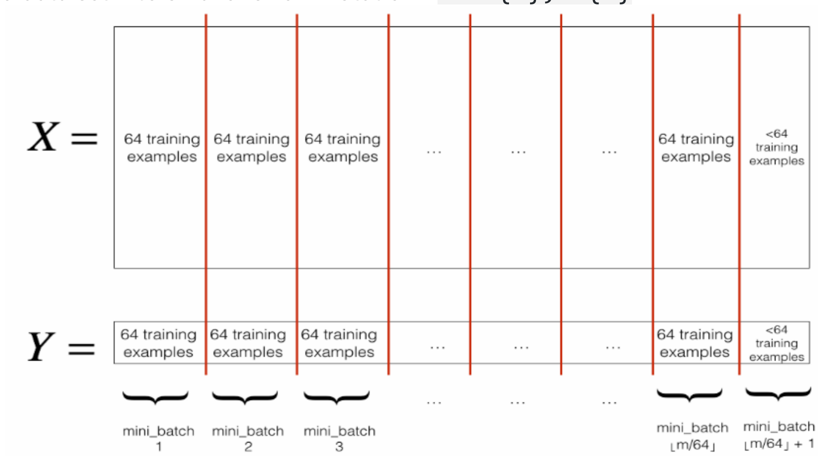
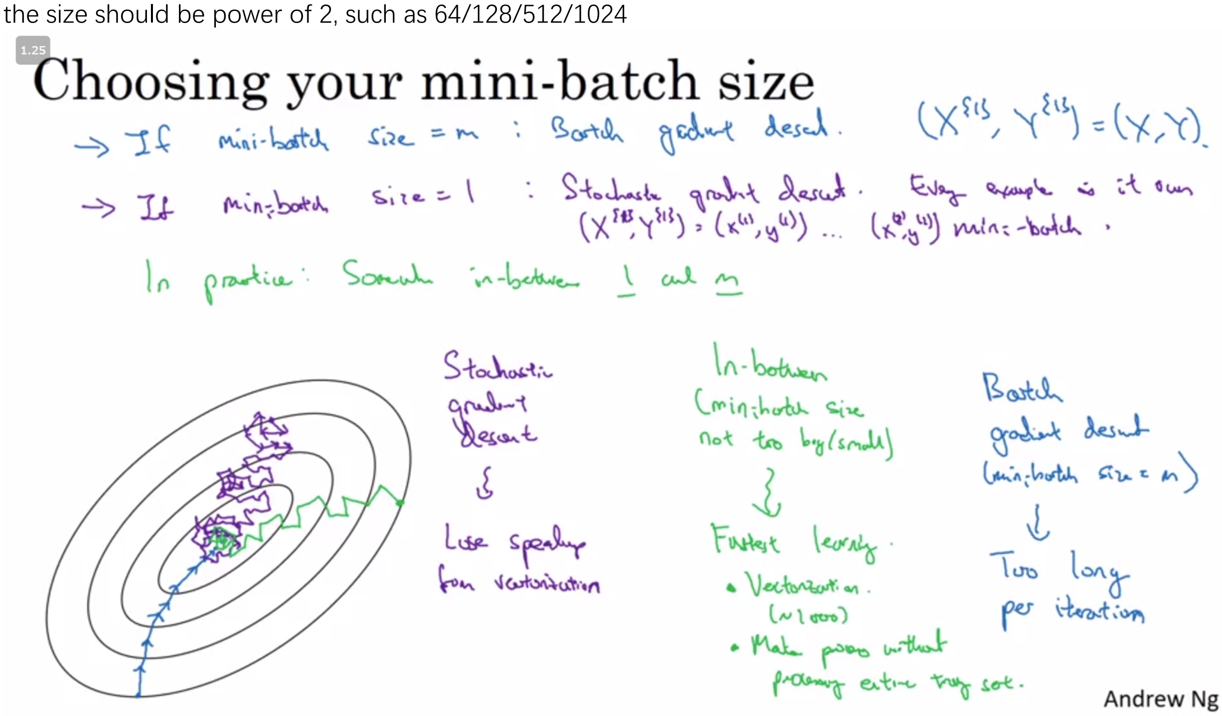
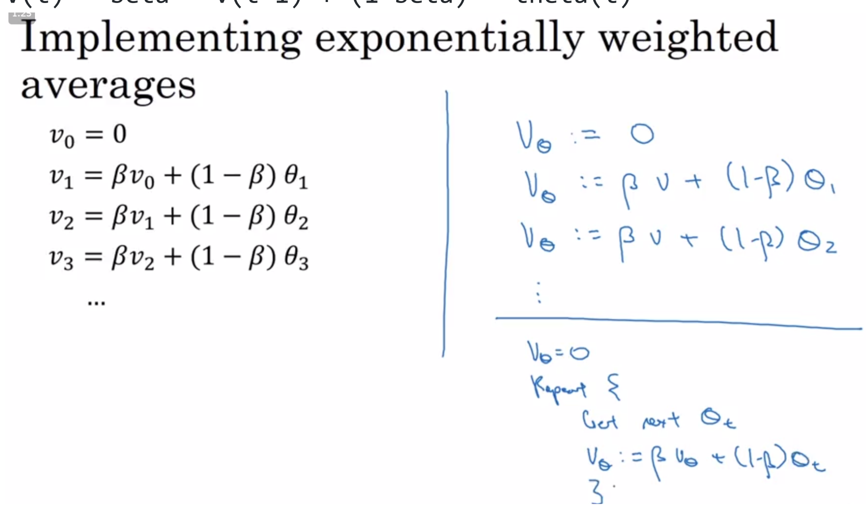
deep Learning_c2_week2

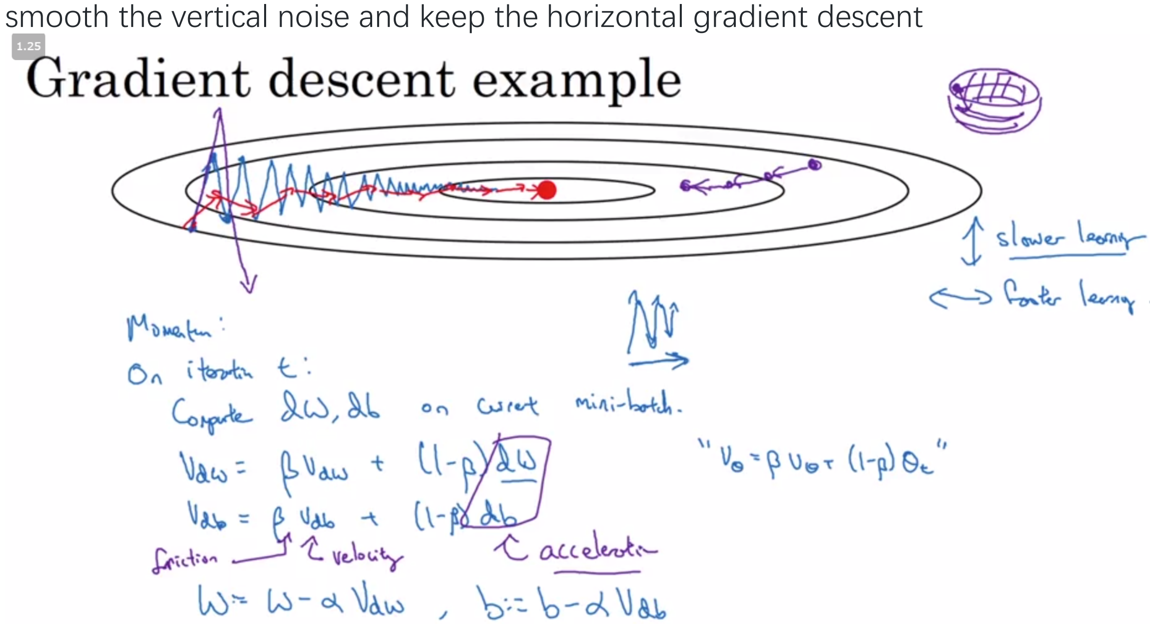
Created:	19/02/05 20:19	**Updated:**	19/02/06 08:33

Optimization

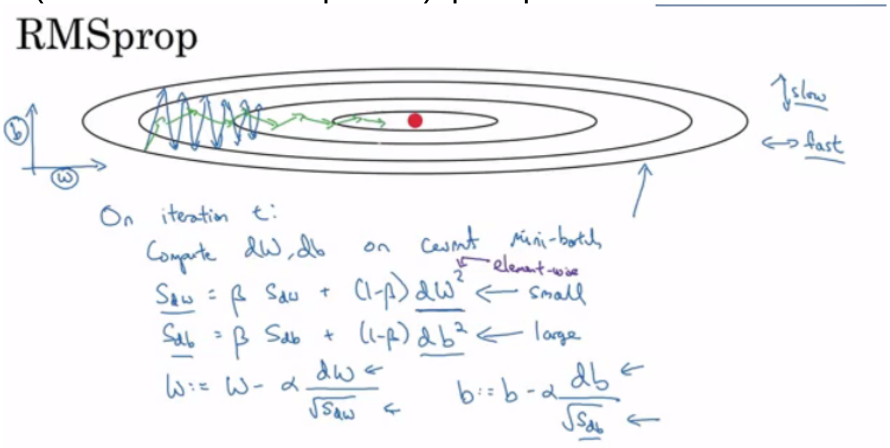
- ## mini-batch gradient descent
  - why
    - suppose data set has **10 billion** examples,
    - it's really slow to compute gradient for each iteration
    - don't have enough memory to contain whole data set
  - how
    - split the data set into smaller size. #notation: `t: X{t}, Y{t}`
    - 

    - pseudo code
      - `for t = 1:No_of_batches # this is called an epoch`
      - `AL, caches = forward_prop(X{t}, Y{t})`
      - `cost = compute_cost(AL, Y{t})`
      - `grads = backward_prop(AL, caches)`
      - `update_parameters(grads)`
  - how to choose mini-batch size
    - the size should be power of 2, such as 64/128/512/1024
    - 

- # 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)`
      - 

    - 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 border 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
      - 

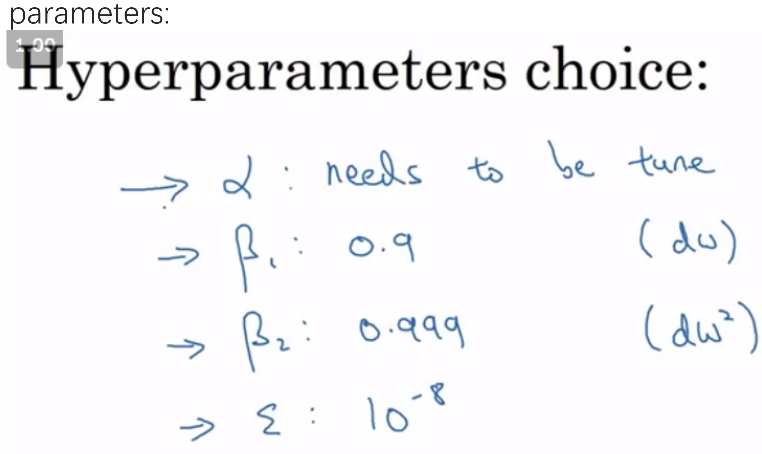
    - pseudo code
      - `vdW = 0, vdb = 0`
      - `on iteration t:`

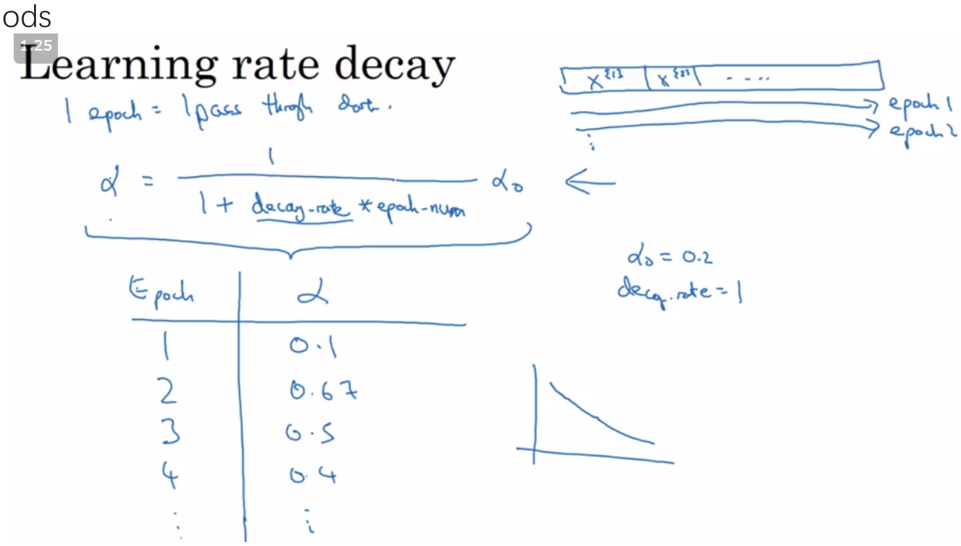
        - `# can be mini-batch or batch gradient descent`
        - `compute dw, db on current mini-batch`
        -
        - `vdW = beta * vdW + (1 - beta) * dW`
        - `vdb = beta * vdb + (1 - beta) * db`
        - `W = W - learning_rate * vdW`
        - `b = b - learning_rate * vdb`
- RMS (Root mean square) prop
    - 

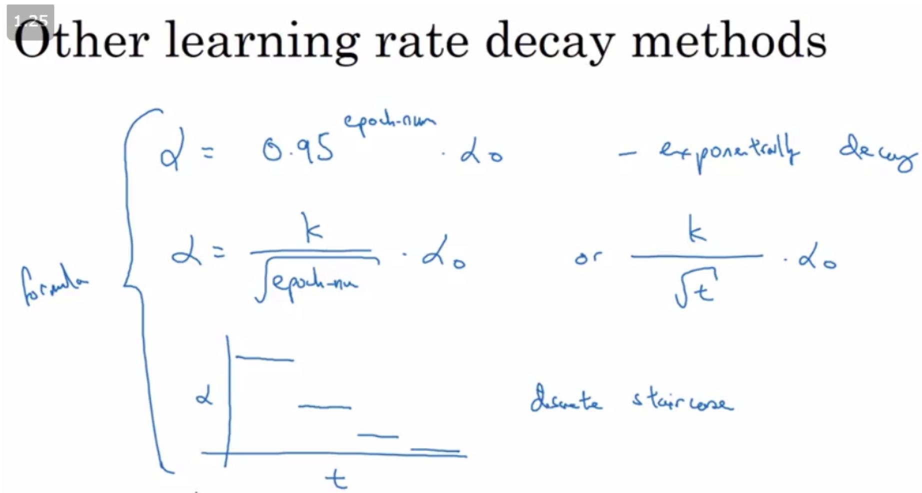
    - Pseudo code:
    -

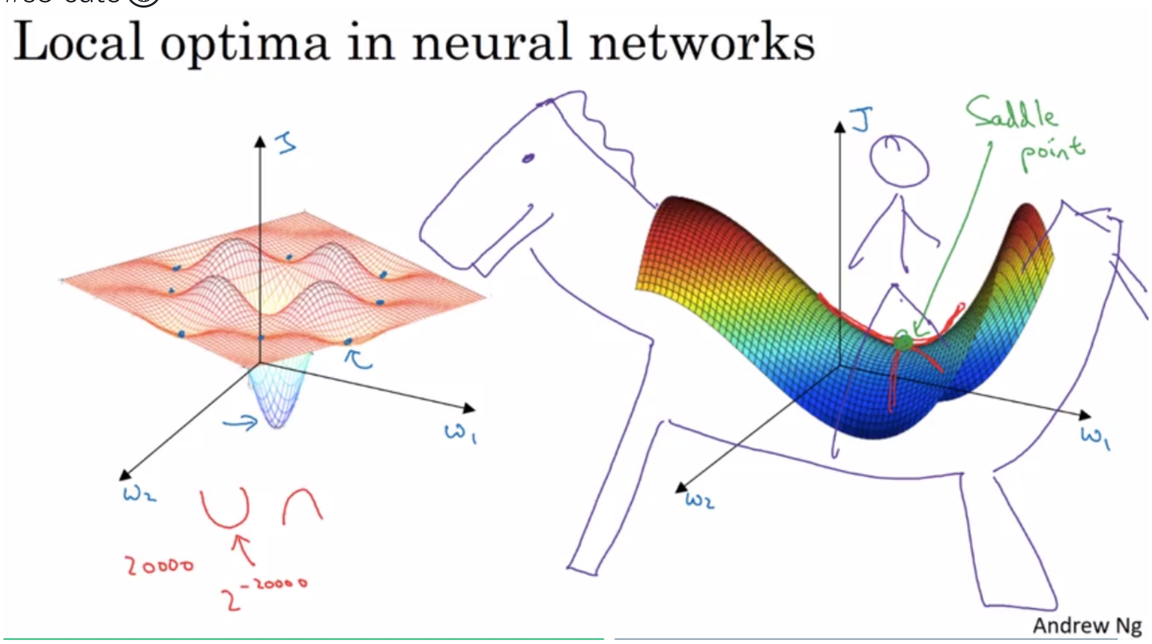
```
sdW = 0, sdb = 0
on iteration t:
# can be mini-batch or batch gradient descent
compute dw, db on current mini-batch
sdW = (beta * sdW) + (1 - beta) * dW^2 # squaring is element-wise
sdb = (beta * sdb) + (1 - beta) * db^2 # squaring is element-wise
W = W - learning_rate * dW / sqrt(sdW)
b = B - learning_rate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

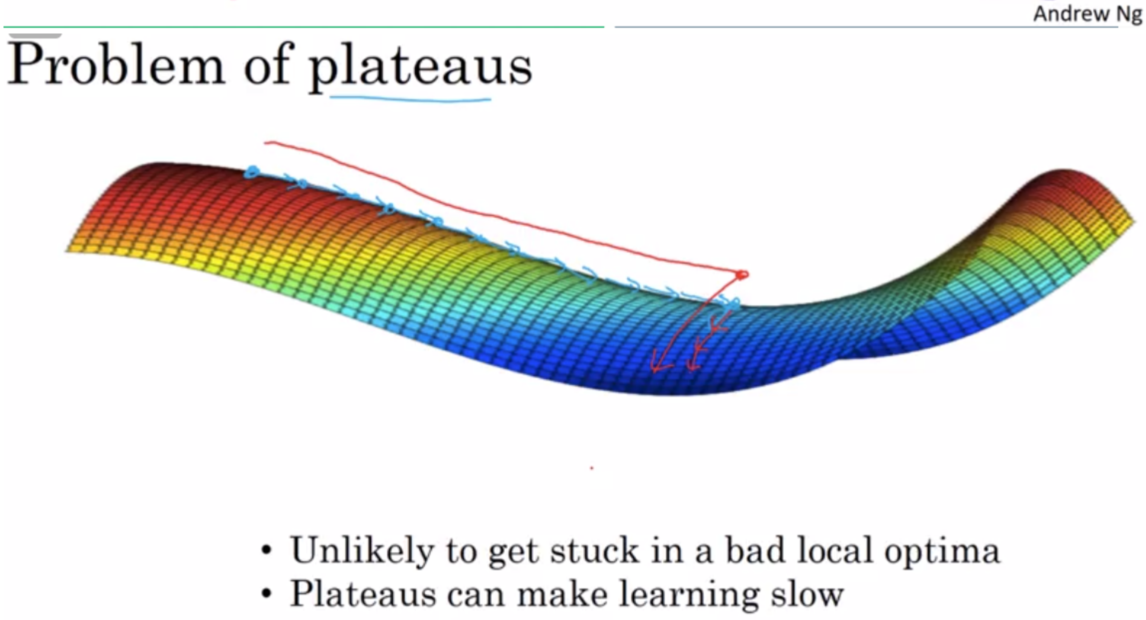
    - tips: ensure that `sdW` is not zero by adding a small value `epsilon` (e.g. `epsilon = 10^-8`) to it
- # Adam optimization algorithm
    - stand for adaptive moment estimation
    - combine **momentum and RMS prop** together
        - $$\begin{cases} v_{W^{[l]}} = \beta_1 v_{W^{[l]}} + (1-\beta_1)\frac{\partial J}{\partial W^{[l]}} \\ v^{corrected}_{W^{[l]}} = \frac{v_{W^{[l]}}}{1-(\beta_1)^t} \\ s_{W^{[l]}} = \beta_2 s_{W^{[l]}} + (1-\beta_2)(\frac{\partial J}{\partial W^{[l]}})^2 \\ s^{corrected}_{W^{[l]}} = \frac{s_{W^{[l]}}}{1-(\beta_2)^t} \\ W^{[l]} = W^{[l]} - \alpha\frac{v^{corrected}_{W^{[l]}}}{\sqrt{s^{corrected}_{W^{[l]}}}+\varepsilon} \end{cases}$$
        - pseudo code:

        - `vdW = 0, vdW = 0`
        - `sdW = 0, sdb = 0`
        - `on iteration t:`
        - `# can be mini-batch or batch gradient descent`
        - `compute dw, db on current mini-batch`
        -
        - `vdW = (beta1 * vdW) + (1 - beta1) * dW # momentum`
        - `vdb = (beta1 * vdb) + (1 - beta1) * db # momentum`
        -
        - `sdW = (beta2 * sdW) + (1 - beta2) * dW^2 # RMSprop`
        - `sdb = (beta2 * sdb) + (1 - beta2) * db^2 # RMSprop`
        -
        - `vdW = vdW / (1 - beta1^t) # fixing bias`
        - `vdb = vdb / (1 - beta1^t) # fixing bias`
        -
        - `sdW = sdW / (1 - beta2^t) # fixing bias`
        - `sdb = sdb / (1 - beta2^t) # fixing bias`
        -
        - `W = W - learning_rate * vdW / (sqrt(sdW) + epsilon)`
        - `b = B - learning_rate * vdb / (sqrt(sdb) + epsilon)`
    - hyper parameters:
        - 

        - 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
        - 

        - 

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


    - 

    -