Scaling up learning with SGD

CSE 446: Machine Learning

Slides by Emily Fox

Presented by Anna Karlin

April 29, 2019

©2017 Emily Fox

Gradient Descent: workhorse of ML

- Typically choosing parameters **w** to minimize

$$\ell(\mathbf{w}) = \sum_{(\mathbf{x}_i, y_i) \in \text{Train}} \ell((\mathbf{x}_i, y_i), w)$$

e.g.

$$\ell(\mathbf{x}_i, y_i) = (y_i - \mathbf{x}_i^T \mathbf{w})^2$$

$$\ell(\mathbf{x}_i, y_i) = \log(1 + \exp(-y_i \mathbf{x}_i^T \mathbf{w}))$$

$$\ell(\mathbf{w}) = \sum_{(\mathbf{x}_i, y_i) \in \text{Train}} \ell((\mathbf{x}_i, y_i), w) + \lambda ||\mathbf{w}||_2^2$$

Gradient descent can be slow though….

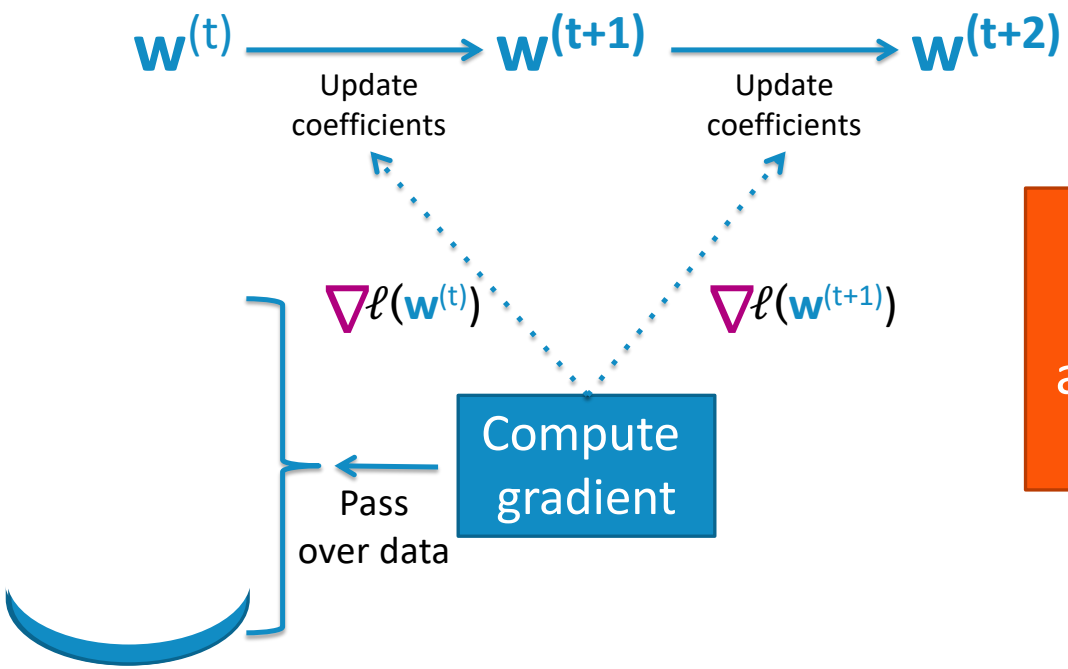
Sum over data points

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

Contribution of data point $\mathbf{x}_i, y_i$ to gradient

©2017 Emily Fox

Why gradient descent is slow…



Every update requires a full pass over data

©2017 Emily Fox

Every step requires touching every data point!!!

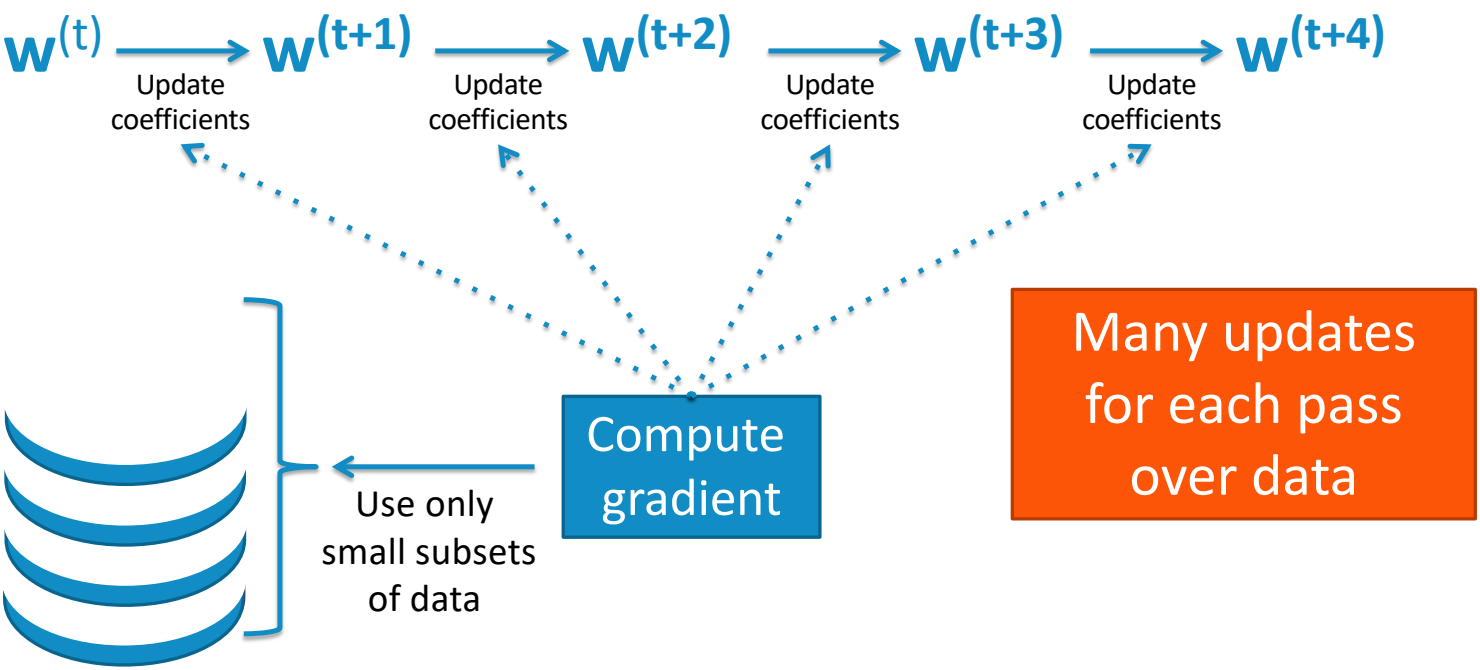
Sum over data points

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

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

©2017 Emily Fox

Stochastic gradient descent



©2017 Emily Fox

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

Sum over data points

Gradient descent

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

Each time, pick different data point i

Stochastic gradient descent

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

Randomly selected

©2017 Emily Fox

Stochastic gradient descent

random order

Each time, pick different data point i

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

until converged

for i=1, …, N

Sum over data points

for j=0,…,d

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

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

t ← t + 1

©2017 Emily Fox

Stochastic gradient for L2-regularized objective

Total derivative $= \sum_{i=1}^{N} \frac{\partial \ell_i(\mathbf{w})}{\partial \mathbf{w}_j} + 2\lambda w_j$ — partial[j]

What about regularization term?

Each time, pick different data point i

Stochastic gradient descent

Total derivative $\approx \frac{\partial \ell_i(\mathbf{w})}{\partial \mathbf{w}_j} + \frac{2}{N}\lambda w_j$ — partial[j]

Each data point contributes 1/N to regularization

©2017 Emily Fox

Comparing computational time per step

Gradient descent

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

Stochastic gradient descent

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

Time to compute contribution of $\mathbf{x}_i, \mathbf{y}_i$	# 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

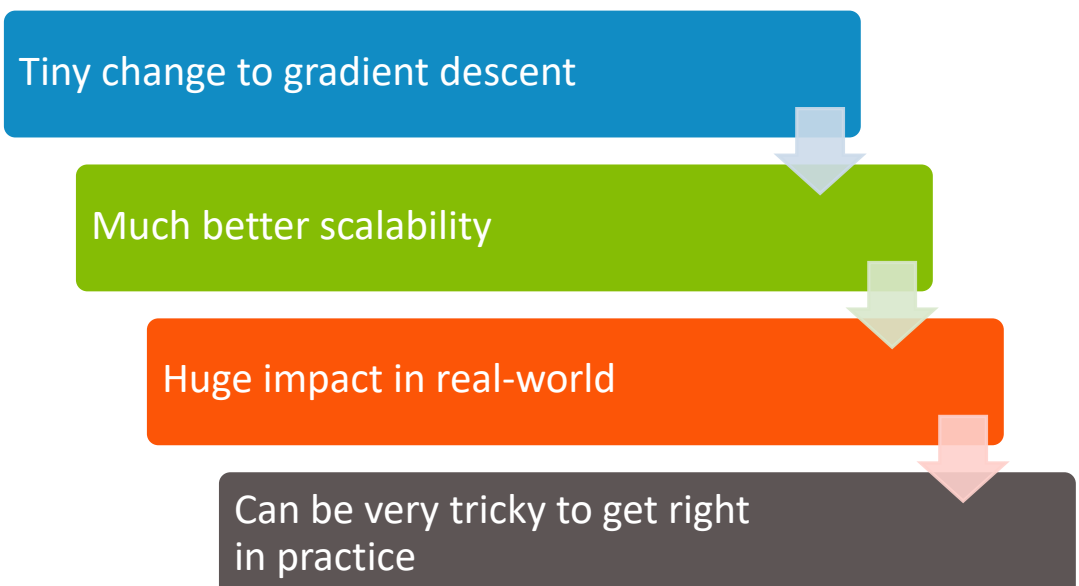
©2017 Emily Fox

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	Usually slower	Moderate
Stochastic gradient	Always fast	Faster	Usually faster	Very high

©2017 Emily Fox

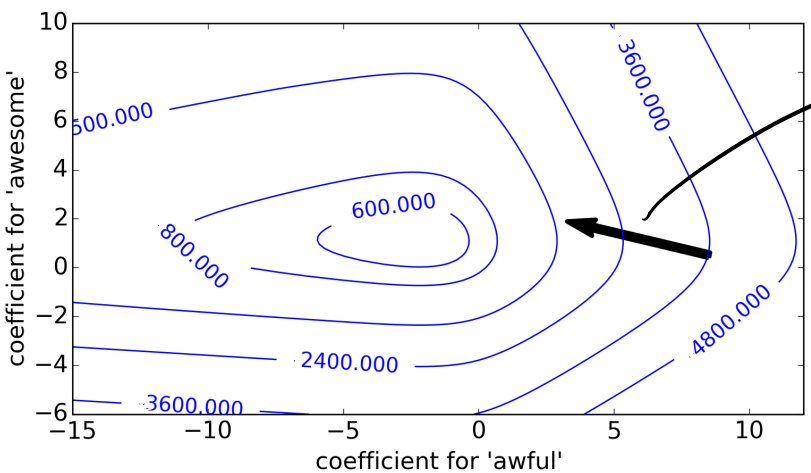
Summary of stochastic gradient



©2017 Emily Fox

Why would stochastic gradient ever work???

Negative gradient is direction of steepest descent

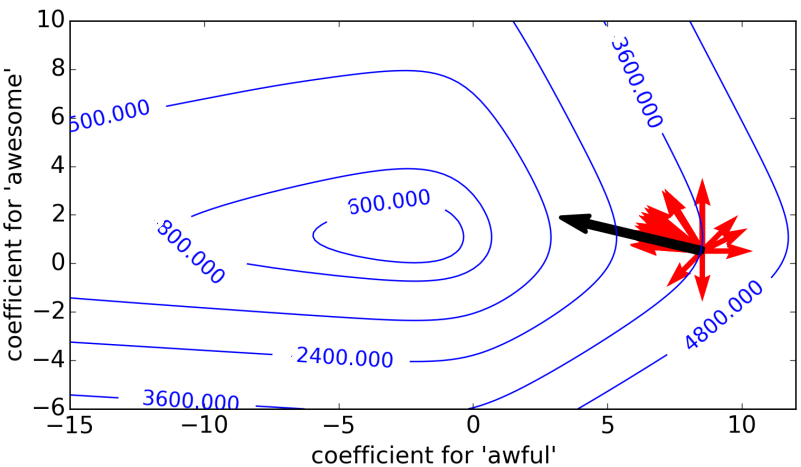


negative gradient

Negative gradient is "best" direction, but any direction that goes "down" would be useful

©2017 Emily Fox

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



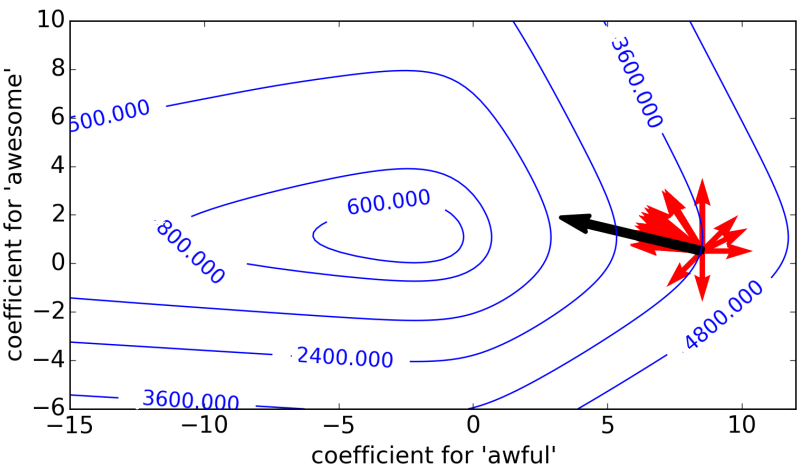
Sum over data points

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

For most data points, contribution points "down"

©2017 Emily Fox

Stochastic gradient:
Pick a data point and move in direction

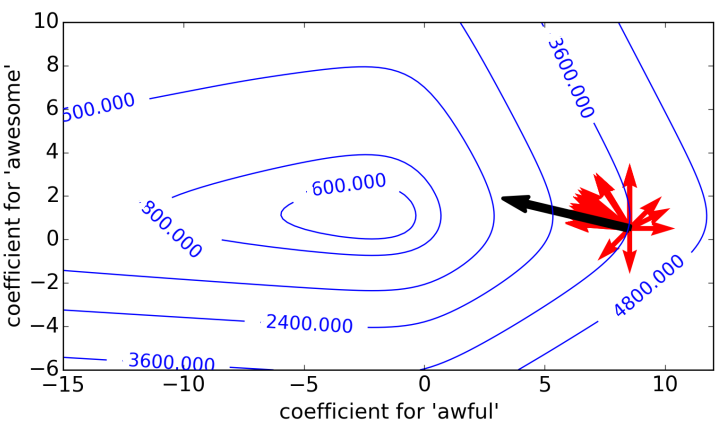


$$\frac{\partial \ell(\mathbf{w})}{\partial \mathbf{w}_j} \approx \frac{\partial \ell_i(\mathbf{w})}{\partial \mathbf{w}_j}$$

Most of the time, total loss will decrease (likelihood will increase)

©2017 Emily Fox

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



until converged

for i=1,…,N *(in random order)*

for j=0,…,d

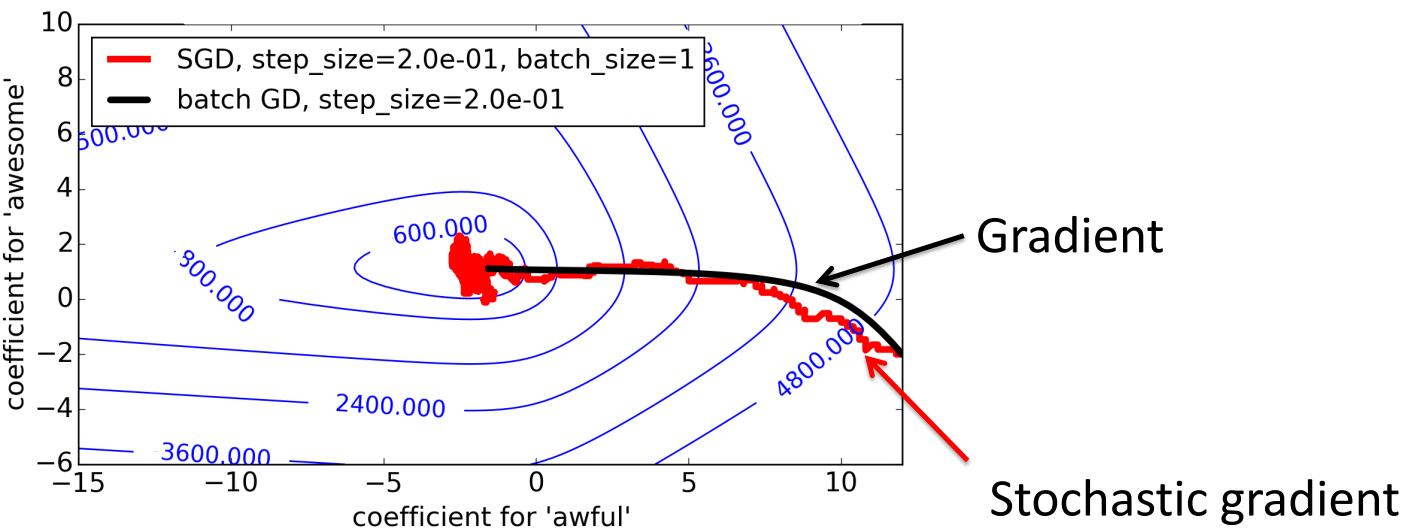
$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta \frac{\partial \ell_i(\mathbf{w})}{\partial w_j}$$

t ← t + 1

©2017 Emily Fox

Convergence path

Convergence paths

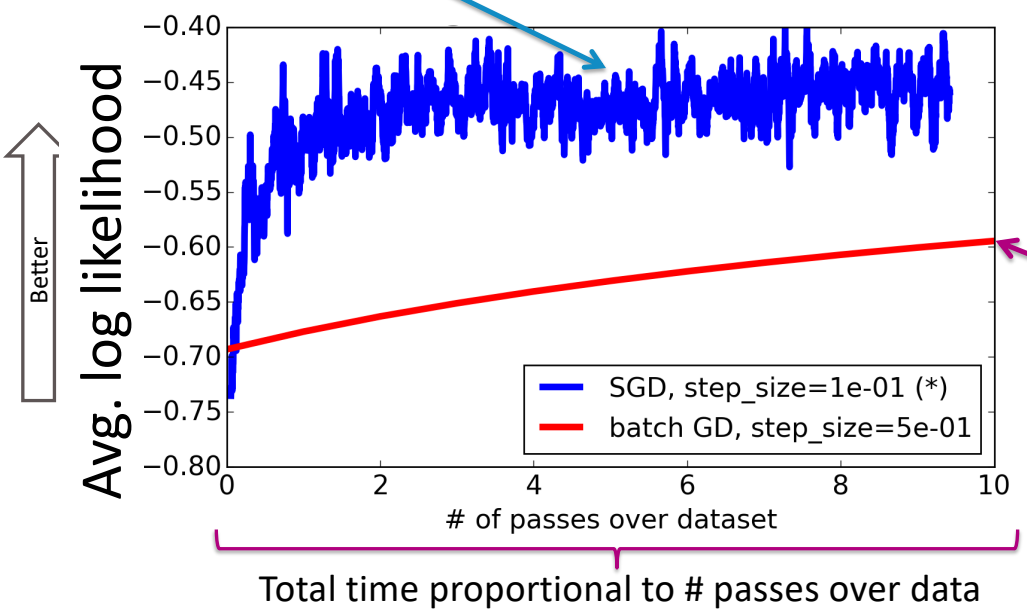


©2017 Emily Fox

Stochastic gradient convergence is "noisy"

Stochastic gradient makes "noisy" progress

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

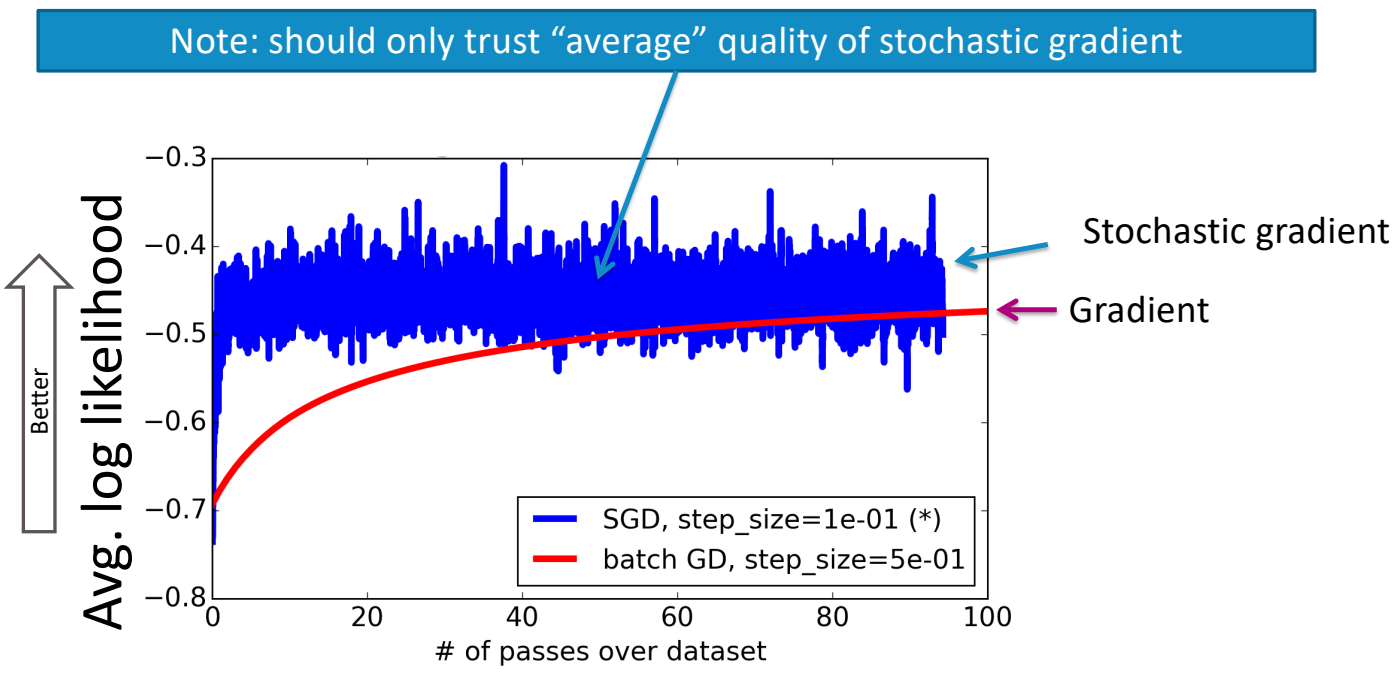


Gradient usually increases likelihood smoothly

Total time proportional to # passes over data

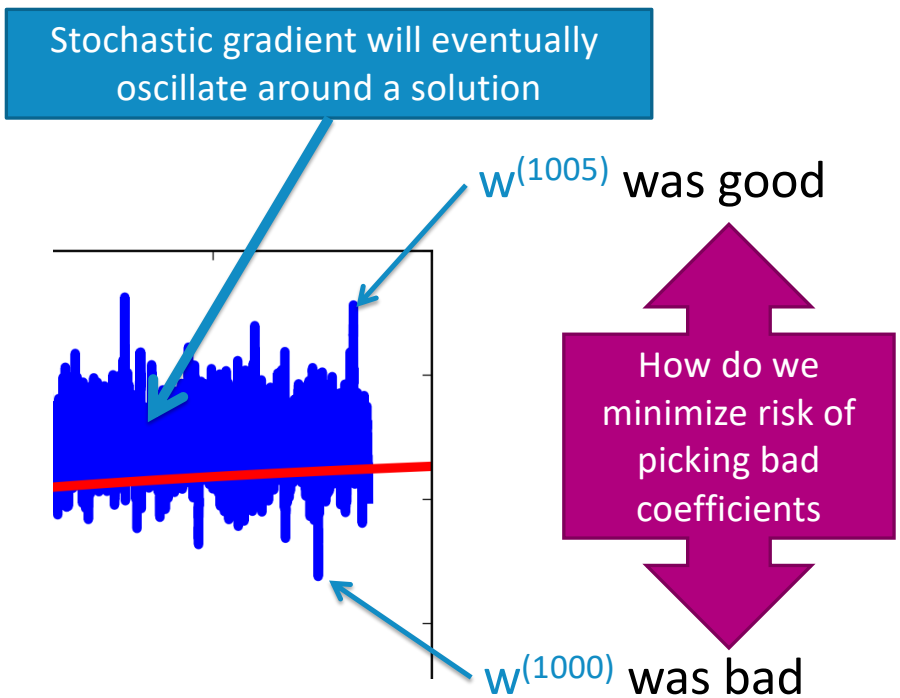
©2017 Emily Fox

Eventually, gradient catches up



©2017 Emily Fox

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



Minimize noise: don't return last learned coefficients

Output average:

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

©2017 Emily Fox

Stochastic gradient descent more formally

Learning Problems as Expectations

- Minimizing loss in training data:
  - Given dataset:
    - Sampled iid from some distribution p(**x**, y) on features/labels:
  - Loss function, e.g., squared loss, logistic loss,…
  - We often minimize loss in training data:

$$\ell(\mathbf{w}) = \frac{1}{N_{\text{Train}}} \sum_{(\mathbf{x}_i, y_i) \in \text{Train}} \ell((\mathbf{x}_i, y_i), \mathbf{w})$$

- However, we should really minimize expected loss on all data:

$$\ell(\mathbf{w}) = \int \ell((\mathbf{x}, y), \mathbf{w})\, p(\mathbf{x}, y)\, d\mathbf{x}dy = E_{(\mathbf{x}, y)}[\ell((\mathbf{x}, y)\mathbf{w})]$$

- So, we are approximating the integral by the average on the training data

©2017 Emily Fox

Gradient Descent in Terms of Expectations

- "True" objective function:

$$\ell(\mathbf{w}) = \int \ell((\mathbf{x}, y), \mathbf{w})\, p(\mathbf{x}, y)\, d\mathbf{x}dy = E_{(\mathbf{x}, y)}[\ell((\mathbf{x}, y)\mathbf{w})]$$

- Taking the gradient:

$$\nabla\ell(\mathbf{w}) = \int \nabla\ell((\mathbf{x}, y), \mathbf{w})\, p(\mathbf{x}, y)\, d\mathbf{x}dy = E_{(\mathbf{x}, y)}[\nabla_{\mathbf{w}}\ell((\mathbf{x}, y)\mathbf{w})]$$

- "True" gradient descent rule:

$$\mathbf{w}_{t+1} := \mathbf{w}_t - \eta E_{\mathbf{x}}[\nabla_{\mathbf{w}}\ell((\mathbf{x}, y), \mathbf{w}_t)]$$

How do we estimate expected gradient?

$$\mathbf{w}_{t+1} := \mathbf{w}_t - \eta\nabla_{\mathbf{w}}\ell((\mathbf{x}_t, y_t)\mathbf{w}_t) \quad \text{where } (\mathbf{x}_t, y_t) \sim_{\text{i.i.d.}} p(\mathbf{x}, y)$$

SGD: Stochastic Gradient Descent

- "True" gradient:

$$\nabla \ell(\mathbf{w}) = \int \nabla \ell((\mathbf{x}, y), \mathbf{w})\, p(\mathbf{x}, y)\, d\mathbf{x}dy = E_{(\mathbf{x}, y)}[\nabla_{\mathbf{w}} \ell((\mathbf{x}, y)\mathbf{w})]$$

- What if we estimate gradient with just one sample???
  - Unbiased estimate of gradient
  - Very noisy!

$$\nabla_{\mathbf{w}} \ell((\mathbf{x}_t, y_t), \mathbf{w}_t) \quad \text{where } \mathbf{x}_t \sim_{\text{i.i.d.}} p(\mathbf{x}, y)$$

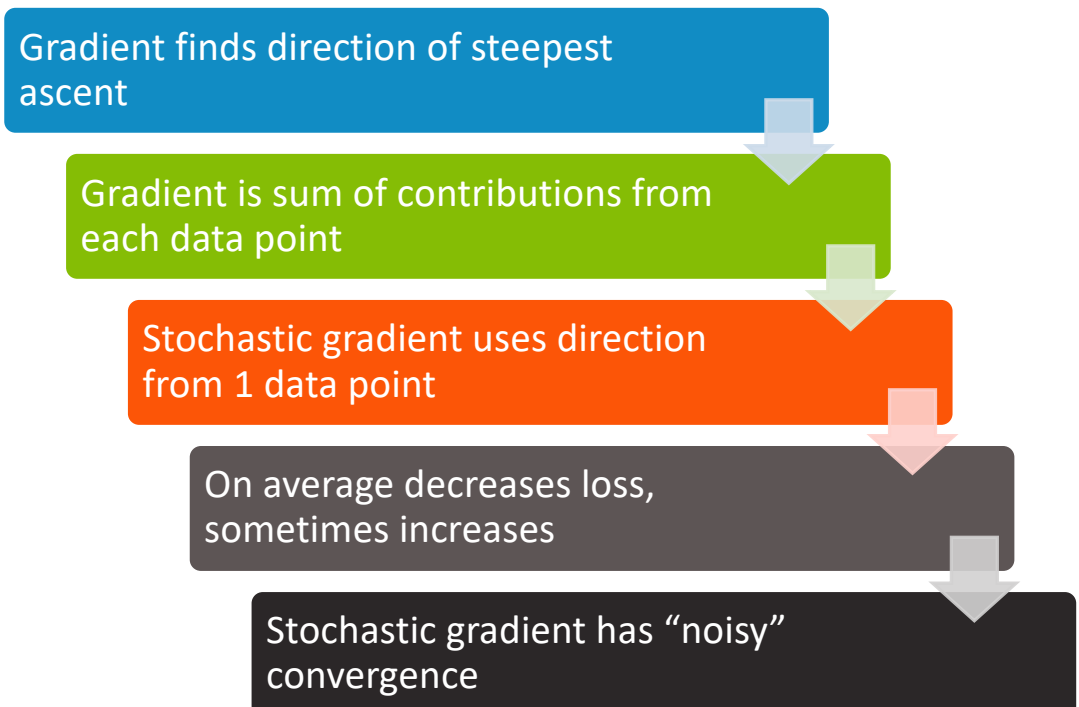
  - This is stochastic gradient descent (SGD)
  - VERY useful in practice!!!

©2017 Emily Fox

Amazing theorem

- With appropriately selected step-size, under very conditions, SGD guaranteed to converge!

Summary of why stochastic gradient works



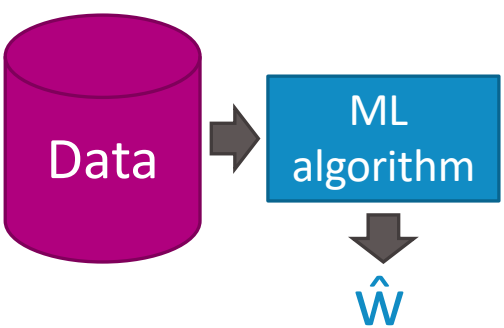
©2017 Emily Fox

Online learning:
Fitting models from streaming data

Batch vs online learning

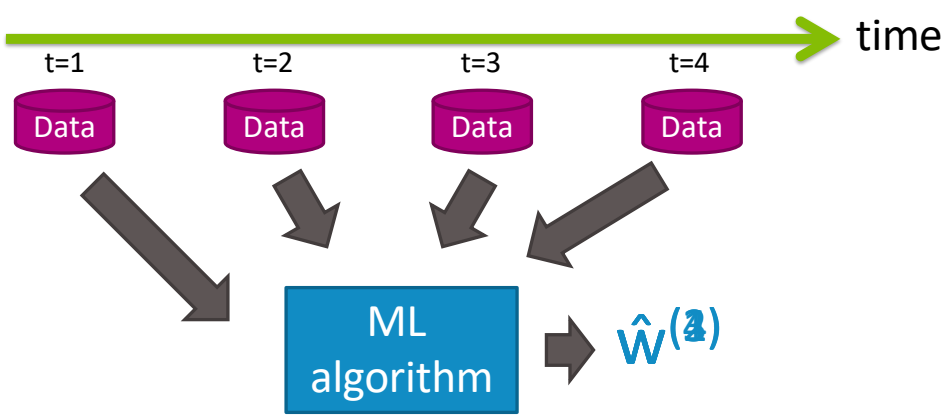
Batch learning

- All data is available at start of training time

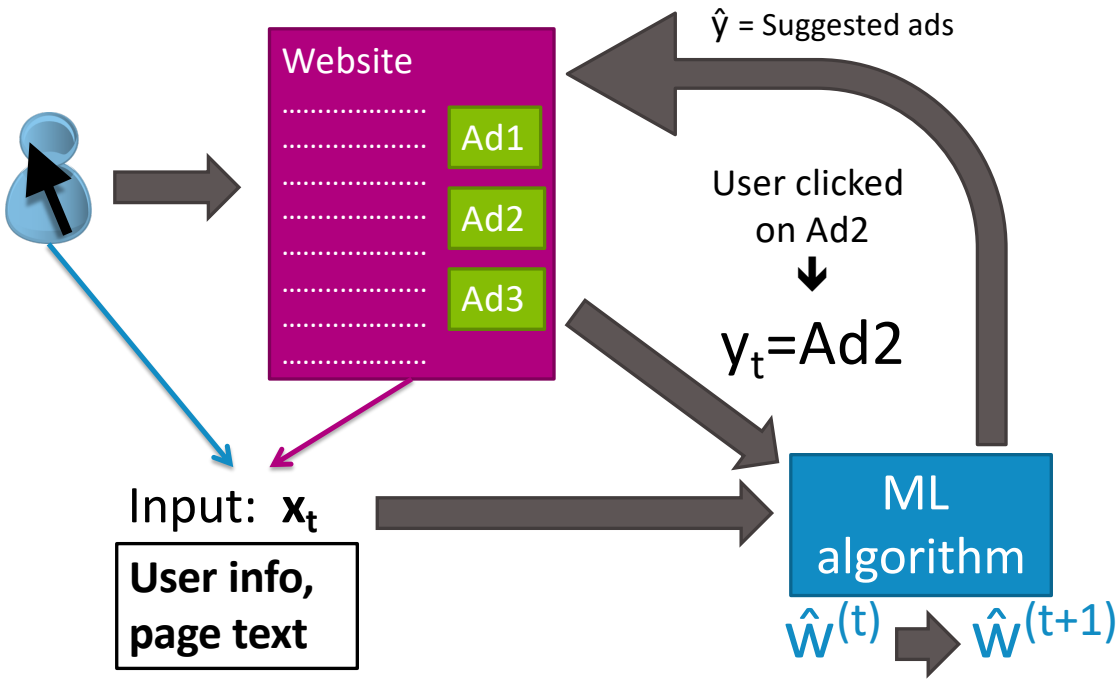


Online learning

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



Online learning example: Ad targeting



©2017 Emily Fox

Online learning problem

- Data arrives over each time step t:
  - Observe input $\mathbf{x}_t$
    - 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!

©2017 Emily Fox

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

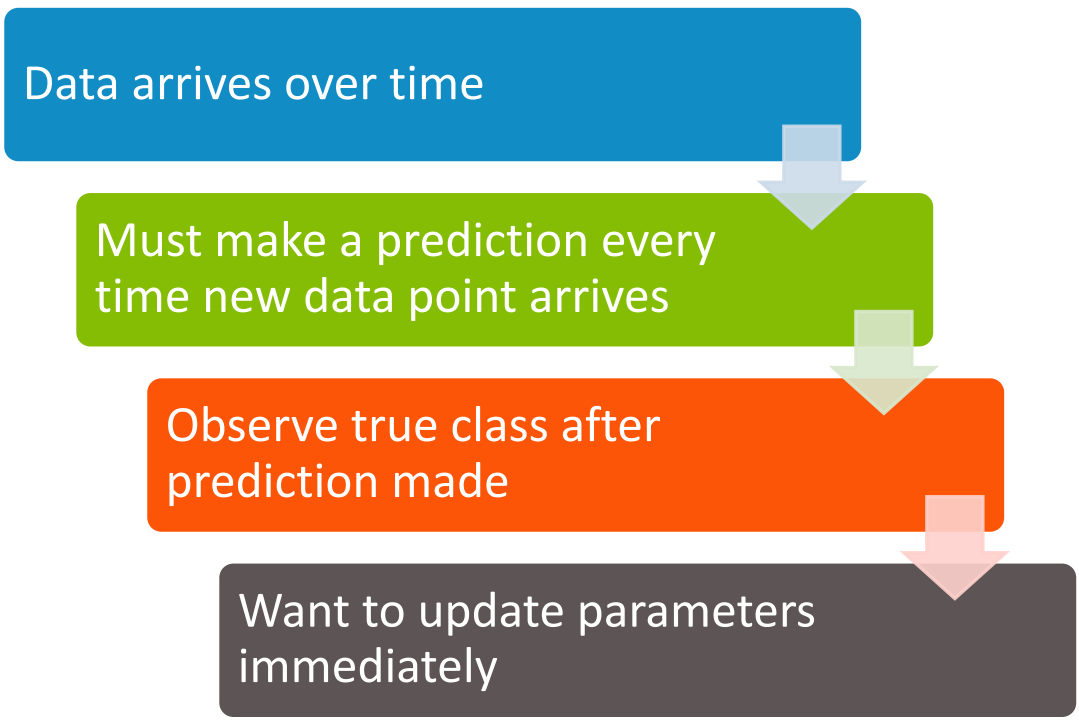
- init $\mathbf{w}^{(1)}$=0, t=1
- Each time step t:
  - Observe input $\mathbf{x}_t$
  - Make a prediction $\hat{y}_t$
  - Observe true output y_t
  - Update coefficients:

for j=0,…,D

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

©2017 Emily Fox

Summary of online learning



©2017 Emily Fox

Summary of stochastic gradient descent

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

©2017 Emily Fox

The Perceptron Algorithm

CSE 446: Machine Learning
Slides by Emily Fox (mostly)

Presented by Anna Karlin

April 29, 2019

©2017 Emily Fox

Can we do binary classification without a model?

The perceptron algorithm

For simplicity in this lecture we will assume that b=0

The perceptron algorithm [Rosenblatt '58, '62]

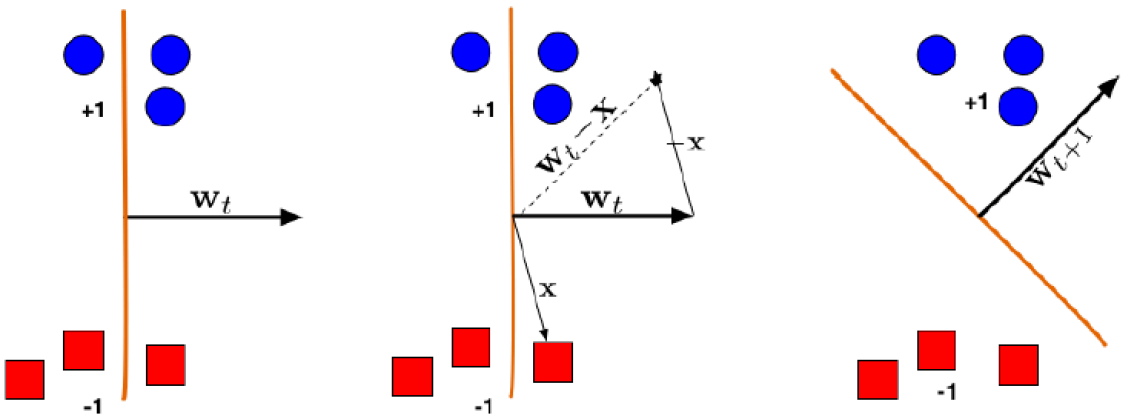
- Classification setting: y in {-1,+1}
- Linear model
  - Prediction:

- Training:
  - Initialize weight vector:
  - At each time step:
    - Observe features:
    - Make prediction:
    - Observe true class:

    - Update model:
      - If prediction is not equal to truth

©2017 Emily Fox

Picture due to Killian Weinberger



- Initialize $\mathbf{w}_0$.
- for $t := 1$ to T
  - Observe feature vector $\mathbf{x}_t$.
  - Make a prediction: $\hat{y} = \text{sign}(\mathbf{x}_t^T \mathbf{w}_t)$
  - Observe the true label $y_t \in \{-1, 1\}$.
  - Update the model:
    - If $\hat{y} = y_t$, then $\mathbf{w}_{t+1} := \mathbf{w}_t$.
    - Otherwise ($\hat{y} \neq y_t$), then $\mathbf{w}_{t+1} := \mathbf{w}_t + y_t \mathbf{x}_t$.

How many mistakes can a perceptron make?

Assumption: data is linearly separable

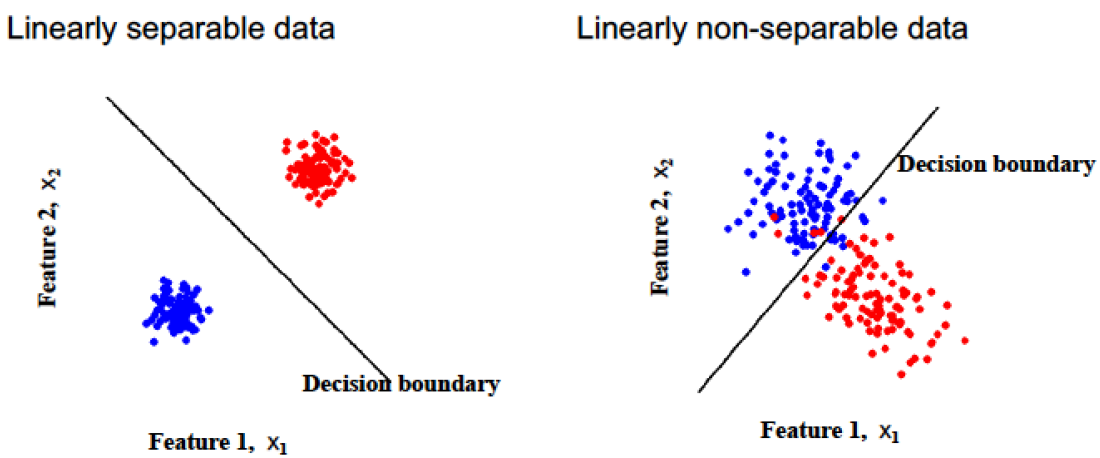


Figure by Alexander Ihler

Linear separability: More formally, using margin

Data linearly separable, if there exists

- a vector
- a margin

such that

©2017 Emily Fox

Perceptron analysis: Linearly separable case

Theorem [Block, Novikoff]:

- Given a sequence of labeled examples:
- Each feature vector has bounded norm:
- If dataset is linearly separable:

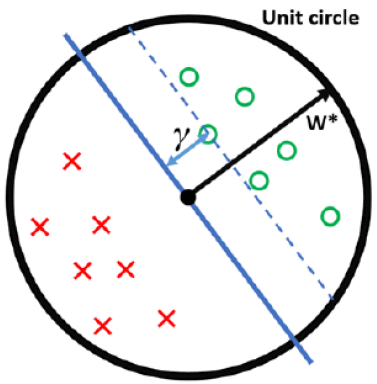


Figure by Kilian Weinberger

Then the # mistakes made by the online perceptron on this sequence is bounded by

©2017 Emily Fox

Perceptron proof for linearly separable case

- Every time we make a mistake, we get w closer to w^*:
  - Mistake at time t:
  - Taking dot product with w^*:
  - Thus after m mistakes:

$$\mathbf{w}_{t+1} := \mathbf{w}_t + y_t \mathbf{x}_t$$

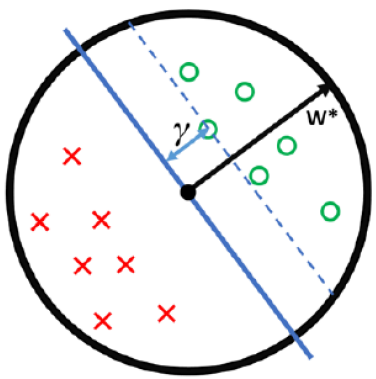


Figure by Kilian Weinberger

- On the other hand, norm of $w^{(t+1)}$ doesn't grow too fast:

$$||\mathbf{w}_{t+1}||^2 = ||\mathbf{w}_t||^2 + 2y_t(\mathbf{w}_t \cdot \mathbf{x}_t) + ||\mathbf{x}_t||^2$$

  - Thus, after m mistakes:

- Putting all together:

©2017 Emily Fox

Beyond linearly separable case

- Perceptron algorithm is super cool!
  - No assumption about data distribution!
    - Could be generated by an adversary, no need to be iid
  - Makes a fixed number of mistakes, and it's done forever!
    - Even if you see infinite data

©2017 Emily Fox

Rosenblatt 1957

"the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

The New York Times, 1958

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, de-

©2017 Emily Fox

Beyond linearly separable case

- Perceptron algorithm is super cool!
  - No assumption about data distribution!
    - Could be generated by an adversary, no need to be iid
  - Makes a fixed number of mistakes, and it's done for ever!
    - Even if you see infinite data

- Possible to motivate the perceptron algorithm as an implementation of SGD for minimizing a certain loss function. (see next homework)

©2017 Emily Fox

Beyond linearly separable case

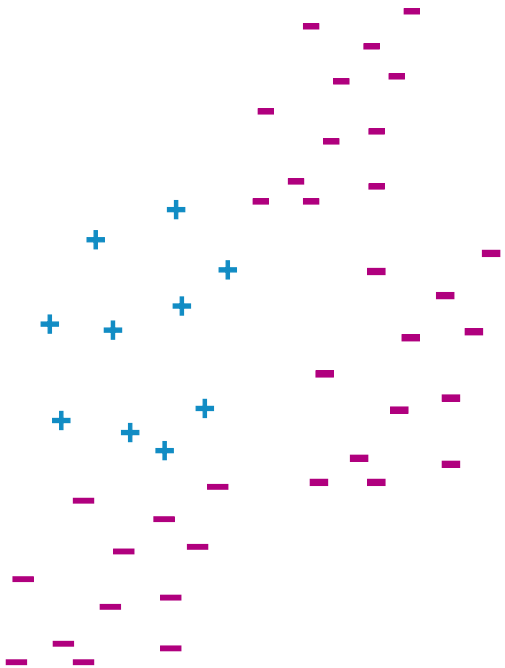
- Perceptron algorithm is super cool!
  - No assumption about data distribution!
    - Could be generated by an adversary, no need to be iid
  - Makes a fixed number of mistakes, and it's done for ever!
    - Even if you see infinite data

- However, real world is not linearly separable
  - Can't expect never to make mistakes again
  - If data not separable, cycles forever and hard to detect.
  - Even if separable, may not give good generalization accuracy (make have small margin).

©2017 Emily Fox

The kernelized perceptron

What to do if data are not linearly separable?



Use feature maps…

$$\phi(x) : \mathbb{R}^d \rightarrow F$$

- Start with set of inputs for each observation **x** = (x[1],x[2],…, x[d])

and training set: $\{(\mathbf{x}_i, y_i)\}_{i=1..n}$

- Define feature map that transforms each input vector $\mathbf{x}_i$ to higher dimensional feature vector $\phi(\mathbf{x}_i)$

Example: $(x_i[1], x_i[2], x_i[3])^T$

$\phi(\mathbf{x}_i) = (1, x_i[1], x_i[1]^2, x_i[1]x_i[2], x_i[2], x_i[2]^2, \cos(\pi\, x_i[3]/6))^T$

©2017 Emily Fox

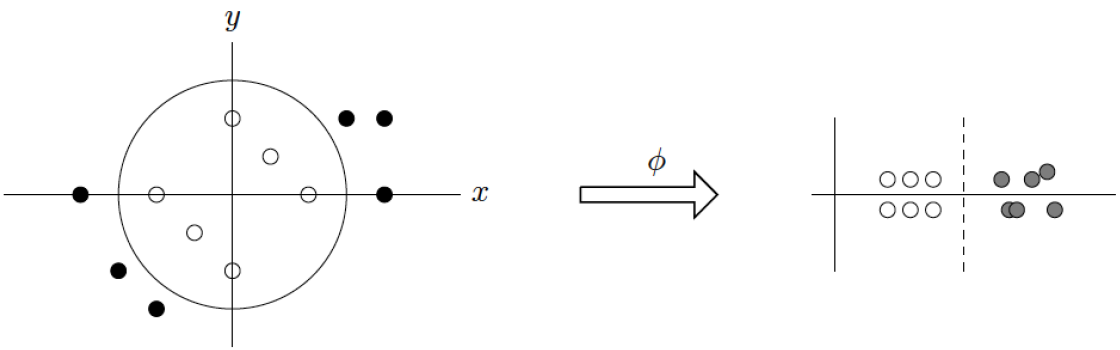
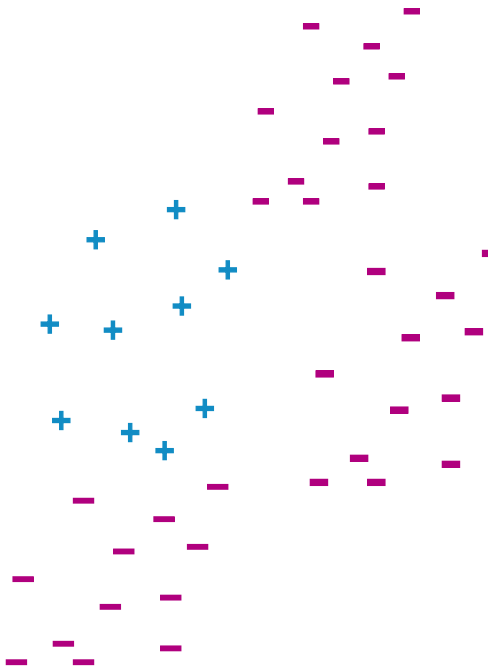


Figure 5.2: Data that is not linearly separable in the input space $\mathbb{R}^2$ but that is linearly separable in the "ϕ-space," $\phi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2)$.

For instance, the hyperplane $\phi(\mathbf{x})^T\mathbf{w}^* = 0$ for $\mathbf{w}^* = (-4, 0, 0, 1, 0, 1)$ circle $x_1^2 + x_2^2 = 4$ in the original space, such as in Figure 5.2.

From "Foundations of Data Science" by Blum, Kannan, Hopcroft

What to do if data are not linearly separable?



Use feature maps…

$$\phi(x) : \mathbb{R}^d \rightarrow F$$

$$\Phi(\mathbf{x}) = \begin{pmatrix} 1 \\ x[1] \\ x[2] \\ \dots \\ x[1]^2 \\ x[2]^2 \\ \dots \\ x[1]x[2] \\ x[1]x[3] \\ \dots \end{pmatrix} \quad \text{where} \quad \mathbf{x} = (x[1], \dots, x[d])$$

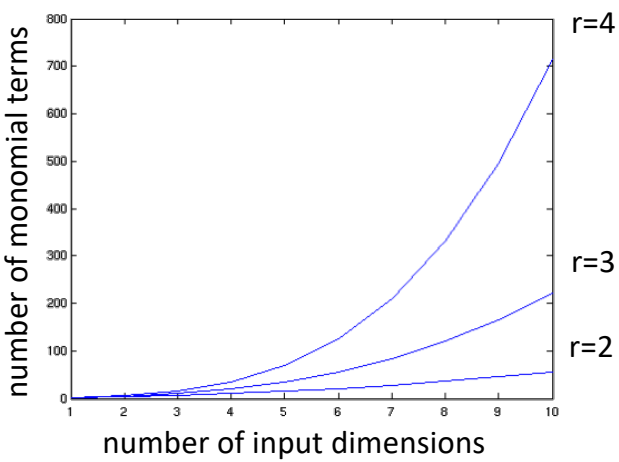
$$\Phi(\mathbf{x}) = \begin{pmatrix} 1 \\ x[1] \\ \dots \\ \dots \\ x[1]e^{\sin(x[3])} \\ e^{-x[2]^2/\sigma^2} \\ \dots \end{pmatrix}$$

Could even be infinite dimensional….

©2017 Emily Fox

Example: Higher order polynomials

Feature space can get really large really quickly!



$$\phi(\mathbf{u}) = \begin{pmatrix} 1 \\ u_1 \\ u_2 \\ \dots \\ u_1^2 \\ u_1 u_2 \\ \dots \\ u_1^5 u_3 u_{17}^4 \\ \dots \end{pmatrix}$$

d – input dimension
r – degree of polynomial

$$\binom{r+d-1}{r} \text{ terms}$$

grows fast!
r = 6, d = 100
about 1.6 billion terms

©2017 Emily Fox

Kernel trick

- Allows us to do these apparently intractable calculations efficiently!

Perceptron revisited

Initialize $\mathbf{w}_0$.
for $t := 1$ to T
- Observe feature vector $\mathbf{x}_t$.
- Make a prediction: $\hat{y} = \text{sign}(\mathbf{x}_t^T \mathbf{w}_t)$
- Observe the true label $y_t \in \{-1, 1\}$.
- Update the model:
  - If $\hat{y} = y_t$, then $\mathbf{w}_{t+1} := \mathbf{w}_t$.
  - Otherwise ($\hat{y} \neq y_t$), then $\mathbf{w}_{t+1} := \mathbf{w}_t + y_t \mathbf{x}_t$.

Write weight vector in terms of mistaken data points only.

Let $M^{(t)}$ be time steps up to t when mistakes were made:

Prediction rule becomes:

When using high dimensional features:

Classification only depends on inner products!

Why does dependence on inner products help?

- Because sometimes they can be computed **much much much more efficiently**.

$$\phi(\mathbf{u}) = \begin{pmatrix} 1 \\ u_1 \\ u_2 \\ \dots \\ u_1^2 \\ u_1 u_2 \\ \dots \\ u_1^5 u_3 u_{17}^4 \\ \dots \end{pmatrix} \qquad \phi(u) \cdot \phi(v) = \begin{pmatrix} 1 \\ u_1 \\ u_2 \\ \dots \\ u_1^2 \\ u_1 u_2 \\ \dots \\ u_1^5 u_3 u_{17}^4 \\ \dots \end{pmatrix} \cdot \begin{pmatrix} 1 \\ v_1 \\ v_2 \\ \dots \\ v_1^2 \\ v_1 v_2 \\ \dots \\ v_1^5 v_3 v_{17}^4 \\ \dots \end{pmatrix}$$

$$\binom{r+d-1}{r}$$

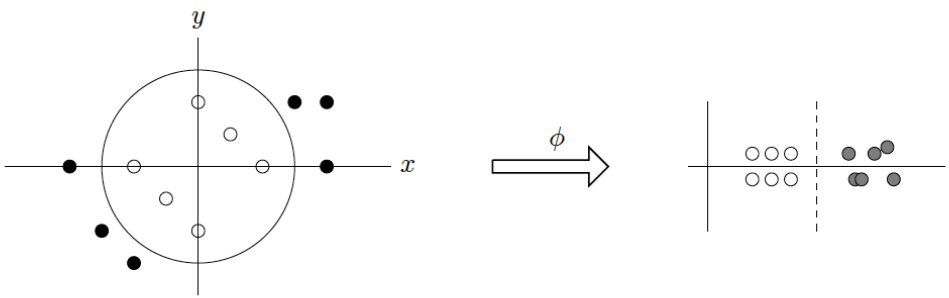


Figure 5.2: Data that is not linearly separable in the input space $\mathbb{R}^2$ but that is linearly separable in the "ϕ-space," $\phi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2)$, corresponding to the kernel function $K(\mathbf{x}, \mathbf{x}') = (1 + x_1x_1' + x_2x_2')^2$.

For instance, the hyperplane $\phi(\mathbf{x})^T\mathbf{w}^* = 0$ for $\mathbf{w}^* = (-4, 0, 0, 1, 0, 1)$ circle $x_1^2 + x_2^2 = 4$ in the original space, such as in Figure 5.2.

From "Foundations of Data Science" by Blum, Kannan, Hopcroft

Dot-product of polynomials: why useful?

- Because sometimes they can be computed much much much more efficiently.

$$\phi(\mathbf{u}) = \begin{pmatrix} 1 \\ u_1 \\ u_2 \\ \dots \\ u_1^2 \\ u_1 u_2 \\ \dots \\ u_1^5 u_3 u_{17}^4 \\ \dots \end{pmatrix} \qquad \phi(u) \cdot \phi(v) = \begin{pmatrix} 1 \\ u_1 \\ u_2 \\ \dots \\ u_1^2 \\ u_1 u_2 \\ \dots \\ u_1^5 u_3 u_{17}^4 \\ \dots \end{pmatrix} \cdot \begin{pmatrix} 1 \\ v_1 \\ v_2 \\ \dots \\ v_1^2 \\ v_1 v_2 \\ \dots \\ v_1^5 v_3 v_{17}^4 \\ \dots \end{pmatrix}$$

So with appropriate constants in front

$$\phi(\mathbf{u}) \cdot \phi(\mathbf{v}) = (1 + \mathbf{u} \cdot \mathbf{v})^r$$

$$(1 + \mathbf{u} \cdot \mathbf{v})^r = [1 + u_1 v_1 + u_2 v_2 + \dots u_d v_d]^r$$

Typical term: $C \cdot (u_1 v_1)^{i_1} (u_2 v_2)^{i_2} \cdots (u_d v_d)^{i_d}$ where $\sum_{j=1}^{d} i_j \le r$

Kernel trick

- Allows us to do these apparently intractable calculations efficiently!
- If we can compute inner products efficiently, can run the algorithm (e.g., Perceptron) efficiently.
- Hugely important idea in ML.

Finally, the kernel trick
Kernelized perceptron

- Every time you make a mistake, remember ($\mathbf{x}_t$,y_t)

- Kernelized perceptron prediction for x:

$$\text{sign}(\mathbf{w}_t \cdot \phi(\mathbf{x})) = \sum_{i \in M^{(t)}} y_i (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}))$$
$$= \sum_{i \in M^{(t)}} y_i K(\mathbf{x}_i, \mathbf{x}).$$

$$K(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u}) \cdot \phi(\mathbf{v}) = (1 + \mathbf{u} \cdot \mathbf{v})^r$$

©2017 Emily Fox

Common kernels

- Polynomials of degree exactly p

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^p$$

- Polynomials of degree up to p

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^p$$

- **Gaussian (squared exponential) kernel**

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{||\mathbf{u} - \mathbf{v}||^2}{2\sigma^2}\right)$$

- Many many many others

©2017 Emily Fox

Kernelizing a machine learning algorithm

- Prove that the solution is always in span of training points. I.e.,
- Rewrite the algorithm so that all training or test inputs are accessed only through inner products with other inputs.
- Choose (or define) a kernel function and substitute $K(\mathbf{x}_i, \mathbf{x}_j)$ for $\mathbf{x}_i^T \mathbf{x}_j$

So with appropriate constants in front in definition of $\phi(\cdot)$

$$K(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u}) \cdot \phi(\mathbf{v}) = (1 + \mathbf{u} \cdot \mathbf{v})^r$$

What you need to know

- Linear separability in higher-dim feature space
- The kernel trick
- Kernelized perceptron
- Polynomial and other common kernels (will see more later)

©2017 Emily Fox