CPSC 340: Machine Learning and Data Mining

Gradient Descent

Fall 2019

Last Time: Change of Basis

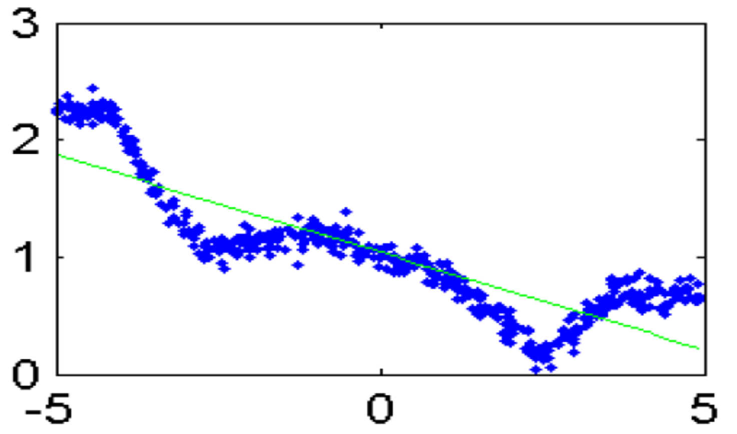
- Last time we discussed change of basis:
  - E.g., polynomial basis:

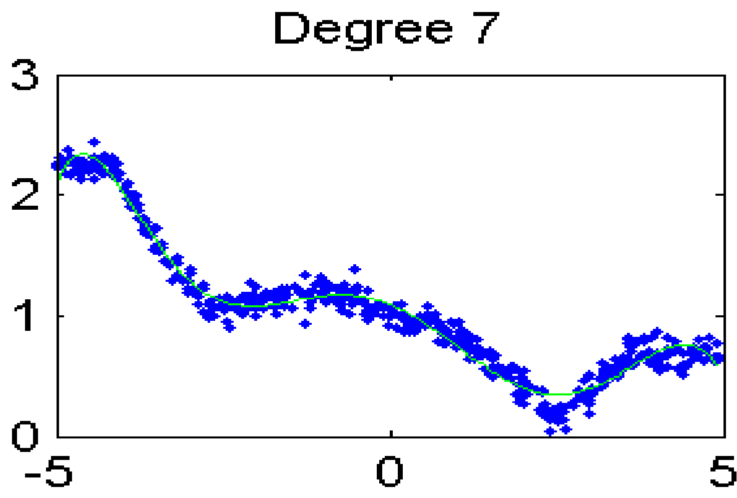
Replace $X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ with $Z = \begin{bmatrix} 1 & x_1 & (x_1)^2 & \cdots & (x_1)^p \\ 1 & x_2 & (x_2)^2 & \cdots & (x_2)^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & (x_n)^2 & \cdots & (x_n)^p \end{bmatrix}$

  - You can fit non-linear models with linear regression.

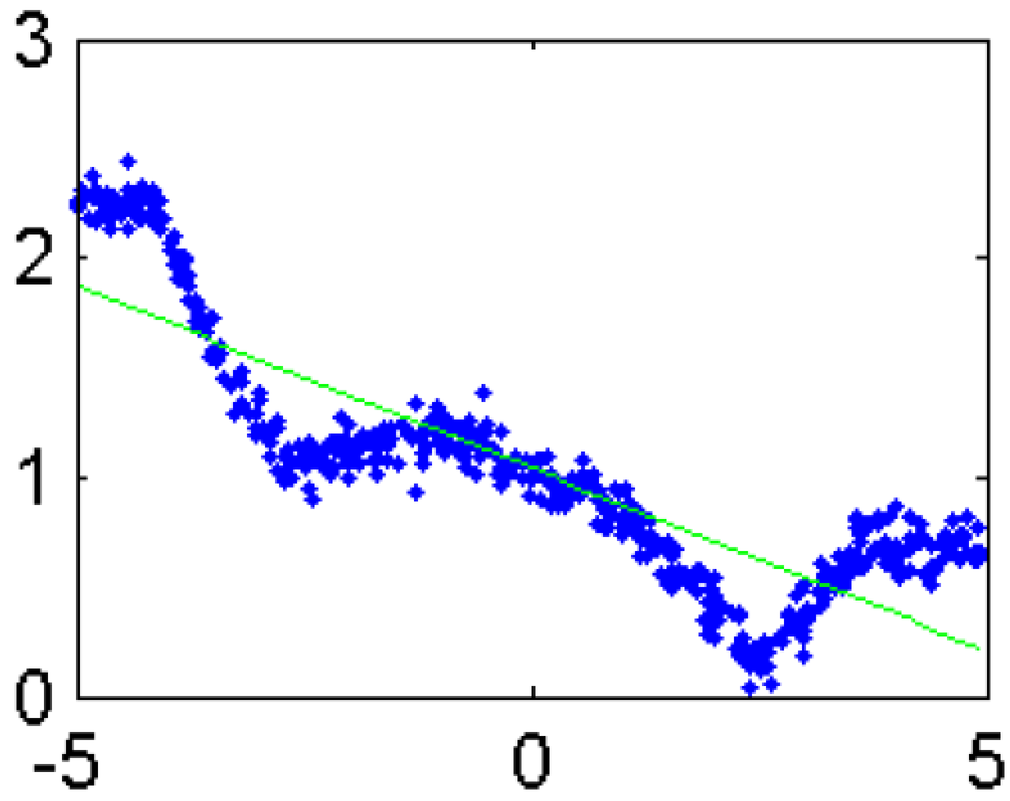
$$\hat{y}_i = v^T z_i = w_0 + w_1 x_i + w_2 x_i^2 + w_3 x_i^3 + \cdots + w_p x_i^p$$

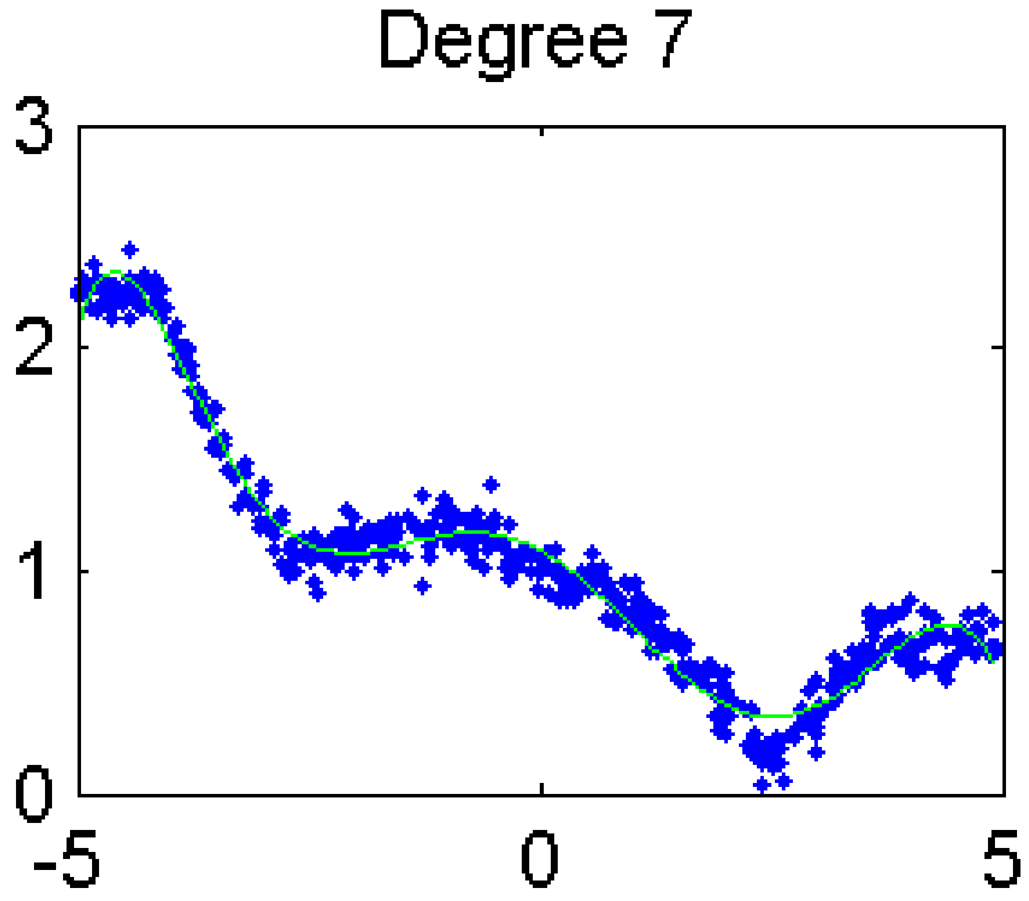
  - Just treat 'Z' as your data, then fit linear model.





General Polynomial Features





- If you have more than one feature, you can include interactions:
  - With p=2, in addition to $(x_{i1})^2$ and $(x_{i2})^2$ you could include $x_{i1}x_{i2}$.

"Change of Basis" Terminology

- Instead of "nonlinear feature transform", in machine learning it is common to use the expression "change of basis".
  - The z_i are the "coordinates in the new basis" of the training example.

- "Change of basis" means something different in math:
  - Math: basis vectors must be linearly independent (in ML we don't care).
  - Math: change of basis must span the same space (in ML we change space).

- Unfortunately, saying "change of basis" in ML is common.
  - When I say "change of basis", just think "nonlinear feature transform".

Change of Basis Notation (MEMORIZE)

- Linear regression with original features:
  - We use 'X' as our "n by d" data matrix, and 'w' as our parameters.
  - We can find d-dimensional 'w' by minimizing the squared error:

$$f(w) = \frac{1}{2}\|Xw - y\|^2$$

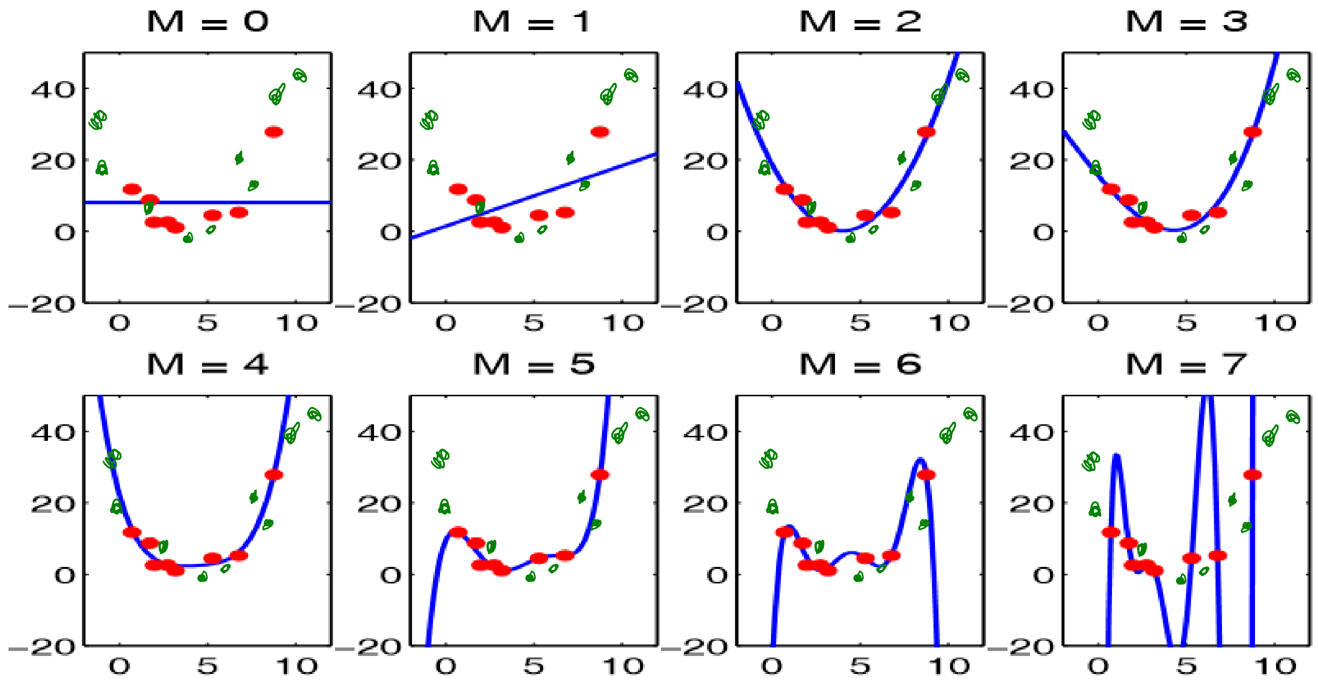
- Linear regression with nonlinear feature transforms:
  - We use 'Z' as our "n by k" data matrix, and 'v' as our parameters.
  - We can find k-dimensional 'v' by minimizing the squared error:

$$f(v) = \frac{1}{2}\|Zv - y\|^2$$

- Notice that in both cases the target is still 'y'.

Degree of Polynomial and Fundamental Trade-Off

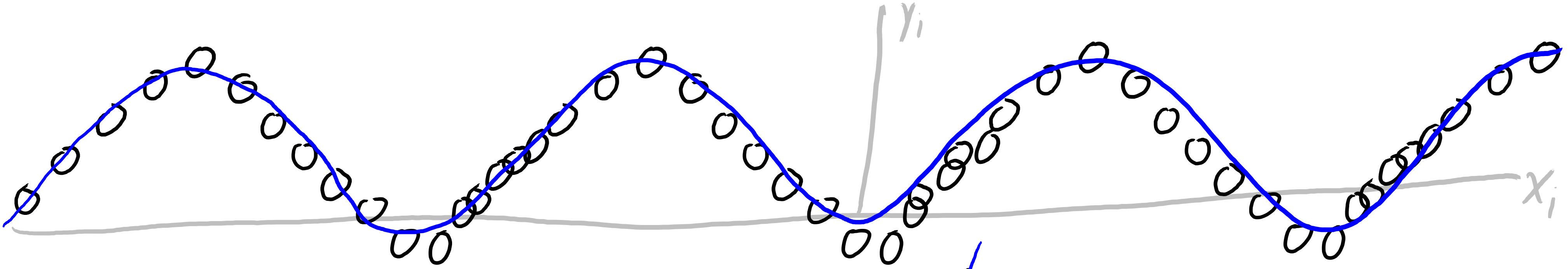
- As the polynomial degree increases, the training error goes down.



- But approximation error goes up: we start overfitting with large ‘p’.
- Usual approach to selecting degree: validation or cross-validation.

Beyond Polynomial Transformations

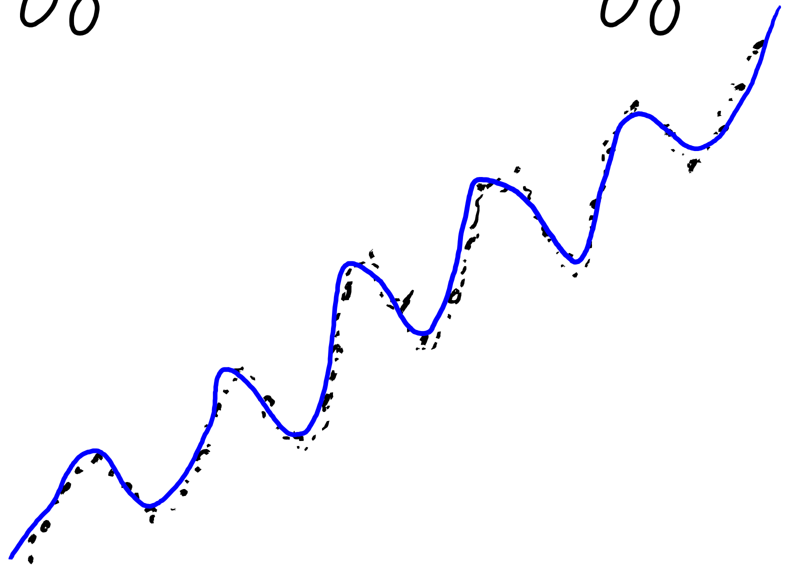
- Polynomials are not the only possible transformation:
  - Exponentials, logarithms, trigonometric functions, etc.
  - The right non-linear transform will vastly improve performance.



For periodic data we might use

$$Z = \begin{bmatrix} \sin(x_1) \\ \sin(x_2) \\ \vdots \\ \sin(x_n) \end{bmatrix}$$

$$\hat{y}_i = v^T z_i$$
$$= w_1 \sin(x_i)$$



You can have different types of bases

$$Z = \begin{bmatrix} x_1 & \sin(6x_1) \\ x_2 & \sin(6x_2) \\ \vdots & \vdots \\ \underbrace{x_n}_{\text{linear}} & \underbrace{\sin(6x_n)}_{\text{periodic}} \end{bmatrix}$$

xkcd

(pause)

Optimization Terminology

- When we minimize or maximize a function we call it "optimization".
  - In least squares, we want to solve the "optimization problem":

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

  - The function being optimized is called the "objective".
    - Also sometimes called "loss" or "cost", but these have different meanings in ML.
  - The set over which we search for an optimum is called the domain.
  - Often, instead of the minimum objective value, you want a minimizer.
    - A set of parameters 'w' that achieves the minimum value.

Discrete vs. Continuous Optimization

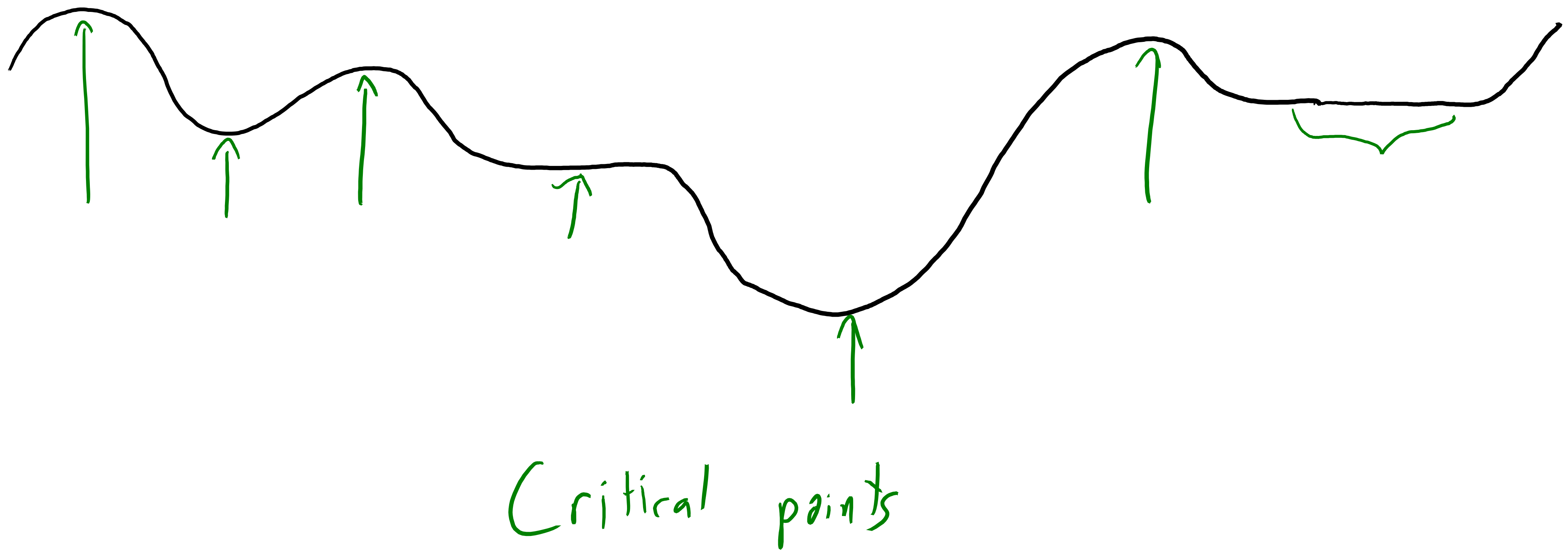
- We have seen examples of continuous optimization:
  - Least squares:
    - Domain is the real-valued set of parameters ‘w’.
    - Objective is the sum of the squared training errors.

- We have seen examples of discrete optimization:
  - Fitting decision stumps:
    - Domain is the finite set of unique rules.
    - Objective is the number of classification errors (or infogain).

- We have also seen a mixture of discrete and continuous:
  - K-means: clusters are discrete and means are continuous.

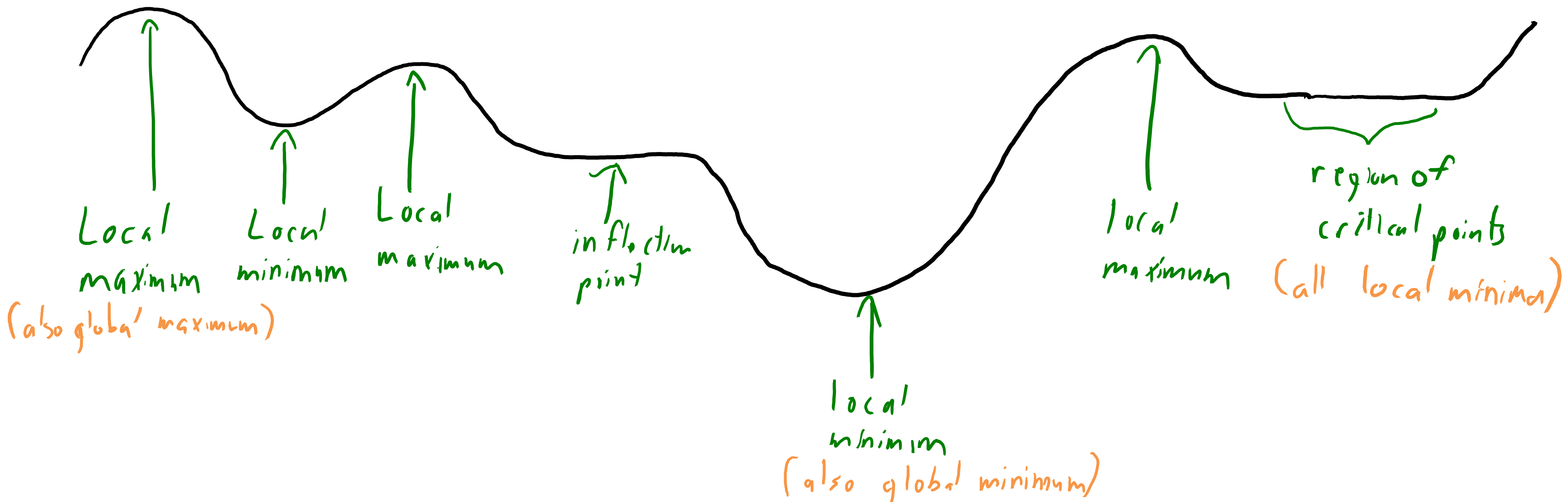
Stationary/Critical Points

- A 'w' with ∇ f(w) = 0 is called a stationary point or critical point.
  - The slope is zero so the tangent plane is "flat".



Stationary/Critical Points

- A ‘w’ with ∇ f(w) = 0 is called a stationary point or critical point.
  - The slope is zero so the tangent plane is “flat”.



  - If we’re minimizing, we would ideally like to find a global minimum.
    - But for some problems the best we can do is find a stationary point where ∇ f(w)=0.

Motivation: Large-Scale Least Squares

- Normal equations find 'w' with ∇ f(w) = 0 in $O(nd^2 + d^3)$ time.

$$\underbrace{\underbrace{(X^TX)}_{\substack{O(nd^2)\\ \text{(matrix multiply)}}} w = \underbrace{X^Ty}_{\substack{O(nd)\\ \text{(matrix-vector)}}}}_{} \rightarrow \text{Solving a } d \times d \text{ system is } O(d^3)$$

  - Very slow if 'd' is large.

- Alternative when 'd' is large is gradient descent methods.
  - Probably the most important class of algorithms in machine learning.

Gradient Descent for Finding a Local Minimum

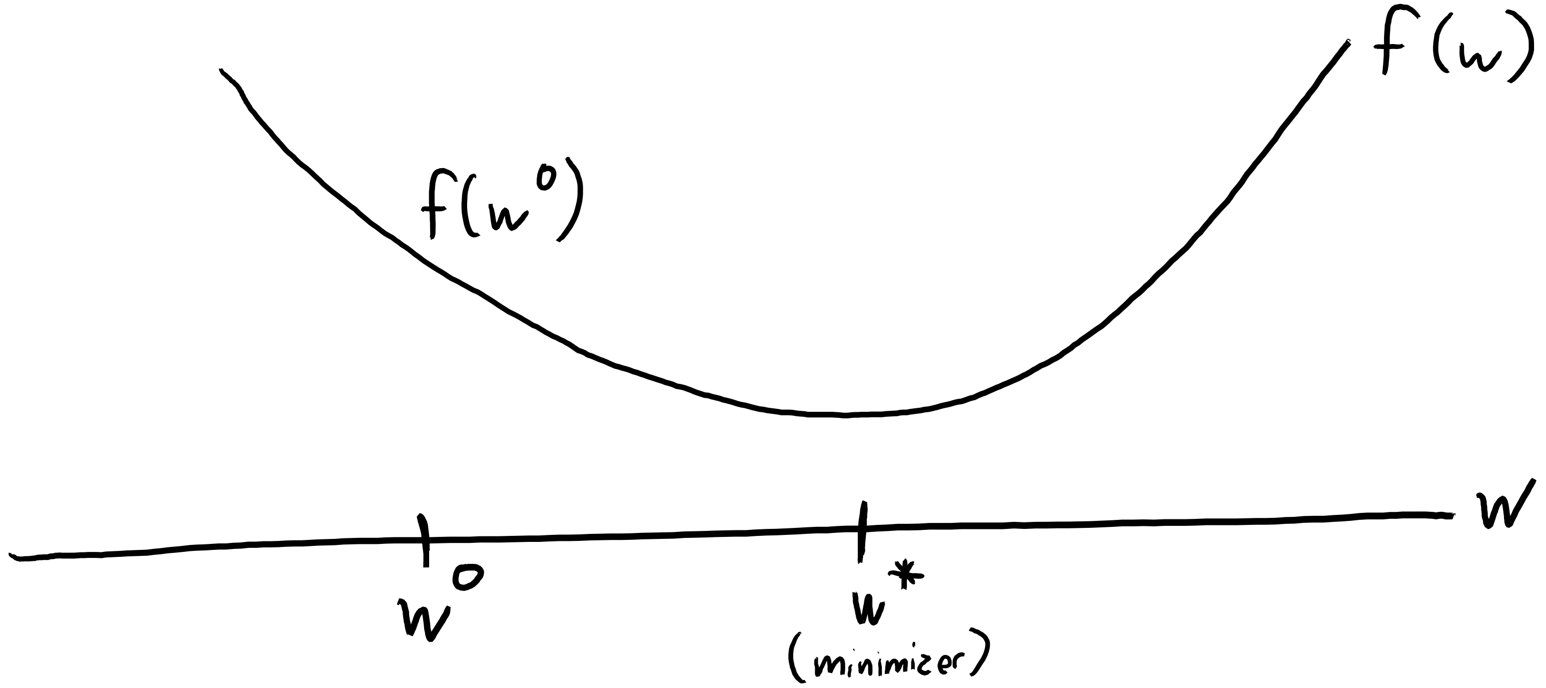
- Gradient descent is an iterative optimization algorithm:
  - It starts with a "guess" w^0.
  - It uses the gradient $\nabla f(w^0)$ to generate a better guess w^1.
  - It uses the gradient $\nabla f(w^1)$ to generate a better guess w^2.
  - It uses the gradient $\nabla f(w^2)$ to generate a better guess w^3.

    …
  - The limit of w^t as 't' goes to ∞ has $\nabla f(w^t) = 0$.

- It converges to a global optimum if 'f' is "convex".

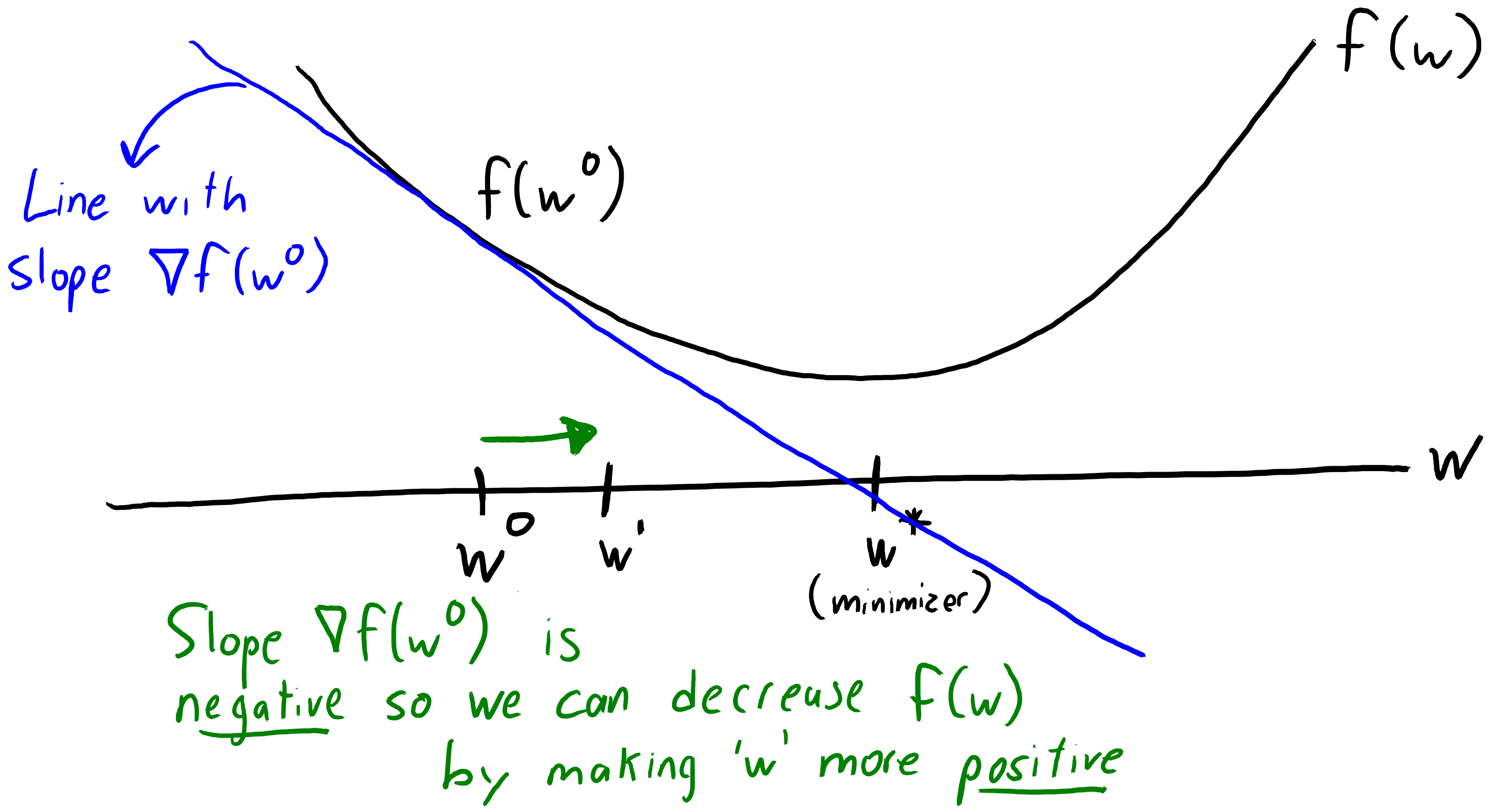
Gradient Descent for Finding a Local Minimum

- Gradient descent is based on a simple observation:
  - Give parameters 'w', the direction of largest decrease is $-\nabla$ f(w).



Gradient Descent for Finding a Local Minimum

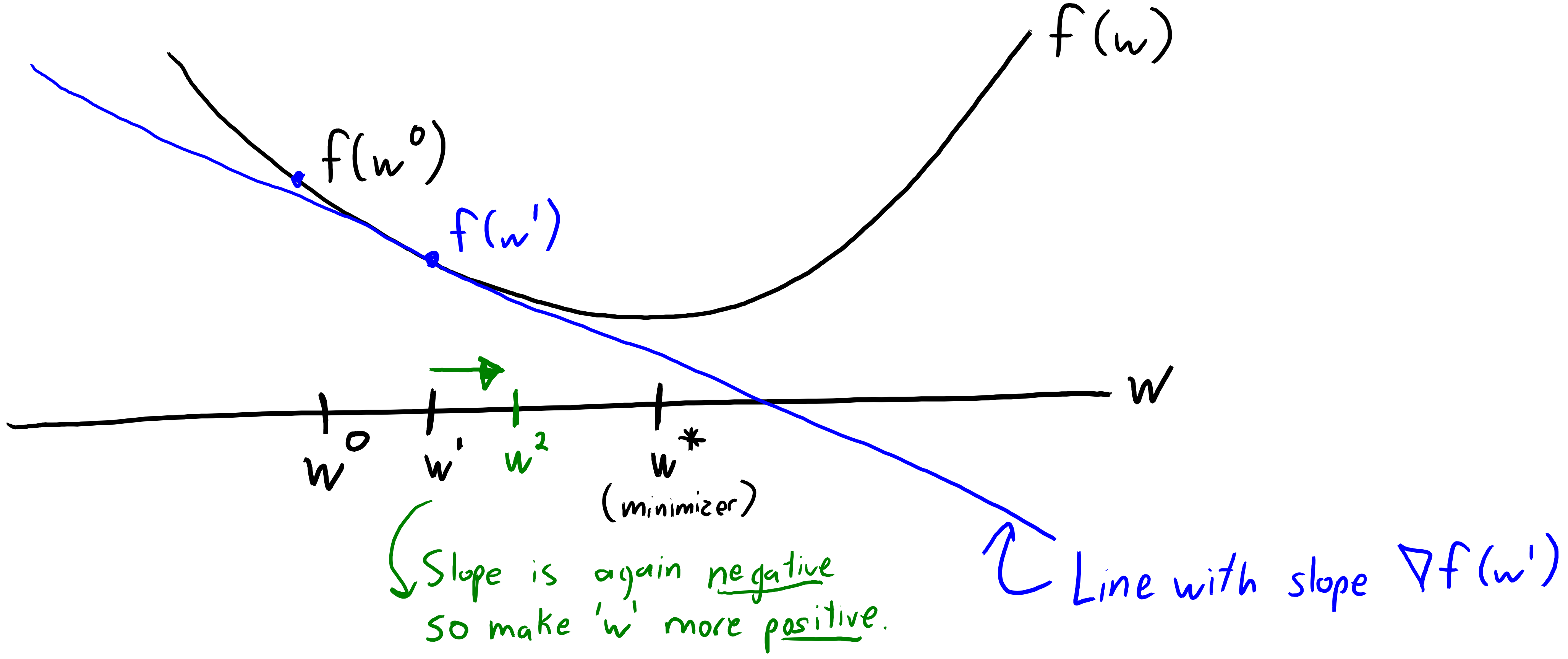
- Gradient descent is based on a simple observation:
  - Give parameters ‘w’, the direction of largest decrease is $-\nabla f(w)$.



$f(w)$
Line with slope $\nabla f(w^0)$
$f(w^0)$
w
w^0
w^1
w^*
(minimizer)
Slope $\nabla f(w^0)$ is negative so we can decrease $f(w)$ by making ‘w’ more positive

Gradient Descent for Finding a Local Minimum

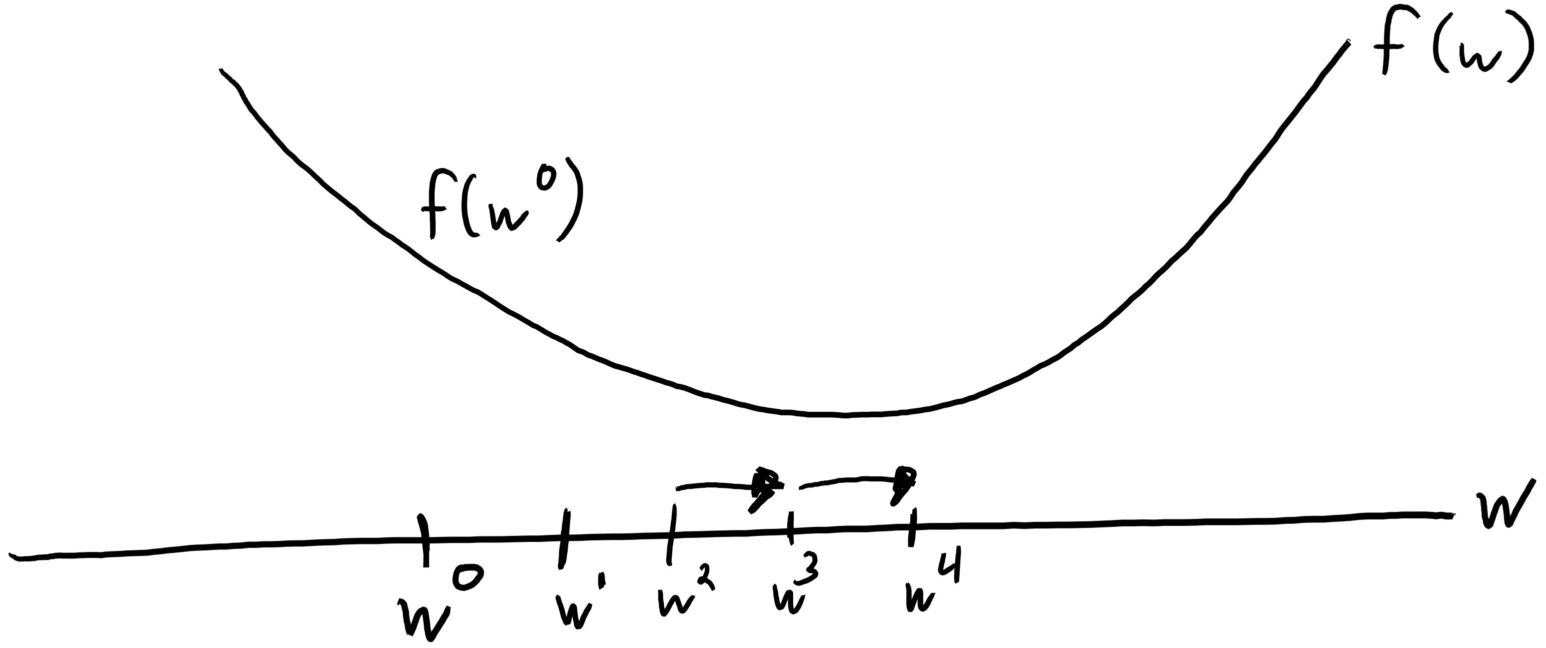
- Gradient descent is based on a simple observation:
  - Give parameters ‘w’, the direction of largest decrease is $-\nabla f(w)$.



f(w)
$f(w^0)$
$f(w^1)$
w
w^0
w^1
w^2
w^*
(minimizer)
Slope is again negative so make 'w' more positive.
Line with slope $\nabla f(w^1)$

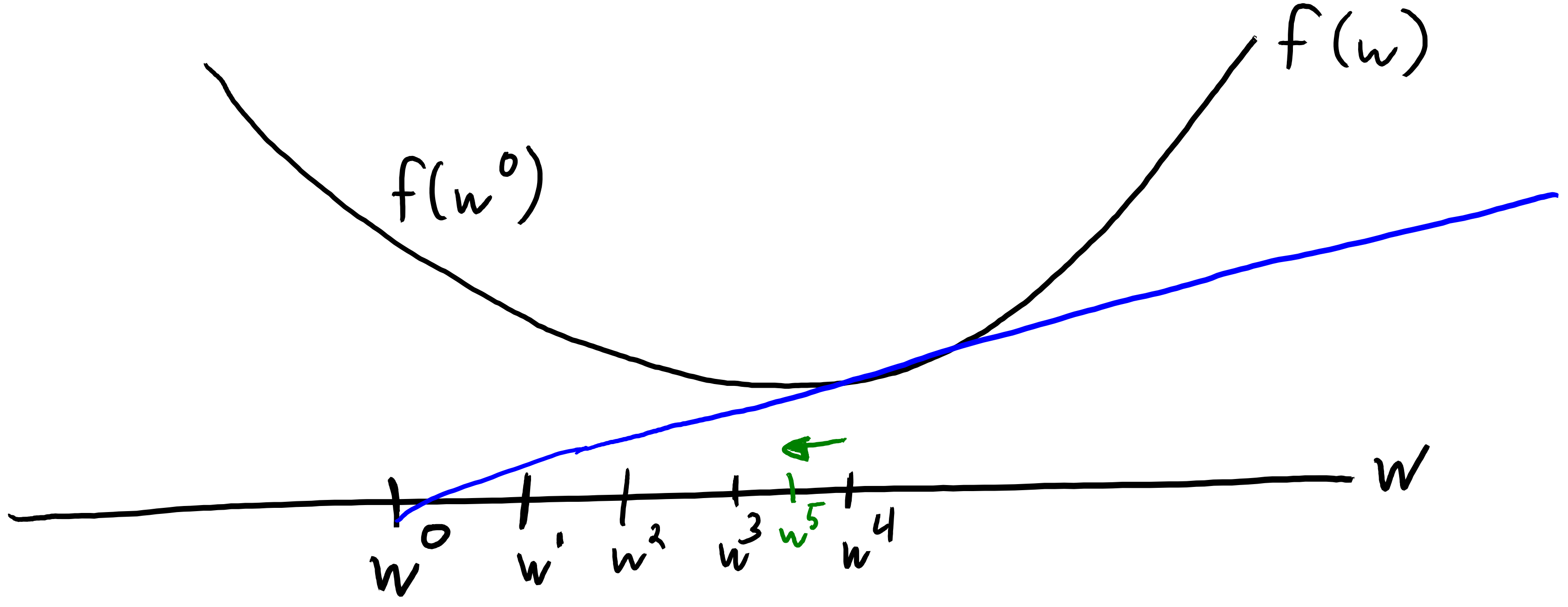
Gradient Descent for Finding a Local Minimum

- Gradient descent is based on a simple observation:
  - Give parameters ‘w’, the direction of largest decrease is $-\nabla f(w)$.



Gradient Descent for Finding a Local Minimum

- Gradient descent is based on a simple observation:
  - Give parameters ‘w’, the direction of largest decrease is $-\nabla f(w)$.



Now the slope $\nabla f(w^4)$ is positive so we move in the negative direction.

Gradient Descent for Finding a Local Minimum

- We start with some initial guess, w^0.
- Generate new guess by moving in the negative gradient direction:

$$w^1 = w^0 - \alpha^0 \nabla f(w^0)$$

  - This decreases 'f' if the "step size" α^0 is small enough.
  - Usually, we decrease α^0 if it increases 'f' (see "findMin").
- Repeat to successively refine the guess:

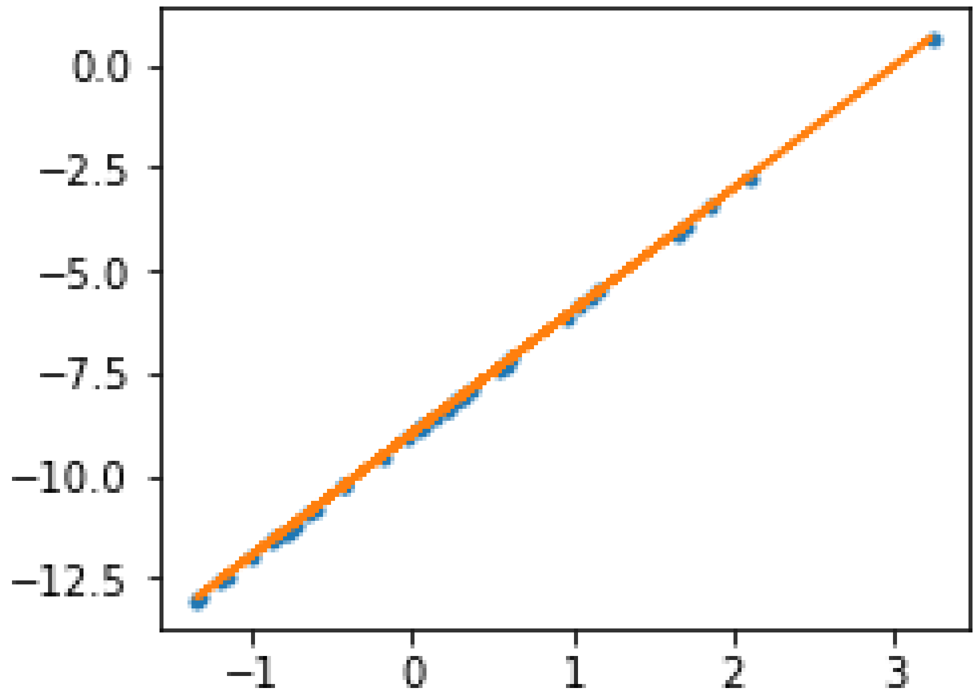
$$w^{t+1} = w^t - \alpha^t \nabla f(w^t) \quad \text{for } t = 1, 2, 3, \ldots$$

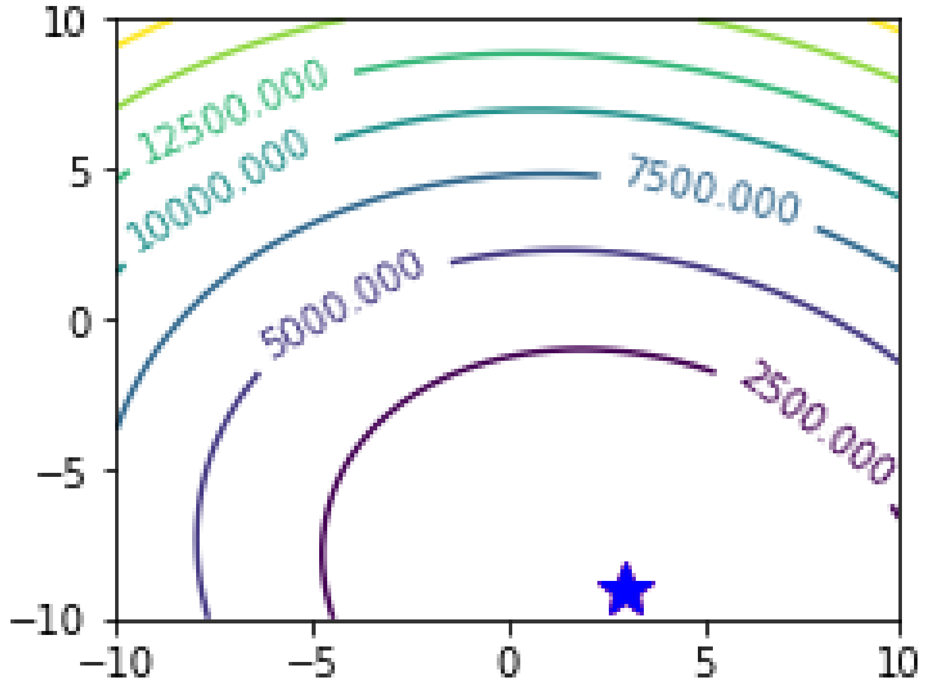
- Stop if not making progress or $\underbrace{\|\nabla f(w^t)\| \leq \varepsilon}_{\text{Approximate local minimum}}$

$\varepsilon \rightarrow$ Some small scalar.

Data Space vs. Parameter Space

- Usual regression plot is in the "x vs. y" data space (left):

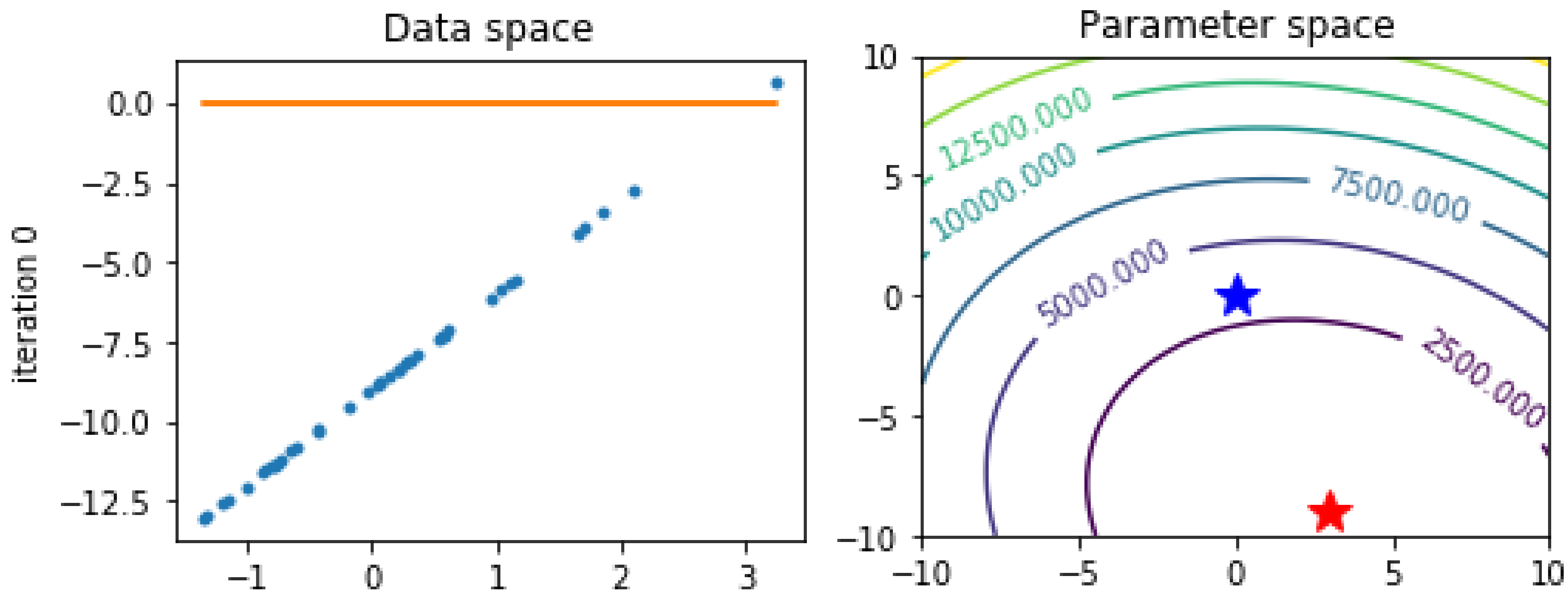




- On the right is plot of the "intecerpt vs. slope" parameter space.
  - Points in parameter space correspond to models (* is least squares parameters).

Gradient Descent in Data Space vs. Parameter Space

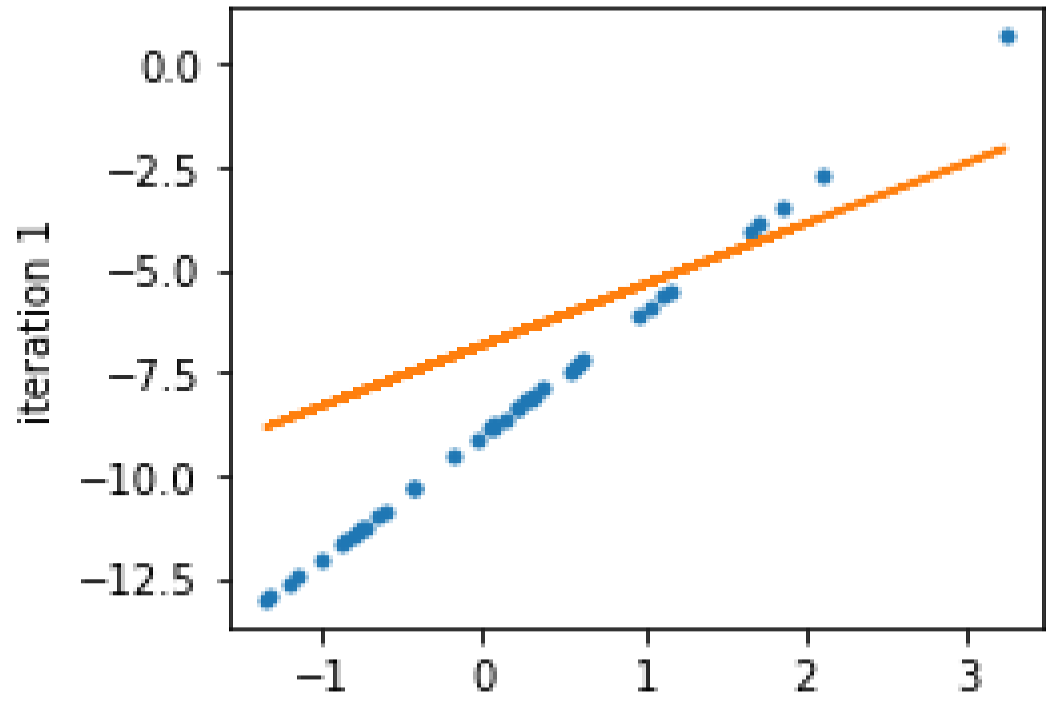
- Gradient descent starts with an initial guess in parameter space:

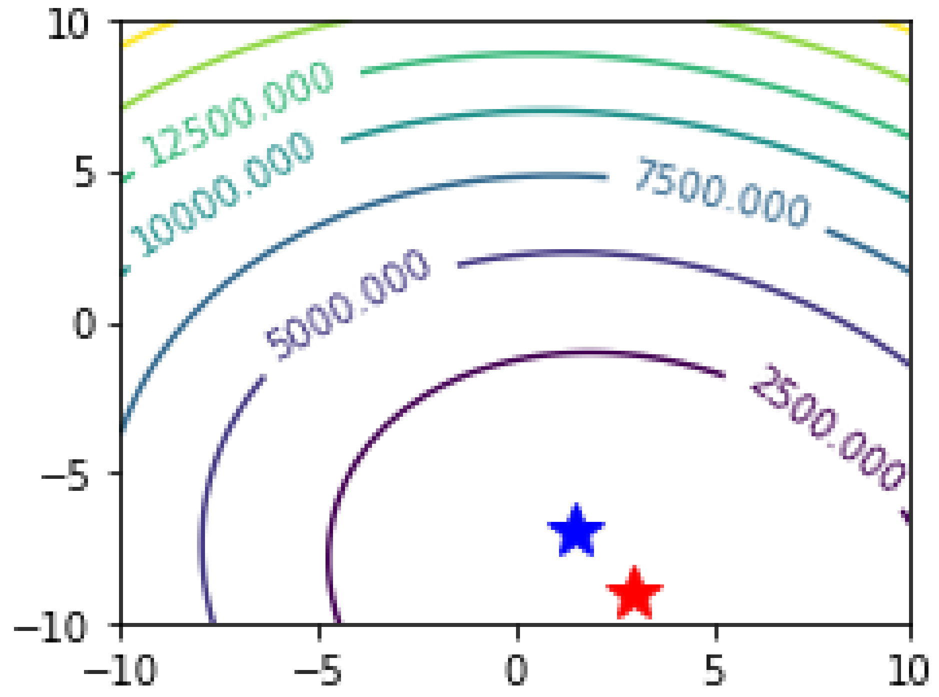


  – And each iteration tries to move guess closer to solution.

Gradient Descent in Data Space vs. Parameter Space

- Gradient descent starts with an initial guess in parameter space:

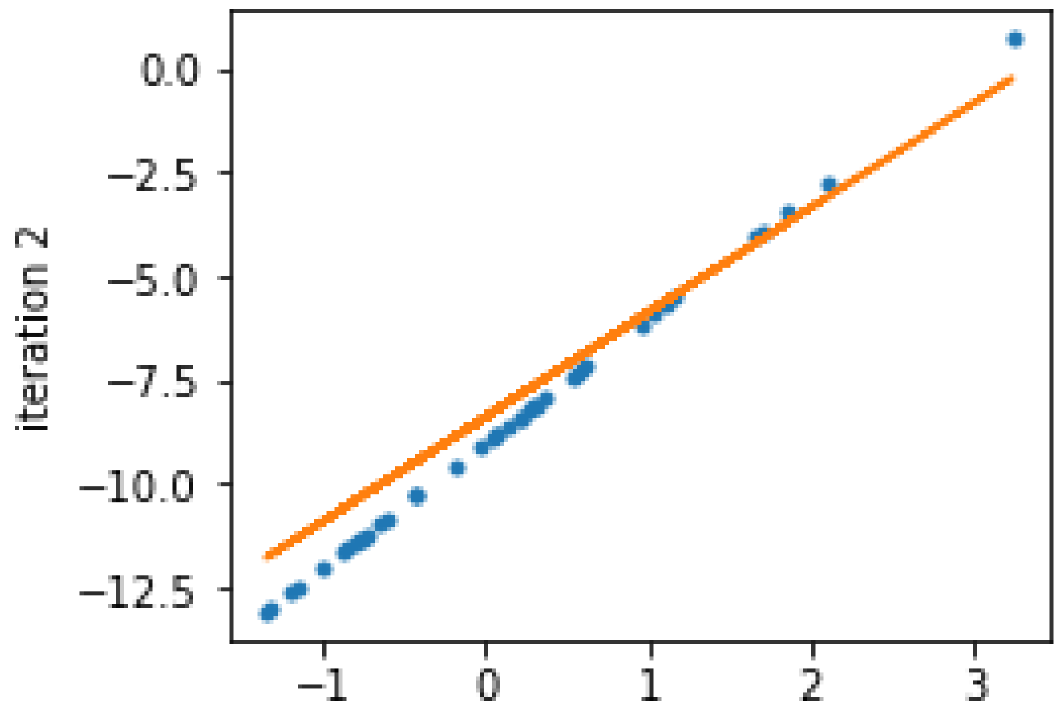


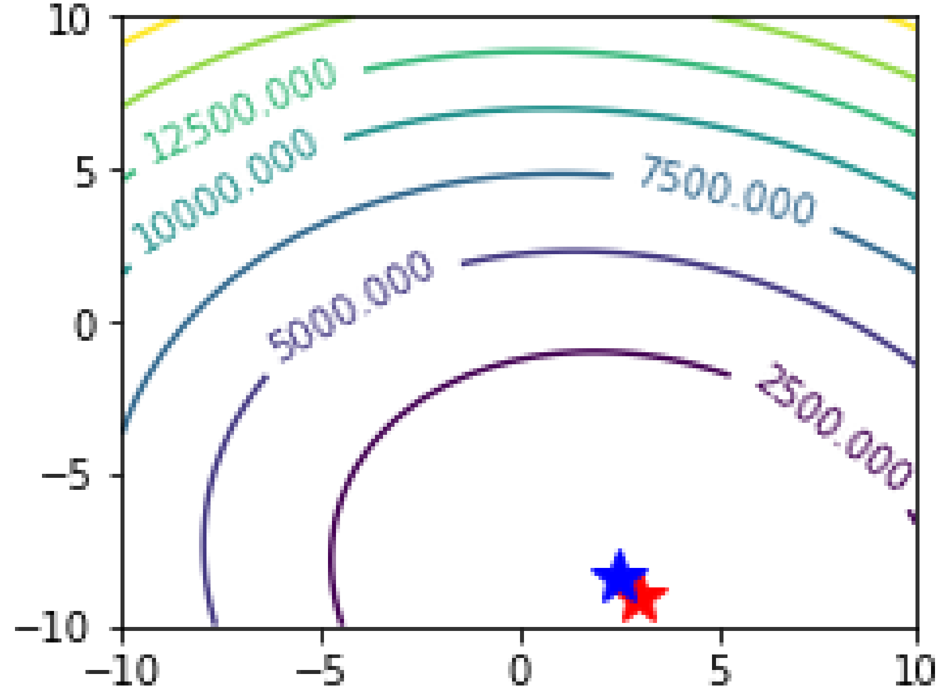


  – And each iteration tries to move guess closer to solution.

Gradient Descent in Data Space vs. Parameter Space

- Gradient descent starts with an initial guess in parameter space:

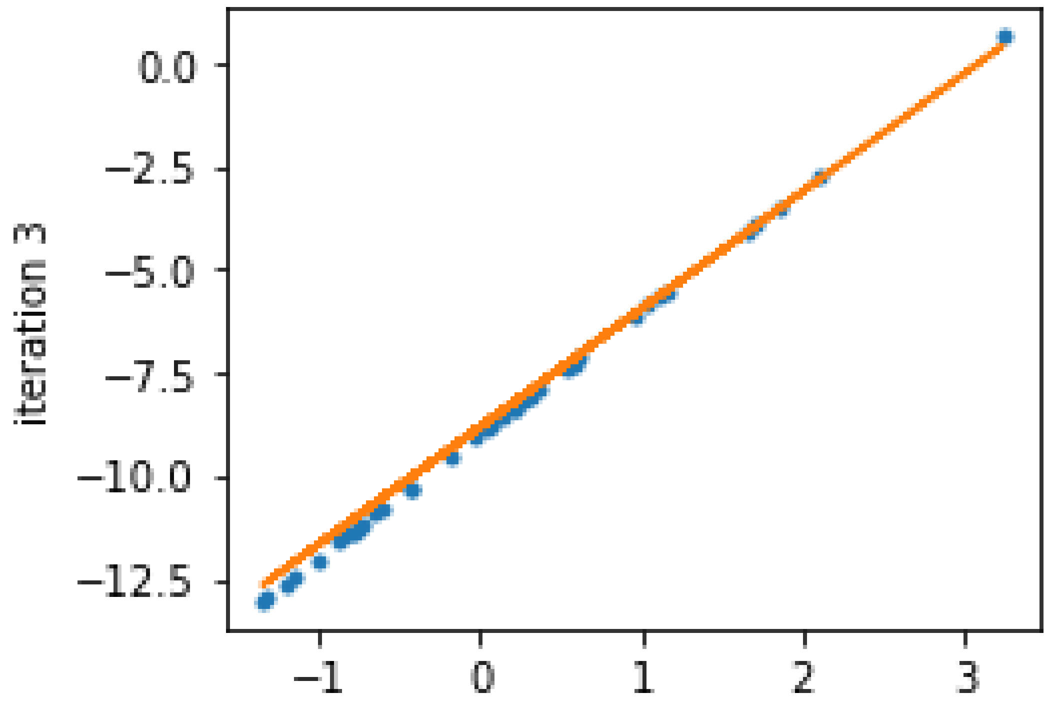


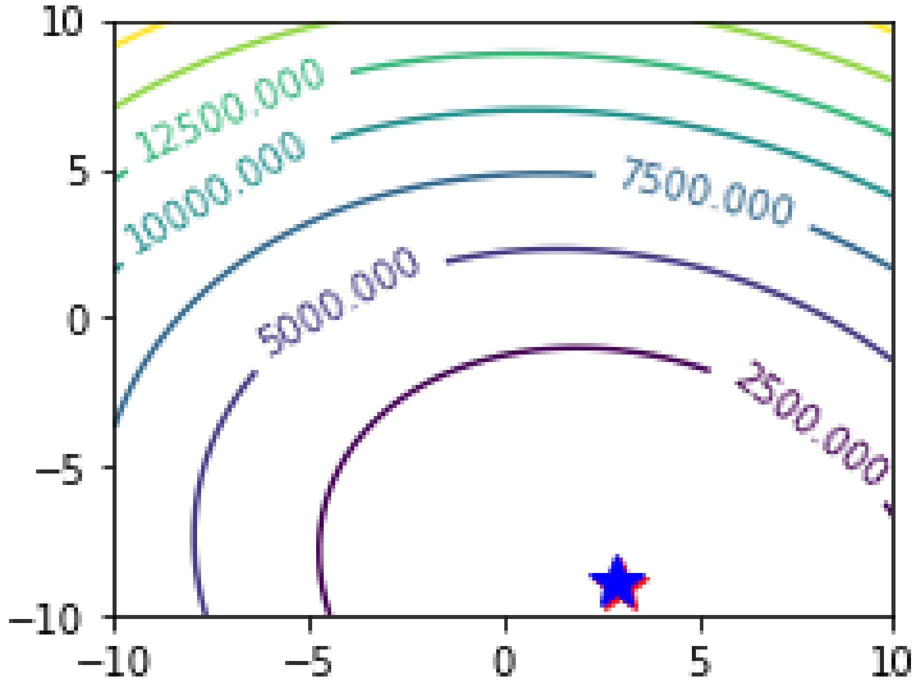


  – And each iteration tries to move guess closer to solution.

Gradient Descent in Data Space vs. Parameter Space

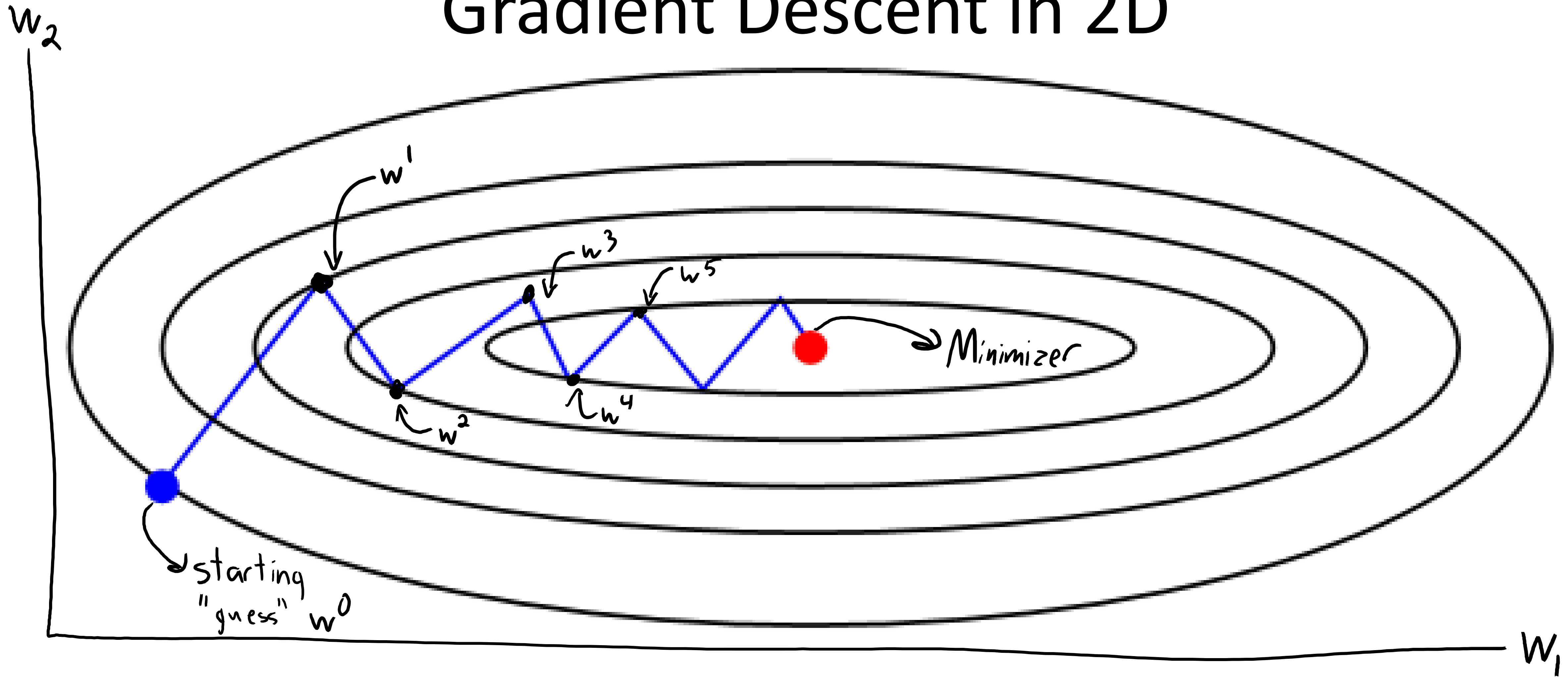
- Gradient descent starts with an initial guess in parameter space:





  - And each iteration tries to move guess closer to solution.

Gradient Descent in 2D



- Under weak conditions, algorithm converges to a ‘w’ with $\nabla f(w) = 0$.
  - ‘f’ is bounded below, ∇f can’t change arbitrarily fast, small-enough constant α^t.

Gradient Descent for Least Squares

- The least squares objective and gradient:

$$f(w) = \frac{1}{2}\|Xw - y\|^2 \qquad \nabla f(w) = X^T(Xw - y)$$

- Gradient descent iterations for least squares:

$$w^{t+1} = w^t - \alpha^t \underbrace{X^T(Xw^t - y)}_{\nabla f(w^t)}$$

- Cost of gradient descent iteration is O(nd) (no need to form X^{T}X).

Bottleneck is computing $\nabla f(w^t) = \underbrace{X^T\underbrace{(\underbrace{Xw^t}_{O(nd)} - y)}_{O(n)}}_{O(nd)}$

Normal Equations vs. Gradient Descent

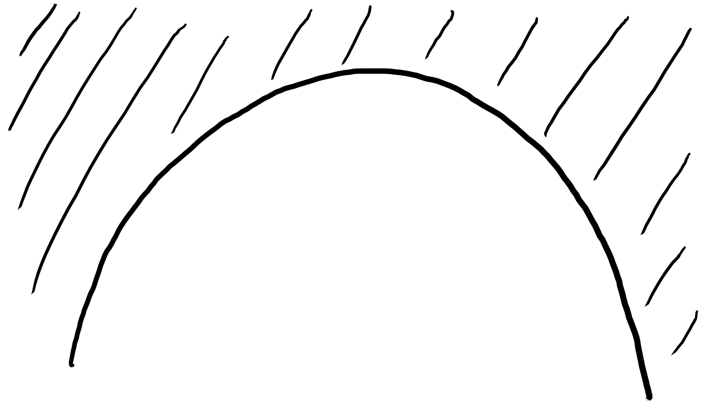
- Least squares via normal equations vs. gradient descent:
  - Normal equations cost $O(nd^2 + d^3)$.
  - Gradient descent costs $O(ndt)$ to run for ‘t’ iterations.
    - Each of the ‘t’ iterations costs $O(nd)$.
  - Gradient descent can be faster when ‘d’ is very large:
    - If solution is “good enough” for a ‘t’ less than minimum($d,d^2/n$).
    - CPSC 540: ‘t’ proportional to “condition number” of X^TX (no direct ‘d’ dependence).
  - Normal equations only solve linear least squares problems.
    - Gradient descent solves many other problems.

Beyond Gradient Descent

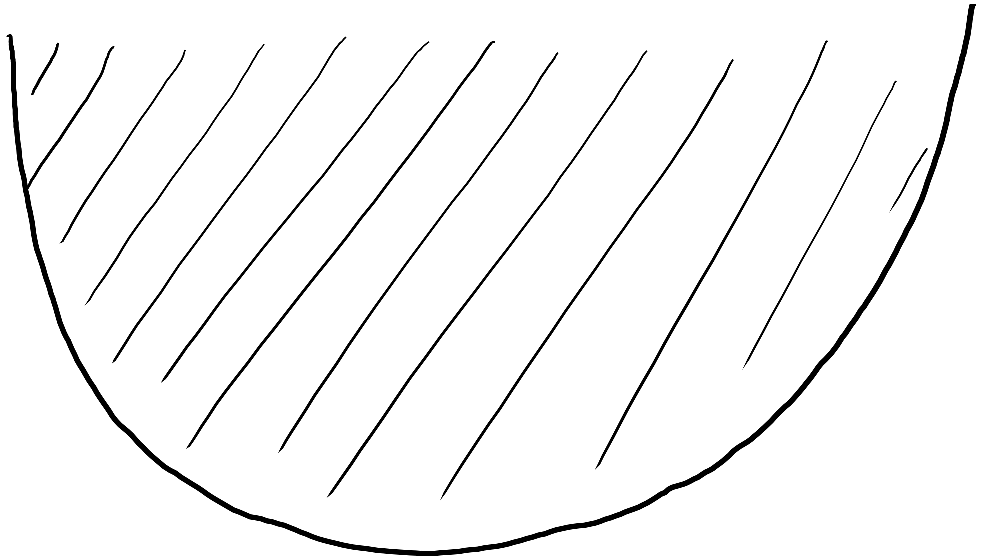
- There are many variations on gradient descent.
  - Methods employing a “line search” to choose the step-size.
  - “Conjugate” gradient and “accelerated” gradient methods.
  - Newton’s method (which uses second derivatives).
  - Quasi-Newton and Hessian-free Newton methods.
  - Stochastic gradient (later in course).
- This course focuses on gradient descent and stochastic gradient:
  - They’re simple and give reasonable solutions to most ML problems.
  - But the above can be faster for some applications.

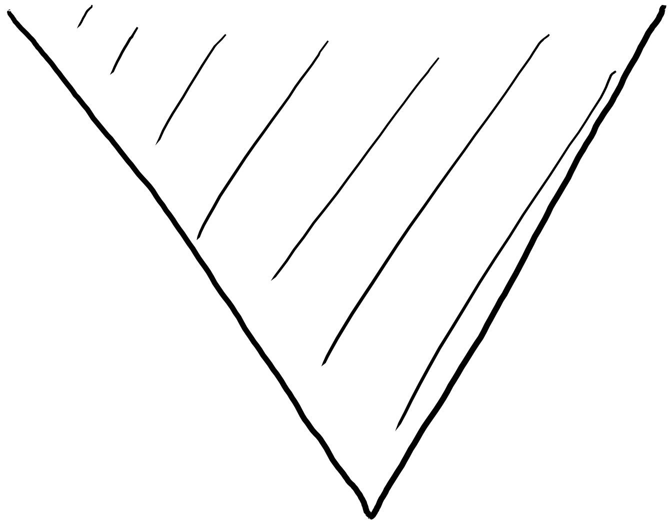
(pause)

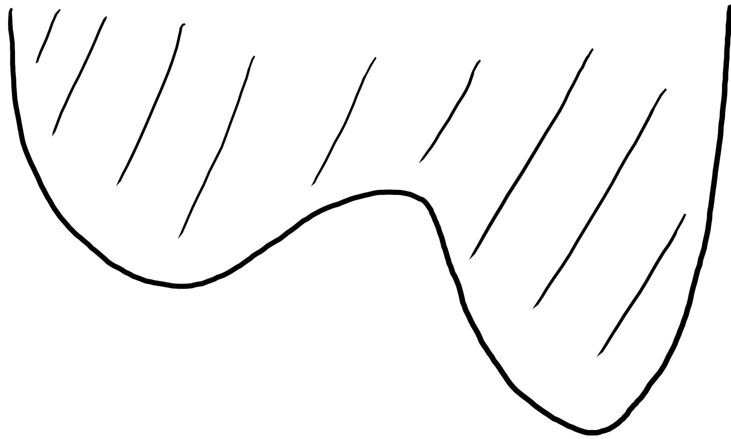
Convex Functions



- Is finding a 'w' with $\nabla f(w) = 0$ good enough?
  - Yes, for convex functions.







- A function is convex if the area above the function is a convex set.
  - All values between any two points above function stay above function.

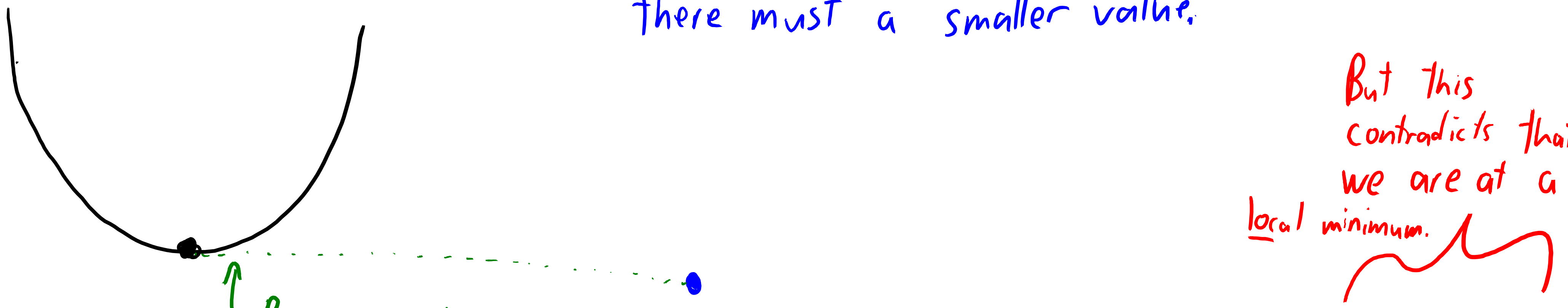
Convex Functions

- All ‘w’ with ∇ f(w) = 0 for convex functions are global minima.

Proof by contradiction:

Consider a local minimum

If this is not global minimum, there must a smaller value.



By convexity we can move along line to global minimum and decrease objective.

But this contradicts that we are at a local minimum.

  – Normal equations find a global minimum because least squares is convex.

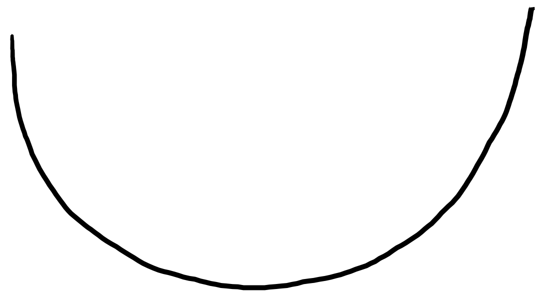
How do we know if a function is convex?

- Some useful tricks for showing a function is convex:
  - 1-variable, twice-differentiable function is convex iff f''(w) ≥ 0 for all 'w'.

Consider $f(w) = \frac{1}{2}aw^2$ for $\underbrace{a > 0}$

We have $f'(w) = aw$
and $f''(w) = a > 0$

By assumption



Consider $f(w) = e^w$

We have $f'(w) = e^w$
and $f''(w) = e^w > 0$

By definition of exponential function.

How do we know if a function is convex?

- Some useful tricks for showing a function is convex:
  - 1-variable, twice-differentiable function is convex iff $f''(w) \geq 0$ for all 'w'.
  - A convex function multiplied by non-negative constant is convex.

We showed that $f(w) = e^w$ is convex, so $f(w) = 10e^w$ is convex.

How do we know if a function is convex?

- Some useful tricks for showing a function is convex:
  - 1-variable, twice-differentiable function is convex iff f''(w) ≥ 0 for all 'w'.
  - A convex function multiplied by non-negative constant is convex.
  - Norms and squared norms are convex.

$\|w\|$, $\|w\|^2$, $\|w\|_1$, $\|w\|_\infty$, $\|w\|_1^2$, and so on are all convex.

How do we know if a function is convex?

- Some useful tricks for showing a function is convex:
  - 1-variable, twice-differentiable function is convex iff $f''(w) \geq 0$ for all ‘w’.
  - A convex function multiplied by non-negative constant is convex.
  - Norms and squared norms are convex.
  - The sum of convex functions is a convex function.

$$f(w) = \underbrace{10e^{w}}_{\text{From earlier}} + \underbrace{\frac{1}{2}}_{\text{constant}}\underbrace{\|w\|^2}_{\text{norm squared}} \quad \text{is convex}$$

How do we know if a function is convex?

- Some useful tricks for showing a function is convex:
  - 1-variable, twice-differentiable function is convex iff $f''(w) \geq 0$ for all 'w'.
  - A convex function multiplied by non-negative constant is convex.
  - Norms and squared norms are convex.
  - The sum of convex functions is a convex function.

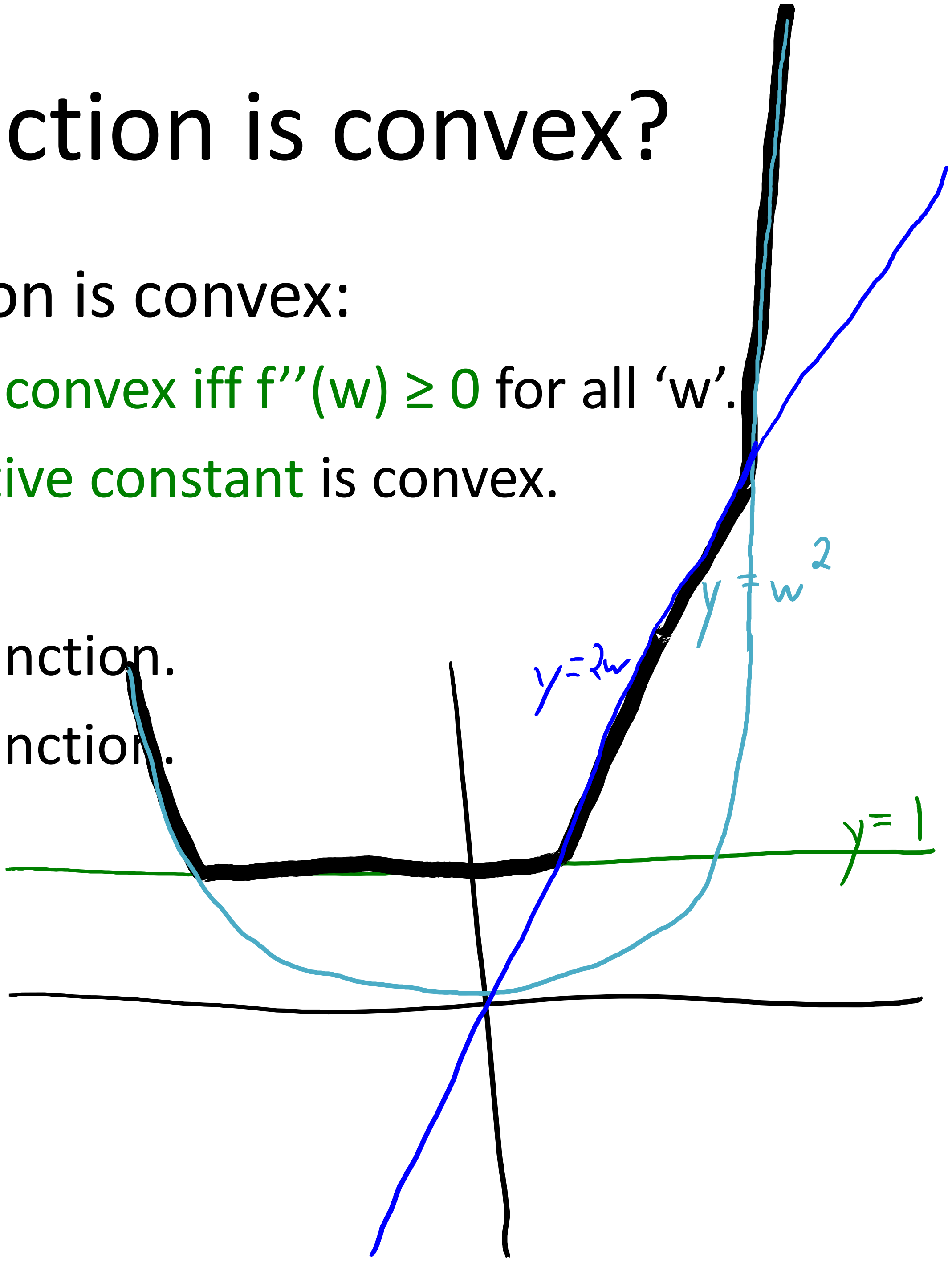
$$f(w) = w^T x_i = \underbrace{w_1 x_{i1}}_{\text{convex}} + \underbrace{w_2 x_{i2}}_{\text{convex}} + \cdots + \underbrace{w_d x_{id}}_{\text{convex}}$$

Second derivative of each term is 0.

How do we know if a function is convex?

- Some useful tricks for showing a function is convex:
  - 1-variable, twice-differentiable function is convex iff f''(w) ≥ 0 for all 'w'.
  - A convex function multiplied by non-negative constant is convex.
  - Norms and squared norms are convex.
  - The sum of convex functions is a convex function.
  - The max of convex functions is a convex function.

$$f(w) = \max\{\underbrace{1, 2w, w^2}_{\text{convex}}\} \text{ is convex.}$$



How do we know if a function is convex?

- Some useful tricks for showing a function is convex:
  - 1-variable, twice-differentiable function is convex iff f''(w) ≥ 0 for all 'w'.
  - A convex function multiplied by non-negative constant is convex.
  - Norms and squared norms are convex.
  - The sum of convex functions is a convex function.
  - The max of convex functions is a convex function.
  - Composition of a convex function and a linear function is convex.

If $f(w) = g(\underbrace{Xw - y}_{\text{linear function}})$ then 'f' is convex if 'g' is convex

How do we know if a function is convex?

- Some useful tricks for showing a function is convex:
  - 1-variable, twice-differentiable function is convex iff $f''(w) \geq 0$ for all 'w'.
  - A convex function multiplied by non-negative constant is convex.
  - Norms and squared norms are convex.
  - The sum of convex functions is a convex function.
  - The max of convex functions is a convex function.
  - Composition of a convex function and a linear function is convex.
- But: not true that composition of convex with convex is convex:

Even if 'f' is convex and 'g' is convex, $f(g(w))$ might <u>not</u> be convex.

E.g. x^2 is convex and $-\log(x)$ is convex but $-\log(x^2)$ is <u>not</u> convex.

Example: Convexity of Linear Regression

- Consider linear regression objective with squared error:

$$f(w) = \|Xw - y\|^2$$

- We can use that this is a convex function composed with linear:

Let $g(r) = \|r\|^2$, which is convex because it's a squared norm.

Then $f(w) = g(Xw - y)$, which is convex because it's a convex function composed with a linear function

Convexity in Higher Dimensions

- Twice-differentiable ‘d’-variable function is convex iff:
  - Eigenvalues of Hessian $\nabla^2 f(w)$ are non-negative for all ‘w’.

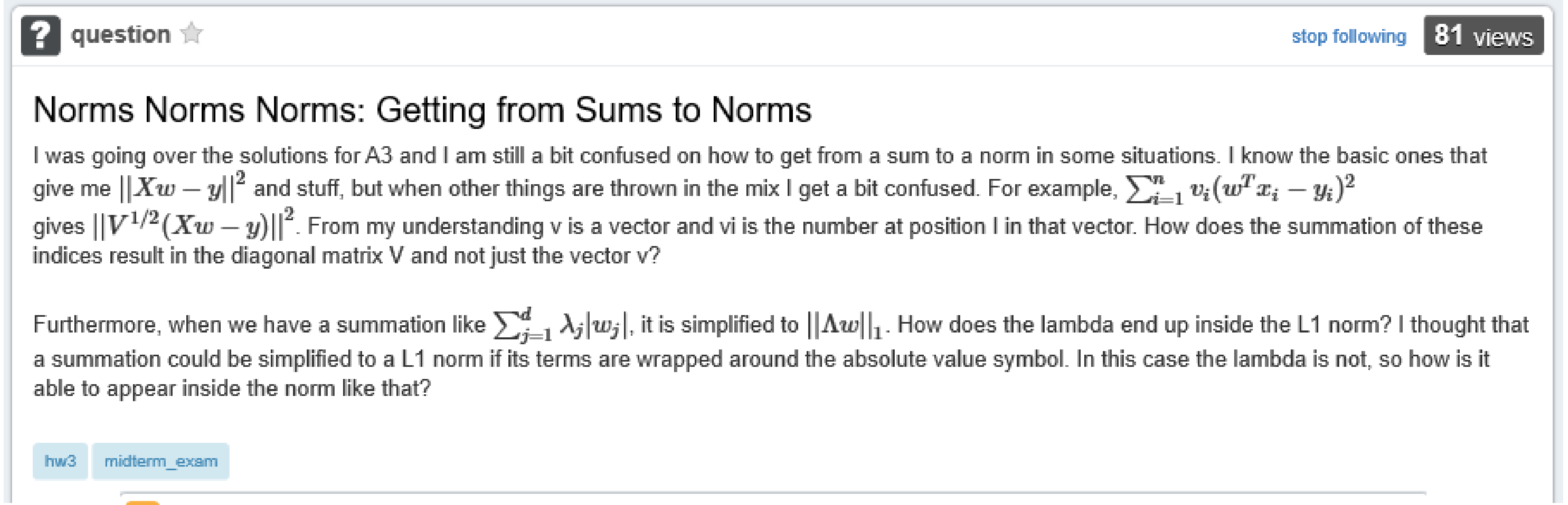
- True for least squares where $\nabla^2 f(w)$ = X^TX for all ‘w’.
  - It may not be obvious that this matrix has non-negative eigenvalues.

- Unfortunately, sometimes it is hard to show convexity this way.
  - Usually easier to just use some of the rules as we did on the last slide.

Summary

- Gradient descent finds critical point of differentiable function.
  - Finds global optimum if function is convex.
- Convex functions:
  - Set of functions with property that $\nabla f(w) = 0$ implies ‘w’ is a global min.
  - Can (usually) be identified using a few simple rules.

- Next time:
  - Linear regression without the outlier sensitivity…



question

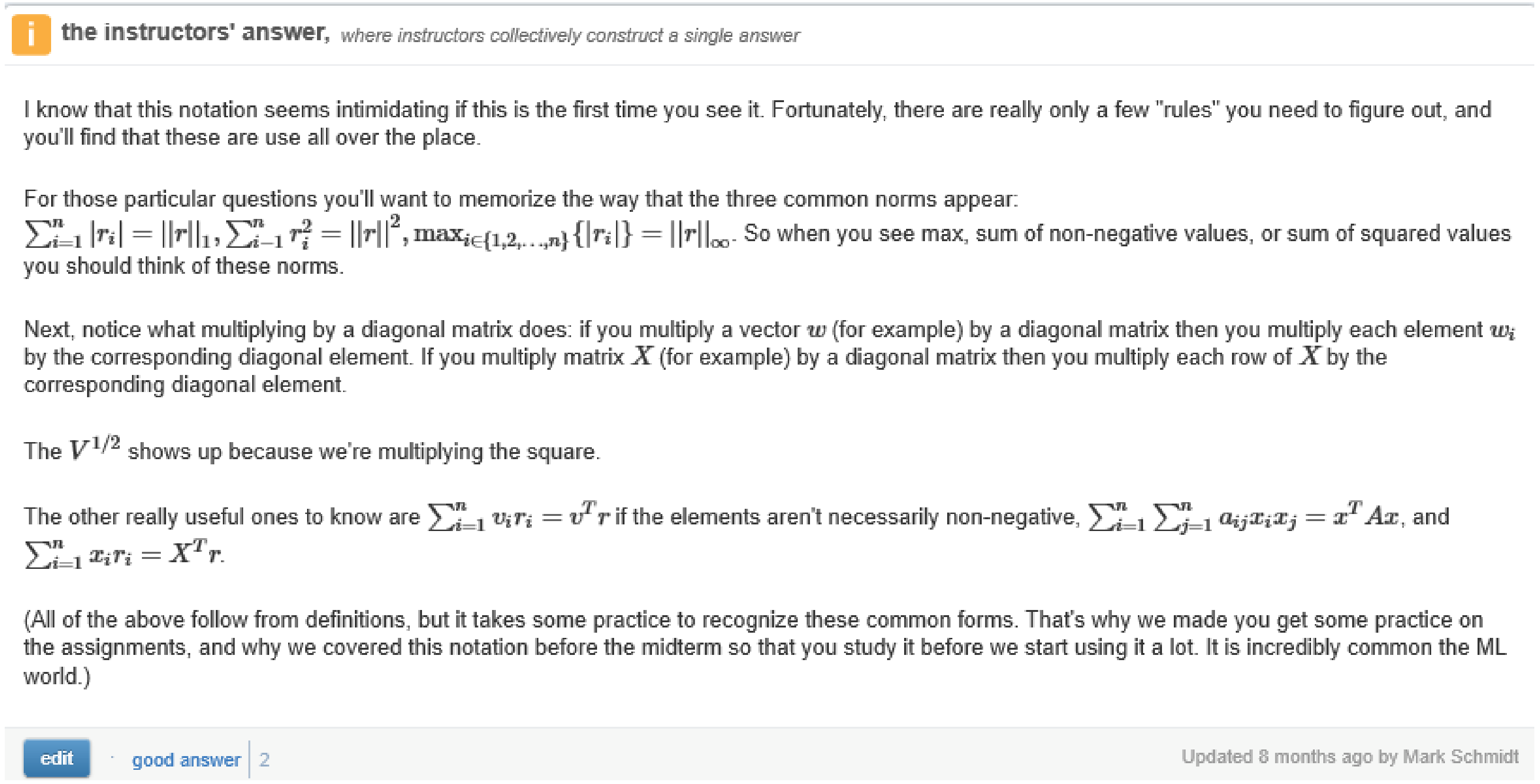
stop following 81 views

Norms Norms Norms: Getting from Sums to Norms

I was going over the solutions for A3 and I am still a bit confused on how to get from a sum to a norm in some situations. I know the basic ones that give me $||Xw-y||^2$ and stuff, but when other things are thrown in the mix I get a bit confused. For example, $\sum_{i=1}^{n} v_i(w^Tx_i - y_i)^2$ gives $||V^{1/2}(Xw-y)||^2$. From my understanding v is a vector and vi is the number at position I in that vector. How does the summation of these indices result in the diagonal matrix V and not just the vector v?

Furthermore, when we have a summation like $\sum_{j=1}^{d} \lambda_j|w_j|$, it is simplified to $||\Lambda w||_1$. How does the lambda end up inside the L1 norm? I thought that a summation could be simplified to a L1 norm if its terms are wrapped around the absolute value symbol. In this case the lambda is not, so how is it able to appear inside the norm like that?

hw3 midterm_exam



the instructors' answer, *where instructors collectively construct a single answer*

I know that this notation seems intimidating if this is the first time you see it. Fortunately, there are really only a few "rules" you need to figure out, and you'll find that these are use all over the place.

For those particular questions you'll want to memorize the way that the three common norms appear: $\sum_{i=1}^{n}|r_i| = ||r||_1, \sum_{i=1}^{n} r_i^2 = ||r||^2, \max_{i\in\{1,2,...,n\}}\{|r_i|\} = ||r||_\infty$. So when you see max, sum of non-negative values, or sum of squared values you should think of these norms.

Next, notice what multiplying by a diagonal matrix does: if you multiply a vector w (for example) by a diagonal matrix then you multiply each element w_i by the corresponding diagonal element. If you multiply matrix X (for example) by a diagonal matrix then you multiply each row of X by the corresponding diagonal element.

The $V^{1/2}$ shows up because we're multiplying the square.

The other really useful ones to know are $\sum_{i=1}^{n} v_i r_i = v^T r$ if the elements aren't necessarily non-negative, $\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_ix_j = x^TAx$, and $\sum_{i=1}^{n} x_i r_i = X^T r$.

(All of the above follow from definitions, but it takes some practice to recognize these common forms. That's why we made you get some practice on the assignments, and why we covered this notation before the midterm so that you study it before we start using it a lot. It is incredibly common the ML world.)

edit · good answer | 2 Updated 8 months ago by Mark Schmidt

Constraints, Continuity, Smoothness

- Sometimes we need to optimize with constraints:
  - Later we'll see "non-negative least squares".

$$\min_{w \geq 0} \frac{1}{2} \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

  - A vector 'w' satisfying w ≥ 0 (element-wise) is said to be "feasible".
- Two factors affecting difficulty are continuity and smoothness.
  - Continuous functions tend to be easier than discontinuous functions.
  - Smooth/differentiable functions tend to be easier than non-smooth.
  - See the calculus review here if you haven't heard these words in a while.

Convexity, min, and argmin

- If a function is convex, then all critical points are global optima.

- However, convex functions don’t necessarily have critical points:
  - For example, f(x) = a*x, f(x) = exp(x), etc.

- Also, more than one ‘x’ can achieve the global optimum:
  - For example, f(x) = c is minimized by any ‘x’.

Why use the negative gradient direction?

- For a twice-differentiable 'f', multivariable Taylor expansion gives:

$$f(w^{t+1}) = f(w^t) + \nabla f(w^t)^T(w^{t+1}-w^t) + \frac{1}{2}(w^{t+1}-w^t)\underbrace{\nabla^2 f(v)}_{\text{for some 'v' between } w^{t+1} \text{ and } w^t.}(w^{t+1}-w^t)$$

- If gradient can't change arbitrarily quickly, Hessian is bounded and:

$$f(w^{t+1}) = f(w^t) + \nabla f(w^t)^T(w^{t+1}-w^t) + \underbrace{O(\|w^{t+1}-w^t\|^2)}_{\text{becomes negigible as } w^{t+1} \text{ gets close to } w^t}$$

  - But which choice of w^{t+1} decreases 'f' the most?
    - As $||w^{t+1}-w^t||$ gets close to zero, the value of w^{t+1} minimizing $f(w^{t+1})$ in this formula converges to $(w^{t+1} - w^t) = -\alpha^t \nabla f(w^t)$ for some scalar α^t.
    - So if we're moving a small amount, the optimal w^{t+1} is: $w^{t+1} = w^t - \alpha_t \nabla f(w^t)$ for some scalar α_t.

Normalized Steps

Question from class: "can we use $w^{t+1} = w^t - \frac{1}{\|\nabla f(w^t)\|} \nabla f(w^t)$"

This will work for a while, but notice that

$$\|w^{t+1} - w^t\| = \left\| \frac{1}{\|\nabla f(w^t)\|} \nabla f(w^t) \right\|$$

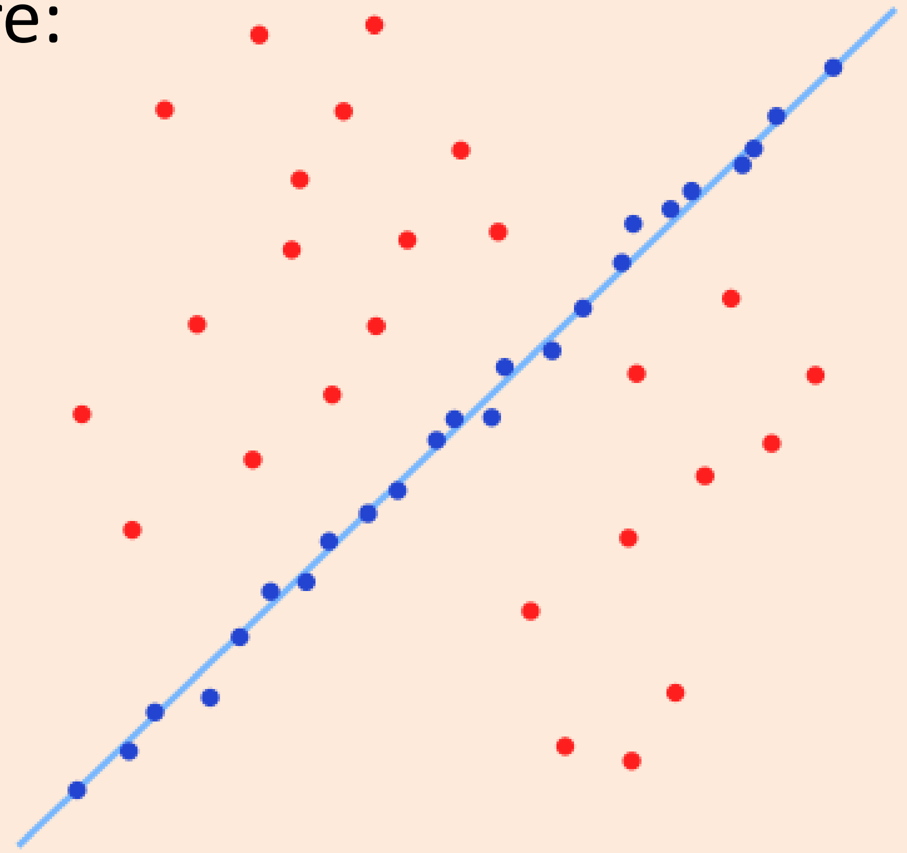
$$= \frac{1}{\|\nabla f(w^t)\|} \|\nabla f(w^t)\|$$

$$= 1$$

So the algorithm <u>never converges</u>

Random Sample Consensus (RANSAC)

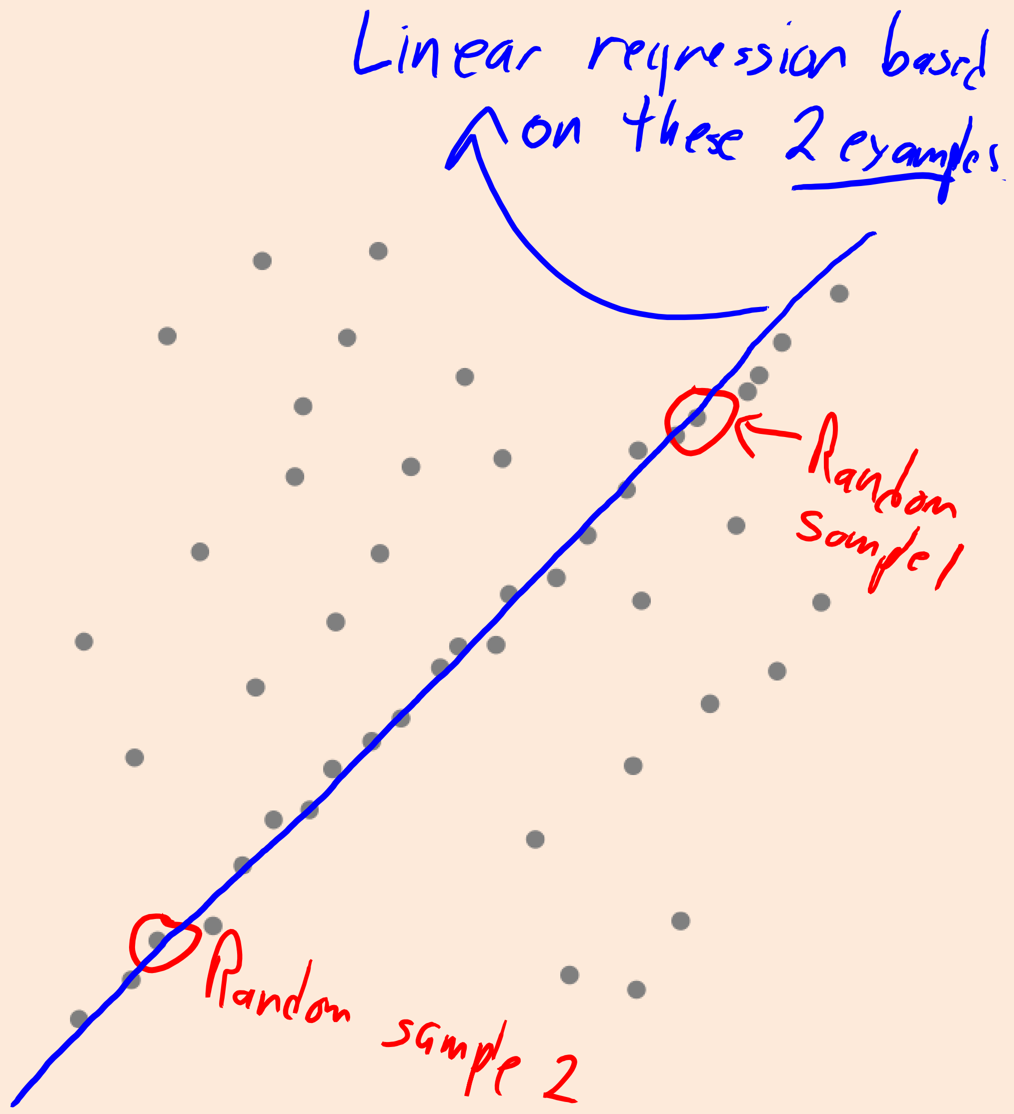
- In computer vision, a widely-used generic framework for robust fitting is random sample consensus (RANSAC).
- This is designed for the scenario where:
  - You have a large number of outliers.
  - Majority of points are “inliers”:
    it’s really easy to get low error on them.



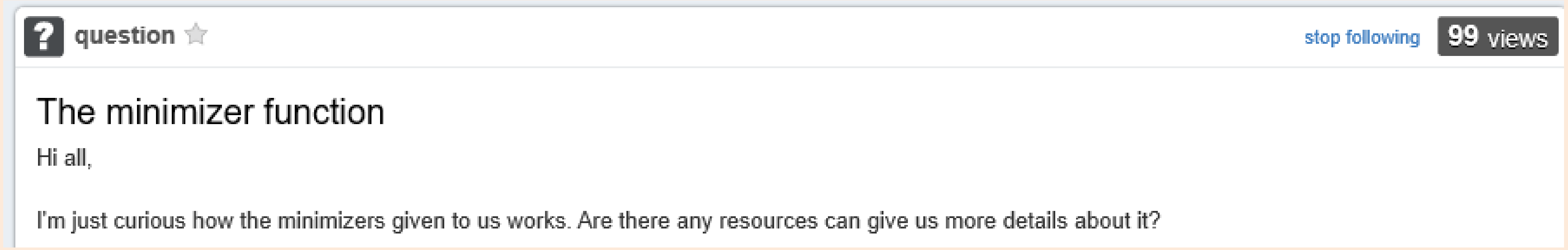
https://en.wikipedia.org/wiki/Random_sample_consensus

Random Sample Consensus (RANSAC)

- RANSAC:
  - Sample a small number of training examples.
    - Minimum number needed to fit the model.
    - For linear regression with 1 feature, just 2 examples.
  - Fit the model based on the samples.
    - Fit a line to these 2 points.
    - With 'd' features, you'll need 'd+1' examples.
  - Test how many points are fit well based on the model.
  - Repeat until we find a model that fits at least the expected number of "inliers".
- You might then re-fit based on the estimated "inliers".



Optimizer "findMin" Details



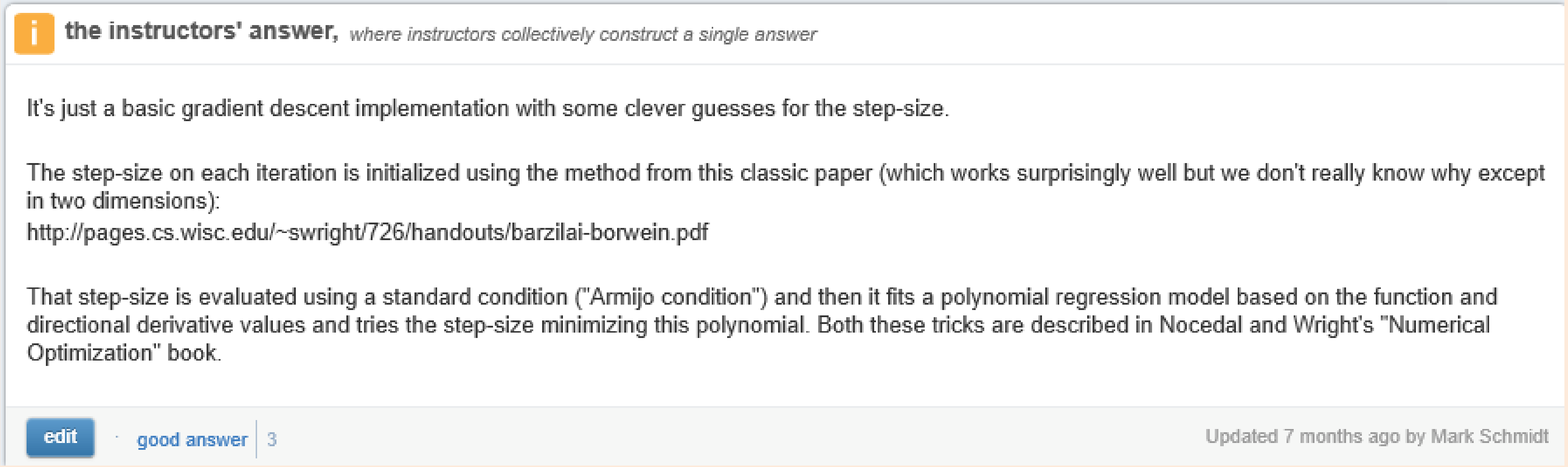
question

stop following 99 views

The minimizer function

Hi all,

I'm just curious how the minimizers given to us works. Are there any resources can give us more details about it?



the instructors' answer, *where instructors collectively construct a single answer*

It's just a basic gradient descent implementation with some clever guesses for the step-size.

The step-size on each iteration is initialized using the method from this classic paper (which works surprisingly well but we don't really know why except in two dimensions):
http://pages.cs.wisc.edu/~swright/726/handouts/barzilai-borwein.pdf

That step-size is evaluated using a standard condition ("Armijo condition") and then it fits a polynomial regression model based on the function and directional derivative values and tries the step-size minimizing this polynomial. Both these tricks are described in Nocedal and Wright's "Numerical Optimization" book.

edit · good answer | 3

Updated 7 months ago by Mark Schmidt