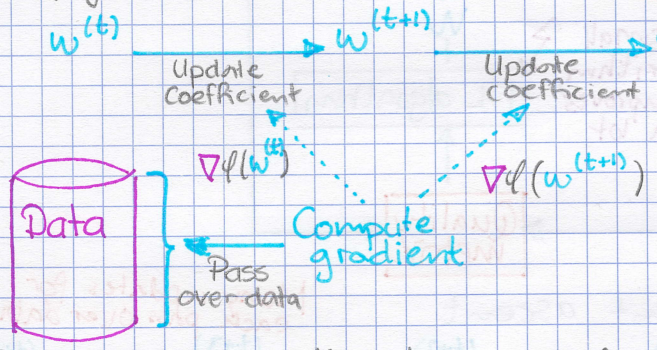
Scaling to Huge Datasets & Online Learning

Gradient ascent won't scale to today's huge datasets

Why gradient ascent is slow...

Every update requires a full pass over data



Data sets are getting huge, and we need them!

WWW
4.8 B webpages

twitter
500M Tweets/day

Internet of Things
Sensors everywhere

YouTube
300 hours uploaded/min → 1B users, ad revenue → 5B views/day

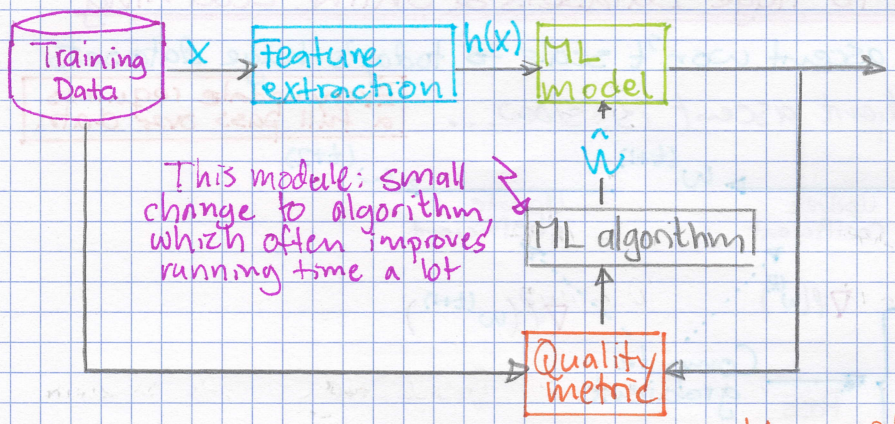
Need ML algorithm to learn from billions of video views every day, & to recommend ads within milliseconds

Timeline of scalable machine learning & stochastic gradient

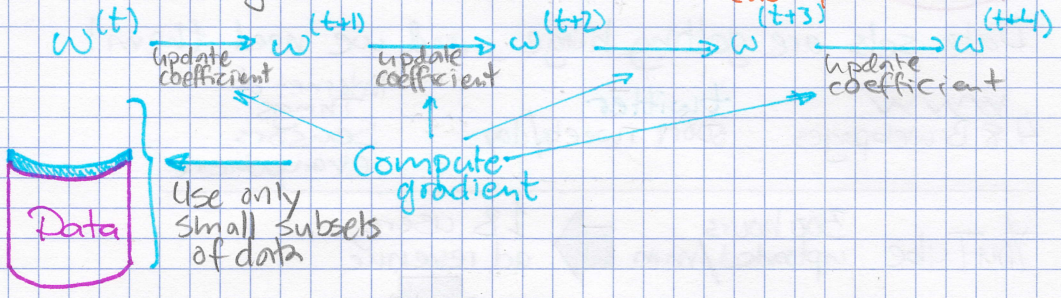
ML improves (significantly) with bigger dataset

Data size: Small data (1996) → Big data (2006) → Bigger data (2016)

	Small data (1996)	Big data (2006)	Bigger data (2016)
Model complexity	• Complex Needed for accuracy	Simple • Needed for speed	Complex • Needed for accuracy • Parallelism, GPUs, computer clusters.
Hot topics	Kernel Graphical models	Logistic regression Matrix factorization	Boosted trees Tensor factorization Deep learning Massive graphical models



Stochastic gradient ascent



Why gradient ascent won't scale

How expensive is gradient ascent?

Sum over data points

$$\frac{\partial \ell(w)}{\partial w_j} = \sum_{i=1}^{N} h_j(x_i)\left(\mathbb{1}[y_i = +1] - P(y = +1 \mid x_i, w)\right)$$

Contribution of data point x_i, y_i to gradient

$$\frac{\partial \ell_i(w)}{\partial w_j}$$

Every step requires touching every data point!!!

$$\frac{\partial \ell(w)}{\partial w_j} = \sum_{i=1}^{N} \frac{\partial \ell_i(w)}{\partial w_j}$$

Time to compute contribution of x_i, y_i	# of data points (N)	Total time to compute 1 step of gradient ascent
1 millisecond	1000	1 sec
1 sec	1000	16.7 min
1 millisecond	10 million	2.8 hours
1 millisecond	10 billion	115.7 days

Stochastic gradient: Learning one data point at a time
Instead of all data points for gradient, use 1 data point only

Gradient ascent

$$\frac{\partial \ell(w)}{\partial w_j} = \sum_{i=1}^{N} \frac{\partial \ell_i(w)}{\partial w_j}$$

(Sum over data points)

Stochastic gradient ascent

$$\frac{\partial \ell(w)}{\partial w_j} \approx \frac{\partial \ell_i(w)}{\partial w_j}$$

(Each time, pick different data point i)

Stochastic gradient ascent for logistic regression

init $w^{(1)}=0, t=1$

until converged:

  for $i = 1, \ldots, N$

    for $j = 0, \ldots, D$

      partial[j] $= h_j(x_i)\left(\mathbb{1}[y_i = +1] - P(y=+1 \mid x_i, w^{(t)})\right)$

      (Each time, Pick different data point i)

    $w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta\, \text{partial}[j]$

  $t \leftarrow t+1$

Comparing computational time per step

Gradient ascent

$$\frac{\partial \ell(w)}{\partial w_j} = \sum_{i=1}^{N} \frac{\partial \ell_i(w)}{\partial w_j}$$

Stochastic gradient ascent

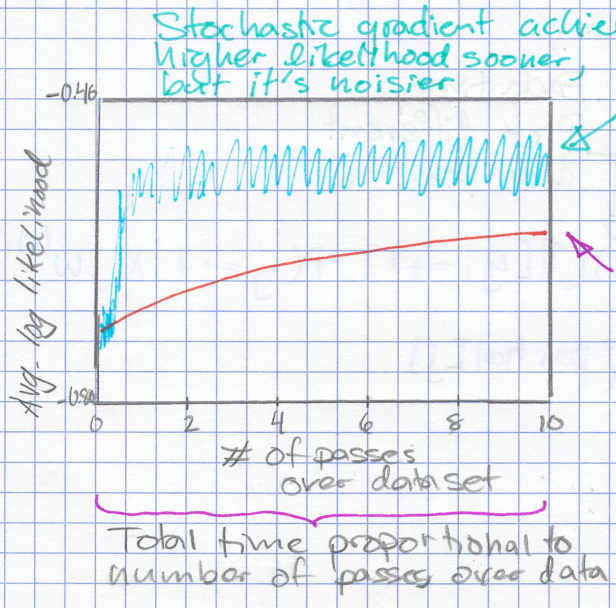
$$\frac{\partial \ell(w)}{\partial w_j} \approx \frac{\partial \ell_i(w)}{\partial w_j}$$

Time to compute contribution of x_i, y_i	# of data points (N)	total time for 1 step of gradient	Total time for 1 step of stochastic gradient
1 millisecond	1000	1 second	1 millisecond
1 second	1000	16.7 minutes	1 second
1 millisecond	10 million	2.8 hours	1 millisecond
1 millisecond	10 Billion	115.7 days	1 millisecond

Comparing gradient to stochastic gradient

Which one is better??? Depends...

Algorithm	Time per iteration	Total time to convergence for large data: In theory	Total time to convergence for large data: In practice	Sensitivity to parameters
Gradient	Slow for large data	Slower	Often slower	Moderate
Stochastic gradient	Always fast	Faster	Often faster	Very high



"Eventually, gradient catches up"

Summary of stochastic gradient

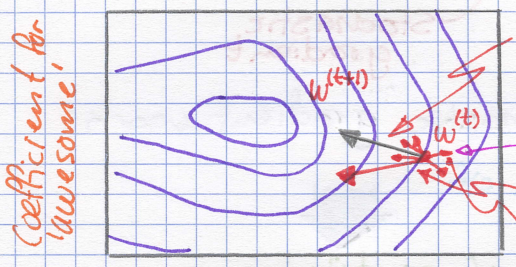
Tiny change to gradient ascent

Much better scalability

Huge impact in real-world

Very tricky to get right in practice

Why would stochastic gradient ever work?



exact gradient

For most data points, contribution points "up"

still OK
$\ell(w') > \ell(w^{(t)})$

Bad direction

Gradient is "best" direction, but any direction that goes "up" would be useful

In ML, steepest direction is sum of "little directions" from each data point

sum over data points

$$\frac{\partial \ell(w)}{\partial w_j} = \sum_{i=1}^{N} \frac{\partial \ell_i(w)}{\partial w_j}$$

contribution from each x_i, y_i

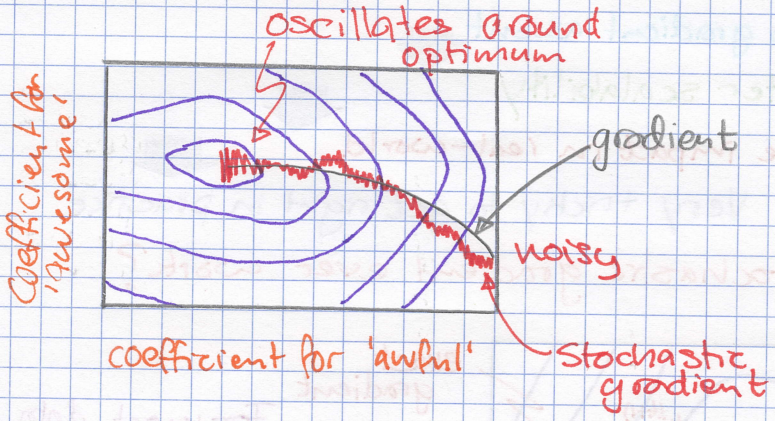
pick one direction

$$\frac{\partial \ell(w)}{\partial w_j} \approx \frac{\partial \ell_i(w)}{\partial w_j}$$

Most of the time, total likelihood will increase

Stochastic gradient ascent: Most iterations increase likelihood, but sometimes decrease it → On average, make progress

Convergence paths



Summary why stochastic gradient works

Gradient finds direction of steepest ascent

↓

Gradient is sum of contributions from each data point

↓

Stochastic gradient uses direction from 1 data point

↓

On average increases likelihood, sometimes decreases

↓

Stochastic gradient has "noisy" convergence

Stochastic gradient: practical tricks

Stochastic gradient ascent

Init $w^{(1)} = 0$, $t = 1$

until converged

for $i = 1, \ldots, N$

for $j = 0, \ldots, D$

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \frac{\partial \ell_i(w)}{\partial w_j}$$

$t \leftarrow t + 1$

Order of data can introduce bias

x[1] = #awesome	x[2] = #awful	y = sentiment
0	2	-1
3	3	-1
2	4	-1
0	3	-1
0	1	-1
2	1	+1
4	1	+1
1	1	+1
2	1	+1

Stochastic gradient updates parameters 1 data point at a time

Systematic order in data can introduce significant bias, eg, all negative points first, or temporal order, younger first, or...

Shuffle data before running stochastic gradient!

Stochastic gradient ascent

Shuffle data ← Before running stochastic gradient, make sure data is shuffled!

init $w^{(1)} = 0$, $t = 1$

until converged:

for $i = 1, \ldots, N$

for $j = 0, \ldots, D$

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \frac{\partial \ell_i(w)}{\partial w_j}$$

$t \leftarrow t + 1$

Choosing the step size η

Picking step size for stochastic gradient is very similar to picking step size for gradient

But stochastic gradient is a lot more unstable

If step size is too small, stochastic gradient slow to converge

If step size is too large, stochastic gradient oscillates

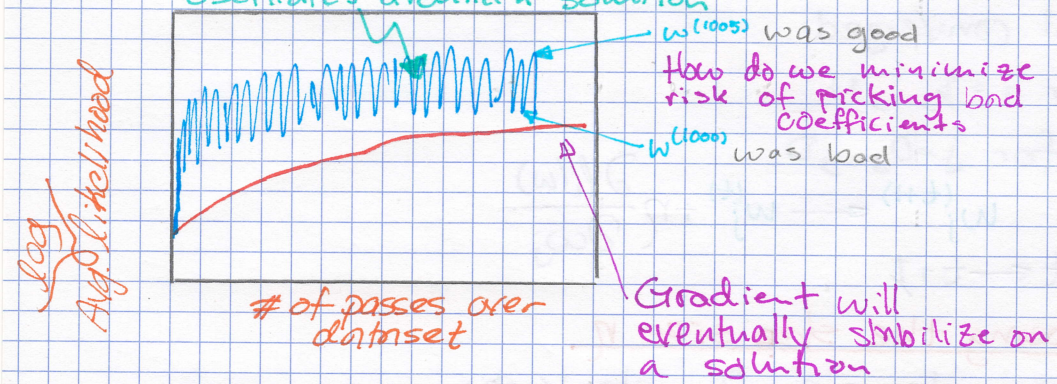
Simple rule of thumb for picking step size η similar to gradient

- Unfortunately, picking step size requires a lot, a lot of trial and error; much worst than gradient
- Try a several values, exponentially spaced
  - Goal: plot learning curves to
    - find one η that is too small
    - find one η that is too large
- Advanced tip: step size that decreases with iterations is very important for stochastic gradient e.g., $\eta_t = \frac{\eta_0}{t}$ ← constant; t ↳ iteration #

Don't trust the last coefficients

Stochastic gradient will never fully "converge"

Stochastic gradient will eventually oscillates around a solution



The last coefficients may be really good or really bad!!!

Stochastic gradient returns average coefficients

- Minimize noise: don't return last learned coefficients
- Instead, output average

$$\hat{w} = \frac{1}{T} \sum_{t=1}^{T} w^{(t)}$$

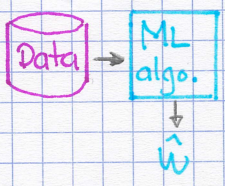
The online learning task

Online learning: Fitting models from streaming data

Batch vs online learning

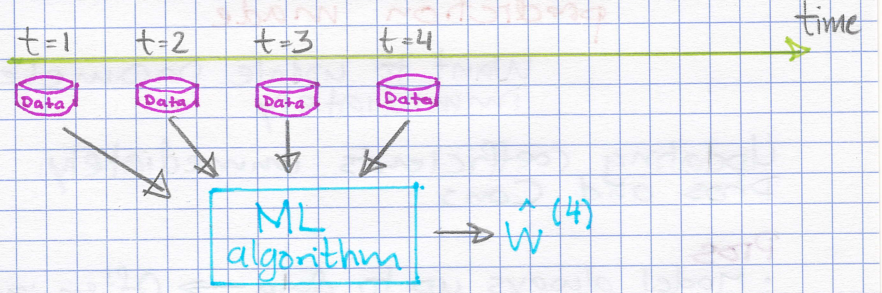
Batch learning

- All data is available at start of training time



Online learning

- Data arrives (streams in) over time
  - Must train model as data arrives!



Using stochastic gradient for online learning

Online learning problem

- Data arrives over each time step t:
  - Observe input x_t
    - Info of user, text of webpage
  - Make a prediction $\hat{y}_t$
    - Which ad to show
  - Observe true output y_t
    - Which ad user clicked on

Need ML algorithm to update coefficients each time step!

Stochastic gradient ascent can be used for online learning

- init $w^{(1)} = 0, t = 1$
- Each time step t
  - Observe input x_t
  - Make a prediction $\hat{y}_t$
  - Observe true output y_t
  - Update coefficients

for $j = 0, \ldots, D$

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \frac{\partial \ell_t(w)}{\partial w_j}$$

Summary of online learning

Data arrives over time

Must make a prediction every time new data point arrives

Observe true class after prediction made

Want to update parameters immediately

Updating coefficients immediately: Pros and Cons

Pros

- Model always up to data → Often more accurate
- Lower computational cost
- Don't need to store all data, but often do anyway

Cons

- Overall system is *much* more complex
  - Bad real-world cost in terms of $$$ to build & maintain

Most companies opt for systems that save data and update coefficients every night, or hour, week, ...

Scaling to huge datasets through parallelization

Multicore processors

Computer clusters

↓

Huge opportunity for scaling

↓

Require parallel & distributed ML algorithms