CPSC 340:
Machine Learning and Data Mining

Stochastic Gradient

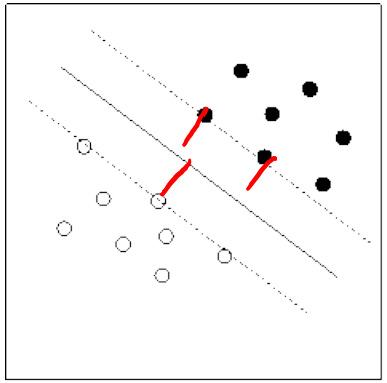
Fall 2016

Admin

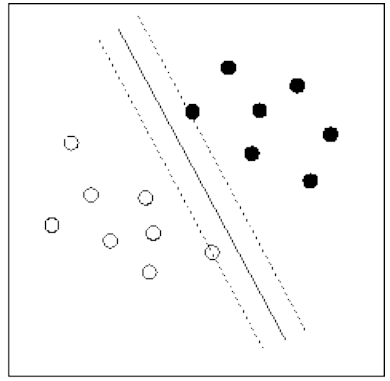
- Assignment 3:
  - 2 late days before class Monday, 3 late days before class Wednesday.
  - Solutions will be posted after class Wednesday.
- Midterm next Friday:
  - Midterm from last year and list of topics posted (covers Assignments 1-3).
    - Tutorials next week will cover practice midterm.
  - In class, 55 minutes, closed-book, cheat sheet: 2-pages each double-sided.

Last Time: SVMs and Kernel Trick

- We discussed the maximum margin view of SVMs:
  - Yields an L2-regularized hinge loss.



(a) Larger margin



(b) Smaller margin

- We introduced the kernel trick:
  - Write model to only depend on inner products between features vectors.

$$\hat{y} = \hat{K}(K + \lambda I)^{-1} y$$

$t \times n$ matrix $\hat{Z}Z^T$ containing inner products between test examples and training examples.

$n \times n$ matrix ZZ^T containing inner products between all training examples.

  - So everything we need to know about z_i is summarized by the $z_i^T z_j$.
  - If you have a kernel function $k(x_i, x_j)$ that computes $z_i^T z_j$, then you don't need to compute the basis z_i explicitly.

http://svr-www.eng.cam.ac.uk/~kkc21/thesis_main/node12.html

Polynomial Kernel with Higher Degrees

- Assume that I have 2 features and want to use the degree-2 basis:

$$z_i = \begin{bmatrix} 1 & \sqrt{2}x_{i1} & \sqrt{2}x_{i2} & x_{i1}^2 & \sqrt{2}x_{i1}x_{i2} & x_{i2}^2 \end{bmatrix}^T$$

- I can compute inner products using:

$$(1 + x_i^T x_j)^2 = 1 + 2x_i^T x_j + (x_i^T x_j)^2$$

$$= 1 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i2}^2 x_{j2}^2$$

$$= \underbrace{\begin{bmatrix} 1 & \sqrt{2}x_{i1} & \sqrt{2}x_{i2} & x_{i1}^2 & \sqrt{2}x_{i1}x_{i2} & x_{i2}^2 \end{bmatrix}}_{z_i^T} \begin{bmatrix} 1 \\ \sqrt{2}x_{j1} \\ \sqrt{2}x_{j2} \\ x_{j2}^2 \\ \sqrt{2}x_{j1}x_{j2} \\ x_{j2}^2 \end{bmatrix}$$

$$= z_i^T z_j$$

z_j

Polynomial Kernel with Higher Degrees

- To get all degree-4 "monomials" I can use:

$$z_i^T z_j = (x_i^T x_j)^4$$

Equivalent to using a z_i with weighted versions of $x_{i1}^4, x_{i1}^3 x_{i2}, x_{i1}^2 x_{i2}^2, x_{i1} x_{i2}^3, x_{i2}^4, \ldots$

- To also get lower-order terms use $z_i^T z_j = (1 + x_i^T x_j)^4$
- The general degree-p polynomial kernel function:

$$k(x_i, x_j) = (1 + x_i^T x_j)^p$$

  - Works for any number of features 'd'.
  - But cost of computing $z_i^T z_j$ is $O(d)$ instead of $O(d^p)$.

Kernel Trick

- Using polynomial basis of degree 'p' with the kernel trick:
  - Compute K and $\hat{K}$:

$$K_{ij} = (1 + x_i^T x_j)^p \qquad \hat{K}_{ij} = (1 + \hat{x}_i^T x_j)^p$$

test example

train example

  - Make predictions using:

$$\hat{y} = \hat{K}(K + \lambda I)^{-1} y$$

$\hat{y}$: $t \times 1$; $\hat{K}$: $t \times n$; $(K + \lambda I)^{-1}$: $n \times n$; y: $n \times 1$

To form $K = ZZ^T$

- Training cost is only $O(n^2d + n^3)$, despite using $O(d^p)$ features.

To invert $n \times n$ matrix

  - Testing cost is only $O(ndt)$.

To form $\hat{K} = \hat{Z}Z^T$

Linear Regression vs. Kernel Regression

Linear Regression

Training

1. Form basis Z from X.
2. Compute $w = (Z^T Z + \lambda I)^{-1} \backslash (Z^T y)$

Testing

1. Form basis $\hat{Z}$ from $\hat{X}$
2. Compute $\hat{y} = \hat{Z} w$

Kernel Regression

Training:

1. Form inner products K from X.
2. Compute $v = (K + \lambda I)^{-1} \backslash y$

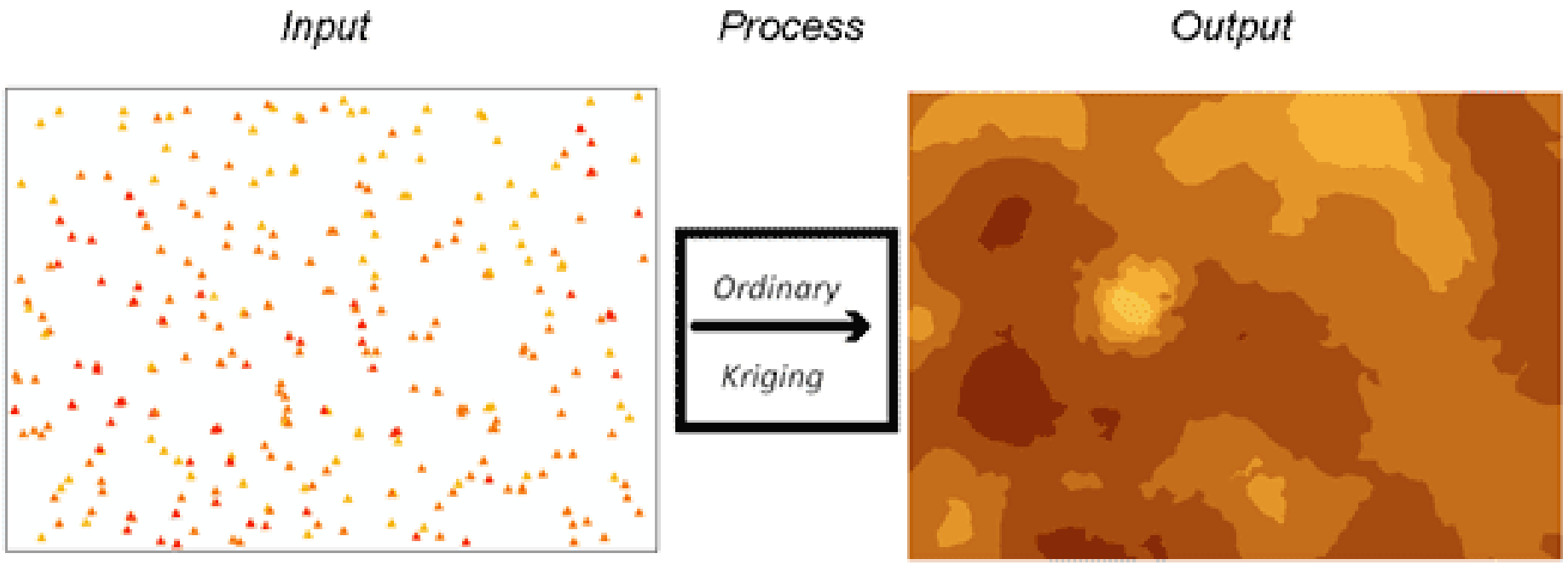
Non-parametric

Testing:

1. Form inner products $\hat{K}$ from X and $\hat{X}$
2. Compute $\hat{y} = \hat{K} v$

Motivation: Finding Gold

- Kernel methods first came from mining engineering ('Kriging'):
  - Mining company wants to find gold.
  - Drill holes, measure gold content.
  - Build a kernel regression model (typically use RBF kernels).



http://www.bisolutions.us/A-Brief-Introduction-to-Spatial-Interpolation.php

Gaussian-RBF Kernel

- Most common kernel is the Gaussian RBF kernel:

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

- Same formula and behaviour as RBF basis, but not equivalent:
  - Before we used RBFs as a basis, now we're using them as inner-product.
- Basis z_i giving the Gaussian RBF kernel is infinite-dimensional.

- Kernel trick lets us fit regression models without explicit features:
  - We can interpret $k(x_i,x_j)$ as a "similarity" between objects x_i and x_j.
  - We don't need z_i and z_j if we can compute 'similarity' between objects.

Kernel Trick for Structure Data

- Consider data that doesn't look like this:

$$X = \begin{bmatrix} 0.5377 & 0.3188 & 3.5784 \\ 1.8339 & -1.3077 & 2.7694 \\ -2.2588 & -0.4336 & -1.3499 \\ 0.8622 & 0.3426 & 3.0349 \end{bmatrix}, \quad y = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix},$$

- But instead looks like this:

$$X = \begin{bmatrix} \text{Do you want to go for a drink sometime?} \\ \text{J'achète du pain tous les jours.} \\ \text{Fais ce que tu veux.} \\ \text{There are inner products between sentences?} \end{bmatrix}, y = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix}.$$

- Instead of using features, can define kernel between sentences.
  - E,g, "string kernels": weighted frequency of common subsequences.
- There are also "image kernels", "graph kernels", and so on…

Valid Kernels

- What kernel functions $k(x_i,x_j)$ can we use?

- Kernel 'k' must be an inner product in some space:
  - There must exist a mapping from x_i to some z_i such that $k(x_i,x_j) = z_i^T z_j$.

- It can be hard to show that a function satisfies this.
- But there are some simple rules for constructing valid kernels from other valid kernels (bonus slide).

Kernel Trick for Other Methods

- Besides L2-regularized least squares, when can we use kernels?
  - Methods based on Euclidean distances between examples:
    - Kernel k-nearest neighbours.
    - Kernel clustering (k-means, DBSCAN, hierarchical).
    - Kernel outlierness.
    - Kernel "Amazon Product Recommendation".
    - Kernel non-parametric regression.

$$\|z_i - z_j\|^2 = z_i^T z_i - 2 z_i^T z_j + z_j^T z_j$$

  - L2-regularized linear models ("representer theorem"):
    - L2-regularized robust regression.
    - L2-regularized logistic regression.
    - L2-regularized support vector machines.

less obvious but true

With a particular implementation testing cost is reduced from $O(ndt)$ to $O(mdt)$ (m: number of support vectors.)

Motivation: How we train on all of Gmail?

- In the Gmail problem from last time, 'n' and 'd' are huge.
  - 'n' is the number of e-mails.
  - 'd' is (number of features)*(number of users + 1).
- Cost of 1 iteration gradient descent for logistic regression is O(nd):
  - O(nd) to compute $w^T x_i$ for all 'i'.
  - O(n) to compute f(x) and each r_i.
  - O(nd) to multiply X^T by 'r'.

$$f(x) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$$

$$\nabla f(x) = X^T r$$

with $$r_i = -\frac{1}{1 + \exp(y_i w^T x_i)}$$

- But it's cheaper than this because x_i are very sparse:
  - Each e-mail has a limited number of non-zero features,
  - Each e-mail only has "global" features and "local" features for one user.

Motivation: How we train on all of Gmail?

- In the Gmail problem from last time, 'n' and 'd' are huge.
  - 'n' is the number of e-mails.
  - 'd' is (number of features)*(number of users + 1).
- Cost of 1 iteration gradient descent for logistic regression is O(ns):
  - Where 's' is the average number of non-zero features.

$$f(x) = \sum_{i=1}^{n} \log(1+\exp(-y_i w^T x_i))$$

$$\nabla f(x) = X^T r \quad \text{with} \quad r_i = -\frac{1}{1+\exp(y_i w^T x_i)}$$

  - O(ns) to compute $w^T x_i$ for all 'i' (just need non-zero values).
  - O(n) to compute f(x) and each r_i.
  - O(ns) to multiply X^T by 'r' (just need non-zero values).
- But how do we deal with the very large 'n'?

Minimizing Sums with Gradient Descent

- Consider minimizing average of differentiable functions:

$$\operatorname*{argmin}_{w \in \mathbb{R}^d} f(w) \quad \text{where} \quad f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$$

$$f_i(w) = (w^T x_i - y_i)^2$$

- Includes all our differentiable losses as special cases.
- Gradient descent for this problem:

$$w^{t+1} = w^t - \alpha_t \nabla f(w^t)$$

$$= w^t - \alpha_t \left(\frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w^t)\right)$$

G-mail: $f(w) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{\log(1+\exp(y_i w^T x_i))}_{f_i(w)}$

- Nice properties, but iterations require gradients of all 'n' examples.
- Key idea behind stochastic gradient methods:
  - On average, we can decrease 'f' using the gradient of a random example.

Stochastic Gradient Method

- Stochastic gradient method:
  1. Pick a random example i_t.
  2. Perform a gradient descent step based only on this example.

$$w^{t+1} = w^t - \alpha_t \nabla f_{i_t}(w^t)$$

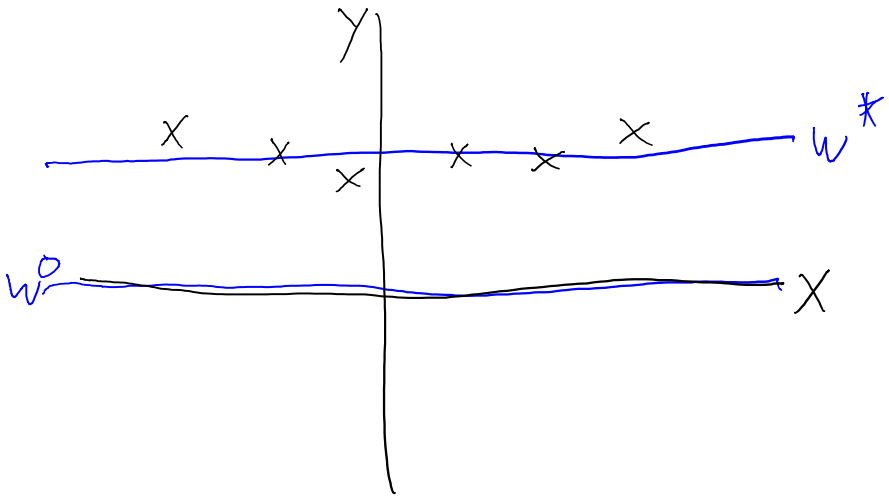
- Intuition: unbiased estimate of full gradient:

$$E_{i_t}[\nabla f_{i_t}(w^t)] = \sum_{i=1}^{n} \left(\frac{1}{n}\right) \nabla f_i(w^t) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w^t) = \nabla f(w^t)$$

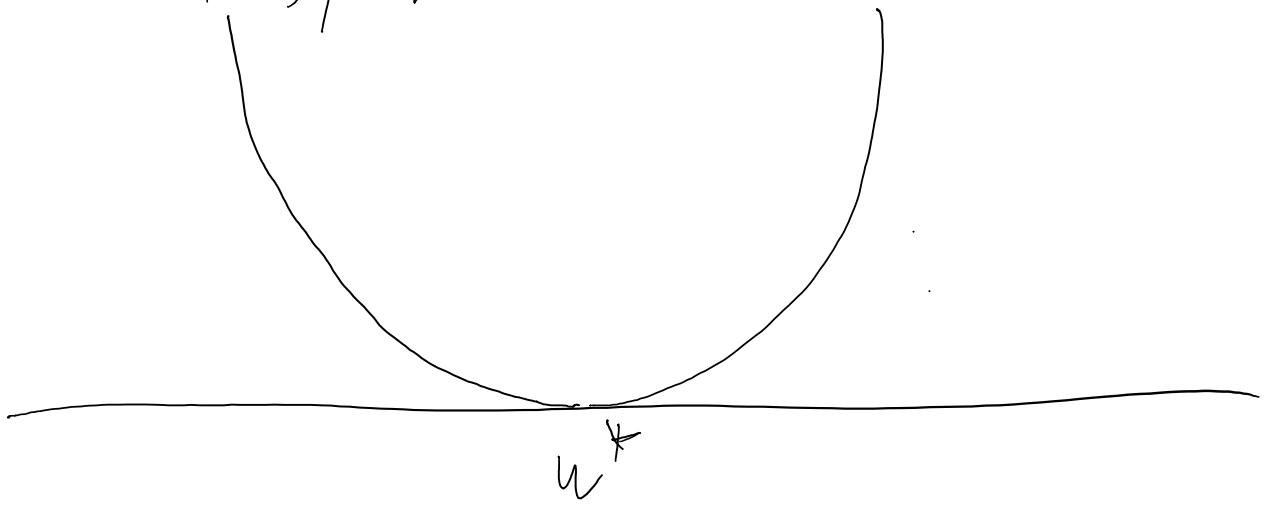
- Key advantage:
  - Iteration cost is O(d), it does not depend on 'n'.
  - If 'n' is 1 billion, it is 1 billion times faster than gradient descent.
- But does this actually work?

Deterministic Gradient Method in Action

Consider just estimating bias:

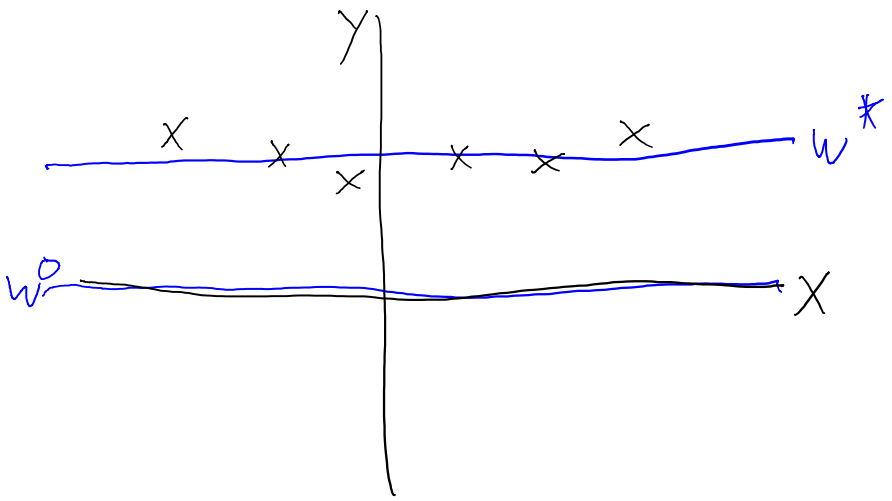


Overall squared error:

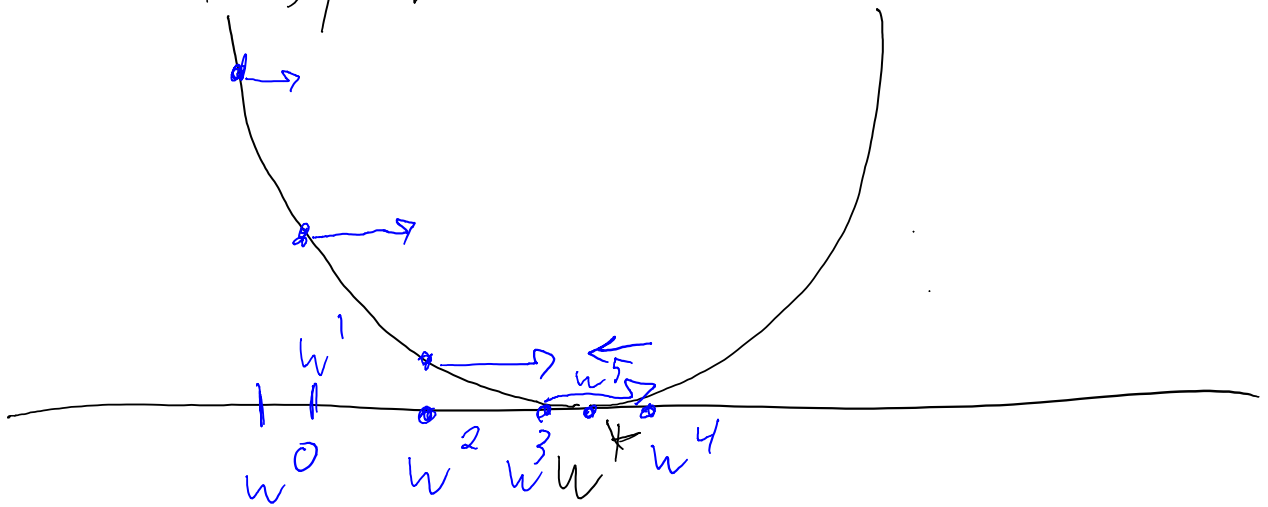


Deterministic Gradient Method in Action

Consider just estimating bias:

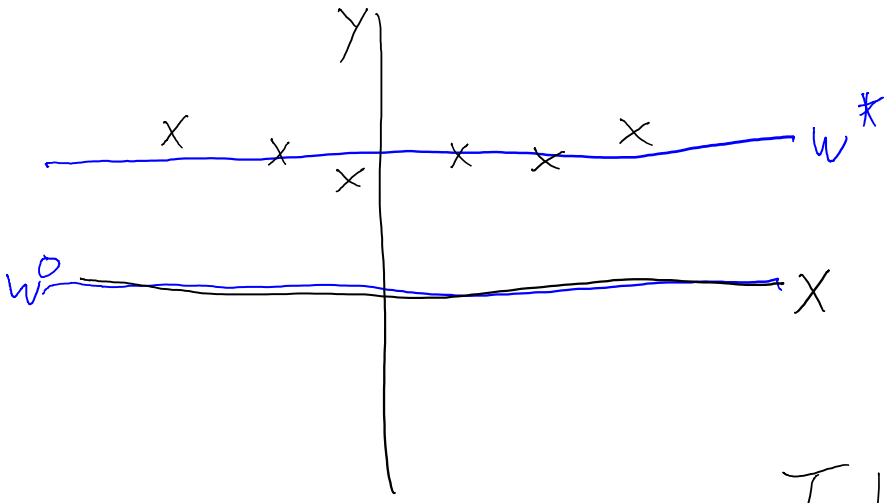


Overall squared error:

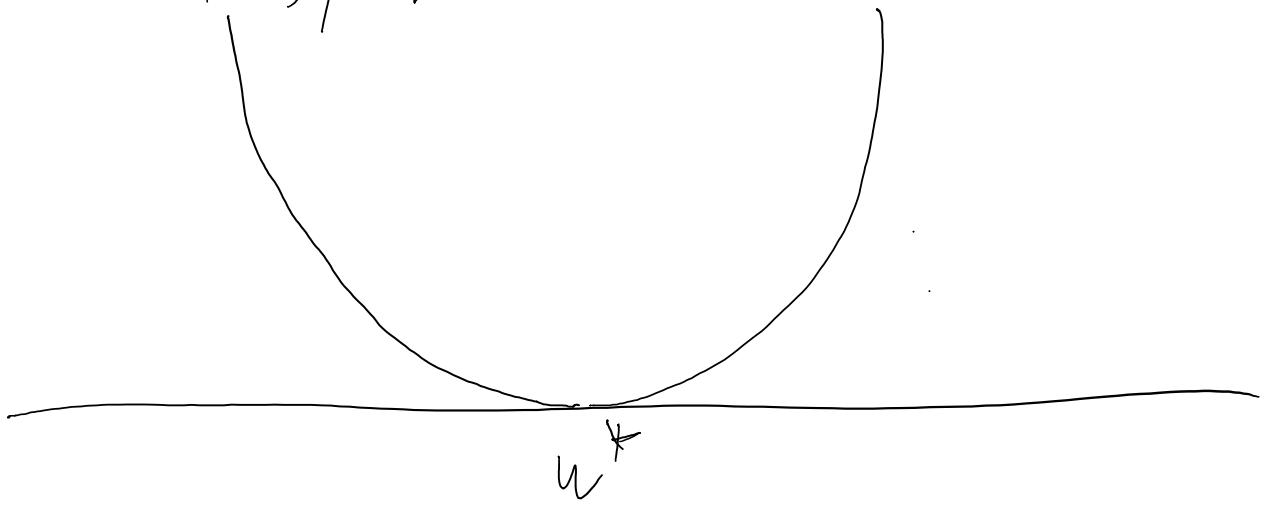


Stochastic Gradient Method in Action

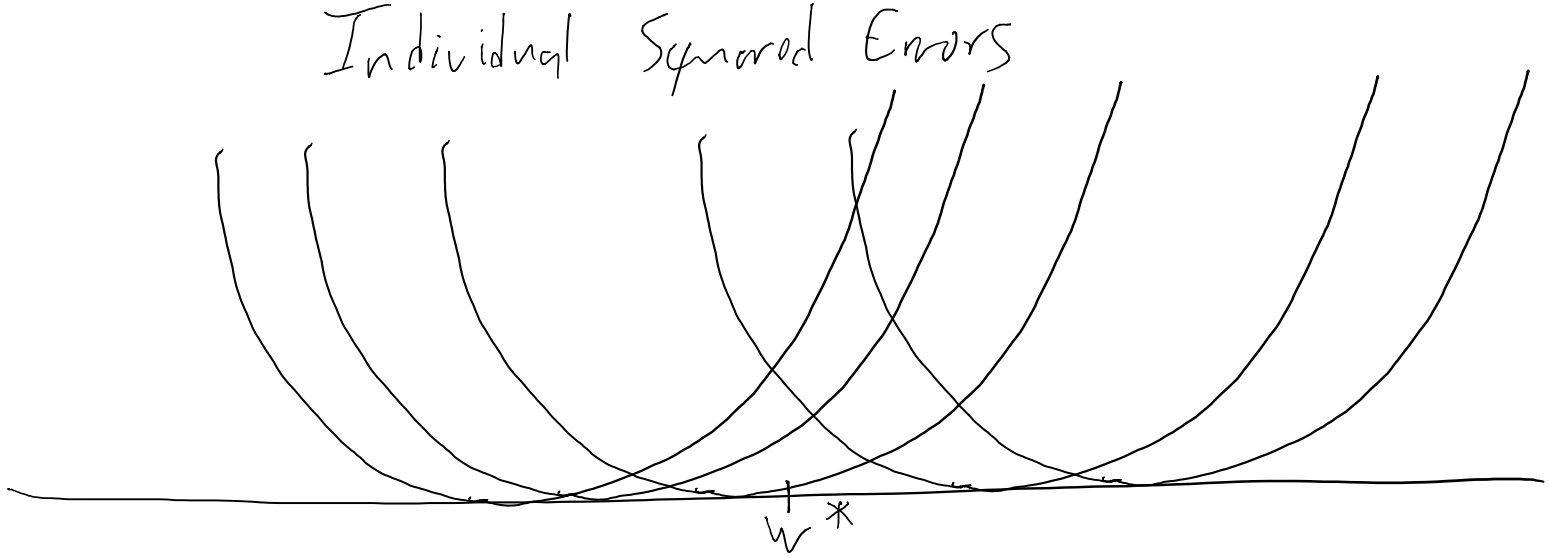
Consider just estimating bias:



Overall squared error:

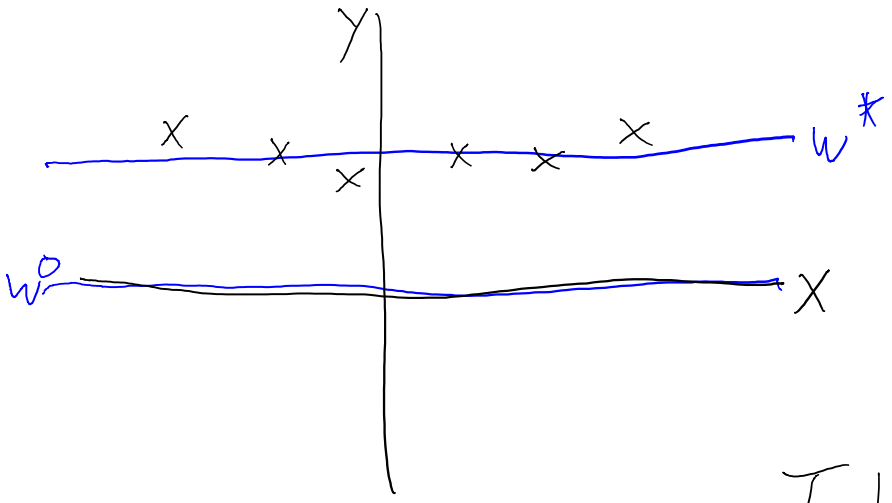


Individual Squared Errors

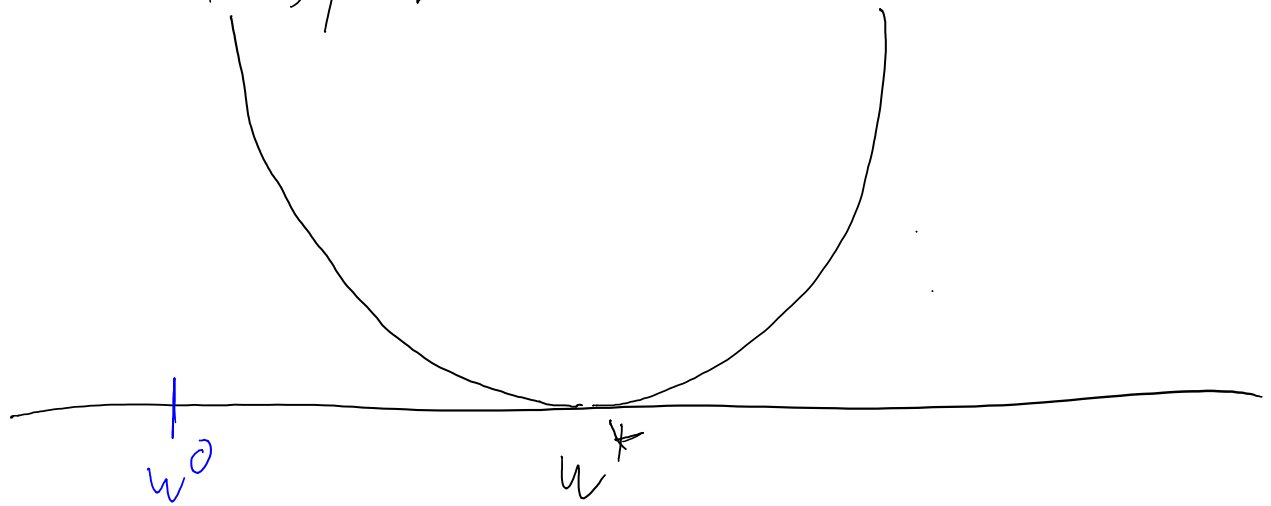


Stochastic Gradient Method in Action

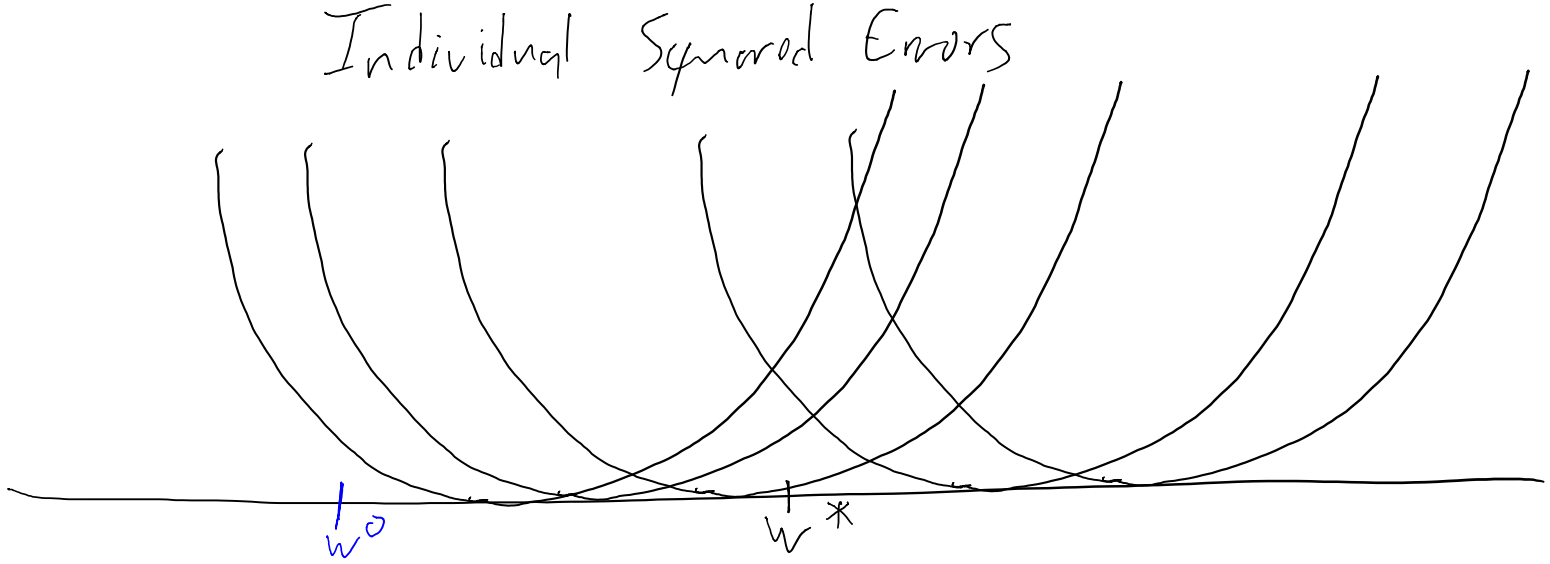
Consider just estimating bias:



Overall squared error:

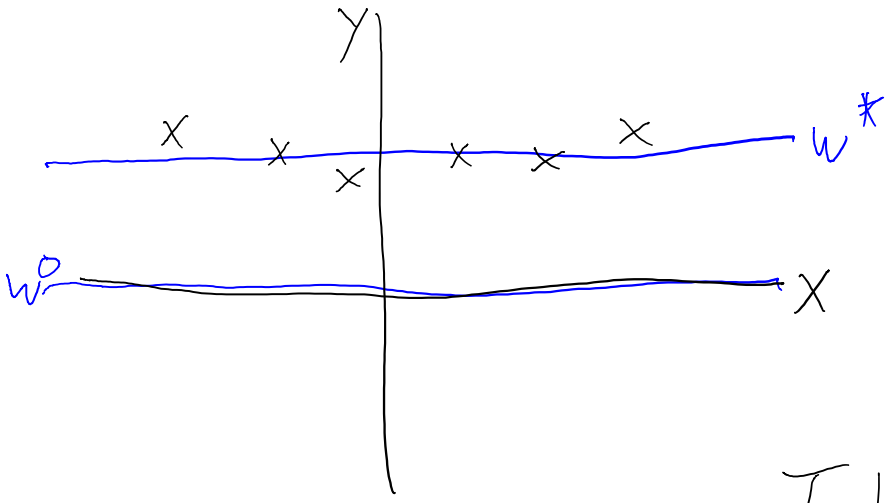


Individual Squared Errors

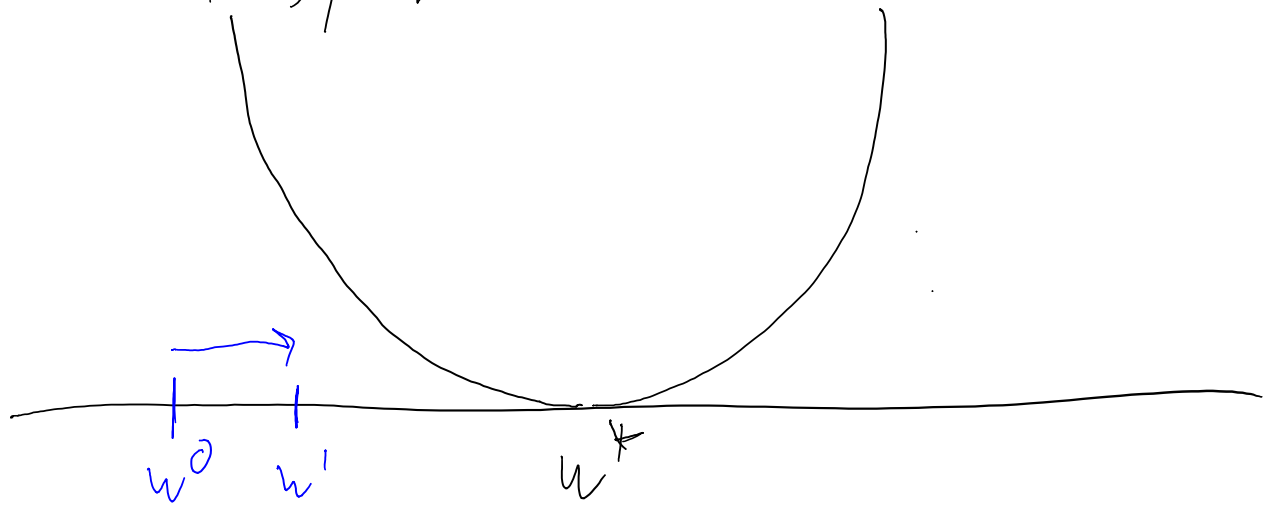


Stochastic Gradient Method in Action

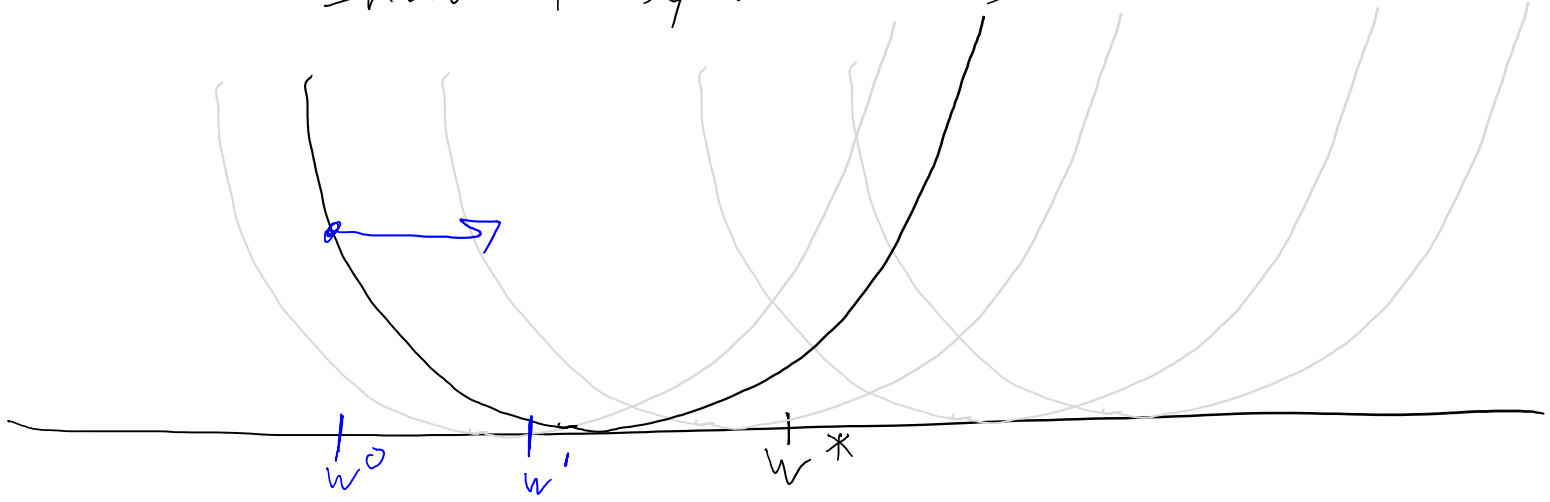
Consider just estimating bias:



Overall squared error:

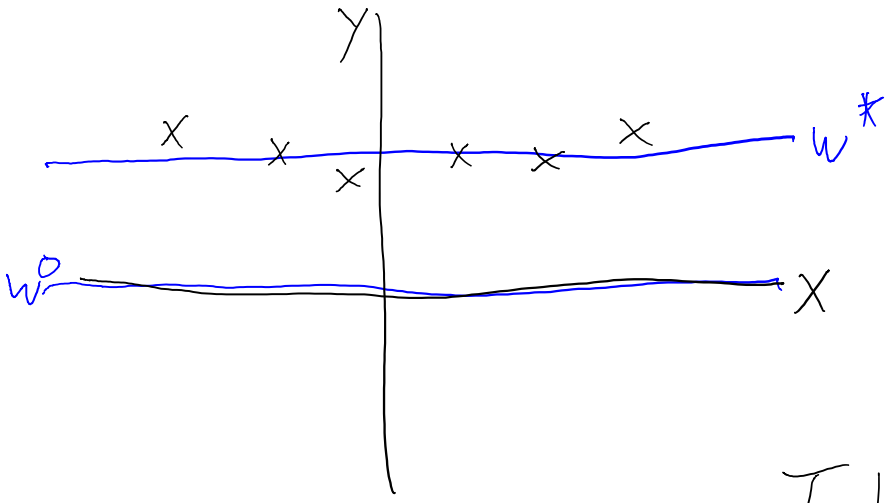


Individual Squared Errors

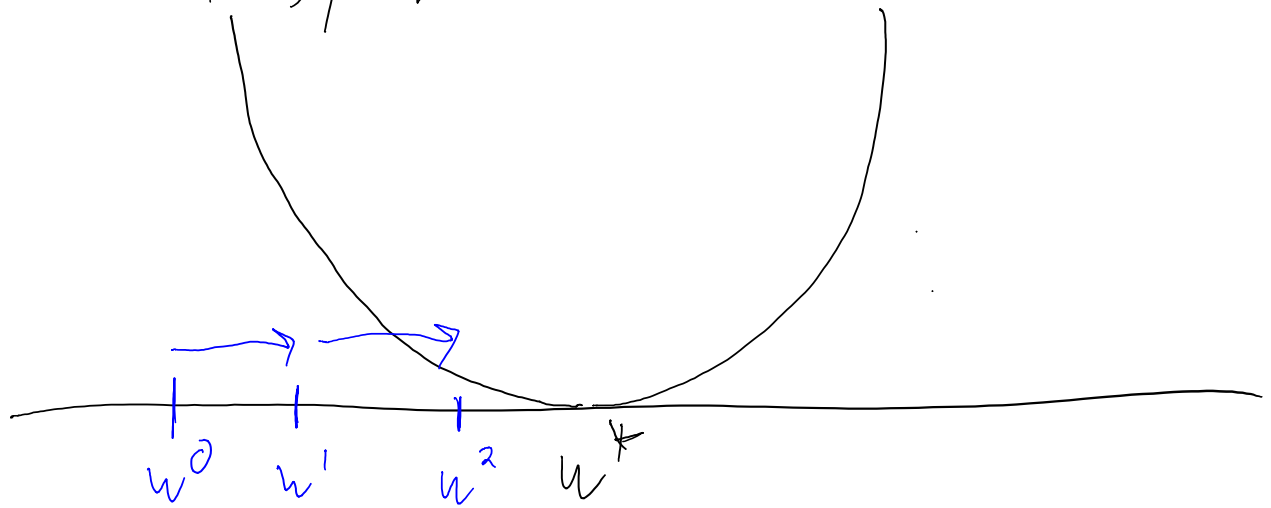


Stochastic Gradient Method in Action

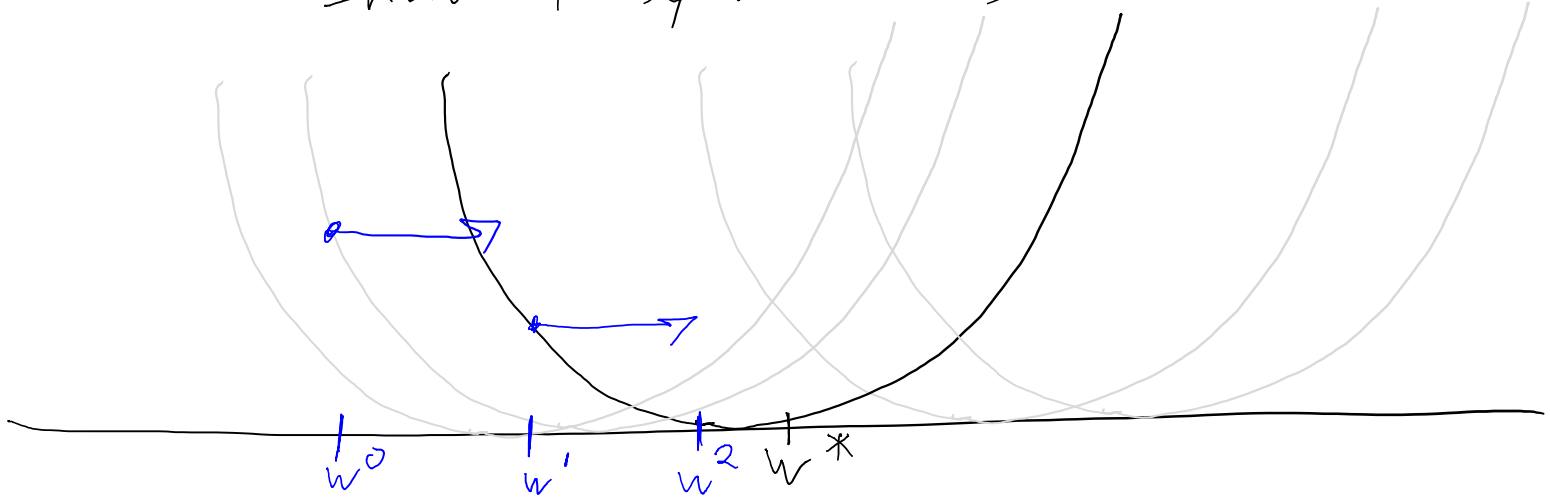
Consider just estimating bias:



Overall squared error:

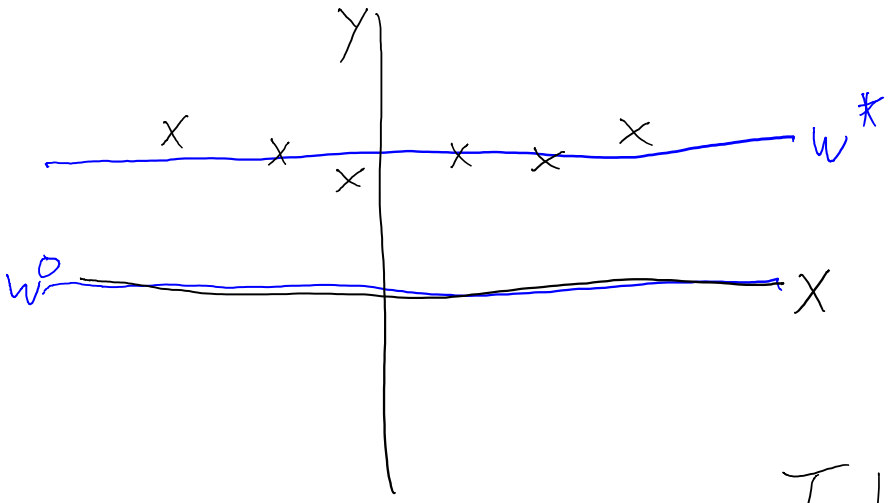


Individual Squared Errors

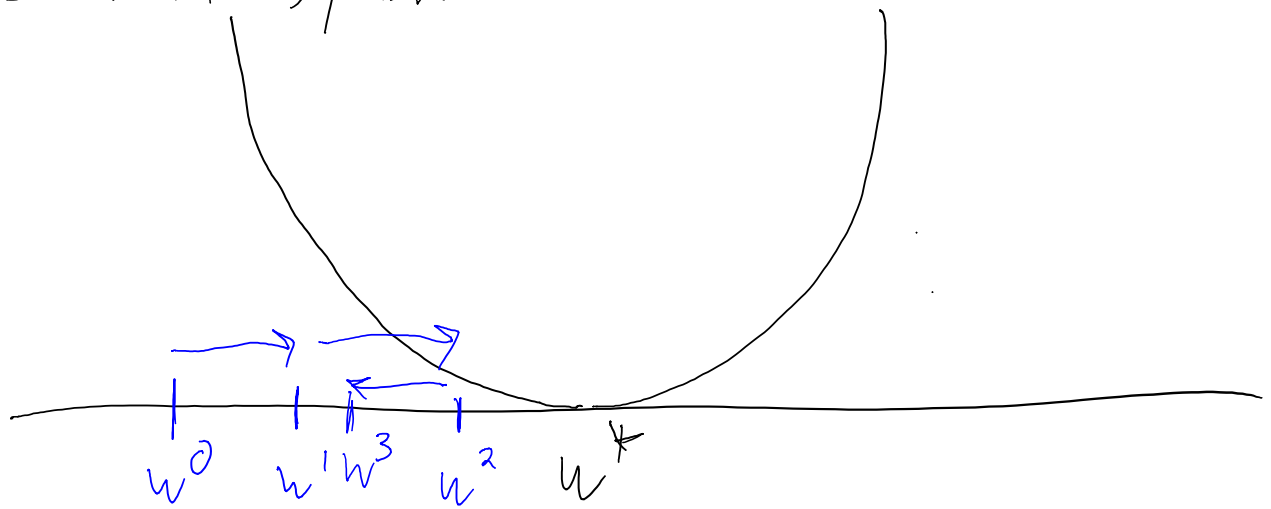


Stochastic Gradient Method in Action

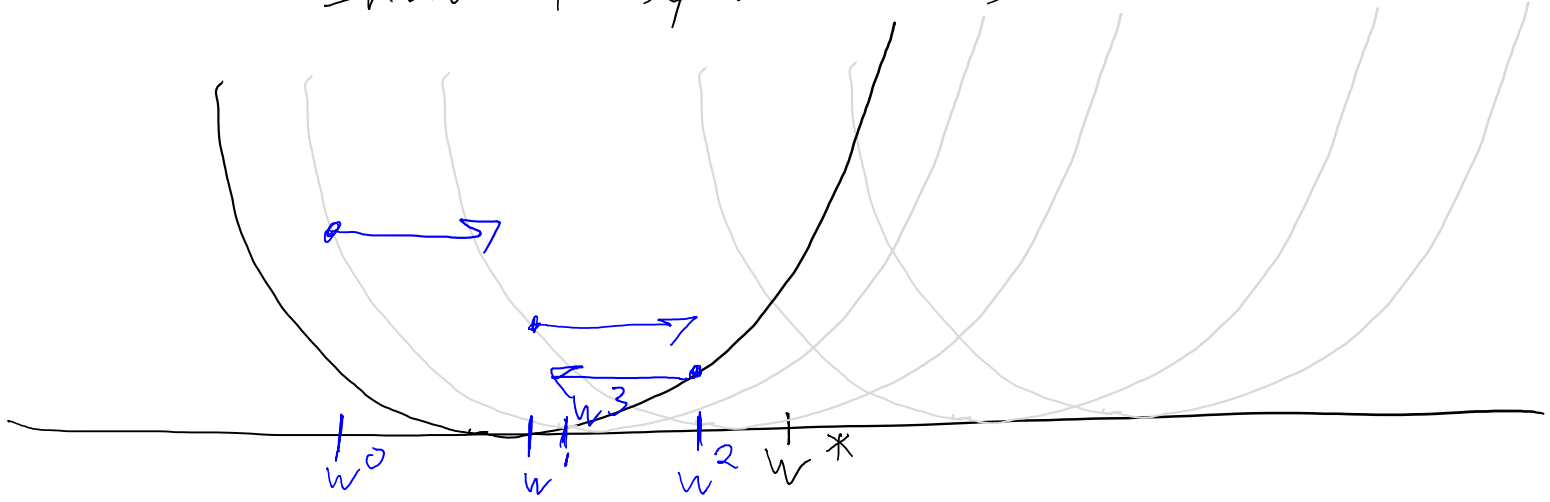
Consider just estimating bias:



Overall squared error:

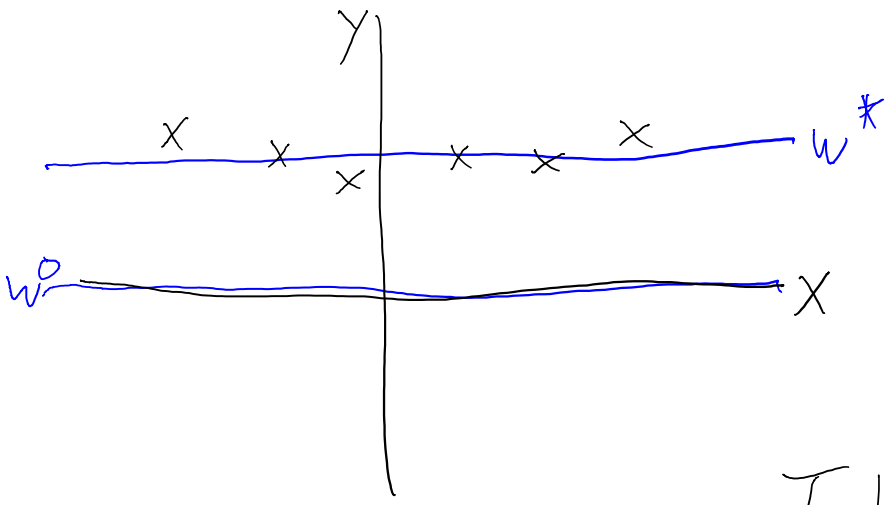


Individual Squared Errors

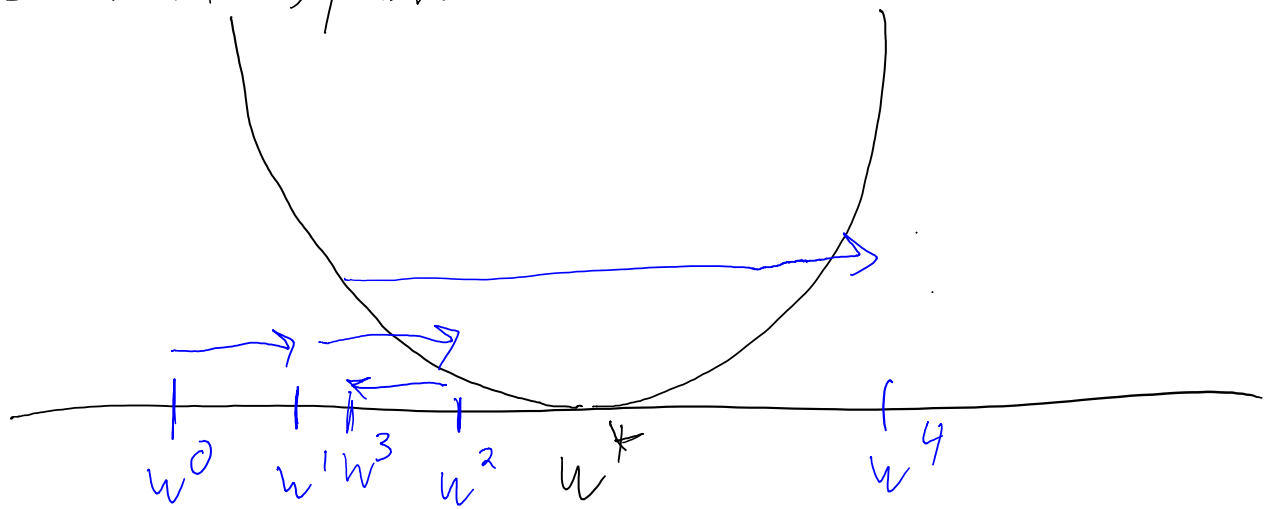


Stochastic Gradient Method in Action

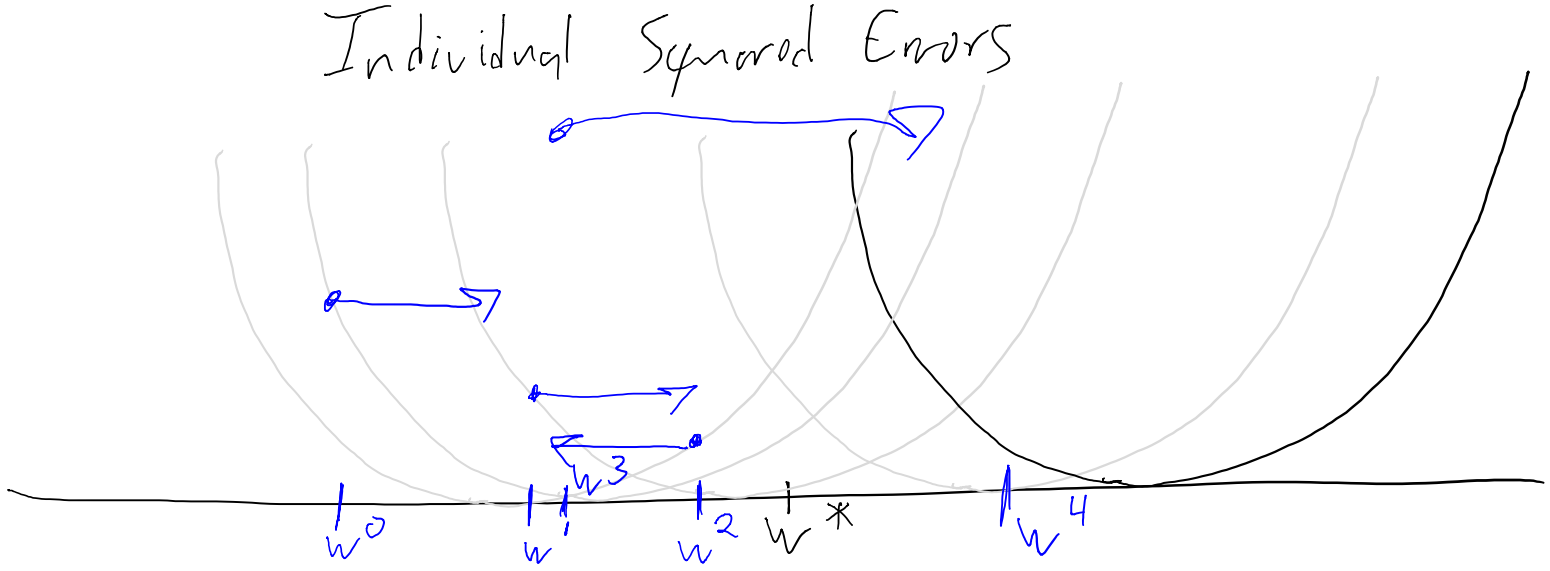
Consider just estimating bias:



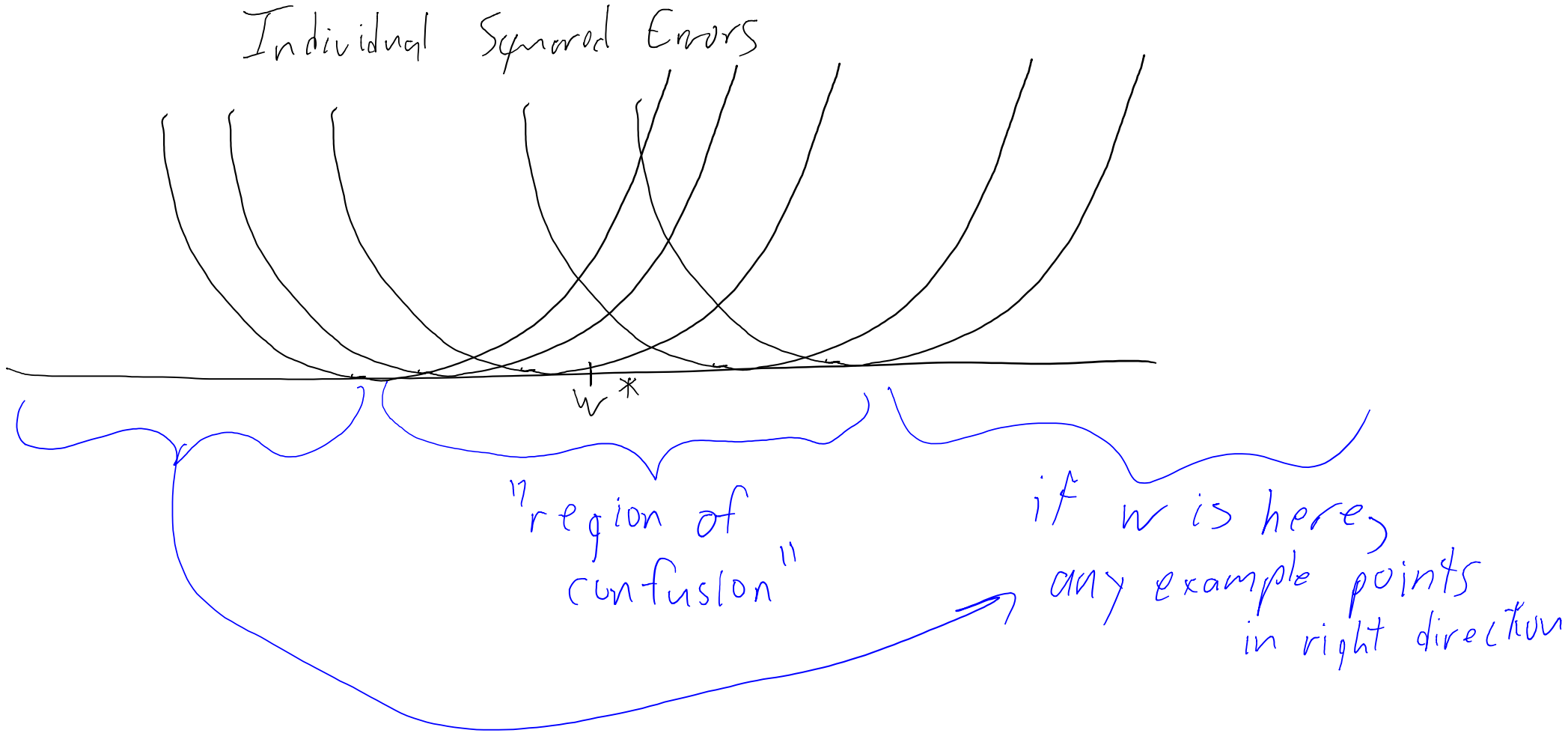
Overall squared error:



Individual Squared Errors



Stochastic Gradient Method in Action



Convergence of Stochastic Gradient

- Problem is that stochastic gradient step might increase error 'f':
  - Since you only look at one example, you can't just check 'f'.
- Key property used for convergence:
  - If the sequence of w^t are sufficiently 'close', we decrease 'f' on average.
  - How 'close' they need to be depends on how close we are to minimum.
- To get convergence, we need a decreasing sequence of step sizes:
  - Need to converge to zero fast enough (makes variance go to 0).
  - Can't converge to zero too quickly (need to be able to get anywhere).
- For example: $\alpha_t = O(\frac{1}{t})$ implies that $\sum_{t=1}^{\infty} \alpha_t = \infty$, $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$

not too small not too big

Summary

- Kernels let us use similarity between objects, rather than features.
- Stochastic gradient methods let us use huge datasets.
- Convergence of stochastic gradient requires decreasing step sizes.

- Next time:
  - Non-binary discrete labels like categories, counts, rankings, etc.