Scaling to Huge Datasets & Online Learning

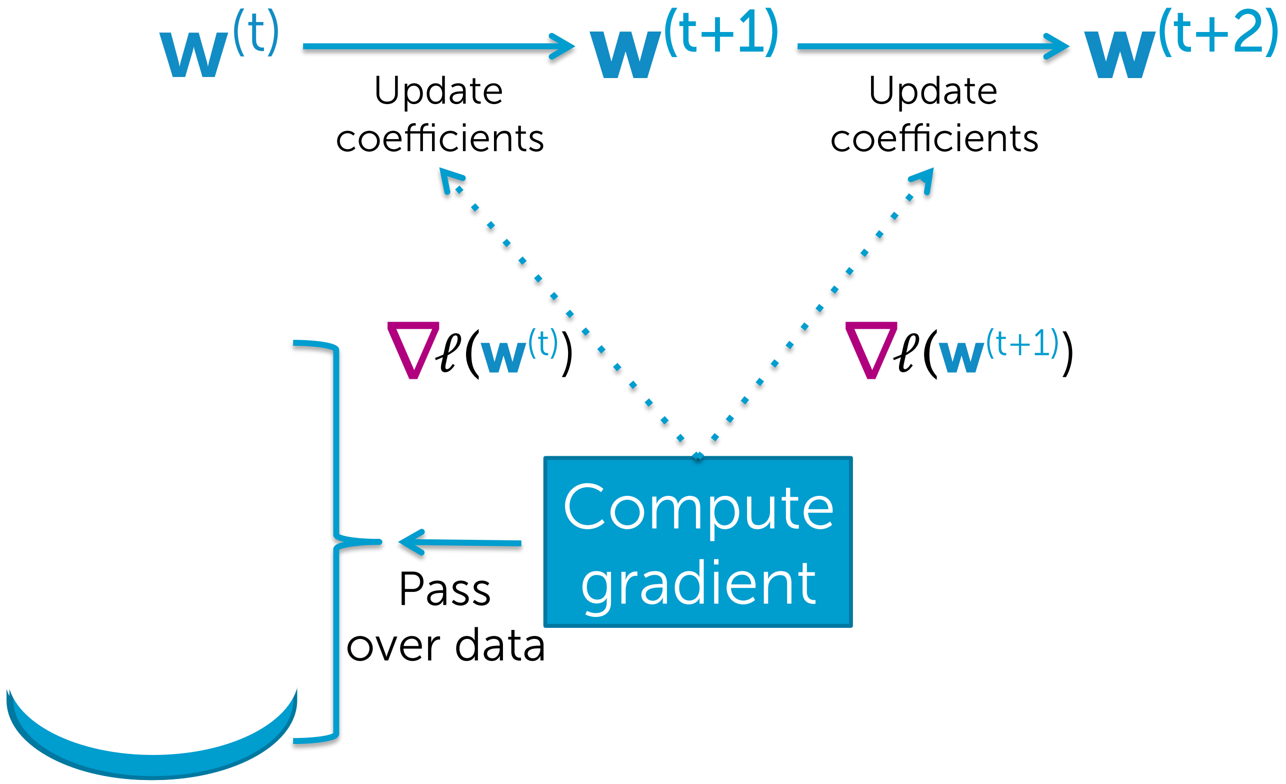
Emily Fox & Carlos Guestrin

Machine Learning Specialization

University of Washington

Why gradient ascent is slow…

Every update requires a full pass over data



$\mathbf{w}^{(t)}$ $\mathbf{w}^{(t+1)}$ $\mathbf{w}^{(t+2)}$

Update coefficients

Update coefficients

$\nabla\ell(\mathbf{w}^{(t)})$

$\nabla\ell(\mathbf{w}^{(t+1)})$

Compute gradient

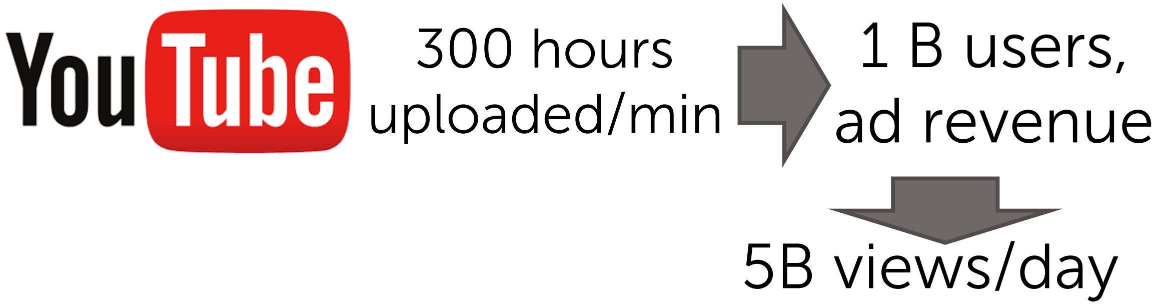
Pass over data

Data sets are getting huge, and we need them!

WWW
4.8B webpages

twitter
500M Tweets/day

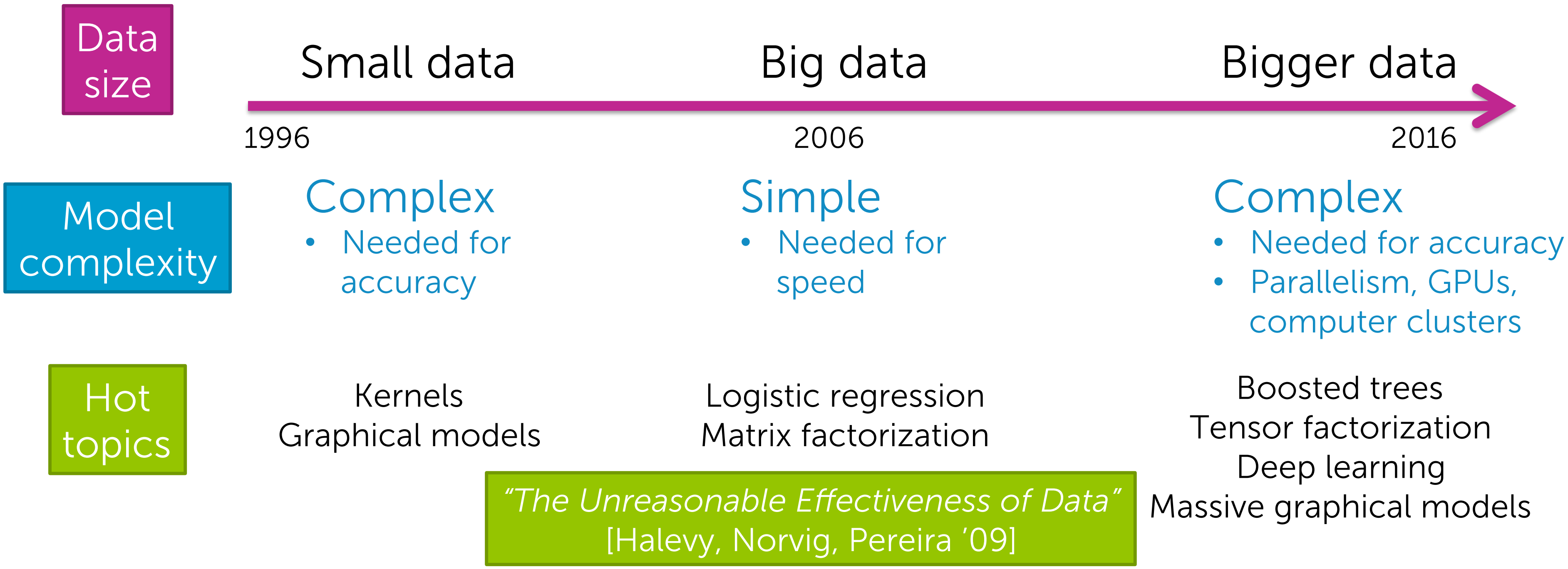
Internet of Things
Sensors everywhere



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

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

ML improves (significantly) with bigger datasets



Data size

Small data

Big data

Bigger data

1996

2006

2016

Model complexity

Complex
- Needed for accuracy

Simple
- Needed for speed

Complex
- Needed for accuracy
- Parallelism, GPUs, computer clusters

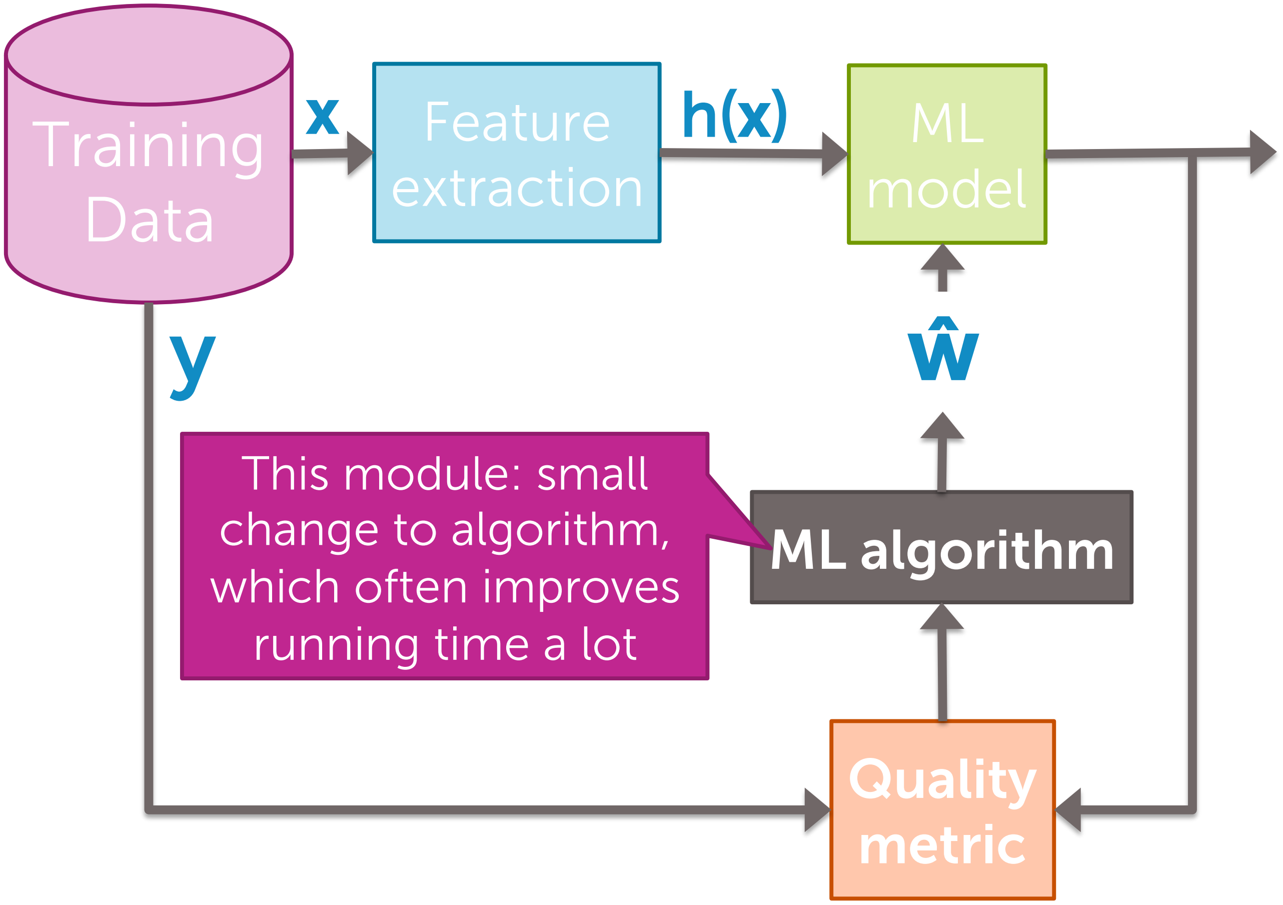
Hot topics

Kernels
Graphical models

Logistic regression
Matrix factorization

Boosted trees
Tensor factorization
Deep learning
Massive graphical models

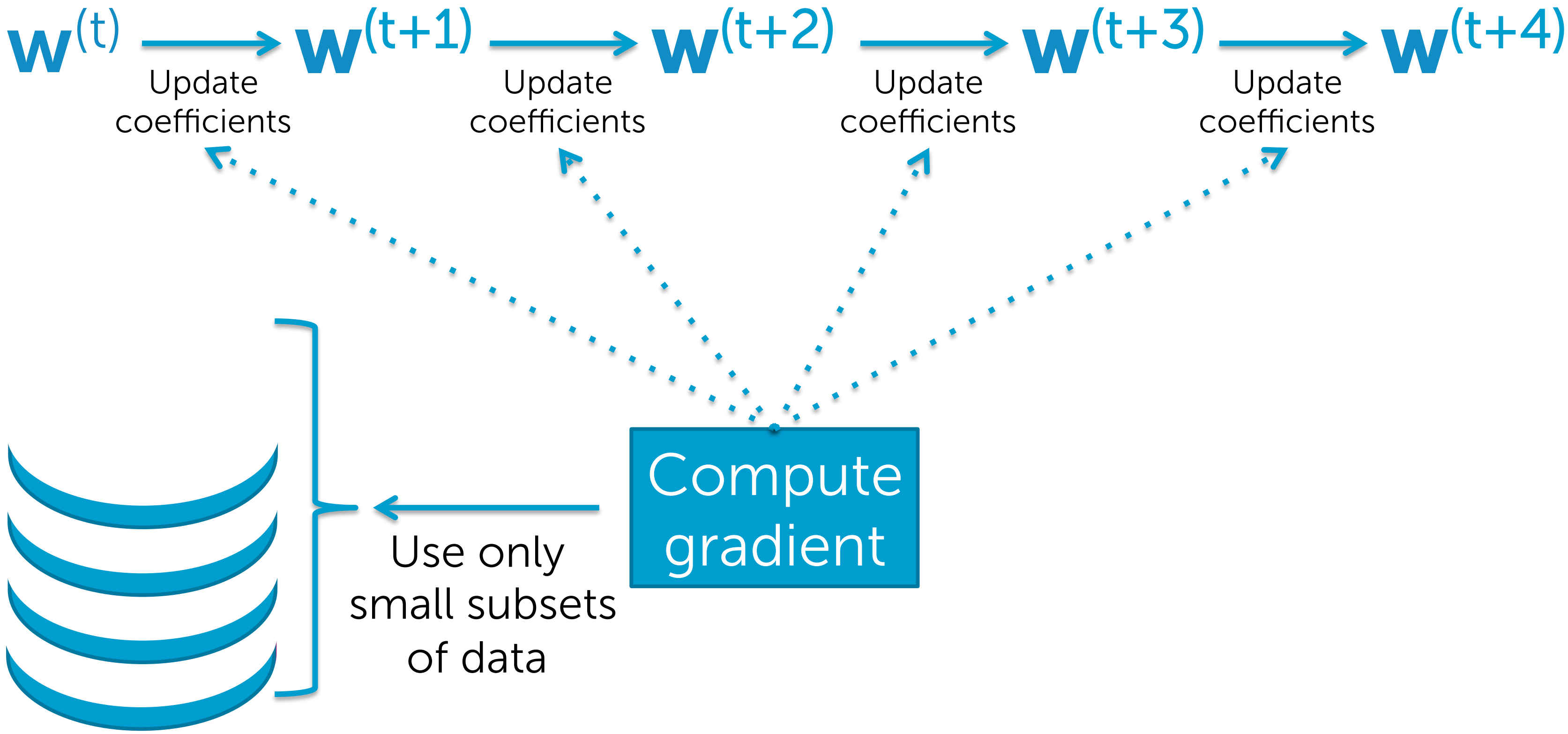
"The Unreasonable Effectiveness of Data"
[Halevy, Norvig, Pereira '09]



This module: small change to algorithm, which often improves running time a lot

Stochastic gradient ascent

Many updates for each pass over data



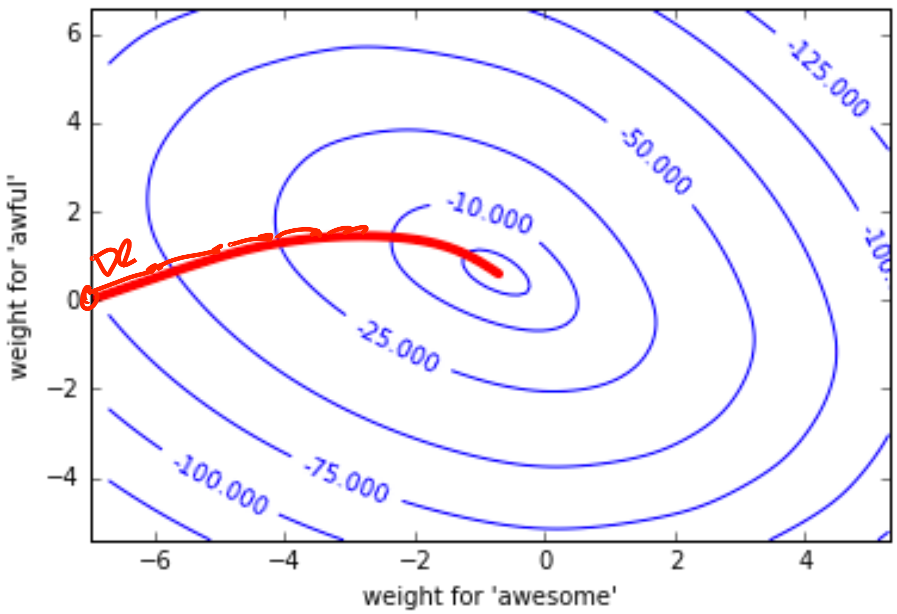
$\mathbf{w}^{(t)} \xrightarrow{\text{Update coefficients}} \mathbf{w}^{(t+1)} \xrightarrow{\text{Update coefficients}} \mathbf{w}^{(t+2)} \xrightarrow{\text{Update coefficients}} \mathbf{w}^{(t+3)} \xrightarrow{\text{Update coefficients}} \mathbf{w}^{(t+4)}$

Compute gradient

Use only small subsets of data

Learning, one data point at a time

Gradient ascent



Algorithm:

```
while not converged
    $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \eta \nabla \ell(\mathbf{w}^{(t)})$
```

How expensive is gradient ascent?

Sum over data points

$$\frac{\partial \ell(\mathbf{w})}{\partial \mathbf{w}_j} = \sum_{i=1}^{N} \underbrace{\underbrace{h_j(\mathbf{x}_i)\Big(\mathbb{1}[y_i=+1] - P(y=+1 \mid \mathbf{x}_i,\mathbf{w})\Big)}_{\text{Contribution of data point } x_i, y_i \text{ to gradient}}}_{\boxed{\frac{\partial \ell_i(\mathbf{w})}{\partial \mathbf{w}_j}}}$$

Every step requires touching every data point!!!

Sum over data points

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

Time to compute contribution of $\mathbf{x}_i,\mathbf{y}_i$	# of data points (N)	Total time to compute 1 step of gradient ascent
1 millisecond	1000	1 sec
1 second	1000	16.7 min
1 millisecond	10 million	2.8 hours
1 millisecond	10 billion	115.7 days

Instead of all data points for gradient, use 1 data point only???

Gradient ascent

Sum over data points

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

Stochastic gradient ascent

Each time, pick different data point i

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

Stochastic gradient ascent

Stochastic gradient ascent for logistic regression

```
init $\mathbf{w}^{(1)}=0$, t=1
~~until converged~~ for i=1,…,N
    for j=0,…,D
        partial[j] = $\cancel{\sum_{i=1}^{N}} h_j(\mathbf{x}_i)\Big(\mathbb{1}[y_i=+1] - P(y=+1 \mid \mathbf{x}_i, \mathbf{w}^{(t)})\Big)$
        $w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta$ partial[j]
    t ← t + 1
```

Sum over data points

Each time, pick different data point i

Comparing computational time per step

Gradient ascent

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

Stochastic gradient ascent

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

Time to compute contribution of $\mathbf{x}_i$,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 sec
1 millisecond	10 million	2.8 hours	1 millisec
1 millisecond	10 billion	115.7 days	1 millisec

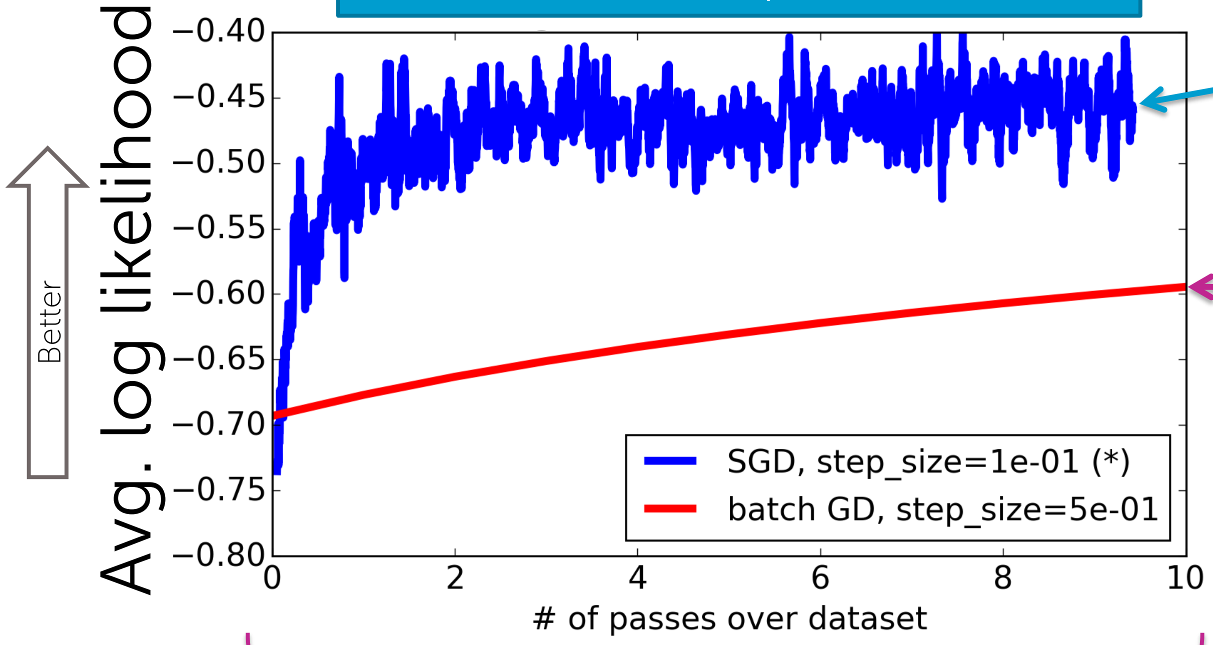
Comparing gradient to stochastic gradient

Which one is better??? Depends…

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

Comparing gradient to stochastic gradient

Stochastic gradient achieves higher likelihood sooner, but it's noisier



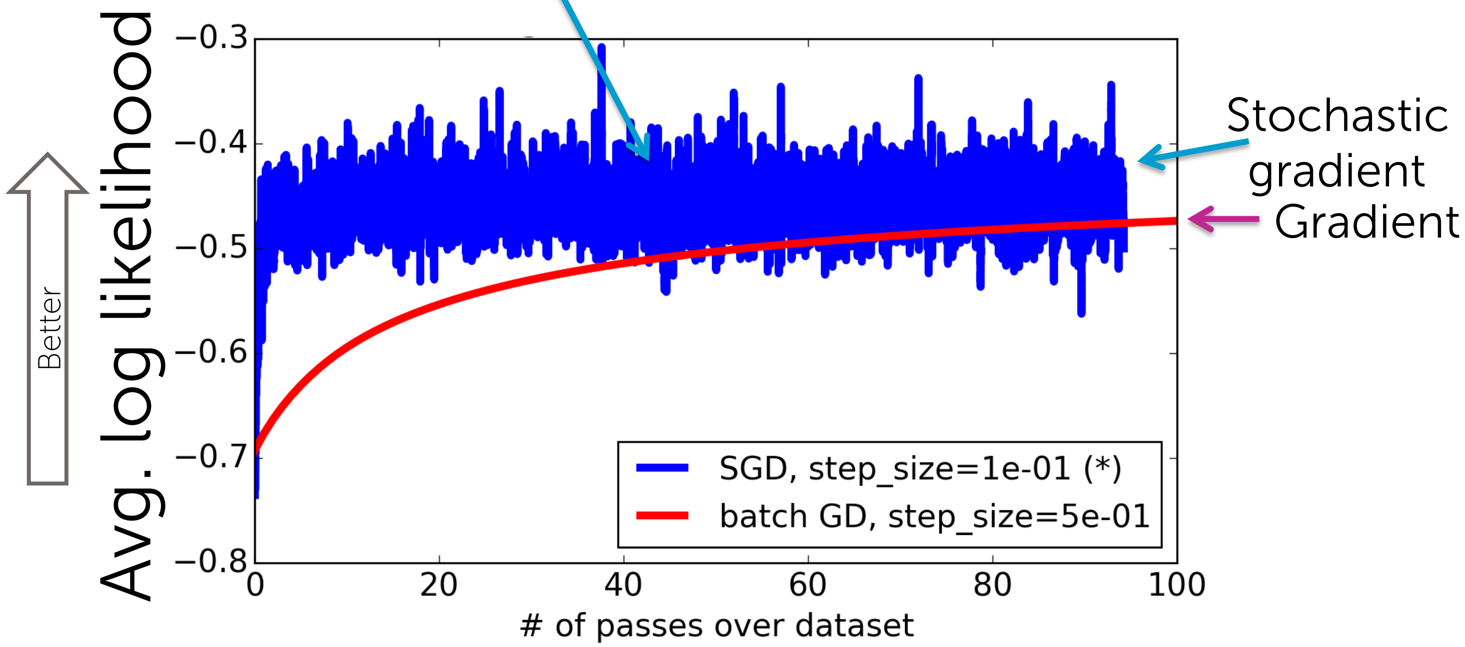
Stochastic gradient

Gradient

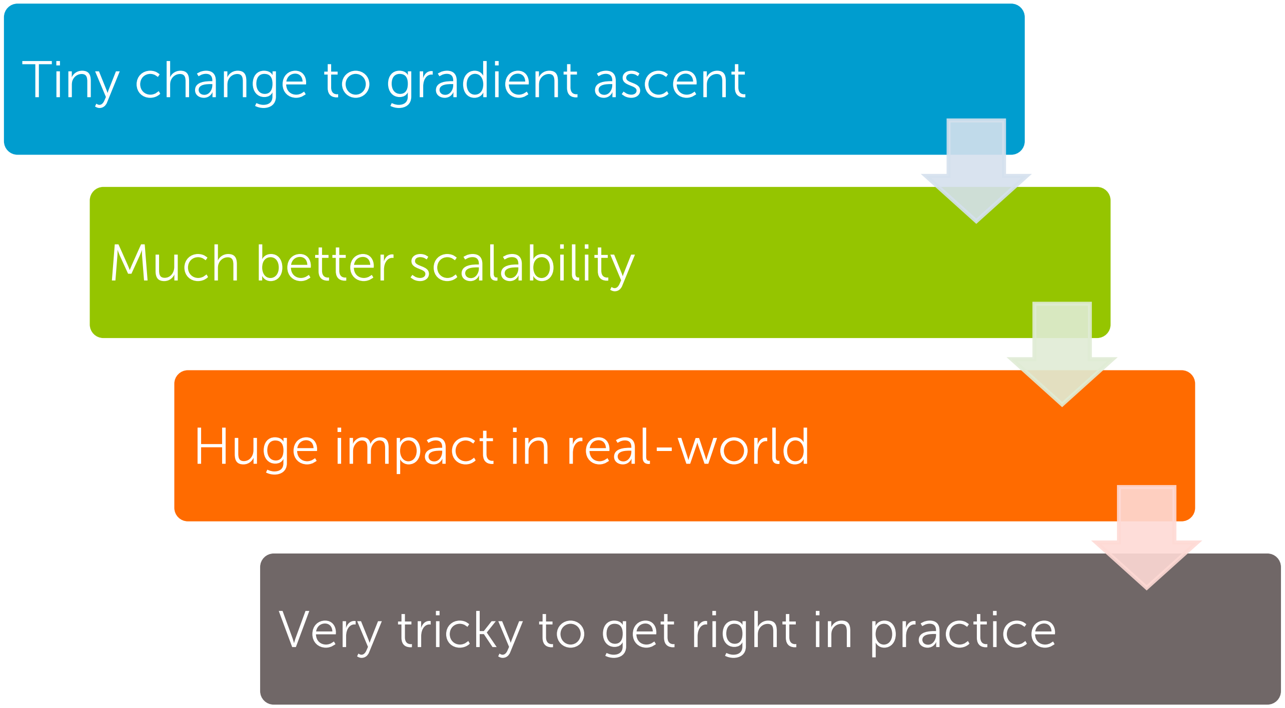
Total time proportional to number of passes over data

Eventually, gradient catches up

Note: should only trust "average" quality of stochastic gradient (more discussion later)



Summary of stochastic gradient



Tiny change to gradient ascent

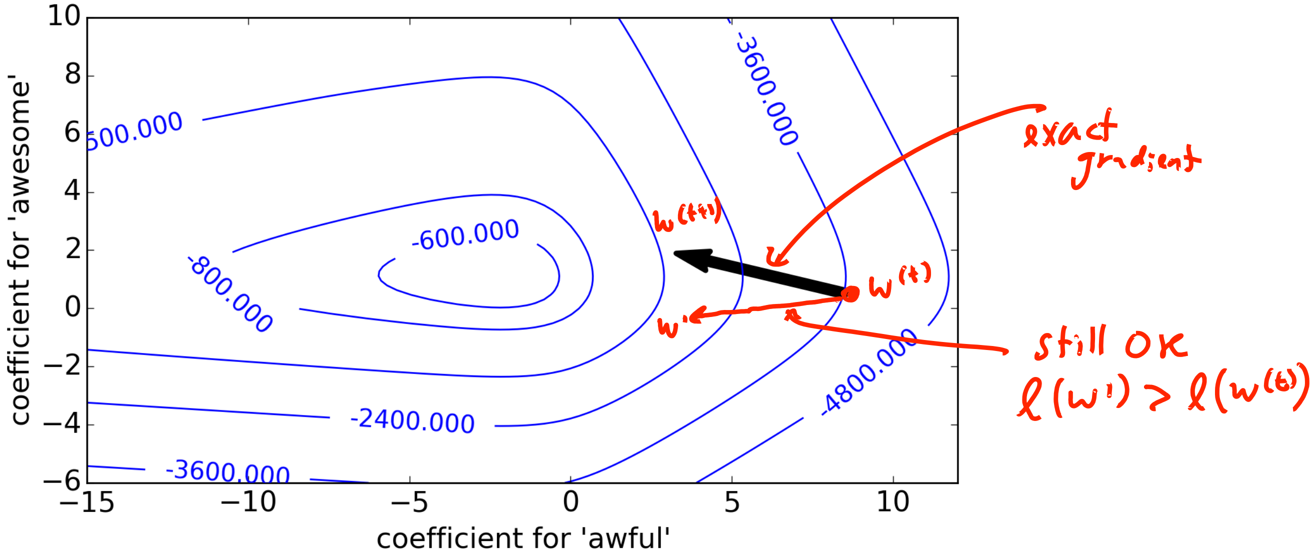
Much better scalability

Huge impact in real-world

Very tricky to get right in practice

Why would stochastic gradient ever work???

Gradient is direction of steepest ascent

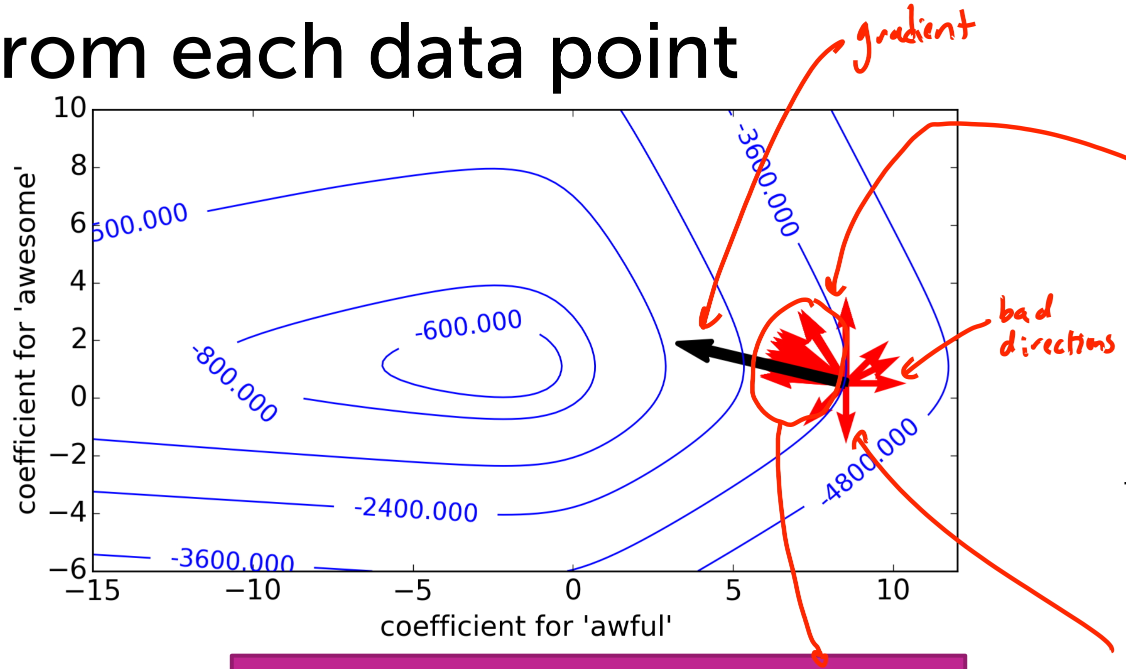


exact gradient

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

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

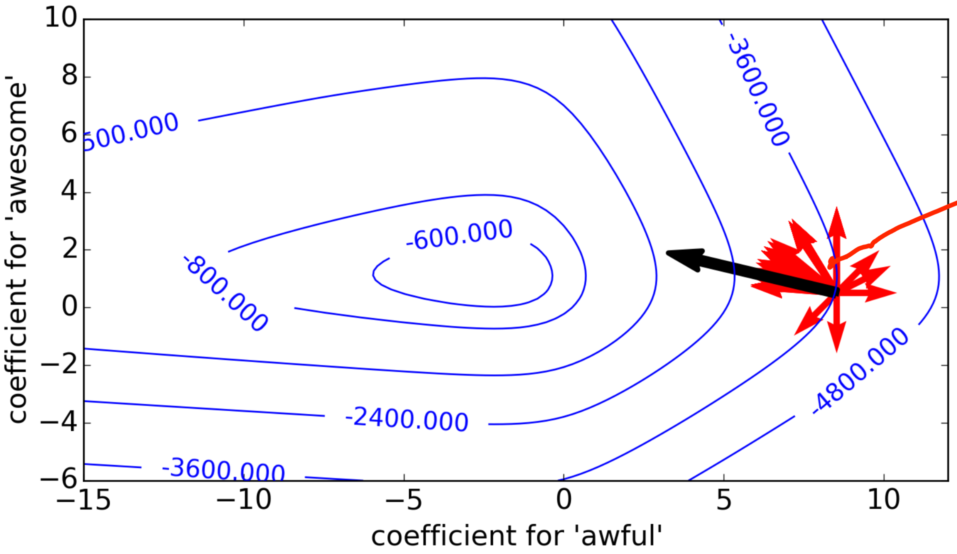


Sum over data points

$$\frac{\partial \ell(\mathbf{w})}{\partial \mathbf{w}_j} = \sum_{i=1}^{N} \underbrace{\frac{\partial \ell_i(\mathbf{w})}{\partial \mathbf{w}_j}}_{\text{contribution from each } x_i y_i}$$

For most data points, contribution points "up"

Stochastic gradient: pick a data point and move in direction

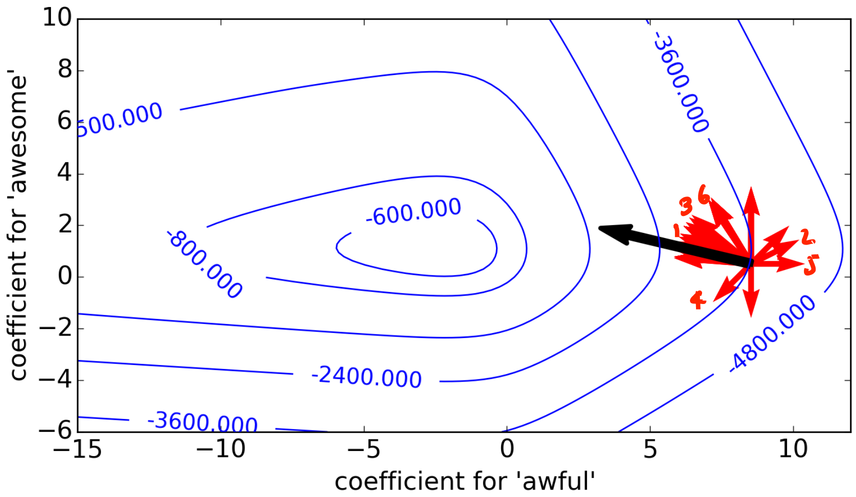


pick one of these

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

Most of the time, total likelihood will increase

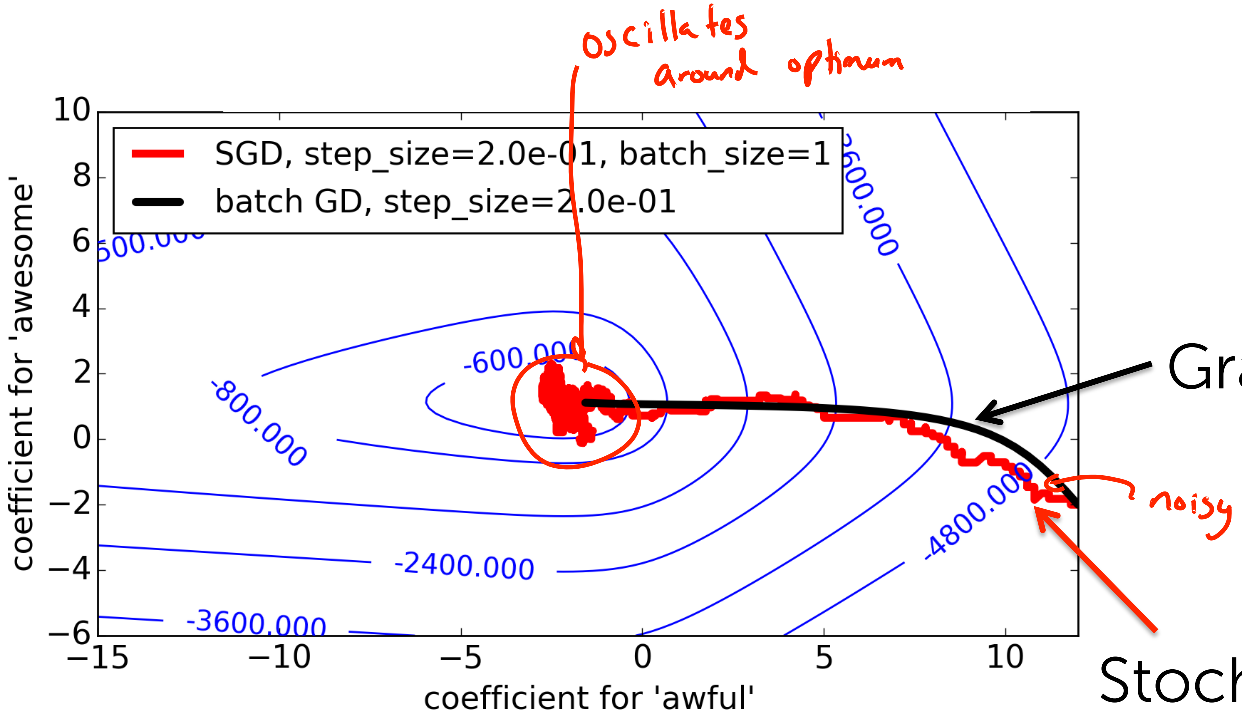
Stochastic gradient ascent: Most iterations increase likelihood, but sometimes decrease it ➔ On average, make progress



```
until converged
    for i=1,...,N
        for j=0,...,D
            $w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \frac{\partial \ell_i(\mathbf{w})}{\partial \mathbf{w}_j}$
        t ← t + 1
```

Convergence path

Convergence paths



oscillates around optimum

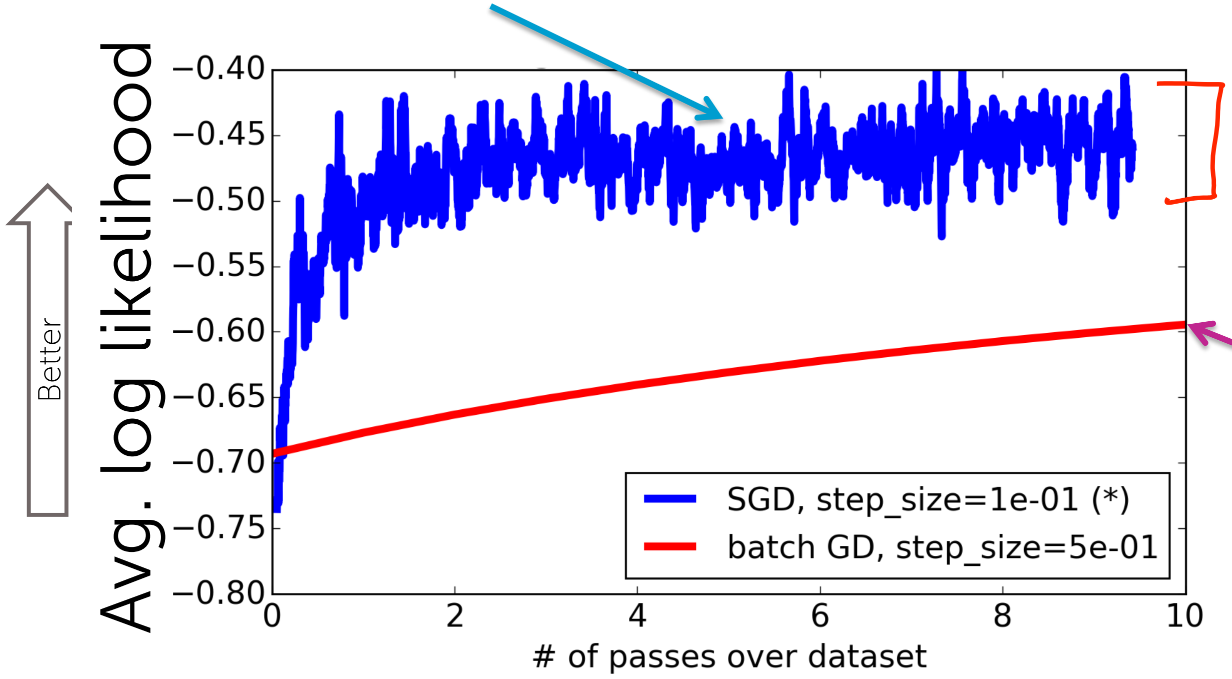
noisy

Gradient

Stochastic gradient

Stochastic gradient convergence is “noisy”

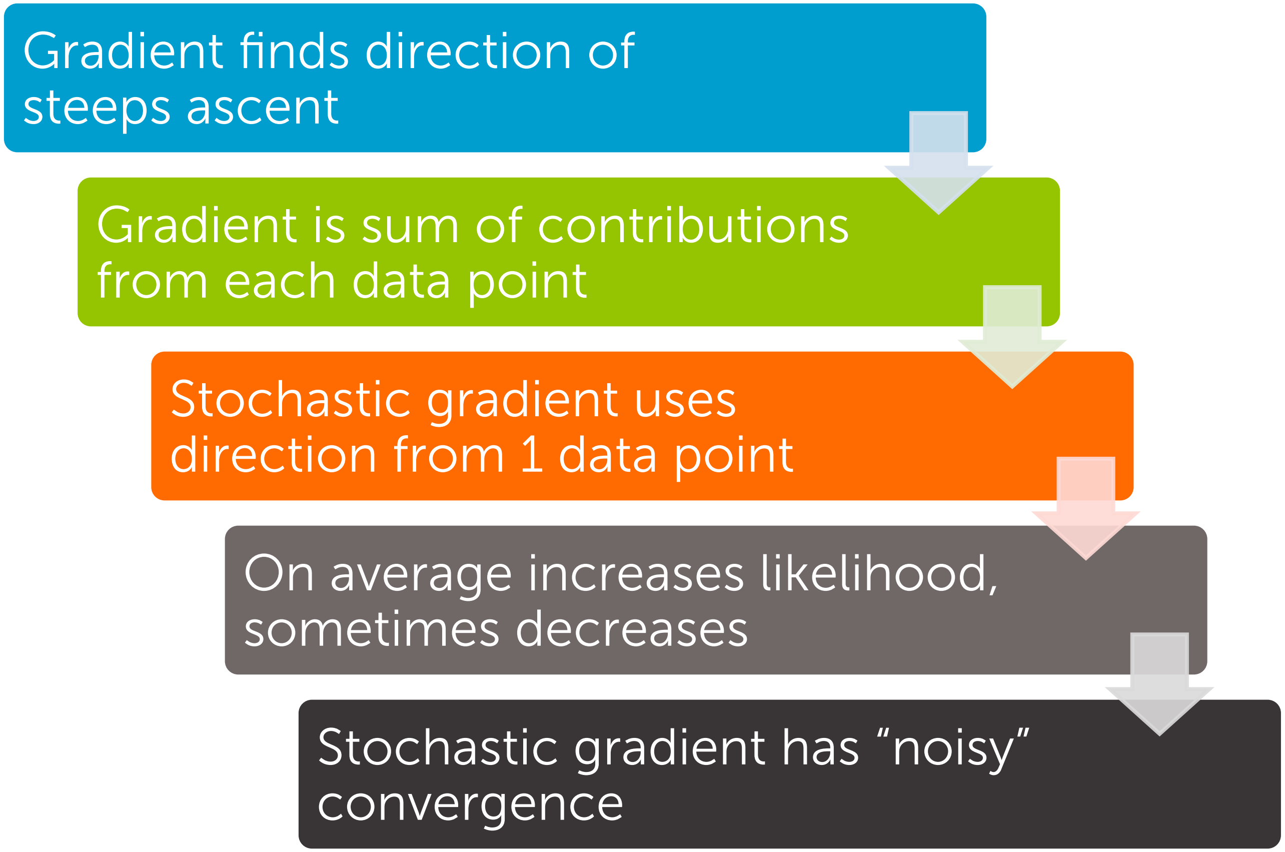
Stochastic gradient makes “noisy” progress



oscillate around optimum

Gradient usually increases likelihood smoothly

Summary of why stochastic gradient works



Gradient finds direction of steeps 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 $\mathbf{w}^{(1)}=0$, $t=1$
until converged
    for $i=1,\ldots,N$
        for $j=0,\ldots,D$
            $w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \frac{\partial \ell_i(\mathbf{w})}{\partial \mathbf{w}_j}$
        $t \leftarrow t + 1$
```

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,
e.g., all negative points first, or temporal order, younger first, or …

Shuffle data before running stochastic gradient!

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

Shuffle rows

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

Stochastic gradient ascent

Before running stochastic gradient, make sure data is shuffled

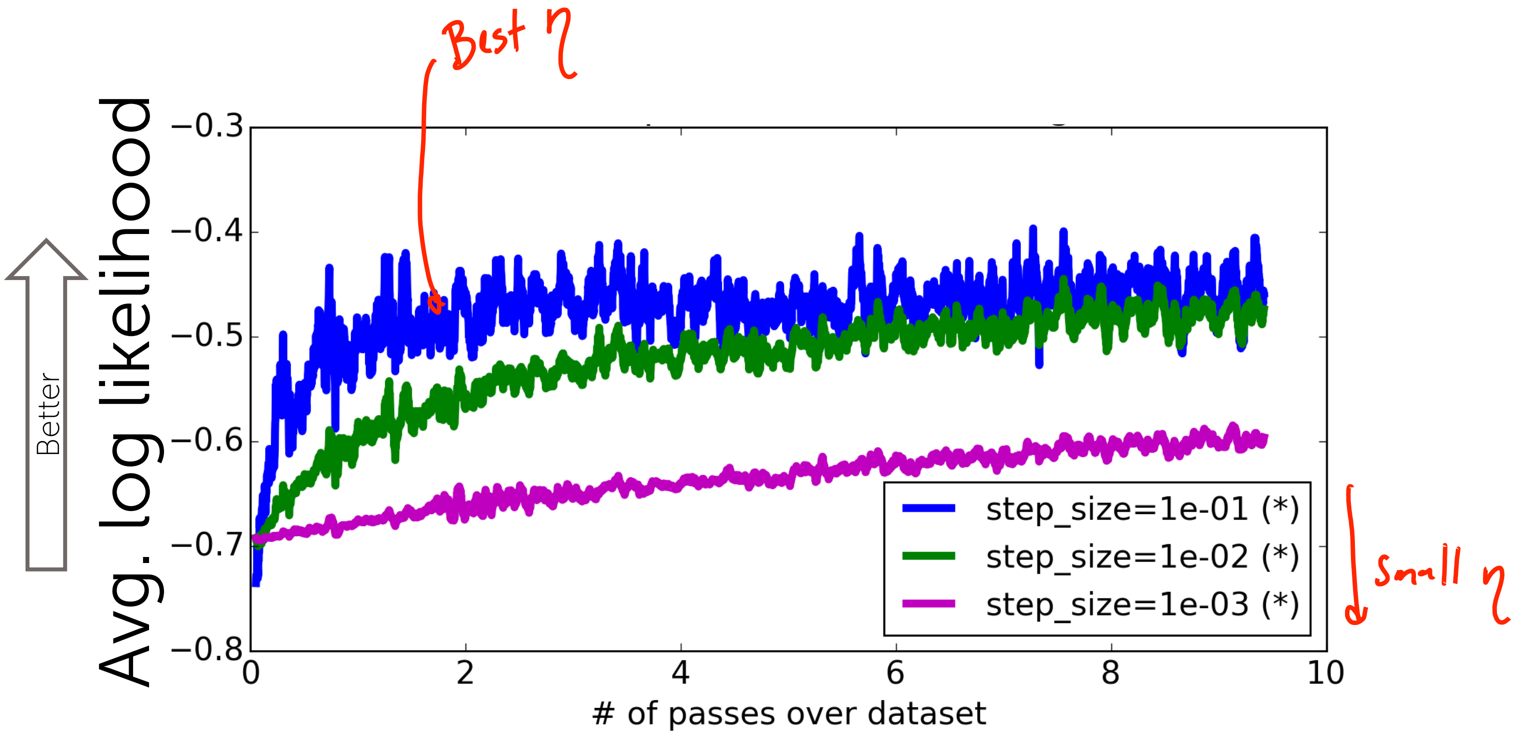
```
Shuffle data
init $\mathbf{w}^{(1)}=0$, t=1
until converged
    for i=1,...,N
        for j=0,...,D
            $w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \frac{\partial \ell_i(\mathbf{w})}{\partial \mathbf{w}_j}$
        t ← t + 1
```

Choosing the step size η

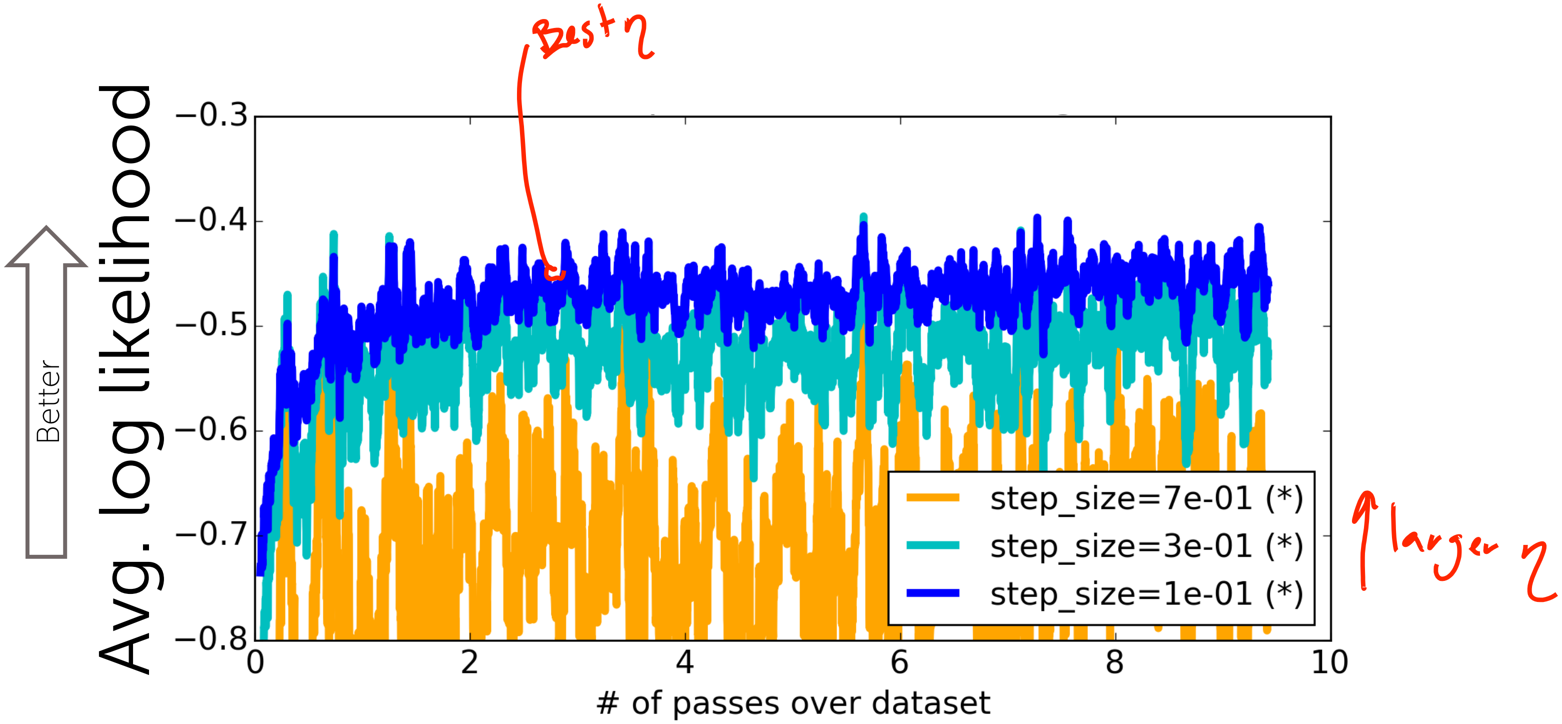
Picking step size
for **stochastic gradient**
is very similar to
picking step size
for **gradient**

But stochastic gradient
is a lot more unstable… ☹

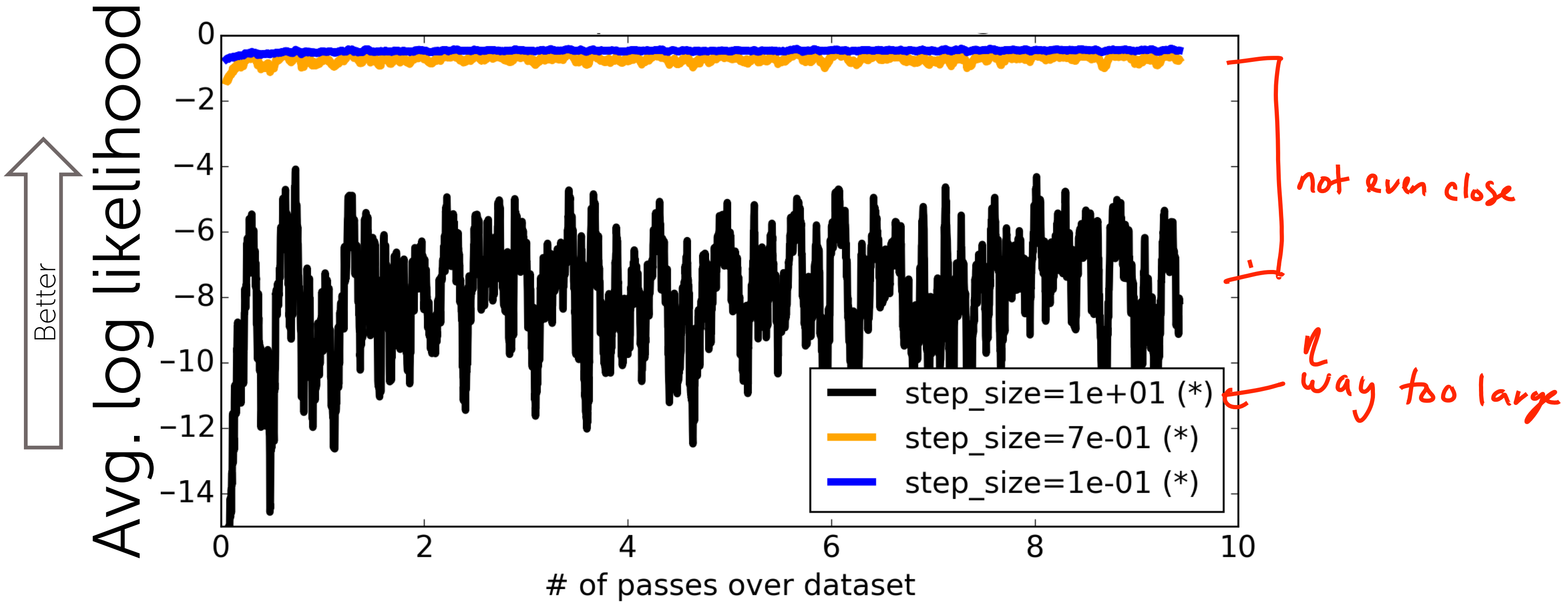
If step size is too small, stochastic gradient slow to converge



If step size is too large, stochastic gradient oscillates



If step size is very large, stochastic gradient goes crazy ☹



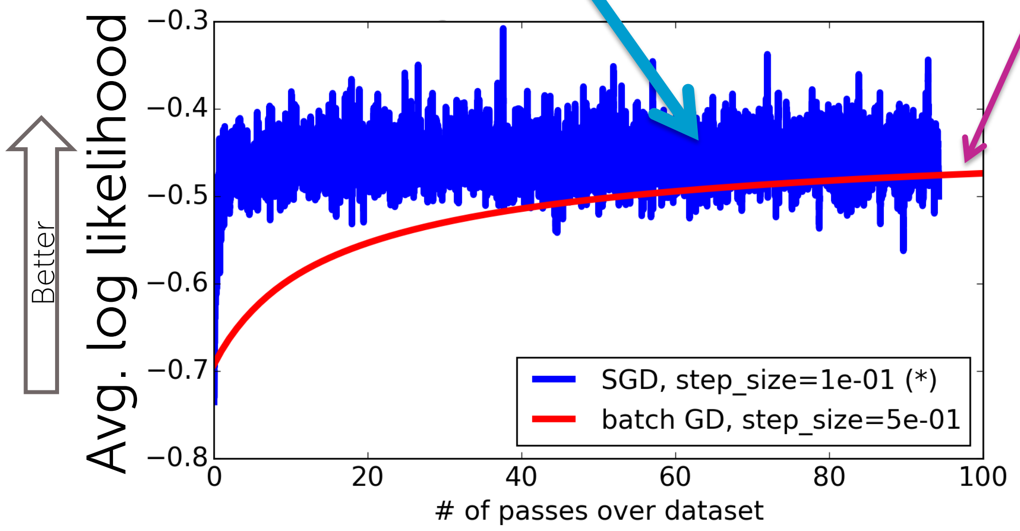
Simple rule of thumb for picking step size η similar to gradient

- 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., $\eta_t = \frac{\eta_0 \leftarrow \text{constant}}{t \leftarrow \text{iteration \#}}$

Don’t trust the last coefficients… ☹

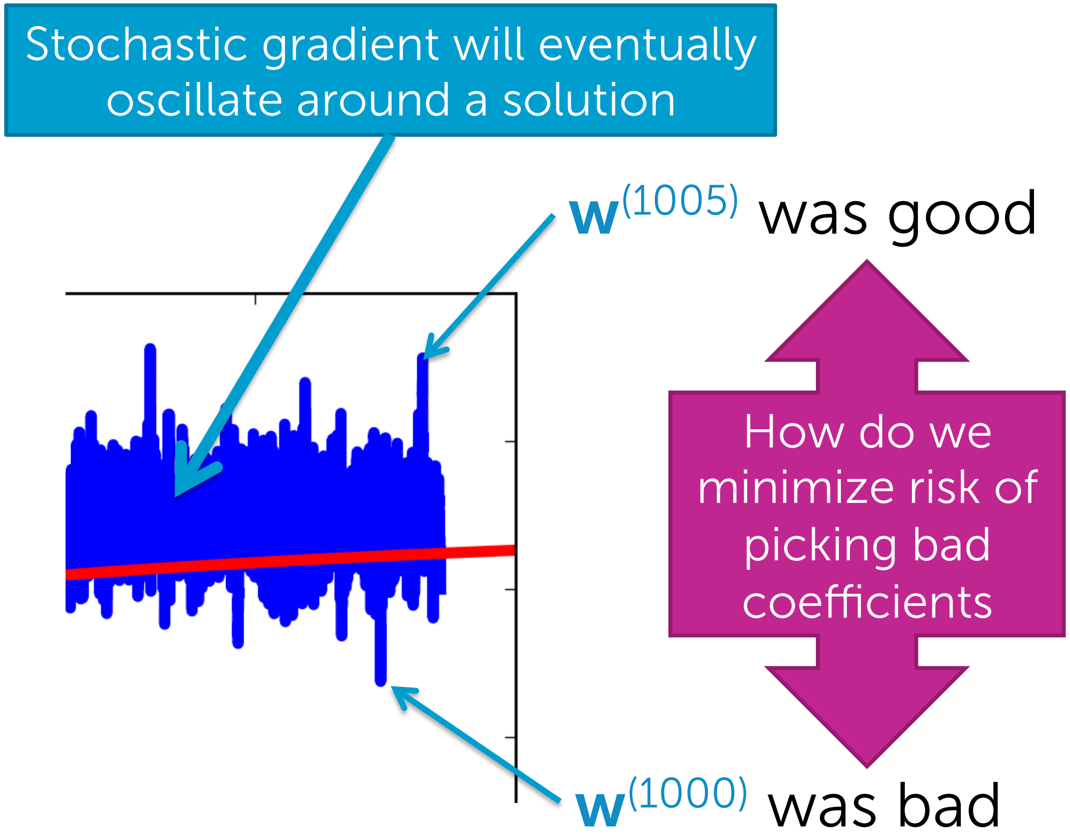
Stochastic gradient never fully "converges"



Stochastic gradient will eventually oscillate around a solution

Gradient will eventually stabilize on a solution

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



Stochastic gradient will eventually oscillate around a solution

$\mathbf{w}^{(1005)}$ was good

How do we minimize risk of picking bad coefficients

$\mathbf{w}^{(1000)}$ was bad

Stochastic gradient returns average coefficients

- Minimize noise:
  don't return last learned coefficients

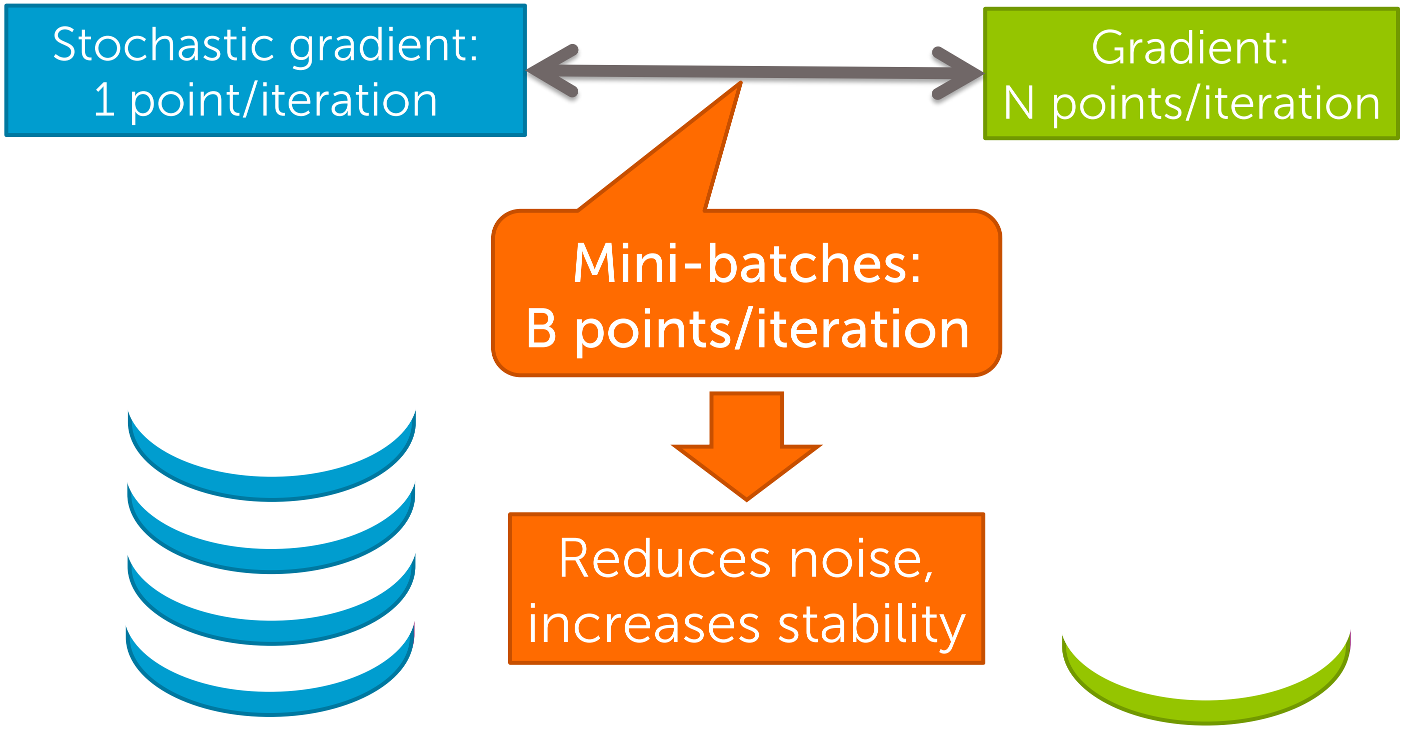
- Instead, output average:

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

Learning from batches of data

OPTIONAL

Gradient/stochastic gradient: two extremes



Stochastic gradient: 1 point/iteration

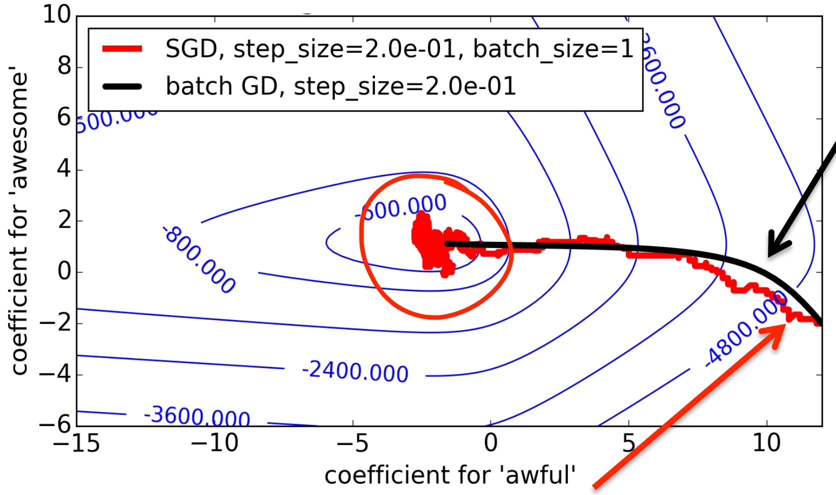
Gradient: N points/iteration

Mini-batches: B points/iteration

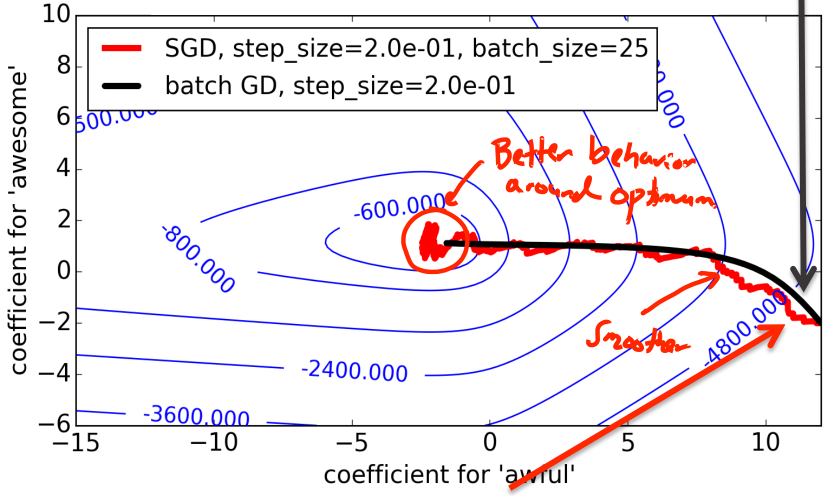
Reduces noise, increases stability

Convergence paths

Gradient



Stochastic gradient
Batch size = 1



Better behavior around optimum

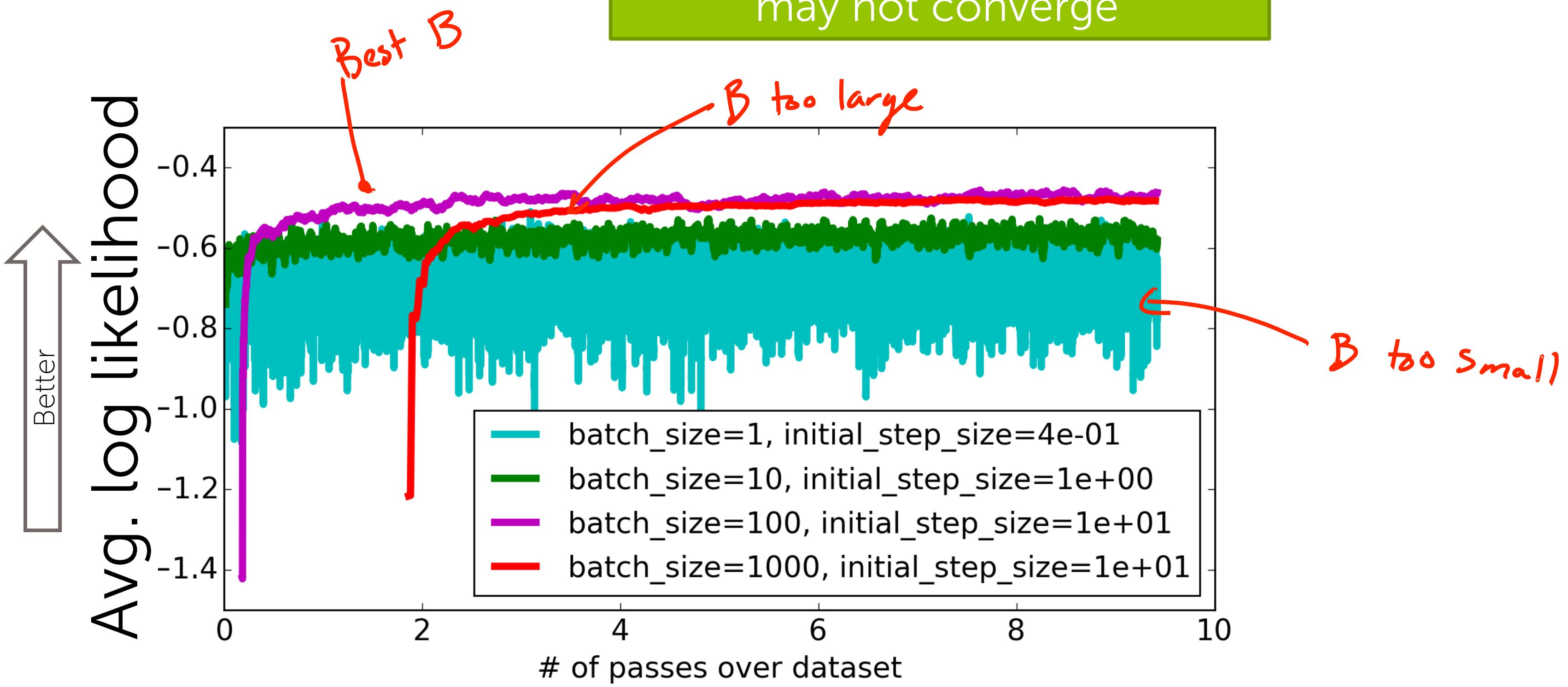
Smoother

Stochastic gradient
Batch size = 25

Batch size effect

Too large → Slow convergence

Too small → Noisy, may not converge



Stochastic gradient ascent with mini-batches

```
Shuffle data
init $\mathbf{w}^{(1)}=0$, t=1
until converged
    for k=0,…,N/B-1
        for j=0,…,D
            $w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \sum_{i=1+k*B}^{(k+1)*B} \frac{\partial \ell_i(\mathbf{w})}{\partial \mathbf{w}_j}$
        t ← t + 1
```

For each mini-batch

Sum over data points in mini-batch k

Measuring convergence

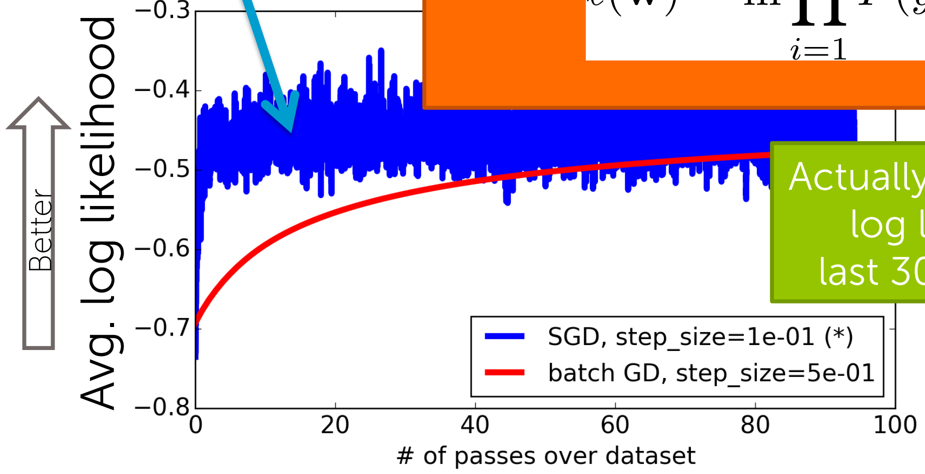
OPTIONAL

How did we make these plots???

Need to compute log likelihood of data at every iteration???

➔ Really really slow, product over all data points!

$$\ell(\mathbf{w}) = \ln \prod_{i=1}^{N} P(y_i \mid \mathbf{x}_i, \mathbf{w})$$



Actually, plotting average log likelihood over last 30 mini-batches…

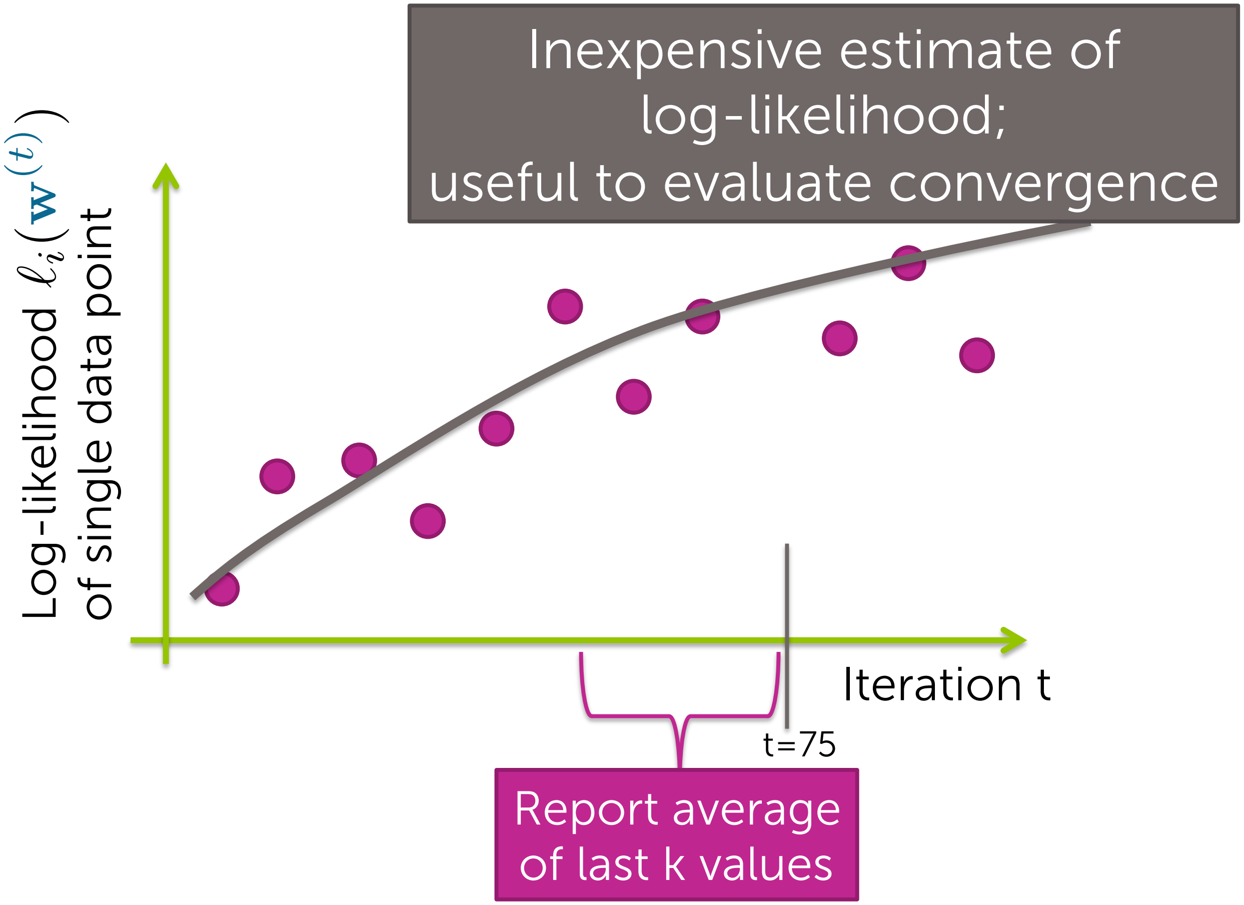
Computing log-likelihood during run of stochastic gradient ascent

Log-likelihood of data point i is simply:

$$\ell_i(\mathbf{w}^{(t)}) = \begin{cases} \ln P(y=+1 \mid \mathbf{x}_i, \mathbf{w}^{(t)}), & \text{if } y_i = +1 \\ \ln\left(1 - P(y=+1 \mid \mathbf{x}_i, \mathbf{w}^{(t)})\right), & \text{if } y_i = -1 \end{cases}$$

```
init $\mathbf{w}^{(1)}$=0, t=1
until converged
    for i=1,…,N
        for j=0,…,D
            partial[j] = $h_j(\mathbf{x}_i)\left(\mathbb{1}[y_i=+1] - P(y=+1 \mid \mathbf{x}_i, \mathbf{w}^{(t)})\right)$
            $w_j^{(t+1)}$ ← $w_j^{(t)}$ + η partial[j]
        t ← t + 1
```

Estimate log-likelihood with sliding window

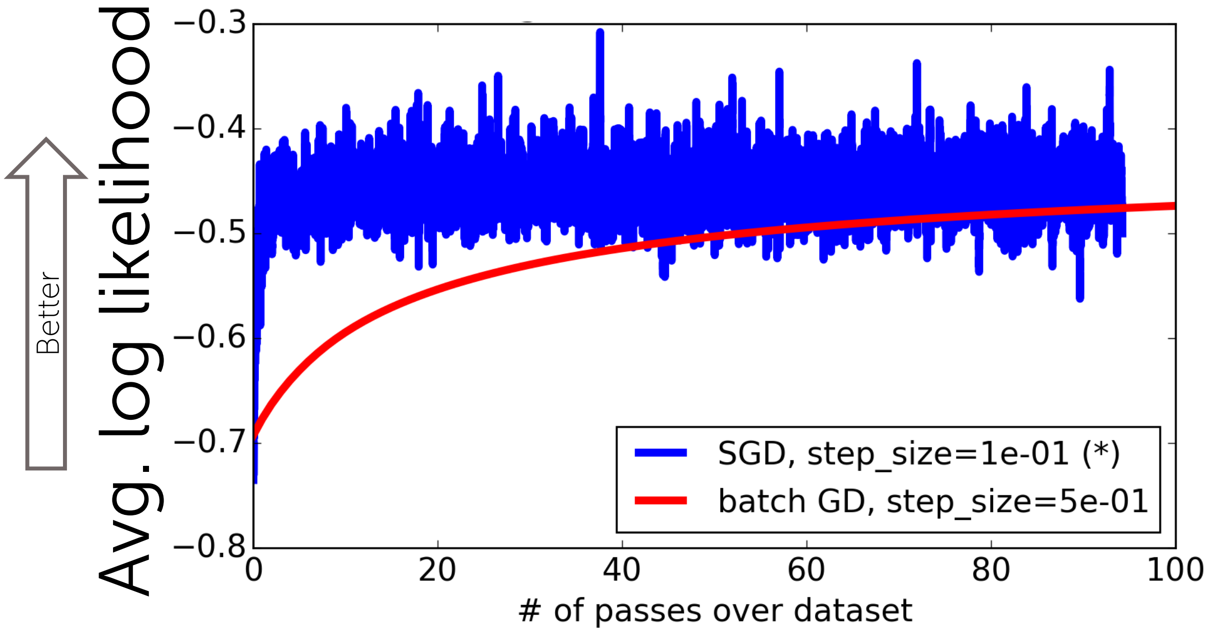


Inexpensive estimate of log-likelihood; useful to evaluate convergence

Report average of last k values

That’s what average log-likelihood meant… ☺

(In this case, over last k=30 mini-batches, with batch-size B = 100)



Adding regularization

OPTIONAL

Consider specific total cost

$$\max_{w}\ \text{Total quality} = \underbrace{\text{measure of fit}}_{\ell(\mathbf{w})} - \underbrace{\text{measure of magnitude of coefficients}}_{||\mathbf{w}||_2^2}$$

$\uparrow$ log data likelihood

$\uparrow$ L_2 penalty

Gradient of L_2 regularized log-likelihood

Total quality =

$$\underbrace{\text{measure of fit}}_{\ell(\mathbf{w})} - \underbrace{\text{measure of magnitude of coefficients}}_{\lambda \|\mathbf{w}\|_2^2}$$

Total derivative $= \displaystyle\sum_{i=1}^{N} \frac{\partial \ell_i(\mathbf{w})}{\partial \mathbf{w}_j} - 2\,\lambda\, w_j$

Stochastic gradient for regularized objective

$$\text{Total derivative} = \sum_{i=1}^{N} \frac{\partial \ell_i(\mathbf{w})}{\partial \mathbf{w}_j} - 2\,\lambda\, w_j$$

- What about regularization term?

Each time, pick different data point i

Stochastic gradient ascent

$$\text{Total derivative} \approx \frac{\partial \ell_i(\mathbf{w})}{\partial \mathbf{w}_j} - \frac{2}{N}\,\lambda\, w_j$$

Each data point contributes 1/N to regularization

Stochastic gradient ascent with regularization

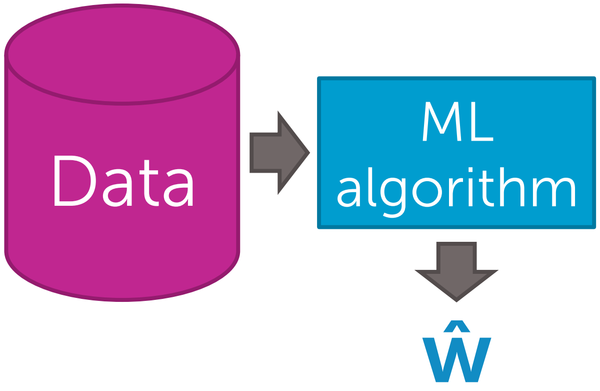
```
Shuffle data
init $\mathbf{w}^{(1)}=0$, t=1
until converged
    for i=1,…,N
    for j=0,…,D
        $w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \left( \frac{\partial \ell_i(\mathbf{w})}{\partial \mathbf{w}_j} - \frac{2}{N} \lambda w_j \right)$
    t ← t + 1
```

Online learning: Fitting models from streaming data

Batch vs online learning

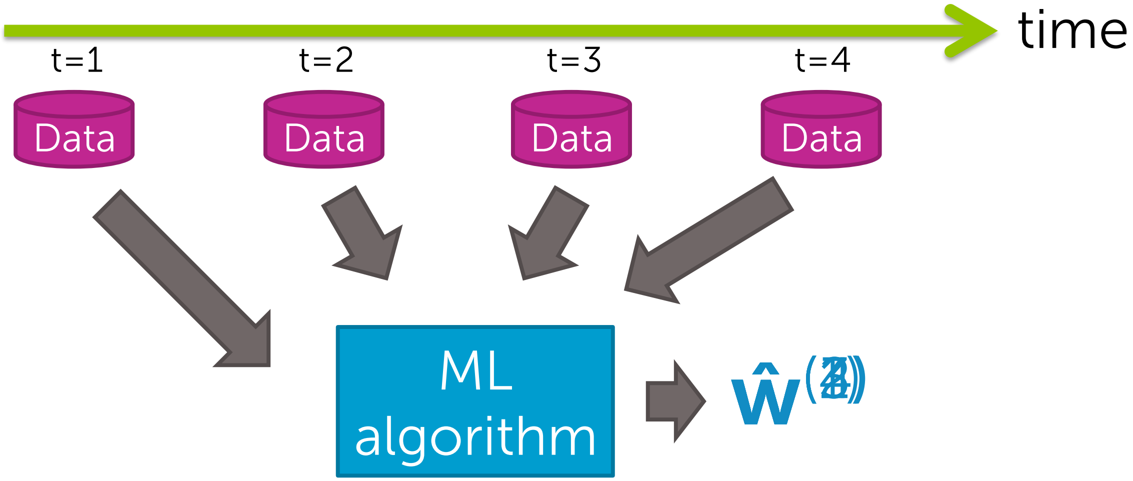
Batch learning

- All data is available at start of training time

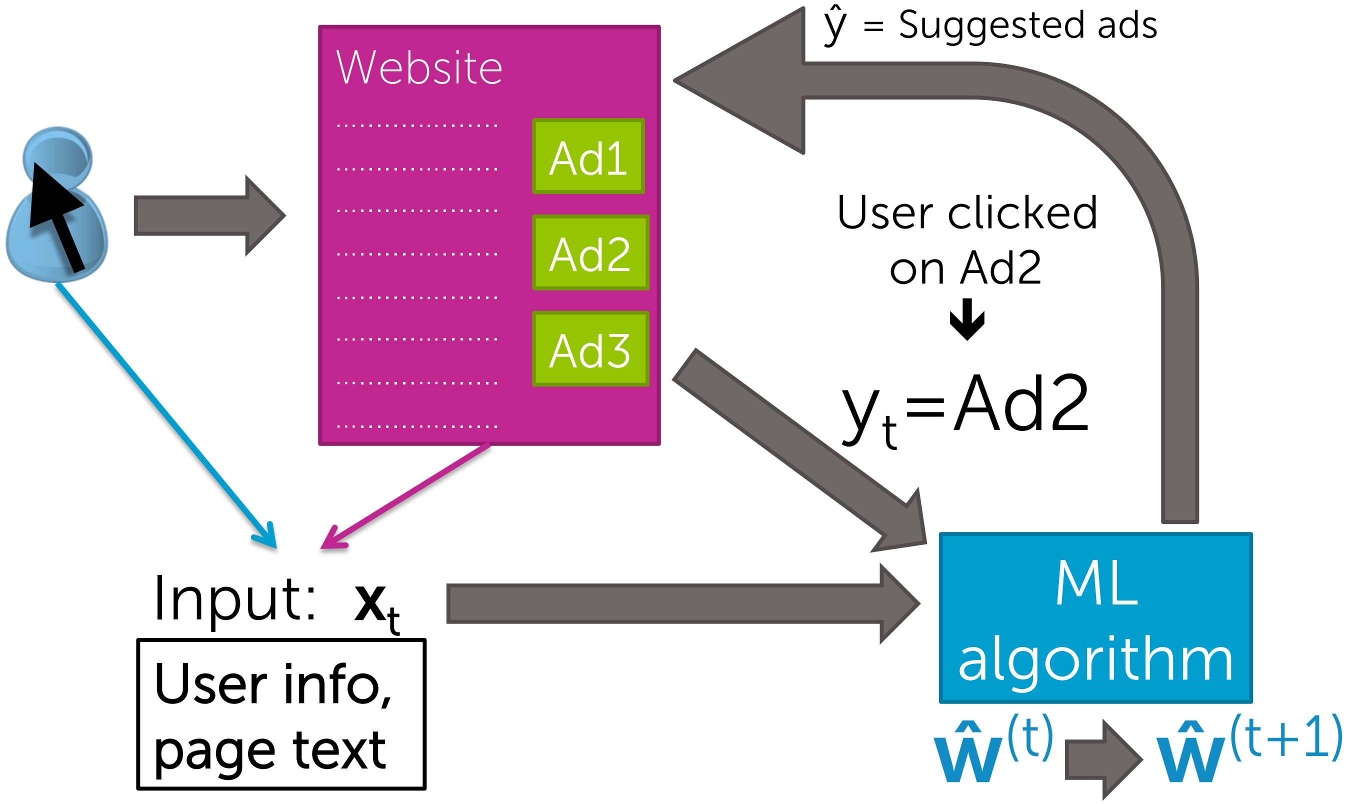


Online learning

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


time
t=1
t=2
t=3
t=4
Data
Data
Data
Data
ML algorithm
$\hat{w}^{(1)}$ $\hat{w}^{(2)}$ $\hat{w}^{(3)}$ $\hat{w}^{(4)}$

Online learning example:
Ad targeting



$\hat{y}$ = Suggested ads

Website

Ad1

Ad2

Ad3

User clicked on Ad2

y_t=Ad2

Input: $\mathbf{x}_t$

User info, page text

ML algorithm

$\hat{\mathbf{w}}^{(t)} \rightarrow \hat{\mathbf{w}}^{(t+1)}$

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

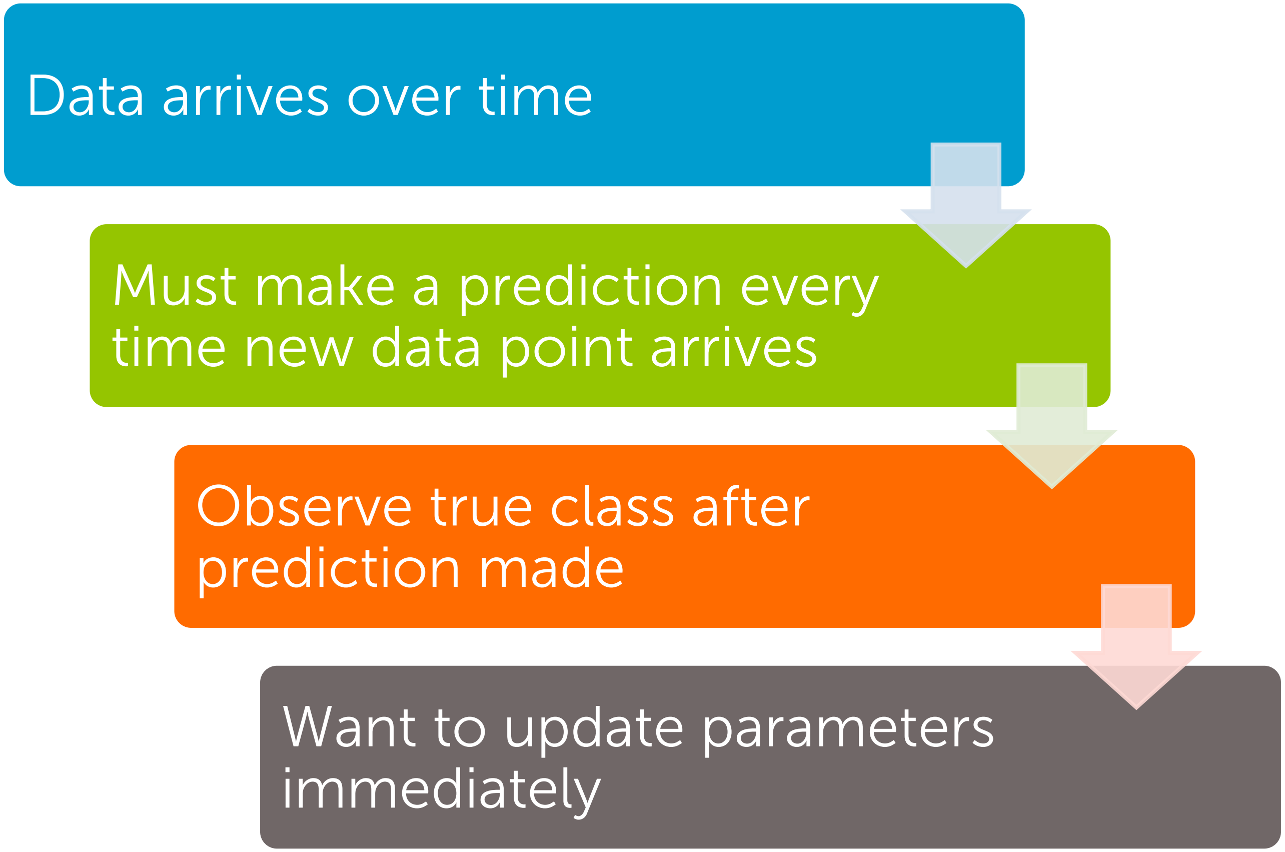
Need ML algorithm to
update coefficients each time step!

Stochastic gradient ascent can be used for online learning!!!

- init $\mathbf{w}^{(1)}=0$, 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,...,D
    $w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \frac{\partial \ell_t(\mathbf{w})}{\partial \mathbf{w}_j}$
```

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 date ➔ Often more accurate
- Lower computational cost
- Don't need to store all data, but often do anyway

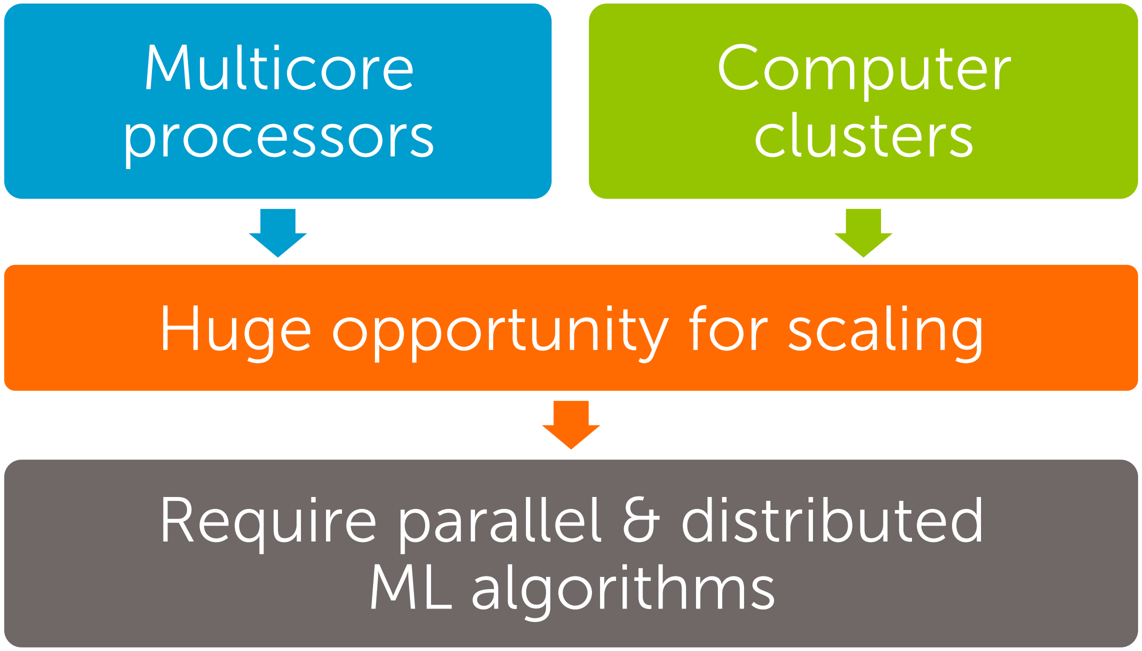
Cons

- 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,…

Summary of scaling to huge datasets & online learning

Scaling through parallelism



Multicore processors

Computer clusters

Huge opportunity for scaling

Require parallel & distributed ML algorithms

What you can do now…

- Significantly speedup learning algorithm using stochastic gradient
- Describe intuition behind why stochastic gradient works
- Apply stochastic gradient in practice
- Describe online learning problems
- Relate stochastic gradient to online learning