SVAN 2016 Mini Course: Stochastic Convex Optimization Methods in Machine Learning

Mark Schmidt
University of British Columbia, May 2016
www.cs.ubc.ca/~schmidtm/SVAN16

Some images from this lecture are taken from Google Image Search.

Last Time: Supervised Learning

- We discussed supervised learning:
  - We have a set of inputs x_i and a corresponding output y_i.
  - Food allergy example:
    - x_i is the quantities of food we ate on day 'i'.
    - y_i is the level of IgE we measure on day 'i'.
  - The goal is to learn a function 'f' such that $(f(x_i) - y_i)$ is small.
- We introduced standard notation for supervised learning:

$$X = \underbrace{\begin{bmatrix} \text{---} & x_1^T & \text{---} \\ \text{---} & x_2^T & \text{---} \\ \text{---} & x_3^T & \text{---} \\ & \vdots & \\ \text{---} & x_n^T & \text{---} \end{bmatrix}}_{n \times d} \qquad y = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}}_{d \times 1}$$

Last Time: Linear Regression and Least Squares

- We considered the special case of linear regression:

$$\hat{y}_i = w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + \cdots + w_d x_{id} = w^T x_i$$

prediction for example 'i'

regression weights

linear combination of x_i.

- To fit this model, a classic approach is least squares:

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

- Which we can write in matrix notation as:

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \quad \|Xw - y\|^2$$

Solution: $w = (X^T X)^{-1} X^T y$

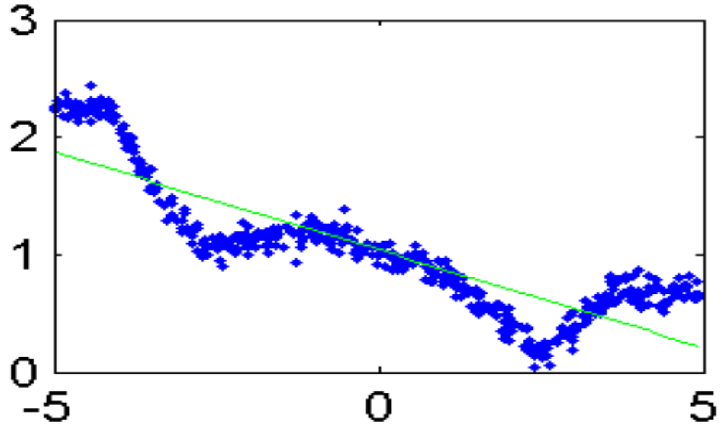
(invertible $X^T X$)

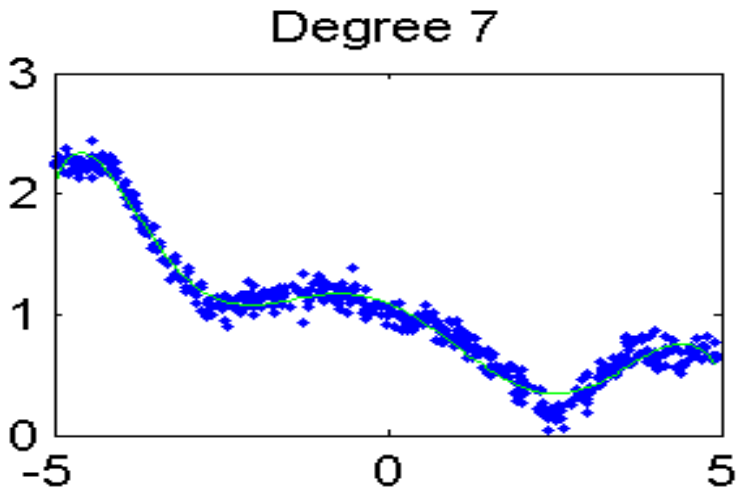
Last Time: Nonlinear Basis

- Change of basis allows nonlinear functions with linear regression:

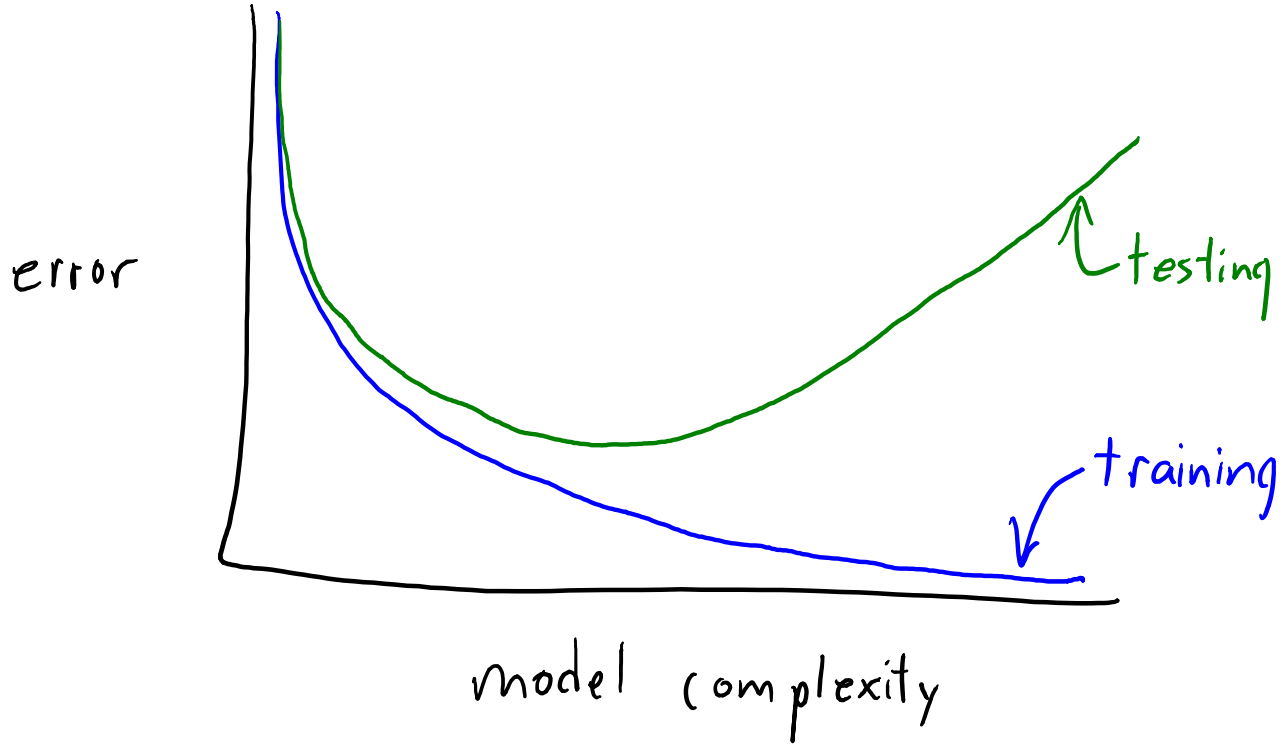
$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$X_{poly} = \begin{bmatrix} 1 & x_1 & (x_1)^2 & \cdots & (x_1)^p \\ 1 & x_2 & (x_2)^2 & \cdots & (x_2)^p \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & (x_n)^2 & \cdots & (x_n)^p \end{bmatrix}$$





Fundamental Trade-Off of Machine Learning



- Same trade-off exists as we add more features:
  - More features means lower training error.
  - More features means training error is worse approximation of test error.

Controlling Complexity

- We know that complex models can overfit.
- But usually the "true" mapping from x_i to y_i is complex.
- So what do we do???

- There are many possible answers:
  - Model averaging: average over multiple models to decrease variance.
  - Regularization: add a penalty on the complexity of the model.

L2-Regularization

- Our standard least squares formulation:

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \ \frac{1}{2}\|Xw - y\|^2$$

- Standard regularization strategy is to add a penalty on the L2-norm:

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \ \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2$$

- Regularization parameter λ controls 'strength' of regularization:
  - If λ is large then it forces 'w' to be very small: low complexity.
  - If λ is tiny then 'w' can be get huge: high complexity.
- Has been re-invented several times:
  - Tikhonov regularization, ridge regression, etc.

L2-Regularization

- In terms of fundamental trade-off:
  - Regularization increases training error.
  - Regularization makes training error a better approximation of test error.
- How should you choose λ?
  - Theory: as 'n' grows λ should be in the range O(1) to $O(n^{-1/2})$.
  - Practice: optimize validation set or cross-validation error.
    - This almost always decreases the test error.
- How do you compute 'w'?

Ridge Regression Calculation

Objective: $f(w) = \frac{1}{2}(y - Xw)^T(y - Xw) + \frac{\lambda}{2} w^T w.$

Gradient: $\nabla f(w) = X^T X w - X^T y + \lambda w$

Setting to zero: $X^T X w + \lambda w = X^T y$, or

$$(X^T X + \lambda I) w = X^T y.$$

Pre-multiply by $(X^T X + \lambda I)^{-1}$, which always exists:

$$\boxed{w = (X^T X + \lambda I)^{-1} X^T y.}$$

In Matlab:

```
w=(X'*X+lambda*eye(d))^-1(X'*y)
```

Cost: same as least squares.

Why use L2-Regularization?

- Mark says: "You should always use regularization."
- "Almost always improves test error" should already convince you.
- But here are more reasons:
  1. Solution 'w' is unique.
  2. Does not require X'X to be invertible.
  3. Solution 'w' is less sensitive to changes in X or y.
  4. You can use Cholesky factorization instead of LU factorization.
  5. Makes large-scale methods for computing 'w' run faster.
  6. Stein's paradox: if $d \geq 3$, regularization moves us closer to 'true' w.
  7. In the worst case you just set λ small and get the same performance.

(pause)

Parametric vs. Non-Parametric

- Polynomials are not the only possible bases:
  - Common to use exponentials, logarithms, trigonometric functions, etc.
  - The right basis will vastly improve performance.
  - But when you have a lot of features, the right basis may not be obvious.
- The above bases are parametric model:
  - The size of the model *does not depend* on the number of training examples 'n'.
  - As 'n' increases, you can estimate the model more accurately.
  - But at some point, more data doesn't help because model is too simple.
- Alternative is non-parametric models:
  - Size of the model grows with the number of training examples.
  - Model gets more complicated as you get more data.
  - You can model very complicated functions where you don't know the right basis.

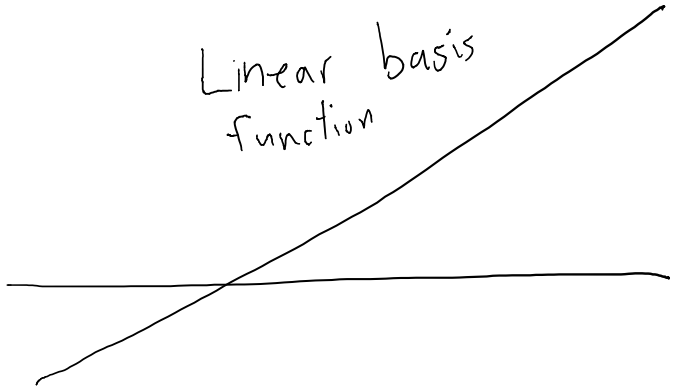
Non-Parametric Basis: RBFs

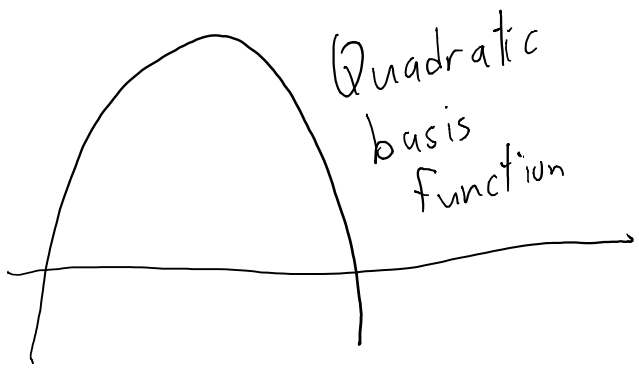
- Radial basis functions (RBFs):
  - Non-parametric bases that depend on distances to training points.
- Most common example is Gaussian or squared exponential:

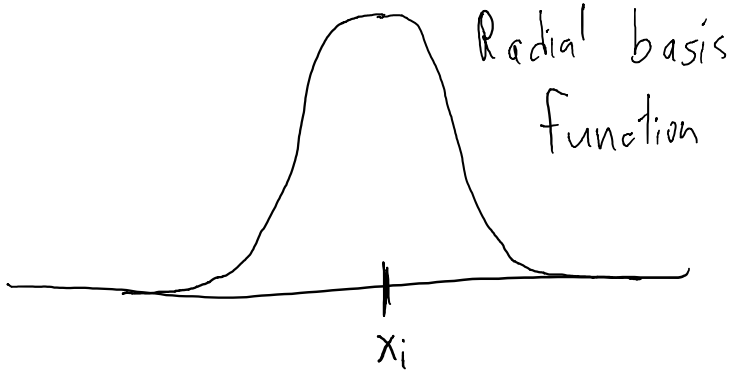
training example

$$f(x) = \exp\left(-\frac{\|x - x_i\|^2}{2\sigma^2}\right)$$

parameter (speed that it goes to zero)







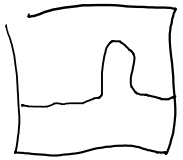
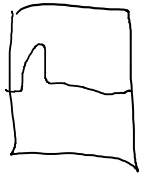
Where did constant $\sigma\sqrt{2\pi}$ go?
– not needed:
$w^T x_i = (\frac{1}{c} w)^T (c x_i)$

Non-Parametric Basis: RBFs

Note: each basis function depends on all original features:
$\|x - x_i\|^2 = \sum_{j=1}^{d} (x_j - x_{ij})^2$

- Radial basis functions (RBFs):
  - Non-parametric bases that depend on distances to training points.
- Most common example is Gaussian or squared exponential:

$$X_{rbf} = \begin{bmatrix} \exp\left(-\frac{\|x_1 - x_1\|^2}{2\sigma^2}\right) & \exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right) & \cdots & \exp\left(-\frac{\|x_1 - x_n\|^2}{2\sigma^2}\right) \\ \exp\left(-\frac{\|x_2 - x_1\|^2}{2\sigma^2}\right) & \ddots & & \vdots \\ \vdots & & \ddots & \\ \exp\left(-\frac{\|x_n - x_1\|^2}{2\sigma^2}\right) & \cdots & \cdots & \exp\left(-\frac{\|x_n - x_n\|^2}{2\sigma^2}\right) \end{bmatrix}$$

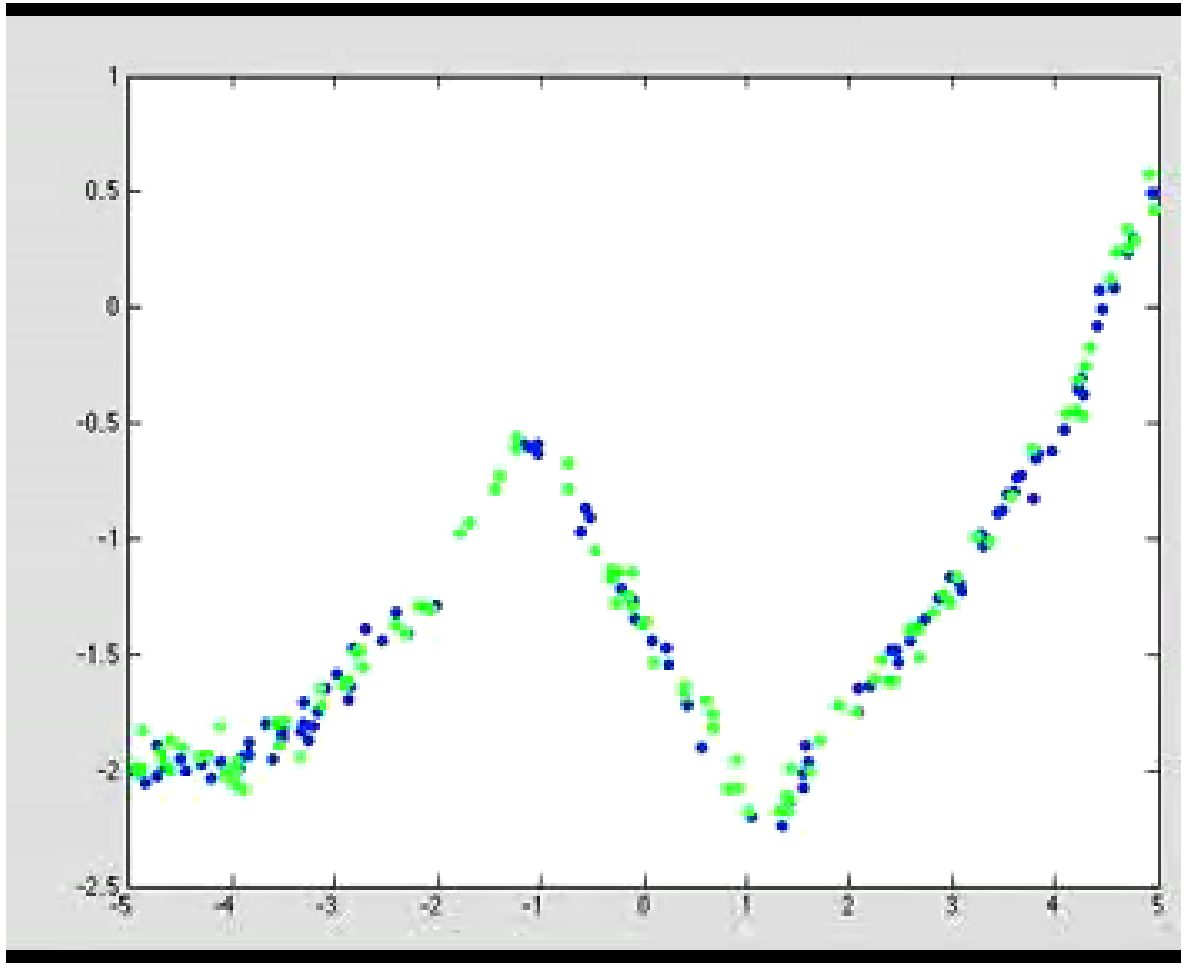
$\hat{y}_i = w_1$  $+ w_2$  $+ \cdots + w_n$ 

X_{rbf} is n by n.

- Gaussian RBFs are universal approximators (compact subets of $\mathbb{R}^d$)
  - Can approximate any continuous function to arbitrary precision.

Non-Parametric Basis: RBFs

- RBF basis for different values of σ:



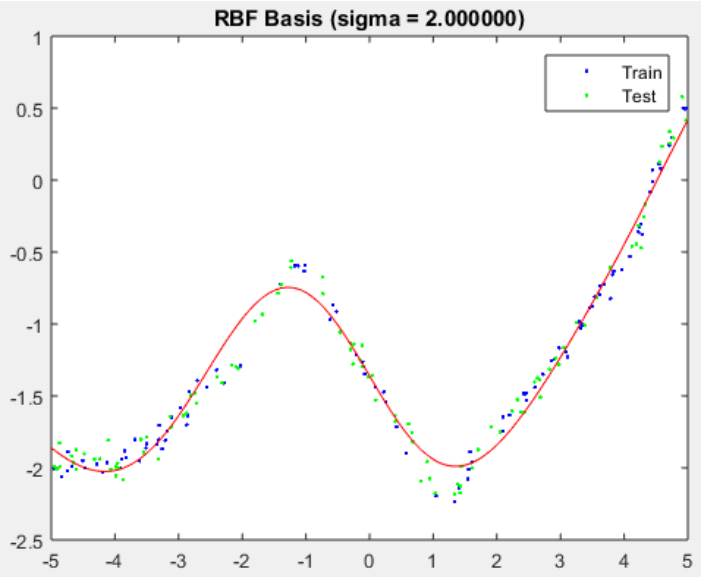
Could add bias and linear basis:

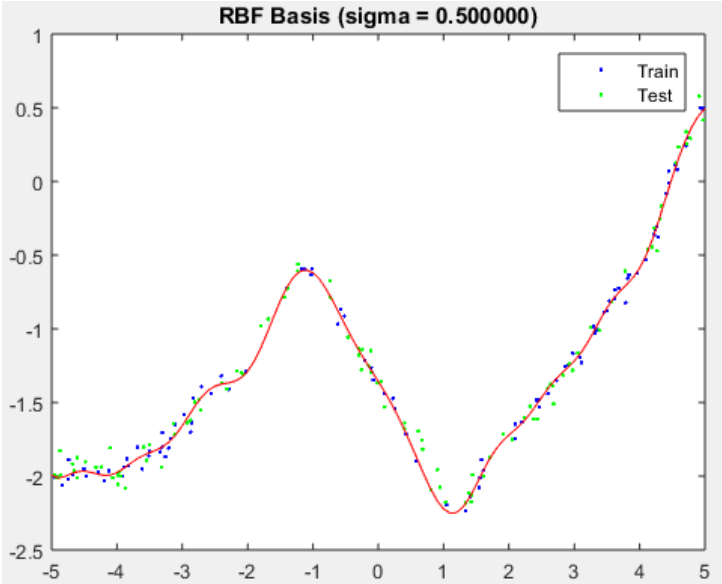
$$\tilde{X} = \underbrace{\begin{bmatrix} | & - x_1 - & \\ | & - x_2 - & \\ | & - x_3 - & \\ \vdots & \vdots & X_{rbf} \\ | & - x_n - & \end{bmatrix}}_{1 \quad d \quad n}$$

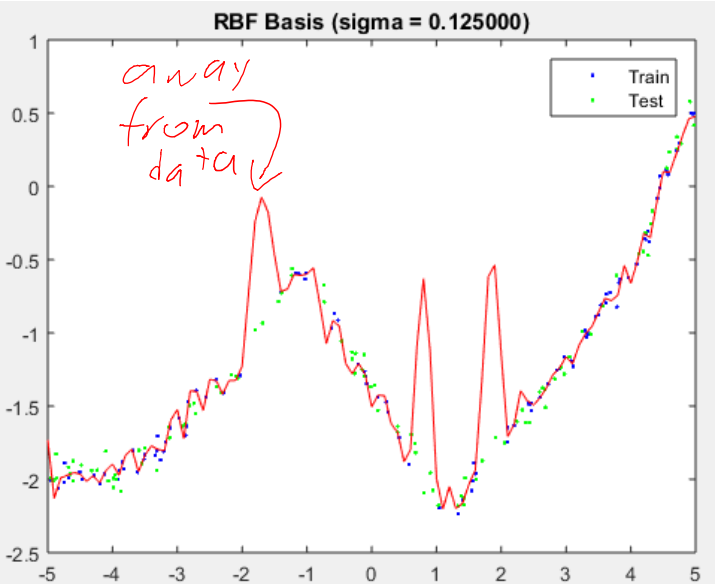
Now it defaults to linear regression instead of 0 away from data.

RBFs, Regularization, and Validation

- Very effective model:
  - RBF basis with L2-regularization and cross-validation to choose σ and λ.



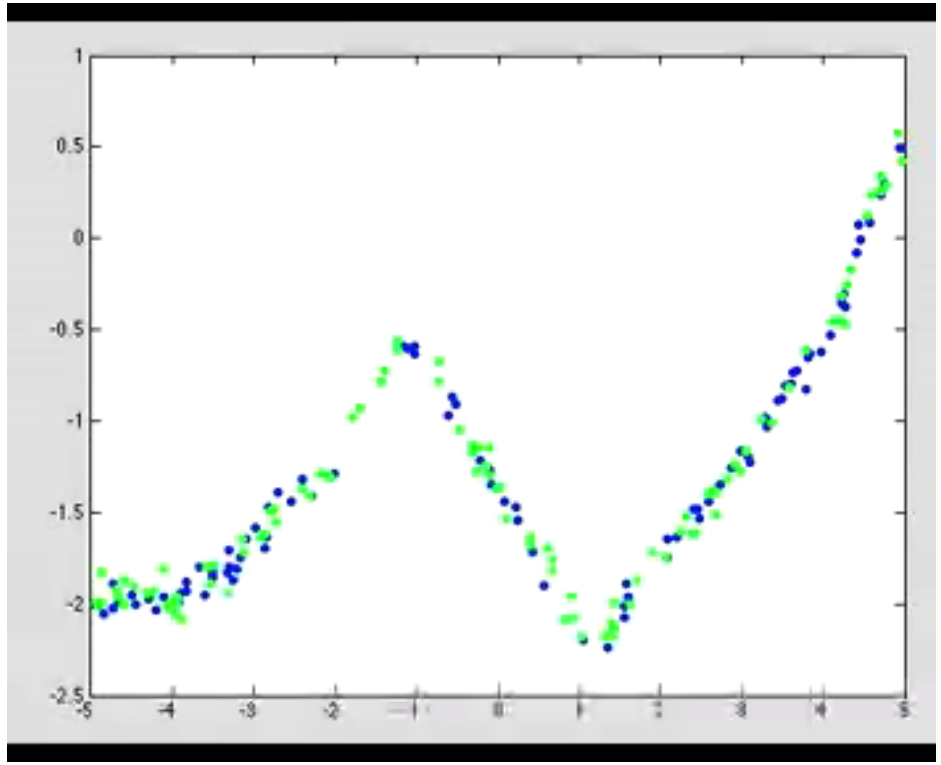




  - Expensive at test time: need distance to all training examples.

RBFs, Regularization, and Validation

- RBF basis with L2-regularization for different values of σ and λ.



- At least one of these models is often a good fit.

(pause)

Alternatives to Squared Error

- Squared error is computationally convenient choice:
  - Solution involves solving a linear system.

$$w = (X^T X + \lambda I)^{-1} X^T y$$

- But it's usually not the right choice:
  - Corresponds to assuming error are normally distributed (later in lecture).
  - Makes it sensitive to outliers or large errors.
  - Makes it inappropriate with restrictions on y (like binary or censored).
- There are many alternatives to squared error.
  - But these have computational implications.

Least Squares with Outliers

- Consider fitting least squares with an outlier in the labels:
  - Observation that is unusually different from the others.

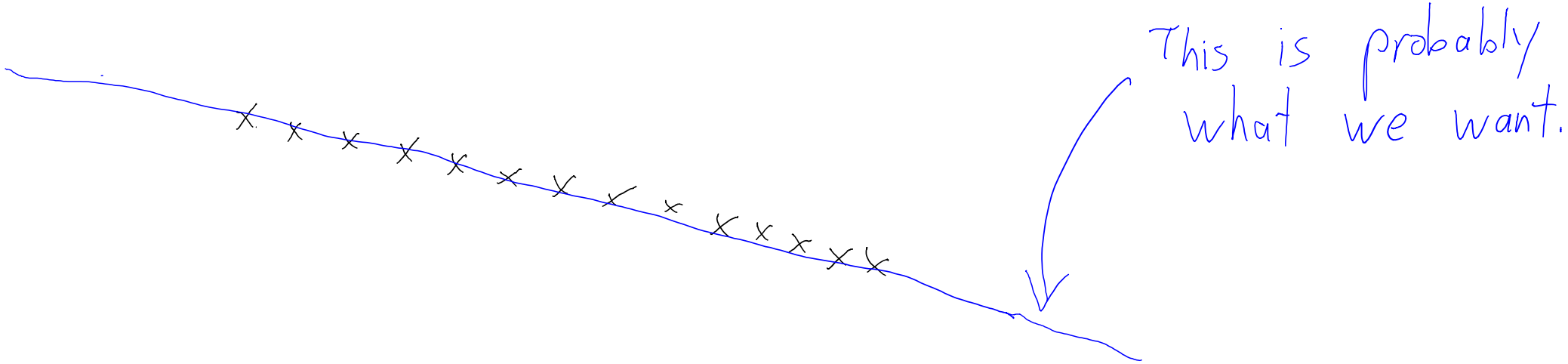
	Egg	Milk	Fish	Wheat	Shellfish	Peanuts	…	IgE
Day 1	0	0.7	0	0.3	0	0		700
Day 2	0.3	0.7	0	0.6	0	0.01		740
Day 3	0	0	0	0.8	0	0		50
Day 4	0.3	0.7	1.2	0	0.10	0.01		40000

- Some sources of outliers:
  - Errors, contamination of data from different distribution, rare events.

Least Squares with Outliers

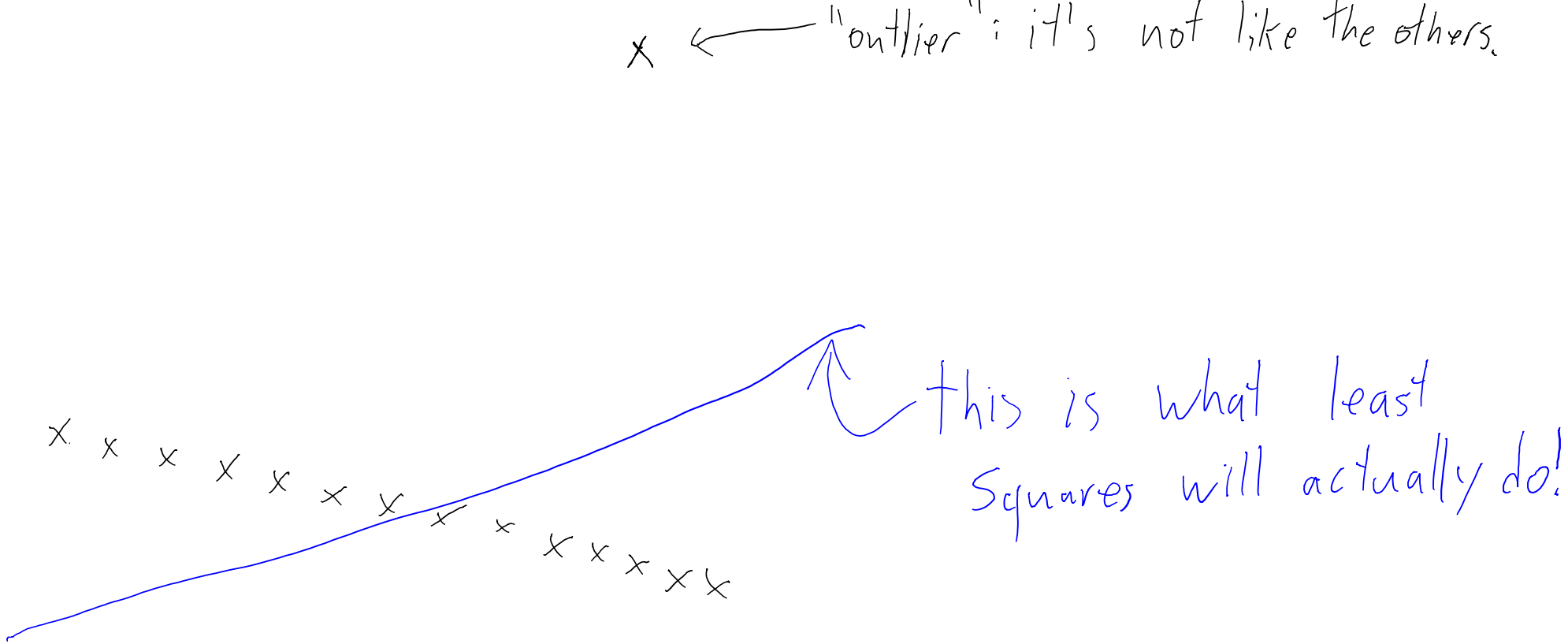
- Consider fitting least squares with an outlier in the labels:

"outlier": it's not like the others.



Least Squares with Outliers

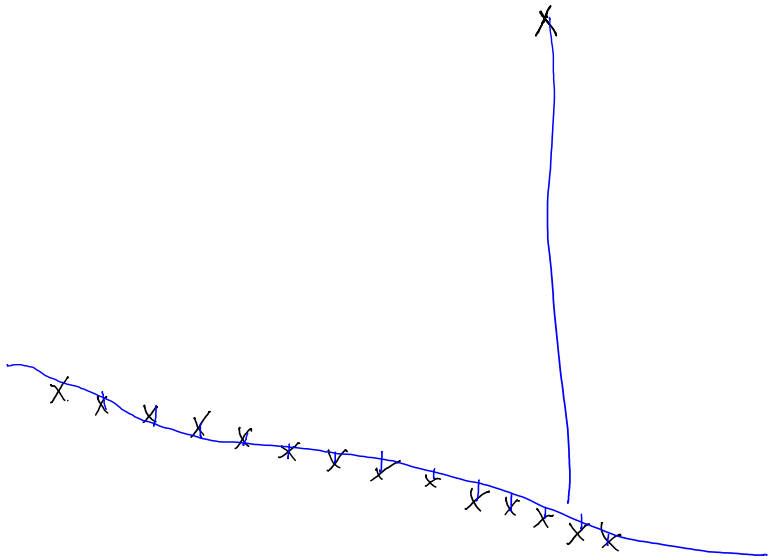
- Consider fitting least squares with an outlier in the labels:

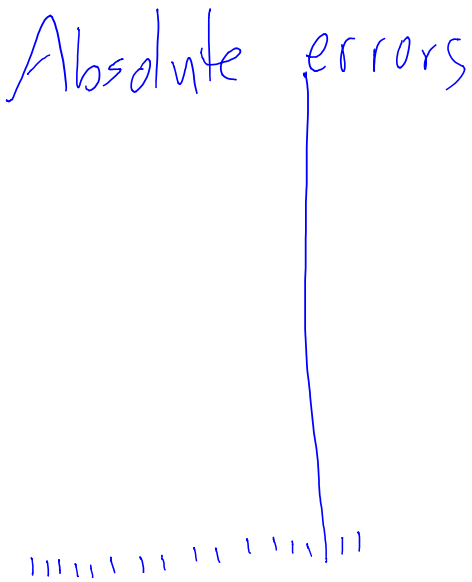


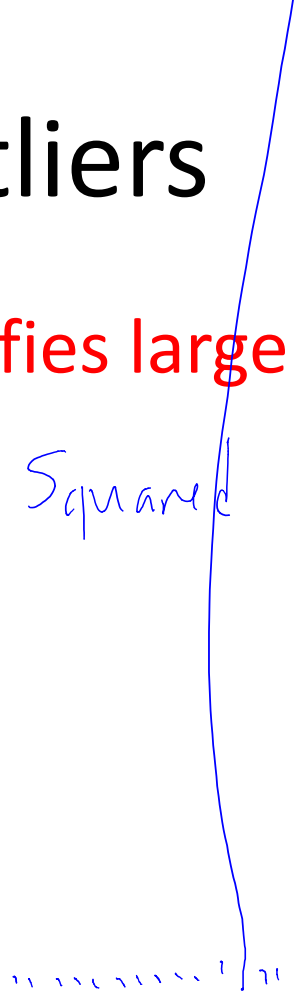
- Least squares is very sensitive to outliers.

Least Squares with Outliers

- Squaring error shrinks small errors, and magnifies large errors:







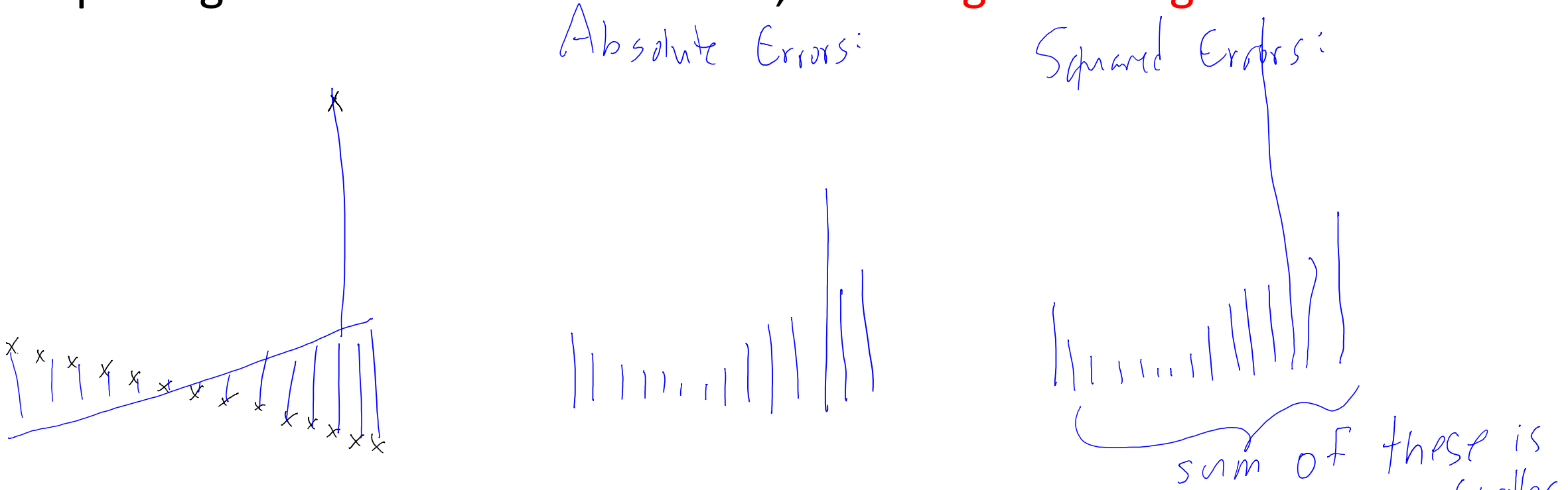
$(5)^2 = 25$

$(1/2)^2 = 1/4$

- Outliers (large error) influence ‘w’ much more than other points.

Least Squares with Outliers

- Squaring error shrinks small errors, and magnifies large errors:



- Outliers (large error) influence 'w' much more than other points.
  - Good if outlier means 'plane crashes', bad if it means 'data entry error'.

Robust Regression

- Robust regression objectives put less focus on far-away points.
- For example, just use absolute error:

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} |w^T x_i - y_i|$$

- Now decreasing 'small' and 'large' errors is equally important.
- In matrix notation, we can write this as minimizing L1-norm:

$$\|r\|_1 = \sum_{i=1}^{n} |r_i|$$

Let "residual" vector 'r' have elements

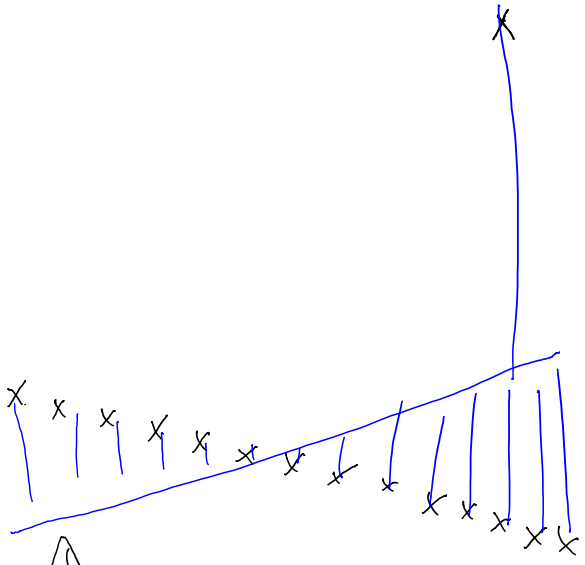
$$r_i = w^T x_i - y_i$$

so $r = Xw - y$

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \|Xw - y\|_1$$

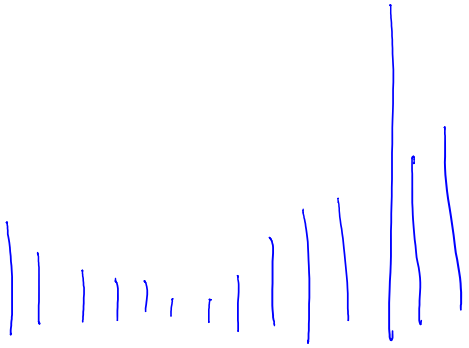
Squared Error vs. Absolute Error

- Comparing squared error absolute error:

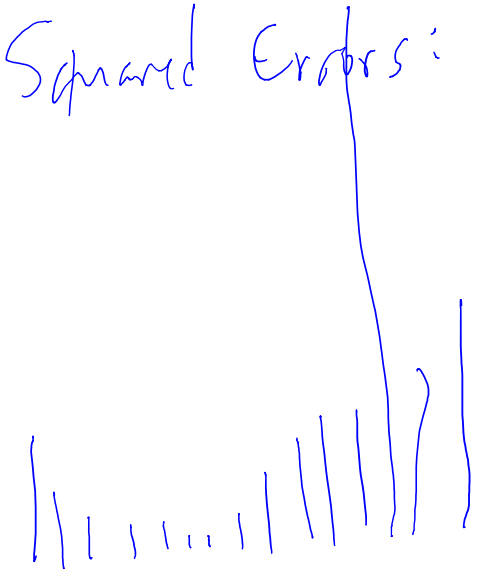


$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \ \frac{1}{2}\|y - Xw\|^2$$

Absolute Errors:

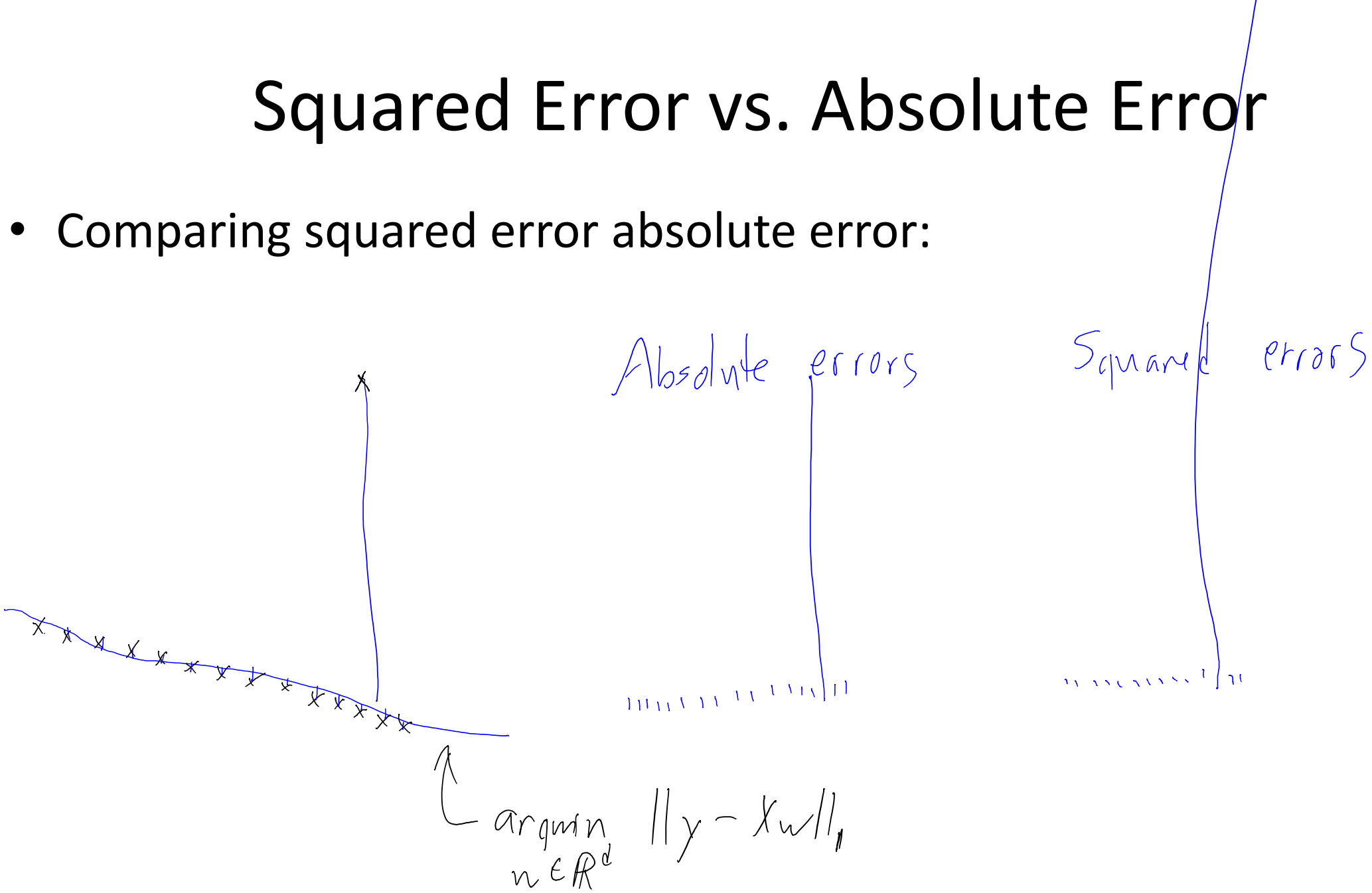


Squared Errors:



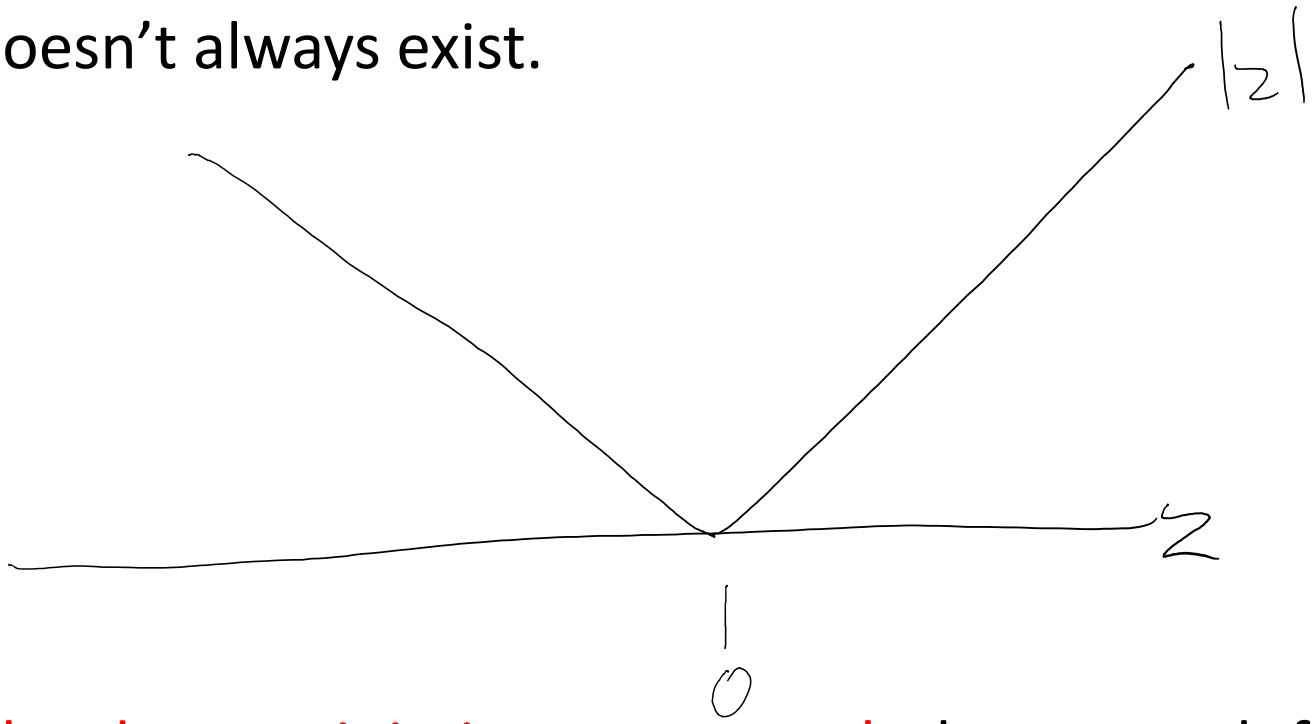
Squared Error vs. Absolute Error

- Comparing squared error absolute error:



Regression with the L1-Norm

- Unfortunately, minimizing the absolute error is harder:
  - Gradient doesn't always exist.



Why not smooth?
- E.g., Huber loss.
- Introduces parameter.
- Smoothing might make the problem harder.

  - Generally, harder to minimize non-smooth than smooth functions.
  - But we can formulate minimize absolute error as a linear program.

Converting into Constrained Problems

- Key observation:
  - Absolute value is maximum of smooth functions: $|w| = \max\{w, -w\}$
- We can convert to minimizing smooth function with constraints:
  1. Replace maximum with new variable, constrained to upper-bound max.
  2. Replace individual constraint with constraint for each element of max.

$$\underset{w\in\mathbb{R}}{\operatorname{argmin}} |w| \iff \underset{w\in\mathbb{R}}{\operatorname{argmin}} \max\{w, -w\} \iff \underset{w\in\mathbb{R}, r\in\mathbb{R}}{\operatorname{argmin}} r \text{ s.t. } r \geq \max\{w, -w\}$$

$$\Updownarrow$$

$$\underset{w\in\mathbb{R}, r\in\mathbb{R}}{\operatorname{argmin}} r \text{ s.t. } r \geq w \text{ and } r \geq -w$$

We must have $r = |w|$ at solution, otherwise constraints are violated or we could decrease 'r'.

Minimizing Absolute Error as Linear Program

- We can apply the same steps to a sum of max functions:

$$\underset{w\in\mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^n |w^Tx_i - y_i| \iff \underset{w\in\mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^n \max\{w^Tx_i - y_i, y_i - w^Tx_i\}$$

$$\iff \underset{w\in\mathbb{R}^d, r\in\mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^n r_i \quad \text{subject to } r_i \geq \max\{w^Tx_i - y_i, y_i - w^Tx_i\} \text{ for all 'i'.}$$

$$\iff \underset{w\in\mathbb{R}^d, r\in\mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^n r_i \quad \text{subject to } r_i \geq w^Tx_i - y_i \text{ and } r_i \geq y_i - w^Tx_i \text{ for all 'i'.}$$

- This is a linear program:
  - Minimizing a linear function subject to linear constraints.
  - We can efficiently solve 'medium-sized' linear programs: Matlab's 'linprog'.
  - There are other linear program formulations of this problems.

‘Brittle’ Regression

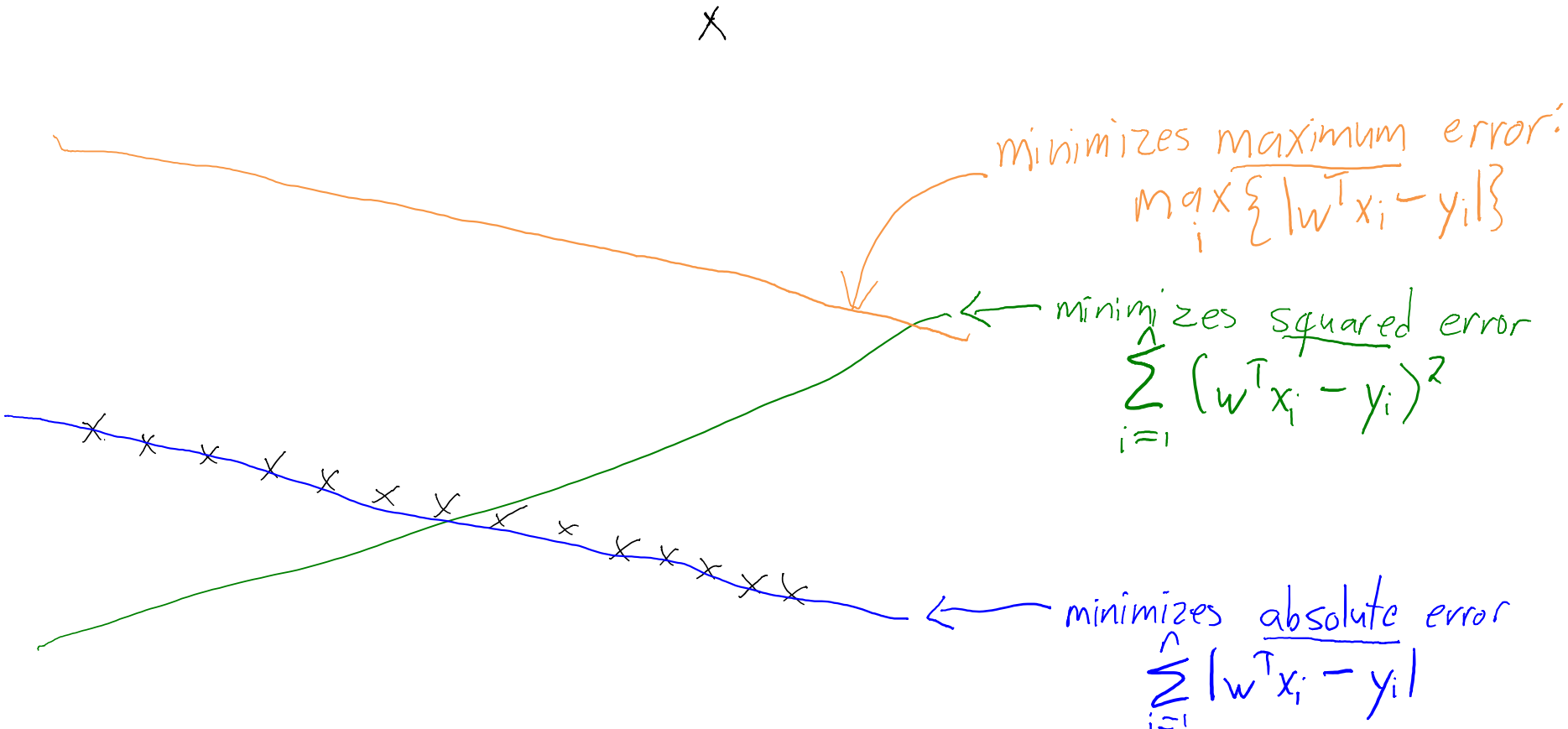
- What if you really care about getting the outliers right?
  - You want best performance on worst training example.
  - For example, if in worst case the plane can crash.
- In this case you can use something like the infinity-norm:

$$\underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \|Xw - y\|_\infty \qquad \|z\|_\infty = \max_i \{|z_i|\}$$

- Very sensitive to outliers (brittle), but worst case will be better.

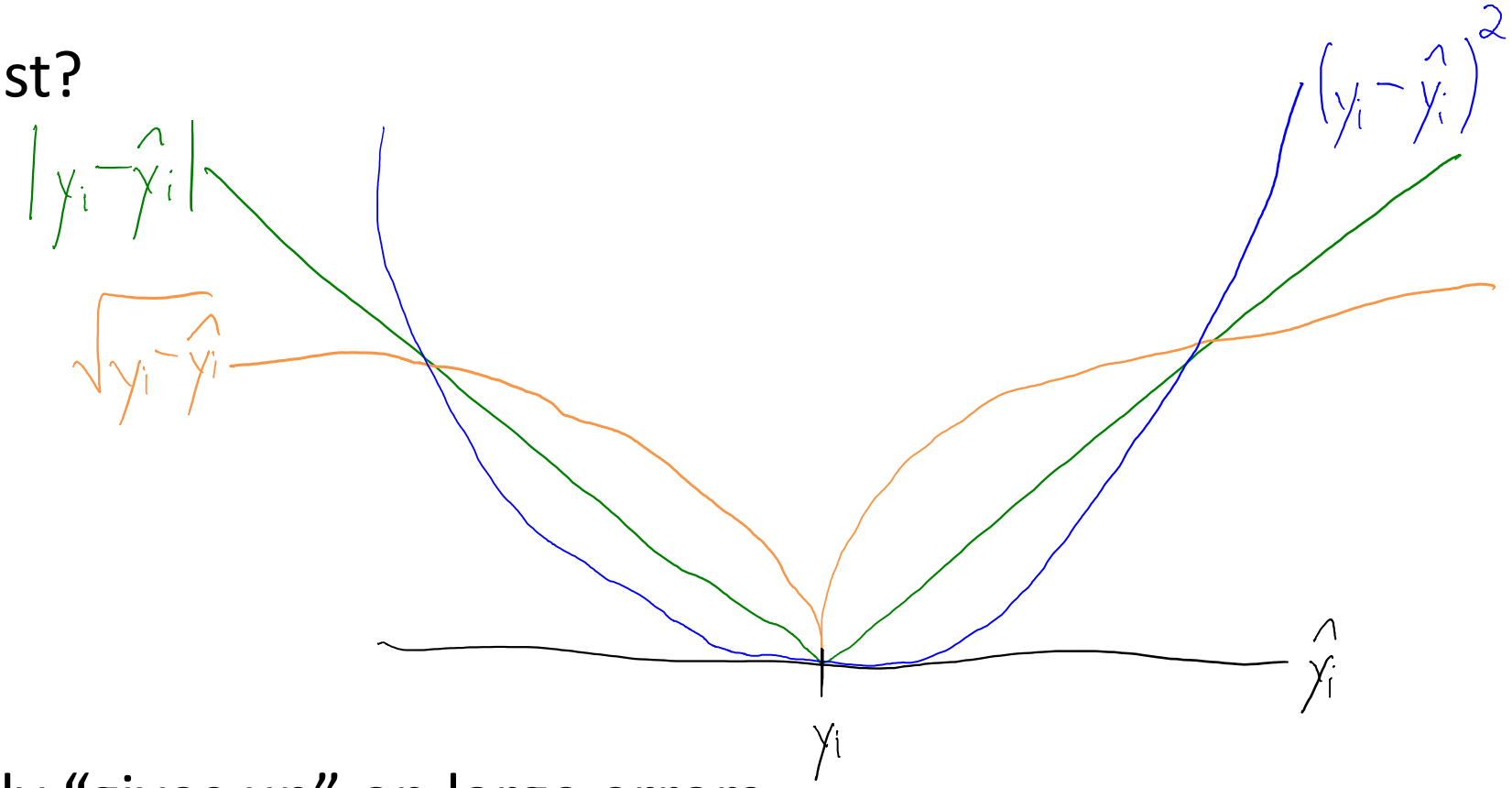
Robust vs. Brittle Regression

- We said that squared error is sensitive to outliers:
  - Absolute error is less sensitive: can be solved as a linear program.
  - Maximum error is more sensitive: can also be solved as linear program.



Very Robust Regression?

- Can we be more robust?



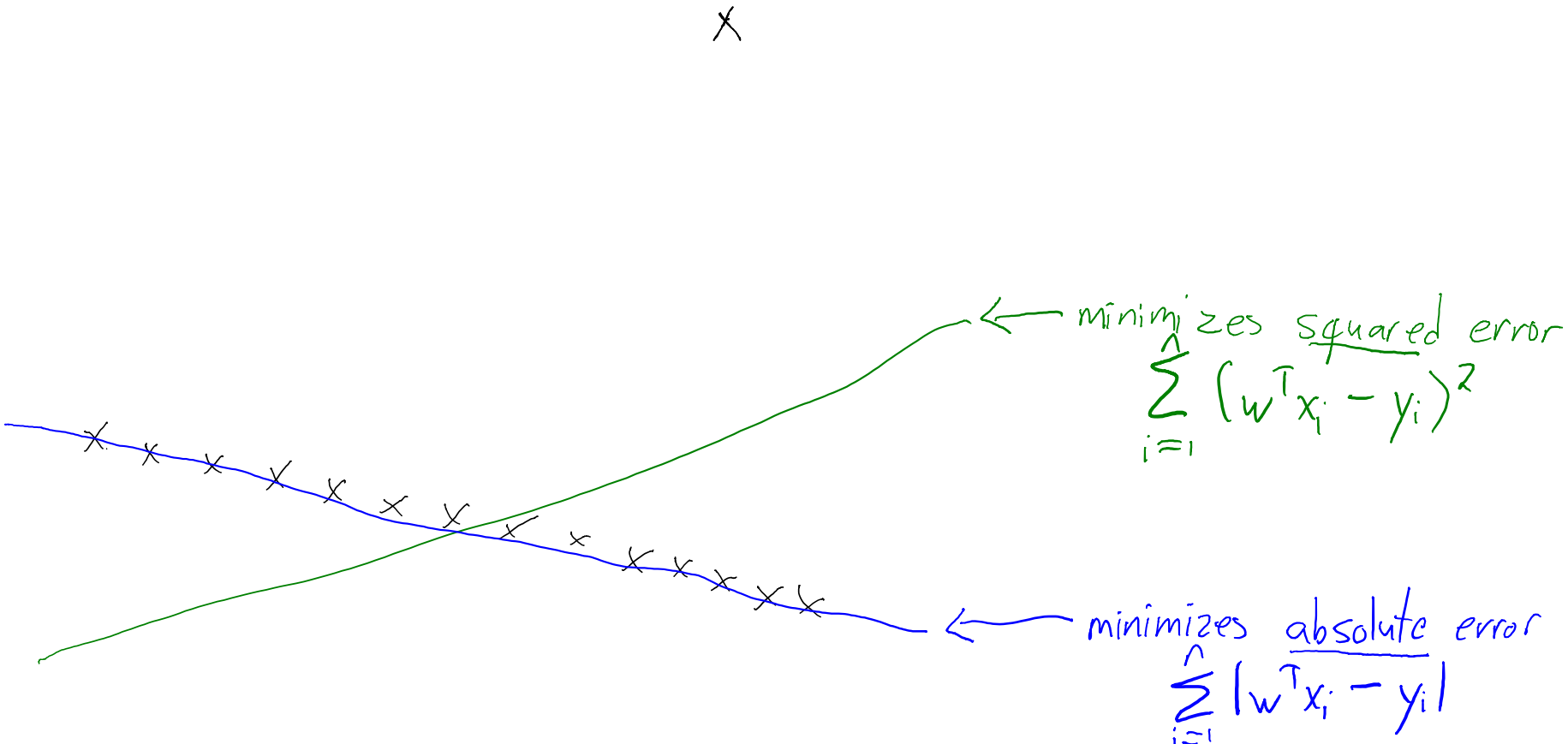
- Very robust: eventually "gives up" on large errors.
- But finding optimal 'w' is NP-hard.
  - Absolute value is the most robust that is not NP-hard.

The ‘Best’ Machine Learning Model

- What is the ‘best’ machine learning model?
  - SVMs? Random forests? Deep learning?
- No free lunch theorem:
  - There is no ‘best’ model that achieves the best test error for every problem.
  - If model A works better than model B on one dataset,
    there is another dataset where model B works better.
- Asking what is the ‘best’ machine learning model is like asking which is ‘best’ among “rock”, “paper”, and “scissors”.
- Caveat of no free lunch (NFL) theorem:
  - The world is very structured, some datasets are more likely than others.
  - Model A could be better than model B on a huge variety of practical applications.
- Machine learning emphasizes models useful across applications.

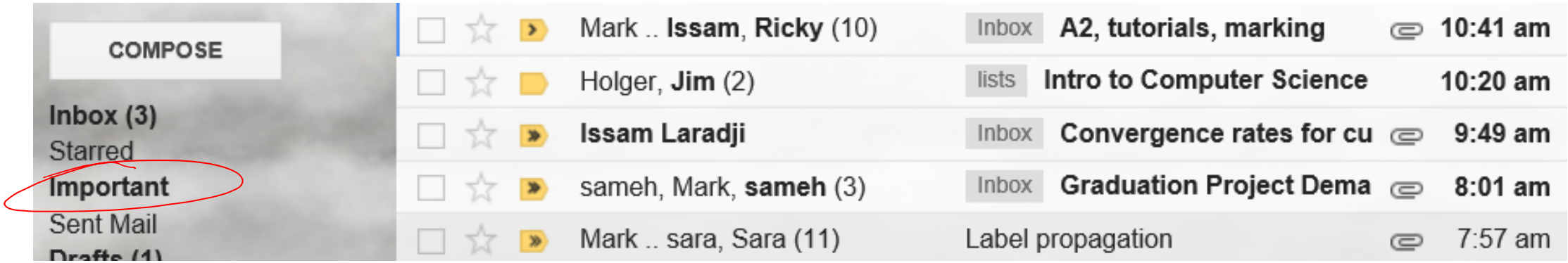
Last Time: Robust Regression

- We said that squared error is sensitive to outliers:
  - Absolute error is less sensitive: can be solved as a linear program.



Motivation: Identifying Important E-mails

- We have a big collection of e-mails:
  - Marked as ‘important’ if user took some action based on them.



- We want to write a program that identifies ‘important’ e-mails?
- Can we formulate as supervised learning?

Supervised Learning Representation for E-mails

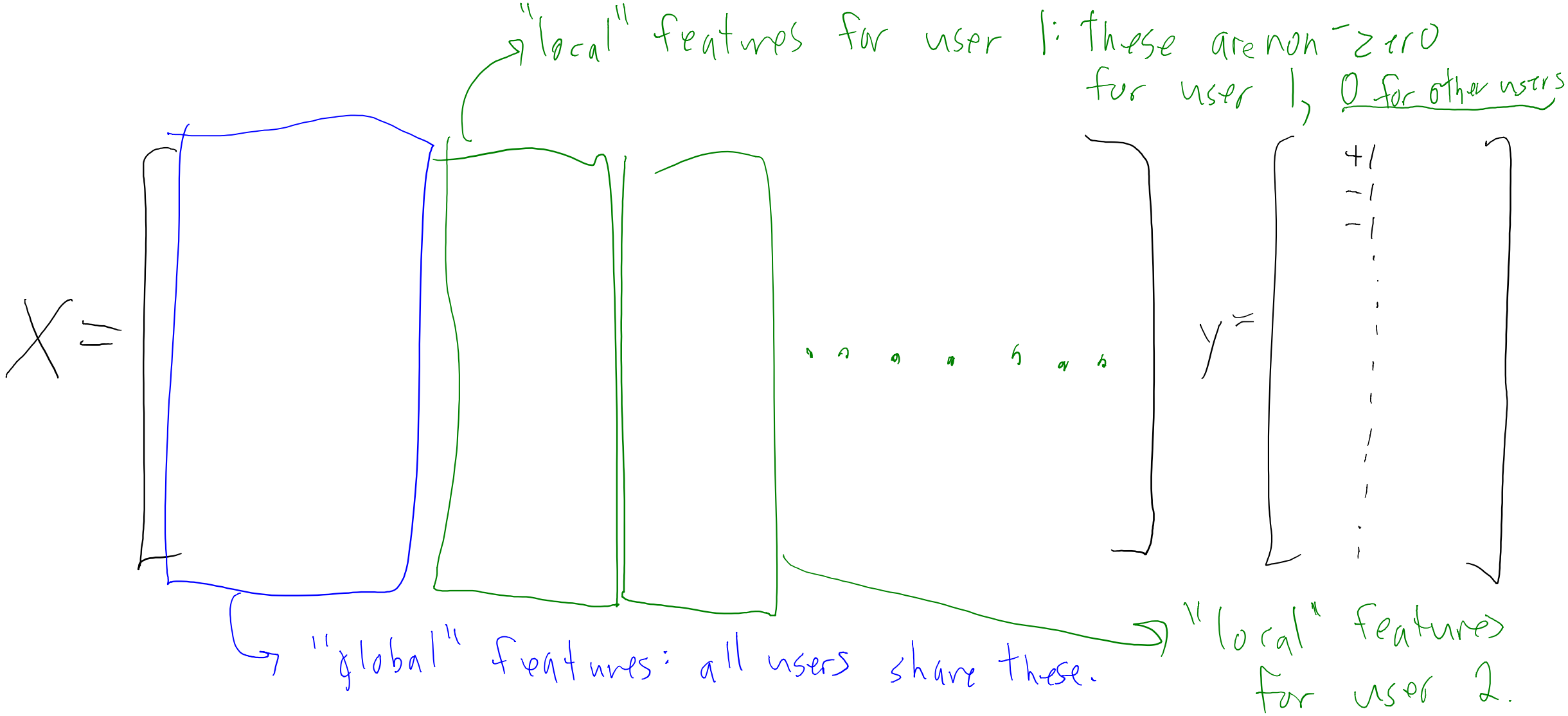
- For e-mail 'i', the target label y_i is binary:
  - +1: "e-mail is important".
  - -1: "e-mail is not important".
  - Classification: supervised learning with discrete labels.
- What are the right features x_i (basis) for e-mails?
  - Use bag of words:
    - "CPSC", "Expedia", "vicodin".
    - Binary "Expedia" feature is 1 if phrase "Expedia" is in the message, and 0 otherwise.
  - Could add phrases:
    - "you're a winner", "CPSC 540".
  - Could add regular expressions:
    - <recipient name>, <sender domain == "mail.com">

Supervised Learning Representation for E-mails

$$X = \begin{array}{c} \begin{array}{cccc} \text{"CPSC"} & \text{"Expedia"} & \text{"Vicodin"} & \langle\text{recipient name}\rangle \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \end{array} \qquad y = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \\ +1 \end{bmatrix}$$

- Can we make personalized predictions?
  - Some messages 'universally' important:
    - "This is your mother, something terrible happened, give me a call ASAP."
  - Some messages may be important to one user but not others.

The Big Global/Local Feature Table



Predicting Importance of E-mail For New User

- Consider a new user:
  - Start out with no information about them.
  - Use global features to predict what is important to generic user.

$\hat{y}_i = \text{sign}(w_g^T x_g)$ → features/parameters shared across all users.

- With more data, update global features and user's local features:
  - Local features make prediction *personalized*.

$\hat{y}_i = \text{sign}(w_g^T x_g + w_u^T x_u)$ → features/parameters specific to user "u".

- G-mails system: classification with logistic regression.

Classification Using Regression?

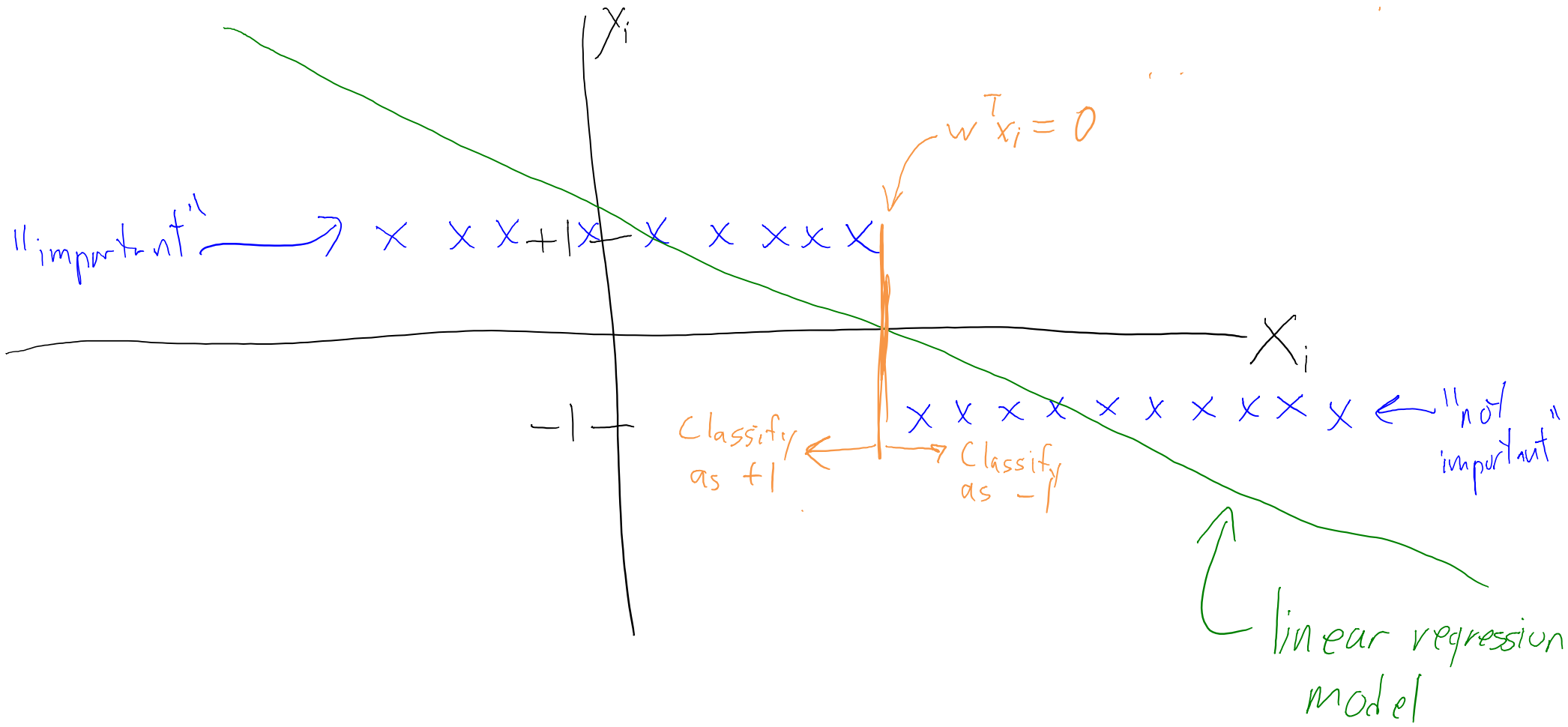
- Usual approach to do binary classification with regression:
  - Code y_i as '+1' for one class and '-1' for the other class.
- Fit a linear regression model:

$$\hat{f}_i = w_1 x_{i1} + w_2 x_{i2} + \cdots + w_d x_{id}$$
$$= w^T x_i$$

- Classify by take the sign (i.e., closer '-1' or '+1'?):

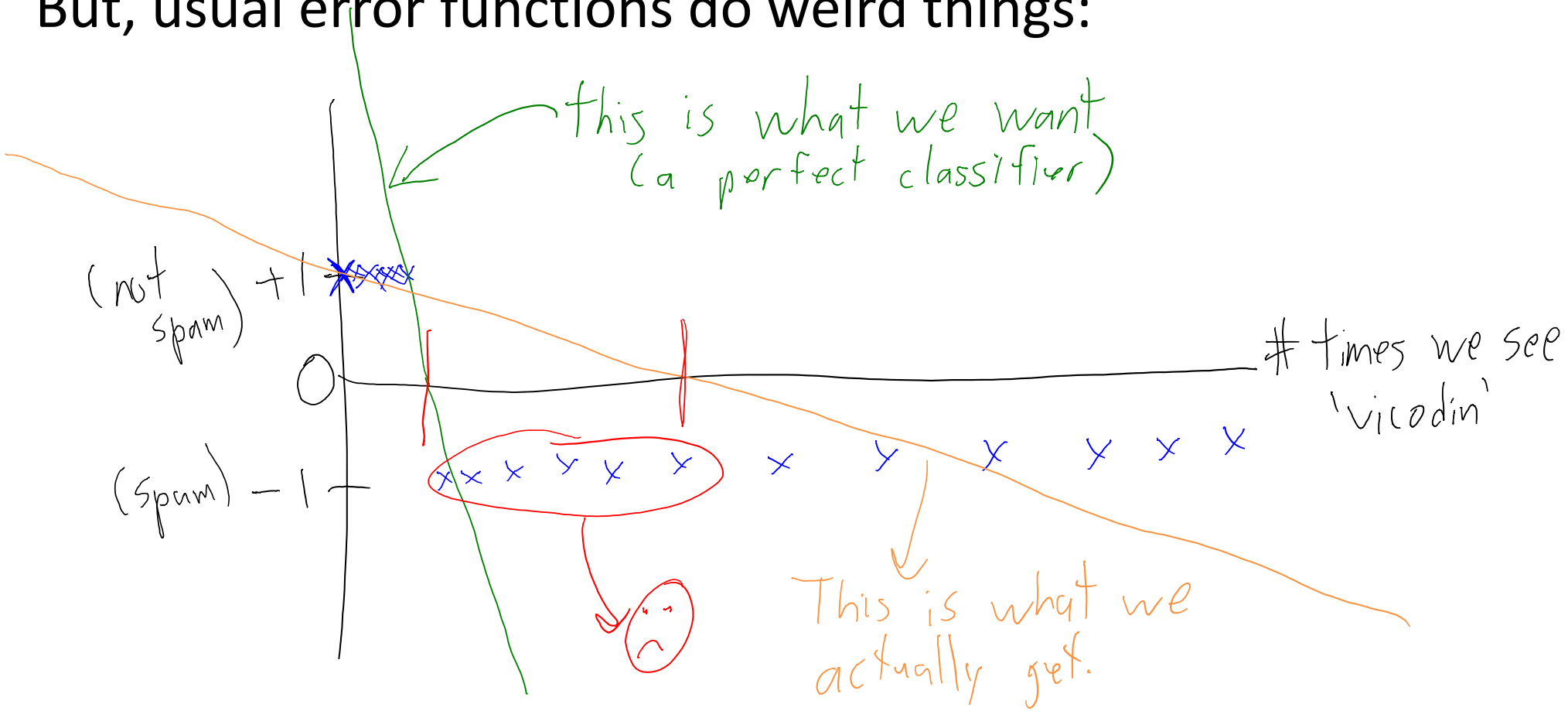
$$\hat{y}_i = \text{sign}(w^T x_i).$$

Classification using Regression



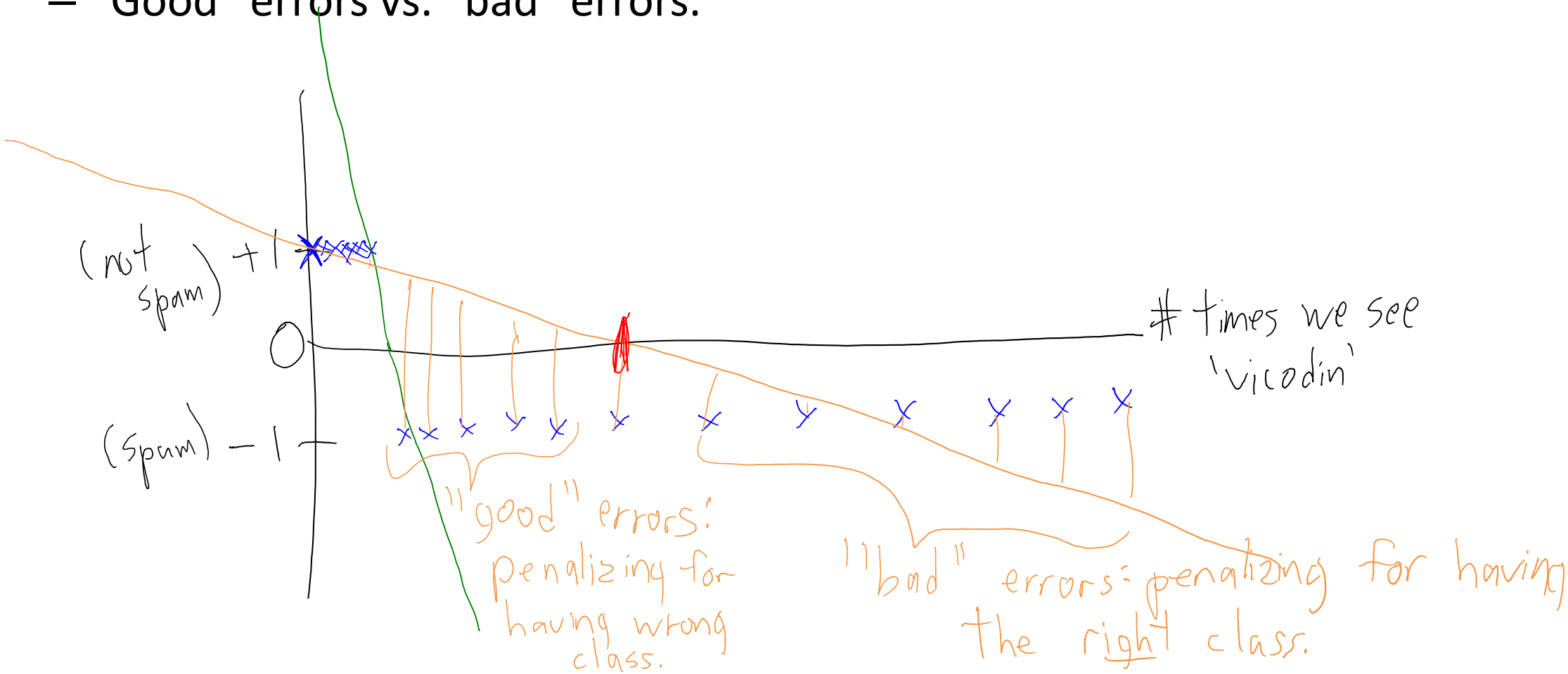
Classification using Regression

- Can use our tricks (e.g., RBF basis, regularization) for classification.
- But, usual error functions do weird things:



Classification Using Regression

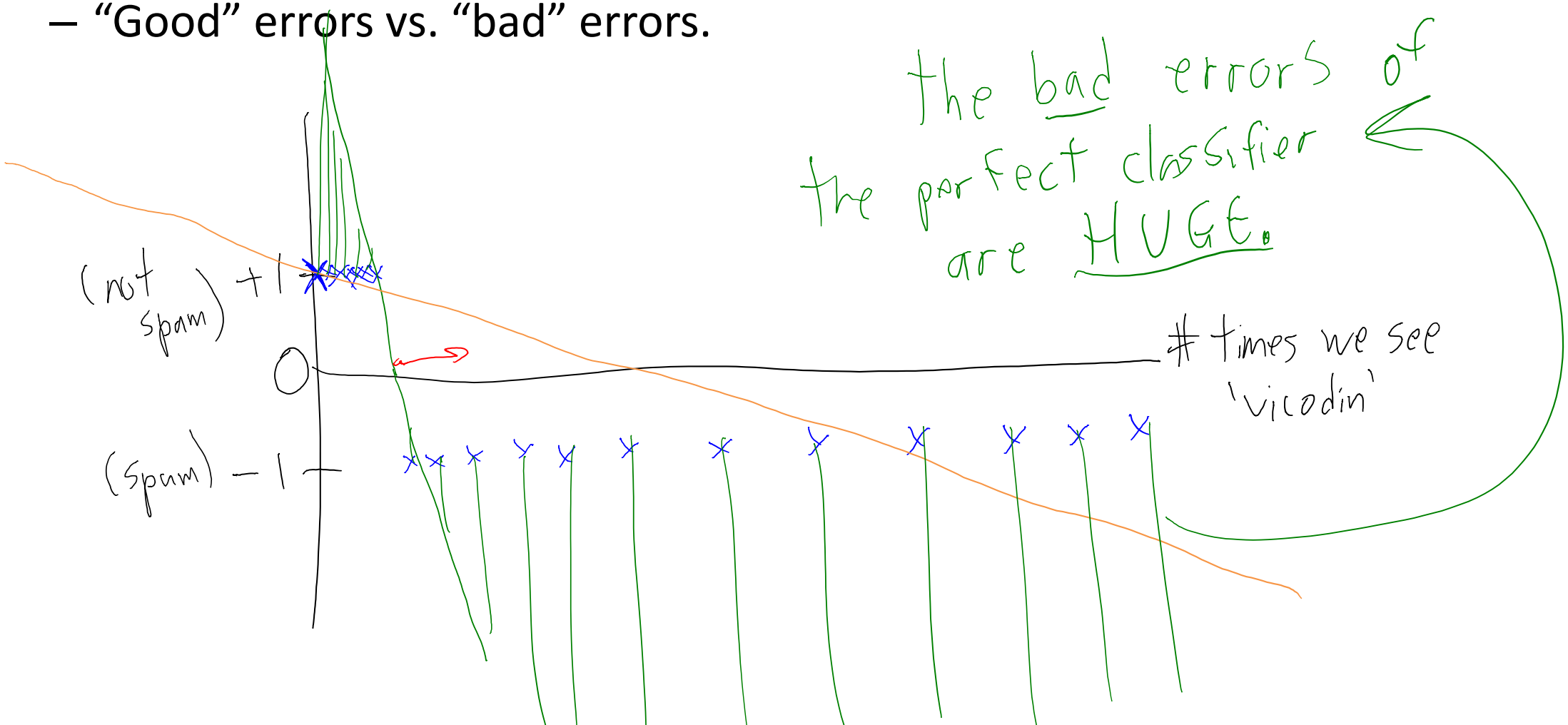
- What went wrong?
  - "Good" errors vs. "bad" errors.



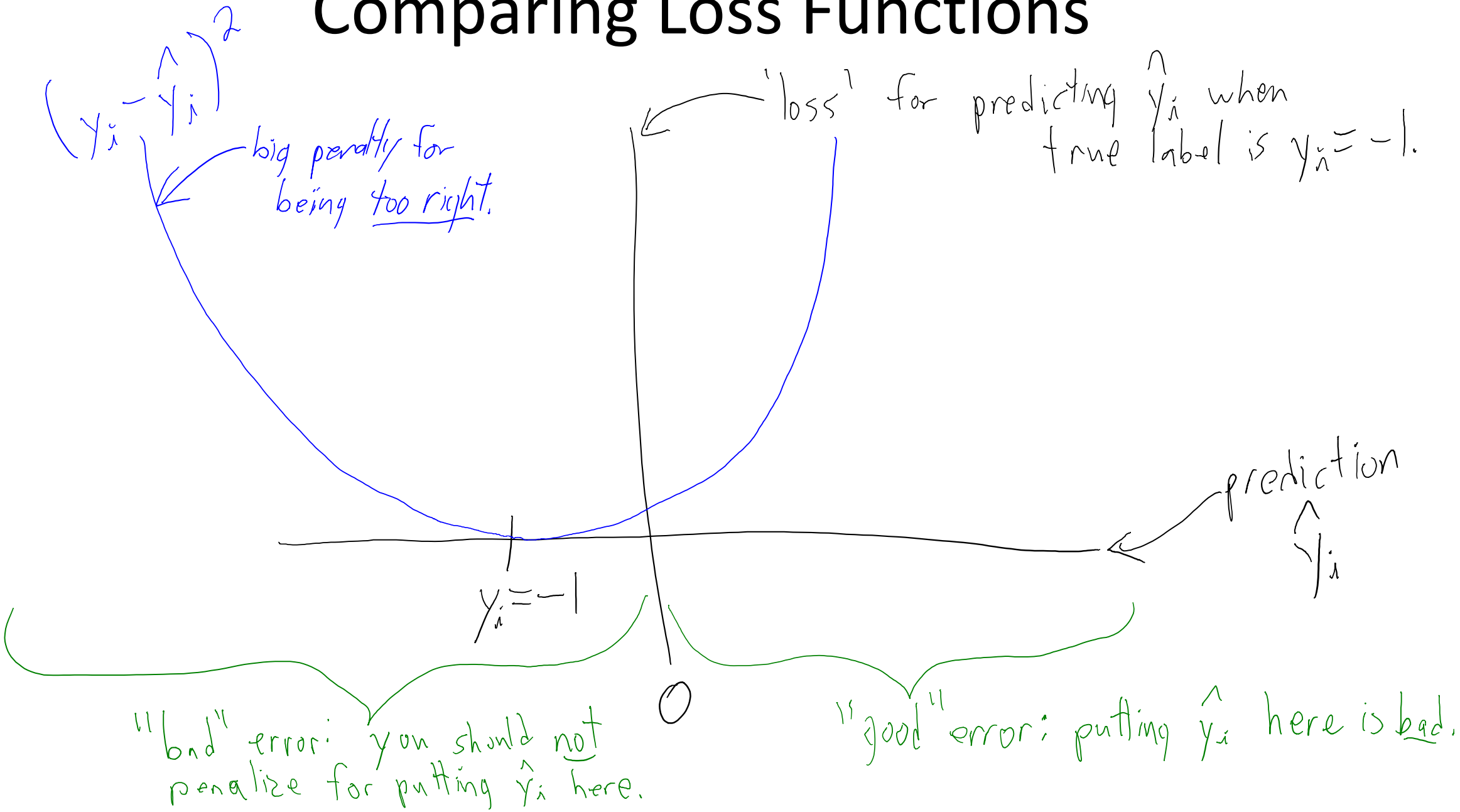
Classification Using Regression

Squared error vs. sign

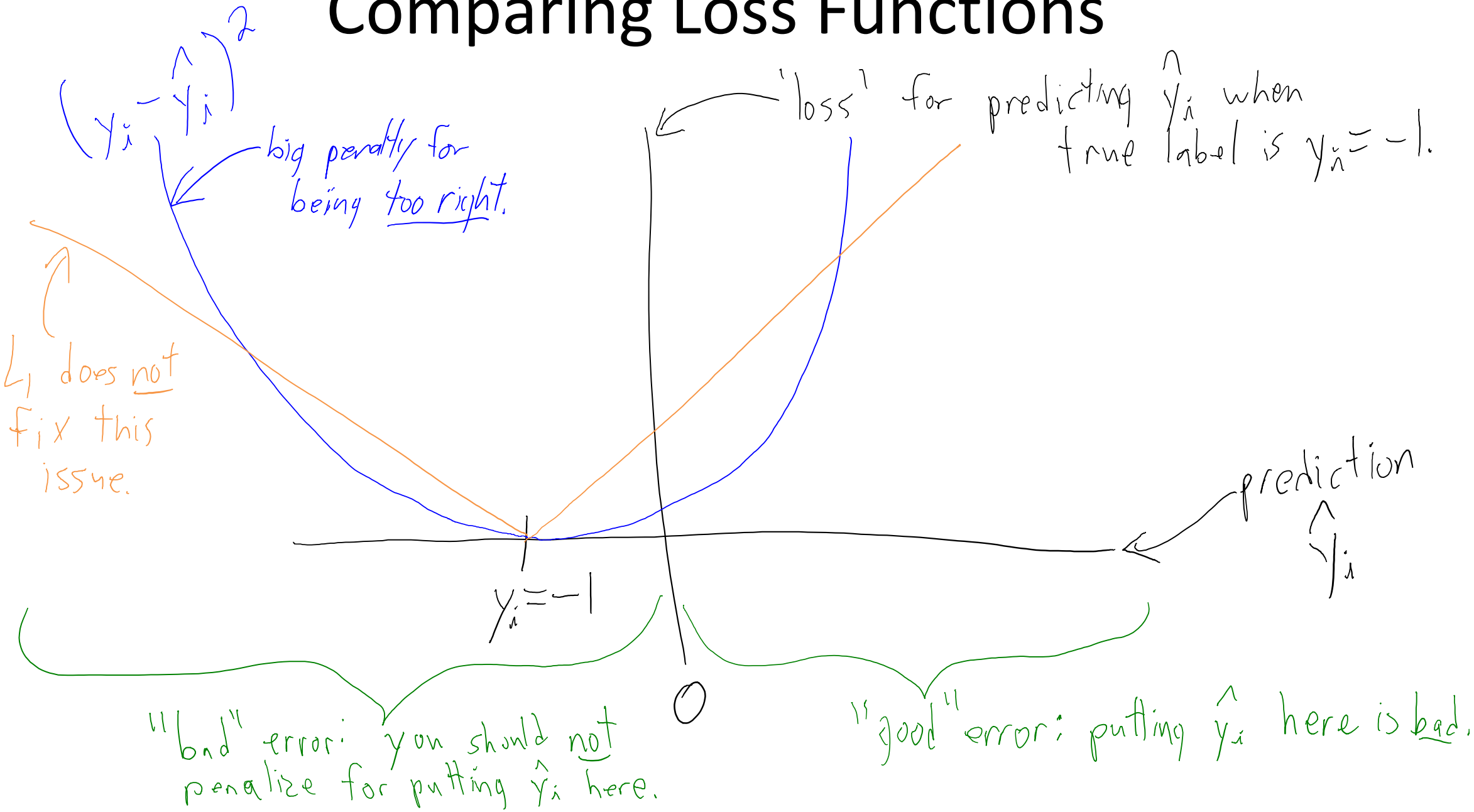
- What went wrong?
  - "Good" errors vs. "bad" errors.



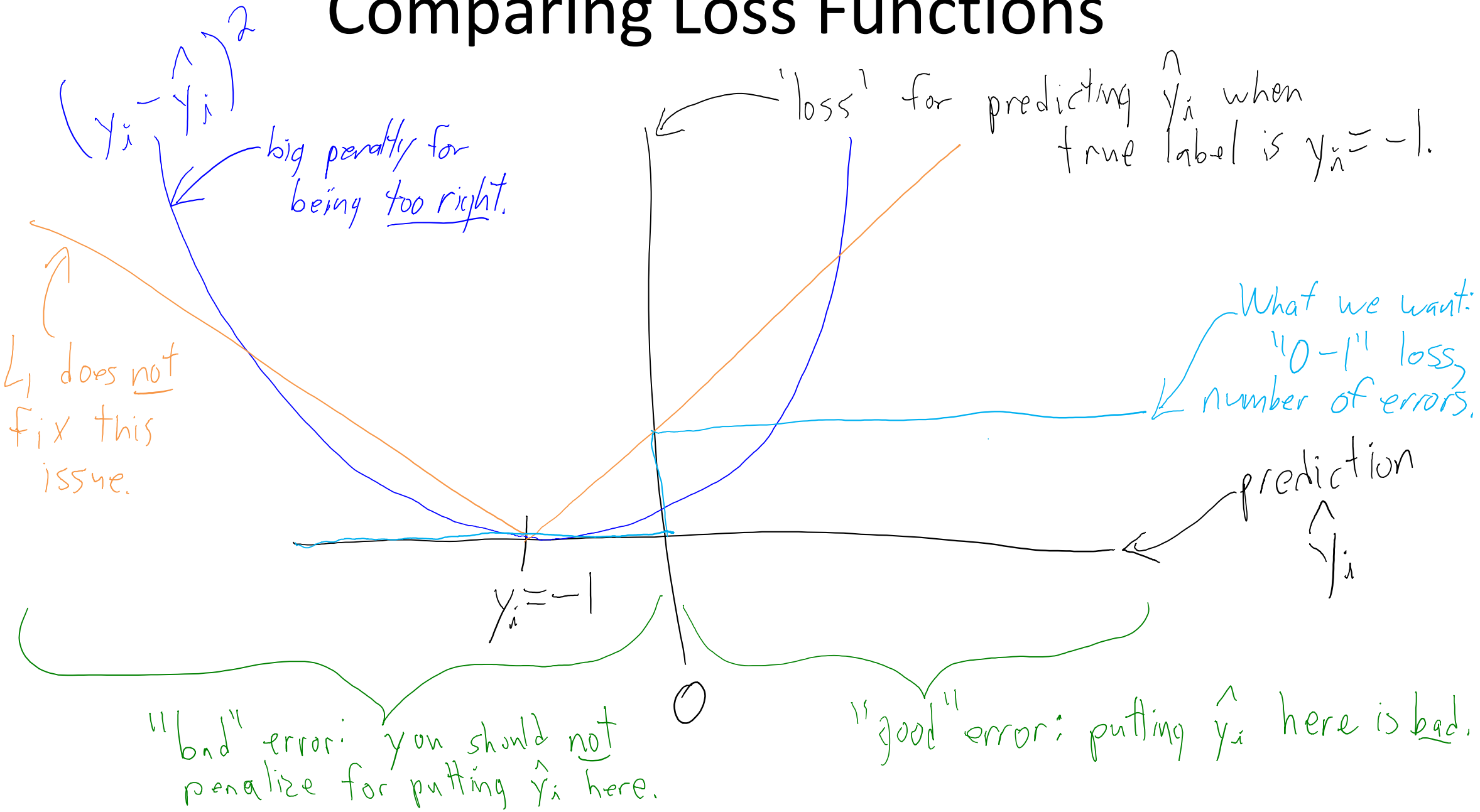
Comparing Loss Functions



Comparing Loss Functions



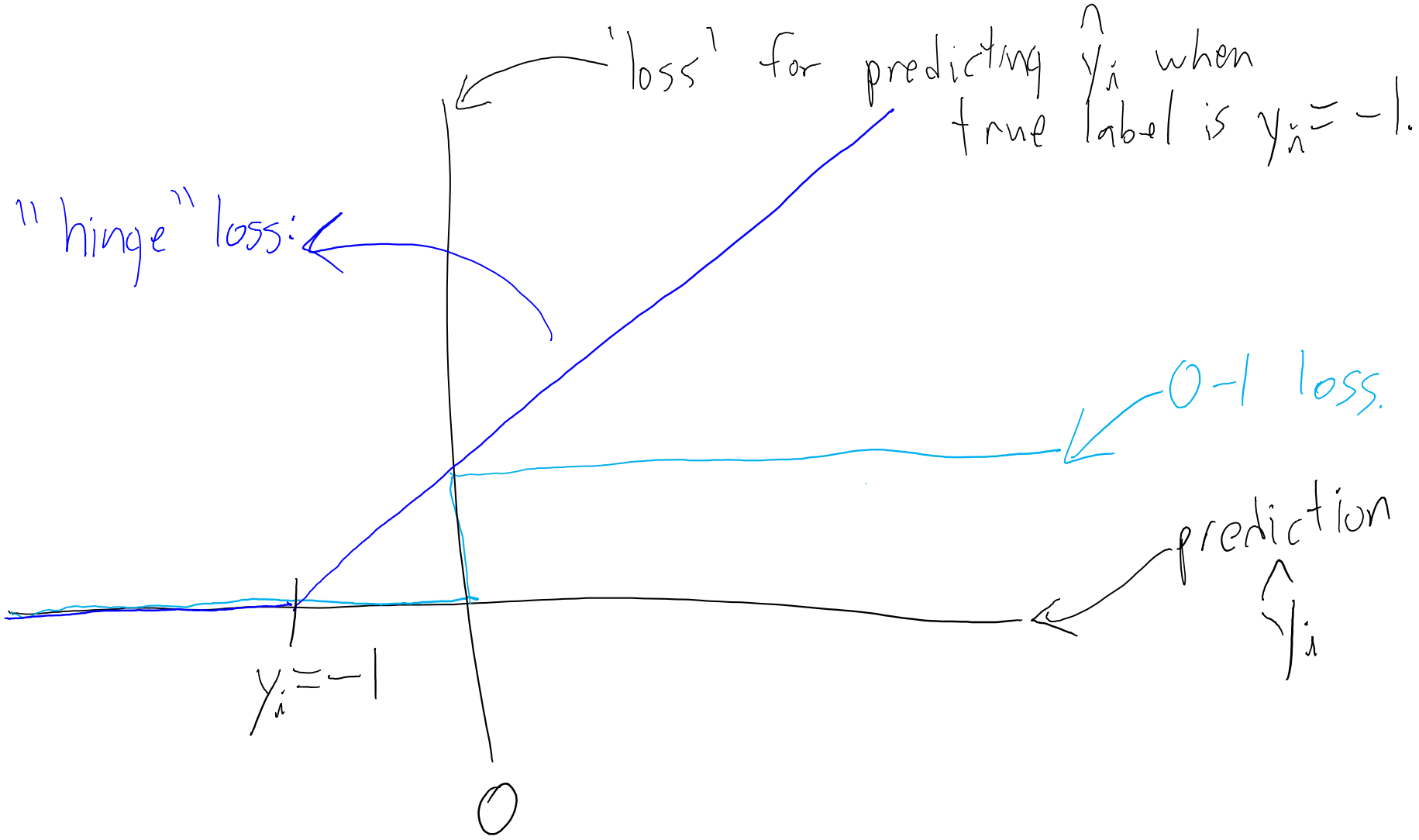
Comparing Loss Functions



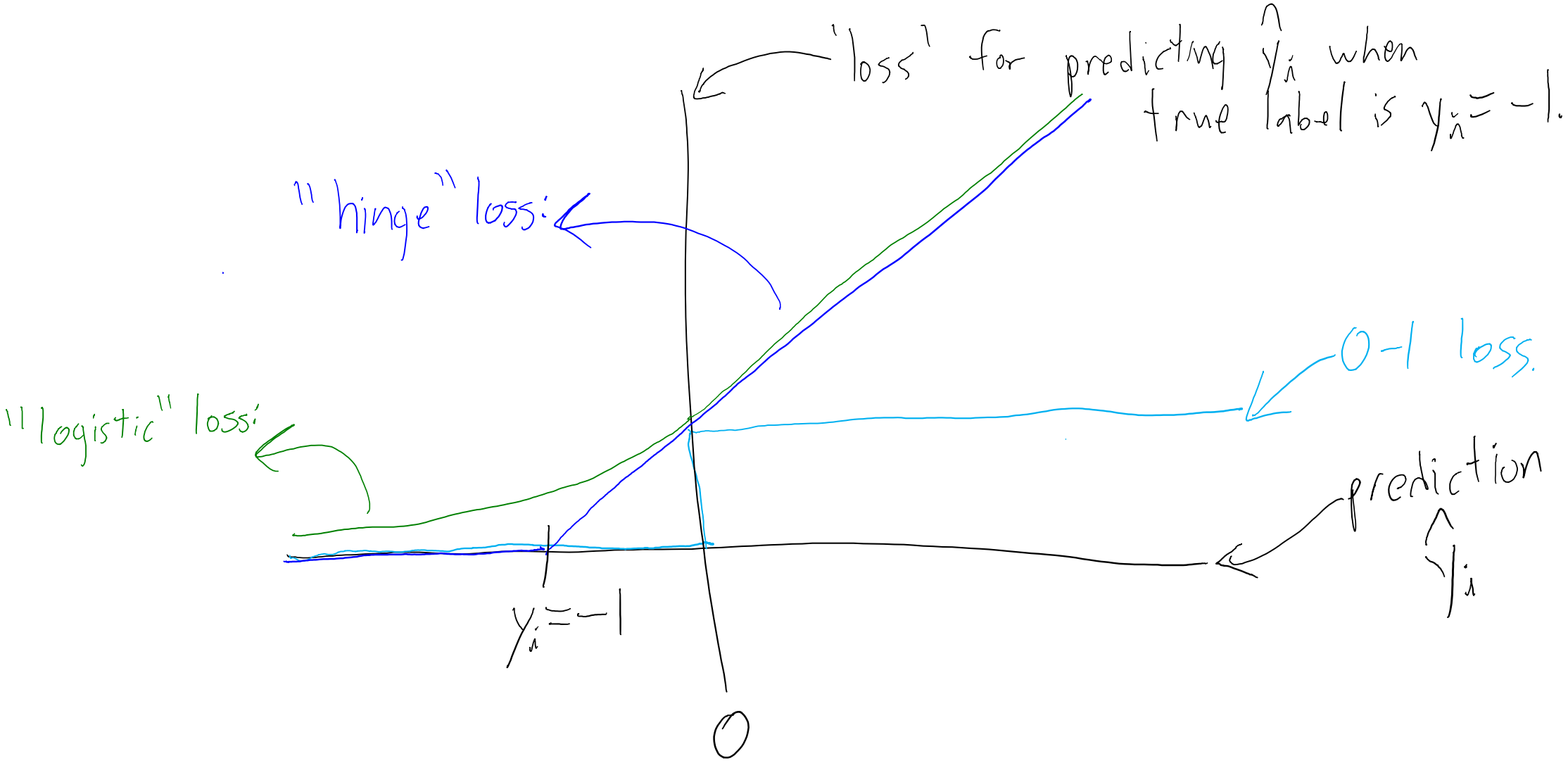
0-1 Loss Function and Tractable Approximations

- The 0-1 loss function is the number of errors after taking the sign.
  - If a perfect classifier exists, you can find one as a linear program.
  - Otherwise, it's NP-hard to minimize 0-1 loss:
    - We do not expect that efficient algorithms exist.
- Tractable alternatives to 0-1 loss:
  - Hinge loss: upper-bound on 0-1 loss that can be written as linear program.
  - Logistic loss: differentiable function similar to hinge loss.

0-1 Loss Function and Tractable Approximations



0-1 Loss Function and Tractable Approximations



Hinge Loss and Support Vector Machines

- Hinge loss is given by:

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} \max\{0, 1 - y_i w^T x_i\}$$

  - Can be written as a linear program using our max trick.
  - Solution will be a perfect classifier, if one exists.
- Support vector machine (SVM) is hinge loss with L2-regularization.

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} \max\{0, 1 - y_i w^T x_i\} + \frac{\lambda}{2}\|w\|^2$$

  - Can be written as a quadratic program using our max trick
    - Quadratic objective with linear constraints.
  - Solution will be perfect classifier, if one exists and λ is small enough.
  - Maximizes margin: maximizes distance of data to decision boundary.

Logistic Regression

- Logistic regression minimizes logistic loss:

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$$

- You can/should also add regularization:

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i)) + \frac{\lambda}{2}\|w\|^2$$

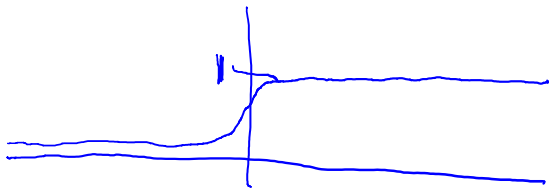
- These can't be written as linear/quadratic programs:
  - But they're differentiable: we'll discuss how to solve them next time.

Logistic Regression and SVMs

- SVMs and logistic regression are used EVERYWHERE!
- Why?
  - Training and testing are both fast, even for "large-scale" problems.
  - It is easy to understand what the weights 'w_j' mean.
  - With high-dimensional features and regularization, often good test error.
  - Otherwise, often good test error with RBF basis and regularization.
  - For logistic regression, predictions have probabilistic interpretation.

If $p(y_i = +1 \mid w, x_i) = \text{sigm}(w^T x_i)$ then minimizing logistic loss corresponds to <u>maximum likelihood estimate</u>.

$\text{sigm}(z) = \frac{1}{1+\exp(-z)}$



Discussion: Probabilistic Interpretation

- Why is probabilistic interpretation important?
  - We can return a probabilistic prediction:

    Instead of $\hat{y}_i = 1$, say that $p(\hat{y}_i = 1 \mid w, x_i) = 99\%$

    or $p(\hat{y}_i = 1 \mid w, x_i) = 51\%$
  - For complicated y_i, it may be easier to define probability than loss.
  - We can talk about maximizing utility:

Predict / True	True 'spam'	True 'not spam'
Predict 'spam'	TP: 0	FP: 100
Predict 'not spam'	FN: 10	TN: 0

$$E[C(\hat{y}_i = \text{spam})] = p(y_i = \text{spam} \mid x_i)\, C(\hat{y}_i = \text{spam}, y_i = \text{spam}) + p(y_i = \text{not spam} \mid x_i)\, C(\hat{y}_i = \text{spam}, y_i = \text{not spam})$$

Predict "not spam" even if $\hat{y}_i =$ "spam" if expected cost of "not spam" is lower.

Maximum Likelihood and MAP Estimation

- Unregularized logistic regression is maximum likelihood solution:
  - Maximize likelihood of data given model parameters.
  - Problem with maximum likelihood:
    - data could be very likely in some very unlikely model from family.
    - E.g., complex model overfits by memorizing the data.
- Regularized logistic regression is MAP (maximum a posteriori):

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmax}}\ p(w \mid y, X) \iff \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} -\log(p(y_i \mid w, x_i)) - \log(p(w))$$

  - Model is a random variable, and we need to find most likely model.
  - Can take into account that complex models are likely to overfit.

Multi-Class Logistic Regression

- Supposed y_i takes values from an unordered discrete set of classes.

$x_i \in \mathbb{R}^d$ $\qquad$ $y_i \in \{1, 2, 3, 4, \ldots, k\}$

"itchy" (class 1)

"scratchy" (class 2)

"poochie" (class 3)

- Standard model:
  - Use a 'd'-dimensional weight vector 'w_c' for each class 'c'.
  - Try to make inner-product $w_c^T x_i$ big when 'c' is the true label 'y_i'.
  - Classify by finding largest inner-product: $\hat{y}_i = \underset{c}{\operatorname{argmax}} \{ w_c^T x_i \}$

(Also exist models for ordered classes or count data)

http://simpsons.wikia.com/wiki/Simpsons_Wiki

Multi-Class Logistic Regression

We have a <u>parameter matrix</u> $W = \underbrace{\begin{bmatrix} | & | & | & & | \\ w_1 & w_2 & w_3 & \cdots & w_k \\ | & | & | & & | \end{bmatrix}}_{k}$ $\Big\}\, d$

To make a <u>prediction</u>, compute

$$W^T x_i = \begin{bmatrix} w_1^T x_i \\ w_2^T x_i \\ \vdots \\ w_k^T x_i \end{bmatrix}$$

predict "2"

and compute <u>maximum</u>.

We want a <u>loss function</u> that will make $w_c^T x_i$ <u>big</u> when c is the true label y_i and will otherwise make $w_c^T x_i$ small.

We can define probability using softmax function:

$$p(y_i = c \mid W, x_i) = \frac{\exp(w_c^T x_i)}{\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)} \propto \exp(w_c^T x_i)$$

If $k=3$:

$$p(y_i \mid W, x_i) = \frac{\exp(w_{y_i}^T x_i)}{\exp(w_1^T x_i) + \exp(w_2^T x_i) + \exp(w_3^T x_i)}$$

$\uparrow$ true

To fit model, use

$$-\log p(y_i \mid W, x_i) = -w_{y_i}^T x_i + \log\left(\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)\right)$$

Generalizes sigmoid:

- special case of $k=2$ and $w_2 = 0$

Course Roadmap

- Part 1: Overview of Machine Learning
  - Linear models: change of basis, regularization, loss functions.
  - Basics of learning theory: Training vs. test error, bias-variance, fundamental trade-off, no free lunch.
  - Probabilistic learning principles: Maximum likelihood, MAP estimation, loss functions.
- Part 2: Large-scale machine learning.
  - Why are SVMs/logistic easy while minimizing number of errors is hard?
  - How do we fit these models to huge datasets?

Summary

- Regularization: allows complicated models by penalizing complexity.
- Radial basis functions: non-parametric universal basis.
- Robust regression models: more suitable when we have outliers.
- Converting non-smooth problems to constrained smooth problems.
- No free lunch: there is no 'best' machine learning model.
- SVMs and logistic regression: more suitable losses for classification.
- MLE and MAP: probabilistic interpretation to losses/regularizers.
- Softmax loss to model discrete yi, other losses can be derived.

  – Next time: Why is logistic easy while minimizing number of errors is hard?