Massively Shallow Learning with FFX

Trent McConaghy, PhD

ascribe

solido
DESIGN AUTOMATION

Mysteries of the universe..

What does AI encompass?

WTF is genetic programming or symbolic regression? Why should I care?

Is Deep Learning cool or what?

How *does* Google find furry robots?

What *is* technology anyway?

Technology

Technology


The Exciting New F² ("Fork Fan")
Designed by World Renown Entrepeneur: Rod Ryan
Cools down all those "too hot" to eat foods before they get to your mouth!
Never burn your tounge again!
Go ahead, be in a hurry. Never wait for your food to cool down ever again.
Featuring:
* High Tech Ergonomic Design
* Two Speed "Whisper Quiet" Fan
* Right and Left Handed Compatible
* Stainless Steel Anti-Corrosion Materials
* Dishwasher Safe!
"This is the BEST new kitchen innovation I have ever seen! Ideal for prison food!" Martha Stewart
Worth1000.com
CE

Technology

Technology – Alternate Definition

"We can say that solving least-squares problems … is a (mature) *technology*, that can be reliably used by many people who do not know, and do not need to know, the details."

- Boyd and Vandenberghe, Convex Optimization, 2004

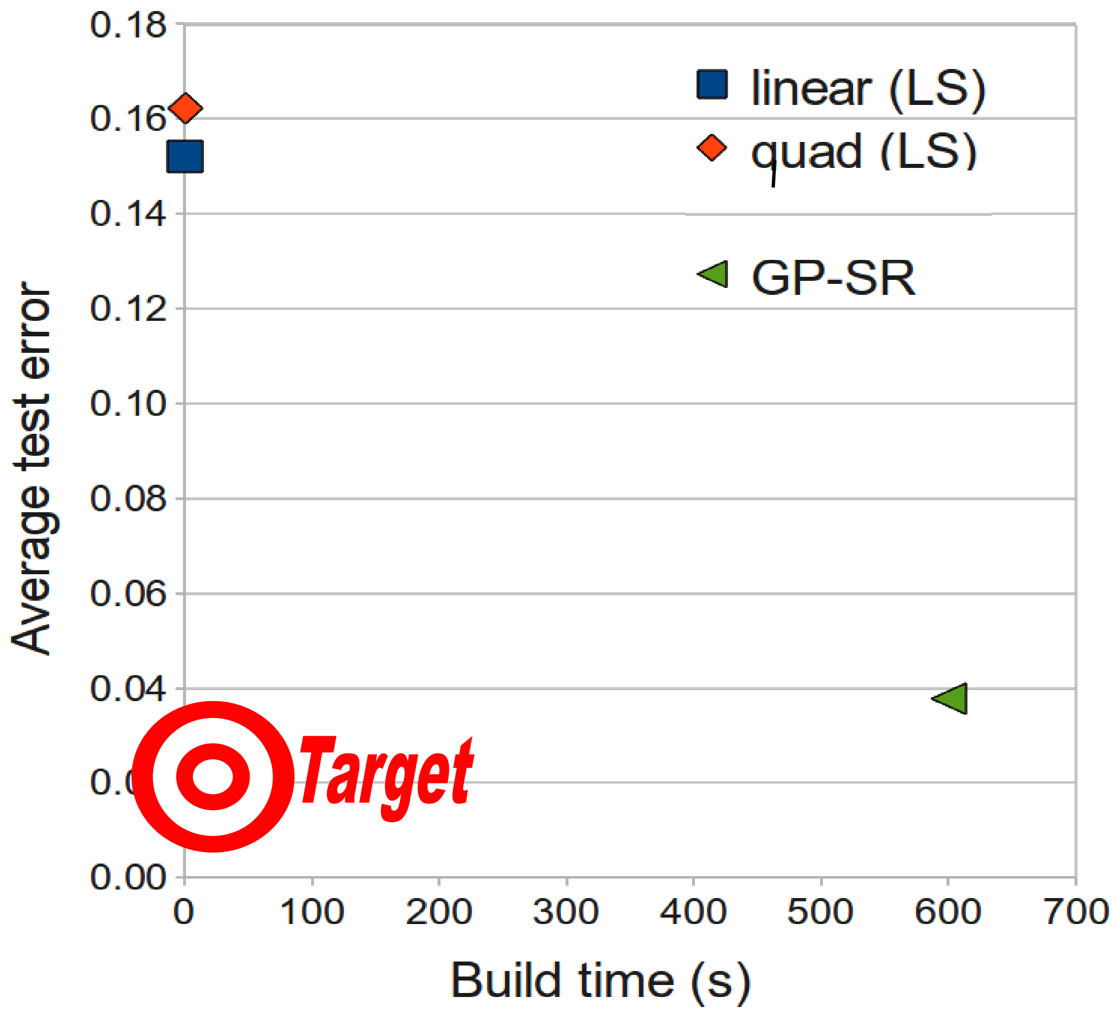
On becoming a “tool”

- Long time standard tools: LS regression, matrix inversion, FFT, SQP, SAT solvers, CLP, …
- Recent standard tool: convex optimization – became popular in the late 90s. “It just works.”
- GP was popularized in the early 90s
  - And is not a standard tool (for many reasons)
- Deep learning became popular in the 10s
  - And is not standard tool (for many reasons)

Summary:
Aiming for SR* as a Technology

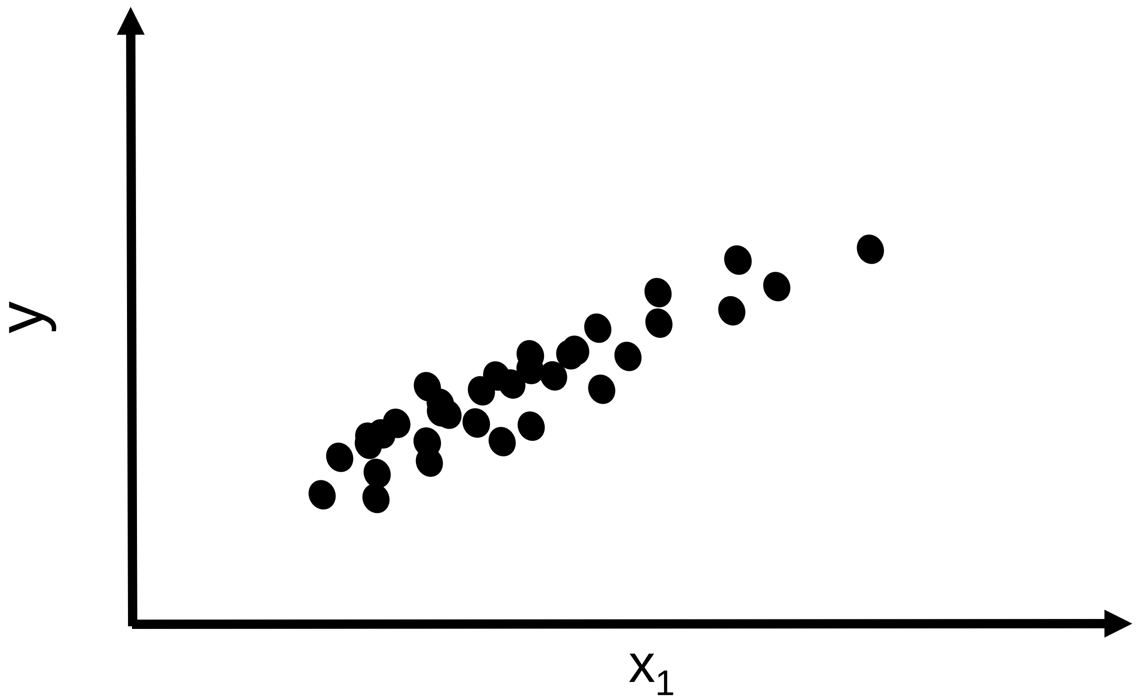
*** SR ≠ Shopping Robot**

Summary of Goal
Speed of LS, Accuracy of GP-SR (CAFFEINE)



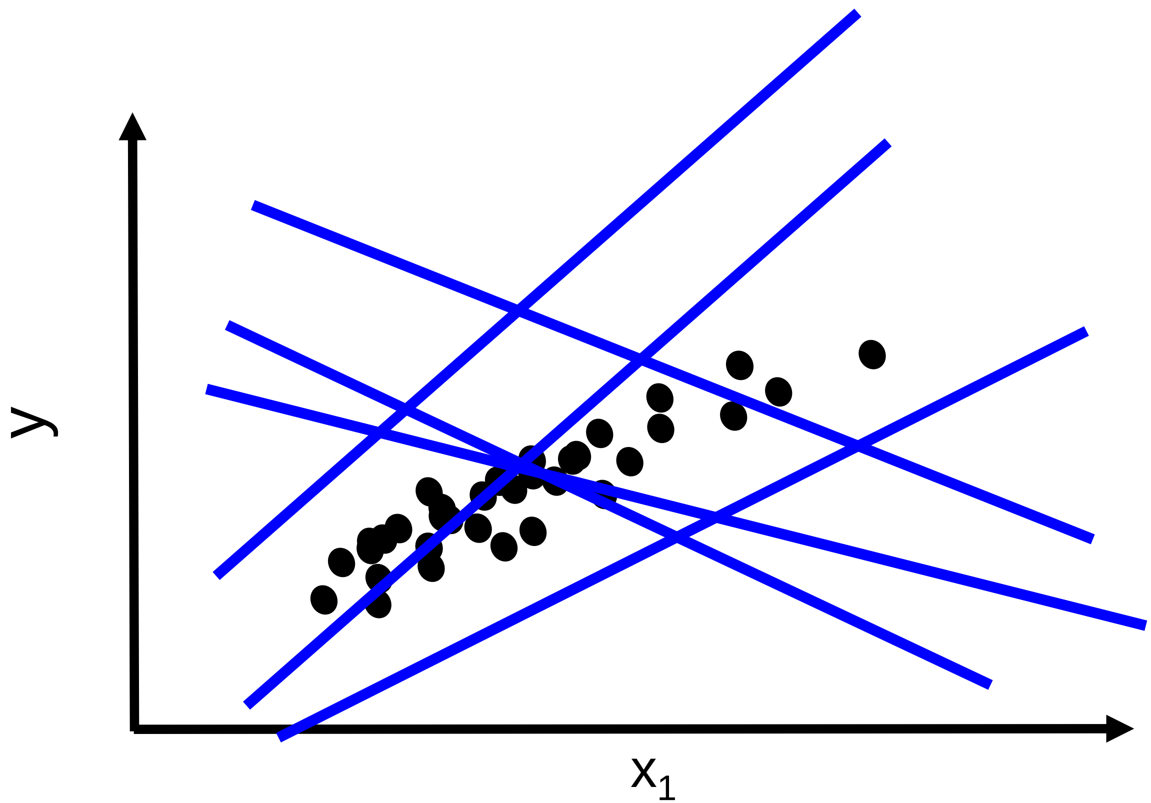
A (Re) Introduction to Regression

1D Linear Least-Squares Regression



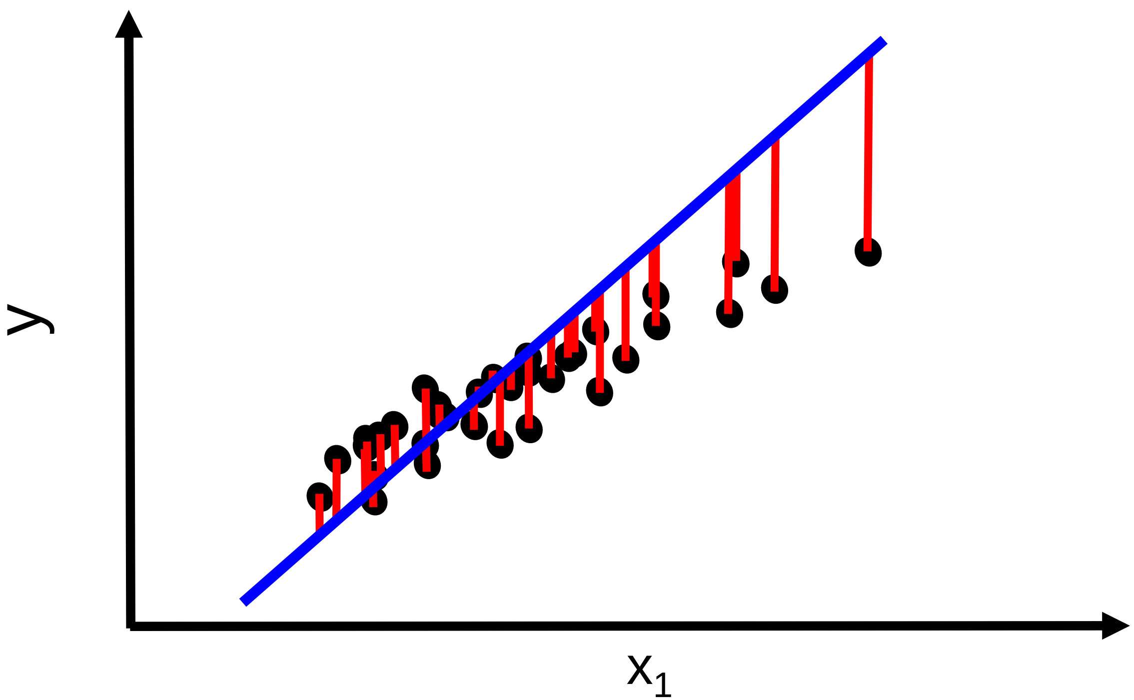
1D Linear LS Regression

Many possible linear models!



1D Linear LS Regression

Find linear model that
minimizes $\sum(\text{yhat}_i - y_i)^2$
for all *i* in training data



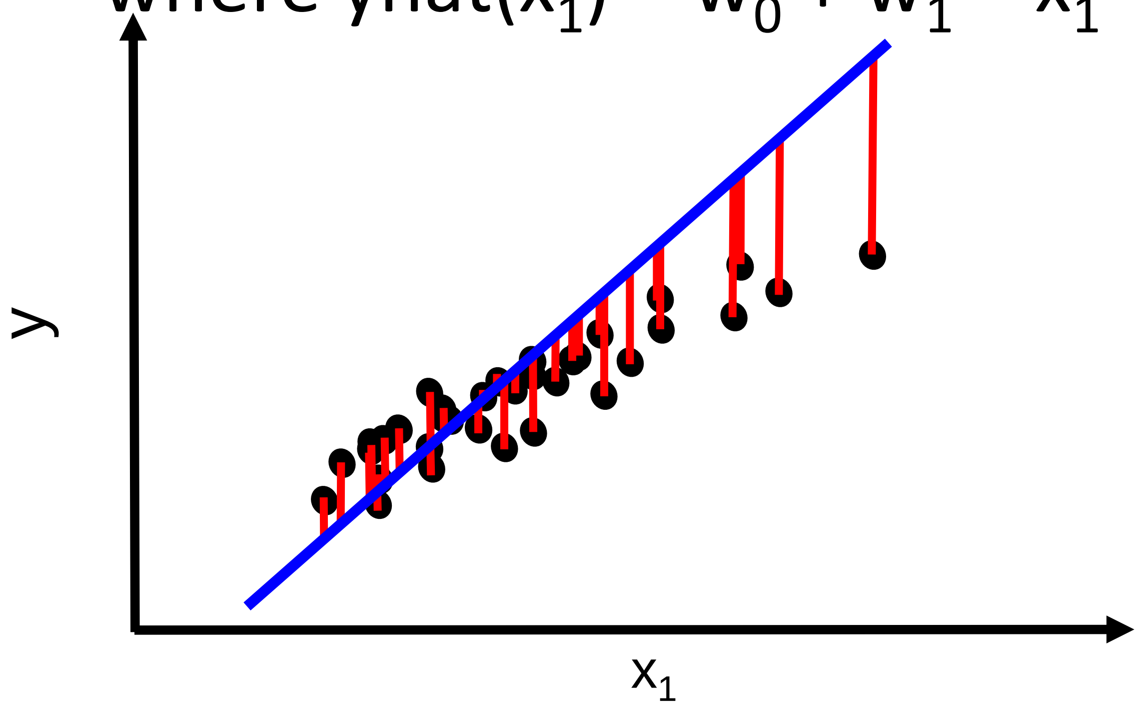
1D Linear LS Regression

Find linear model that minimizes $\sum(\text{yhat}_i - y_i)^2$

That is:

$[w_0, w_1]^* = \text{argmin} \sum(\text{yhat}_i - y_i)^2$

where $\text{yhat}(x_1) = w_0 + w_1 * x_1$



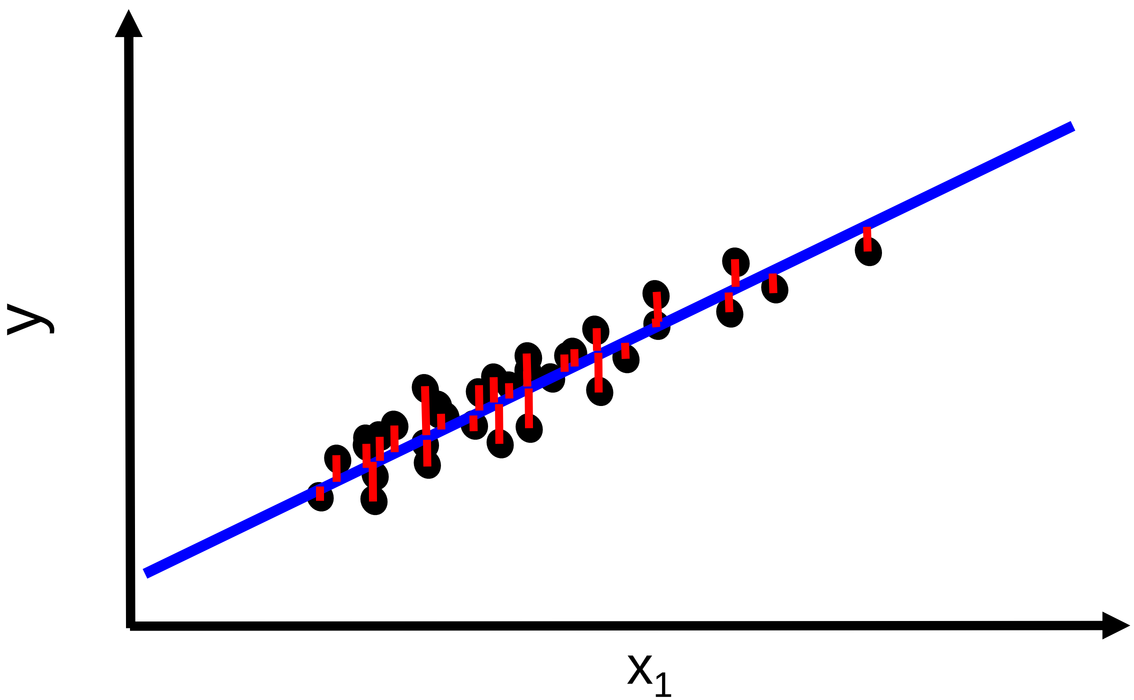
1D Linear LS Regression

$y = 1.1 + 2.3 * x_1$

i.e. $w_0=1.1$, $w_1=2.3$

Found with "least-squares learning"

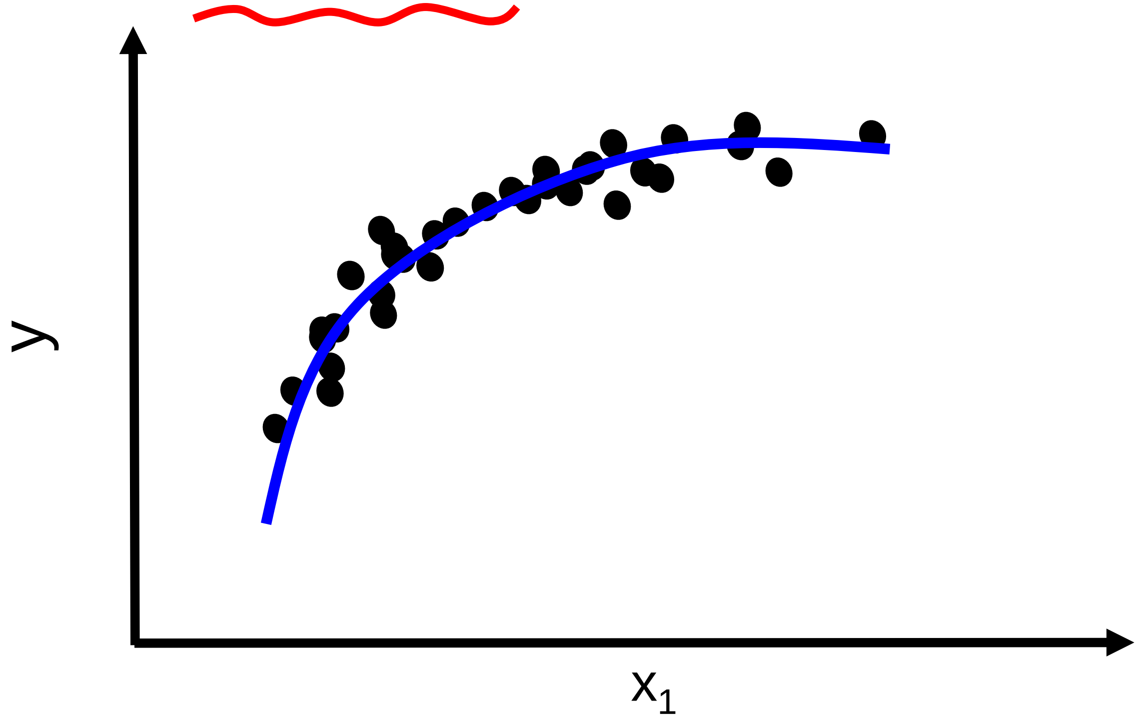
(amounts to ≈matrix inversion)



1D *Quadratic* LS Regression

$[w_0, w_1, w_{11}]^* = \text{argmin} \sum(\text{yhat}_i - y_i)^2$
where $\text{yhat}(x_1) = w_0 + w_1 * x_1 + w_{11} * x_1^2$

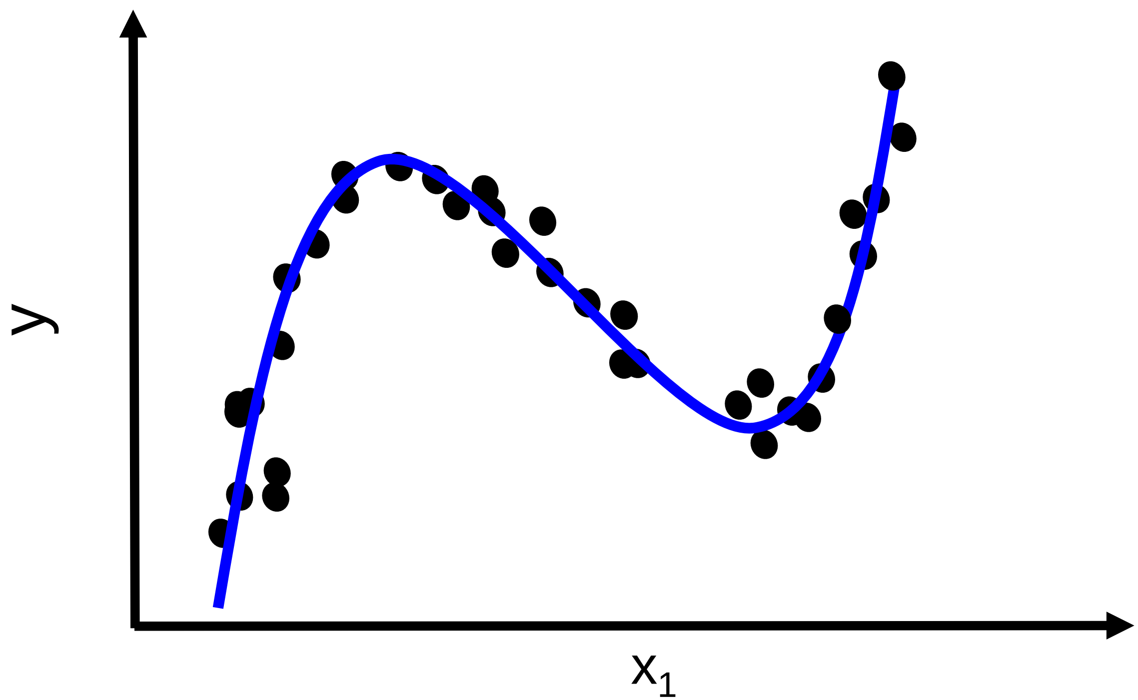
We are applying linear (LS) learning on linear & nonlinear basis functions. OK!



1D *Nonlinear* LS Regression

$[w_0, w_1, w_{sin}]^* = \text{argmin} \sum(\text{yhat}_i - y_i)^2$
where $\text{yhat}(x_1) = w_0 + w_1 * x_1 + w_{sin} * \sin(x_1)$

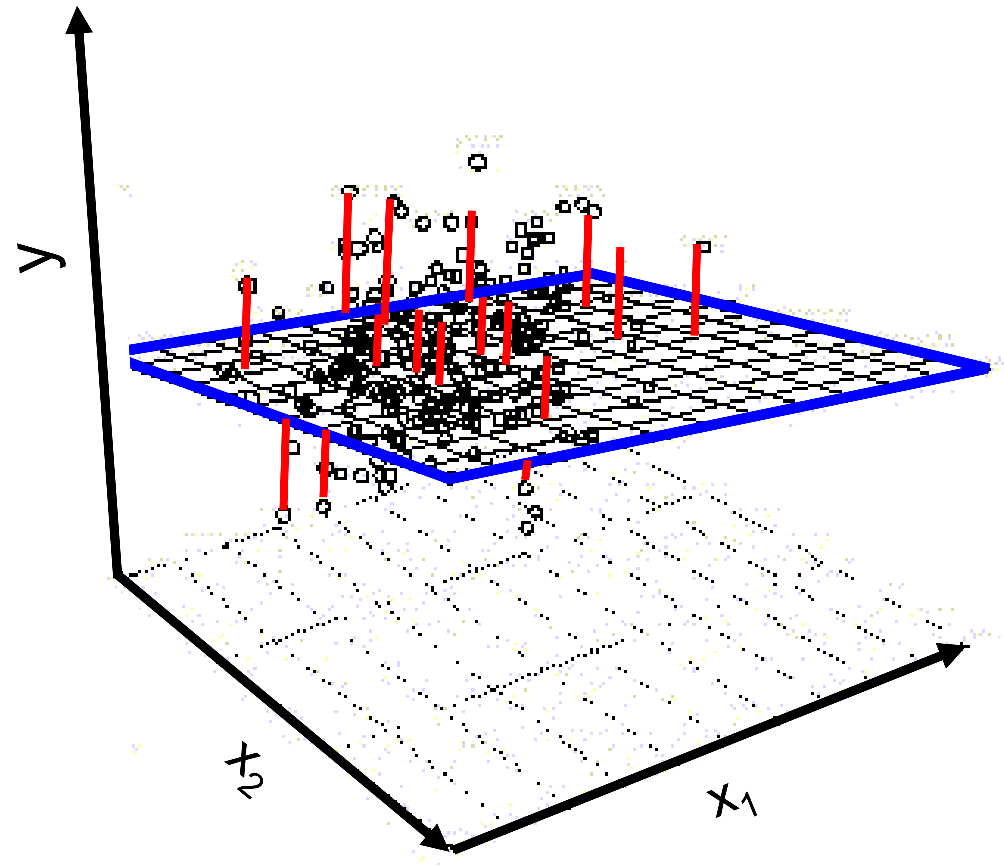
We are applying linear (LS) learning on linear & nonlinear basis functions. OK!



2D Linear LS Regression

$$[w_0, w_1, w_2]^* = \text{argmin} \sum (yhat_i - y_i)^2$$

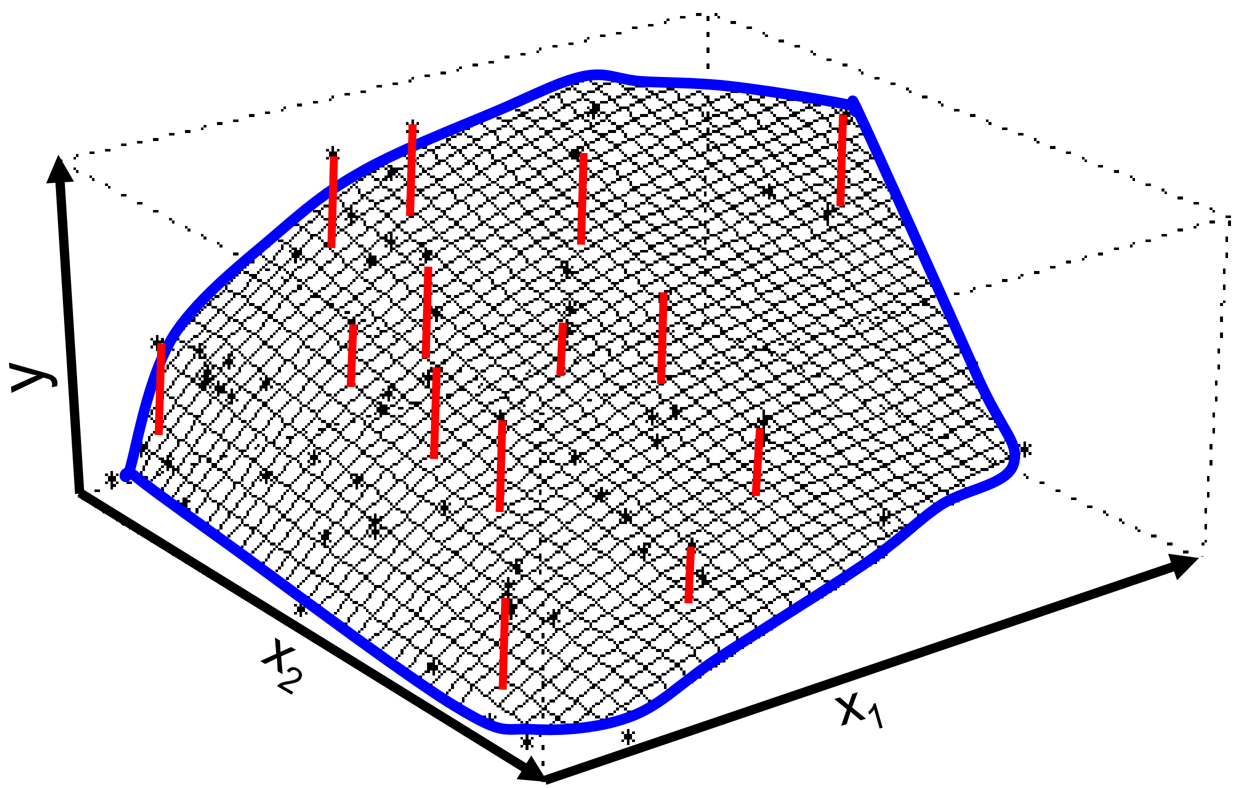
where $yhat(\mathbf{x}) = w_0 + w_1 * x_1 + w_2 * x_2$



2D Quadratic LS Regression

$[w_0, w_1, w_2, w_{11}, w_{22}, w_{12}]^* = \text{argmin} \sum(\text{yhat}_i - y_i)^2$

where $\text{yhat}(\mathbf{x}) = w_0 + w_1 * x_1 + w_{11} * x_1^2 + w_{22} * x_2^2 + w_{12} * x_1 * x_2$



Generalized Linear Model (GLM)

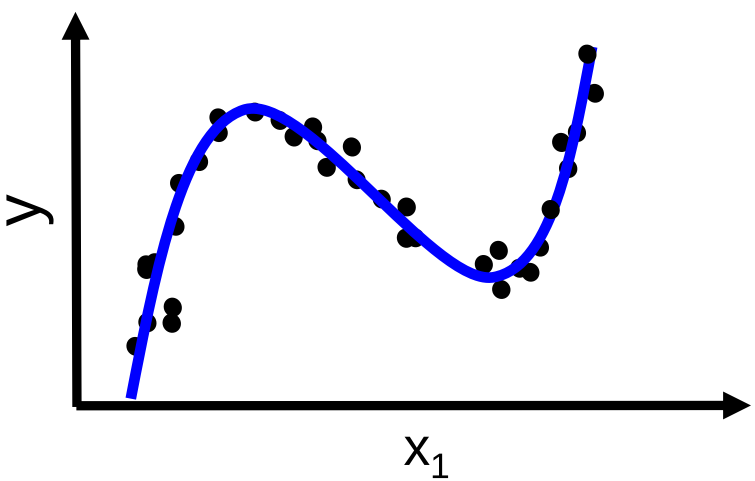
Generalized linear model (GLM) of B basis functions.

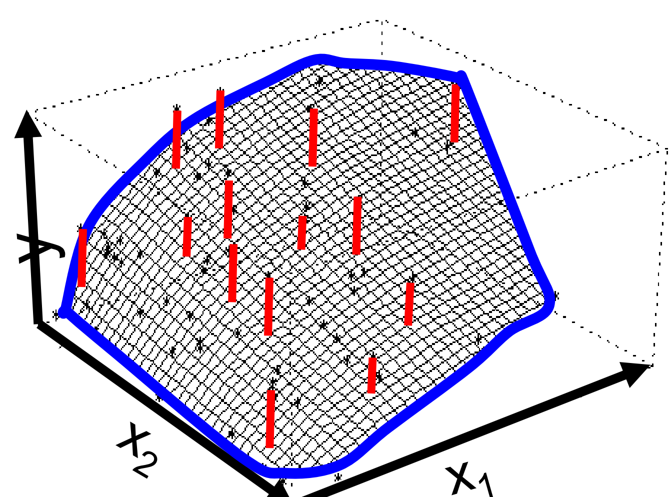
*yhat(**x**) =* $w_0 + w_1 * f_1(\mathbf{x}) + w_2 * f_2(\mathbf{x}) + \ldots + w_B * f_B(\mathbf{x})$

Just treat each basis function as an input variable, and LS-learn!

Examples:

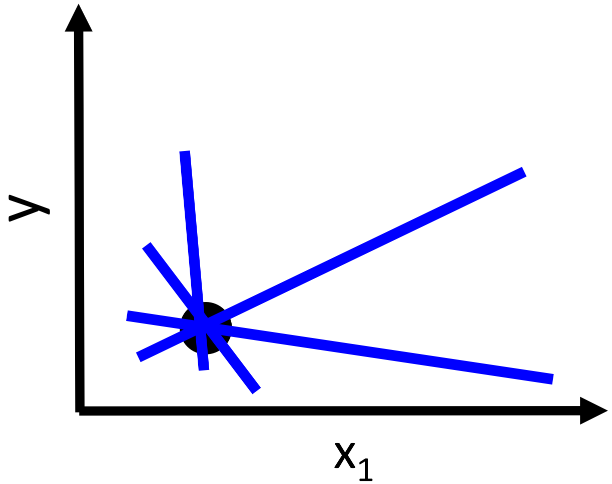
- $\text{yhat}(x_1) = w_0 + w_1 * x_1 + w_{11} * x_1{}^2$
- $\text{yhat}(x_1) = w_0 + w_1 * x_1 + w_{sin} * \sin(x_1)$
- $\text{yhat}(\mathbf{x}) = w_0 + w_1 * x_1 + w_{11} * x_{12} + w_{22} * x_{22} + w_{12} * x_1 * x_2$
- polynomials, SVMs, FFNNs, many GP SR. Universal approximator!



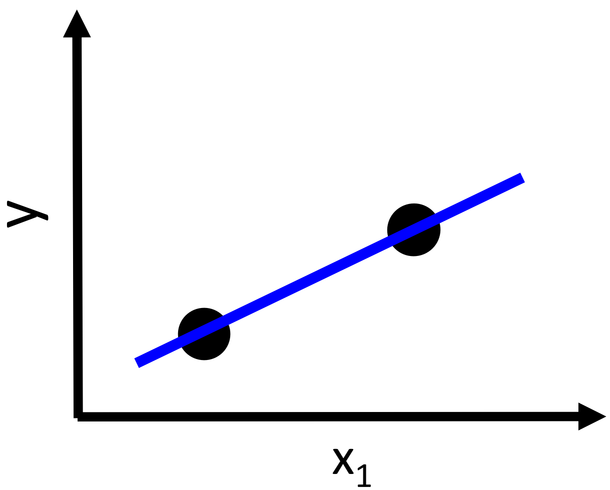


Constraint on LS Regression?

(1D Example)



1 Sample – too few



2 Samples – enough

General rule?

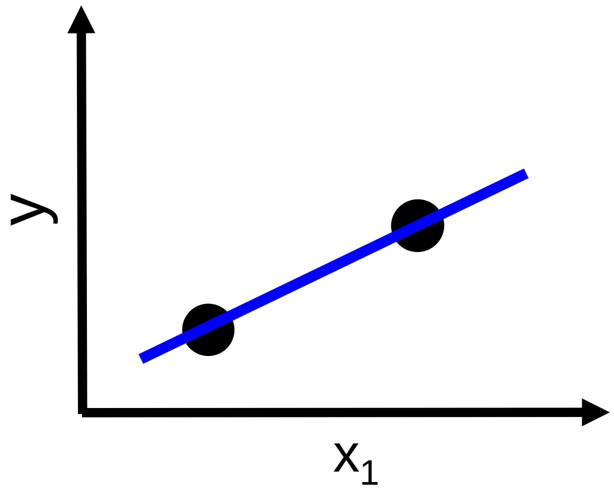
Constraint on LS Regression

General Rule:

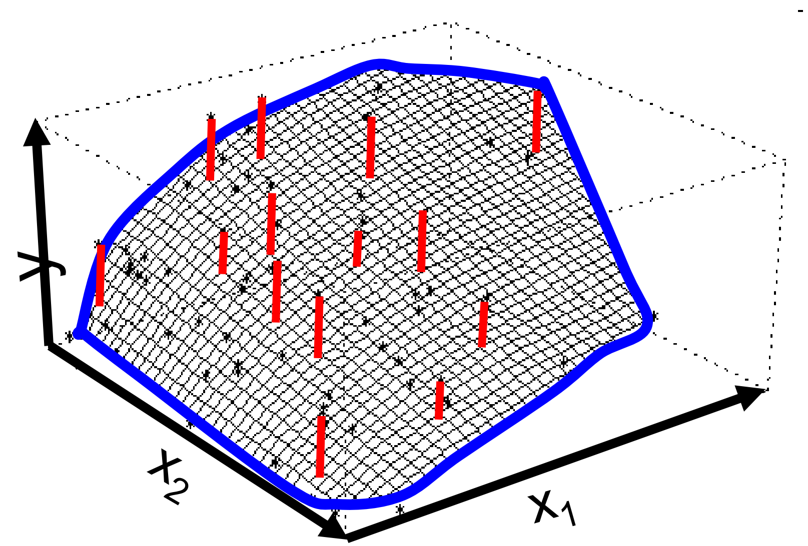
- **If *n* variables, need N ≥ n+1 training samples**

Examples:

1D Lin: $[w_0, w_1]^*$
$= \text{argmin} \sum(\text{yhat}_i - y_i)^2$
Needs ≥ 1+1 = 2 training samples.



2D Quad $[w_0, w_1, w_2, w_{11}, w_{22}, w_{12}]^*$
$= \text{argmin} \sum(\text{yhat}_i - y_i)^2$
Needs ≥ 6+1 = 7 training samples.



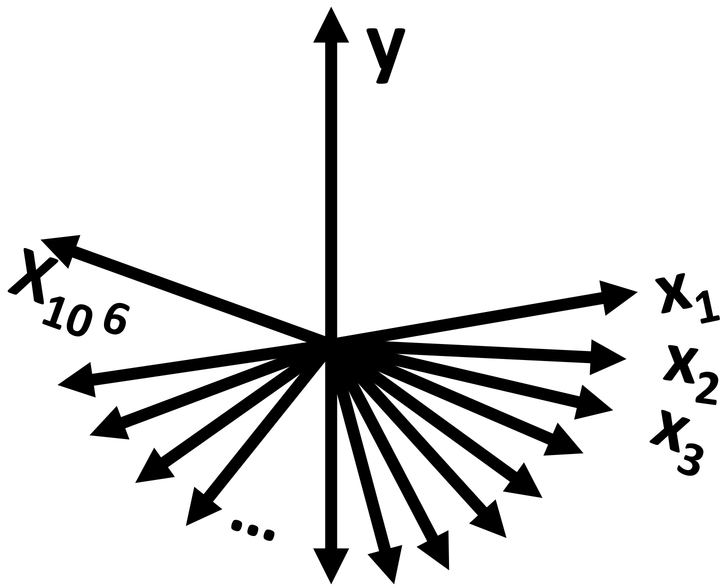
LS Regression On High Dimensionality

Consider 10,000 basis functions in a GLM
Q: Can we fit this with LS-learning?
A: Yes! (As long as ≥10,001 samples)*

Consider 1M basis functions in a GLM
Q: Can we fit this with LS-learning?
A: Yes! (As long as ≥1M+1 samples)*

*and no memory issues etc

Regression in 10^6D ?



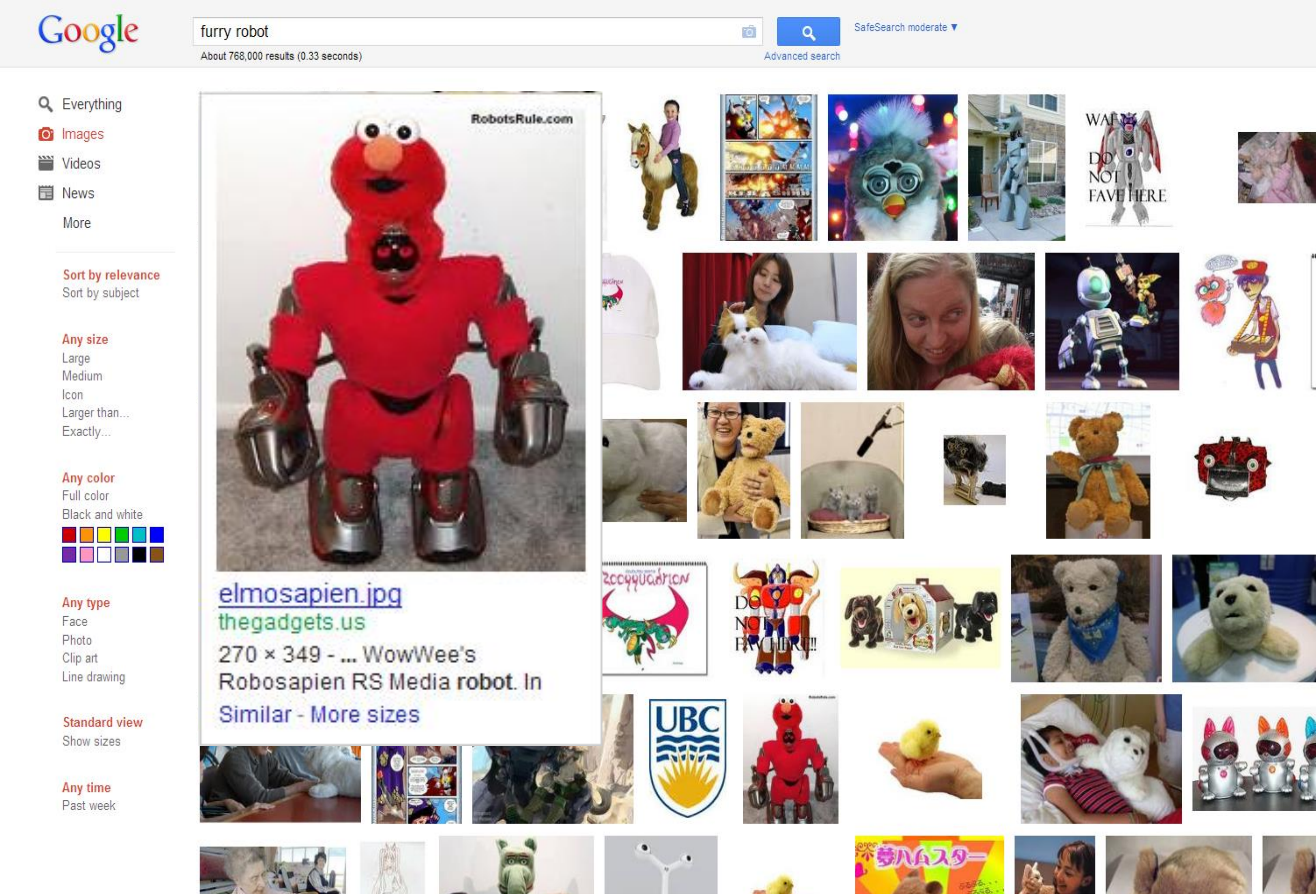
How?? (and why??)

90˚ turn…

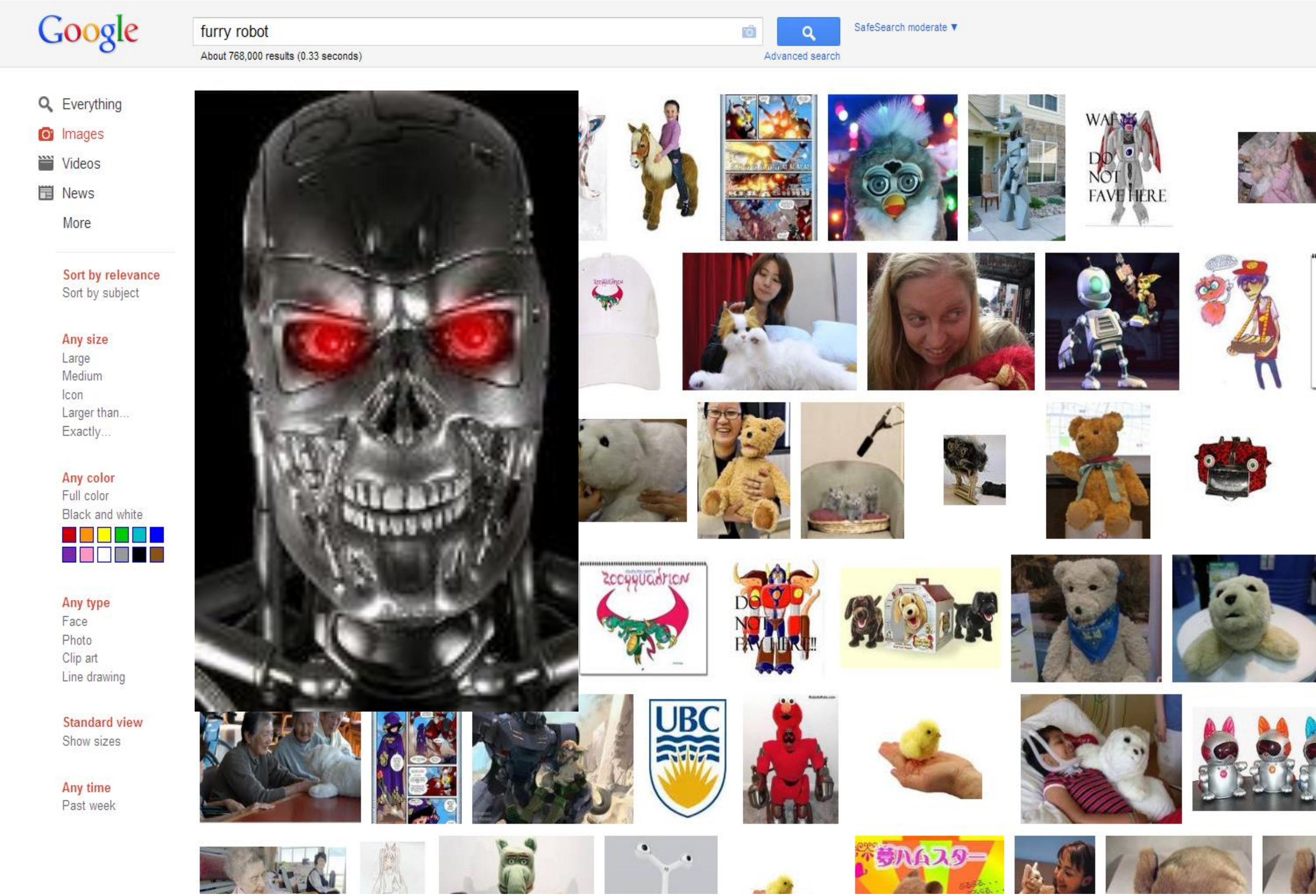
Google
images

furry robot

Search Images



How does Google find furry robots?

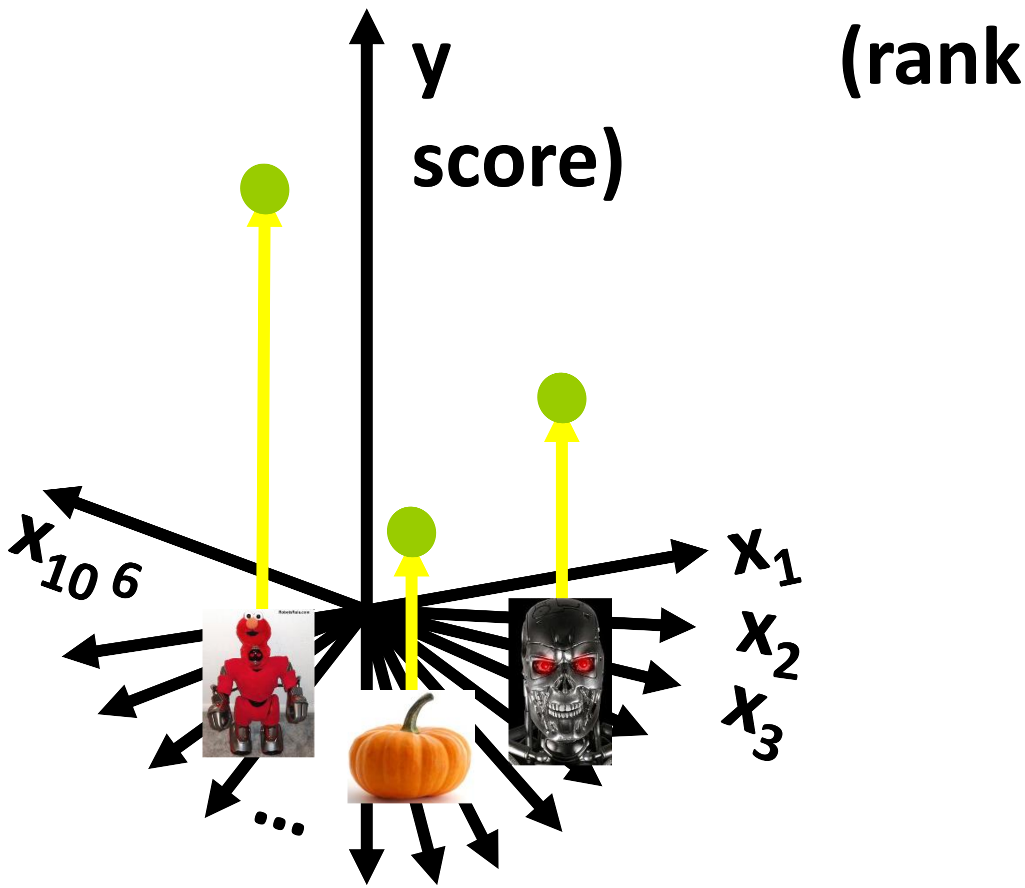


How does Google *accurately* find furry robots?

Q: How does Google (accurately) find furry robots?

A:

1. Treat images as 1000x1000 = 10^6 input variables (!)
2. Do regression on "known" images (furry vs. non)
3. Rank the other images. Easy! ☺



[NIPS 2010]

Q: State of the art in image search? (NIPS ’09)
A: BHALR!*

*__B__ig, __H__airy, __A__udacious __L__inear __R__egression

1000 pixels x 1000 pixels = 1M input variables
100-1000 samples.
Then apply linear regression or classification

Q: State of the art in image search? (NIPS ’09)
A: BHALR!*

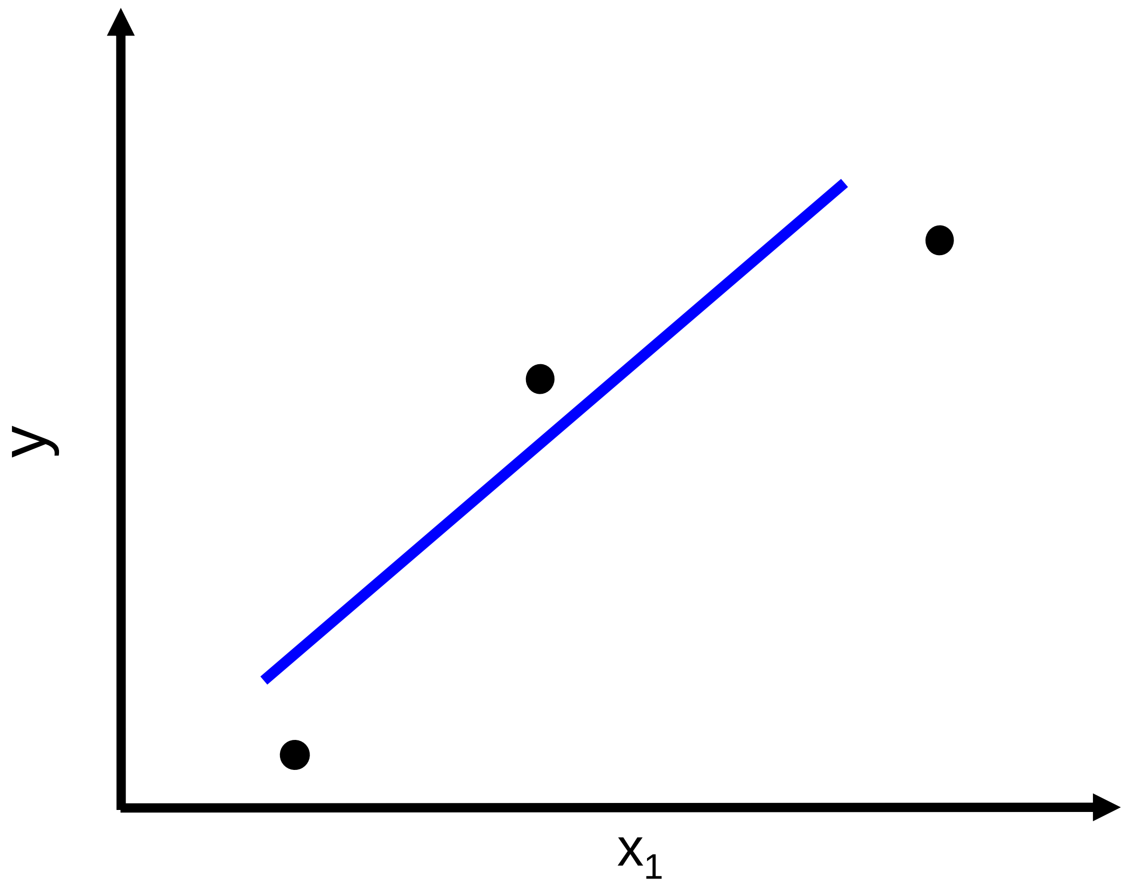
***B**ig, **H**airy, **A**udacious **L**inear **R**egression

1000 pixels x 1000 pixels = 1M input variables
100-1000 samples.
Then apply linear regression or classification

But 100 << 1M. *HOW ??*

Linear Regression

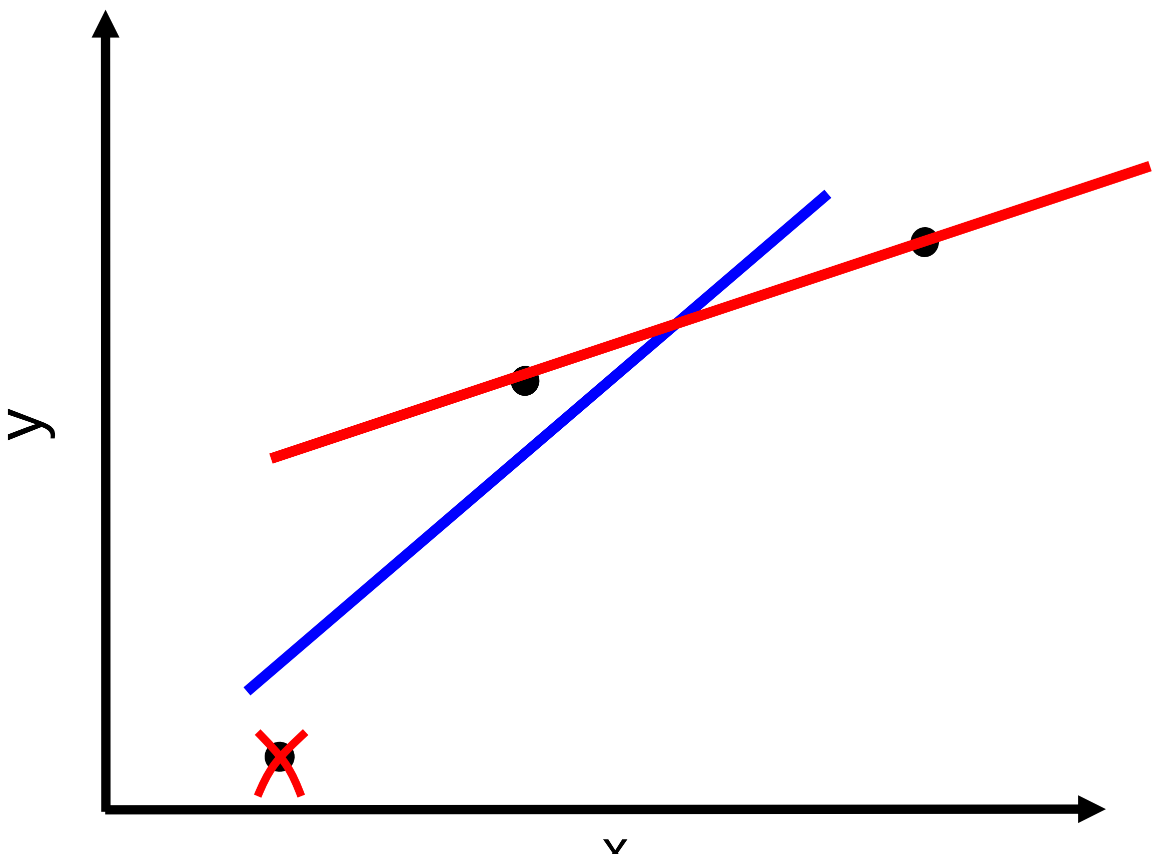
Q: What happens when samples $N \rightarrow$ # variables n ?



Linear Regression

Q: What happens when # samples $N \rightarrow$ # variables n ?

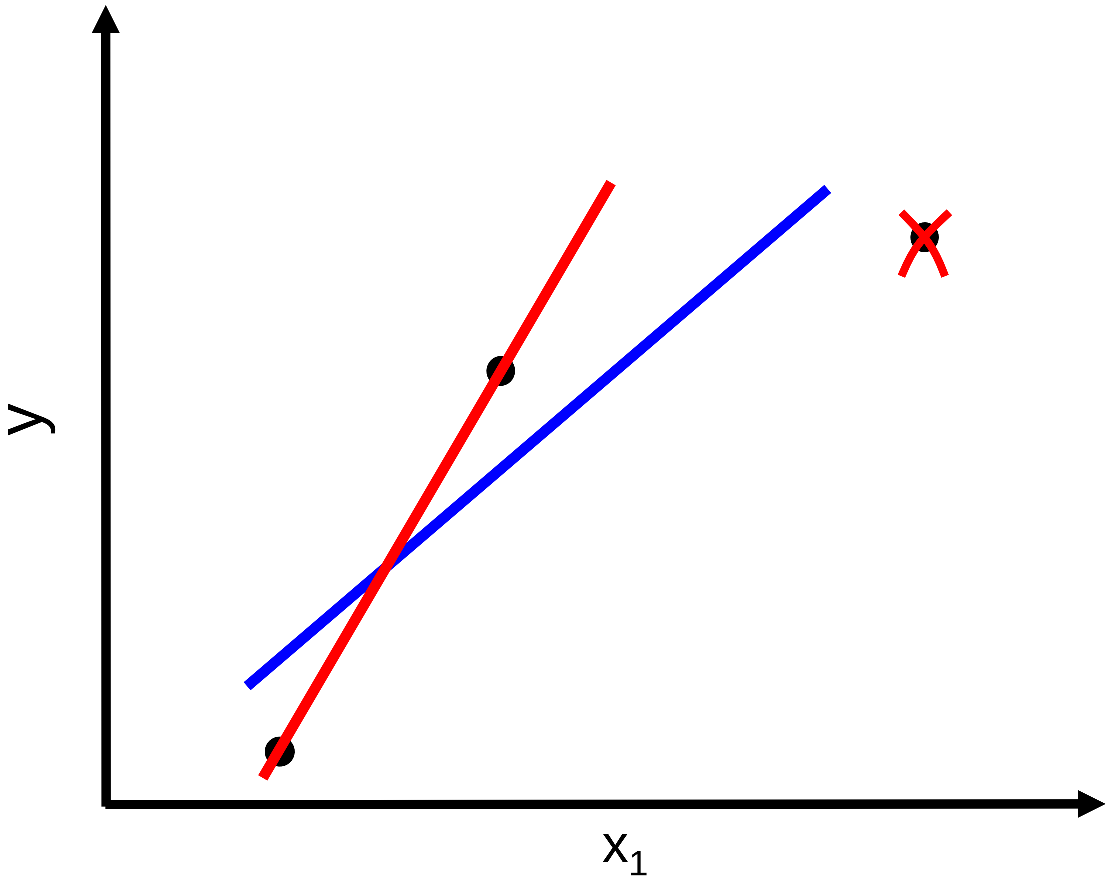
A: Model gets more sensitive!



Linear Regression

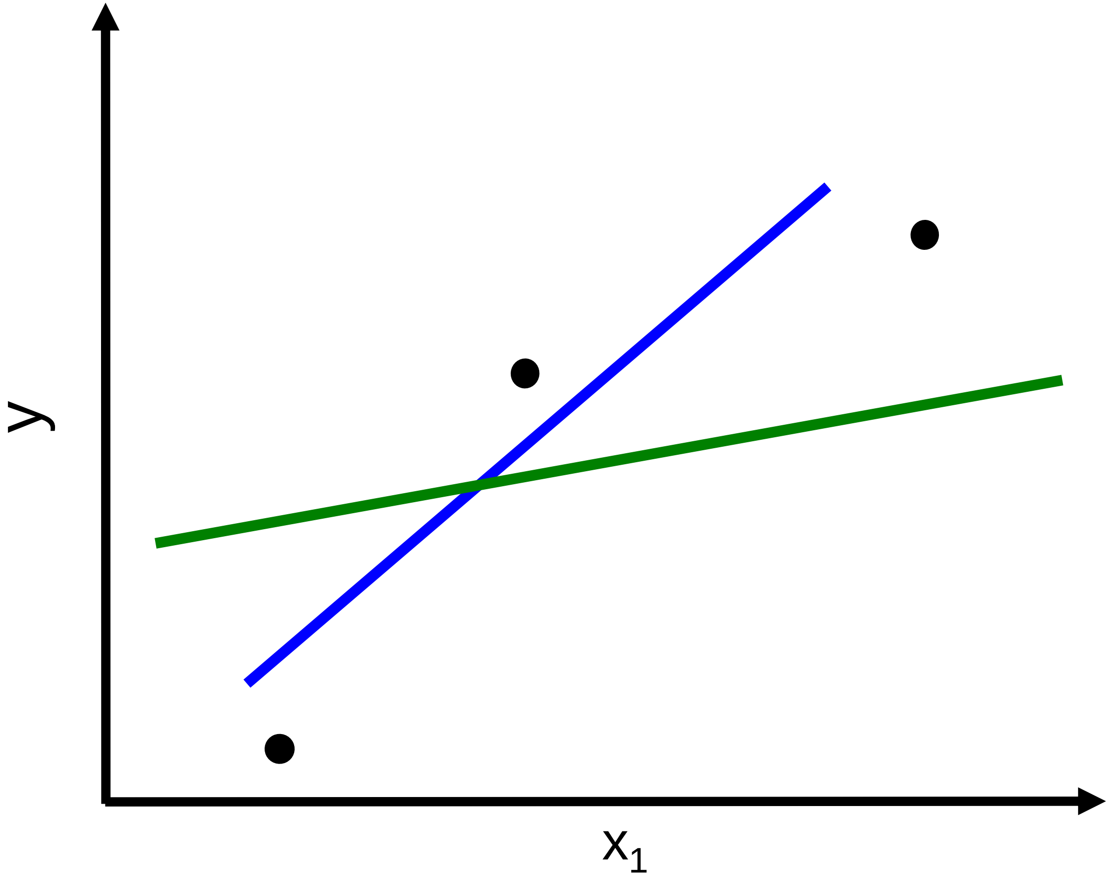
Q: What happens when # samples $N \rightarrow$ # variables n ?

A: Model gets more sensitive!



Linear Regression

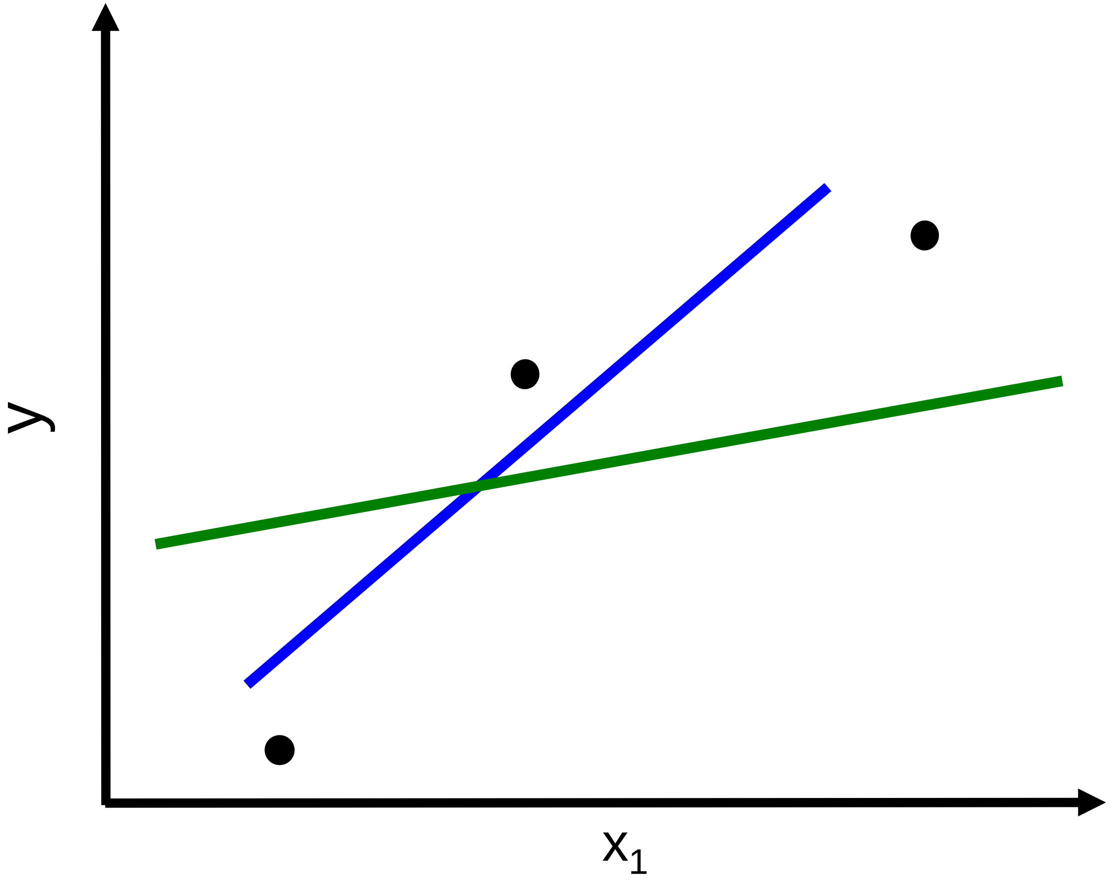
A model that's "less sensitive"



Linear Regression

A model that's "less sensitive"

Smaller |dy/dx| means less sensitive



Linear Regression

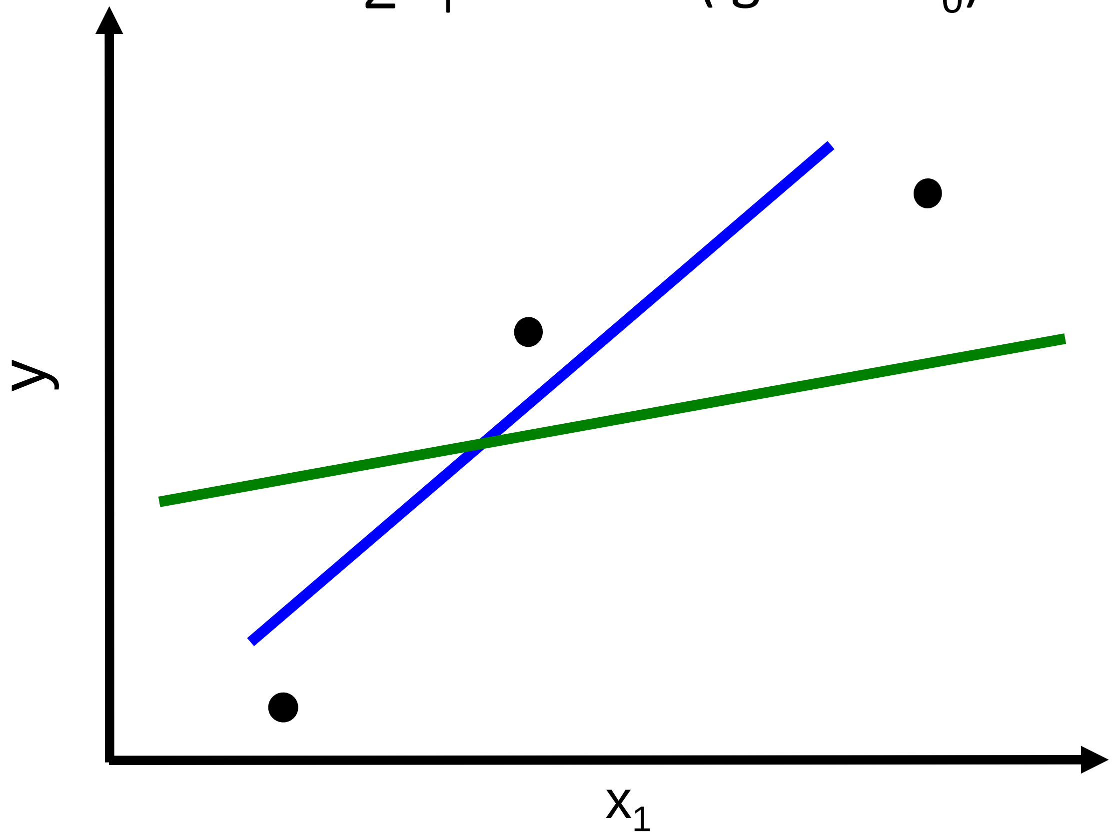
A model that's "less sensitive"

Smaller |dy/dx| means less sensitive

i.e. given $yhat(x_1) = w_0 + w_1 * x_1$

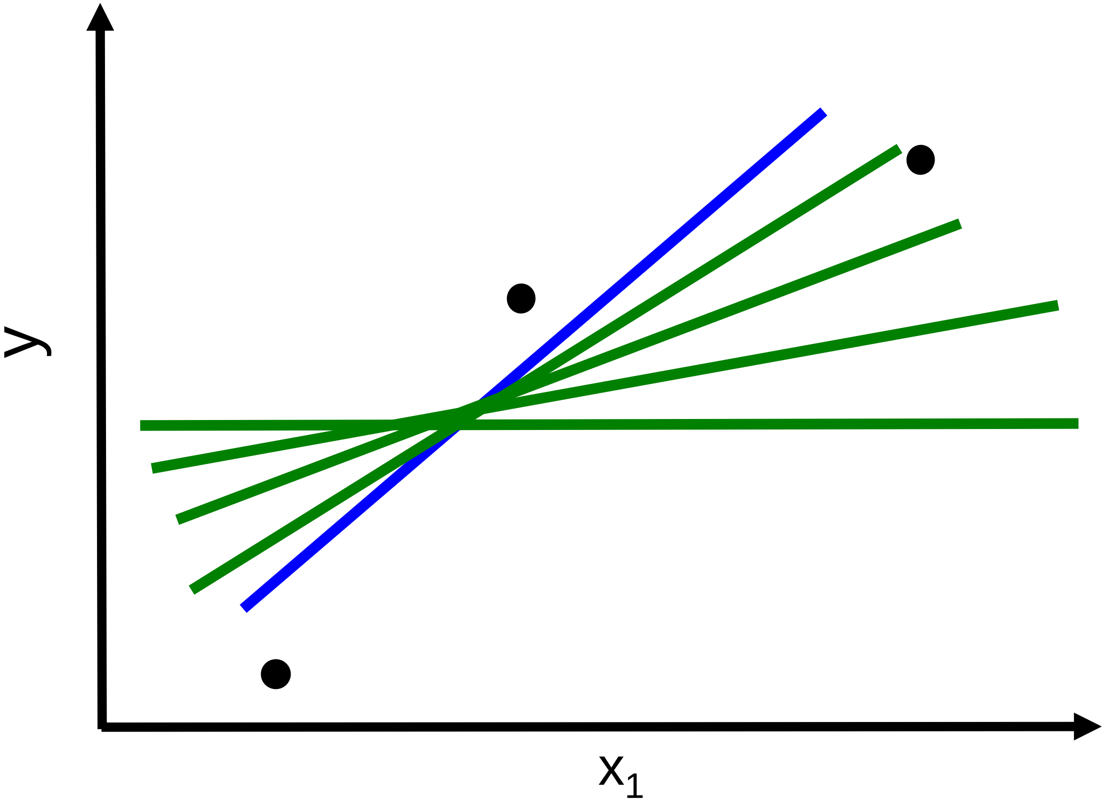
A smaller $|w_1|$ means less sensitive

or smaller $\sum w_i$ for $n > 1$ (ignore w_0)



Linear Regression

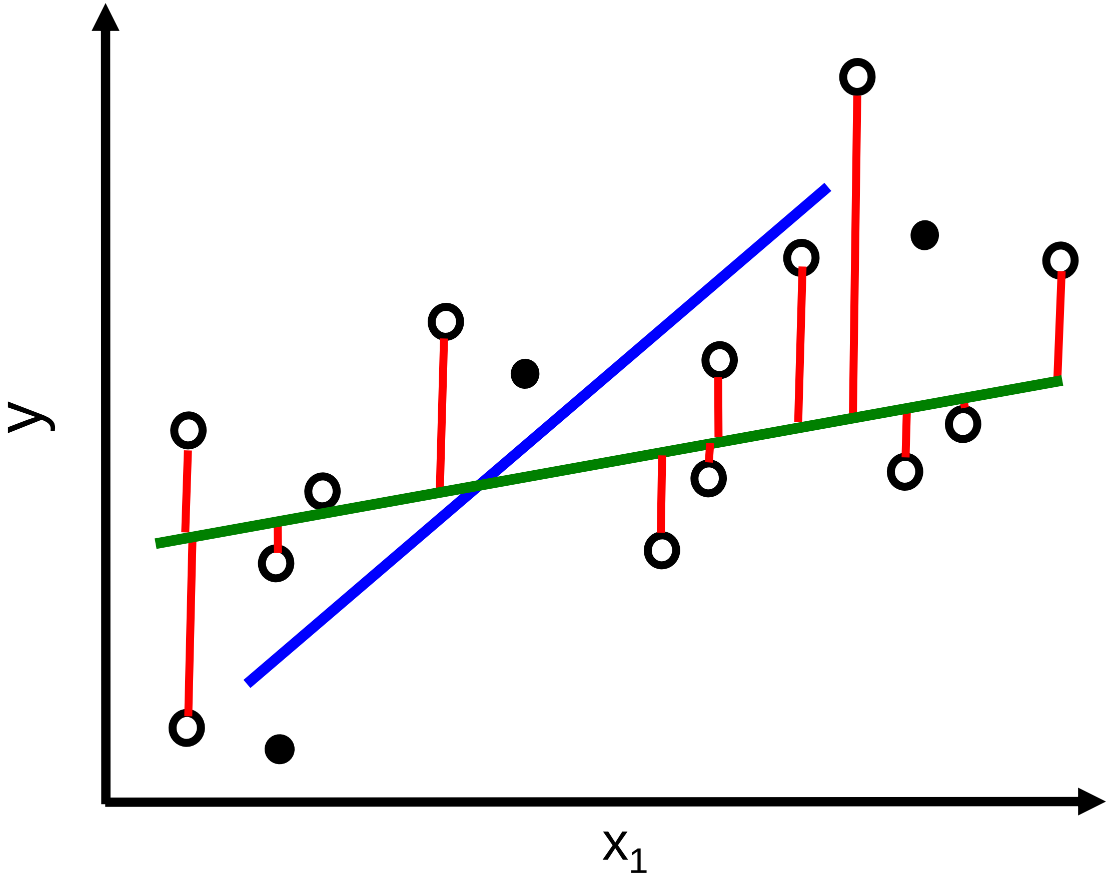
Least-sensitive model has slope of 0
(By definition)
(And also when viewed pragmatically as a model)



Linear Regression

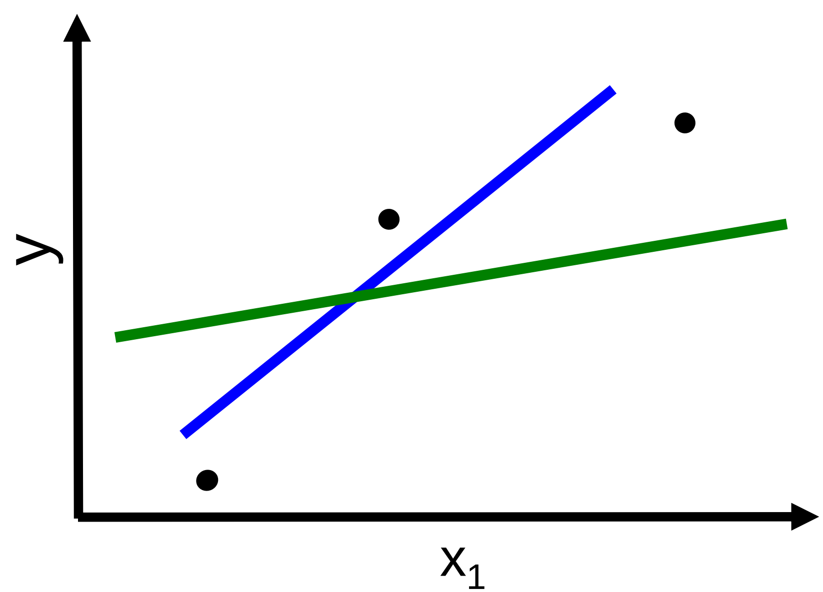
A model that's "less sensitive"

"less sensitive" ≈ lower future prediction error
(in light of less training data)



Linear Regression

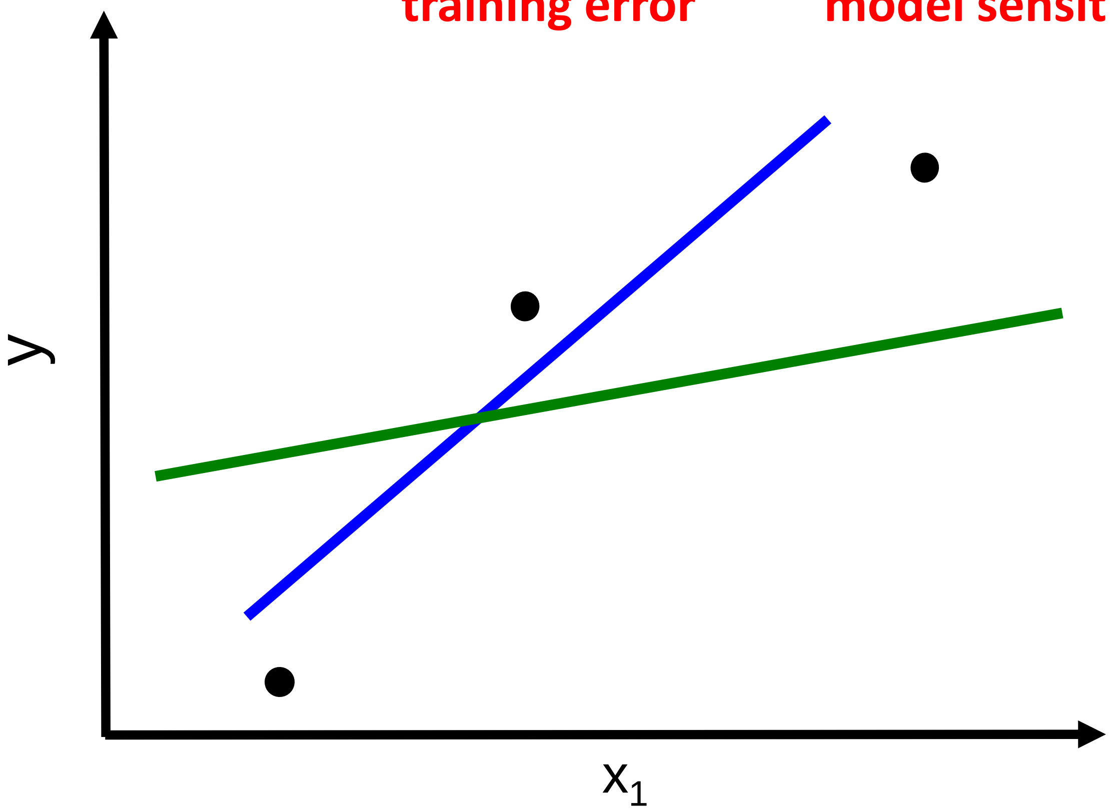
- Aim: minimize *future* prediction error
- Pragmatic Issue: we only have access to training data!
- Trick: minimize sensitivity ≈ minimize future prediction error
- But *do* consider training data to bias the model (otherwise we end up with a constant – useless!)
- So: **minimize a combination of training error vs. sensitivity** (bias vs. variance tradeoff) (explanation-of-data vs. overfitting)



Linear Regression

- Minimize a combination of training error and model sensitivity
- Formulation:

$$\mathbf{w}^* = \text{argmin} \; (\underbrace{\textstyle\sum(\text{yhat}_i(\mathbf{w}) - y_i)^2}_{\textbf{training error}} \; + \; \underbrace{\lambda * \textstyle\sum|w_i|}_{\textbf{model sensitivity}} \;)$$



Linear Regression

- Minimize a combination of training error and sensitivity

- Formulation:

  $\mathbf{w}^* = \text{argmin} \left(\sum(\text{yhat}_i(\mathbf{w}) - y_i)^2 + \lambda * \sum|w_i| \right)$

  [Lasso]

  OR

  $\mathbf{w}^* = \text{argmin} \left(\sum(\text{yhat}_i(\mathbf{w}) - y_i)^2 + \lambda * \sum w_i^2 \right)$

  [Ridge Regression]

  … [Elastic Net, Gradient Directed Regularization, …]

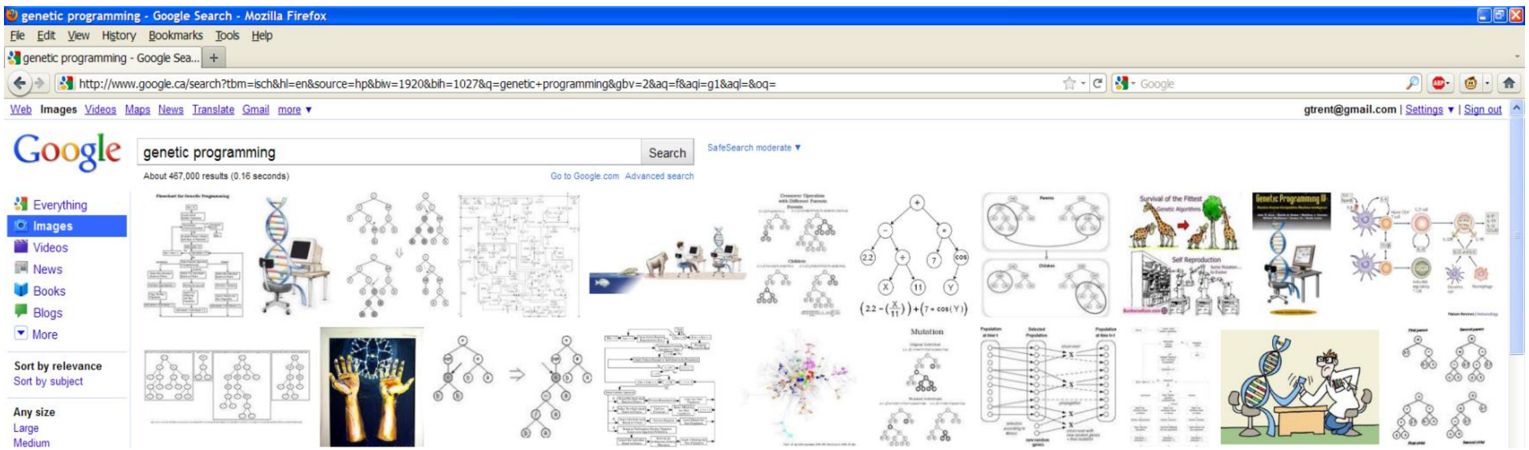
This is regularized linear learning

Regularized Linear Regression

- **Cool property #1:** solving a regularized learning problem is just as fast (or faster) than solving a least-squares learning problem!
    - Why: convex optimization problem – one big hill

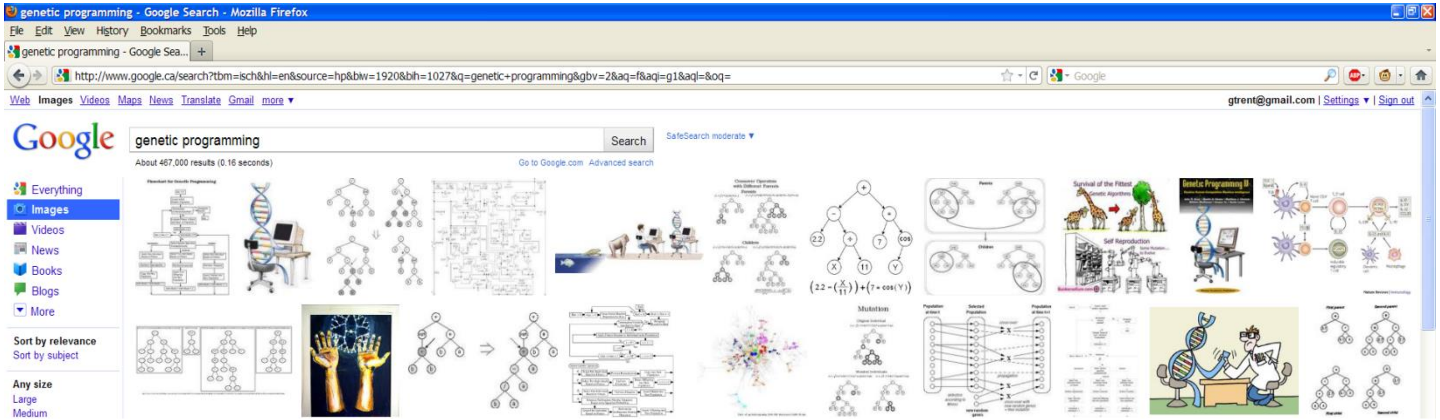
Regularized Linear Regression

- Remember BHALR image search problem?
  - n = 1M variables, N=1000 samples



Regularized Linear Regression

- Remember BHALR image search problem?
  - n = 1M variables, N=1000 samples



- **Cool property #2:** can have more coefficients than samples! That is, can handle $n >> N$!
  - Because the regularization term minimizes the sensitivity, i.e. the "degree of screwup"

  $\mathbf{w}^* = \text{argmin} \left(\sum(\text{yhat}_i(\mathbf{w}) - y_i)^2 + \lambda * \sum|w_i| \right)$

Regularized Linear Regression

When solving $\mathbf{w}^* = \operatorname{argmin} \left(\sum(\text{yhat}_i(\mathbf{w}) - y_i)^2 + \lambda * \sum|w_i| \right)$,
What is a good value for λ?

- **Case: λ=0** $\sum(\text{yhat}_i(\mathbf{w}) - y_i)^2 + \underbrace{\lambda * \sum|w_i|}_{0}$

…reduces to least-squares

Regularized Linear Regression

When solving **w*** = argmin ($\sum(\text{yhat}_i(\mathbf{w}) - y_i)^2 + \lambda * \sum|w_i|$),
What is a good value for λ?

- **Case: λ=0** $\quad \sum(\text{yhat}_i(\mathbf{w}) - y_i)^2 + \cancelto{0}{\lambda * \sum|w_i|}$

...reduces to least-squares

- **Case: λ=∞** $\quad \cancelto{0}{\sum(\text{yhat}_i(\mathbf{w}) - y_i)^2 + \lambda} * \sum|w_i|$

...gives a constant (w_0=const; $w_1=w_2=\ldots = 0$)

Regularized Linear Regression

When solving **w*** = argmin ($\sum(\text{yhat}_i(\mathbf{w}) - y_i)^2 + \lambda * \sum|w_i|$),
What is a good value for λ?

- **Case: λ=0** $\quad \sum(\text{yhat}_i(\mathbf{w}) - y_i)^2 + \cancelto{0}{\lambda * \sum|w_i|}$

...reduces to least-squares

- **Case: λ=∞** $\quad \cancelto{0}{\sum(\text{yhat}_i(\mathbf{w}) - y_i)^2 + \lambda} * \sum|w_i|$

...gives a constant (w_0=const; $w_1=w_2=\ldots = 0$)

- **Case: λ in-between**

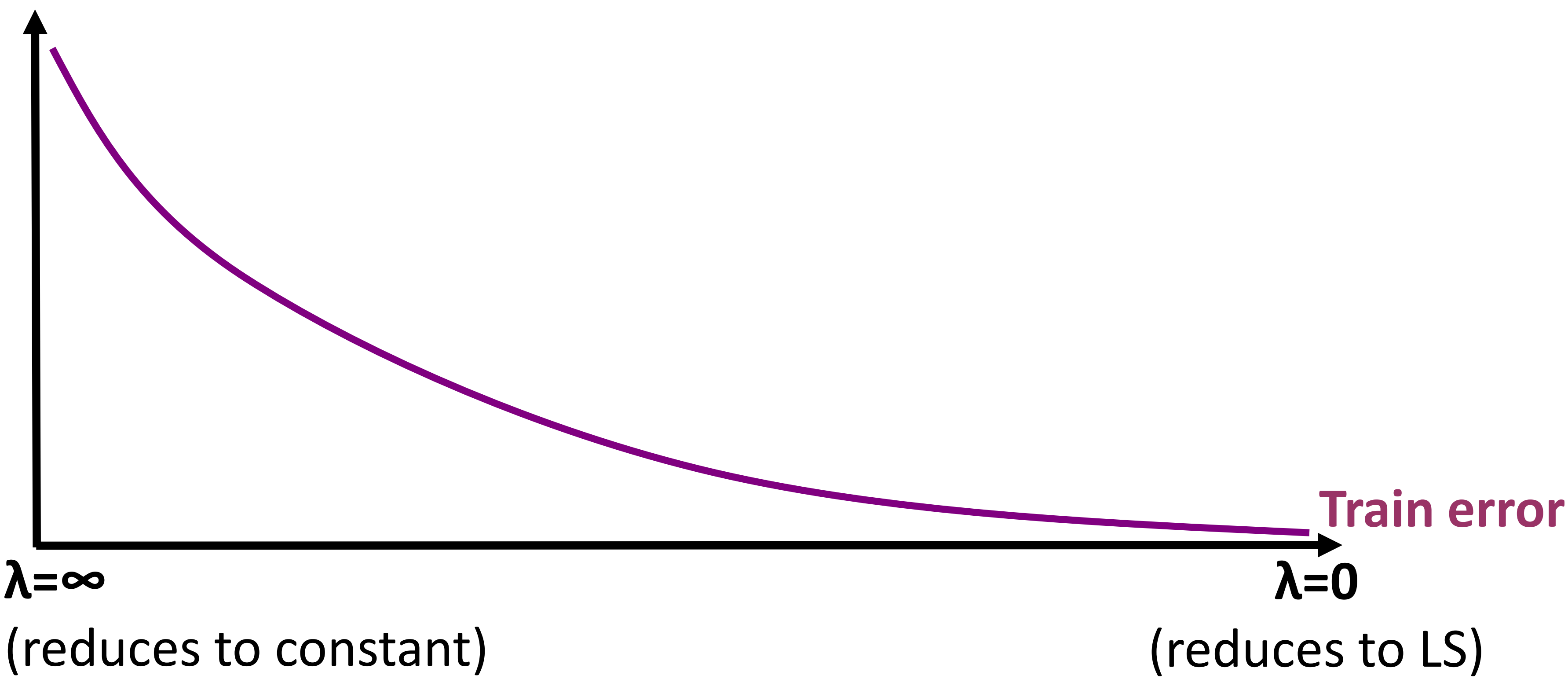
...is a balance between constant & LS.

Regularized Linear Regression

When solving **w*** = argmin ($\sum(\text{yhat}_i(\mathbf{w}) - y_i)^2 + \lambda * \sum|w_i|$),

~~**What is a good value for λ?**~~

Learn w* at *many* values of λ



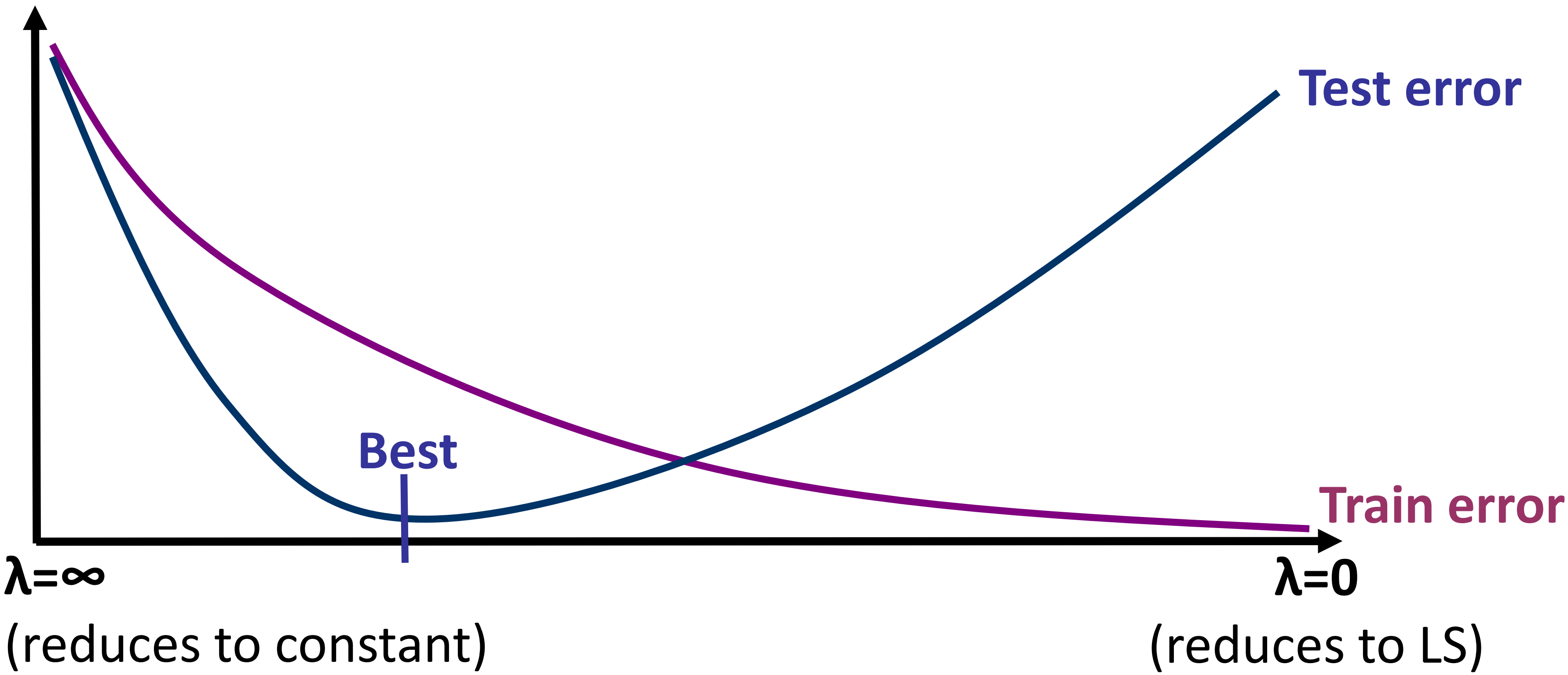
Regularized Linear Regression

When solving **w*** = argmin ($\sum(\text{yhat}_i(\mathbf{w}) - y_i)^2 + \lambda * \sum|w_i|$),

~~**What is a good value for λ?**~~

Learn w* at *many* values of λ, and keep "best"

("Best" = best error on a left-out test set.)



Regularized Linear Regression

Algorithm

λ= huge (e.g. 1e40)

w = **0**

while λ > 1e-10

  $\lambda = \lambda / 10$

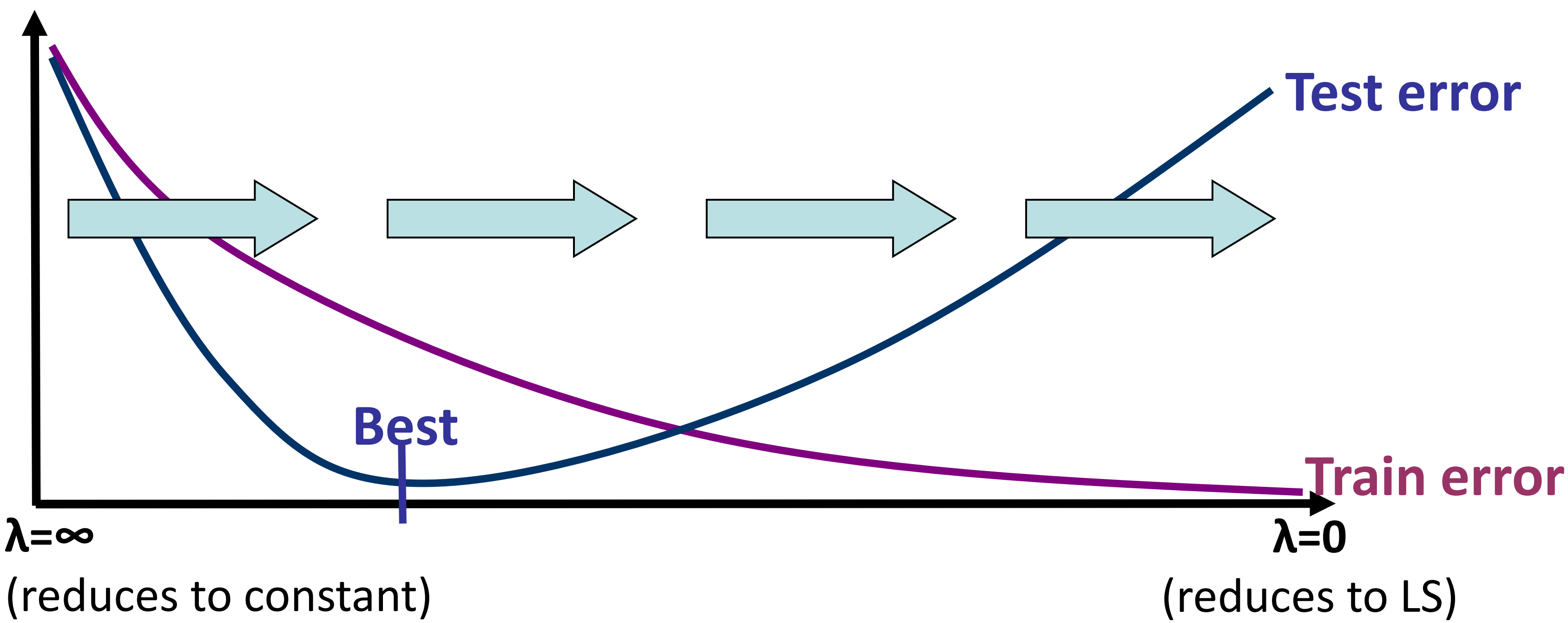
  $\mathbf{w}$ = solveAt($\mathbf{X}_{train}$, $\mathbf{y}_{train}$, λ, $\mathbf{w}_{init}=\mathbf{w}$)

  Compute error on test set

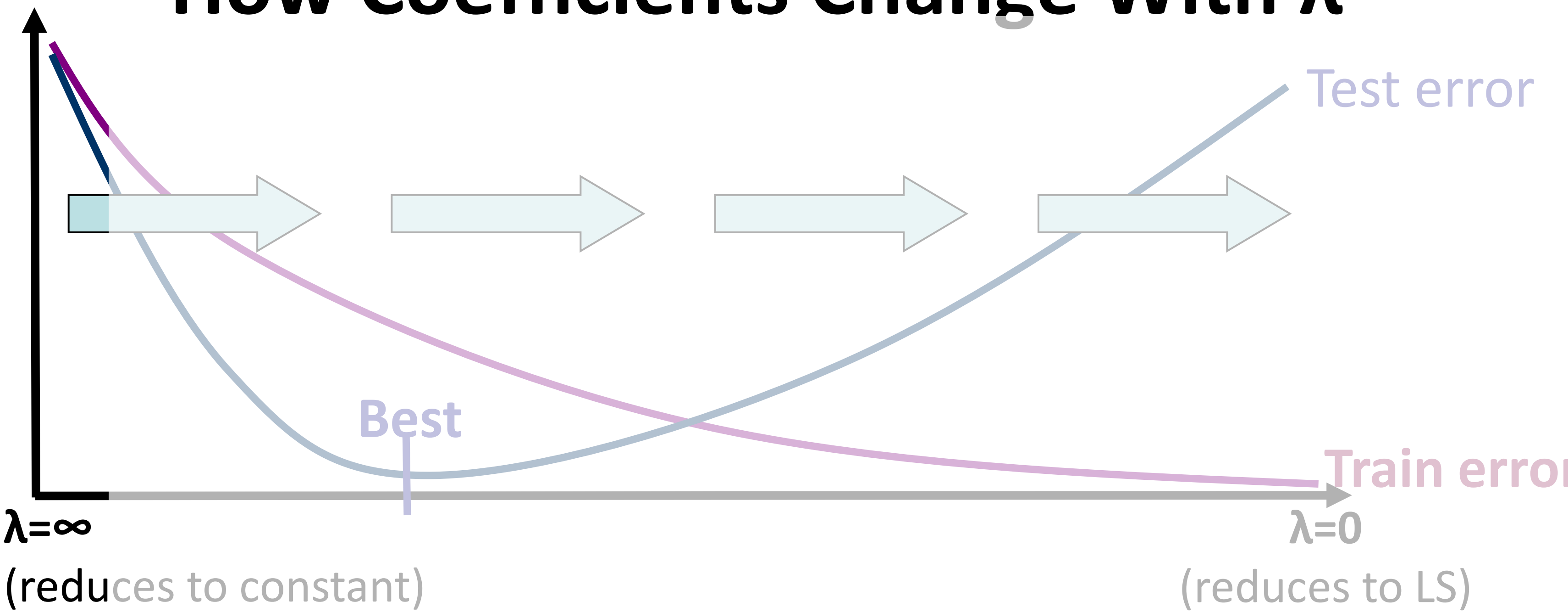
Return **w** with best test error

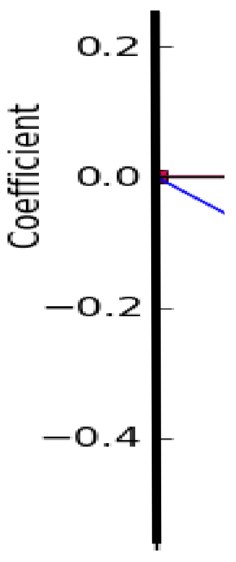
Solves

$\mathbf{w}^*$ = argmin ($\sum(\text{yhat}_i(\mathbf{w}) - y_i)^2 + \lambda * \sum|w_i|$)



Regularized Linear Regression: How Coefficients Change With λ

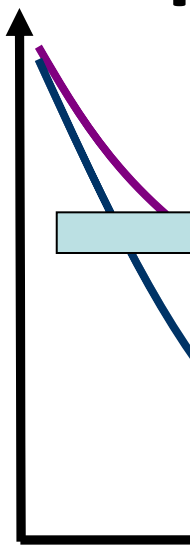




w_3 "pops in"

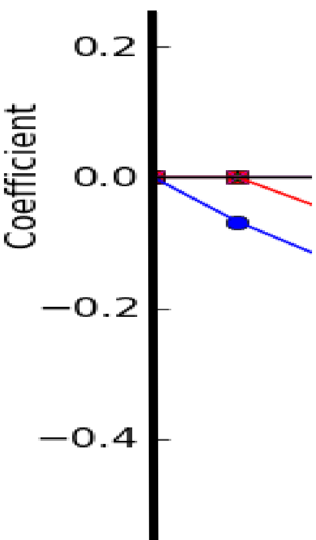
(All other w_i=0)

Regularized Linear Regression: How Coefficients Change With λ



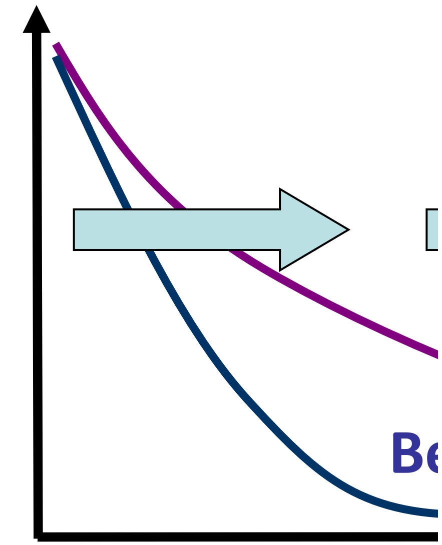
λ=∞

(reduces t



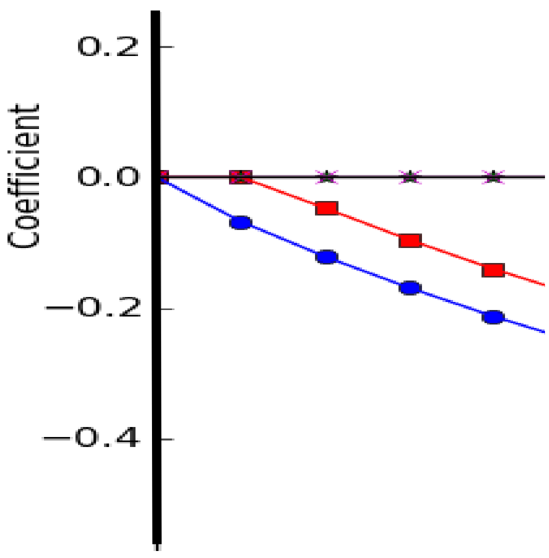
w_6 “pops in”

Regularized Linear Regression: How Coefficients Change With λ

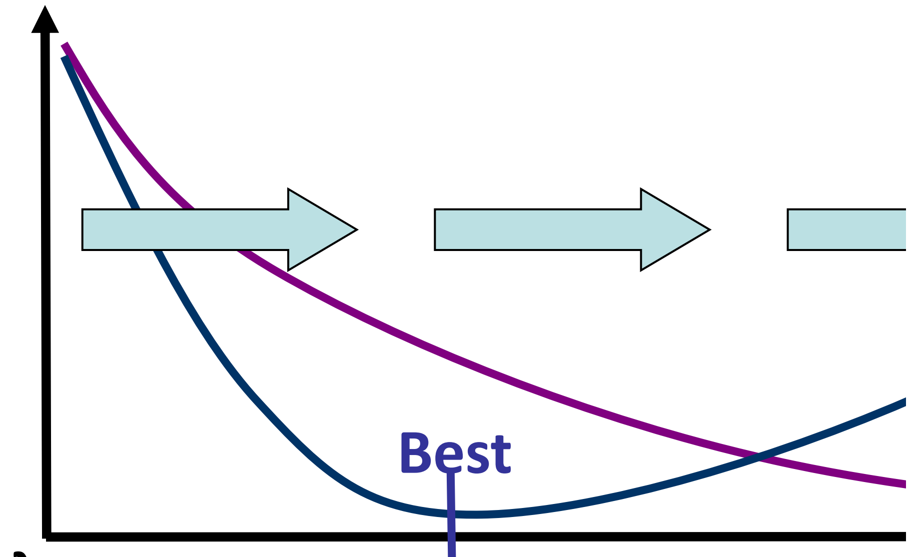


λ=∞

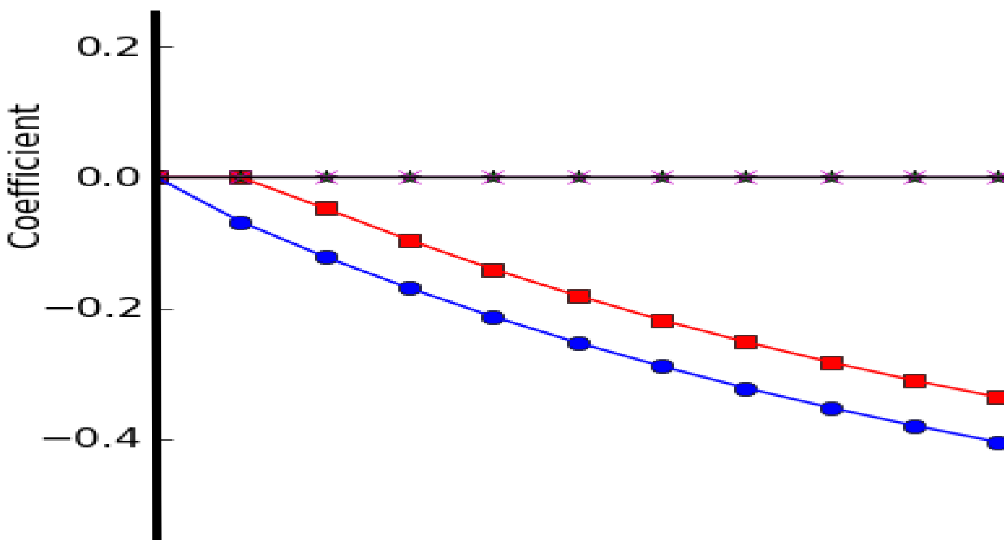
(reduces to constant



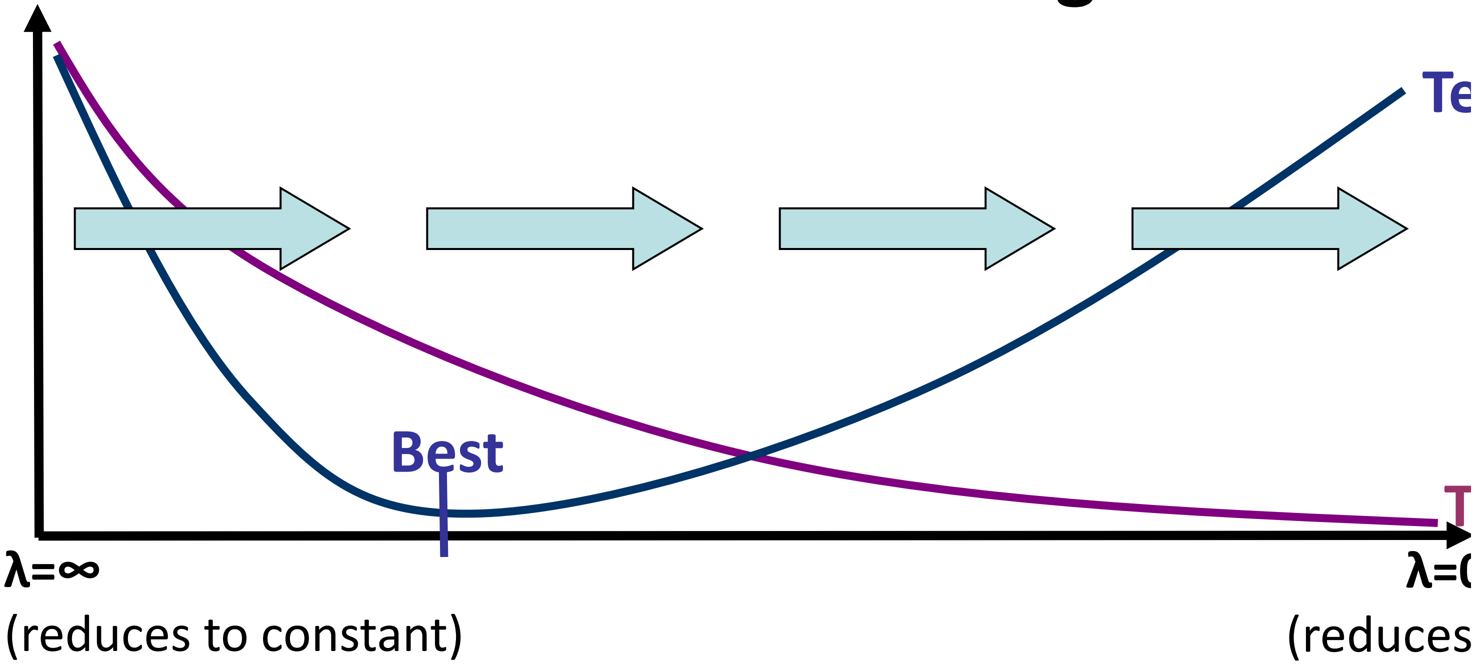
Regularized Linear Regression: How Coefficients Change With λ

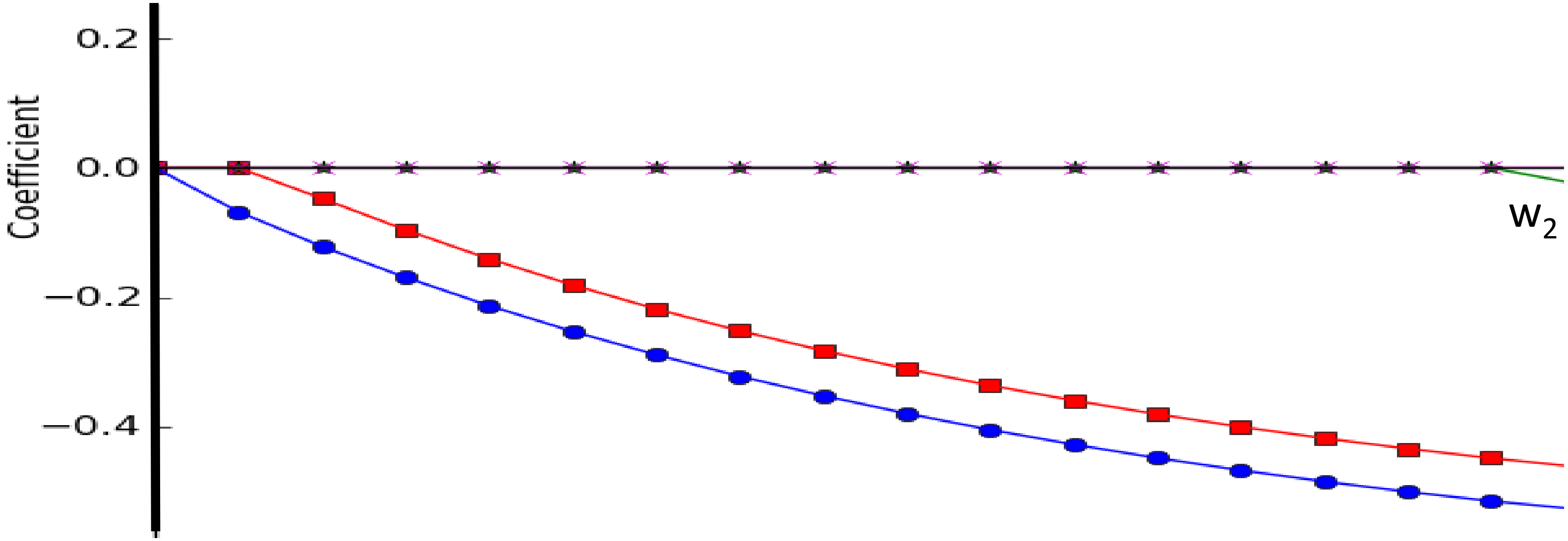


λ=∞
(reduces to constant)



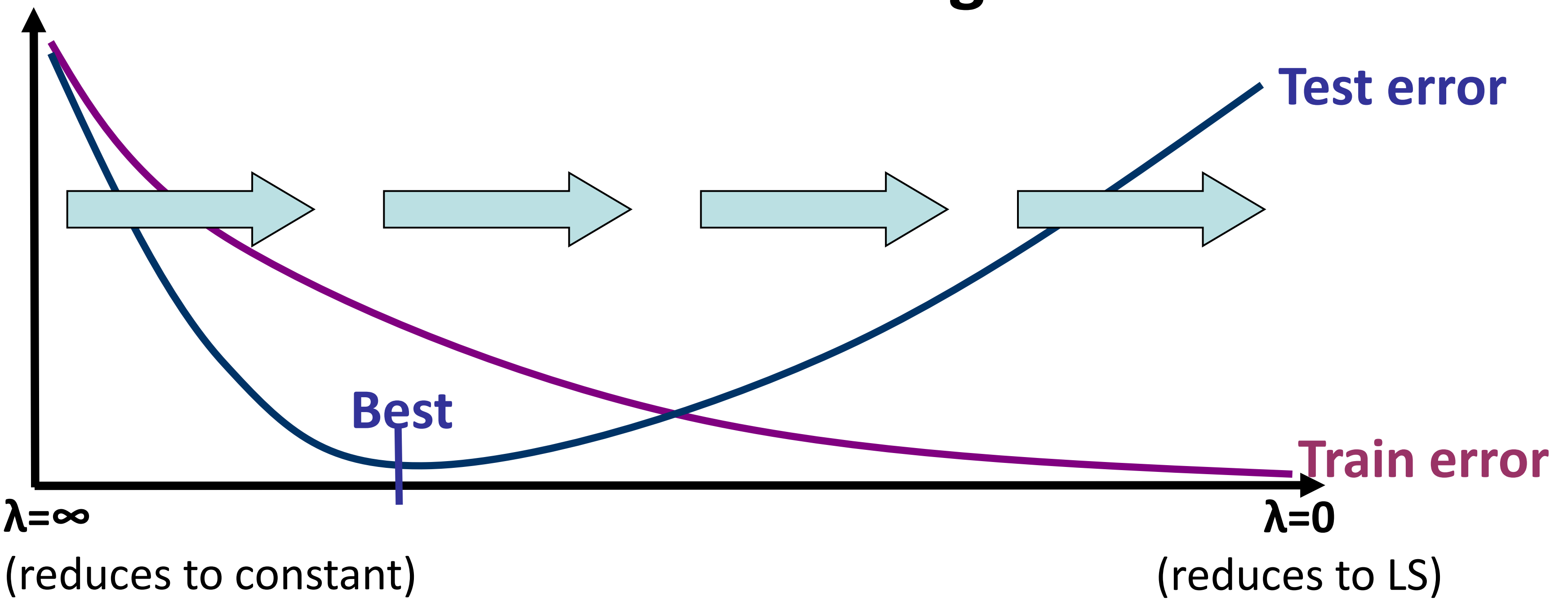
Regularized Linear Regression: How Coefficients Change With λ

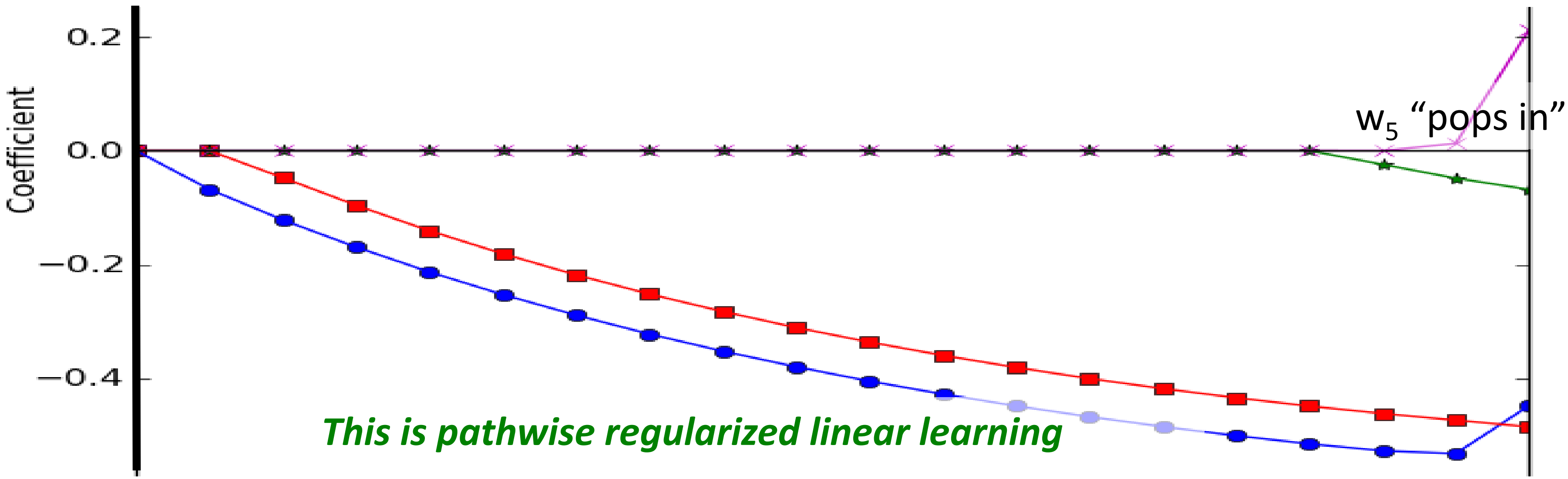




w_2 "pops in"

Regularized Linear Regression: How Coefficients Change With λ





Regularized Linear Regression

- **Cool property #3:** solving a full regularized path is ≈ as fast as solving single regularized problem
(or a least-squares learning problem)

Why fast:

Hot starts on local optimize

<u>**Algorithm**</u>

λ= huge (e.g. 1e40)

$\mathbf{w} = \mathbf{0}$

while $\lambda > 1e\text{-}10$

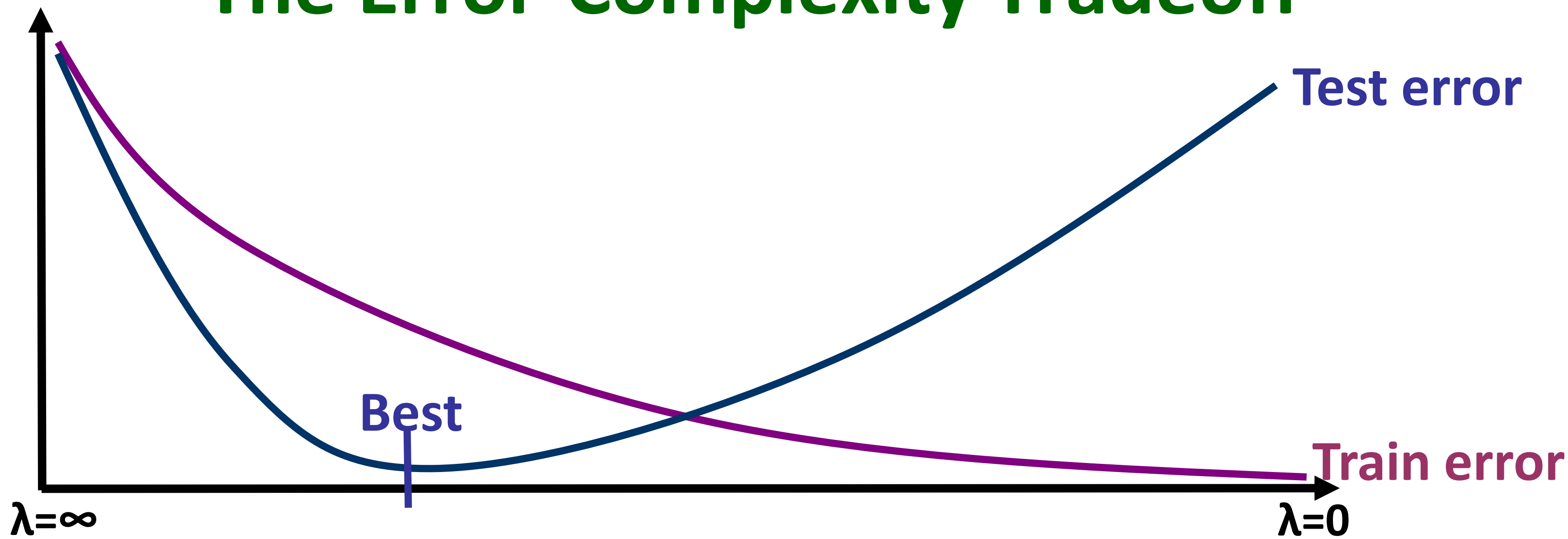
    $\lambda = \lambda / 10$

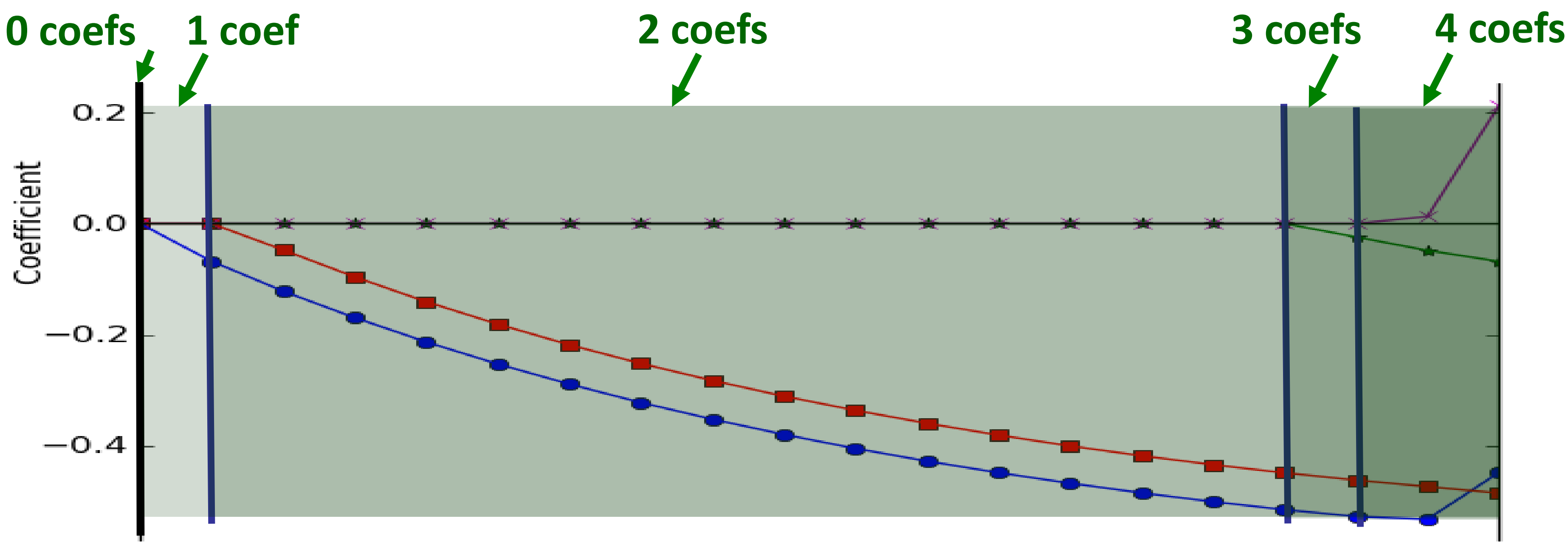
    $\mathbf{w}$ = solveAt($\mathbf{X}_{\mathbf{train}}$, $\mathbf{y}_{\mathbf{train}}$, λ, $\mathbf{w}_{\mathbf{init}}=\mathbf{w}$)

    Compute error on test set

Return **w** with best test error

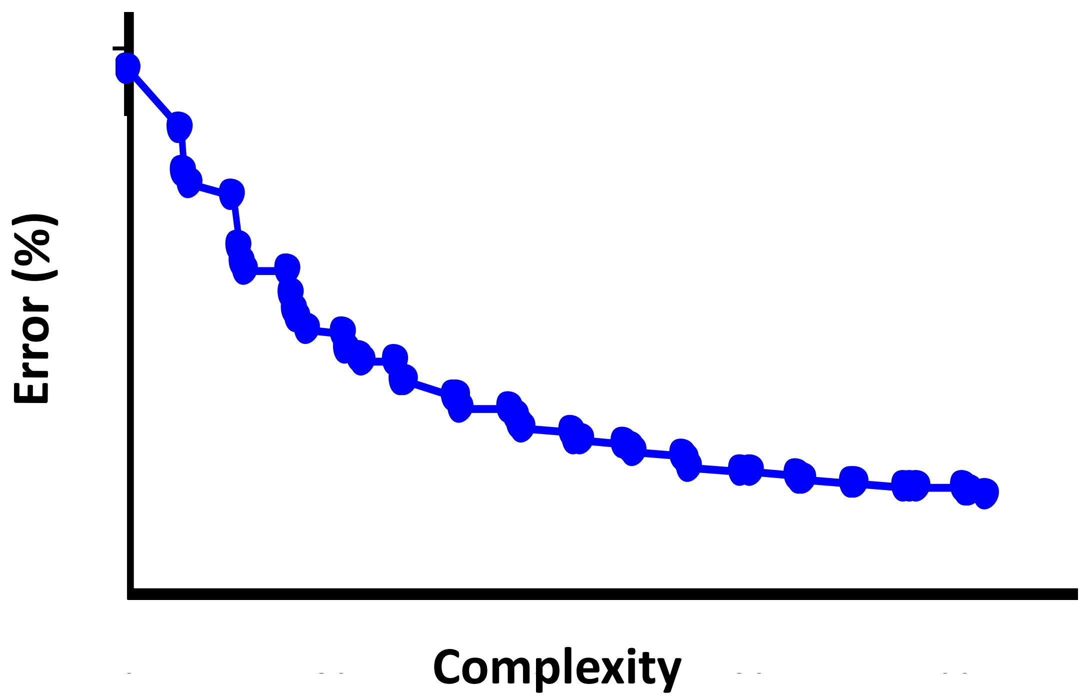
Regularized Linear Regression: The Error-Complexity Tradeoff





Regularized Linear Regression

- **Cool property #4:** solving a full regularized path gives us error-complexity tradeoffs!
  - train error versus # coefs (bases)
  - test error versus # coefs (bases)



Recap on Linear Regression

• Generalized linear models: **nonlinear basis functions** with linearly-learned coefficients!

Path-based Regularized Linear Regression:

• Can have more coefficients than samples! That is, can handle ***n* >> *N***!

    • **BHALR**: 1M basis functions for 1K samples

• Solving path is ≈ as **fast** as solving a least-squares learning problem! (Convex problem!)

• Solving path gives **error vs. complexity tradeoffs**!

One final trick:

• Can cast a **rational-learning** problem f(x)/(1+g(x)) as a linear-learning problem. See paper for details.

FFX: Fast Function Extraction Technology

FFX Step 1/3: GenerateBases()

Inputs: X #input training data
Outputs: B #list of bases

\# Generate univariate bases
$B_1 = \{\}$
for each input variable $v = \{x_1, x_2, \ldots\}$
for each exponent $exp = \{0.5, 1.0, 2.0\}$
let expression $b_{exp} = v^{exp}$
if ok(eval(b_{exp}, X))
add b_{exp} to B_1
for each operator $op = \{abs(), log_{10}, \ldots\}$
let expression $b_{op} = op(b_{exp})$
if ok(eval(b_{op}, X))
add b_{op} to B_1

\# Generate interacting-variable bases
$B_2 = \{\}$
for $i = 1$ to length(B_1)
let expression $b_i = B_1[i]$
for $j = 1$ to $i - 1$
let expression $b_j = B_1[j]$
if b_j is not an operator # disallow $op() * op()$
let expression $b_{inter} = b_i * b_j$
if ok(eval(b_{inter}, X))
add b_{inter} to B_2
return $B = B_1 \cup B_2$

"Replace linear bases with a crazy amount of nonlinear ones"

FFX Step 2/3: PathFollow() [using BHALR]

Inputs: X, y, B #input data, output data, bases
Outputs: A #list of coefficent-vectors

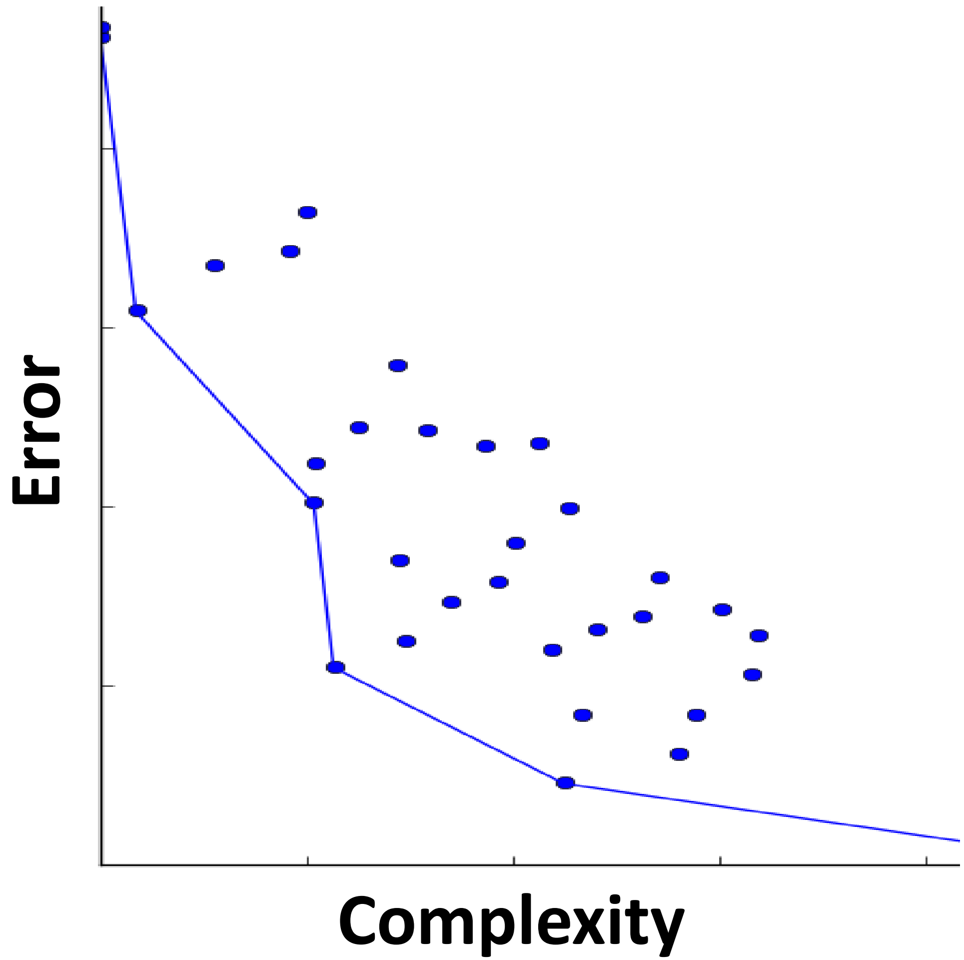
\# Compute X_B
for i = 1 to length(B)
    $X_B[\boldsymbol{i}]$ = eval($B[\boldsymbol{i}]$, X)

\# Generate $\boldsymbol{\lambda}_{vec}$ = range of λ values
$\lambda_{max} = max(|\boldsymbol{X}^T\boldsymbol{y}|)/(N * \rho)$
$\boldsymbol{\lambda}_{vec} = logspace(log_{10}(\lambda_{max} * eps), log_{10}(\lambda_{max}), N_\lambda)$

\# Main path-following
$\boldsymbol{A} = \{\}$
$N_{bases} = 0$
$i = 0$
$\boldsymbol{a} = \{0, 0, \ldots\}$
while $N_{bases} < N_{max-bases}$ and i < length($\boldsymbol{\lambda}_{vec}$)
    $\lambda = \boldsymbol{\lambda}_{vec}[\boldsymbol{i}]$
    $\boldsymbol{a}$ = elasticNetLinearFit($\boldsymbol{X}_B$, $\boldsymbol{y}$, λ, ρ, $\boldsymbol{a}$)
    N_{bases} = number of nonzero values in $\boldsymbol{a}$ (not counting offset)
    if $N_{bases} < N_{max-bases}$
        add $\boldsymbol{a}$ to $\boldsymbol{A}$
    $i = i + 1$
return $\boldsymbol{A}$

“Generate set of models, at increasing complexity”

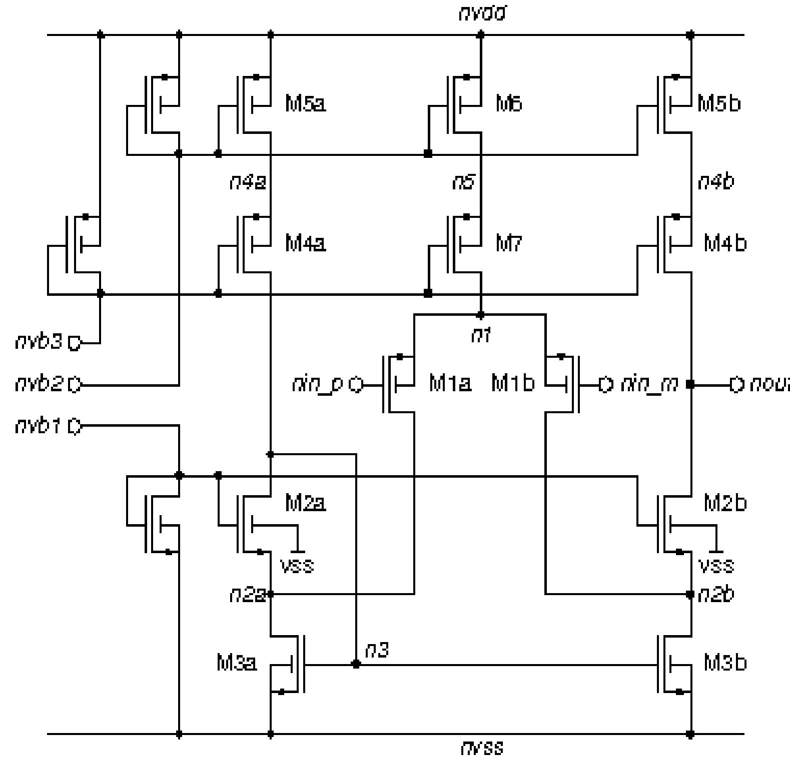
FFX Step 3/3: NondominatedFilter()



FFX Benchmarks

FFX Benchmarks: Same Setup as CAFFEINE

- High Speed amplifier
- 13 design variables
  - Vds, Vgs, Ids (operating-point driven formulation)
- orthogonal hypercube sampling
- 243 training samples
- 243 testing samples



FFX Step 1: The 176 Candidate 1-Variable Bases

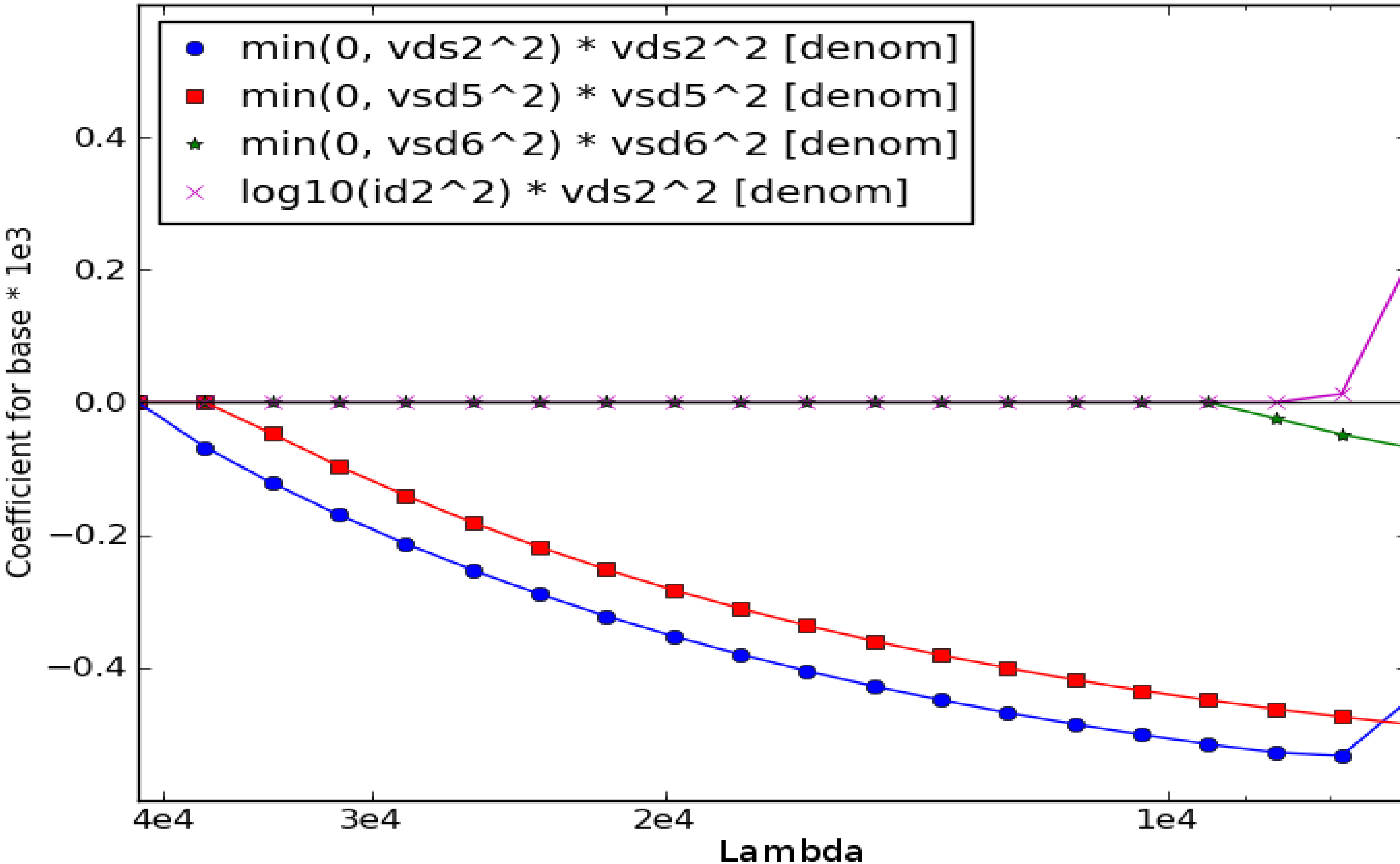
$v_{sg1}^{0.5},\ abs(v_{sg1}^{0.5}),\ max(0, v_{sg1}^{0.5}),\ min(0, v_{sg1}^{0.5}),\ log_{10}(v_{sg1}^{0.5}),\ v_{sg1},\ abs(v_{sg1}),\ max(0, v_{sg1}),\ min(0, v_{sg1}),$
$log_{10}(v_{sg1}),\ v_{sg1}^{2},\ max(0, v_{sg1}^{2}),\ min(0, v_{sg1}^{2}),\ log_{10}(v_{sg1}^{2}),\ v_{gs2}^{0.5},\ abs(v_{gs2}^{0.5}),\ max(0, v_{gs2}^{0.5}),\ min(0, v_{gs2}^{0.5}),$
$log_{10}(v_{gs2}^{0.5}),\ v_{gs2},\ abs(v_{gs2}),\ max(0, v_{gs2}),\ min(0, v_{gs2}),\ log_{10}(v_{gs2}),\ v_{gs2}^{2},\ max(0, v_{gs2}^{2}),\ min(0, v_{gs2}^{2}),$
$log_{10}(v_{gs2}^{2}),\ v_{ds2}^{0.5},\ abs(v_{ds2}^{0.5}),\ max(0, v_{ds2}^{0.5}),\ min(0, v_{ds2}^{0.5}),\ log_{10}(v_{ds2}^{0.5}),\ v_{ds2},\ abs(v_{ds2}),\ max(0, v_{ds2}),$
$min(0, v_{ds2}),\ log_{10}(v_{ds2}),\ v_{ds2}^{2},\ max(0, v_{ds2}^{2}),\ min(0, v_{ds2}^{2}),\ log_{10}(v_{ds2}^{2}),\ v_{sg3}^{0.5},\ abs(v_{sg3}^{0.5}),\ max(0, v_{sg3}^{0.5}),$
$min(0, v_{sg3}^{0.5}),\ log_{10}(v_{sg3}^{0.5}),\ v_{sg3},\ abs(v_{sg3}),\ max(0, v_{sg3}),\ min(0, v_{sg3}),\ log_{10}(v_{sg3}),\ v_{sg3}^{2},\ max(0, v_{sg3}^{2}),$
$min(0, v_{sg3}^{2}),\ log_{10}(v_{sg3}^{2}),\ v_{sg4}^{0.5},\ abs(v_{sg4}^{0.5}),\ max(0, v_{sg4}^{0.5}),\ min(0, v_{sg4}^{0.5}),\ log_{10}(v_{sg4}^{0.5}),\ v_{sg4},\ abs(v_{sg4}),$
$max(0, v_{sg4}),\ min(0, v_{sg4}),\ log_{10}(v_{sg4}),\ v_{sg4}^{2},\ max(0, v_{sg4}^{2}),\ min(0, v_{sg4}^{2}),\ log_{10}(v_{sg4}^{2}),\ v_{sg5}^{0.5},\ abs(v_{sg5}^{0.5}),$
$max(0, v_{sg5}^{0.5}),\ min(0, v_{sg5}^{0.5}),\ log_{10}(v_{sg5}^{0.5}),\ v_{sg5},\ abs(v_{sg5}),\ max(0, v_{sg5}),\ min(0, v_{sg5}),\ log_{10}(v_{sg5}),\ v_{sg5}^{2},$
$max(0, v_{sg5}^{2}),\ min(0, v_{sg5}^{2}),\ log_{10}(v_{sg5}^{2}),\ v_{sd5}^{0.5},\ abs(v_{sd5}^{0.5}),\ max(0, v_{sd5}^{0.5}),\ min(0, v_{sd5}^{0.5}),\ log_{10}(v_{sd5}^{0.5}),\ v_{sd5},$
$abs(v_{sd5}),\ max(0, v_{sd5}),\ min(0, v_{sd5}),\ log_{10}(v_{sd5}),\ v_{sd5}^{2},\ max(0, v_{sd5}^{2}),\ min(0, v_{sd5}^{2}),\ log_{10}(v_{sd5}^{2}),$
$v_{sd6}^{0.5},\ abs(v_{sd6}^{0.5}),\ max(0, v_{sd6}^{0.5}),\ min(0, v_{sd6}^{0.5}),\ log_{10}(v_{sd6}^{0.5}),\ v_{sd6},\ abs(v_{sd6}),\ max(0, v_{sd6}),\ min(0, v_{sd6}),$
$log_{10}(v_{sd6}),\ v_{sd6}^{2},\ max(0, v_{sd6}^{2}),\ min(0, v_{sd6}^{2}),\ log_{10}(v_{sd6}^{2}),\ i_{d1},\ abs(i_{d1}),\ max(0, i_{d1}),\ min(0, i_{d1}),\ i_{d1}^{2},$
$max(0, i_{d1}^{2}),\ min(0, i_{d1}^{2}),\ log_{10}(i_{d1}^{2}),\ i_{d2}^{0.5},\ abs(i_{d2}^{0.5}),\ max(0, i_{d2}^{0.5}),\ min(0, i_{d2}^{0.5}),\ log_{10}(i_{d2}^{0.5}),\ i_{d2},\ abs(i_{d2}),$
$max(0, i_{d2}),\ min(0, i_{d2}),\ log_{10}(i_{d2}),\ i_{d2}^{2},\ max(0, i_{d2}^{2}),\ min(0, i_{d2}^{2}),\ log_{10}(i_{d2}^{2}),\ i_{b1}^{0.5},\ abs(i_{b1}^{0.5}),\ max(0, i_{b1}^{0.5}),$
$min(0, i_{b1}^{0.5}),\ log_{10}(i_{b1}^{0.5}),\ i_{b1},\ abs(i_{b1}),\ max(0, i_{b1}),\ min(0, i_{b1}),\ log_{10}(i_{b1}),\ i_{b1}^{2},\ max(0, i_{b1}^{2}),\ min(0, i_{b1}^{2}),$
$log_{10}(i_{b1}^{2}),\ i_{b2}^{0.5},\ abs(i_{b2}^{0.5}),\ max(0, i_{b2}^{0.5}),\ min(0, i_{b2}^{0.5}),\ log_{10}(i_{b2}^{0.5}),\ i_{b2},\ abs(i_{b2}),\ max(0, i_{b2}),\ min(0, i_{b2}),$
$log_{10}(i_{b2}),\ i_{b2}^{2},\ max(0, i_{b2}^{2}),\ min(0, i_{b2}^{2}),\ log_{10}(i_{b2}^{2}),\ i_{b3}^{0.5},\ abs(i_{b3}^{0.5}),\ max(0, i_{b3}^{0.5}),\ min(0, i_{b3}^{0.5}),\ log_{10}(i_{b3}^{0.5}),$
$i_{b3},\ abs(i_{b3}),\ max(0, i_{b3}),\ min(0, i_{b3}),\ log_{10}(i_{b3}),\ i_{b3}^{2},\ max(0, i_{b3}^{2}),\ min(0, i_{b3}^{2}),\ log_{10}(i_{b3}^{2})$

FFX Step 1: Some Candidate 2-Variable Bases (3374 total)

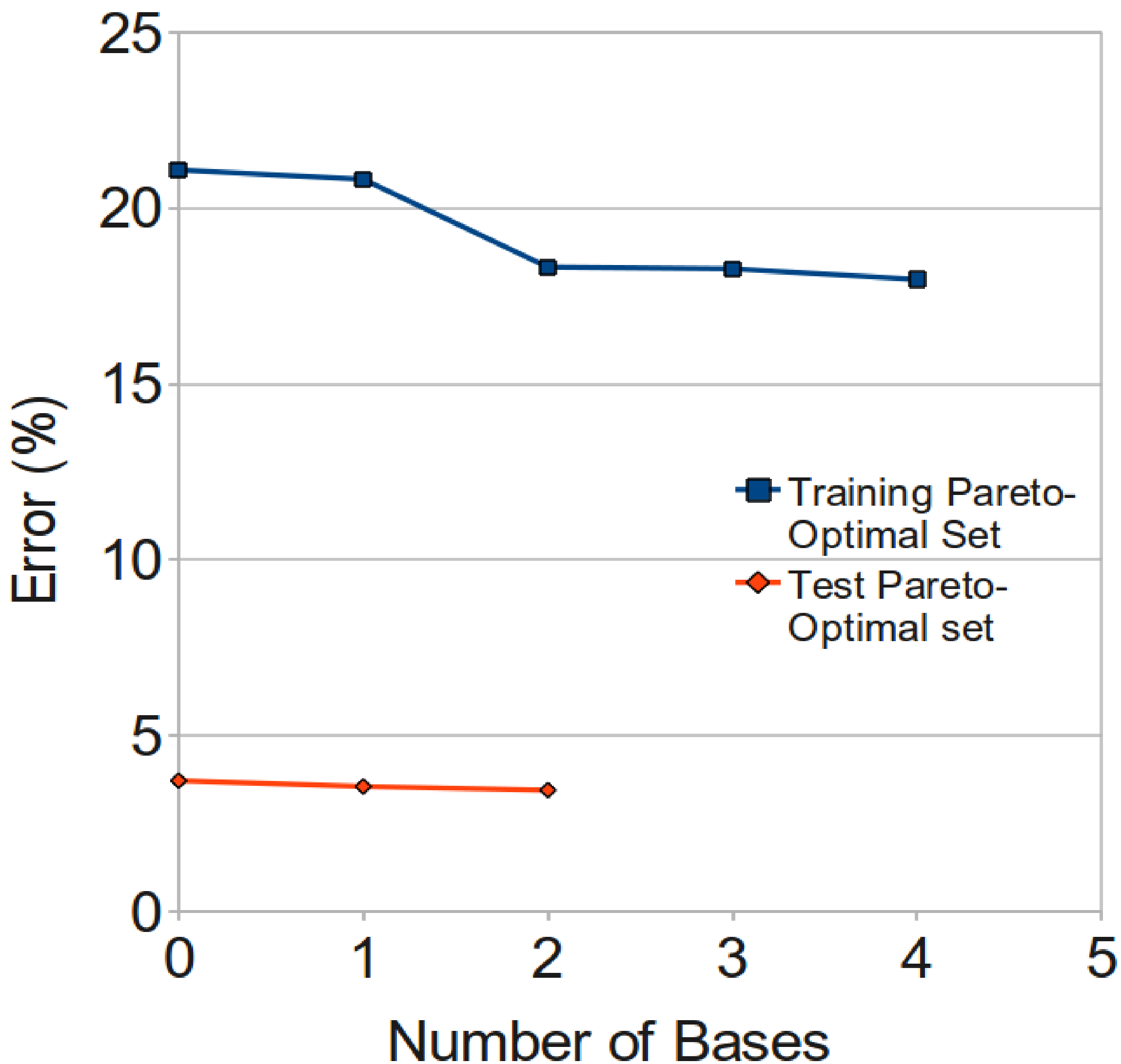
$log_{10}(i_{b3}^2) * i_{d2}^2, log_{10}(i_{b3}^2) * i_{b1}^{0.5}, log_{10}(i_{b3}^2) * i_{b1}, log_{10}(i_{b3}^2) * i_{b1}^2, log_{10}(i_{b3}^2) * i_{b2}^{0.5}, log_{10}(i_{b3}^2) * i_{b2}, log_{10}(i_{b3}^2) * i_{b2}^2, log_{10}(i_{b3}^2) * i_{b3}^{0.5}, log_{10}(i_{b3}^2) * i_{b3}, log_{10}(i_{b3}^2) * i_{b3}^2$

(and 3364 more)

FFX Step 2: PathFollow: First Four Bases (A_{LF} problem)



FFX Step 3: Nondominated Filter Error vs. # Bases (A_{LF} problem)



FFX Step 3: Final Pareto-Optimal Set

Total Runtime <5 s (1 GHz CPU)

This is Fast Function Extraction

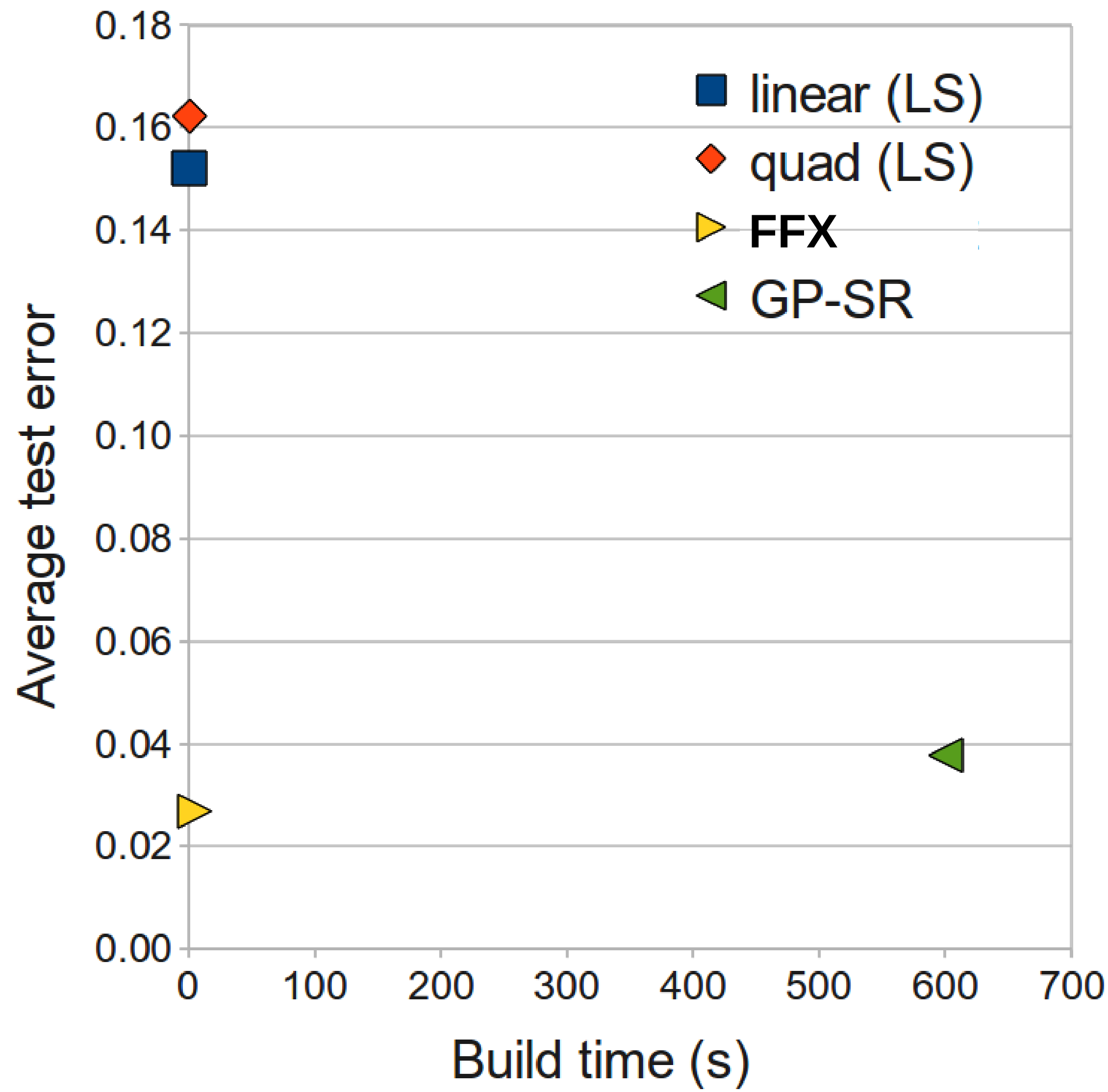
Test error (ϵ_{test}) (%)	**Extracted Function**
3.72	37.619
3.55	$\frac{37.379}{1.0 - 6.78e\text{-}5 * min(0, v_{ds2}^2) * v_{ds2}^2}$
3.45	$\frac{37.020}{1.0 - 1.22e\text{-}4 * min(0, v_{ds2}^2) * v_{ds2}^2 - 4.72e\text{-}5 * min(0, v_{sd5}^2) * v_{sd5}^2}$

FFX Functions with Lowest Test Error on 6 Different Problems.

Problem	**Test error** (ϵ_{test}) (%)	**Extracted Function**
A_{LF}	3.45	$\frac{37.020}{1.0-1.22e\text{-}4*min(0,v_{ds2}^2)*v_{ds2}^2-4.72e\text{-}5*min(0,v_{sd5}^2)*v_{sd5}^2}$
PM	1.51	$\frac{90.148}{1.0-8.79e\text{-}6*min(0,v_{sg1}^2)*v_{sg1}^2+2.28e\text{-}6*min(0,v_{ds2}^2)*v_{ds2}^2}$
SR_n	2.10	$\frac{-5.21e7}{1.0-8.22e\text{-}5*min(0,v_{gs2}^2)*v_{gs2}^2}$
SR_p	4.74	$2.35e7$
V_{offset}	2.16	$-0.0020 - 1.22e\text{-}23 * min(0, v_{gs2}^2) * v_{gs2}^2$
$log_{10}(f_u)$	2.17	$0.74 - 1.10e\text{-}5 * min(0, v_{sg1}^2) * v_{sg1}^2$ $+1.88e\text{-}5 * min(0, v_{ds2}^2) * v_{ds2}^2$

Compare FFX vs. GP-SR

Average test time & build errors over 6 problems



Scaling Up FFX?

FFX So Far

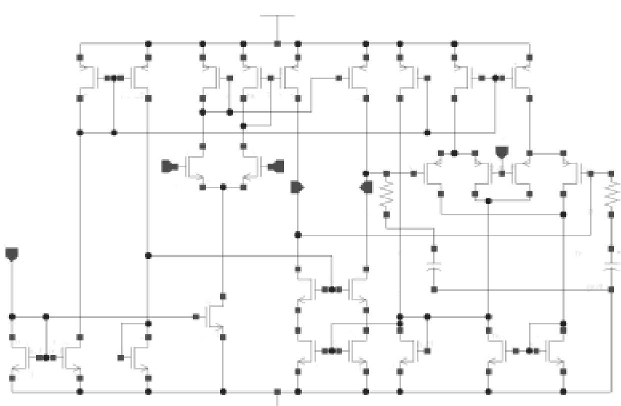
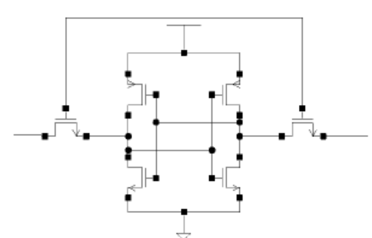
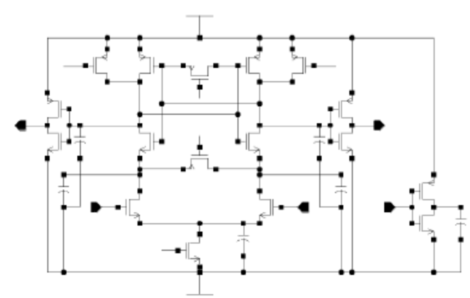
- Problems: 13 input variables, 256 samples
- Results: <5 s, best error
- Pretty good!

- What about 100-1000+ input variables…?

12 Larger Problems
Up to 1468 input variables

Circuit	# Devices	# Process variables	Outputs Modeled
opamp	30	215	AV (gain), BW (bandwidth), PM (phase margin), SR (slew rate)
bitcell	6	30	$cell_i$ (read current)
sense amp	12	125	$delay$, pwr (power)
voltage reference	11	105	$DVREF$ (difference in voltage), PWR (power)
GMC filter	140	1468	$ATTEN$ (attenuation), IL
comparator	62	639	BW (bandwidth)

The opamp and voltage reference had 800 Monte Carlo sample points, the comparator and GMC filter 2000, and bitcell and sense amp 5000.

Other Approaches on 30T Opamp Problems

(215 input vars.) [McConaghy GPTP 2009]

Problem	GP (CAFF-EINE)	Boost tree (SGB)	Bootstr. tree (RF)	LVSR-GDR	LVSR-GDR-tune
30T AV	$\gg$10.0	0.6418	0.8183	0.0765	0.1073
30T BW	$\gg$10.0	0.5686	0.7730	0.0378	0.0442
30T PM	$\gg$10.0	0.5894	0.7656	0.0732	0.0693
30T SR	$\gg$10.0	0.5208	0.7436	0.1642	0.1403

- A “direct” GP-SR approach did terrible
- Resorted to a latent-variable SR approach for good results

Scaling Up FFX

- What about 100-1000 input variables…?
- Summary of results:
  - *Out of memory*
  - *Time for some theory…*

Computational Complexity of FFX?

- **Step one.** Let e be the number of exponents and o be the number of nonlinear operators. Therefore the number of univariate bases per variable is $(o + 1) * e$. (The $+1$ is when no nonlinear operator is applied; or, equivalently, unity). With n as the number of input variables, then the total number of univariate bases is $(o+1)*e*n$. With N samples, the univariate part of step one has a complexity of $O((o + 1) * e * n * N)$. Since e and o are constants, this reduces to $O(n * N)$. The number of bivariate bases is $p = O(n^2)$, so the bivarate part of step one has complexity $O(n^2 * N)$.

Computational Complexity of FFX?

- **Step two.** Elastic net path-following is the dominant part. The cost of an older elastic-net learning technique, LARS, was approximately that of one least-squares (LS) fitting according to p.93 of (Hastie et al., 2008). Since then, the coordinate descent algorithm (Friedman et al., 2010) has been shown to be 10x faster. Nonetheless, we will use LS as a baseline. With p input variables, LS fitting with QR decomposition has complexity $O(N * p^2)$. Because $p = O(n^2)$, FFX has approximate complexity $O(N * n^4)$.

Computational Complexity of FFX?

- **Step three.** Reference (Deb et al., 2002) shows that nondominated filtering has complexity $O(N_o * N_{nondom})$ where N_o is the number of objectives, and N_{nondom} is the number of nondominated individuals. In the SR cases, N_o is a constant (at 2) and $N_{nondom} \leq N_{max-bases}$ where $N_{max-bases}$ is a constant (≈ 5). Therefore, FFX step three complexity is $O(1)$.

The complexity of FFX is the maximum of steps one, two, and three, which is ***$O(N * n^4)$***.

 # samples

 # input variables

Improving FFX

A batch-style riff on MARS.

Revised FFX Algorithm:

1. Learn univariate coefficients
2. Only combine the $k \leq O(\sqrt{n})$ most important basis functions
3. Pathwise-learn univariate & combination
4. Nondominated filter

Complexity down to **$O(N{*}n^2)$** !

Improving FFX

A batch-style riff on MARS.

Revised FFX Algorithm:

1. Learn univariate coefficients
2. Only combine the $k \leq O(\sqrt{n})$ most important basis functions
3. Pathwise-learn univariate & combination
4. Nondominated filter

Complexity down to **$O(N*n^2)$** !

Improving Complexity to *$O(N*n^2)$:*

A batch-style riff on MARS.

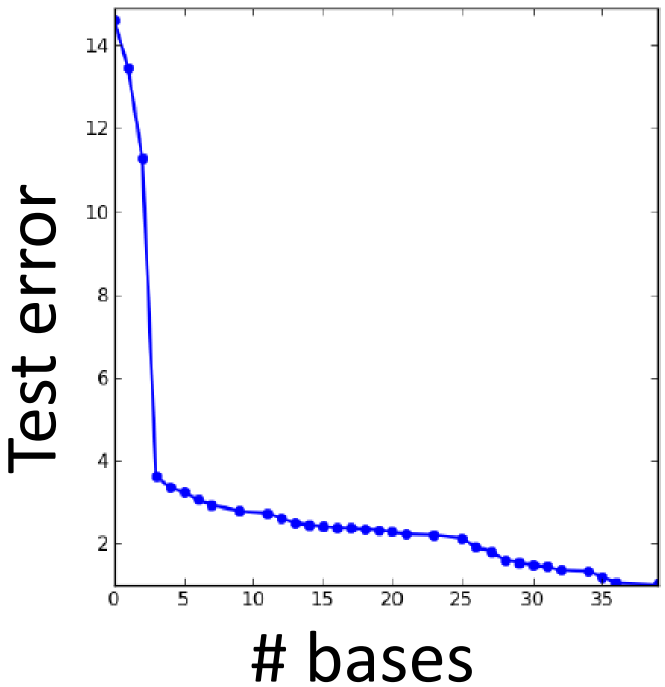
Revised algorithm:

1. First learn univariate coefficients
2. Only combine the $k \leq O(\sqrt{n})$ most important basis functions
3. Pathwise-learn univariate & combination
4. Nondominated filter

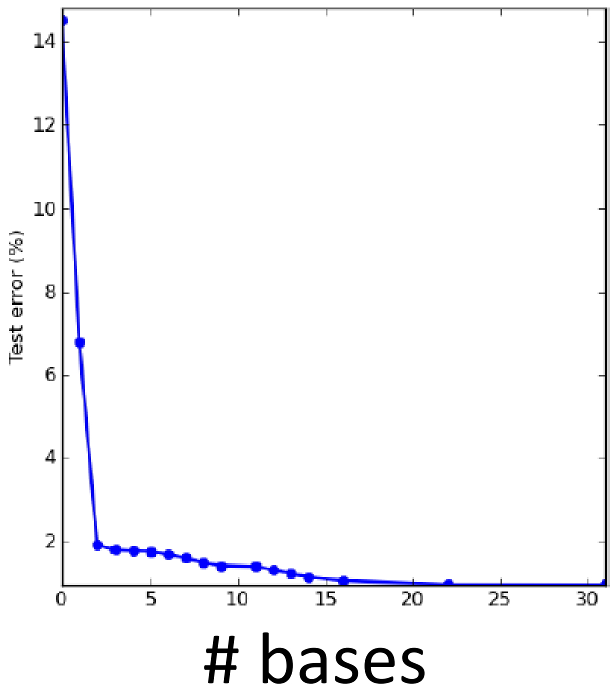
Complexity down to $O(N*n^2)$!

Overall runtime 5-30 s

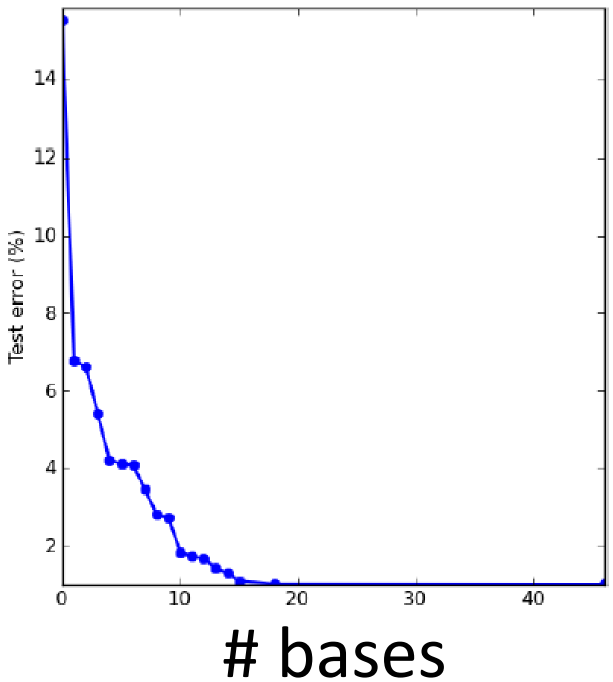
Test Error vs. Complexity
Large Problems 1-3 (of 12). <30 s!



Opamp AV

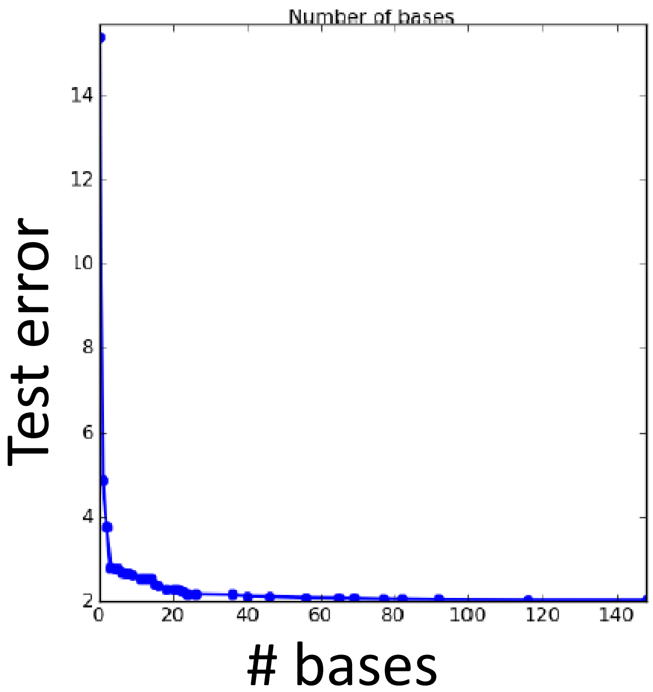


Opamp BW

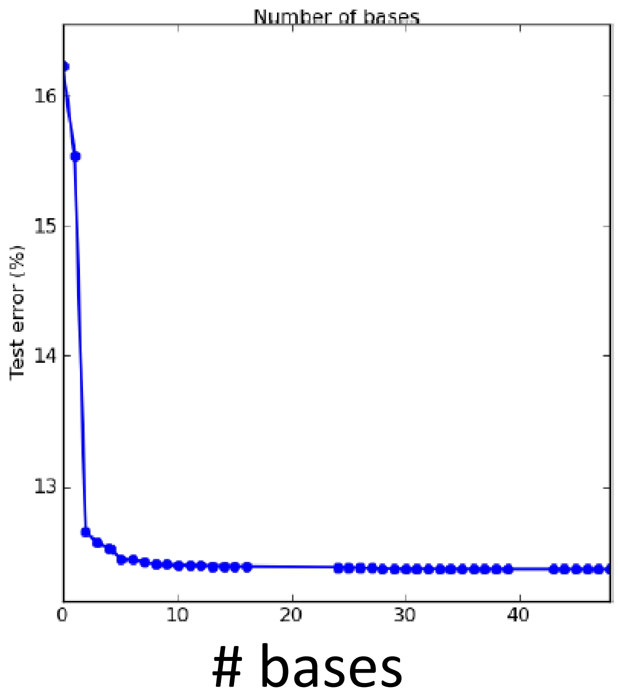


Opamp PM

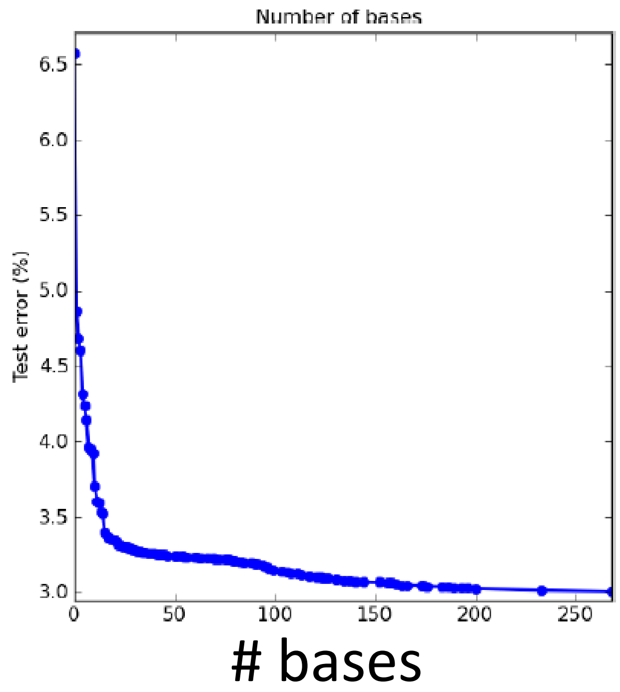
Test Error vs. Complexity
Large Problems 4-6 (of 12). <30 s!



Opamp SR

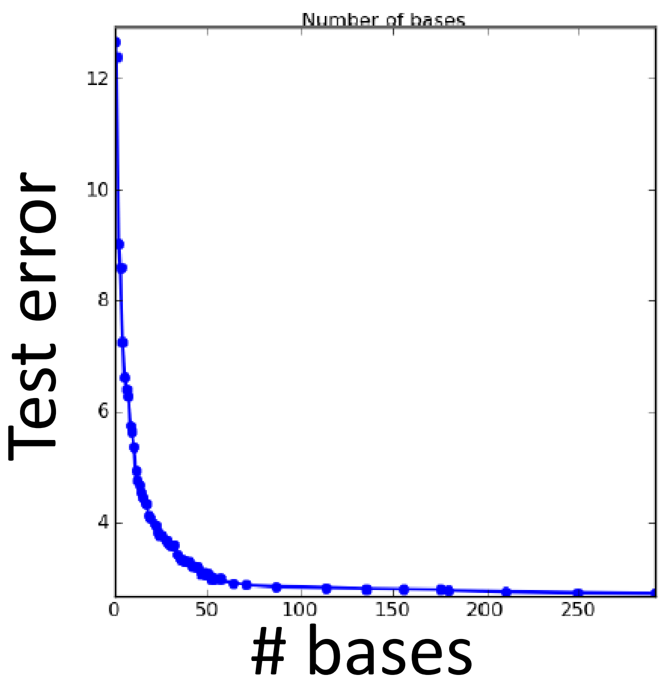


Bitcell cell_i

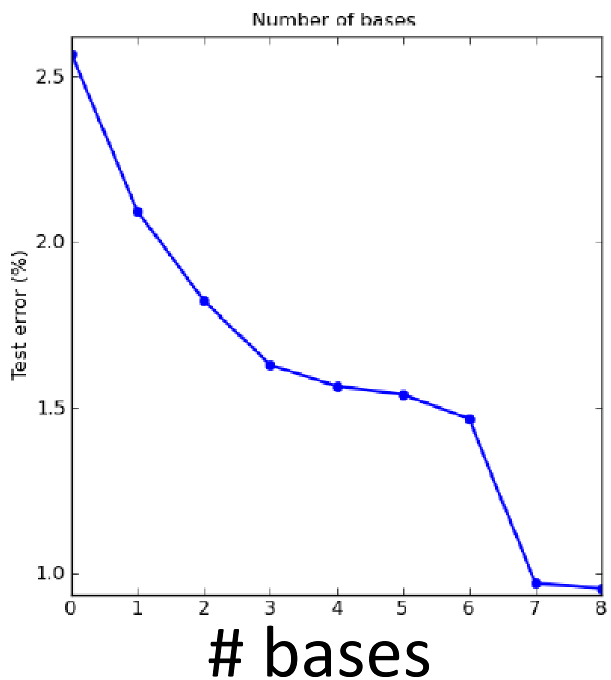


Sense amp delay

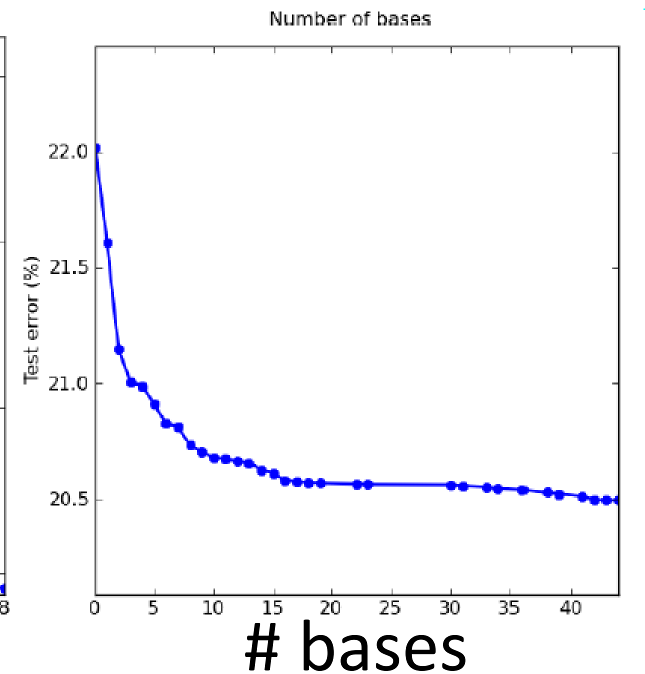
Test Error vs. Complexity
Large Problems 7-9 (of 12). <30 s!



Sense amp PWR



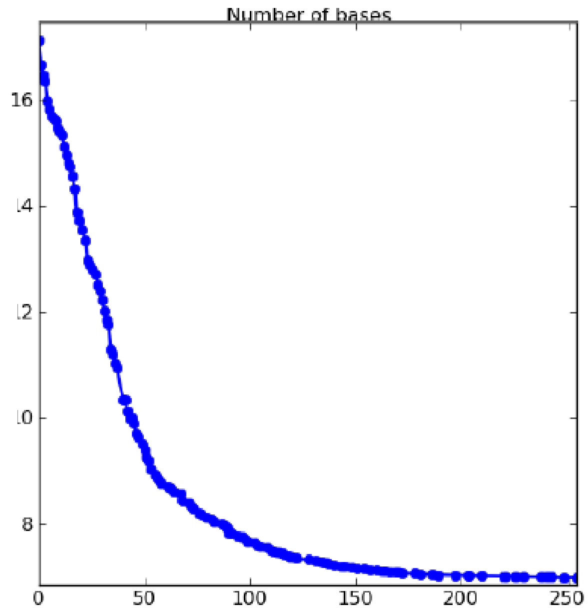
Voltage reference DVREF



Voltage reference power

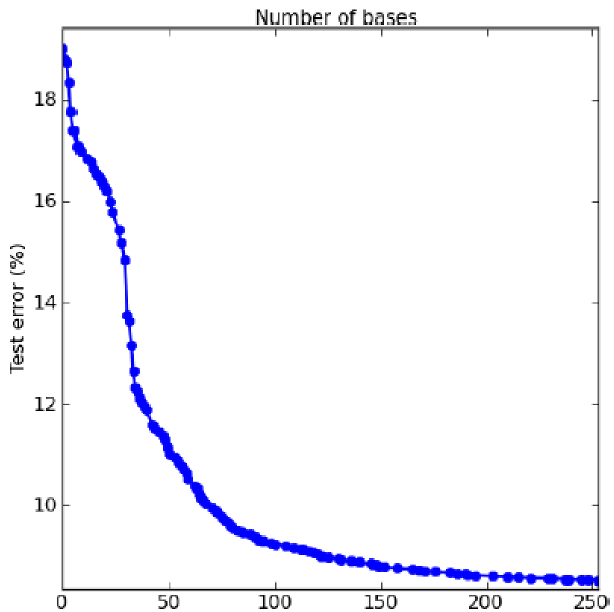
Test Error vs. Complexity
Large Problems 10-12 (of 12). <30 s!

Test error



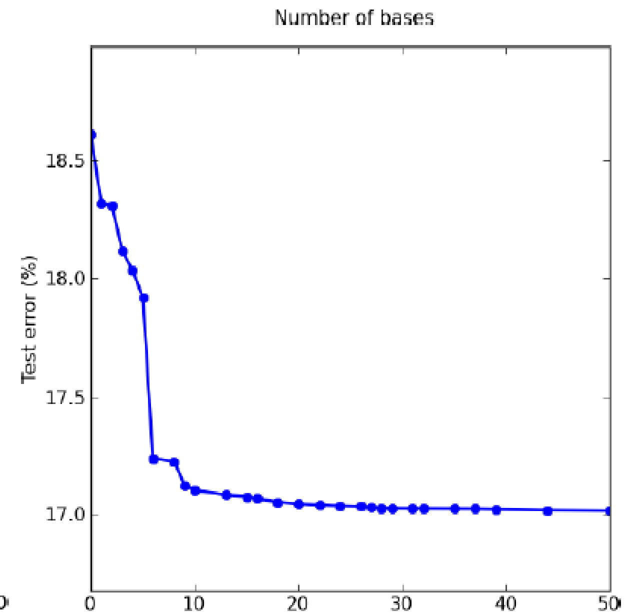
bases

GMC filter IL



bases

GMC filter ATTEN



bases

Comparator BW

Opamp PM Equations. <30 s!

# Bases	Test error (ϵ_{test}) (%)	Extracted Function
0	15.5	59.6
1	6.8	$59.6 - 0.303 * dxl$
2	6.6	$59.6 - 0.308 * dxl - 0.00460 * cgop$
3	5.4	$59.6 - 0.332 * dxl - 0.0268 * cgop + 0.0215 * dvthn$
4	4.2	$59.6 - 0.353 * dxl - 0.0457 * cgop + 0.0403 * dvthn - 0.0211 * dvthp$
5	4.1	$59.6 - 0.354 * dxl - 0.0460 * cgop - 0.0217 * dvthp + 0.0198 * dvthn + 0.0134 * abs(dvthn) * dvthn$
6	4.07	$59.6 - 0.354 * dxl - 0.0466 * cgop - 0.0224 * dvthp + 0.0202 * dvthn + 0.0135 * abs(dvthn) * dvthn + 0.000550 * DXL$
$\vdots$	$\vdots$	$\vdots$
46	1.0	$(58.9 - 0.136 * dxl + 0.0299 * dvthn - 0.0194 * max(0, 0.784 - dvthn) + \ldots)/(1.0 + \ldots)$

Voltage Reference DVREF. <30 s!

# Bases	Test error (ϵ_{test}) (%)	Extracted Function
0	2.6	512.7
1	2.1	$504/(1.0 + 0.121 * max(0, dvthn + 0.875))$
2	1.8	$503 - 199 * max(0, dvthn + 1.61) - 52.1 * max(0, dvthn + 0.875)$
3	1.6	$496/(1.0 - 0.0447 * max(0, -1.64 - dvthp) * max(0, dvthn + 0.875) - 0.0282 * max(0, -1.90 - dxw) * max(0, dvthn + 0.875) - 0.0175 * max(0, -1.64 - dvthp) * max(0, dvthn + 0.142))$
$\vdots$	$\vdots$	$\vdots$
8	0.9	$476/(1.0 + 0.105 * max(0, dvthn + 1.61) - 0.0397 * max(0, -1.64 - dvthp) * max(0, dvthn + 0.875) - 0.0371 * max(0, -1.90 - dxw) * max(0, dvthn + 0.875) - 0.0151 * max(0, -1.64 - dvthp) * max(0, dvthn + 0.142) \ldots)$

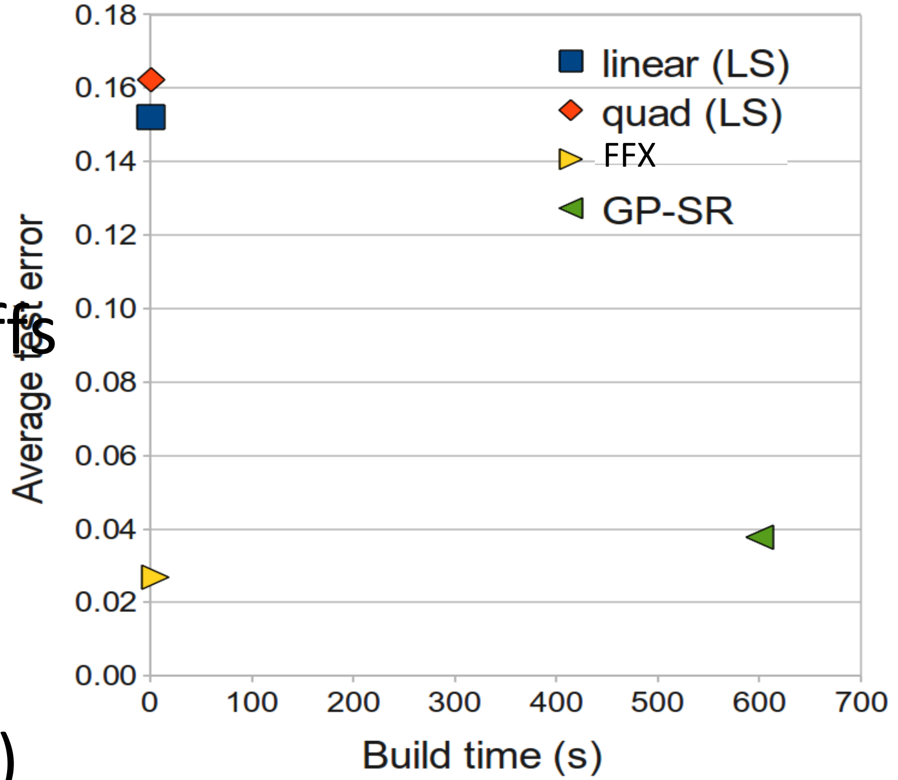
Outline

- Introduction
- Background
- FFX: Fast Function Extraction
- Results
- Scaling Higher?
- <u>Discussion</u>

FFX Summary of Results 1/2

- ≈ as fast as LS-linear:

  <5 s on smaller, <30 s on larger
- As accurate as GP-SR
- Gives error-complexity tradeoffs
- Scalable
- Simple
- Deterministic!
- O(N * n^2) complexity. (Theory!)
- *Massively shallow learning.*



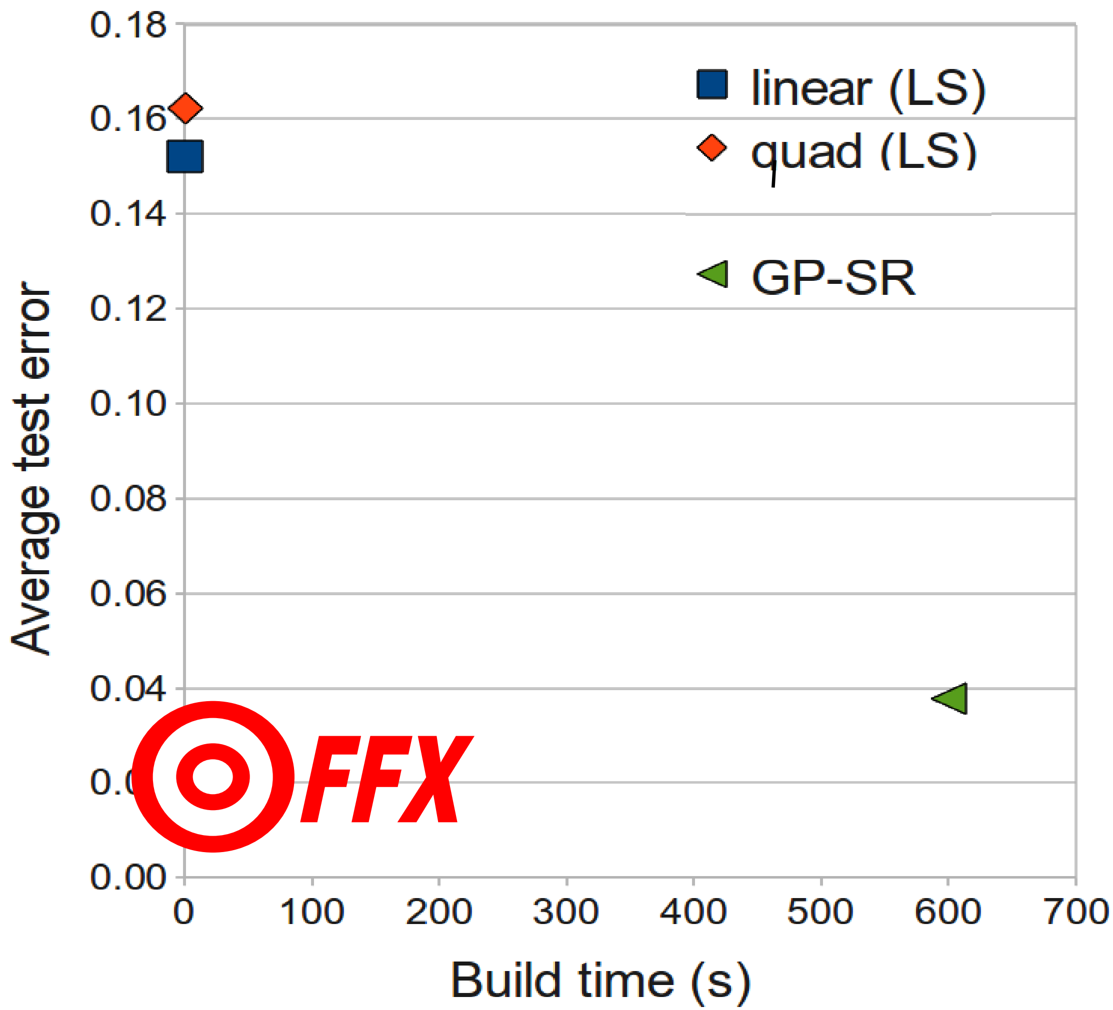
This is Fast Function Extraction

FFX Summary of Results 2/2

- Has been deployed to industry since 2010
- Off-the-shelf, under-the-hood, no fuss
- Solved >10,000 problems in just one application (Solido HSMC)
- Adopted by others in their research with great success (e.g. De Jonghe, Maricau)
- Now 100K+ variables, 100-10K training pts
- Extended for classification too (beat out 20+ other approaches)

FFX ≠ Fork Fan Experience

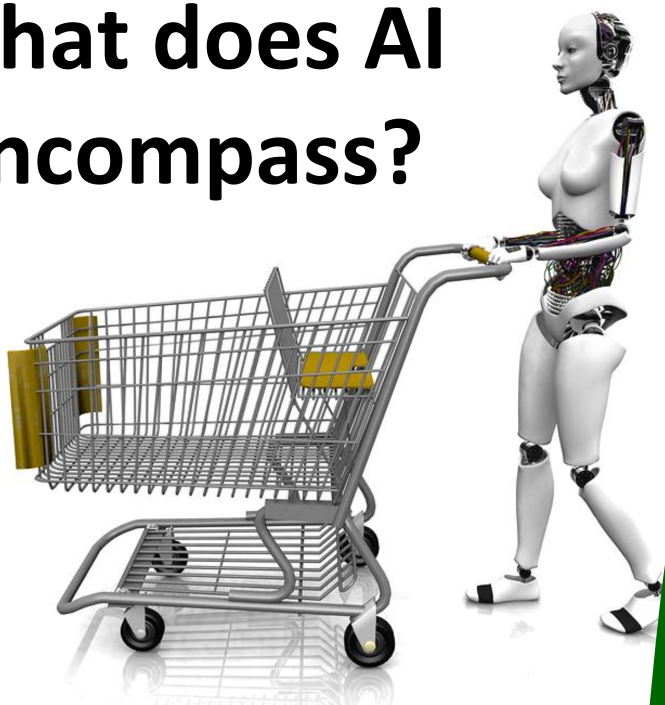
FFX is SR *Technology*: Fast, Scalable, Deterministic



Code is online at trent.st/ffx

Conclusion

What does AI encompass?



WTF is genetic programming or symbolic regression? Why should I care?

Is Deep Learning cool or what?

How *does* Google find furry robots?