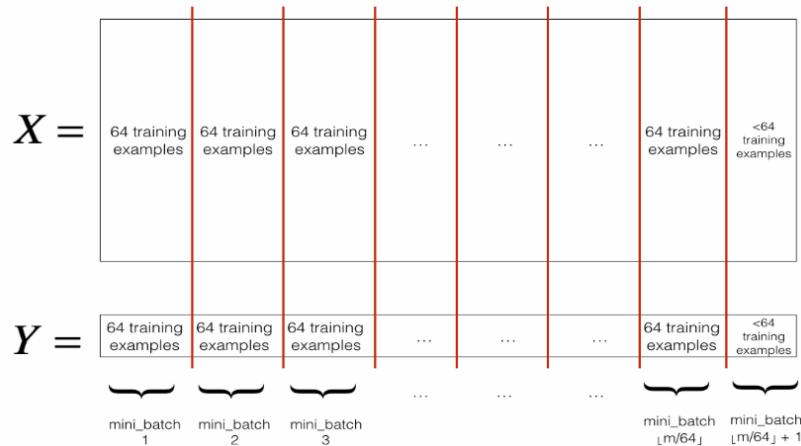


# Optimization

## • mini-batch gradient descent

- why
  - suppose data set has **10 billion** examples,
  - it's really slow to compute gradient for each iteration
  - don't have enough memory to contain whole data set
- how
  - split the data set into smaller size. #notation:  $t: X\{t\}, Y\{t\}$



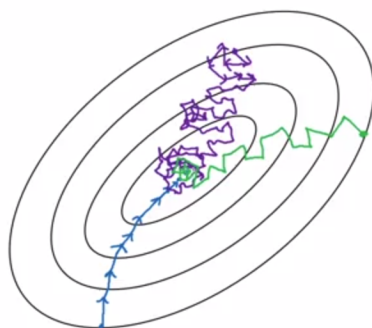
- pseudo code
  - for  $t = 1:\text{No\_of\_batches}$  # this is called an epoch
  - $AL, \text{caches} = \text{forward\_prop}(X\{t\}, Y\{t\})$
  - $\text{cost} = \text{compute\_cost}(AL, Y\{t\})$
  - $\text{grads} = \text{backward\_prop}(AL, \text{caches})$
  - $\text{update\_parameters}(\text{grads})$
- how to choose mini-batch size
  - the size should be power of 2, such as 64/128/512/1024

## Choosing your mini-batch size

→ If mini-batch size =  $m$  : Batch gradient descent.  $(X^{\text{B}}, Y^{\text{B}}) = (X, Y)$

→ If mini-batch size = 1 : Stochastic gradient descent. Every example is its own  $(X^{\text{S}}, Y^{\text{S}}) = (x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$  mini-batch.

In practice: Search in-between 1 and  $m$



Stochastic  
gradient  
descent  
↓  
Less sensitive  
for initialization

In-between  
(mini-batch size  
not too big/small)

Fastest learning.

- Vectorization.
- (w/1000)
- Make passes without  
processing entire training set.

Batch  
gradient descent  
(mini-batch size =  $m$ )  
↓  
Too long  
per iteration

# • Exponentially weighted average (come from moving average)

## ◦ basic concept

- analyze data points by creating a series of **averages** of different subsets of the full data set, which applies **weighting factors** which decrease exponentially.
- reduce noise and smooth data
- general equation:
  - $V(t) = \beta * v(t-1) + (1-\beta) * \theta(t)$

## Implementing exponentially weighted averages

$$\begin{aligned} v_0 &= 0 \\ v_1 &= \beta v_0 + (1-\beta) \theta_1 \\ v_2 &= \beta v_1 + (1-\beta) \theta_2 \\ v_3 &= \beta v_2 + (1-\beta) \theta_3 \\ &\dots \end{aligned}$$

$$\begin{aligned} V_0 &:= 0 \\ V_1 &:= \beta V_0 + (1-\beta) \theta_1 \\ V_2 &:= \beta V_1 + (1-\beta) \theta_2 \\ &\vdots \end{aligned}$$


---

$V_0 = 0$   
Repeat {  
  Get next  $\theta_t$   
   $V_t := \beta V_{t-1} + (1-\beta) \theta_t$   
}

- example here (often used in stock)

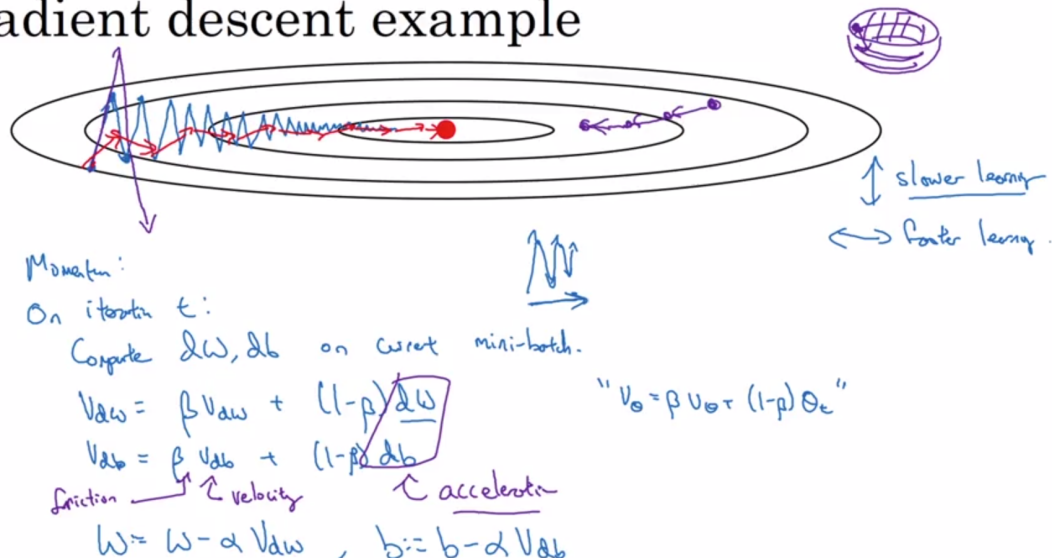
## ◦ bias correction

- at the beginning the average is shifted as  $v(0) = 0$
- to correct this we can amend the equation
  - $v(t) = (\beta * v(t-1) + (1-\beta) * \theta(t)) / (1 - \beta^t)$
- people often not bother because after few iterations the bias would decrease

## ◦ gradient descent with momentum

- to calculate the exponentially weighted averages for your gradients and then update your weights with the new values.
- analogy
  - smooth the vertical noise and keep the horizontal gradient descent

## Gradient descent example



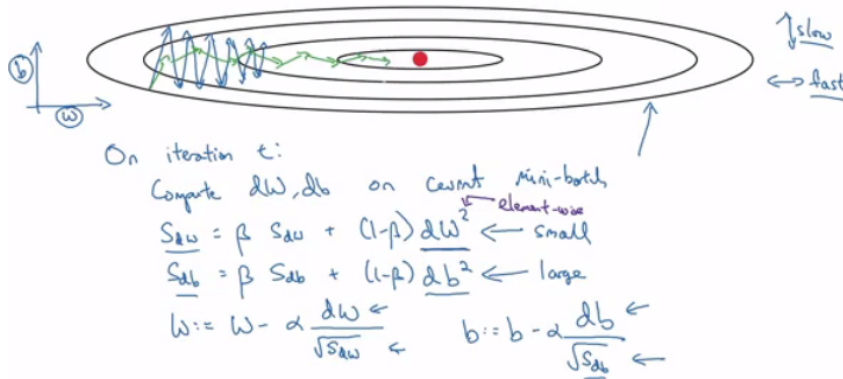
## ▪ pseudo code

- $vdW = 0, vdb = 0$
- on iteration  $t$ :

- # can be mini-batch or batch gradient descent
- compute dw, db on current mini-batch
- 
- $vdW = \beta * vdW + (1 - \beta) * dW$
- $vdb = \beta * vdb + (1 - \beta) * db$
- $W = W - \text{learning\_rate} * vdW$
- $b = b - \text{learning\_rate} * vdb$

## • RMS (Root mean square) prop

### RMSprop



- 
- Pseudo code:
- 

```

sdW = 0, sdb = 0
on iteration t:
# can be mini-batch or batch gradient descent
compute dw, db on current mini-batch
sdW = (beta * sdW) + (1 - beta) * dW^2 # squaring is element-wise
sdb = (beta * sdb) + (1 - beta) * db^2 # squaring is element-wise
W = W - learning_rate * dW / sqrt(sdW)
b = B - learning_rate * db / sqrt(sdb)

```

- tips: ensure that  $sdW$  is not zero by adding a small value  $\epsilon$  (e.g.  $\epsilon = 10^{-8}$ ) to it

## • Adam optimization algorithm

- stand for adaptive moment estimation
- combine **momentum** and **RMS prop** together

$$\left\{ \begin{array}{l} v_{W^{[l]}} = \beta_1 v_{W^{[l]}} + (1 - \beta_1) \frac{\partial J}{\partial W^{[l]}} \\ v_{W^{[l]}}^{\text{corrected}} = \frac{v_{W^{[l]}}}{1 - (\beta_1)^t} \\ s_{W^{[l]}} = \beta_2 s_{W^{[l]}} + (1 - \beta_2) \left( \frac{\partial J}{\partial W^{[l]}} \right)^2 \\ s_{W^{[l]}}^{\text{corrected}} = \frac{s_{W^{[l]}}}{1 - (\beta_2)^t} \\ W^{[l]} = W^{[l]} - \alpha \frac{v_{W^{[l]}}^{\text{corrected}}}{\sqrt{s_{W^{[l]}}^{\text{corrected}} + \epsilon}} \end{array} \right.$$

- 
- pseudo code:

- $vdW = 0, vdb = 0$
- $sdW = 0, sdb = 0$
- on iteration  $t$ :
- # can be mini-batch or batch gradient descent
- compute  $dw, db$  on current mini-batch
- 
- $vdW = (\text{beta1} * vdW) + (1 - \text{beta1}) * dw$  # momentum
- $vdb = (\text{beta1} * vdb) + (1 - \text{beta1}) * db$  # momentum
- 
- $sdW = (\text{beta2} * sdW) + (1 - \text{beta2}) * dw^2$  # RMSprop
- $sdb = (\text{beta2} * sdb) + (1 - \text{beta2}) * db^2$  # RMSprop
- 
- $vdW = vdW / (1 - \text{beta1}^t)$  # fixing bias
- $vdb = vdb / (1 - \text{beta1}^t)$  # fixing bias
- 
- $sdW = sdW / (1 - \text{beta2}^t)$  # fixing bias
- $sdb = sdb / (1 - \text{beta2}^t)$  # fixing bias
- 
- $W = W - \text{learning\_rate} * vdW / (\text{sqrt}(sdW) + \text{epsilon})$
- $b = B - \text{learning\_rate} * vdb / (\text{sqrt}(sdb) + \text{epsilon})$

◦ hyper parameters:

## Hyperparameters choice:

→  $\alpha$ : needs to be tune  
 →  $\beta_1$ : 0.9 (dw)  
 →  $\beta_2$ : 0.999 ( $dw^2$ )  
 →  $\epsilon$ :  $10^{-8}$

- 
- usually just tune alpha

## • learning rate decay

- in previous course we set fixed alpha. But if alpha is too small, it take much time to train while if alpha is too large, we may miss the global optimum
- so we want alpha flexible
- methods

### Learning rate decay

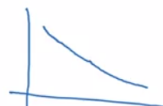
1 epoch = 1 pass through data.

$$\alpha = \frac{1}{1 + \text{decay\_rate} * \text{epoch\_num}} \alpha_0$$

Epoch	$\alpha$
1	0.1
2	0.67
3	0.5
4	0.4
⋮	⋮




$\alpha_0 = 0.2$   
 decay\_rate = 1



## Other learning rate decay methods

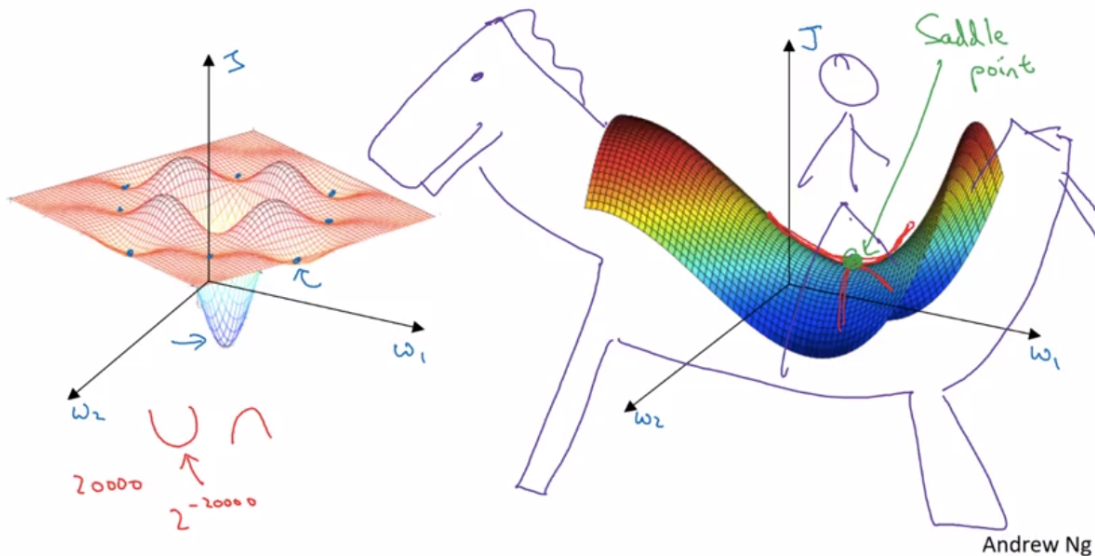
formula

$$\begin{cases} \alpha = 0.95^{\text{epoch-num}} \cdot \alpha_0 & \text{— exponentially decay} \\ \alpha = \frac{k}{\sqrt{\text{epoch-num}}} \cdot \alpha_0 & \text{or } \frac{k}{\sqrt{t}} \cdot \alpha_0 \end{cases}$$


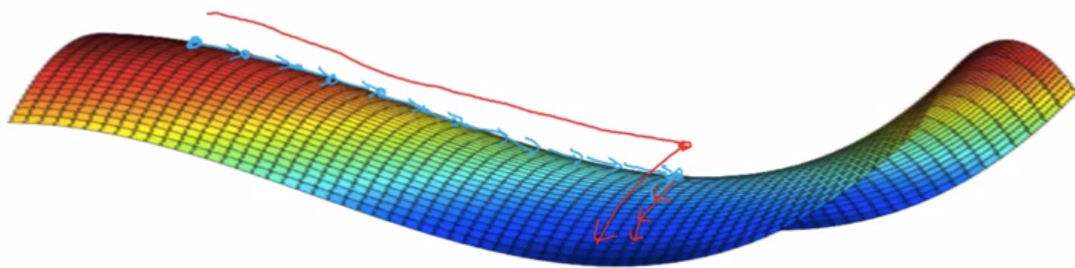
discrete staircase

- even manual tune the alpha when the model is training
- intuition about local optimum
  - #so cute 🐾

## Local optima in neural networks



## Problem of plateaus



- Unlikely to get stuck in a bad local optima
- Plateaus can make learning slow