CPSC 540: Machine Learning

First-Order Methods, L1-Regularization, Coordinate Descent

Winter 2016

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

Admin

- Room: We'll count final numbers today and look for a new one.
- Assignment 1:
  - Due now (via handin).
  - You can use 1 of your 3 late days to hand it in before Thursday's class.
- Assignment 2:
  - Out tomorrow.
  - Due February 2nd.
  - Start early!

Last Time: Convex Functions

- Last time we discussed convex functions:
  - All local minima are global minima (and no saddle points).
  - Three definitions of convex functions (depending on differentiability):

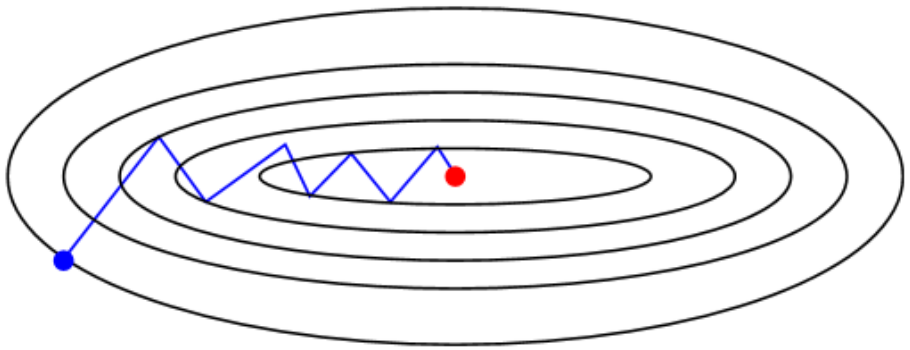
    1. $f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y)$ for all x and y, and $0 \leq \theta \leq 1$.
    2. Once-differentiable and $f(y) \geq f(x) + \nabla f(x)^T (y-x)$ for all x and y.
    3. Twice-differentiable and $\nabla^2 f(x) \succeq 0$ for all x (symmetric positive semi-definite)

  - We discussed ways to show functions are convex:
    - Show one of the above holds.
    - Use operations that preserve convexity.
      - Non-negative sum, composition with affine function, maximum.

Last Time: Gradient Descent

- Gradient descent:
  - Iterative algorithm for finding stationary point of differentiable function.
  - For convex functions it finds a global minimum.

Start with x^0, apply $x^{t+1} = x^t - \alpha_t \nabla f(x^t)$



- Cost of algorithm scales linearly with number of variables 'd':
  - E.g., 't' iterations costs O(ndt) for least squares, logistic regression, etc.
    - Note that the input size is O(nd).
  - For t < d, faster than $O(nd^2 + d^3)$ of least squares and Newton's method.

    Faster in high-dimensions for small 't'.

Last Time: Convergence Rate of Gradient Descent

- We asked "how many iterations 't' before we have an accuracy ε?"
- We assumed strong-convexity and strong-smoothness:

$$\mu I \preceq \nabla^2 f(x) \preceq LI \quad \text{for all } x \text{ and } 0 < \mu \leq L < \infty.$$

identity matrix

($A \succeq B$ means that $y^T A y - y^T B y \geq 0$ for all y)

So $LI \succeq \nabla^2 f(x)$ means that $y^T (LI) y - y^T \nabla^2 y \geq 0$

or $L\|y\|^2 \geq y^T \nabla^2 f(x) y$ for all y.

- By using multivariate 2nd-order Taylor expansion,

$$f(y) = f(x) + \nabla f(x)^T (y-x) + \frac{1}{2}(y-x)^T \nabla^2 f(z)(y-x)$$

for some z for any x and y,

we showed linear convergence rate which implies $t = O(\log(1/\varepsilon))$.

Weaker Assumptions for Linear Convergence

- We can get a linear convergence rate under weaker assumptions:
  - Proof works for any $\alpha < 2/L$.
    - Don't need 'L', just need step-size α small enough.
    - But optimal step-size in proof is $\alpha = 1/L$.
  - Proof works if you take the optimal step-size.

$$\alpha^* = \underset{\alpha > 0}{\operatorname{argmin}} \{ f(x^t + \alpha \nabla f(x^t)) \} \Rightarrow f(x^t + \alpha^* \nabla f(x^t)) \leq f(x^t + \tfrac{1}{L} \nabla f(x^t))$$

    - You can compute this for quadratics: just minimizing a 1D quadratic.
  - Proof can be modified to work approximation of 'L' or line-search.
    - What you typically do in practice.

Weaker Assumptions for Linear Convergence

- We can get a linear convergence rate under weaker assumptions:
  - Proof works for once-differentiable 'f' with L-Lipschitz continuous gradient:

Gradient does not change too quickly: $\|\nabla f(x) - \nabla f(y)\| \leq L\|x-y\|$ for all x and y.

Since this implies: $f(y) \leq f(x) + \nabla f(x)^T(y-x) + \frac{L}{2}\|y-x\|^2$ for all y and x.

(see Nesterov's "Introductory Lectures on Convex Optimization")

  - This doesn't need to hold globally, proof works if we can show:

$$f(x^{t+1}) \leq f(x^t) + \nabla f(x^t)^T(x^{t+1}-x^t) - \frac{L}{2}\|x^{t+1}-x^t\|^2$$ for some L and all x^t and x^{t+1}.

  - Basically, for differentiable functions this is a very weak assumption.

Weaker Assumptions for Linear Convergence

- We can get a linear convergence rate under weaker assumptions:
  - Strong-convexity is defined even for non-differentiable functions:

We say 'f' is μ-strongly convex if $f(x) - \frac{\mu}{2}\|x\|^2$ is a convex function of x.

  - For differentiable functions this is equivalent to:

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{\mu}{2}\|y - x\|^2 \quad \text{for } x \text{ and } y$$

  - This is still a strong assumption:
    - But note if 'f' is convex then 'f(x) + $(\lambda)||x||^2$ is λ-strongly convex.
  - What about non-convex functions?
    - Proof works if gradient grows quickly as you move away from solution.
    - Two phase analysis: prove that algorithm gets near minimum, then analyze local rate.
      - Convergence rate only applies for 't' large enough.

How Hard is Optimization?

- Consider a generic optimization problem:

$$\underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \; f(x)$$

- Assume that a solution 'x^*' exists.
- Assume a "black-box" optimization algorithm:
  - At step 't', algorithms chooses parameters x^t and receives $f(x^t)$.
- How many steps does it take before we find ε-optimal solution?

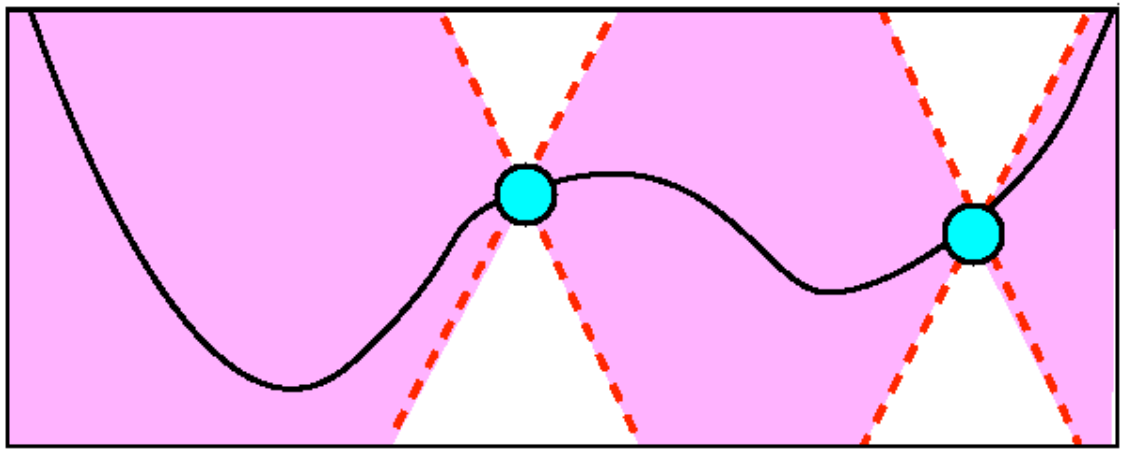
$$f(x^t) - f(x^*) \leq \varepsilon$$

- General function: impossible!

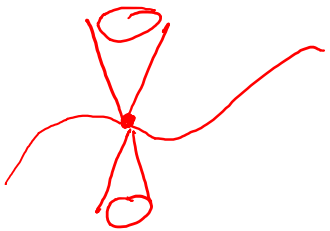
How Hard is Optimization?

- We need to make some assumptions about the function
- Typically, we assume function or gradient can't change too quickly.
  - E.g., function 'f' is Lipschitz-continuous:

$$|f(x) - f(y)| \leq L\|x - y\| \quad \text{for some 'L' and } \underline{\text{all}} \text{ 'x' and 'y'}$$



In two-dimensions, Lipschitz rules out cones:



  - Over $[0,1]^d$, now it's possible to solve the problem in $O(1/\varepsilon^d)$:
    - Exponential in dimensionality, but a small assumption made a bit difference.

Continuous Optimization Zoo

Assumptions	Algorithm	Rate
f is L-Lipschitz, x is bounded	Grid-search	$O(1/\varepsilon^d)$
f is convex but non-smooth	Sub-gradient	$O(1/\varepsilon^2)$
smooth approximation to non-smooth f, f is convex	Gradient	$O(1/\varepsilon^2)$
	Nesterov	$O(1/\varepsilon)$
∇f is L-Lipschitz, f is convex	Gradient	$O(1/\varepsilon)$
	Nesterov	$O(1/\sqrt{\varepsilon})$
f is strongly-convex but non-smooth	Sub-gradient	$O(1/\varepsilon)$
∇f is L-Lipschitz, f is μ-strongly convex	Gradient	$O(\log(\frac{1}{\varepsilon}))$
	Nesterov	$O(\log(\frac{1}{\varepsilon}))$
∇f is L-Lipschitz, f is μ-strongly convex, $\nabla^2 f$ is M-Lipschitz	Quasi-Newton	$O(\log(\frac{1}{\varepsilon}))$

sublinear

linear

superlinear

convexity

better algorithm

smoothness

strong-convexity

strong convexity

approximating 2nd derivatives, but cost is $O(d^2)$.

Gradient Method: Practical Issues

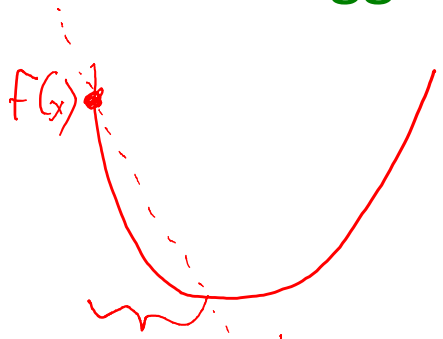
- In practice, you should never use $\alpha = 1/L$.
  - Often you don't know L.
  - Even if did, "local" L may be much smaller than "global" L: use bigger steps.
- Practical options:
  - Adaptive step-size:
    - Start with small 'L' (e.g., L = 1).
    - Double 'L' it if the guaranteed progress inequality from proof is not satisfied:

$$f(x^{t+1}) \leq f(x_t) - \frac{1}{2L}\|\nabla f(x^t)\|^2$$

    - Usually, end it up with much smaller 'L': bigger steps and faster progress.
    - With this strategy, step-size never increases.

Gradient Method: Practical Issues

- In practice, you should never use $\alpha = 1/L$.
  - Often you don't know L.
  - Even if did, "local" L may be much smaller than "global" L: use bigger steps.
- Practical options:
  - Armijo backtracking line-search:
    - On *each* iteration, start with large step-size α.
    - Decreasing α if Armijo condition is not satisfied:



makes sure step is small enough

$$f(x^{t+1}) \leq f(x_t) - \alpha\gamma\|\nabla f(x^t)\|^2 \quad \text{for some } \gamma \in (0, 1/2], \text{ usually } 10^{-4}$$

guess for next value of α

Least squares fit to functions and directional derivs

    - Works very well, particularly if you cleverly initialize/decrease α.
      - Fit linear regression to 'f' as α changes under (quadratic or cubic) basis, set α to minimum.
    - Even more fancy line-search: Wolfe conditions (makes sure α is not too small).

Gradient Method: Practical Issues

- Gradient descent codes requires you to write objective/gradient:

```
function [nll,g] = logisticGrad(w,X,y)
yXw = y.*(X*w);

% Function value
nll = sum(log(1+exp(-yXw)));

% Gradient
g = -X'*(y./(1+exp(yXw)));
end
```

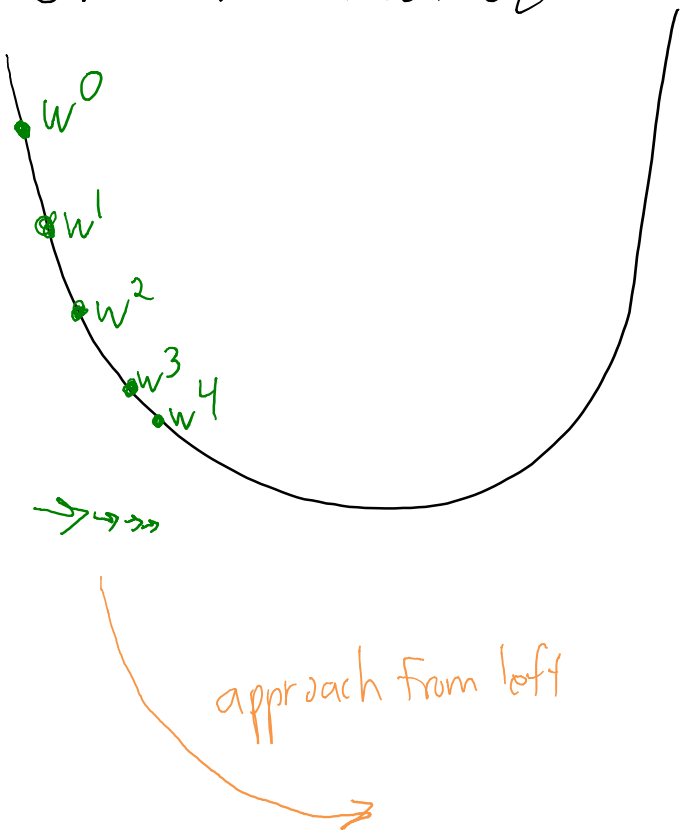
$$f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$$

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

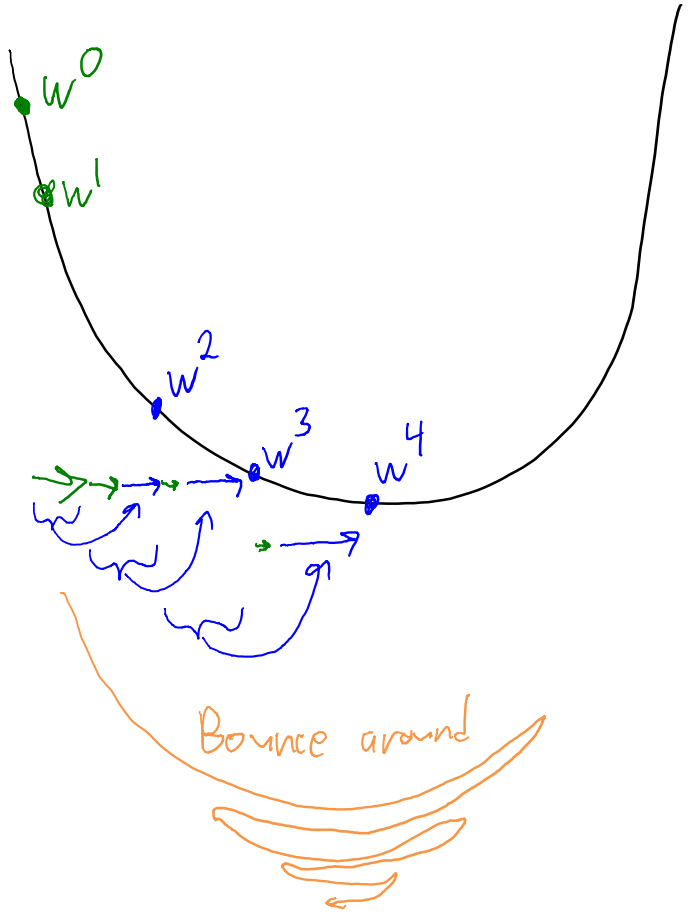
- Make sure to check your derivative code:
  - Numerical approximation to partial derivative: $\nabla_i f(x) \approx \frac{f(x + \delta e_i) - f(x)}{\delta}$
  - Numerical approximation to direction derivative: $\nabla f(x)^T d \approx \frac{f(x + \delta d) - f(x)}{\delta}$

Nesterov's Method

Gradient method



Nesterov/momentum/heavy-ball/conjugate gradient



Nesterov's Method

- Nesterov's accelerated gradient method:

$$x^{t+1} = y^t - \alpha_t \nabla f(y^t)$$
$$y^{t+1} = x^t + \beta_t (x^{t+1} - x^t)$$

$\beta_t = \frac{t-1}{t+2}$ $\quad$ $\alpha_t = \frac{1}{L}$

If 'f' is Convex and ∇f is L-Lipschitz, improves from $O(1/\varepsilon)$ to $O(1/\sqrt{\varepsilon})$ (optimal)

If 'f' is strongly-convex and ∇f is L-Lipschitz, improves from $O(\frac{L}{\mu}\log(\frac{1}{\varepsilon}))$ to $O(\sqrt{\frac{L}{\mu}}\log(\frac{1}{\varepsilon}))$ (close to optimal)

$\beta_t = \frac{1-\sqrt{\frac{\mu}{L}}}{1+\sqrt{\frac{\mu}{L}}}$

- Similar to heavy-ball/momentum method:

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

  – Conjugate gradient: optimal of α and β for strictly-convex quadratics.

Newton's Method

- Can be motivated as a quadratic approximation:

$$f(y) = f(x^t) + \nabla f(x^t)^T(y - x^t) + \frac{1}{2}(y - x^t)\nabla^2 f(z)(y - x^t) \quad \text{for some } z \text{ between } y \text{ and } x^t.$$

$$\approx f(x^t) + \nabla f(x^t)^T(y - x^t) + \frac{1}{2\alpha}(y - x^t)\nabla^2 f(x^t)(y - x^t) \quad (\text{assuming } \nabla^2 f(x^t) \succ 0)$$

- Newton's method is a second-order strategy (uses 2nd derivatives):

$$x^{t+1} = x^t - \alpha_t d^t \quad \text{where } d_t \text{ is the solution of } \nabla^2 f(x^t) d^t = \nabla f(x^t)$$

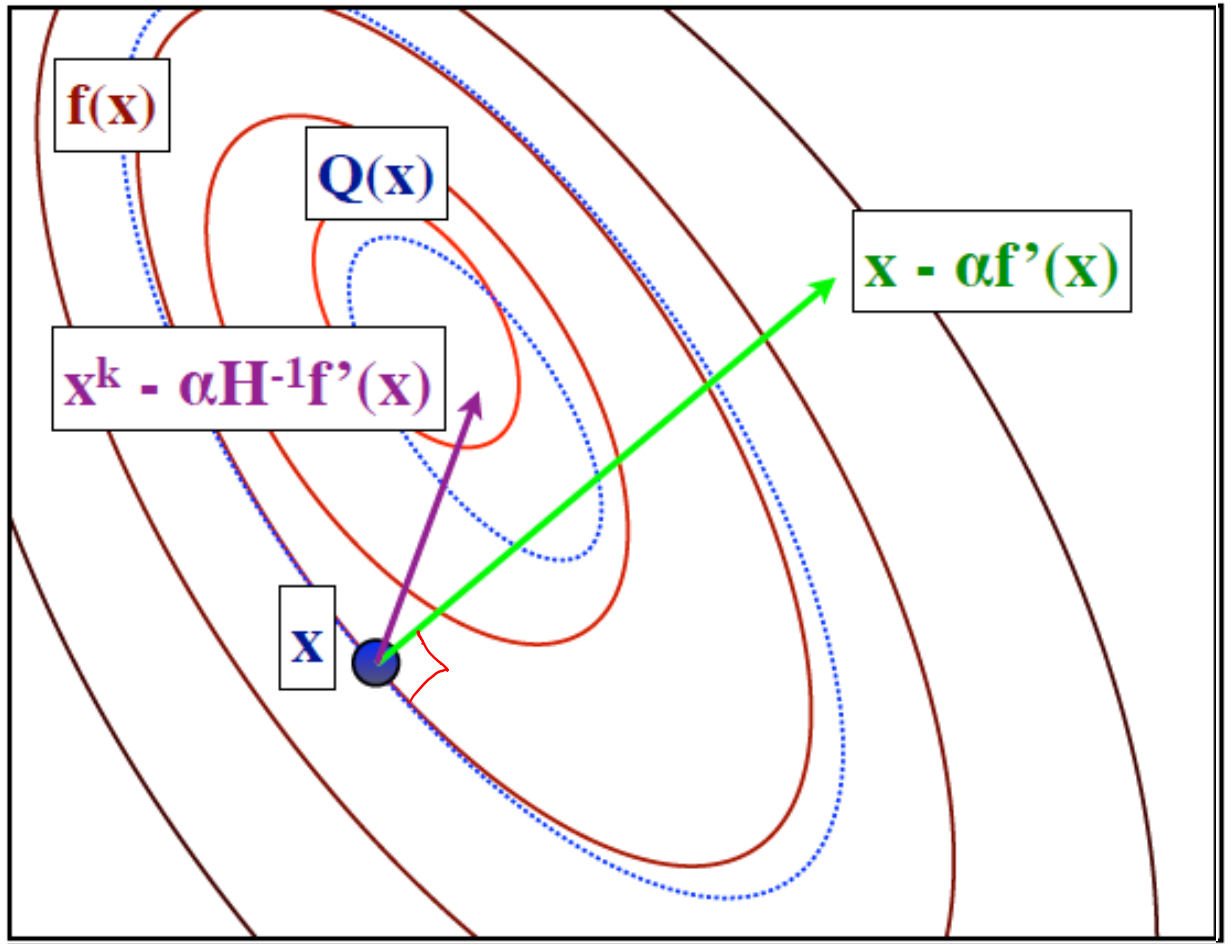
  – In stats, Newton's method applied to functions of form f(Ax) called "IRLS".

- Generalization of Armijo rule:

$$f(x^{t+1}) \leq f(x^t) - \alpha_t \gamma \nabla f(x^t)^T d^t$$

- Step-size α_t goes to 1 as we approach minimizer.

Newton's Method



Convergence Rate of Newton's Method

If $\nabla^2 f(x)$ is Lipschitz-continuous and $\nabla^2 f(x^*) \succeq \mu I$ then for 't' large enough:

$$f(x^{t+1}) - f(x^*) \leq \underbrace{\rho_t}_{\text{progress changes at each iteration 't'}} [f(x^t) - f(x^*)] \text{ with } \lim_{t \to \infty} \rho_t = 0.$$

- Local superlinear convergence: very fast, use it if you can.
- "Cubic regularization" of Newton's method gives global rates.
- But Newton's method is expensive if dimension 'd' is large:

Requires solution of $\underbrace{\nabla^2 f(x^t)}_{\text{'d' by 'd'}} d^t = \nabla f(x^t)$

Practical Approximations to Newton's Method

- Practical Newton-like methods:
  - Diagonal approximation: Approximate $\nabla^2 f(x^t)$ by diagonal H^t with elements $\nabla^2_{ii} f(x^t)$
  - Limited-memory quasi-Newton: Diagonal plus low-rank Hessian approximation, chosen to satisfy "quasi-Newton" equations.
    (L-BFGS)
  - Barzilai-Borwein approximation: Approximate $\nabla^2 f(x^t)$ by identity matrix I, choose initial step-size α_t as least squares solution to quasi-Newton equations.
  - Hessian-free Newton: Apply gradient or conjugate gradient to <u>approximately minimize quadratic approximation</u>.

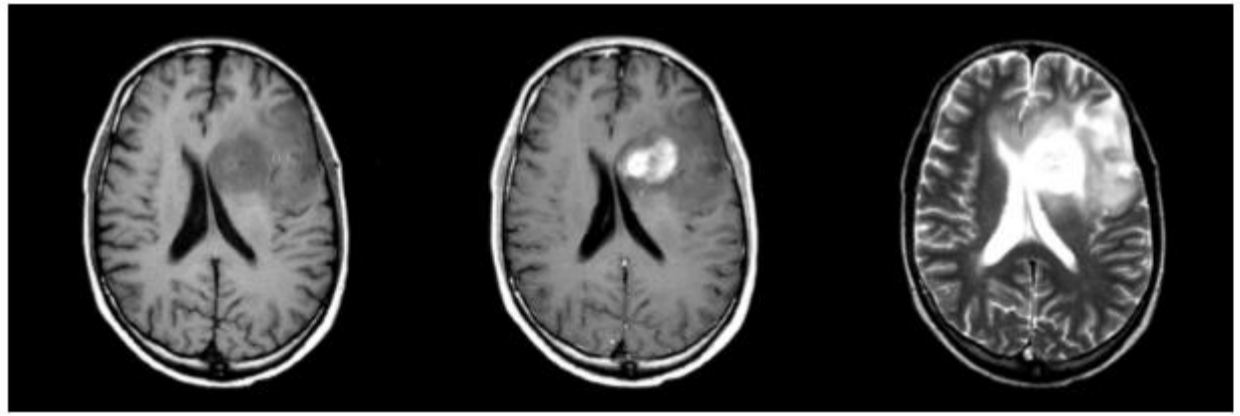
Gradient requires $\nabla f(x^t)v$ but this can be cheaply approximated: $\nabla^2 f(x)d = \lim_{\delta \to 0} \frac{\nabla f(x+\delta d) - \nabla f(x)}{\delta}$

  - Non-linear conjugate gradient.

(pause and take attendance)

Motivation: Automatic Brain Tumor Segmentation

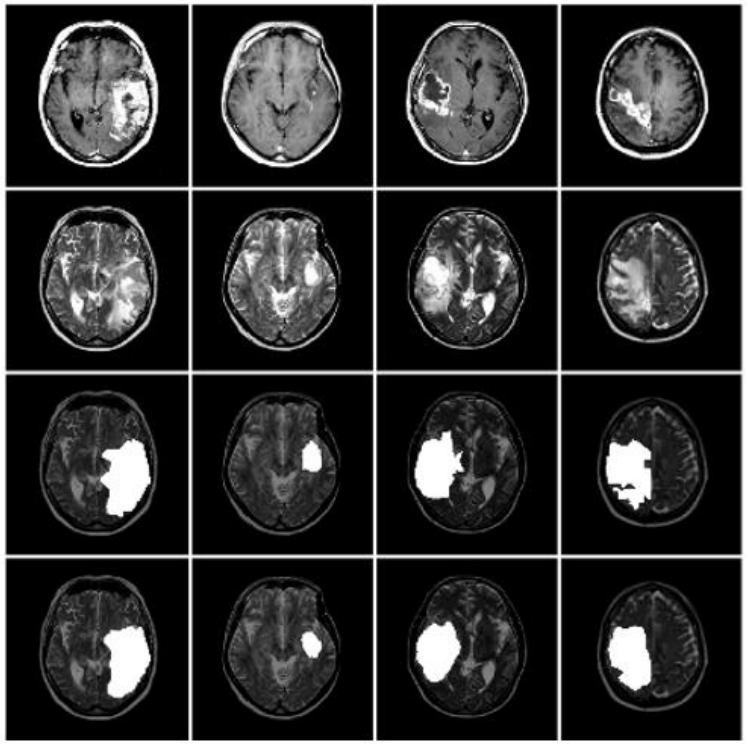
- Task: identifying tumours in multi-modal MRI data.

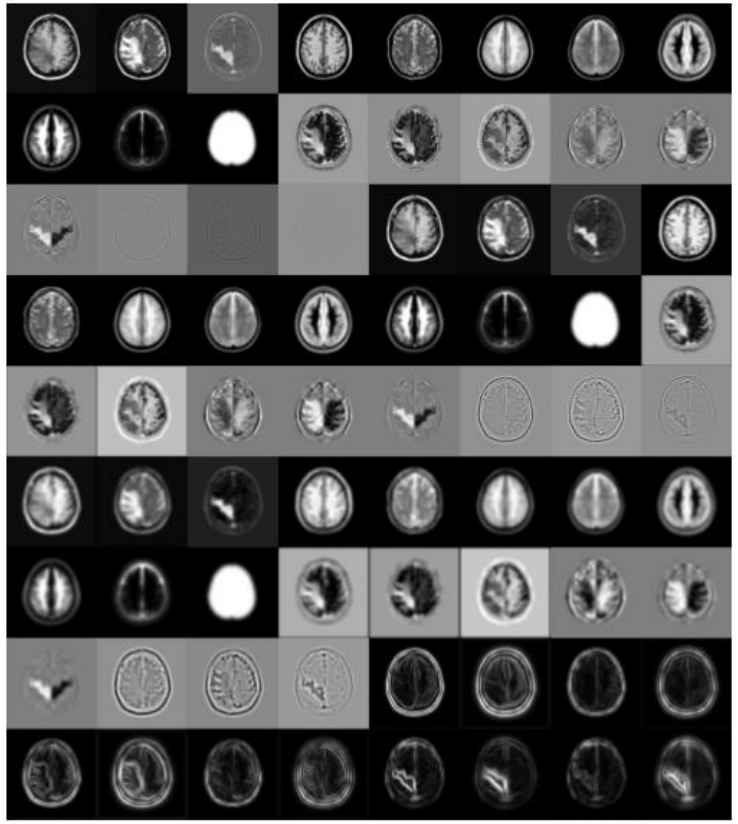


- Applications:
  - image-guided surgery.
  - radiation target planning.
  - quantifying treatment response
  - discovering growth patterns.

Motivation: Automatic Brain Tumor Segmentation

- Formulate as supervised learning:
  - Pixel-level classifier that predicts “tumour” or “non-tumour”.
  - Features: convolutions, expected values (in aligned template), and symmetry (all at multiple scales).



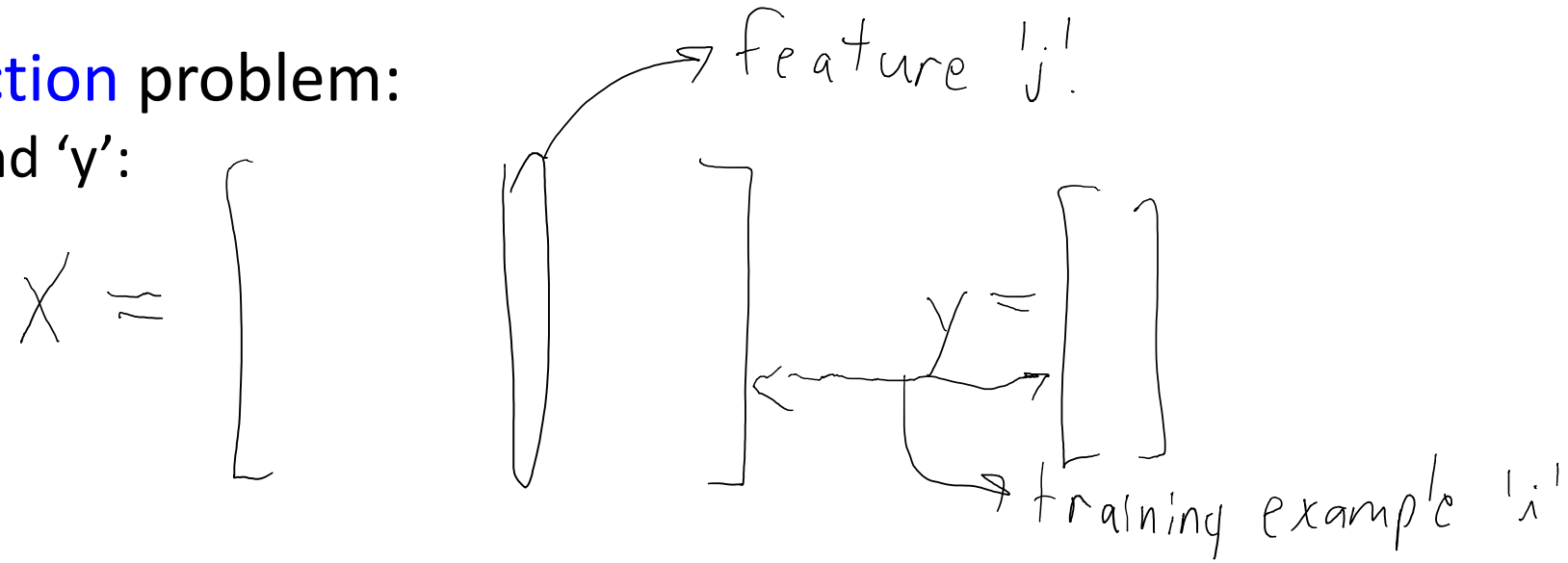


Motivation: Automatic Brain Tumor Segmentation

- Logistic regression was the most effective, *with the right features*.
- But if you used all features, it overfit.
  - We needed feature selection.
- Classical approach:
  - Define some 'score': AIC, BIC, cross-validation error, etc.
  - Search for features that optimize score:
    - Usually NP-hard, so we use greedy:
      - Forward selection, backward selection, stagewise,…
    - In this application, these are too slow.

Feature Selection

- General feature selection problem:
  - Given our usual 'X' and 'y':



  - We think some features/columns of 'X' are irrelevant for predicting 'y'.
- We want to fit a model that uses the 'best' set of features.
  - Special case: choosing 'best' basis from a set of possible bases.
- One of most important problems in ML/statistics, but very very messy.
  - Can be difficult to define what 'relevant' means.
  - For now, a feature is 'relevant' if it helps predict y_i from x_i.

L1-Regularization

- Popular approach to feature selection is L1-regularization:

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{2}\|Xw - y\|^2 + \lambda \|w\|_1$$

$$\|w\|_1 = \sum_{j=1}^{d} |w_j|$$

- Written above for squared loss, but can be used for any loss.
- Advantages:
  - Fast: can apply to large datasets, just minimizing convex function.
  - Reduces overfitting because it simultaneously regularizes.
- Disadvantage:
  - Prone to false positives, particularly if you pick λ by cross-validation.
  - Not unique: there may be infinite solutions.

L1-Regularization

- Key property of L1-regularization: if λ is large, solution w^* is sparse:
  - w^* has many values that are exactly zero.
- What this has to do with feature selection:

$$\hat{y}_i = w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + w_4 x_{i4} + w_5 x_{i5}$$

- If w = [0 0 3 0 -2], then:

$$\hat{y}_i = 0x_{i1} + 0x_{i2} + 3x_{i3} + 0x_{i4} + (-2)x_{i5}$$
$$= 3x_{i3} - 2x_{i5}$$ (features {1,2,4} are ignored)

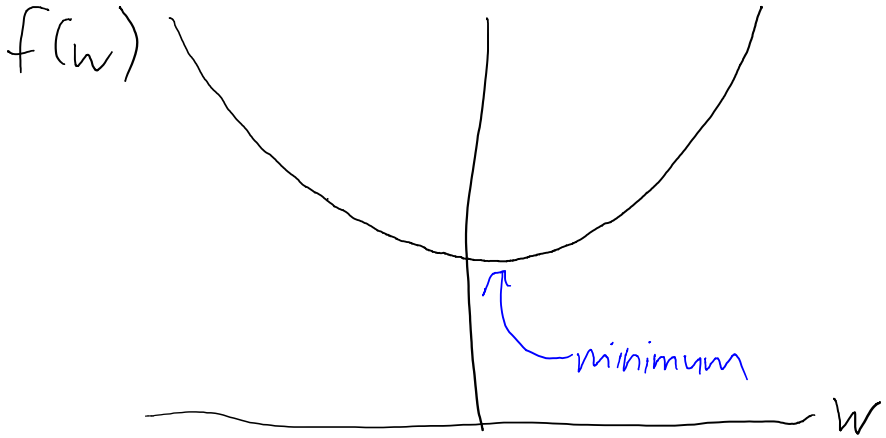
- Why does L1-regularization give sparsity but not L2-regularization?

Sparsity and Least Squares

- Consider 1D least squares objective:

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(y_i - wx_i)^2$$

- This is a convex 1D quadratic function of 'w' (i.e., a parabola):



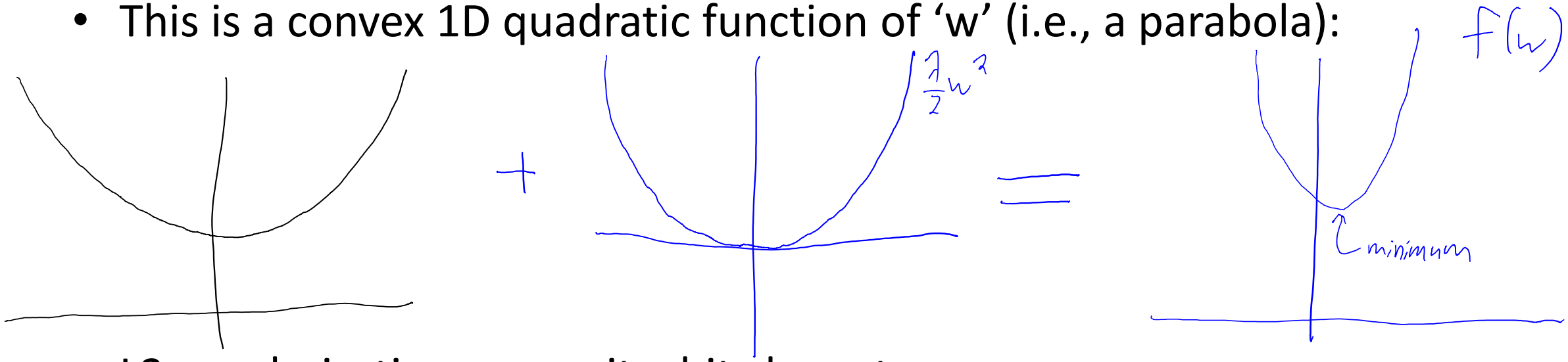
- This variable does not look relevant (minimum is close to 0).
  - If it's really irrelevant, minimum will move to 0 as 'n' goes to infinity.
  - But for finite 'n', minimum of parabola is unlikely to be exactly zero.

Sparsity and L2-Regularization

- Consider 1D L2-regularized least squares objective:

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(y_i - wx_i)^2 + \frac{\lambda}{2}w^2$$

- This is a convex 1D quadratic function of 'w' (i.e., a parabola):



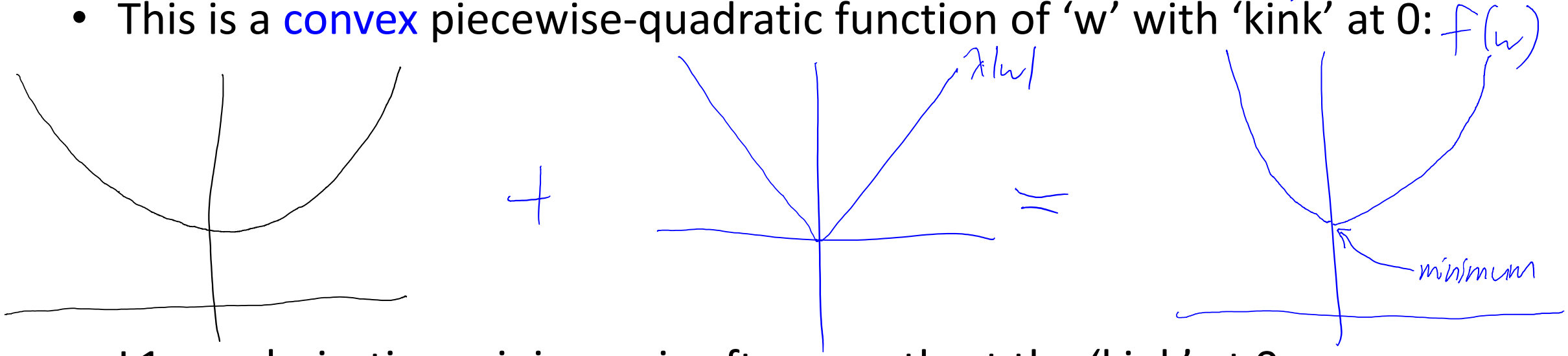
- L2-regularization moves it a bit closer to zero.
  - But there is nothing special about being 'exactly' zero.
  - Unless cost is flat at zero, L2-regularization always sets 'w_j' non-zero.

Sparsity and L1-Regularization

- Consider 1D L1-regularized least squares objective:

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(y_i - wx_i)^2 + \lambda|w| = \begin{cases} \frac{1}{2}\sum_{i=1}^{n}(y_i - wx_i)^2 + \lambda w, & w \geq 0 \\ \frac{1}{2}\sum_{i=1}^{n}(y_i - wx_i)^2 - \lambda w, & w < 0 \end{cases}$$

- This is a convex piecewise-quadratic function of 'w' with 'kink' at 0:

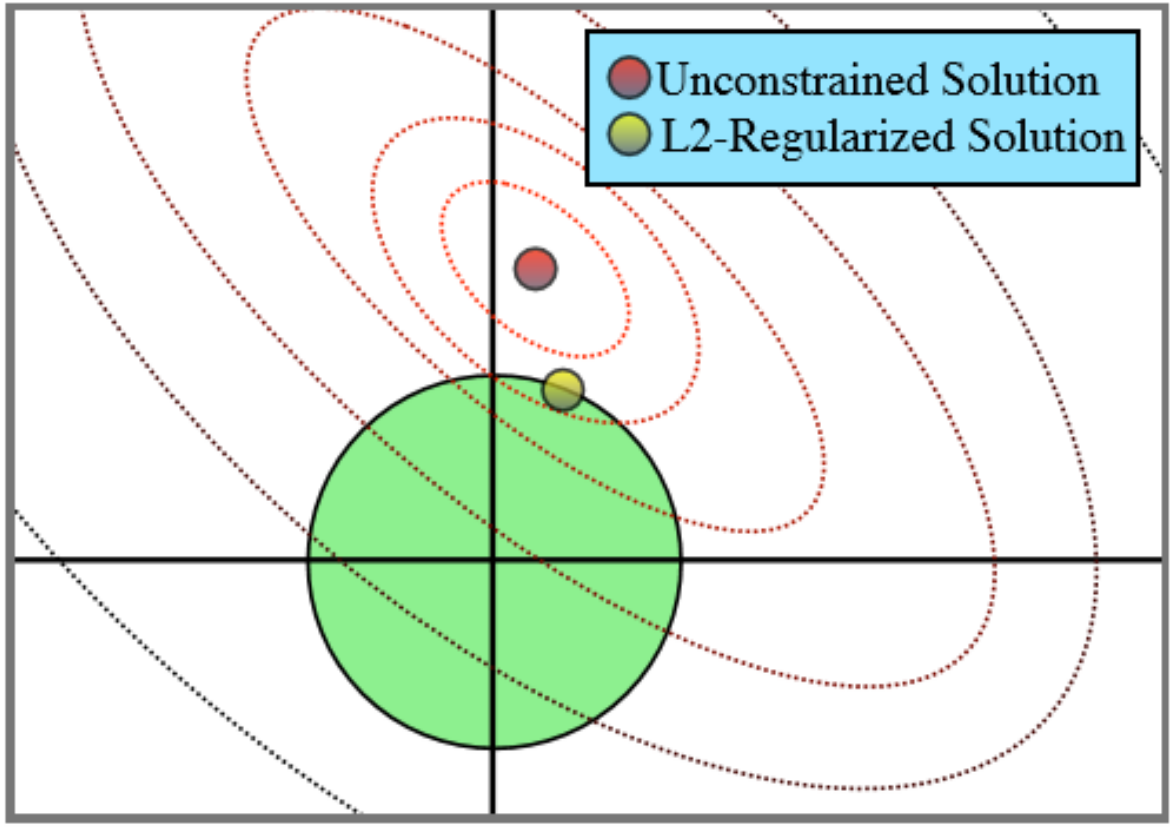


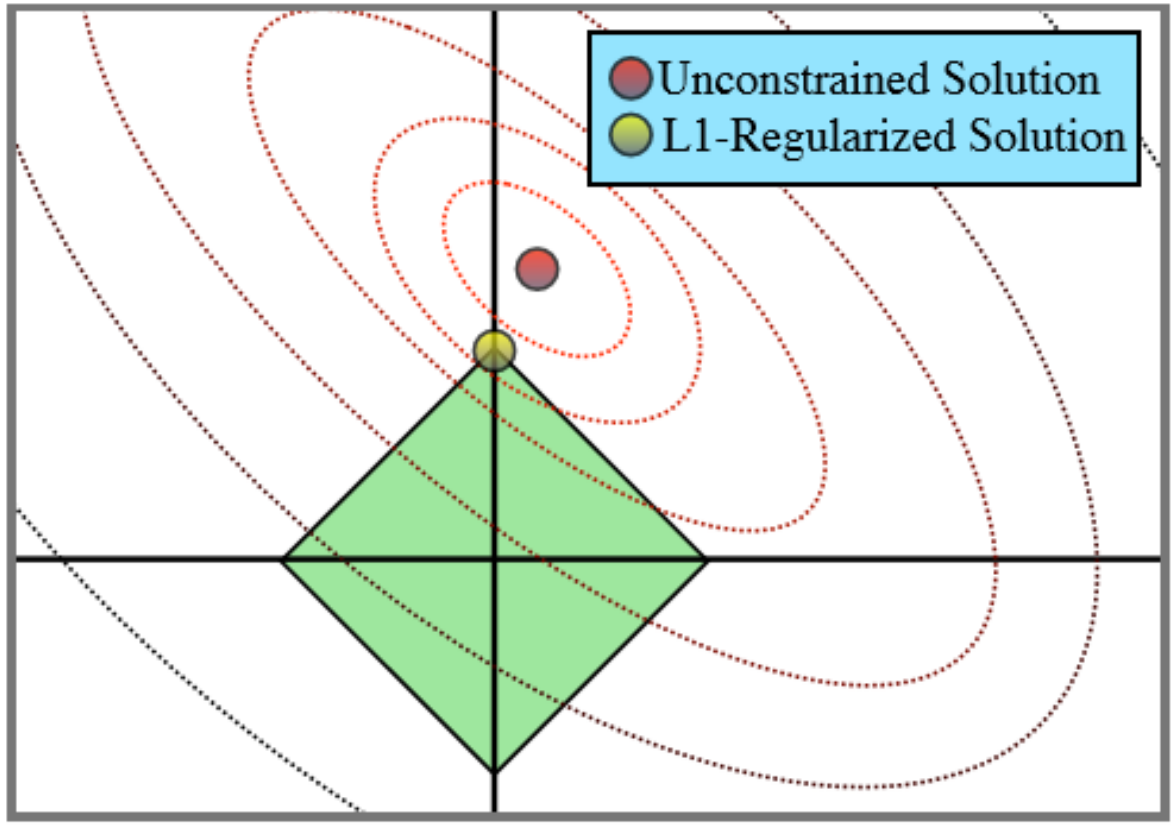
- L1-regularization minimum is often exactly at the 'kink' at 0:
  - It sets the feature to exactly 0, removing it from the model.
  - Big λ means kink is 'steep'. Small λ makes 0 unlikely to be minimum.

Where does sparsity come from?

- Another view on sparsity of L2- vs. L1-regularization:

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{2}\|Xw - y\|^2 + \lambda \|w\|_p \iff \underset{w \in \mathbb{R}^d, r \in \mathbb{R}}{\operatorname{argmin}} \frac{1}{2}\|Xw - y\|^2 + \lambda r \text{ subject to } r \geq \|w\|_p$$





L1-Regularization: Discussion

- "Sample complexity" [Ng, 2004]:
  - L2-regularization: you can learn with linear number of irrelevant features.
  - L1-regularization: you can learn with exponential number of irrelevant.
- "Elastic net":
  - Use both L2-regularization and L1-regularization.
  - Makes problem strongly-convex, so it has a unique solution.
- "Bolasso":
  - Run L1-regularization on boostrap samples.
  - Take features that are non-zero in all samples: fewer false positives.
- Non-convex regularizers:
  - Less sensitive to false positives, but solving optimization is NP-hard.

for example, $\sum_{j=1}^{d} \sqrt{w_j}$ will give higher-sparsity.

Solving L1-Regularization Problems

- How can we minimize non-smooth L1-regularized objectives?

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \ \frac{1}{2}\|Xw - y\|^2 + \lambda\|w\|_1$$

  - And let's assume X^TX is positive-definite, or we add L2-regularization.
    - Either conditions makes it strongly-convex.
- Use our trick to formulate as a quadratic program?
  - $O(d^2)$ or worse.
- Formulate as non-smooth convex optimization?
  - Sub-linear $O(1/\varepsilon)$ convergence rate.
- Make a smooth approximation to L1-norm?
  - Destroys sparsity.

Solving L1-Regularization Problems

- Key insight: this is not a general non-smooth convex function.
  - We can use structure to get large-scale O(log(1/ε)) methods.
- We can write it as:

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \; g(x) + h(x) \quad \text{where 'g' is smooth and 'h' is "simple"}$$

  - This lets us apply proximal-gradient methods (next time).
- We can also write it as:

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \; g(x) + \underbrace{\sum_{j=1}^{d} h_j(x_j)}_{\text{"separable"}} \quad \text{where 'g' is smooth.}$$

  - This lets us apply coordinate optimization methods.

Coordinate Optimization

- We want to optimize a differentiable function:

$$\underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \; f(x)$$

- Coordinate optimization:
  - At each iteration 't', we update one variable 'j_t':

"just change variable j_t"

$$x^{t+1} = x^t + \alpha_t e_{j_t} \quad \text{where} \quad e_j = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \leftarrow \text{position 'j'}$$

- How do we pick the variable 'j_t' to update?
  - Classic choices: cyclic, random, and greedy.
- How do we update the variable we chose?
  - Classic choices: constant step-size, line-search, exact optimization.

Coordinate Optimization

- This is an obvious, old, and widely-used algorithm.
- But until ~2010, we had no theory about when to use it.
  - For some applications it works great, for some applications it's terrible.
- Key insight in ~2010:
  - If you can do 'd' coordinate updates for the cost of one gradient update, then randomized coordinate optimization is faster than gradient descent.
  - Applies to random or greedy selection and 1/L or exact updates.
- When is this true?

Problems Suitable for Coordinate Descent

- Coordinate update is n times faster than gradient update for:

$$\underset{x \in \mathbb{R}^d}{\operatorname{argmin}} \; f(x) = g(Ax)$$

  - Where 'g' is smooth/cheap but bottleneck is multiplication by 'A'.
  - For example, least squares and logistic regression.
  - Key idea: can track the product Ax^t after single-coordinate updates,

$$Ax^{t+1} = A(x^t + \alpha_t e_{j_t}) = \underbrace{Ax^t}_{\text{old value}} + \alpha_t \underbrace{Ae_{j_t}}$$

$O(n)$ because e_{j_t} has <u>one</u> non-zero.

  - And since 'g' is cheap you get gradient for random coordinate by:

$$\nabla f(x) = A^T \nabla g(Aw) \qquad \nabla_j f(x) = \nabla g(Aw)^T a_j$$

column of A: cost is $O(n)$. → Compare to <u>gradient</u> which costs $O(nd)$

- The other class where coordinate update is n times faster:

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{d} \sum_{j=1}^{d} g_{ij}(x_i, x_j)$$

(e.g., graph-based semi-supervised learning)

Analysis of Coordinate Optimization

- To analyze coordinate descent, assume each $\nabla_j f$ is L-Lipschitz:

$$|\nabla_j f(x+\alpha e_j) - \nabla_j f(x)| \leq L|\alpha| \quad \text{for all } x \text{ and } \alpha$$

- For twice-differentiable 'f', equivalent to $\nabla^2_{ii} f(x) \leq L$ for all 'x'.
- Assume 'f' is μ-strongly-convex.
- Assume random coordinate selection and exact coordinate update.
- Then to find ε-optimal solution the number of iterations is:

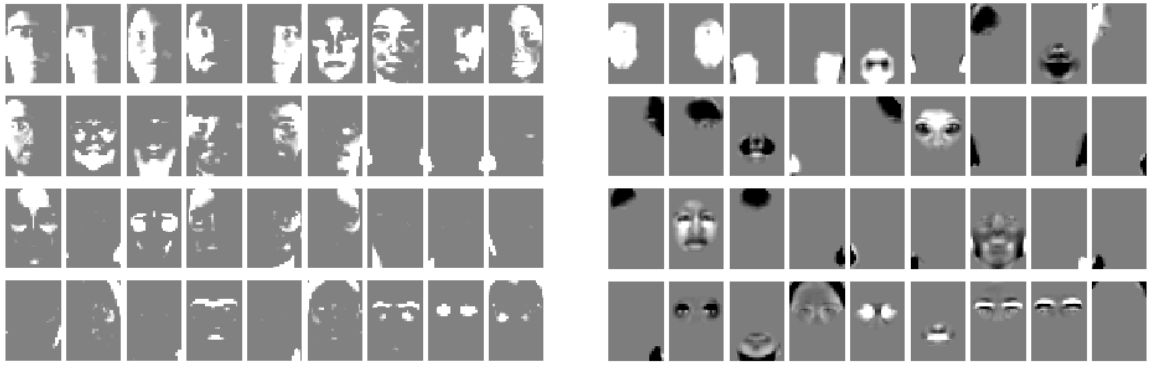
$$O\left(d\frac{L}{\mu}\log\left(\frac{1}{\varepsilon}\right)\right) \text{ (random coordinate descent)} \qquad O\left(\frac{L_f}{\mu}\log\left(\frac{1}{\varepsilon}\right)\right) \text{ (gradient descent)}$$

- 'L' is coordinate-wise and 'L_f' is for full-gradient: $L \leq L_f \leq dL$.
  - Because $L_f \leq dL$, we need fewer gradient descent iterations.
  - Because $L \leq L_f$, we need fewer 'cycles of d' coordinate descent iterations.

Summary

- Weaker assumptions for gradient descent:
  - L-Lipschitz gradient, weakening convexity, practical step sizes.
- Optimization zoo for minimizing continuous functions.
- Faster first-order methods like Nesterov's and Newton's method.
- Feature selection: choosing set of relevant variables.
- L1-regularization: feature selection as convex optimization.
- Coordinate optimization: when updating single variable is fast.

- Next time: multi-task learning and "structured" sparsity.



http://jmlr.org/proceedings/papers/v9/jenatton10a/jenatton10a.pdf