CPSC 340: Machine Learning and Data Mining

Gradient Descent

Fall 2016

Admin

- Assignment 1:
  - Marks up this weekend on UBC Connect.
- Assignment 2:
  - 3 late days to hand it in Monday.
- Assignment 3:
  - Due Wednesday (so we can release solutions before the midterm).
- Tutorial room change: T1D (Monday @5pm) moved to DMP 101.

- Corrections:
  - w = X\y does not compute the least squares estimate.
  - Only certain splines have an RBF representation.

Last Time: RBFs and Regularization

- We discussed radial basis functions:
  - Basis functions that depend on distances to training points:

$$y_i = w_1 \exp\left(-\frac{\|x_i - x_1\|^2}{2\sigma^2}\right) + w_2 \exp\left(-\frac{\|x_i - x_2\|^2}{2\sigma^2}\right) + \cdots + w_n \exp\left(-\frac{\|x_i - x_n\|^2}{2\sigma^2}\right)$$

$$= \sum_{j=1}^{n} w_j \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

  - Flexible bases that can model any continuous function.
- We also discussed regularization:
  - Adding a penalty on the model complexity:

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

  - Best parameter lambda almost always leads to improved test error.
    - L2-regularized least squares is also known as "ridge regression".

Features with Different Scales

- Consider features with different scales:

Egg (#)	Milk (mL)	Fish (g)	Pasta (cups)
0	250	0	1
1	250	200	1
0	0	0	0.5
2	250	150	0

- Should we convert to some standard 'unit'?
  - It doesn't matter for least squares:
    - w_j*(100 mL) gives the same model as w_j*(0.1 L)
    - w_j will just be 1000 times smaller.
  - It also doesn't matter for decision trees or naïve Bayes.

Features with Different Scales

- Consider features with different scales:

Egg (#)	Milk (mL)	Fish (g)	Pasta (cups)
0	250	0	1
1	250	200	1
0	0	0	0.5
2	250	150	0

- Should we convert to some standard 'unit'?
  – It matters for k-nearest neighbours:
    - KNN will focus on large values more than small values.
  – It matters for regularized least squares:
    - Penalization $|w_j|$ means different things if features 'j' are on different scales.

Standardizing Features

$X = \begin{bmatrix} & \boxed{\;} & \end{bmatrix}$

average of column 'j'

- It is common to standardize features:
  - For each feature:
    1. Compute mean and standard deviation:

       $$\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij} \qquad \sigma_j = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij}-\mu_j)^2}$$

    2. Subtract mean and divide by standard deviation:

       Replace x_{ij} with $\frac{x_{ij}-\mu_j}{\sigma_j}$

  - Means that change in 'w_j' have similar effect for any feature 'j'.
- Should we regularize the bias?
  - No! The y-intercept can be anywhere, why encourage it to be close to zero?
  - Yes! Regularizing all variables makes solution unique and it easier to compute 'w'.
  - Compromise: regularize the bias by a smaller amount than other variables?

    $\rightarrow \frac{1}{2}\|Xw - y\|^2 + \sum_{j=1}^{d} \lambda_j w_j$ (λ_j small for bias)

Standardizing Target

- In regression, we sometimes standardize the targets y_i.
  - Puts targets on the same standard scale as standardized features:

$$\text{Replace } y_i \text{ with } \frac{y_i - \mu_y}{\sigma_y}$$

- With standardized target, setting w = 0 predicts average y_i:
  - High regularization makes us predict closer to the average value.
- Other common transformations of y_i are logarithm/exponent:

$$\text{Use } \log(y_i) \text{ or } \exp(\gamma y_i)$$

  - Makes sense for geometric/exponential processes.

Ridge Regression Calculation

$w^T w = w^T I w$

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

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

Set $\nabla f(w) = 0$: $X^T X w + \lambda w = X^T y$

$A^{-1}A = I$

or

$X^T X w + \lambda I w = X^T y$

or

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

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

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

Matlab:

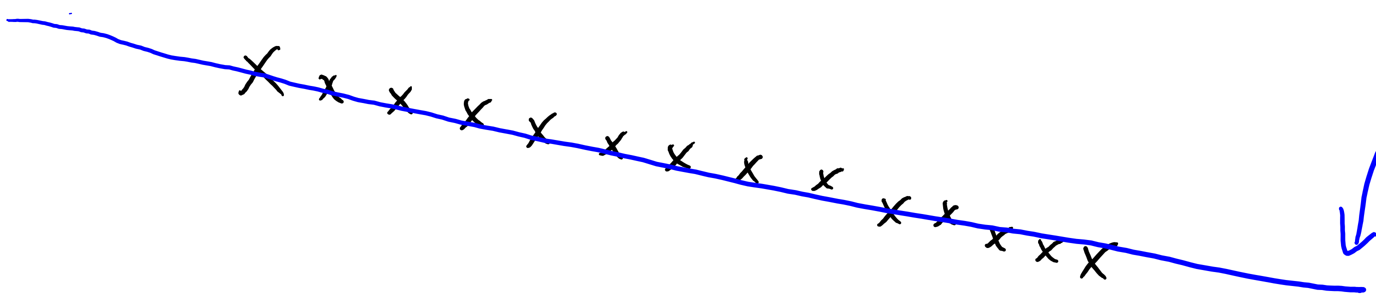
```
w=(X'*X+lambda*eye(d))\(X'*y)
```

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

Least Squares with Outliers

- Consider least squares problem with outliers:

x ← "outlier" that doesn't follow trend

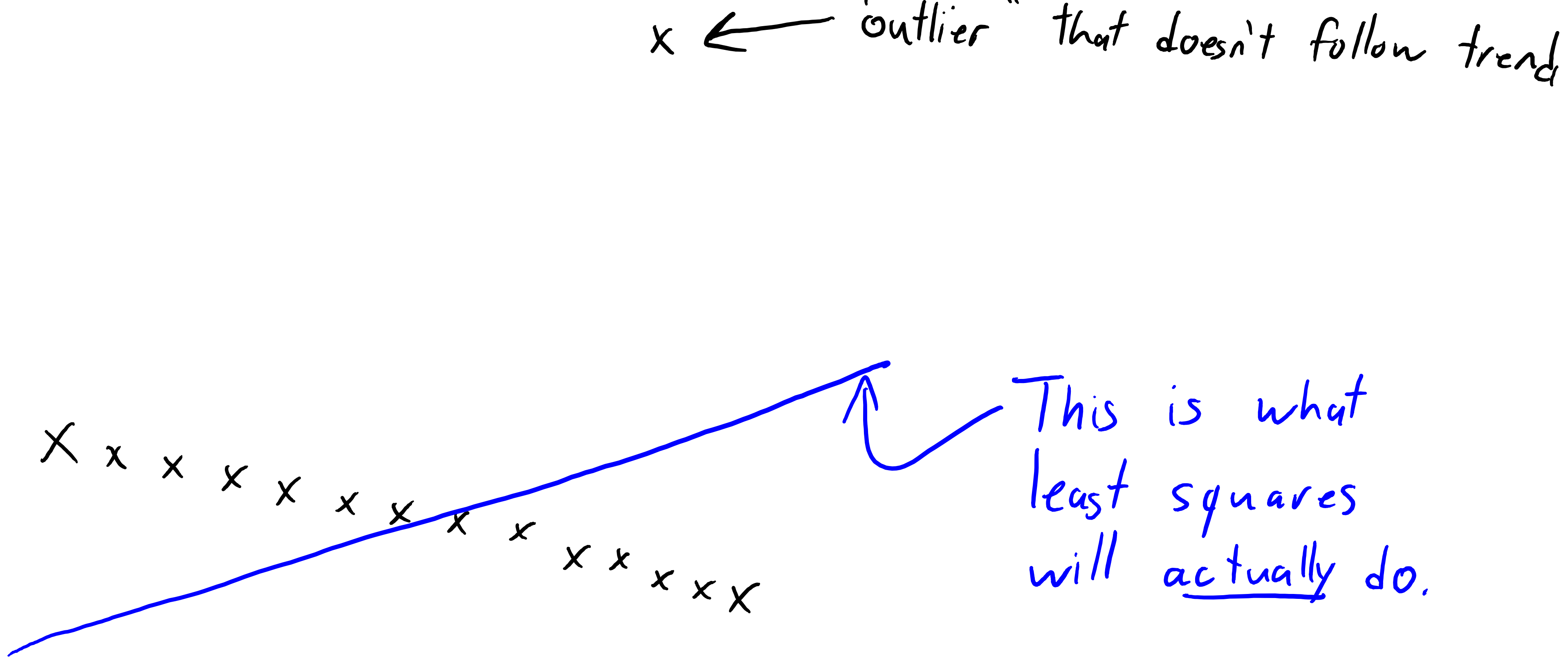


This is what we want least squares to do.

http://setosa.io/ev/ordinary-least-squares-regression

Least Squares with Outliers

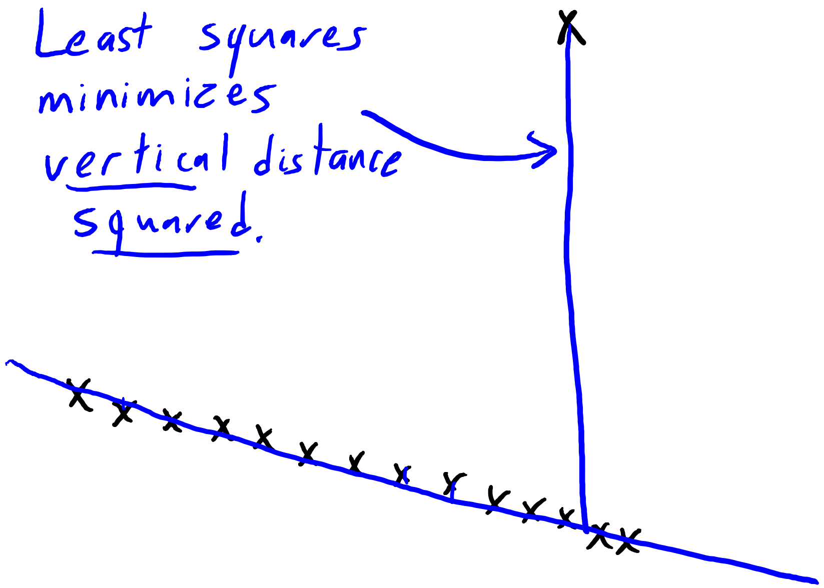
- Consider least squares problem with outliers:

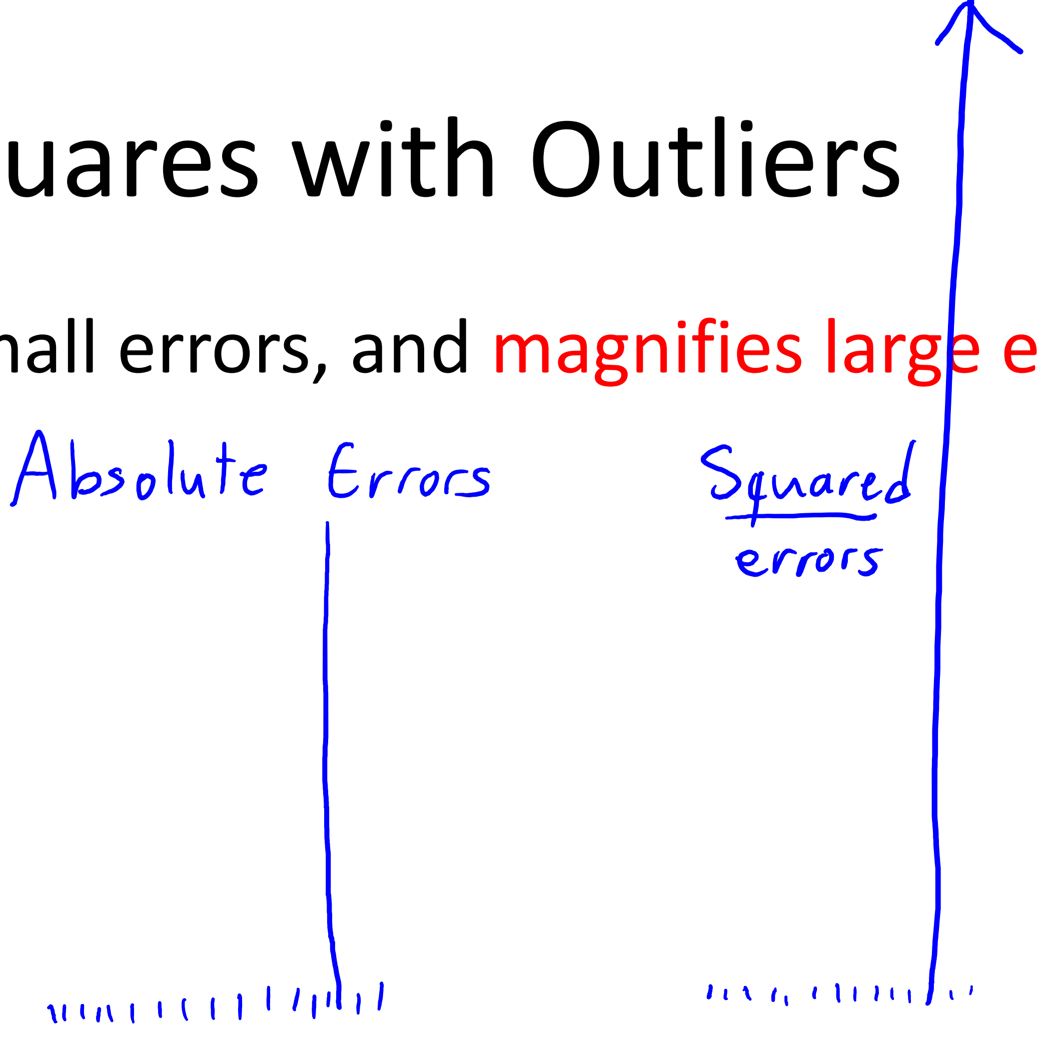


- Least squares is very sensitive to outliers.

Least Squares with Outliers

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

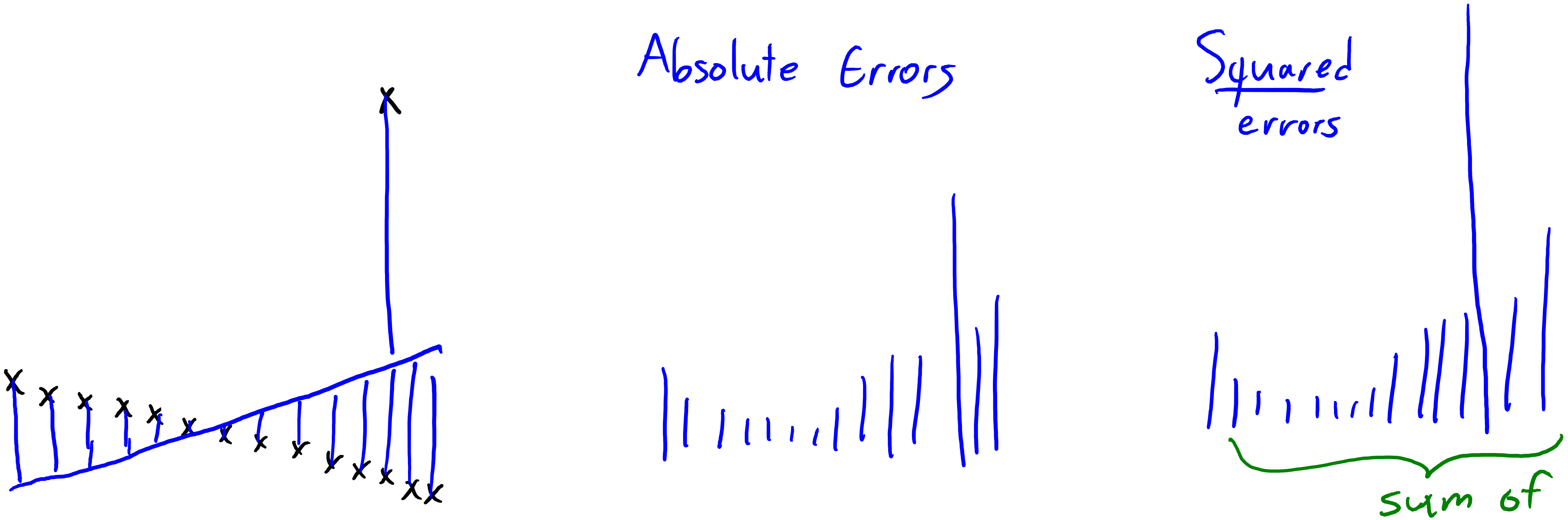




- Outliers (large error) influence 'w' much more than other points.

Least Squares with Outliers

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



sum of these is smaller than for the "correct" line.

- 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 large errors (outliers).
- For example, use absolute error instead of squared error:

$$f(w) = \sum_{i=1}^{n} |w^T x_i - y_i|$$

- Now decreasing 'small' and 'large' errors is equally important.
- Instead of minimizing L2-norm, minimizes L1-norm of residuals:

Least squares:

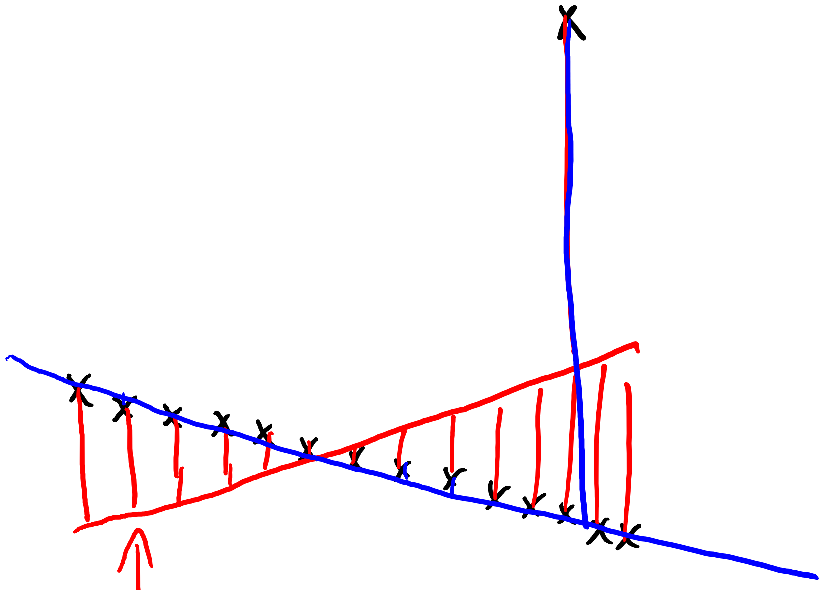
$$f(w) = \frac{1}{2}\|Xw - y\|^2$$

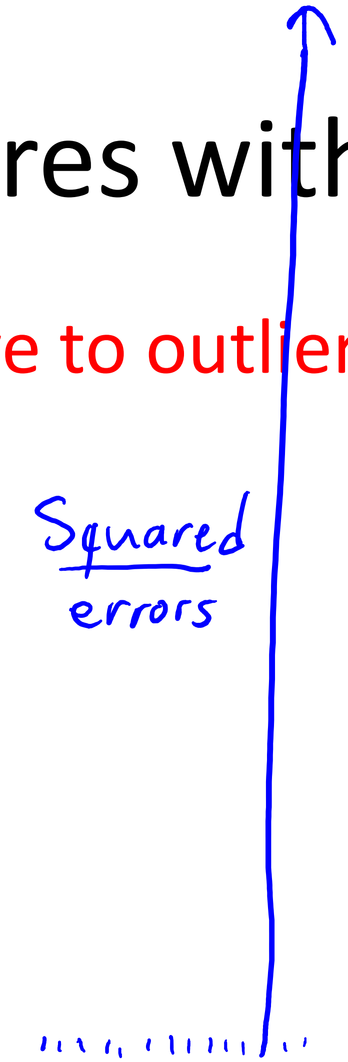
Least absolute error:

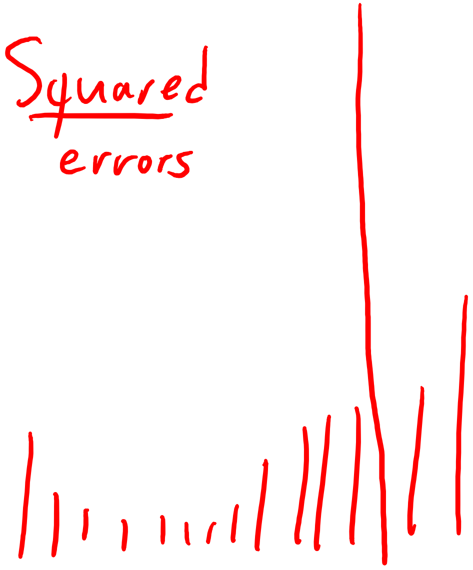
$$f(w) = \|\underbrace{X}_{n \times d}\underbrace{w}_{d \times 1} - \underbrace{y}_{n \times 1}\|_1$$

Least Squares with Outliers

- Least squares is very sensitive to outliers.



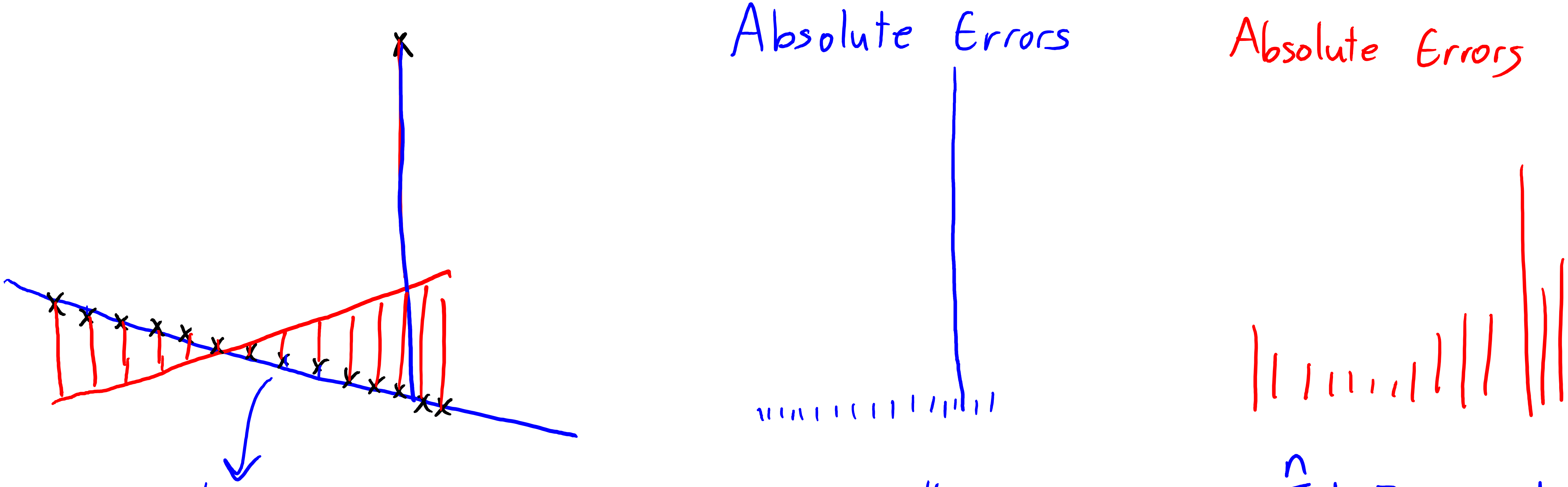




Linear model 'w' minimizing $f(w) = \frac{1}{2}\|Xw - y\|^2$

Least Squares with Outliers

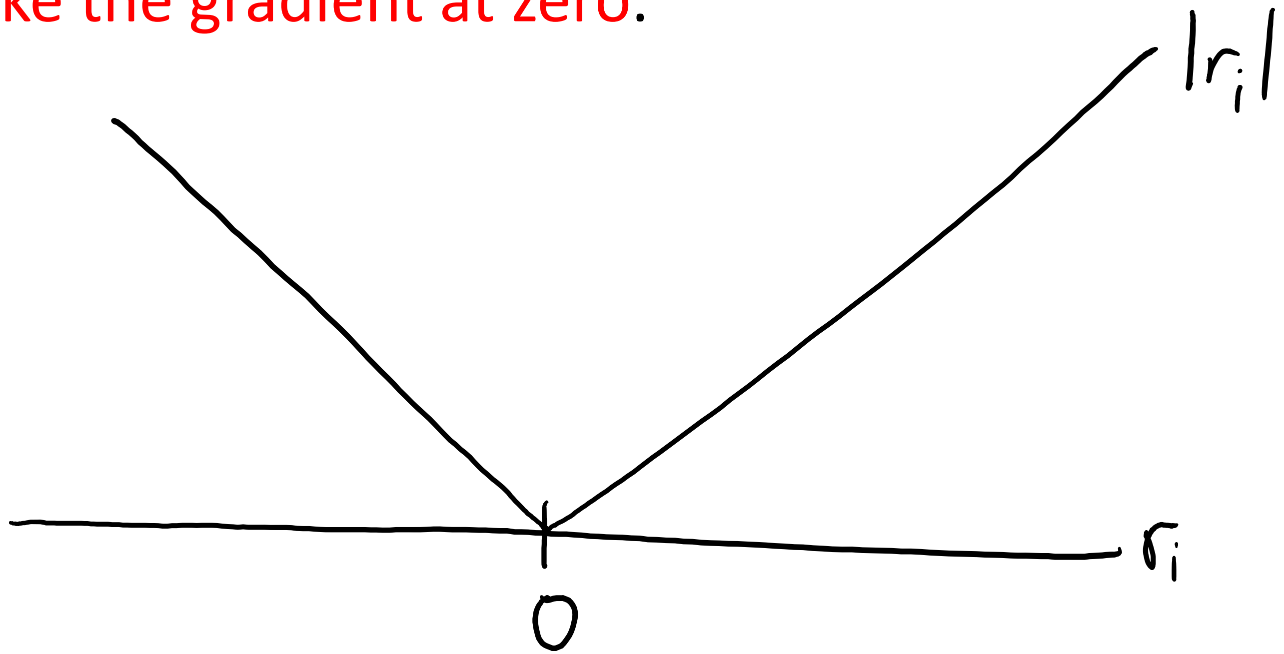
- Absolute error is more robust to outliers:



Linear model 'w' minimizing $f(w) = \|Xw - y\|_1 = \sum_{i=1}^{n} |w^T x_i - y_i|$

Regression with the L1-Norm

- Unfortunately, minimizing the absolute error is harder:
  - We can't take the gradient at zero.



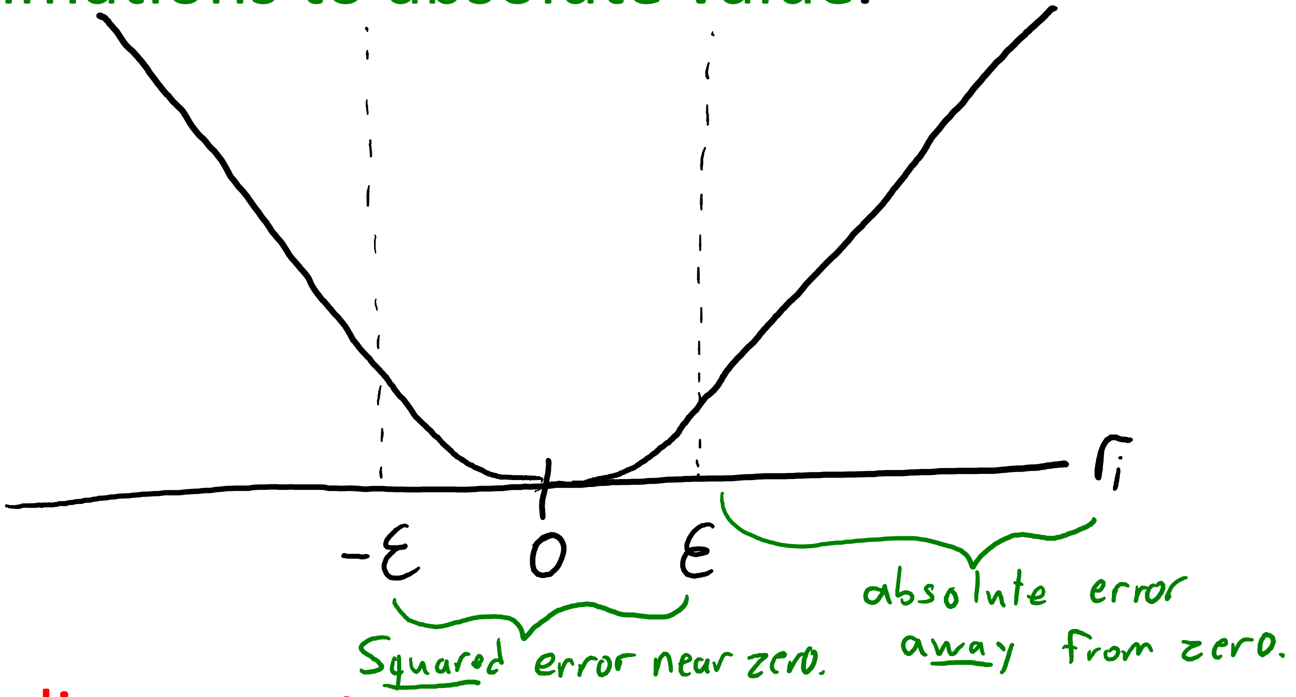
  - Generally, harder to minimize non-smooth than smooth functions.
  - Could solve as 'linear program', but harder than 'linear system'.

Smooth Approximations to the L1-Norm

- There are differentiable approximations to absolute value.
- For example, the Huber loss:

$$f(w) = \sum_{i=1}^{n} h(w^T x_i - y_i)$$

$$h(r_i) = \begin{cases} \frac{1}{2} r_i^2 & \text{for } |r_i| \leq \epsilon \\ \epsilon\left(|r_i| - \frac{1}{2}\epsilon\right) & \text{otherwise} \end{cases}$$



- Setting $\nabla f(x) = 0$ does not give a linear system.
- But we can minimize 'f' using gradient descent:
  - Algorithm for finding local minimum of a differentiable function.

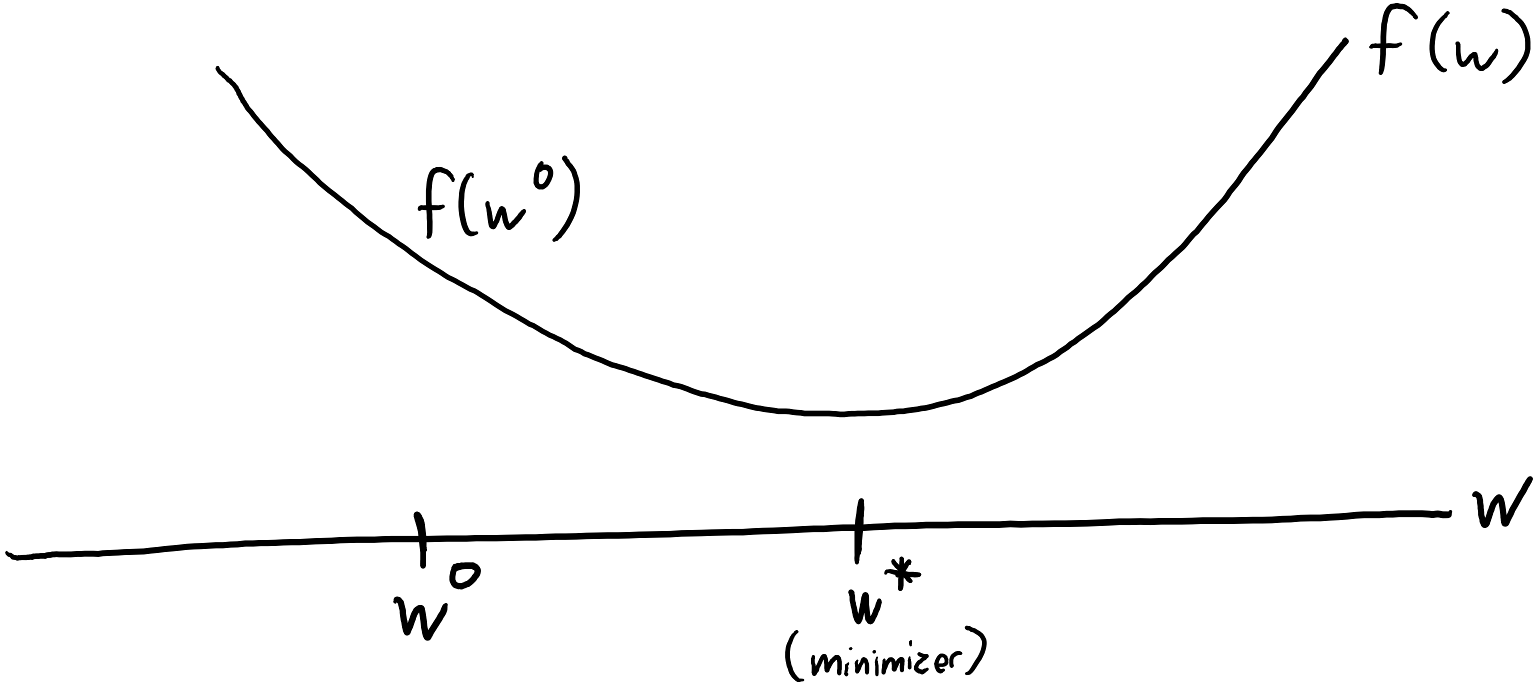
Gradient Descent for Finding a Local Minimum

- Gradient descent is an iterative optimization algorithm:
  - It starts with a "guess" w^0.
  - It uses w^0 to generate a better guess w^1.
  - It uses w^1 to generate a better guess w^2.
  - It uses w^2 to generate a better guess w^3.

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

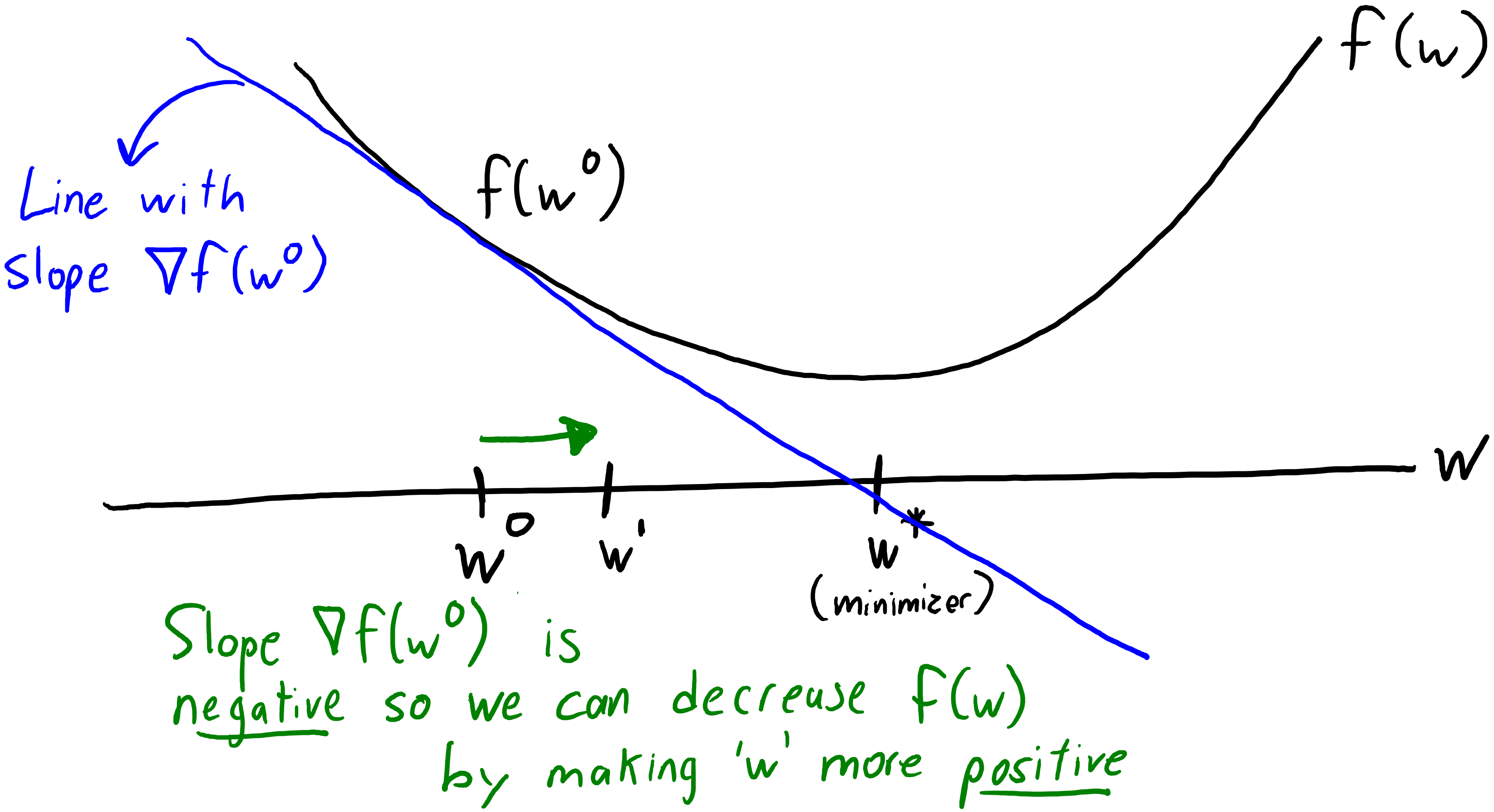
Gradient Descent for Finding a Local Minimum

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



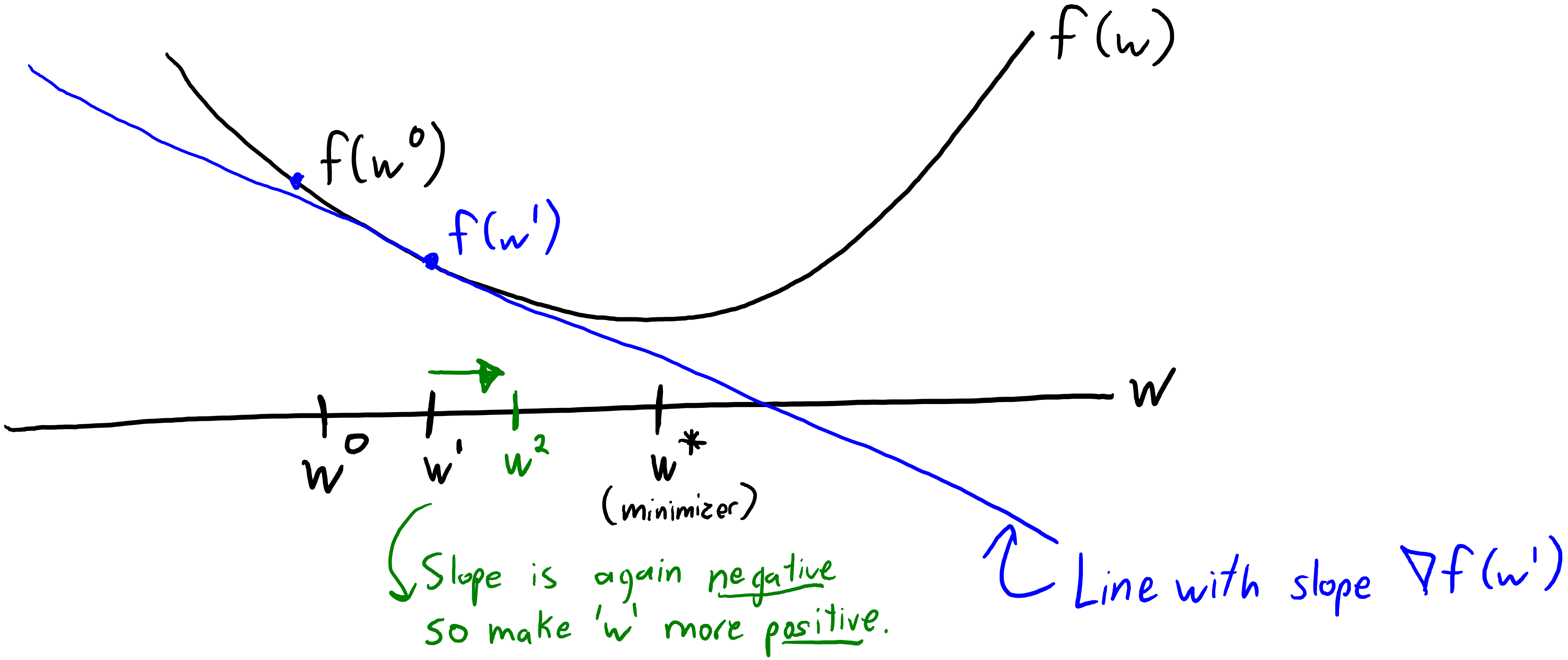
Gradient Descent for Finding a Local Minimum

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



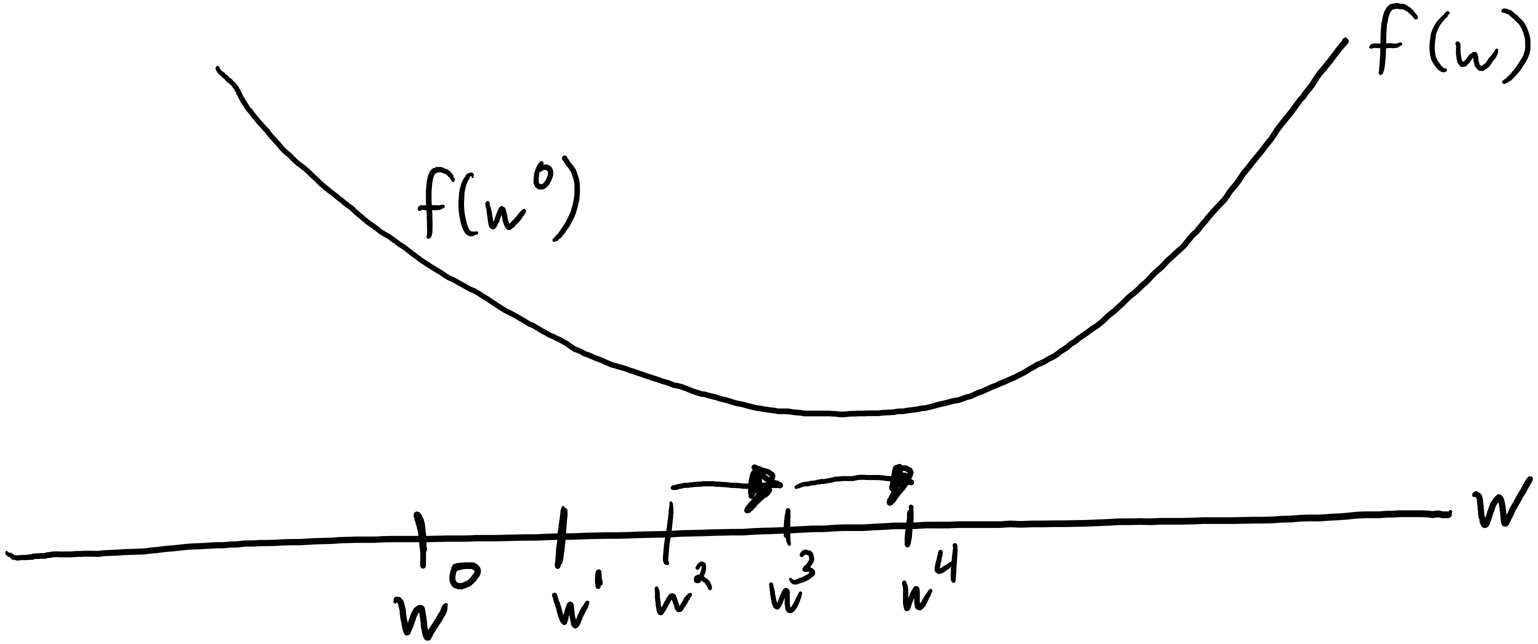
Gradient Descent for Finding a Local Minimum

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



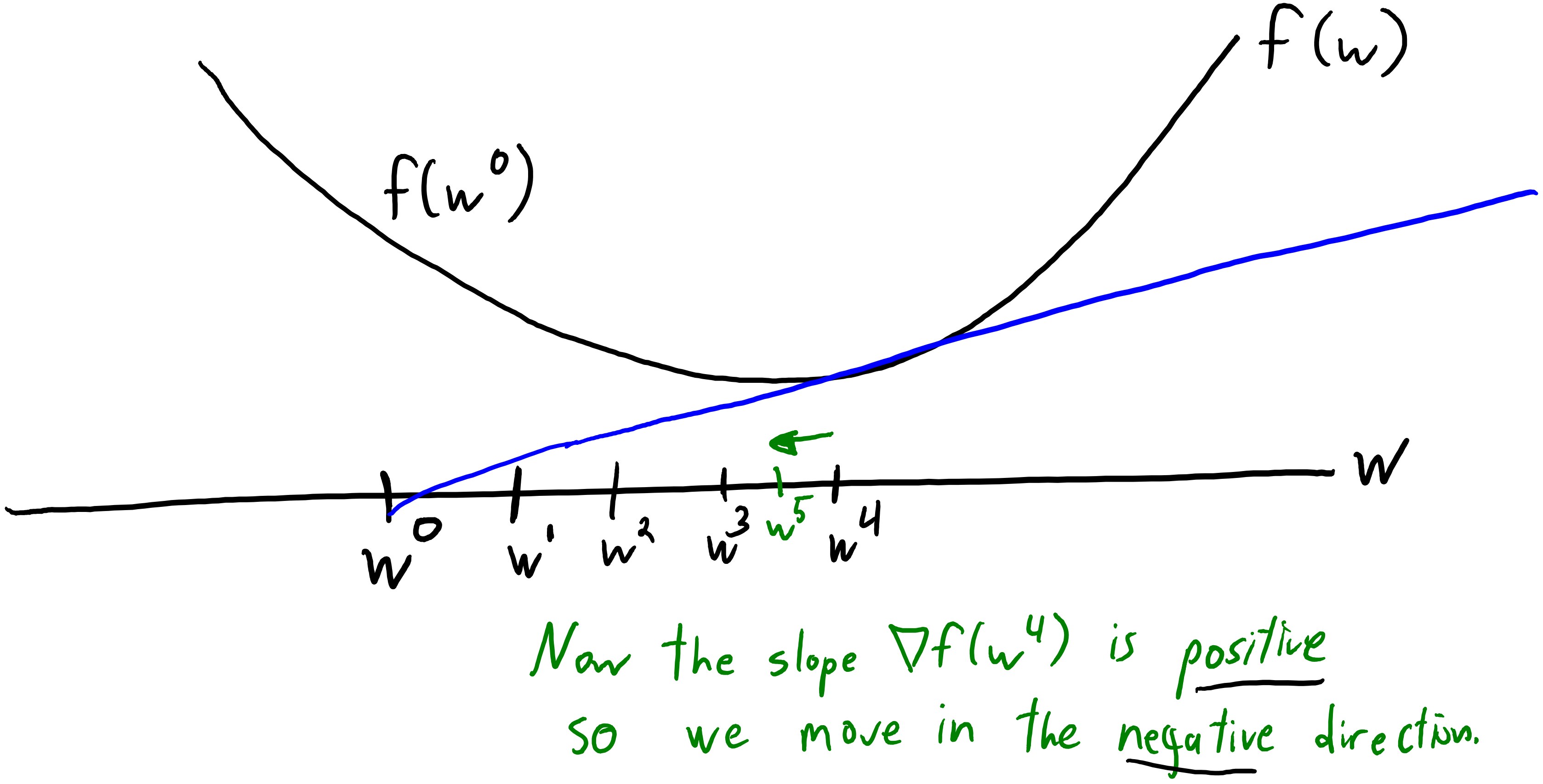
Gradient Descent for Finding a Local Minimum

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



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

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

(scalar α^0 is the `step size', we decrease 'f' for small enough α^0)

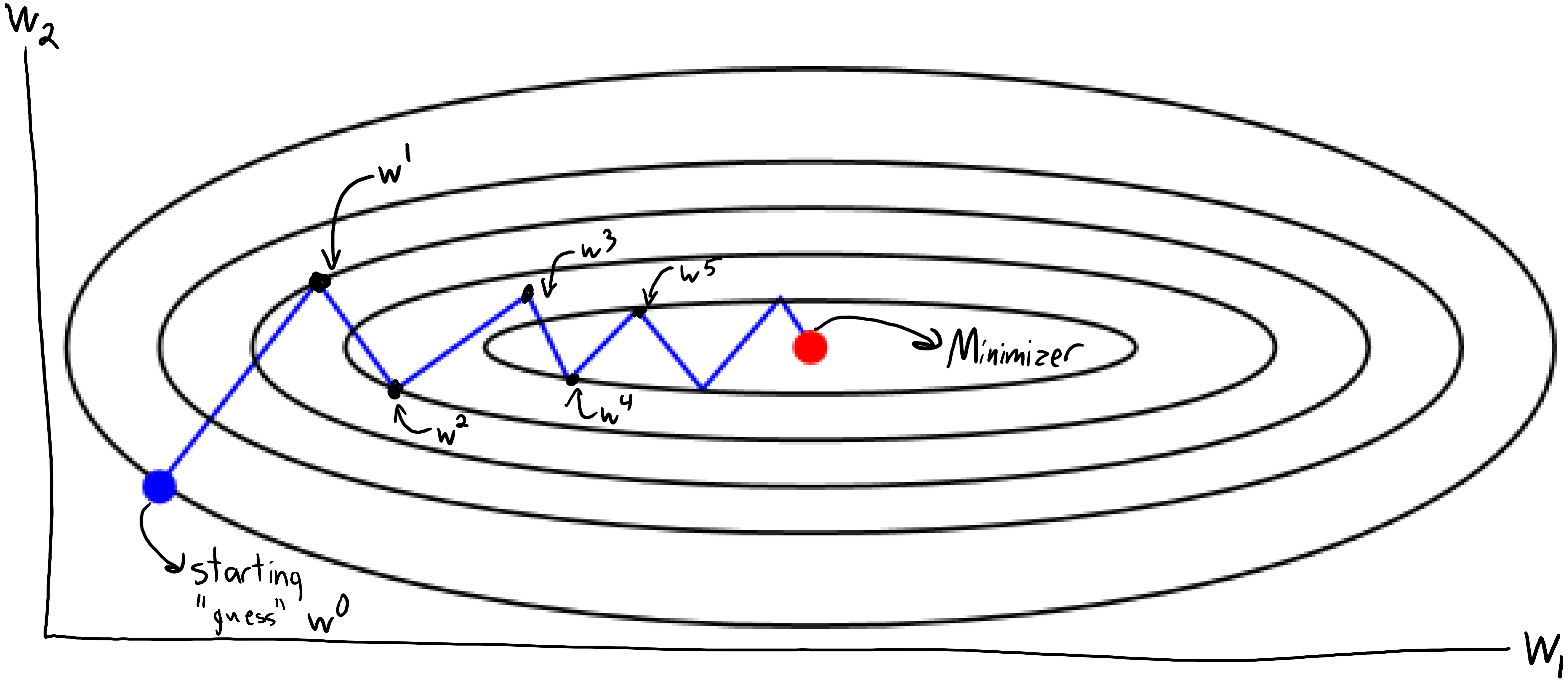
  - Repeat to successively refine the guess:

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

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

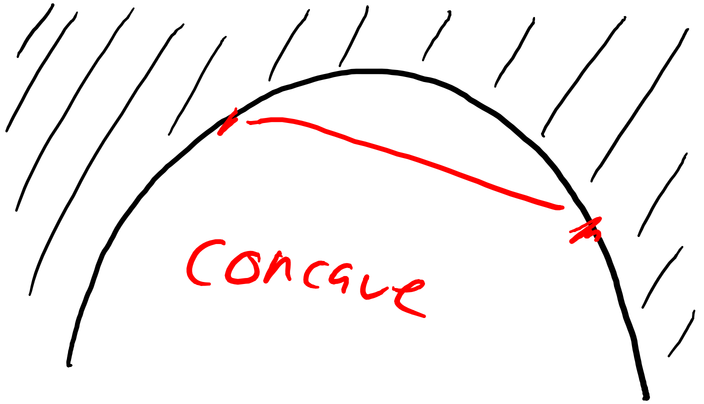
$\delta \rightarrow$ Some small scalar.

Gradient Descent in 2D

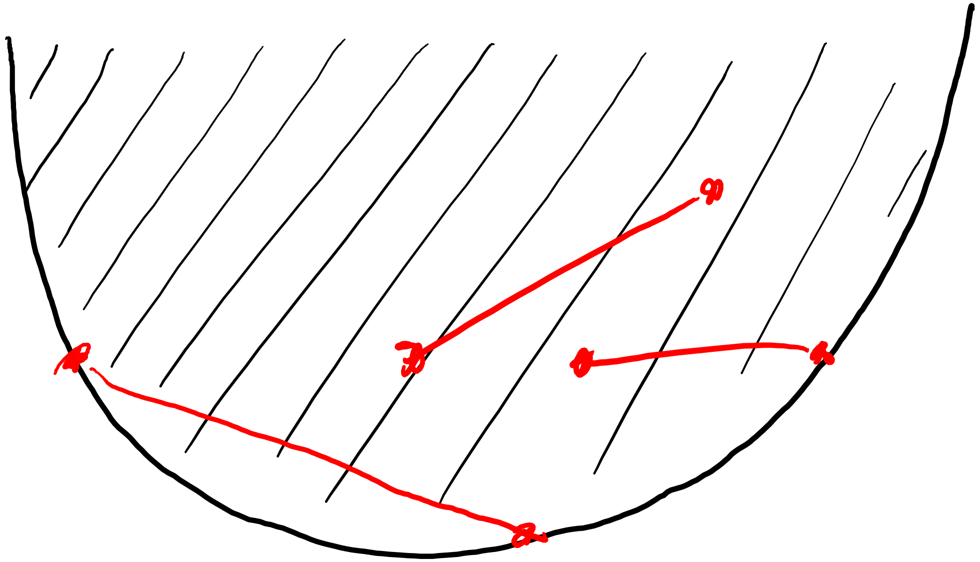


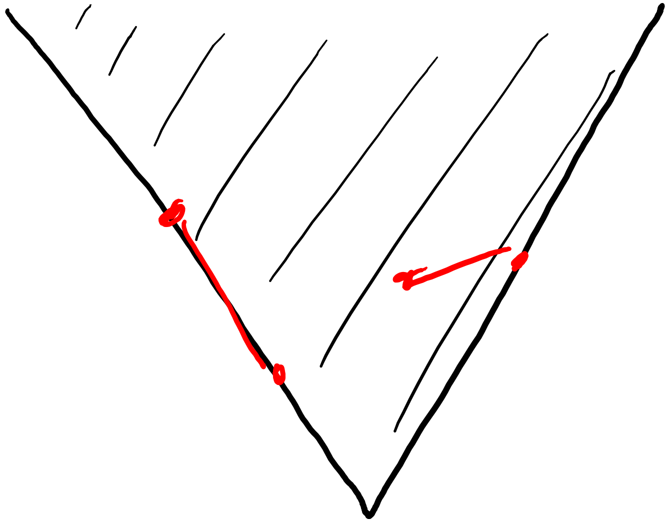
- Under weak conditions, algorithm converges to a local minimum.

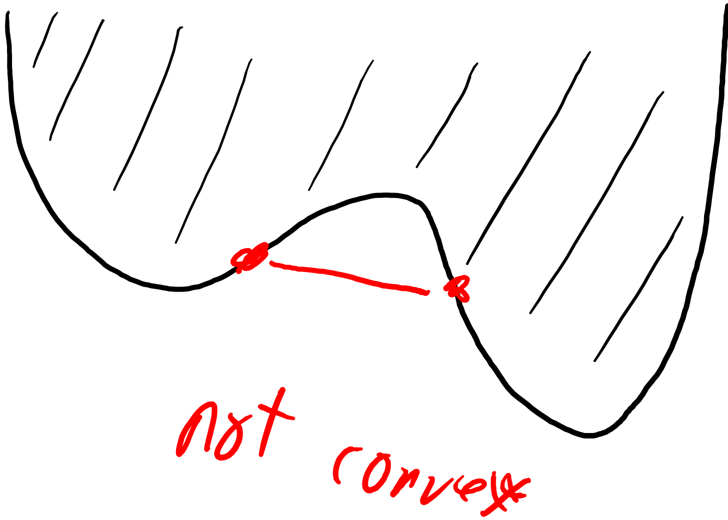
Convex Functions



- Is finding a local minimum good enough?
  - For least squares and Huber loss this is enough: they are 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.

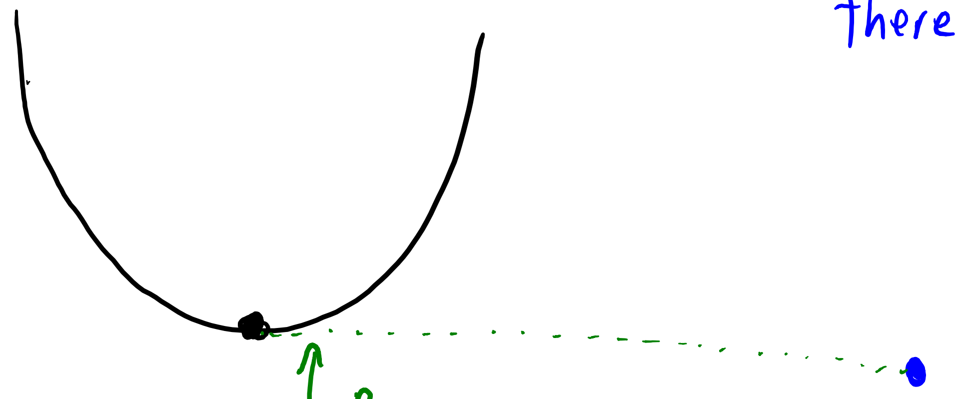
Convex Functions

- All local minima of convex functions are also global minima.

Proof by contradiction:

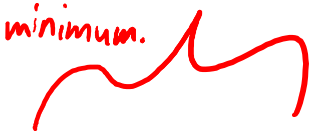
Consider a local minimum

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.



  - Gradient descent finds a global minimum on convex functions.
  - Next time: how do we know if a function is convex?

Gradient Descent

- Least squares via normal equations vs. gradient descent:
  - Normal equations cost $O(nd^2 + d^3)$.

    Forming X^TX costs $O(nd^2)$ and solving a $d \times d$ linear system costs $O(d^3)$

  - Gradient descent costs $O(ndt)$ to run for 't' iterations.

    We can use $X^TXw = X^T(Xw)$ which is just two $n \times d$ matrix multiplications.

    Computing $\nabla f(w) = X^TXw - X^Ty$ only costs $O(nd)$.

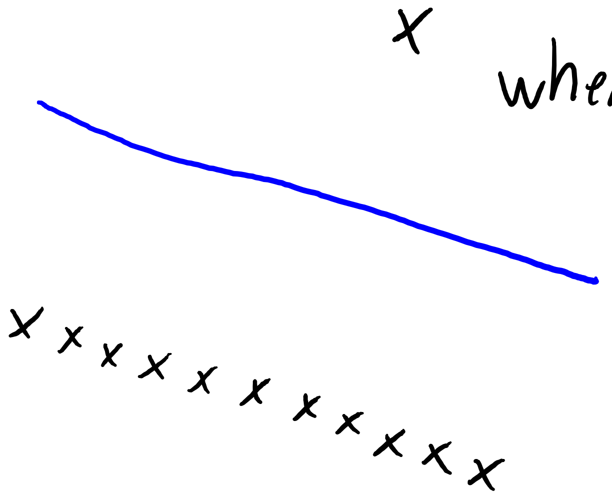
  - Gradient descent can be faster when 'd' is very large:
    - Faster if solution is "good enough" for ($t < d$) and ($t < d^2/n$).
- Improving on gradient descent: Nesterov and Newton method.
  - For L2-regularized least squares, there is also "conjugate" gradient.

Motivation for Considering Worst Case

‘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:

$$f(w) = \|Xw - y\|_\infty$$



where $\|r\|_\infty = \max_i \{|r_i|\}$

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

Log-Sum-Exp Function

- As with the L_1-norm, the L_∞-norm is convex but non-smooth:
  - We can fit it with gradient descent using a smooth approximation.

- Log-sum-exp function is a smooth approximation to max function:

$$\max_i \{z_i\} \approx \log\left(\sum_i \exp(z_i)\right)$$

- Intuition:
  - $\sum_i \exp(z_i) \approx \max_i \exp(z_i)$, as largest element is magnified exponentially.
  - Recall that $\log(\exp(z_i)) = z_i$.

- Notation alert: when I write "log" I always mean natural logarithm:

$$\log(\exp(\alpha)) = \alpha$$

Summary

- Robust regression using L1-norm/Huber is less sensitive to outliers.
- Gradient descent finds local minimum of differentiable function.
- Convex functions do not have non-global local minima.
- Log-Sum-Exp function: smooth approximation to maximum.

- Next time:
  - Finding ‘important’ e-mails, and beating naïve Bayes on spam filtering.

Bonus Slide: Invertible Matrices and Regularization

- Unlike least squares where X^TX may not be invertible, the matrix $(X^TX + \lambda I)$ in always invertible.

- We prove this by showing that $(X^TX + \lambda I)$ is positive-definite, meaning that $v^T(X^TX + \lambda I)v > 0$ for all non-zero 'v'. (Positive-definite matrices are invertible.)

With a generic 'v' such that $v \neq 0$ we have

$$v^T(X^TX + \lambda I)v = v^TX^TXv + \lambda v^Tv$$

$$= \underbrace{\|Xv\|^2}_{\geq 0} + \underbrace{\lambda \sum_{j=1}^{d} v_j^2}_{> 0 \text{ since } v \neq 0.}$$

Bonus Slide: Log-Sum-Exp for Brittle Regression

- To use log-sum-exp for brittle regression:

$$\|Xw - y\|_\infty = \max_i \{|w^T x_i - y_i|\}$$

$$= \max_i \{\max\{w^T x_i - y_i, y_i - w^T x_i\}\}$$ since $|z| = \max\{z, -z\}$

$$= \log\left(\sum_{i=1}^n \exp(w^T x_i - y_i) + \sum_{i=1}^n \exp(y_i - w^T x_i)\right)$$ using log-sum-exp to approximate "max" over $2n$ terms.

Bonus Slide: Log-Sum-Exp Numerical Trick

- Numerical problem with log-sum-exp is that $\exp(z_i)$ might overflow.
  - For example, $\exp(100)$ has more than 40 digits.
- Implementation 'trick': Let $\beta = \max_i \{z_i\}$

$$
\begin{aligned}
\log\left(\sum_i \exp(z_i)\right) &= \log\left(\sum_i \exp(z_i - \beta + \beta)\right) \\
&= \log\left(\sum_i \exp(z_i - \beta)\exp(\beta)\right) \\
&= \log\left(\exp(\beta)\sum_i \exp(z_i - \beta)\right) \\
&= \log(\exp(\beta)) + \log\left(\sum_i \exp(z_i - \beta)\right) \\
&= \beta + \log\left(\sum_i \exp(z_i - \beta)\right)
\end{aligned}
$$

≤ 1 so no overflow

Bonus Slide: 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>

Bonus Slide: Gradient Descent for Non-Smooth?

- "You are unlikely to land on a non-smooth point, so gradient descent should work for non-smooth problems?"
  - Counter-example from Bertsekas' "Nonlinear Programming"

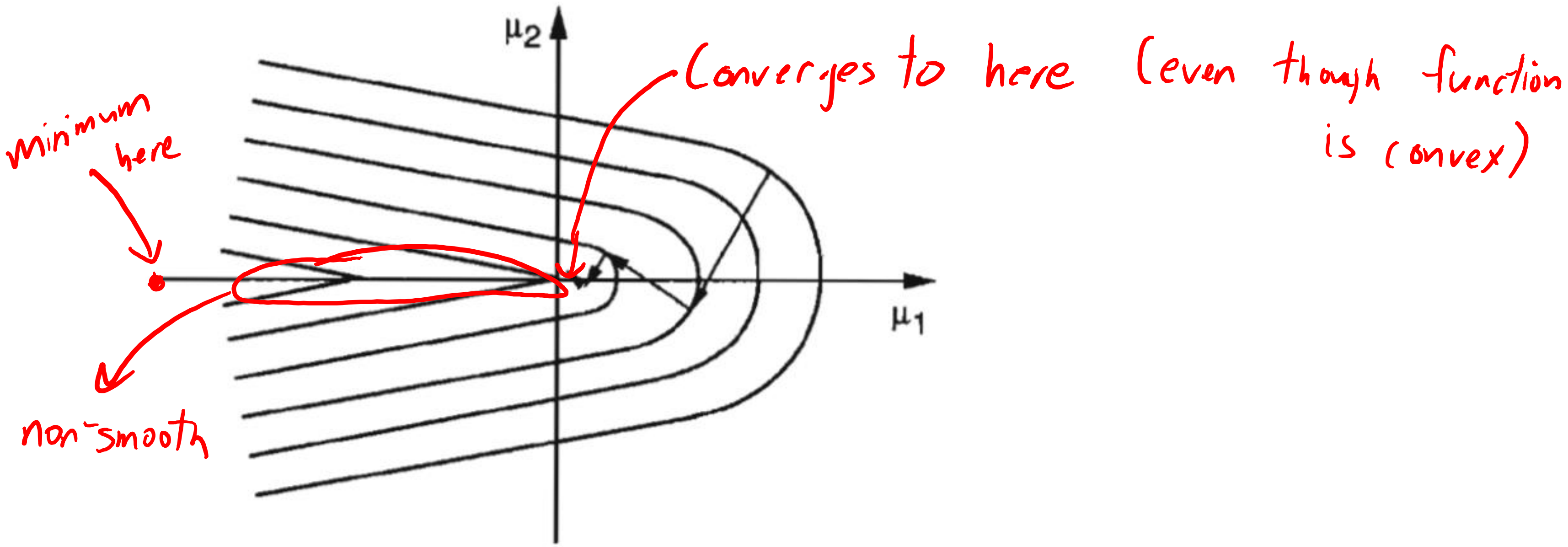


Figure 6.3.8. Contours and steepest ascent path for the function of Exercise 6.3.8.