 \leftarrow 0$ (Initialize 1st moment vector) \rightarrow for momentum

$v_0 \leftarrow 0$ (Initialize 2nd moment vector) \rightarrow for RMSprop

$t \leftarrow 0$ (Initialize timestep)

while θ_t not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)

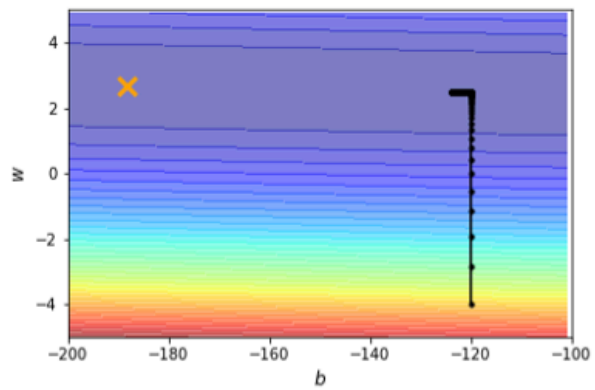
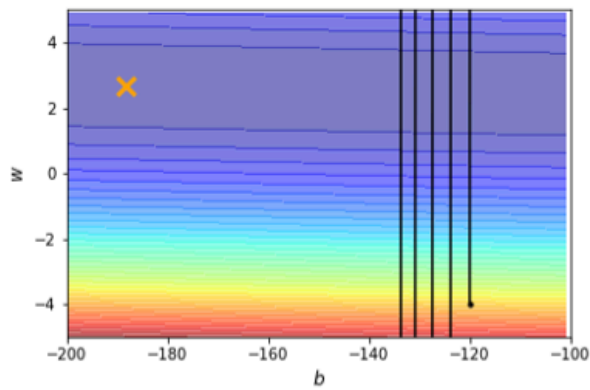
end while

return θ_t (Resulting parameters)

11

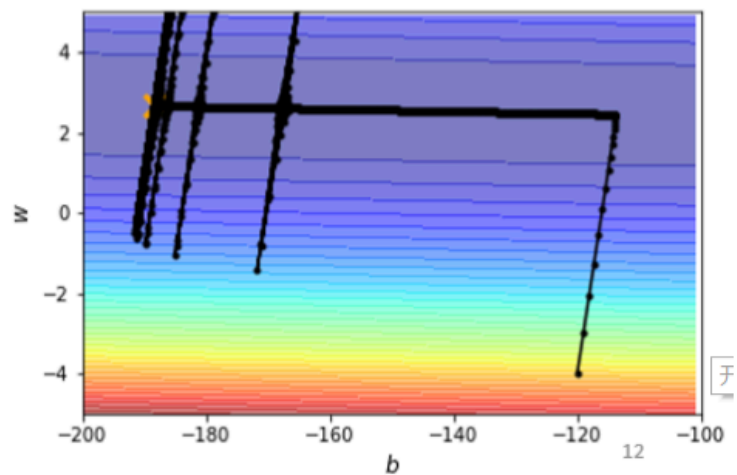
Adam就是RMS Prop加上Momentum, 原始论文<https://arxiv.org/pdf/1412.6980.pdf>

Without Adaptive Learning Rate



$$\theta_i^{t+1} \leftarrow \theta_i^t - \boxed{\frac{\eta}{\sigma_i^t}} g_i^t$$

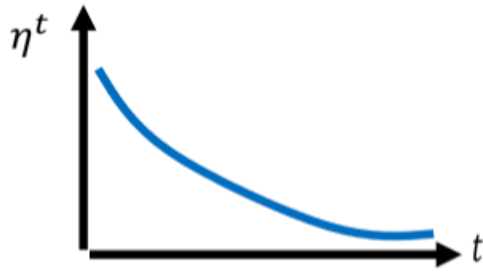
$$\sigma_i^t = \sqrt{\frac{1}{t+1} \sum_{i=0}^t (g_i^t)^2}$$



什么是learning rate的scheduling呢

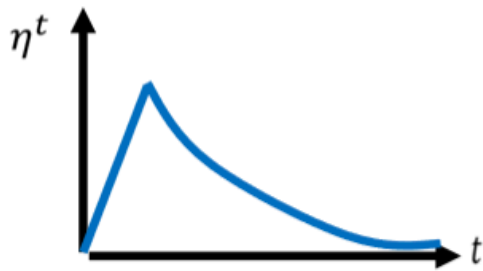
Learning Rate Scheduling

$$\theta_i^{t+1} \leftarrow \theta_i^t - \frac{\eta^t}{\sigma_i^t} g_i^t$$



Learning Rate Decay

After the training goes, we are close to the destination, so we reduce the learning rate.



Warm Up

Increase and then decrease?

At the beginning, the estimate of σ_i^t has large variance.

Please refer to **RAdam** <https://arxiv.org/abs/1908.03265>

这个 η 是一个固定的值,learning rate scheduling的意思就是说,我们不要把 η 当一个常数,我们把它跟时间有关

最常见的策略叫做Learning Rate Decay,也就是说, 随著时间的不断地进行,随著参数不断的update,我们这个 η 让它越来越小

σ 告诉我们,某一个方向它到底有多陡,或者是多平滑,那这个统计的结果,要看得够多笔数据以后,这个统计才精準,所以一开始我们的统计是不精準的

一开始我们的 σ 是不精準的,所以我们一开始不要让我们的参数,走离初始的地方太远,先让它在初始的地方呢,做一些像是探索这样,所以一开始learning rate比较小,是让它探索 收集一些有关error surface的情报,先收集有关 σ 的统计数据,等 σ 统计得比较精準以后,在让learning rate慢慢地爬升

Summary of Optimization

所以我们从最原始的gradient descent,进化到这一个版本

(Vanilla) Gradient Descent

$$\theta_i^{t+1} \leftarrow \theta_i^t - \eta g_i^t$$

Various Improvements

$$\theta_i^{t+1} \leftarrow \theta_i^t - \frac{\eta^t}{\sigma_i^t} m_i^t \cdots \rightarrow \text{Momentum: weighted sum of the previous gradients}$$

这个版本里面

- 我们有Momentum,也就是说我们现在,不是完全顺著gradient的方向,现在不是完全顺著这一个时间点,算出来的gradient的方向,来update参数,而是把过去,所有算出来gradient的方向,做一个加总当作update的方向,这个是momentum
- 接下来应该要update多大的步伐呢,我们要除掉,gradient的Root Mean Square

(Vanilla) Gradient Descent

$$\theta_i^{t+1} \leftarrow \theta_i^t - \eta g_i^t$$

Various Improvements

$$\theta_i^{t+1} \leftarrow \theta_i^t - \frac{\eta^t}{\sigma_i^t} m_i^t \cdots \rightarrow \text{Momentum: weighted sum of the previous gradients}$$

root mean square of the gradients

Consider direction

only magnitude

17

那讲到这边可能有同学会觉得困惑,这一个momentum是考虑,过去所有的gradient,这个 σ 也是考虑过去所有的gradient,一个放在分子一个放在分母,都考虑过去所有的gradient,不就是正好抵销了吗,

但是其实这个Momentum跟这个 σ ,它们在使用过去所有gradient的方式是不一样的,**Momentum是直接把所有的gradient通通都加起来**,所以它有考虑方向,它有考虑gradient的正负号,它有考虑gradient是往左走还是往右走

但是这个**Root Mean Square**,它就不考虑gradient的方向了,它只考虑gradient的大小,记不记得我们在算 σ 的时候,我们都要取平方项,我们都要把gradient取一个平方项,我们是把平方的结果加起来,所以我们只考虑gradient的大小,不考虑它的方向,所以Momentum跟这个 σ ,算出来的结果并不会互相抵销

- 那最后我们还会加上,一个learning rate的scheduling,

(Vanilla) Gradient Descent

$$\theta_i^{t+1} \leftarrow \theta_i^t - \eta g_i^t$$

Various Improvements

$$\theta_i^{t+1} \leftarrow \theta_i^t - \frac{\eta^t}{\sigma_i^t} m_i^t$$

Learning rate scheduling

Momentum: weighted sum of the previous gradients

Consider direction

root mean square of the gradients

only magnitude

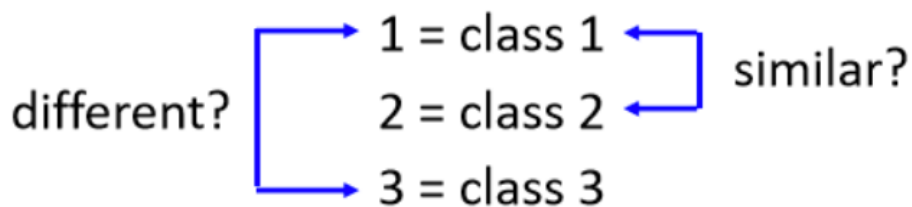
17

Classification

Classification as Regression?

分类是怎么做的呢,Regression就是输入一个向量,然后输出一个数值,我们希望输出的数值跟某一个label,也就是我们要学习的目标,越接近越好,如果是正确的答案就有加Hat,Model的输出没有加Hat

输入一个东西以后,我们的输出仍然是一个scaler,它叫做y 然后这一个y,我们要让它跟正确答案,那个Class越接近越好,但是y是一个数字,我们怎么让它跟Class越接近越好呢,我们必须把Class也变成数字



class只是命名的关系,但是实际并不相似。

Class as one-hot vector

$$\hat{y} = \begin{matrix} \text{Class 1} \\ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \end{matrix} \quad \text{or} \quad \begin{matrix} \text{Class 2} \\ \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \end{matrix} \quad \text{or} \quad \begin{matrix} \text{Class 3} \\ \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{matrix}$$

用One-hot vector来表示的话,就没有说Class1跟Class2比较接近,Class1跟Class3比较远这样子的问题,如果你把这个One-hot vector,用算距离的话,Class之间 两两它们的距离都是一样

Classification with softmax

Regression

$$\begin{array}{c} \text{label} \\ \hat{y} \end{array} \longleftrightarrow \begin{array}{c} y \\ y \end{array} = \begin{array}{c} b \\ b \end{array} + \begin{array}{c} c^T \\ c^T \end{array} \sigma \left(\begin{array}{c} b \\ b \end{array} + \begin{array}{c} W \\ W \end{array} \begin{array}{c} \text{feature} \\ x \\ x \end{array} \right)$$

Regression输入Input x, 使Output y 要跟 label \hat{y} ,越接近越好

Classification

$$\begin{array}{c} y \\ y \end{array} = \begin{array}{c} b' \\ b' \end{array} + \begin{array}{c} W' \\ W' \end{array} \sigma \left(\begin{array}{c} b \\ b \end{array} + \begin{array}{c} W \\ W \end{array} \begin{array}{c} \text{feature} \\ x \\ x \end{array} \right)$$

$$\begin{array}{c} \text{label} \\ \hat{y} \end{array} \longleftrightarrow \begin{array}{c} y' \\ y' \end{array} = \text{softmax} \left(\begin{array}{c} y \\ y \end{array} \right)$$

0 or 1 Make all values between 0 and 1 Can have any value

如果是Classification,input x可能乘上一个W,再加上b 再通过activation function,再乘上W'再加上b' 得到y,我们现在的y它不是一个数值,它是一个向量。

但是在做Classification的时候,我们往往会把y再通过一个叫做Soft-max的function得到y',然后我们才去计算,y'跟y hat之间的距离

为什么要加上Soft-max呢,一个比较简单的解释 (如果是在过去的课程里面,我们会先从generative的Model开始讲起,然后一路讲到Logistic Regression)

这个 \hat{y} 它里面的值,都是0跟1,它是One-hot vector,所以里面的值只有0跟1, 但是y里面有任何值

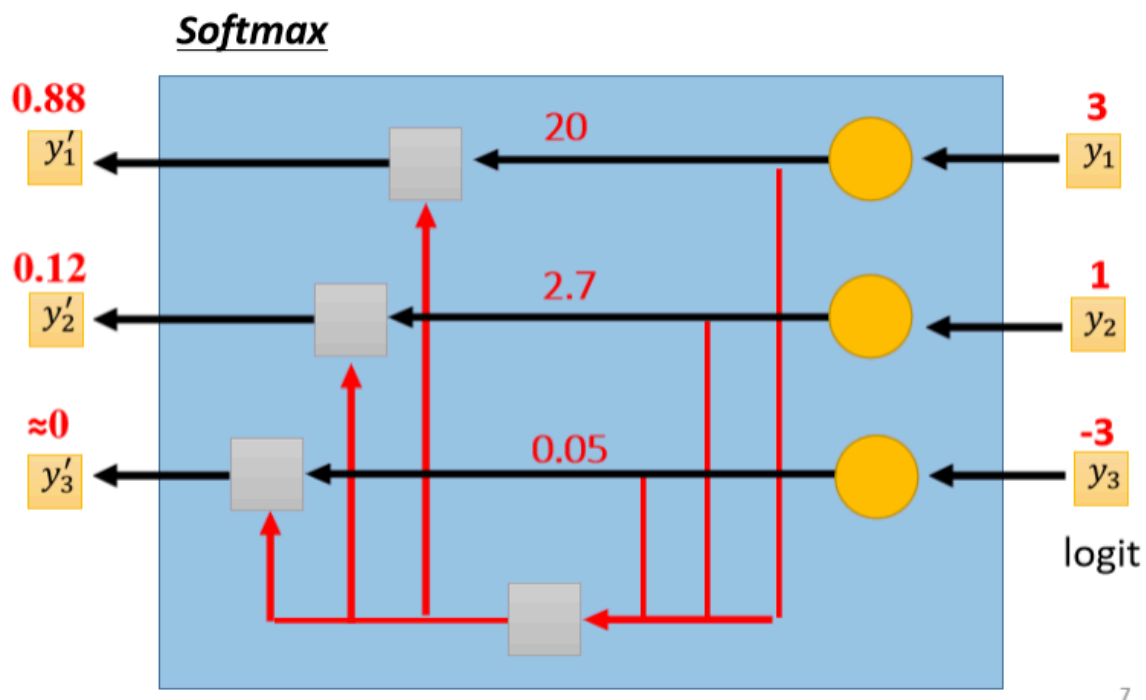
既然我们的目标只有0跟1,但是y有任何值,我们就先把它Normalize到0到1之间,这样才好跟 label 的计算相似度,这是一个比较简单的讲法。把本来y里面可以放任何值,改成挪到0到1之间

Softmax

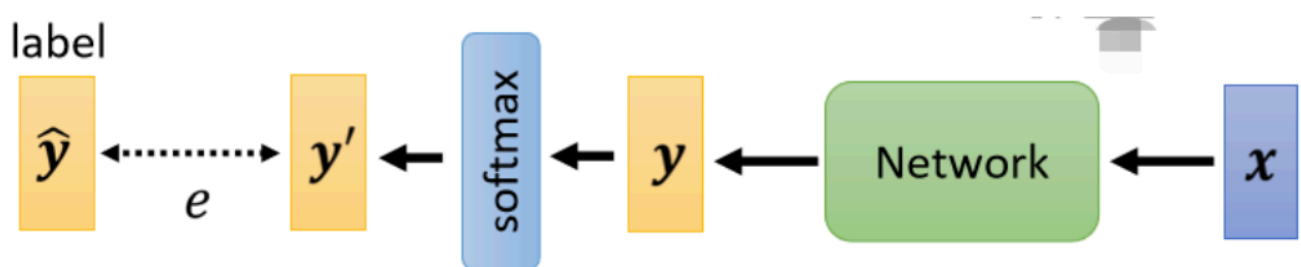
$$y'_i = \frac{\exp(y_i)}{\sum_j \exp(y_i)}$$

先把所有的 y 取一个exponential,就算是负数,取exponential以后也变成正的,然后你再对它做Normalize,除掉所有 y 的exponential值的和,然后你就得到 y'

Soft-max要做的事情,除了Normalized,让 $y_1' y_2' y_3'$ 变成0到1之间,还有和为1以外,它还有一个附带的效果是,它会让大的值跟小的值的差距更大



Loss of Classification



们把 x ,丢到一个Network裡面產生 y 以后,我们会通过soft-max得到 y' ,再去计算 y' 跟 \hat{y} 之间的距离,这个写作 e

计算 y' 跟 \hat{y} 之间的距离不只一种做法,举例来说,如果我喜欢的话,我要让这个距离是Mean Square Error

$$e = \sum_i (\hat{y}_i - y'_i)^2$$

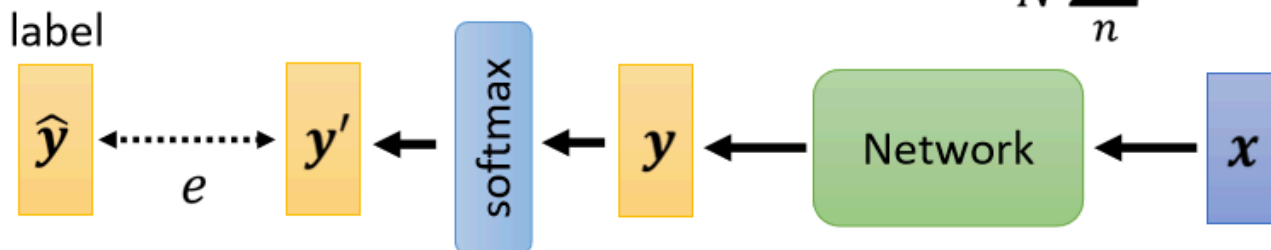
另外一个更常用的做法,叫做Cross-entropy

$$e = - \sum_i \hat{y}_i \ln y'_i$$

- Cross-entropy是summation over所有的 i
- 然后把 \hat{y} 的第 i 位拿出来,乘上 y' 的第 i 位取Natural log
- 然后再全部加起来

Loss of Classification

$$L = \frac{1}{N} \sum_n e_n$$



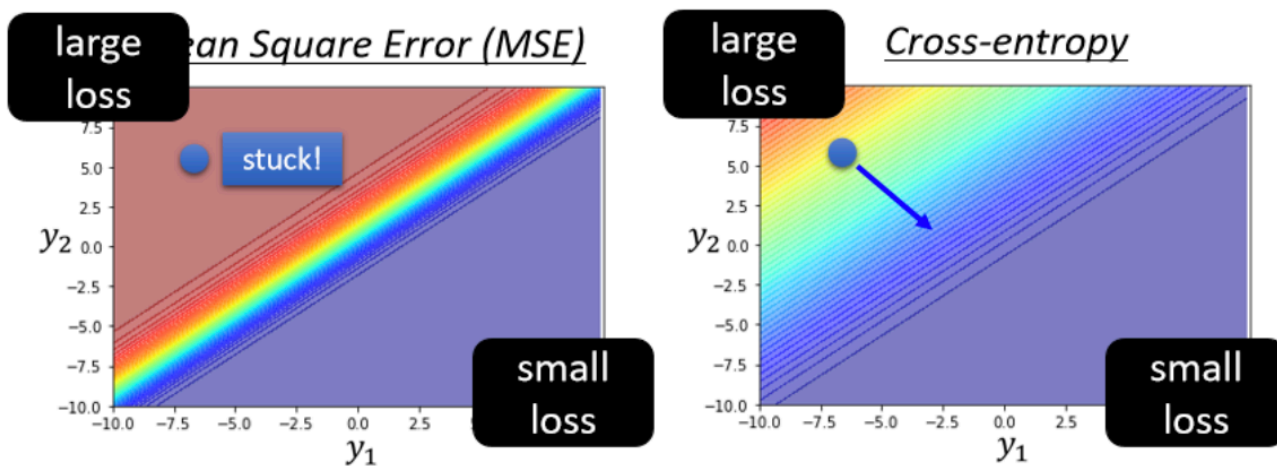
Mean Square Error (MSE) $e = \sum_i (\hat{y}_i - y'_i)^2$

Cross-entropy $e = - \sum_i \hat{y}_i \ln y'_i$

Minimizing cross-entropy is equivalent to **maximizing likelihood**.

Make Minimize Cross-entropy其实就是maximize likelihood是一模一样的东西，只是同一件事不同的讲法

MSE和Cross-entropy:



Changing the loss function can change the difficulty of optimization.

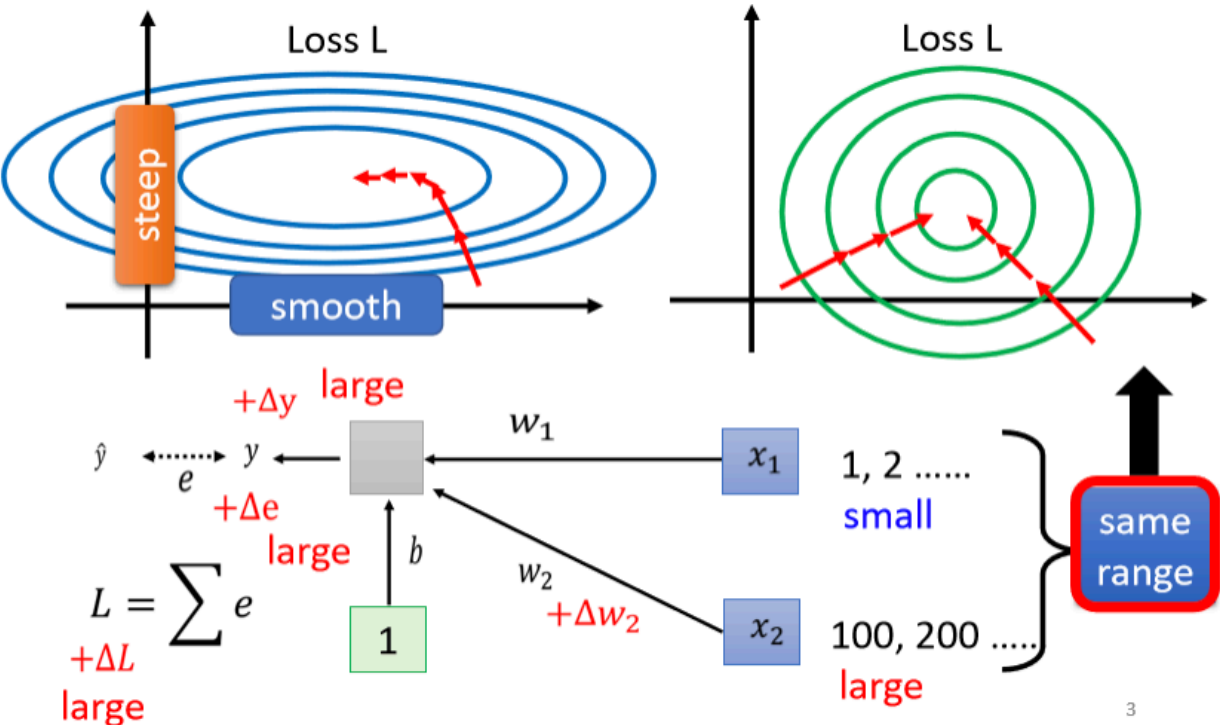
Batch Normalization

我们能不能够直接改error surface 的 landscape,我们觉得说 error surface 如果很崎岖的时候,它比较难 train,那我们能不能够直接把山削平,让它变得比较好 train 呢?

Batch Normalization 就是其中一个,把山削平的想法。

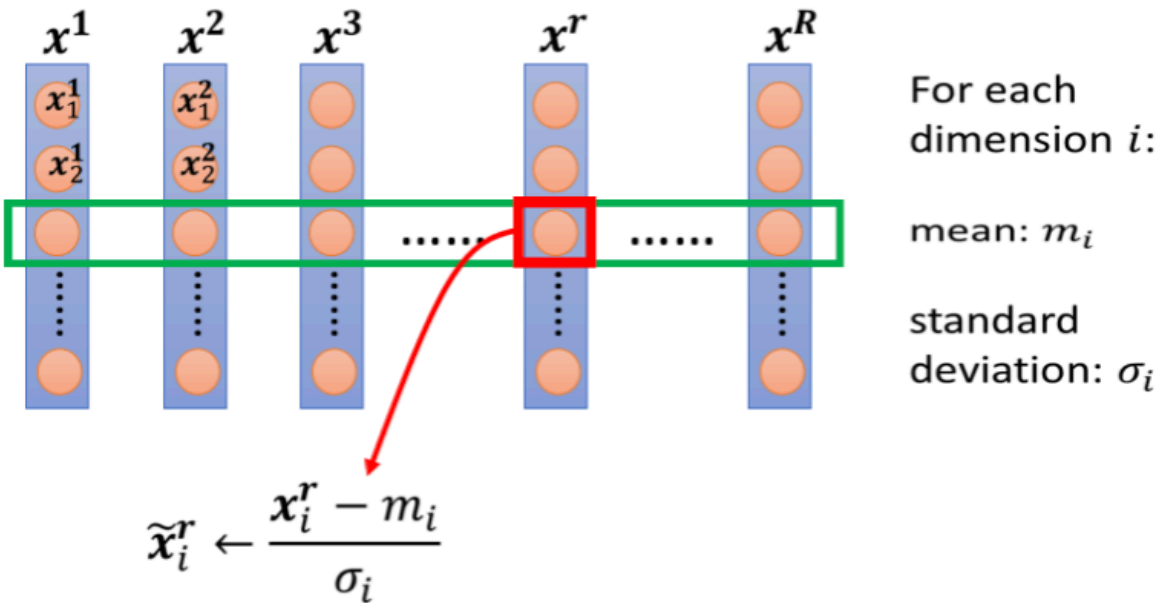
既然在这个 linear 的 model 里面,当我们 input 的 feature,每一个 dimension 的值,它的 scale 差距很大的时候,我们就可能產生像这样子的 error surface,就可能產生不同方向,斜率非常不同,坡度非常不同的 error surface

我们有没有可能给feature 里面不同的 dimension,让它有同样的数值的范围



Feature Normalization

介绍其中的一种可能性,它并不是 Feature Normalization 的全部,假设 x^1 到 x^R ,是我们所有的训练资料的 feature vector



把不同笔资料即不同 feature vector,同一个 dimension 裡面的数值,把它取出来,然后去计算某一个 dimension 的 mean, 它的 mean 呢 就是 m_i , 我们计算第 i 个 dimension 的,standard deviation,我们用 σ_i 来表示它。

那接下来我们就可以做一种 normalization, 其实叫做标准化,其实叫 standardization,不过我们都统称 normalization。

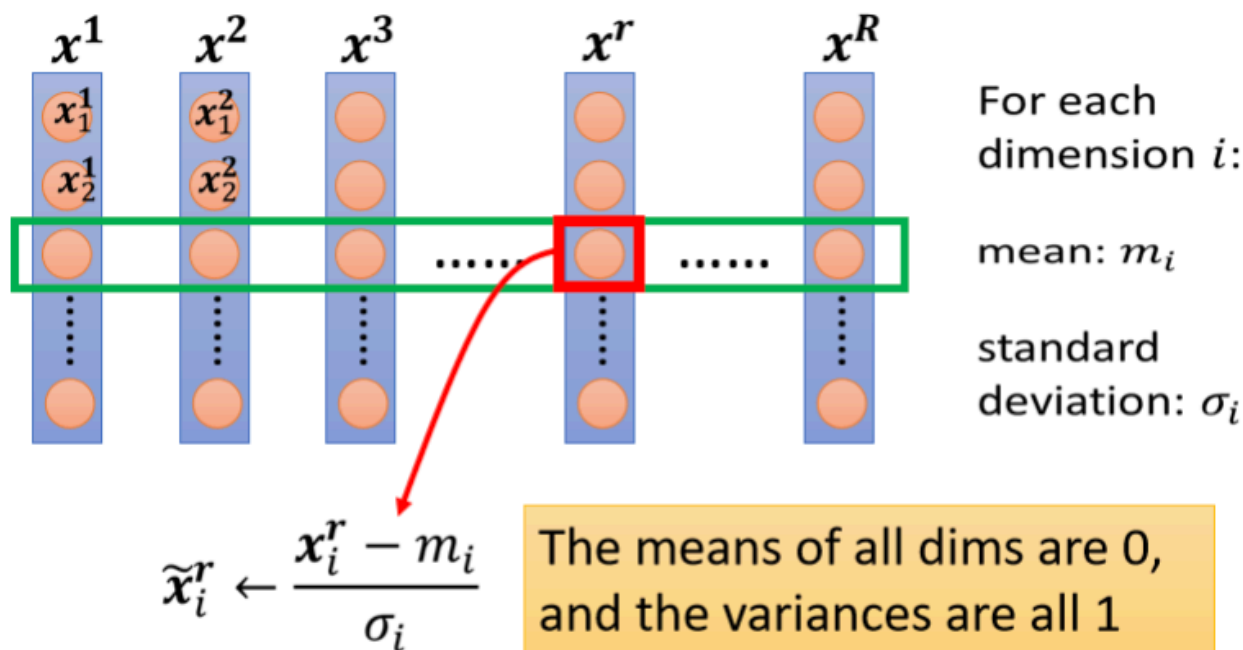
$$\hat{x}_i^r \leftarrow \frac{x_i^r - m_i}{\sigma_i}$$

把这边的某一个数值 x ,减掉这一个 dimension 算出来的 mean,再除掉这个 dimension算出来的 standard deviation,得到新的数值叫做 \tilde{x}

然后得到新的数值以后,再把新的数值把它塞回去,以下都用这个 tilde来代表有被 normalize 后的数值

normalize好处

做 gradient descent 的时候,它的 Loss 收敛更快一点,可以让你的 gradient descent,它的训练更顺利一点,这个是 Feature Normalization



In general, feature normalization makes gradient descent converge faster.

4

- 做完 normalize 以后,这个 dimension 上面的数值就会平均是 0,然后它的 variance 就会是 1,所以这一排数值的分布就都会在 0 上下
- 对每一个 dimension 都做一样的 normalization,就会发现所有 feature 不同 dimension 的数值都在 0 上下,那你可能就可以制造一个比较好的 error surface

Batch Normalization

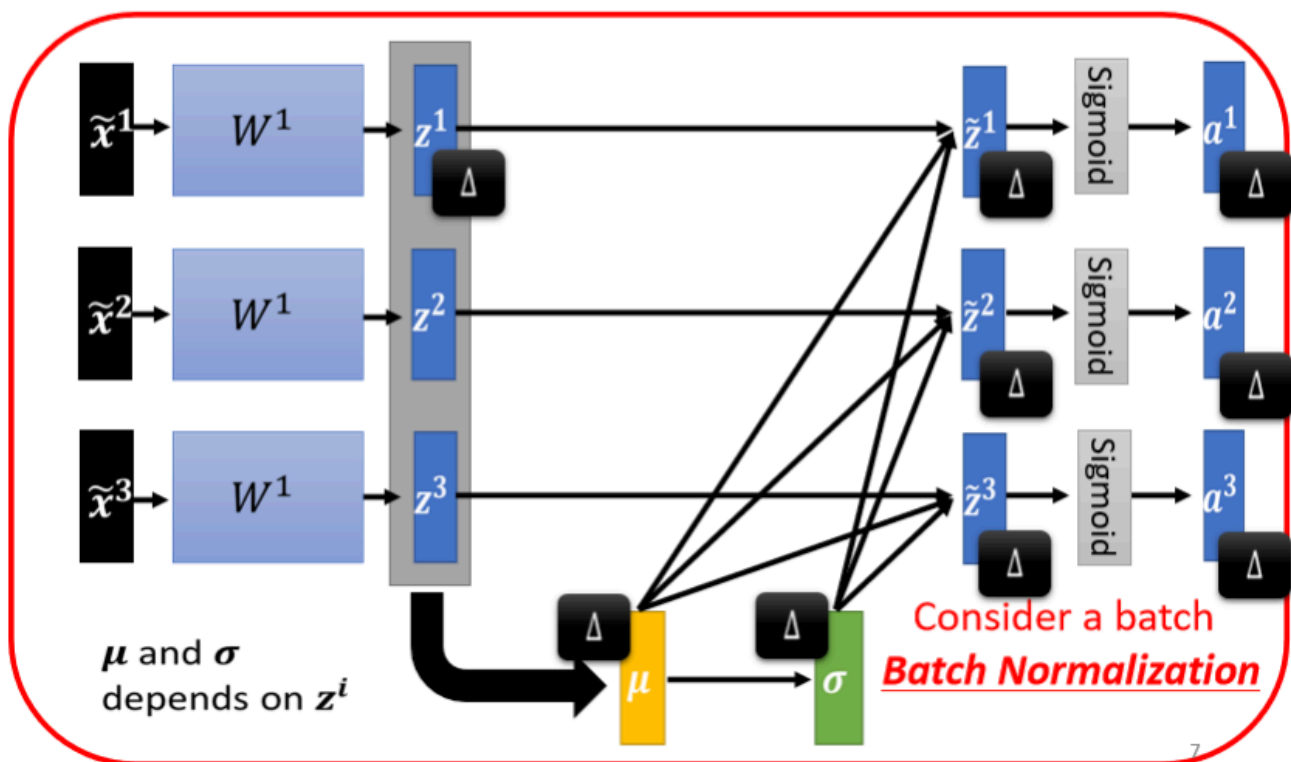
Batch Normalization,是适用于 batch size 比较大的时候,因为 batch size 如果比较大,也许这个 batch size 里面的 data,就足以表示整个 corpus 的分布,可以把这个本来要对整个 corpus 做 Feature Normalization 这件事情,改成只在一个 batch,做 Feature Normalization,作为 approximation,

有一个问题 你一定要有一个够大的 batch,你才算得出 μ 跟 σ ,假设你今天 batch size 设 1,那你就没有什么 μ 或 σ 可以算

Considering Deep Learning

$$\tilde{z}^i = \frac{z^i - \mu}{\sigma}$$

This is a large network!



更多的normalization

- Batch Renormalization <https://arxiv.org/abs/1702.03275>
- Layer Normalization <https://arxiv.org/abs/1607.06450>
- Instance Normalization <https://arxiv.org/abs/1607.08022>
- Group Normalization <https://arxiv.org/abs/1803.08494>
- Weight Normalization <https://arxiv.org/abs/1602.07868>
- Spectrum Normalization <https://arxiv.org/abs/1705.10941>