First-Order Optimization Algorithms for Machine Learning

Subgradient Method

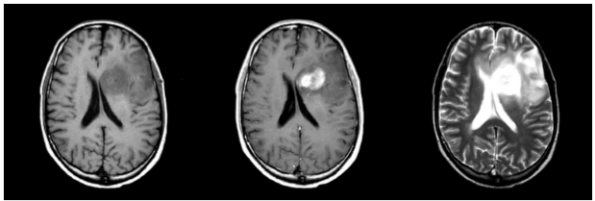
Mark Schmidt

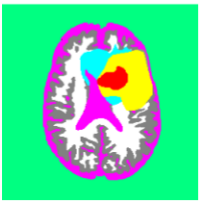
University of British Columbia

Summer 2020

Motivation: Automatic Brain Tumour Segmentation

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

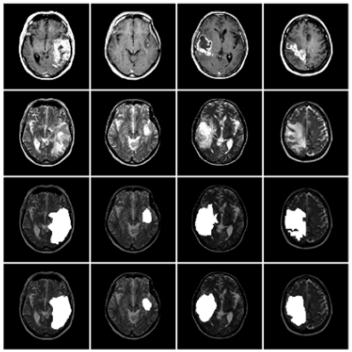


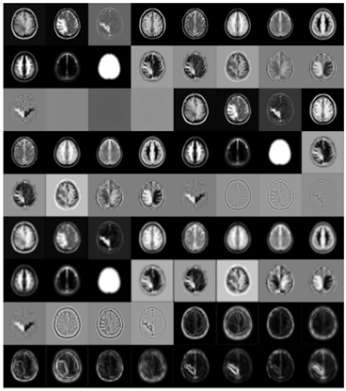


- Applications:
  - Image-guided surgery.
  - Radiation target planning.
  - Quantifying treatment response.
  - Discovering growth patterns.

Motivation: Automatic Brain Tumour 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 Tumour Segmentation

- Logistic regression was among most effective models, 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,...
  - In brain tumour application, even greedy methods were too slow.
    - Just one image gives 8 million training examples.

Feature Selection

- General feature selection problem:
  - Given our usual X and y, we'll use x_j to represent column j:

$$X = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \dots & x_d \\ | & | & & | \end{bmatrix}, \quad y = \begin{bmatrix} | \\ y \\ | \end{bmatrix}.$$

  - We think some features/columns x_j are irrelevant for predicting y.
- We want to fit a model that uses the "best" set of features.

- One of most important problems in ML/statistics, but very very messy.
  - In 340 we saw how difficult it is to define what "relevant" means.

L1-Regularization

- A popular appraoch to feature selection we saw in 340 is L1-regularization:

$$F(w) = f(w) + \lambda \|w\|_1.$$

- Advantages:
  - Fast: can apply to large datasets, just minimizing one function.
    - Convex if f is convex.
  - Reduces overfitting because it simultaneously regularizes.
- Disadvantages:
  - Prone to false positives, particularly if you pick λ by cross-validation.
  - Not unique: there may be infinite solutions.

- There exist many extensions:
  - "Elastic net" adds L2-regularization to make solution unique.
  - "Bolasso" applies this on bootstrap samples to reduce false positives.
  - Non-convex regularizers reduce false positives but are NP-hard.

L1-Regularization

- Key property of L1-regularization: if λ is large, solution w^* is sparse:
  - w^* has many values that are exactly zero.

- How setting variables to exactly 0 performs feature selection in linear models:

$$\hat{y}^i = w_1 x_1^i + w_2 x_2^i + w_3 x_3^i + w_4 x_4^i + w_5 x_5^i.$$

- If $w = \begin{bmatrix} 0 & 0 & 3 & 0 & -2 \end{bmatrix}^\top$ then:

$$\begin{aligned} \hat{y}^i &= 0x_1^i + 0x_2^i + 3x_3^i + 0x_4^i + (-2)x_5^i \\ &= 3x_3^i - 2x_5^i. \end{aligned}$$

- Features $\{1, 2, 4\}$ are not used in making predictions: we "selected" $\{3, 5\}$.
  - To understand why variables are set to exactly 0, we need the notion of subgradient.

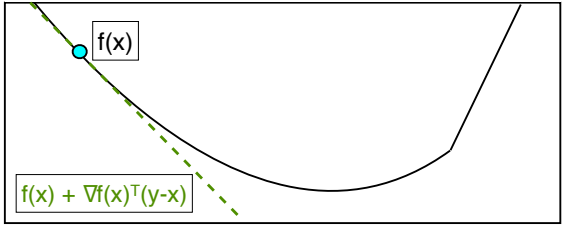
Sub-Gradients and Sub-Differentials

Differentiable convex functions are always above tangent,

$$f(v) \geq f(w) + \nabla f(w)^\top (v - w), \forall w, v.$$

A vector d is a subgradient of a convex function f at w if

$$f(v) \geq f(w) + d^\top (v - w), \forall v.$$



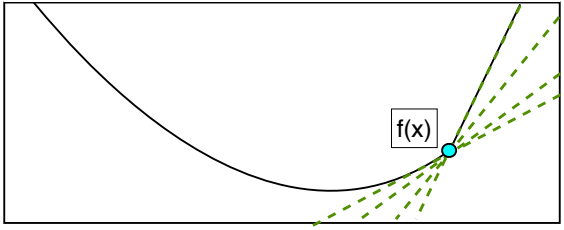
Sub-Gradients and Sub-Differentials

Differentiable convex functions are always above tangent,

$$f(v) \geq f(w) + \nabla f(w)^\top (v - w), \forall w, v.$$

A vector d is a subgradient of a convex function f at w if

$$f(v) \geq f(w) + d^\top (v - w), \forall v.$$



Sub-Gradients and Sub-Differentials Properties

- We can have a set of subgradients called the sub-differential, $\partial f(w)$.
  - Subdifferential is all the possible "tangent" lines.

- For convex functions:
  - Sub-differential is always non-empty (except some weird degenerate cases).
    - Formally, sub-differential guaranteed non-empty on "relative interior".
  - At differentiable w, the only subgradient is the gradient: $\partial f(w) = \{\nabla f(w)\}$.
  - At non-differentiable w, there will be a convex set of subgradients.

  - We have $0 \in \partial f(w)$ iff w is a global minimum.
    - This generalizes the condition that $\nabla f(w) = 0$ for differentiable functions.

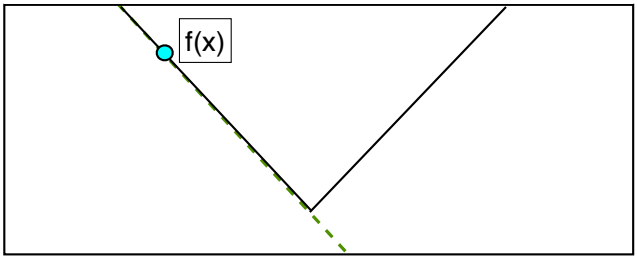
- For non-convex functions:
  - "Global" subgradients may not exist for every w.
  - Instead, we define subgradients "locally" around current w.
    - This is how you define "gradient" of ReLU function in neural networks.

Example: Sub-Differential of Absolute Function

- Sub-differential of absolute value function:

$$\partial |w| = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \\ [-1, 1] & w = 0 \end{cases}$$

  - "Sign of the variable if it's non-zero, anything in $[-1, 1]$ if it's zero."

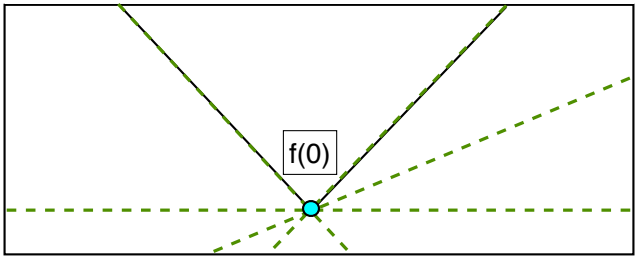


Example: Sub-Differential of Absolute Function

- Sub-differential of absolute value function:

$$\partial |w| = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \\ [-1, 1] & w = 0 \end{cases}$$

  - "Sign of the variable if it's non-zero, anything in $[-1, 1]$ if it's zero."



Sub-Differential of Common Operations

- Some convenient rules for calculating subgradients of convex functions:
  - Sub-differential of max is all convex combinations of argmax gradients:

$$\partial \max\{f_1(x), f_2(x)\} = \begin{cases} \nabla f_1(x) & f_1(x) > f_2(x) \\ \nabla f_2(x) & f_2(x) > f_1(x) \\ \underbrace{\theta \nabla f_1(x) + (1-\theta)\nabla f_2(x)}_{\text{for all } 0 \le \theta \le 1} & f_1(x) = f_2(x) \end{cases}$$

    - This rules gives sub-differential of absolute value, using that $|\alpha| = \max\{\alpha, -\alpha\}$.

  - Sub-differential of sum is all sum of subgradients of individual functions:

$$\partial(f_1(x) + f_2(x)) = d_1 + d_2 \quad \text{for any} \quad d_1 \in \partial f_1(x), d_2 \in \partial f_2(x).$$

  - Sub-differential of composition with affine function works like the chain rule:

$$\partial f_1(Aw) = A^\top \partial f_1(z), \quad \text{where} \quad z = Aw,$$

    and we also have $\partial \alpha f(w) = \alpha \partial f(w)$ for $\alpha > 0$ (non-negative scaling).

Why does L1-Regularization but not L2-Regularization give Sparsity?

- Consider L2-regularized least squares,

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

- Element j of the gradient at $w_j = 0$ is given by

$$\nabla_j f(w) = x_j^\top \underbrace{(Xw - y)}_{r} + \lambda 0.$$

- For $w_j = 0$ to be a solution, we need $0 = \nabla_j f(w^*)$ or that

$$0 = x_j^\top r^* \quad \text{where } r^* = Xw^* - y \text{ for the solution } w^*$$

that column j is orthogonal to the final residual.

  - This is possible, but it is very unlikely (probability 0 for random data).
  - Increasing λ doesn't help.

Why does L1-Regularization but not L2-Regularization give Sparsity?

- Consider L1-regularized least squares,

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|_1.$$

- Element j of the subdifferential at $w_j = 0$ is given by

$$\partial_j f(w) \equiv x_j^\top \underbrace{(Xw - y)}_{r} + \lambda \underbrace{[-1, 1]}_{\partial|w_j|}.$$

- For $w_j = 0$ to be a solution, we need $0 \in \partial_j f(w^*)$ or that

$$0 \in x_j^T r^* + \lambda[-1, 1] \quad \text{or equivalently}$$
$$-x_j^T r^* \in \lambda[-1, 1] \quad \text{or equivalently}$$
$$|x_j^\top r^*| \leq \lambda,$$

that column j is "close to" orthogonal to the final residual.

- So features j that have little to do with y will often lead to $w_j = 0$.
- Increasing λ makes this more likely to happen.

Outline

Solving L1-Regularization Problems

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

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

- Formulate as a quadratic program?
  - $O(d^2)$ or worse.
- Make a smooth approximation to the L1-norm?
  - Destroys sparsity (we'll again just have one subgradient at zero).

- Use a subgradient method?

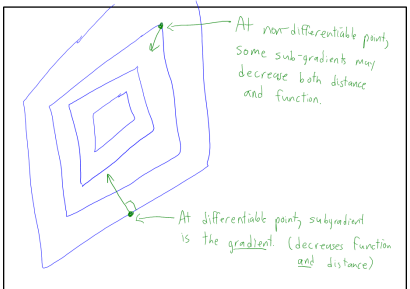
Subgradient Method

- The basic subgradient method:

$$w^{k+1} = w^k - \alpha_k g_k,$$

for any $g_k \in \partial f(w^k)$.

- This can increase the objective even for small α_k.
  - Though for convex f the distance to solutions decreases:
    - $\|w^{k+1} - w^*\| < \|w^k - w^*\|$ for small enough α_k.



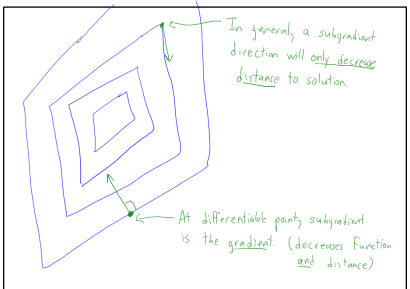
Subgradient Method

- The basic subgradient method:

$$w^{k+1} = w^k - \alpha_k g_k,$$

for any $g_k \in \partial f(w^k)$.

- This can increase the objective even for small α_k.
  - Though for convex f the distance to solutions decreases:
    - $\|w^{k+1} - w^*\| < \|w^k - w^*\|$ for small enough α_k.



Subgradient Method

- The basic subgradient method:

$$w^{k+1} = w^k - \alpha_k g_k,$$

for any $g_k \in \partial f(w^k)$.

- This can increase the objective even for small α_k.
  - Though for convex f the distance to solutions decreases:
    - $\|w^{k+1} - w^*\| < \|w^k - w^*\|$ for small enough α_k.

- The subgradients g_k don't necessarily converge to 0 as we approach a w^*.
  - If we are at a solution w^*, we might move away from it.
  - So as in stochastic gradient, we need decreasing step-sizes like

$$\alpha_k = O(1/k), \quad \text{or} \quad \alpha_k = O(1/\sqrt{k}),$$

in order to converge.
  - This destroys performance.

Convergence Rate of Subgradient Methods

- Subgradient methods are slower than gradient descent:

Assumption	Gradient	Subgradient	Quantity
Convex	$O(1/\epsilon)$	$O(1/\epsilon^2)$	$f(w^t) - f^* \leq \epsilon$
Strongly-Convex	$O(\log(1/\epsilon))$	$O(1/\epsilon)$	$f(w^t) - f^* \leq \epsilon$

- Other subgradient-based methods are not faster.
  - There are matching lower bounds in dimension-independent setting.
  - Includes cutting plane and bundle methods.
    - These tend to be faster in practice, though cost grows with iteration number.

- Also, acceleration doesn't improve subgradient rates.
  - We do NOT go from $O(1/\epsilon^2)$ to $O(1/\epsilon)$ by adding momentum.

- Smoothing f and applying gradient descent doesn't help.
  - May need to have $L = 1/\epsilon$ in a sufficiently-accurate smooth approximation.
  - However, if you smooth and accelerate you can close the gaps a bit (bonus).

Summary

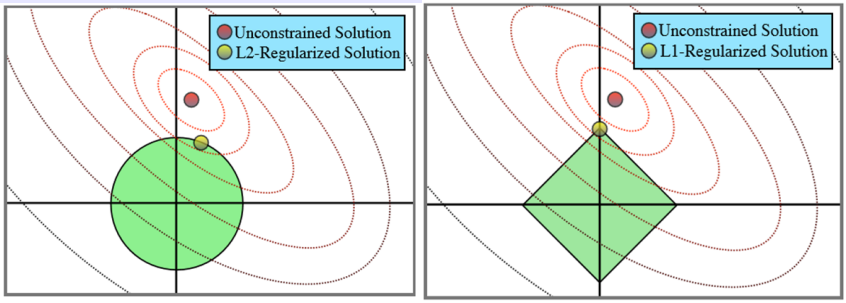
- L1-regularization: feature selection as convex optimization.
- Subgradients: generalize gradients for non-smooth convex functions.
- Subgradient method: optimal but very-slow general non-smooth method.

- Next time: solving problems with "simple" regularizers in $O(\log(1/\epsilon))$.

L1-Regularization vs. L2-Regularization

- Another view on sparsity of L2- vs. L1-regularization using our constraint trick:

$$\operatorname*{argmin}_{w \in \mathbb{R}^d} f(w) + \lambda \|w\|_p \quad \Leftrightarrow \quad \operatorname*{argmin}_{w \in \mathbb{R}^d, \tau \in \mathbb{R}} f(w) + \lambda\tau \text{ with } \tau \geq \|w\|_p.$$



- Notice that L2-regularization has a rotataional invariance.
  - This actually makes it more sensitive to irrelevant features.

Does Smoothing Help?

- Nesterov's smoothing paper gives a way to take a non-smooth convex f and number ϵ, then it constructs a new function f_ϵ such that

$$f(w) \leq f_\epsilon(w) \leq f(w) + \epsilon,$$

so that minimizing $f_\epsilon(w)$ gets us within ϵ of the optimal solution.
  - And further that $f_\epsilon(w)$ is differentiable with $L = O(1/\epsilon)$.

- If we apply gradient descent to the smooth function, we get

$$t = \underbrace{O(L/\epsilon)}_{\text{smoothed problem}} = \underbrace{O(1/\epsilon^2)}_{\text{original problem}},$$

for convex functions (same speed as subgradient).
- For strongly-convex functions we get

$$t = O(L \log(1/\epsilon)) = O((1/\epsilon)\log(1/\epsilon)),$$

which is actually worse than the best subgradient methods by a log factor.

Does Smoothing Help?

- Nesterov's smoothing paper gives a way to take a non-smooth convex f and number ϵ, then it constructs a new function f_ϵ such that

$$f(w) \leq f_\epsilon(w) \leq f(w) + \epsilon,$$

so that minimizing $f_\epsilon(w)$ gets us within ϵ of the optimal solution.
  - And further that $f_\epsilon(w)$ is differentiable with $L = O(1/\epsilon)$.

- If we apply accelerated gradient descent to the smooth function, we get

$$t = O(\sqrt{L/\epsilon}) = O(1/\epsilon),$$

which is faster than subgradient methods.
(same speed as unaccelerated gradient descent)
- For strongly-convex functions the accelerated method gets

$$t = O(\sqrt{L}\log(1/\epsilon)) = O((1/\sqrt{\epsilon})\log(1/\epsilon)),$$

which is faster than subgradient methods (but not linear converence).

What is the best subgradient?

- We considered the deterministic subgradient method,

$$x^{t+1} = x^t - \alpha_t g_t, \text{ where } g_t \in \partial f(x^t),$$

  under any choice of subgradient.

- But what is the "best" subgradient to use?
  - Convex functions have directional derivatives everywhere.
  - Direction $-g_t$ that minimizes directional derivative is minimum-norm subgradient,

$$g^t = \underset{g \in \partial f(x^t)}{\text{argmin}} \; ||g||$$

  - This is the steepest descent direction for non-smooth convex optimization problems.

  - You can compute this for L1-regularization, but not many other problems.
  - Used in best deterministic L1-regularization methods, combined with Newton.