ART.T458 Advanced Machine learning
6 Advances in gradient based methods

Masamichi Shimosaka

Associate Professor
Dept. of Computer Science

This course

Foundations of machine learning (1st half)

- Interdisciplinary fields of
  - probability theory
  - statistics
  - optimization

Recent topic(s) of machine learning (2nd half)

- Deep learning!
- Deep learning for X!

Today's topic

Today's Model

- No new (prediction) models are introduced today!

Theoretical part: advances in gradient methods

- Accelerated proximal gradient methods
  - Convergence rate of proximal gradient methods
  - Acceleration, momentum, FISTA: Accelerated proximal gradient method
- Subgradient method
- Projected gradient method
- Stochastic optimization
  - From batch to stochastic way
  - vs. batch training in terms of convergence rate & computational cost
- Adaptive step size in gradient methods
  - AdaGrad, adam

Review: Proximal gradient model

Simple gradient based model for non-smooth function

- Loss function: smooth convex function
- Regularization: (non-)smooth convex function

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \left(\frac{1}{2} ||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2 + \lambda ||\boldsymbol{w}||_1 \right)$$

$\psi(\boldsymbol{w})$ $\Omega(\boldsymbol{w})$

- Minimization of upper bound of original problem sequentially

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \left(\psi(\boldsymbol{w}^{(0)}) + \nabla\psi(\boldsymbol{w}^{(0)})^\top \left(\boldsymbol{w} - \boldsymbol{w}^{(0)} \right) + \frac{\gamma}{2} ||\boldsymbol{w} - \boldsymbol{w}^{(0)}||_2^2 + \Omega(\boldsymbol{w}) \right)$$

Upper bound of $\psi(\boldsymbol{w})$ around w0 $\Omega(\boldsymbol{w})$

$\tilde{\psi}(\boldsymbol{w}|\boldsymbol{w}^{(0)})$

- The following update is iterated until its convergence

$$\boldsymbol{w}^{(t)} = \operatorname{prox}_{\eta_t \Omega} \left(\boldsymbol{w}^{(t-1)} - \eta_t \nabla\psi(\boldsymbol{w}^{(t-1)}) \right)$$

Gradient descent update

Proximal operation after gradient descent update

Review: Monotonic decrease property in proximal gradient methods

With proximal gradient method, the following inequality holds (a1:derive this inequality)**

$$\psi(\boldsymbol{w}^{(t+1)}) + \lambda||\boldsymbol{w}^{(t+1)}||_1 \leq \tilde{\psi}(\boldsymbol{w}^{(t+1)}|\boldsymbol{w}^{(t)}) \leq \tilde{\psi}(\boldsymbol{w}^{(t)}|\boldsymbol{w}^{(t)}) = \psi(\boldsymbol{w}^{(t)}) + \lambda||\boldsymbol{w}^{(t)}||_1$$

Objective function is monotonically decreased

- Since the function is non-negative (i.e. objective has lower bound), the parameter converges
- Actually, we can use this update for non γ-smooth function (similar to backtracking technique)

Review: Monotonic decrease property in proximal gradient methods

With proximal gradient method, the following inequality holds (a1:derive this inequality)**

$$\psi(\boldsymbol{w}^{(t+1)}) + \lambda||\boldsymbol{w}^{(t+1)}||_1 \leq \tilde{\psi}(\boldsymbol{w}^{(t+1)}|\boldsymbol{w}^{(t)}) \leq \tilde{\psi}(\boldsymbol{w}^{(t)}|\boldsymbol{w}^{(t)}) = \psi(\boldsymbol{w}^{(t)}) + \lambda||\boldsymbol{w}^{(t)}||_1$$

Objective function is monotonically decreased

- Since the function is non-negative (i.e. objective has lower bound), the parameter converges
- Actually, we can use this update for non γ-smooth function (similar to backtracking technique)

We need to consider the property of the algorithm in terms of **convergence rate**

Review: Monotonic decrease property in proximal gradient methods

With proximal gradient method, the following inequality holds (a1:derive this inequality)**

$$\psi(\boldsymbol{w}^{(t+1)}) + \lambda||\boldsymbol{w}^{(t+1)}||_1 \leq \tilde{\psi}(\boldsymbol{w}^{(t+1)}|\boldsymbol{w}^{(t)}) \leq \tilde{\psi}(\boldsymbol{w}^{(t)}|\boldsymbol{w}^{(t)}) = \psi(\boldsymbol{w}^{(t)}) + \lambda||\boldsymbol{w}^{(t)}||_1$$

Objective function is monotonically decreased

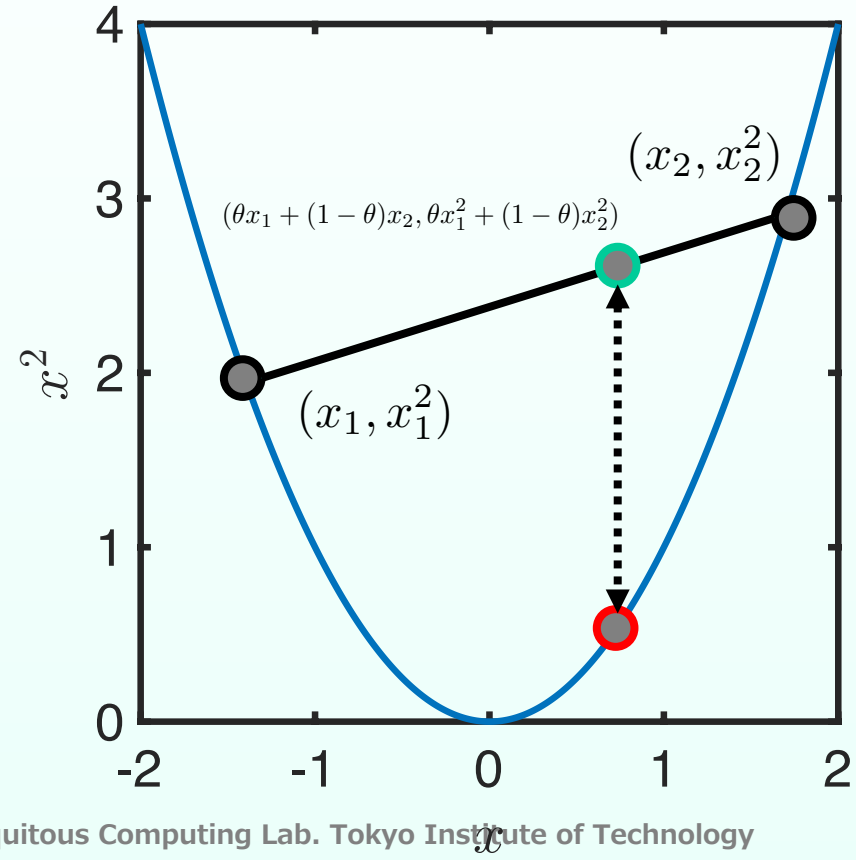
- Since the function is non-negative (i.e. objective has lower bound), the parameter converges
- Actually, we can use this update for non γ-smooth function (similar to backtracking technique)

- Convergence rate
  - Depends on types of loss function
  - Similar to smoothness, strength in convexity should be introduced.

α-strongly convex function

- Important property of loss function as well as γ-smooth function
  - c.f. Convex: $f(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \leq \theta f(\boldsymbol{x}) + (1-\theta) f(\boldsymbol{y}) \quad \forall \boldsymbol{x}, \boldsymbol{y}, \theta \in [0,1]$

- Definition : $\frac{\alpha}{2}\theta(1-\theta)||\boldsymbol{x}-\boldsymbol{y}||_2^2 + f\left(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}\right) \leq \theta f(\boldsymbol{x}) + (1-\theta) f(\boldsymbol{y})$
  - Curvature of f is not too close to zero



$\alpha > 0, \forall \boldsymbol{x}, \boldsymbol{y}, \theta \in (0,1)$

α-strongly convex function (cont'd)

- If function $f(\boldsymbol{x})$ is α-strongly convex function, $f(\boldsymbol{x}) - \frac{\alpha}{2}||\boldsymbol{x}||_2^2$ is convex

- If function $f(\boldsymbol{x})$ is α-strongly convex function, the following inequality holds, $\nabla^2 f(\boldsymbol{x}) \succeq \alpha \boldsymbol{I}$ (i.e. $\nabla^2 f(\boldsymbol{x}) - \alpha \boldsymbol{I} \succeq \boldsymbol{0}$)

- Confirm whether the following functions are strongly convex or not (**a2)
  - $x \in \mathbb{R}, \exp(x)$
  - $x \in \mathbb{R}, x^2$
  - $x \in \mathbb{R}, x^4$

Review: Convergence rate

Q-linear

- Parameter can be exponentially converged

$$\exists r \in (0,1), k' \in \mathbb{N}, \forall k \geq k', \frac{||\boldsymbol{w}^{(k+1)} - \hat{\boldsymbol{w}}||}{||\boldsymbol{w}^{(k)} - \hat{\boldsymbol{w}}||} \leq r$$

We use another metric here

- Estimating # of iteration when $J(\boldsymbol{w}^{(t)}) - J(\hat{\boldsymbol{w}}) \leq \epsilon$ holds

$$J(\boldsymbol{w}) = \psi(\boldsymbol{w}) + \Omega(\boldsymbol{w})$$

Loss function　Regularization

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, J(\boldsymbol{w})$$

- Ex. If $J(\boldsymbol{w}^{(t)}) - J(\hat{\boldsymbol{w}}) \leq \frac{2\gamma}{(t+1)^2}||\hat{\boldsymbol{w}} - \boldsymbol{w}^{(0)}||_2^2$,

  $J(\boldsymbol{w}^{(t)}) - J(\hat{\boldsymbol{w}}) \leq \epsilon$ reaches after $O(1/\sqrt{\epsilon})$ iterations

Convergence rate of (proximal) gradient method

Convergence rate does not depend on regularization

- This result can be applied to standard gradient descent

$\psi(\boldsymbol{w})$	Non α-strongly convex	α-strongly convex
Non γ-smooth	$O(1/\epsilon^2)$	$O(1/\epsilon)$
γ-smooth	$O(1/\epsilon)$	$O(-\ln \epsilon)$

Corresponds to Q-linear

Acceleration of (proximal) gradient

Original of proximal gradient

$$\boldsymbol{w}^{(t)} = \mathrm{prox}_{\eta_t \Omega}\left(\boldsymbol{w}^{(t-1)} - \eta_t \nabla \psi(\boldsymbol{w}^{(t-1)})\right)$$

Gradient descent update

Proximal operation after gradient descent update

Accelerated proximal gradient: use history of w

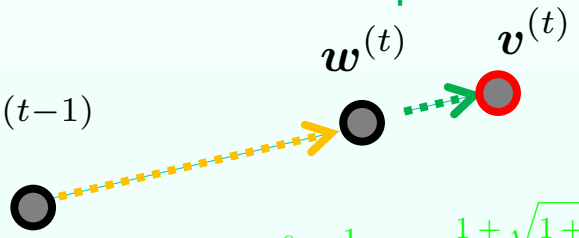
$$\boldsymbol{w}^{(t)} = \mathrm{prox}_{\eta_t \Omega}\left(\boldsymbol{v}^{(t-1)} - \eta_t \nabla \psi(\boldsymbol{v}^{(t-1)})\right)$$

Accelerated gradient descent update

Proximal operation after **accelerated** gradient descent update

$$\boldsymbol{v}^{(t)} = \boldsymbol{w}^{(t)} + q(t)(\boldsymbol{w}^{(t)} - \boldsymbol{w}^{(t-1)})$$

Some scalar function
(carefully designed)



$$q(t) = \frac{s_t - 1}{s_{t+1}},\ s_t = \frac{1 + \sqrt{1 + 4s_{t-1}^2}}{2}$$

$$q(t) = \frac{t-1}{t+2}$$

- C.f. Nesterov acceleration, momentum method

Convergence rate of accelerated (proximal) gradient method

Convergence rate does not depend on regularization

- This result can be applied to standard gradient descent

$\psi(\boldsymbol{w})$	Non α-strongly convex	α-strongly convex
Non γ-smooth	$O(1/\epsilon^2)$	$O(1/\epsilon)$
γ-smooth	$O(1/\epsilon)$ ⇓ $O(1/\sqrt{\epsilon})$	$O(-\ln \epsilon)$ ⇓ $O(-\ln \epsilon)$

Corresponds to Q-linear

with acceleration

Rate will be better with acceleration (even if it is in the same order)

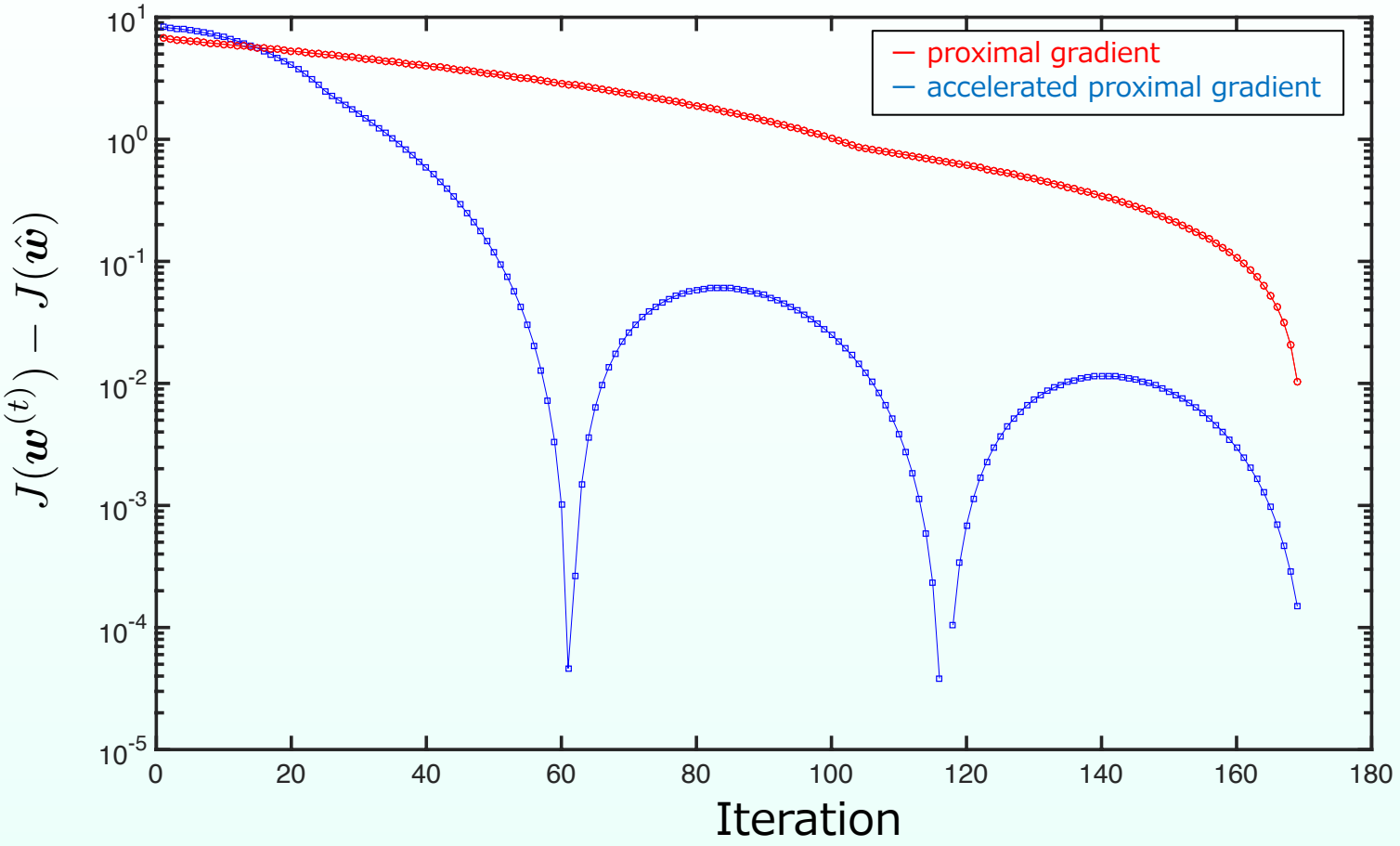
Acceleration has relation to momentum in NN community

FISTA: Fast ISTA: accelerated proximal gradient

FISTA in action with toy example

$$J(\boldsymbol{w}) = (\boldsymbol{w} - \boldsymbol{\mu})^\top \boldsymbol{A}(\boldsymbol{w} - \boldsymbol{\mu}) + \lambda ||\boldsymbol{w}||_1$$

$$\boldsymbol{A} = \begin{pmatrix} 300 & 0.5 \\ 0.5 & 1 \end{pmatrix}, \boldsymbol{\mu}^\top = \begin{pmatrix} 1 & 2 \end{pmatrix}, \lambda = 0.89$$

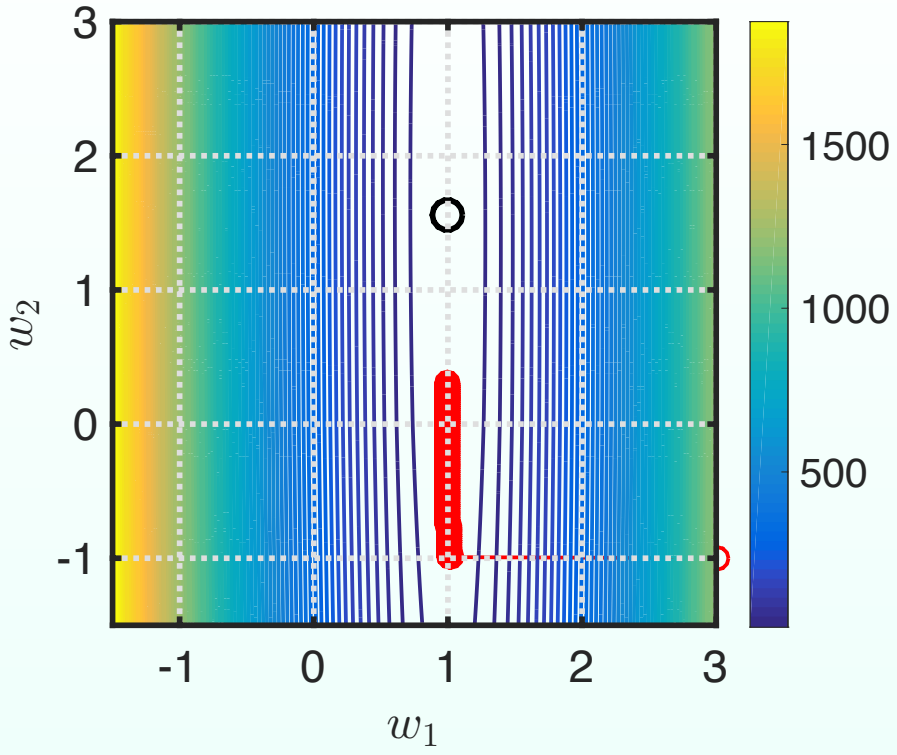


FISTA in action with toy example (cont'd)

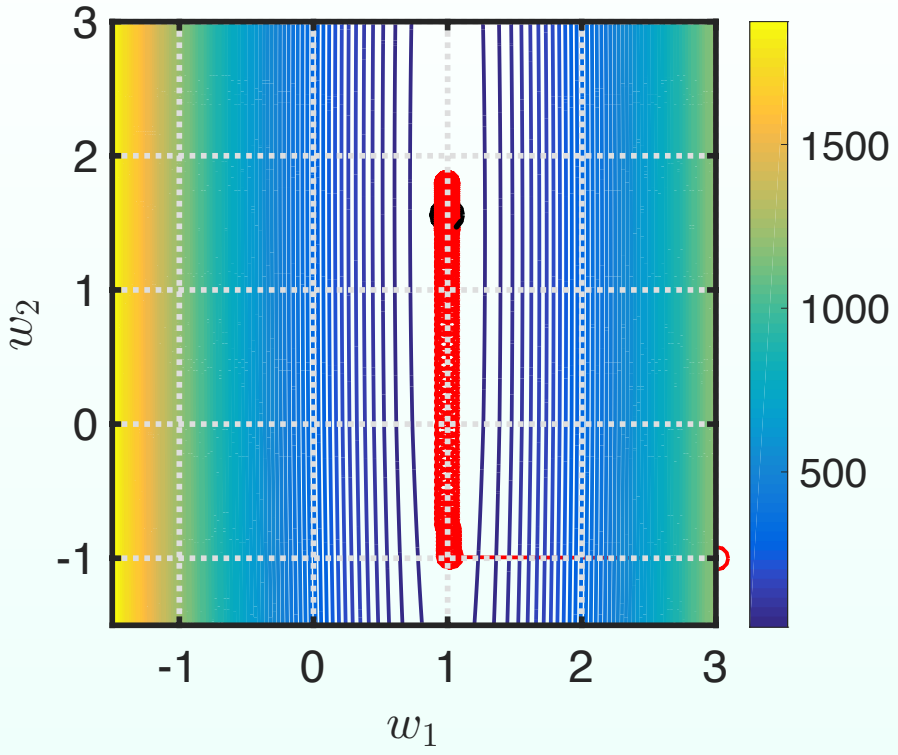
$$J(\boldsymbol{w}) = (\boldsymbol{w}-\boldsymbol{\mu})^\top \boldsymbol{A}(\boldsymbol{w}-\boldsymbol{\mu}) + \lambda||\boldsymbol{w}||_1$$

$$\boldsymbol{A} = \begin{pmatrix} 300 & 0.5 \\ 0.5 & 1 \end{pmatrix}, \boldsymbol{\mu}^\top = \begin{pmatrix} 1 & 2 \end{pmatrix}, \lambda = 0.89, \eta_t = 1/\gamma$$

Proximal gradient



Accelerated Proximal gradient



In some condition, accelerated proximal gradient gets faster convergence

Subgradient method

Subgradient

Generalized derivative for non-smooth function

- Ex. of non-smooth function
  - $f(\boldsymbol{w}) = ||\boldsymbol{w}||_1$ we could not obtain derivative at $\boldsymbol{w} = \boldsymbol{0}$
  - $f(w) = \max(0, 1 - w)$ we could not obtain derivative at $w = 1$

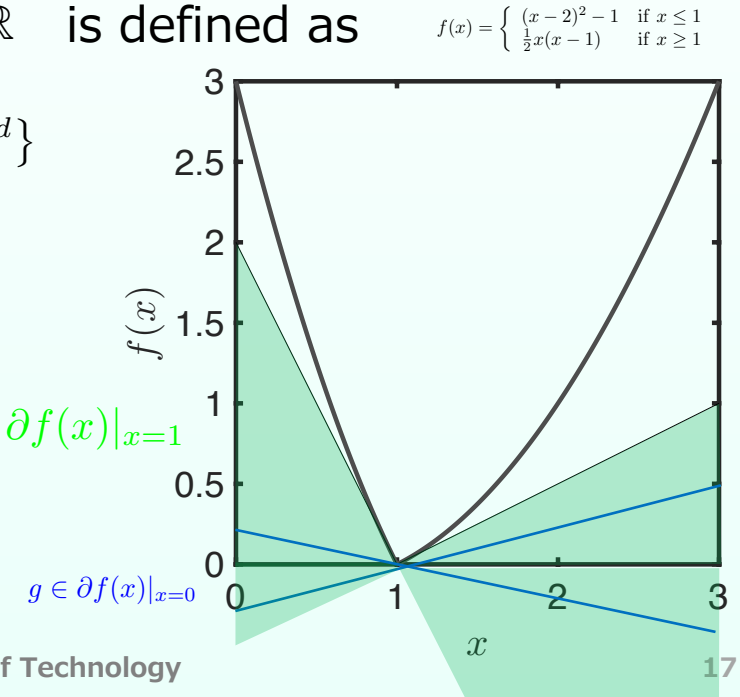
Definition of subderivative

- Subderivative of convex function $f : \mathbb{R}^d \to \mathbb{R}$ is defined as

$$\partial f(\boldsymbol{w}) = \{\boldsymbol{g} \in \mathbb{R}^d | (\tilde{\boldsymbol{w}} - \boldsymbol{w})^\top \boldsymbol{g} + f(\boldsymbol{w}) \leq f(\tilde{\boldsymbol{w}}),\ \forall \tilde{\boldsymbol{w}} \in \mathbb{R}^d\}$$

- Subgradient $\boldsymbol{g} \in \partial f(\boldsymbol{w})$

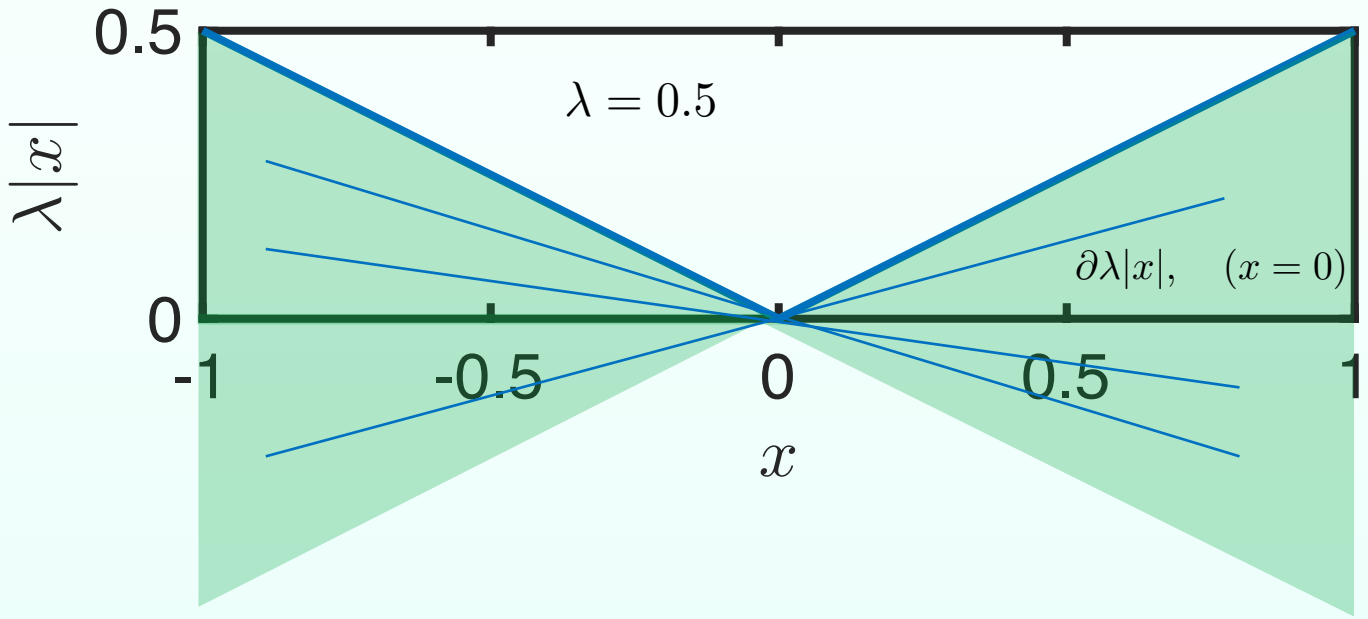
- In convex function, this definition is equiv. to 1st condition of convex function



Subgradient of L1 norm

L1 norm in 1d $f(x) = \lambda|x|, \lambda > 0$

$$g = \begin{cases} \lambda & \text{if } x > 0 \\ \eta\lambda & \text{if } x = 0,\ \eta \in [-1, 1] \\ -\lambda & \text{if } x < 0 \end{cases}$$



Necessary condition of local minima

Necessary condition of local minima with subgradient

- Even if the objective is non-smooth, we can reuse the necessary condition for smooth function
  - For continuous differentiable function $\frac{\partial \psi(\boldsymbol{w})}{\partial \boldsymbol{w}} + \frac{\partial \lambda \Omega(\boldsymbol{w})}{\partial \boldsymbol{w}} = \mathbf{0}$
  - For non-smooth function $\mathbf{0} \in \partial \psi(\boldsymbol{w}) + \lambda \partial \Omega(\boldsymbol{w})$
- Ex. Lasso

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \left(\frac{1}{2} ||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2 + \lambda ||\boldsymbol{w}||_1 \right)$$

- Optimality condition can be efficiently obtained without Lagrangian

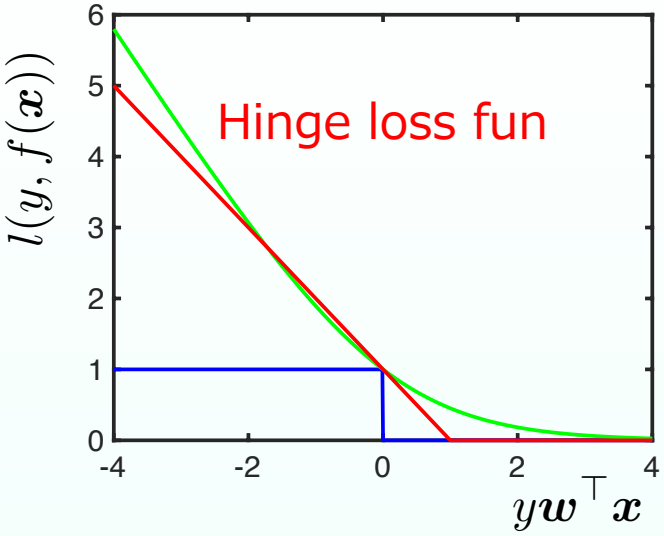
$$-\lambda \mathbf{1} \leq \boldsymbol{X}^\top (\boldsymbol{X}\hat{\boldsymbol{w}} - \boldsymbol{y}) \leq \lambda \mathbf{1}$$

Hinge-loss minimization with subgradient methods

Review: L2 regularized hinge-loss minimization (SVM)

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left(\sum_{i=1}^{n} \max(0, 1 - y_i \boldsymbol{w}^\top \boldsymbol{x}_i) + \lambda \boldsymbol{w}^\top \boldsymbol{w} \right)$$

Hinge-loss: Non-smooth function at y w'x = 1



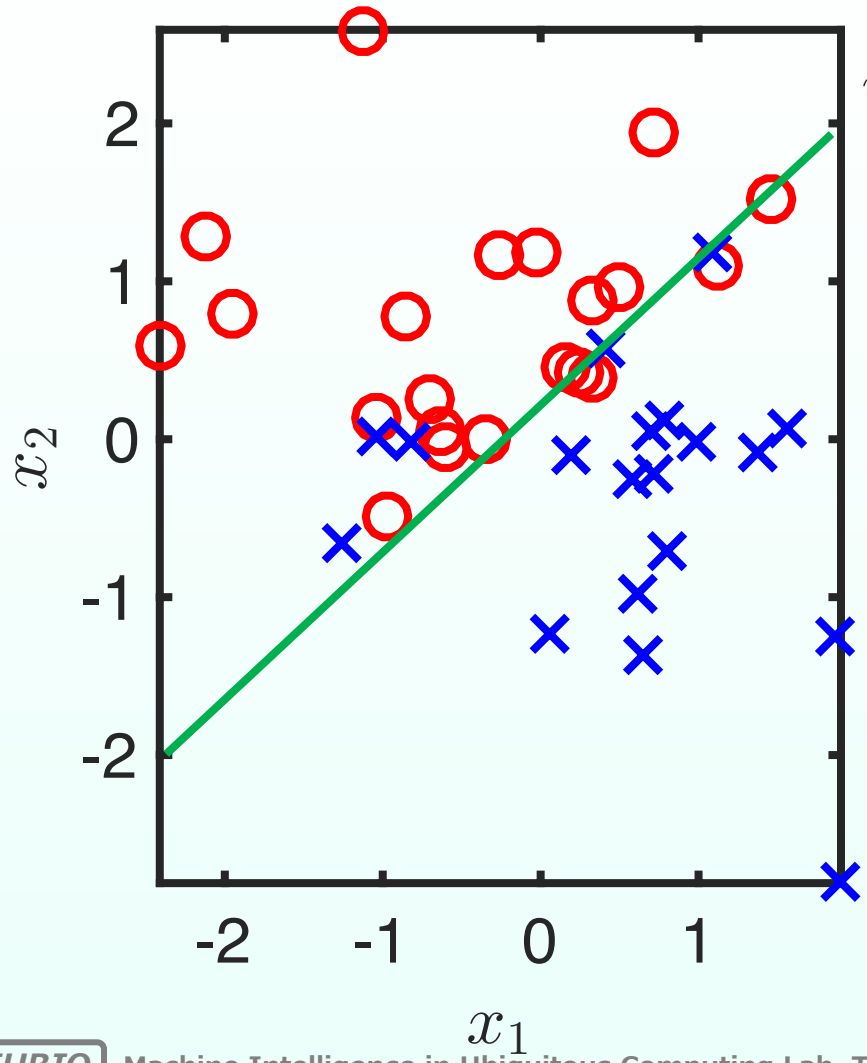
- $\boldsymbol{g} \in \partial l_i(\boldsymbol{w})$: Subgradient of hinge-loss can be written as

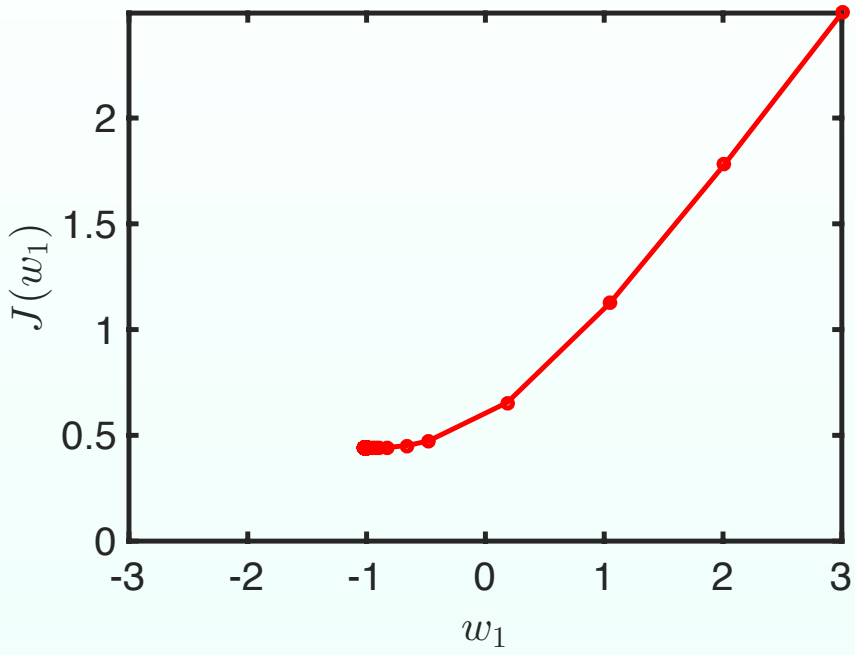
$$\boldsymbol{g} = \begin{cases} \square & \text{if } y_i \boldsymbol{w}^\top \boldsymbol{x}_i > 1 \\ & \text{if } y_i \boldsymbol{w}^\top \boldsymbol{x}_i = 1 \\ & \text{if } y_i \boldsymbol{w}^\top \boldsymbol{x}_i < 1 \end{cases}, \ \eta \in [0, 1]$$

- We can apply gradient descent with the subgradient of hinge-loss
- The subgradient method for this model is like "perceptron" learning

Toy example of subgradient based SVM

Binary classification toy example with 2d input





Projected gradient

Review: constraint optimization

Constraint optimization

- Separating non-smooth function from objective into constraints

$$\left.\begin{array}{rl} \text{minimize} & \psi(\boldsymbol{w}) \\ \text{subject to} & g_i(\boldsymbol{w}) = 0,\ i = 1, \ldots, m \\ & h_i(\boldsymbol{w}) \leq 0,\ i = 1, \ldots, r \end{array}\right\} \ \boldsymbol{w} \in \mathcal{C}$$

- We can rewrite constraint condition as region $\mathcal{C}$

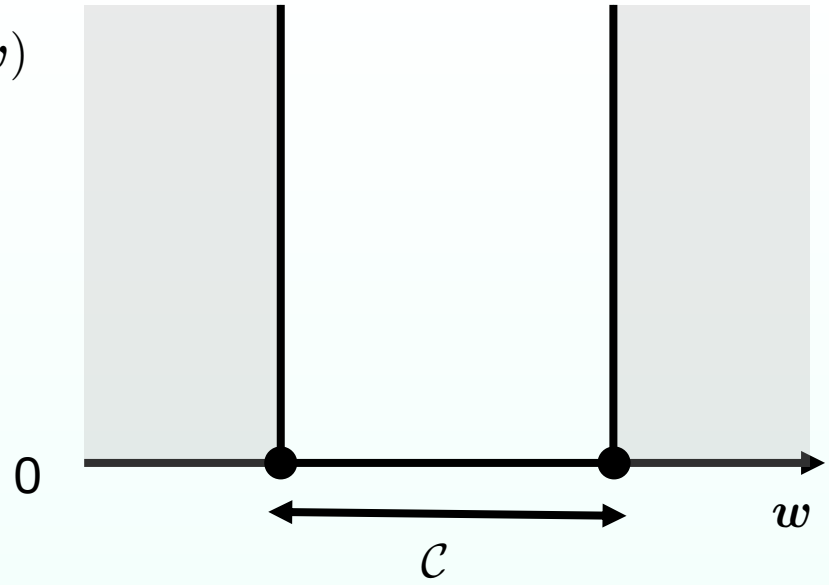
- This problem can be written as $\text{minimize}_{\boldsymbol{w} \in \mathcal{C}} \psi(\boldsymbol{w})$

- From now, we consider (proximal) gradient method for this problem

Indicator function of Region

Indicator function for (proximal) gradient method

$$I_{\mathcal{C}}(\boldsymbol{w}) = \begin{cases} 0 & \boldsymbol{w} \in \mathcal{C} \\ \infty & \boldsymbol{w} \notin \mathcal{C} \end{cases}$$



The problem can be written as

$$\text{minimize}_{\boldsymbol{w} \in \mathcal{C}} \psi(\boldsymbol{w}) \leftrightarrow \text{minimize}_{\boldsymbol{w}} \psi(\boldsymbol{w}) + I_{\mathcal{C}}(\boldsymbol{w})$$

Applying proximal gradient with indicator function

The problem we consider now

$$\text{minimize}_{\boldsymbol{w}} \psi(\boldsymbol{w}) + I_{\mathcal{C}}(\boldsymbol{w})$$

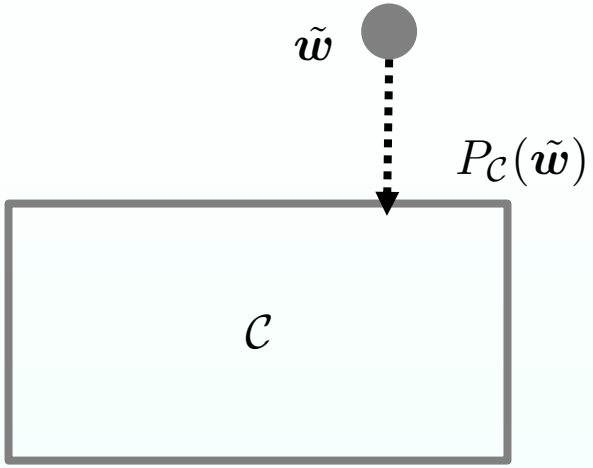
Proximal gradient with indicator function

$$\boldsymbol{w}^{(t)} = \text{prox}_{\eta_t I_{\mathcal{C}}}(\underbrace{\boldsymbol{w}^{(t-1)} - \eta_t \nabla \psi(\boldsymbol{w}^{(t-1)})}_{\tilde{\boldsymbol{w}}})$$

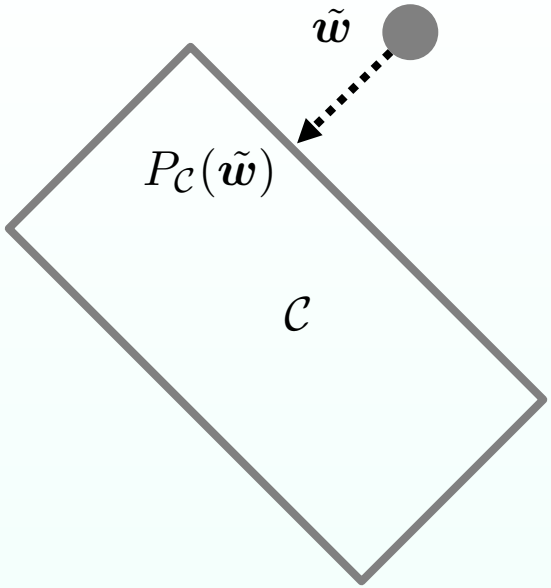
$$\begin{aligned}
\text{prox}_{\eta_t I_{\mathcal{C}}}(\tilde{\boldsymbol{w}}) &= \underset{\boldsymbol{w}}{\text{argmin}} \left(\frac{1}{2} ||\boldsymbol{w} - \tilde{\boldsymbol{w}}||_2^2 + I_{\mathcal{C}}(\boldsymbol{w}) \right) \\
&= \underset{\boldsymbol{w} \in \mathcal{C}}{\text{argmin}} \left(||\boldsymbol{w} - \tilde{\boldsymbol{w}}||_2^2 \right) \\
&= P_{\mathcal{C}}(\tilde{\boldsymbol{w}})
\end{aligned}$$

Toy example of projection in 2d space

Case 1



Case 2



We can analytically obtain projection step for

- Ex. Box constraint $\mathcal{C} = \{\boldsymbol{w} | w_i \in [r_i, R_i], \forall i = 1, \ldots, d, r_i \leq R_i\}$

$$\{P_{\mathcal{C}}(\tilde{\boldsymbol{w}})\}_i = \begin{cases} R_i & \text{if } \tilde{w}_i \geq R_i \\ \tilde{w}_i & \text{if } \tilde{w}_i \in [r_i, R_i] \\ r_i & \text{if } \tilde{w}_i < r_i \end{cases}$$

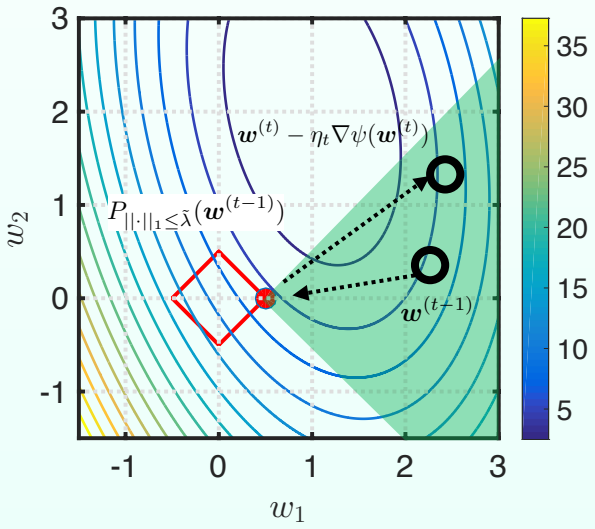
Review: Geometric interpretation of Lasso

Lasso and its equivalent problem (with L1 constraints)

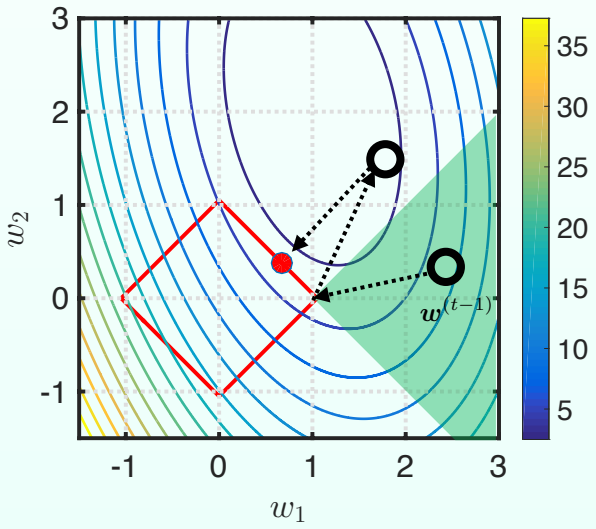
$$\underset{\boldsymbol{w}}{\text{argmin}} \left(\frac{1}{2} ||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2 + \lambda ||\boldsymbol{w}||_1 \right) \Longleftrightarrow \begin{array}{ll} \text{minimize}_{\boldsymbol{w}} & \frac{1}{2} ||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2 \\ \text{subject to} & ||\boldsymbol{w}||_1 \leq \tilde{\lambda} \end{array}$$

– We consider to apply projected gradient for constraint problem

With small $\tilde{\lambda}$, parameter is sparse



With large $\tilde{\lambda}$, parameter is not sparse



Stochastic optimization

Batch training is not always feasible in large scale machine learning

Batch training

- The optimization methods we had thought are known as batch training
- In batch training, all the training data for loss and its gradient are used

$$\hat{f}^{\mathrm{REG}}(\boldsymbol{x}) = \underset{f}{\operatorname{argmin}} \underbrace{\left(\underbrace{\frac{1}{n}\sum_{i=1}^{n} l(f(\boldsymbol{x}_i), y_i)}_{\text{loss} : \psi(\boldsymbol{w})} + \underbrace{\lambda\Omega(\boldsymbol{w})}_{\text{regularization} : \Omega(\boldsymbol{w})} \right)}_{J(\boldsymbol{w})}$$

- We always use gradient information: $\dfrac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}$
  - In linear logistic regression, this requires $O(dn)$ operation per iteration

- For large scale ML, we want to avoid such computational cost

Stochastic optimization

Original (batch training)

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left(\frac{1}{n} \sum_{i=1}^{n} l(\boldsymbol{x}_i, y_i, \boldsymbol{w}) + \lambda \Omega(\boldsymbol{w}) \right) = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left(\frac{1}{n} \sum_{i=1}^{n} l_i(\boldsymbol{w}) + \lambda \Omega(\boldsymbol{w}) \right)$$

Loss per sample　regularization : $\Omega(\boldsymbol{w})$　Loss per sample
(with simplified representation)

(Online) Stochastic optimization

- Iterate some update process $t = 1, \ldots, T$ with initial guess $\boldsymbol{w}^{(0)}$
  - 1. $z_t \sim \mathcal{U}(z|n)$: randomly pickup index ranging from 1 to n
  - 2. update parameter $\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)}$ using $l_{z_t}\left(\boldsymbol{w}^{(t-1)}\right)$, and $\lambda\Omega(\boldsymbol{w}^{(t-1)})$

Equivalent problem with (online) stochastic optimization

During iteration process, it is equivalent to solve the following problem

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left(\frac{1}{T} \sum_{t=1}^{T} l_{z_t}(\boldsymbol{w}) + \lambda \Omega(\boldsymbol{w}) \right)$$

Update parameter with finite iteration

- If T → ∞, the problem is equivalent to the following problem

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left(\mathbb{E}_{z \sim \mathcal{U}(z|n)} \left[l_z(\boldsymbol{w}) \right] + \lambda \Omega(\boldsymbol{w}) \right)$$

Expectation of per sample loss function

C.f. batch training

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left(\frac{1}{n} \sum_{i=1}^{n} l_i(\boldsymbol{w}) + \lambda \Omega(\boldsymbol{w}) \right)$$

Stochastic gradient descent (SGD)

Applying (proximal) GD to stochastic optimization

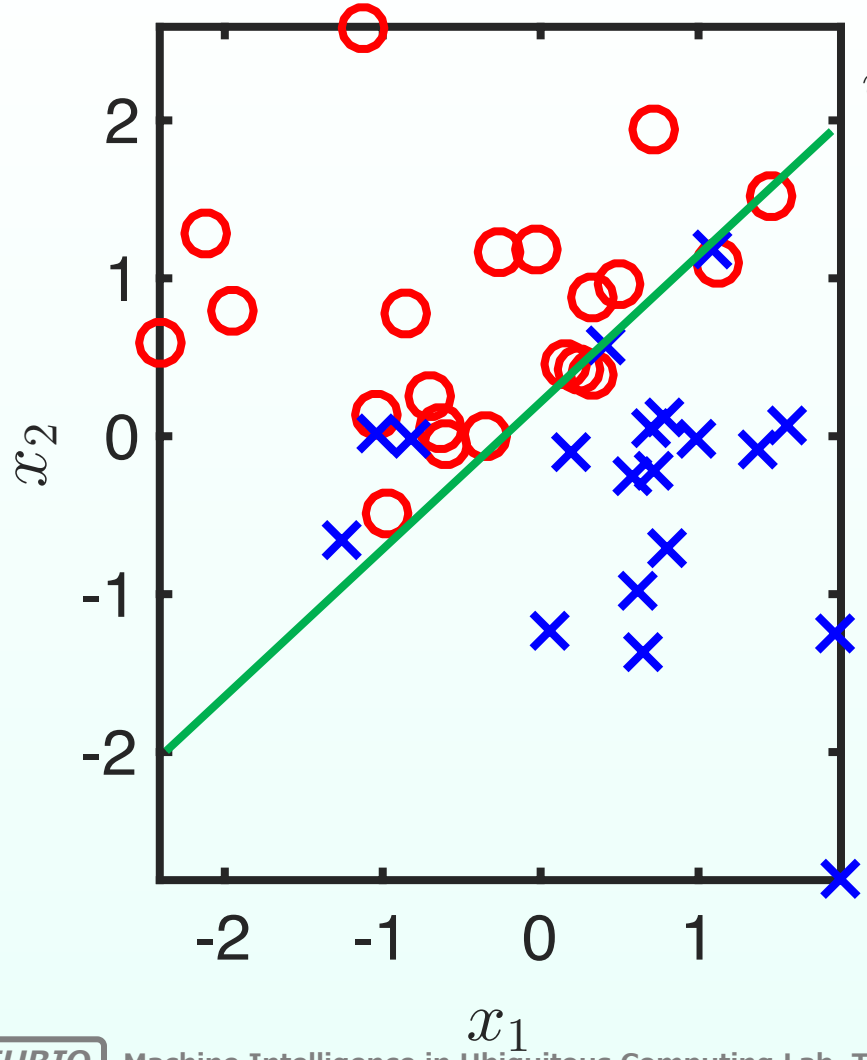
- Iterate some update process $t = 1, \ldots, T$ with initial guess: $\boldsymbol{w}^{(0)}$
  - 1. $z_t \sim \mathcal{U}(z|n)$: randomly pickup index ranging from 1 to n
  - 2. compute gradient information from single sample: $\boldsymbol{g}_t \in \partial l_{z_t}(\boldsymbol{w}^{(t-1)})$
  - 3. update the parameter using gradient (with proximal operation)

$$\boldsymbol{w}^{(t)} = \mathrm{prox}_{\eta_t \lambda \Omega(\cdot)}(\boldsymbol{w}^{(t-1)} - \eta_t \boldsymbol{g}_t)$$

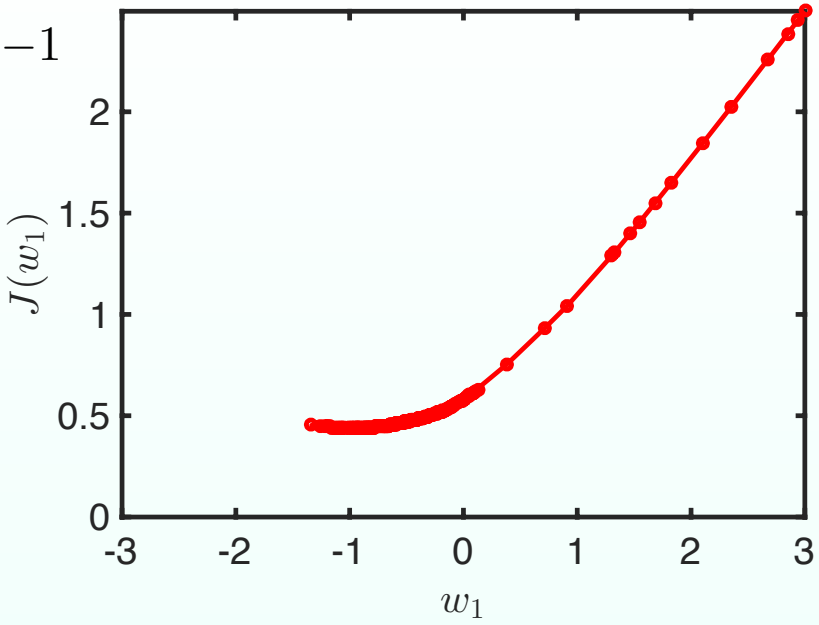
- Comparison with batch training
  - Gradient is made from the single sample instead of whole dataset
- Mini-batch training
  - Mix of Batch and (online) stochastic training
  - Use $m \ll n$ samples for obtaining gradient rather than single sample

Action of SGD with toy example

L2 regularized hinge loss minimization in 2d



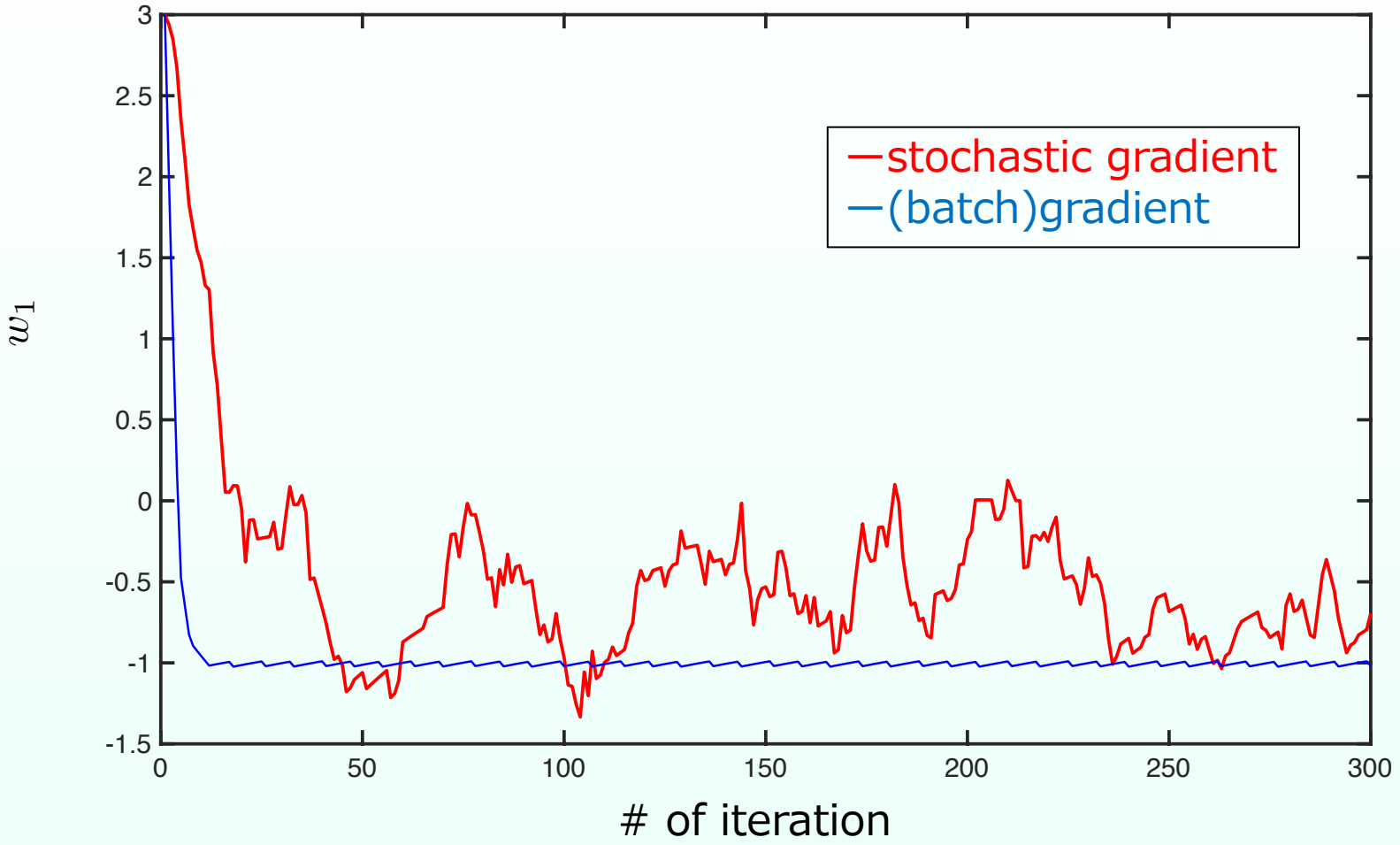
$w_1 = -1$



Action of SGD with toy example (cont'd)

Stochastic gradient converges gradually

- With small size dataset, use batch training!



Stochastic gradient with momentum

Momentum instead of using gradient from single data

- To remove the instability due to the robustness

- Original

$$\boldsymbol{g}_t \in \partial l_{z_t}(\boldsymbol{w}^{(t-1)})$$

$$\boldsymbol{w}^{(t)} = \mathrm{prox}_{\eta_t \lambda \Omega(\cdot)}(\boldsymbol{w}^{(t-1)} - \eta_t \boldsymbol{g}_t)$$

- Momentum update (like acceleration)

$$\boldsymbol{g}_t \in \partial l_{z_t}(\boldsymbol{w}^{(t-1)})$$

$$\begin{aligned}\boldsymbol{\mu}_t &= \beta \boldsymbol{\mu}_{t-1} + (1-\beta)\boldsymbol{g}_t \\ &= (1-\beta)\left(\boldsymbol{g}_t + \sum_{\tau} \beta^{\tau} \boldsymbol{g}_{t-\tau}\right), \ \beta \in (0,1)\end{aligned}$$

Momentum

a.k.a. Polynomial decay averaging

$$\boldsymbol{w}^{(t)} = \mathrm{prox}_{\eta_t \lambda \Omega(\cdot)}(\boldsymbol{w}^{(t-1)} - \eta_t \boldsymbol{\mu}_t)$$

Convergence rate of SGD

Gap between estimation and ground truth

$\mathcal{E}_{z_{1:T}\sim\mathcal{U}(z|n)}\left[J(\boldsymbol{w}^{(T)})\right] - J(\hat{\boldsymbol{w}})$ **will be bounded by**

Non α-strongly convex	α-strongly convex
$O(\frac{1}{\sqrt{T}})$	$O(\frac{1}{T})$

- Ground truth $\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left(\sum_i l_i(\boldsymbol{w}) + \lambda\Omega(\boldsymbol{w}) \right)$
  - Usually, theoretical analysis uses weighted average $\bar{\boldsymbol{w}}^{(T)} = \frac{1}{T+1}\sum_{\tau=0}^{T} \boldsymbol{w}^{(\tau)}$, and its variants rather than using $\boldsymbol{w}^{(T)}$
- C.f. Regret is also frequently used $R(T) = \sum_{t=1}^{T}\left(l_{z_t}\left(\boldsymbol{w}^{(t)}\right) + \lambda\Omega(\boldsymbol{w}^{(t)})\right) - \inf_{\boldsymbol{w}} \sum_{t=1}^{T} (l_{z_t}(\boldsymbol{w}) + \lambda\Omega(\boldsymbol{w}))$

Convergence rate of SGD comparison with batch training

- We assume alpha strongly convex function here
- We omit dimensionality here (since this has no effects in comparison)

	Batch	(online) stochastic
Per iteration	$O(n)$	$O(1)$
# of iteration to reach the gap is bounded by ϵ	$O(-\ln \epsilon)$	$O(\epsilon^{-1})$
Total cost	$O(-n \ln \epsilon)$	$O(\epsilon^{-1})$
Generalization error bounded by $O(1/n)$.	$O(n \ln n)$	$O(n)$

We assume loss is α-strongly convex function

- We need additional iteration to reach the accuracy in stochastic case, but we do not need heavy computational cost for loss calculation in each iteration
- **The total cost can be reduced to $O(n)$ from $O(n \ln n)$**

Adaptive step size estimation in (stochastic) optimization

Review: (stochastic)(proximal) gradient method

Now, you can use "gradient" method for large scale ML

- Thanks to stochastic (sub-)gradient
- With Non-smooth (sparse regularization) term thanks to prox. ops

$$\boldsymbol{w}^{(t)} = \mathrm{prox}_{\eta_t \lambda \Omega}(\boldsymbol{w}^{(t-1)} - \eta_t \boldsymbol{g}_t)$$

$$\boldsymbol{g}_t \in \partial l_{z_t}(\boldsymbol{w}^{(t-1)})$$

$$z_t \sim \mathcal{U}(z|n)$$

- As for step size, we assume it is fix (w.r.t. γ smooth) or back-tracking

Review: (stochastic)(proximal) gradient method

Now, you can use "gradient" method for large scale ML

- Thanks to stochastic (sub-)gradient
- With Non-smooth (sparse regularization) term thanks to prox. ops

$$\boldsymbol{w}^{(t)} = \mathrm{prox}_{\eta_t \lambda \Omega}(\boldsymbol{w}^{(t-1)} - \eta_t \boldsymbol{g}_t)$$

$$\boldsymbol{g}_t \in \partial l_{z_t}(\boldsymbol{w}^{(t-1)})$$

$$z_t \sim \mathcal{U}(z|n)$$

- As for step size, we assume it is fix (w.r.t. γ smooth) or back-tracking

Is it really reasonable for any (convex) optimization?

Review: (stochastic)(proximal) gradient method

Now, you can use "gradient" method for large scale ML

- Thanks to stochastic (sub-)gradient
- With Non-smooth (sparse regularization) term thanks to prox. ops

$$\boldsymbol{w}^{(t)} = \mathrm{prox}_{\eta_t \lambda \Omega}(\boldsymbol{w}^{(t-1)} - \eta_t \boldsymbol{g}_t)$$

$$\boldsymbol{g}_t \in \partial l_{z_t}(\boldsymbol{w}^{(t-1)})$$

$$z_t \sim \mathcal{U}(z|n)$$

- As for step size, we assume it is fix (w.r.t. γ smooth) or back-tracking

Is it really reasonable for any (convex) optimization?

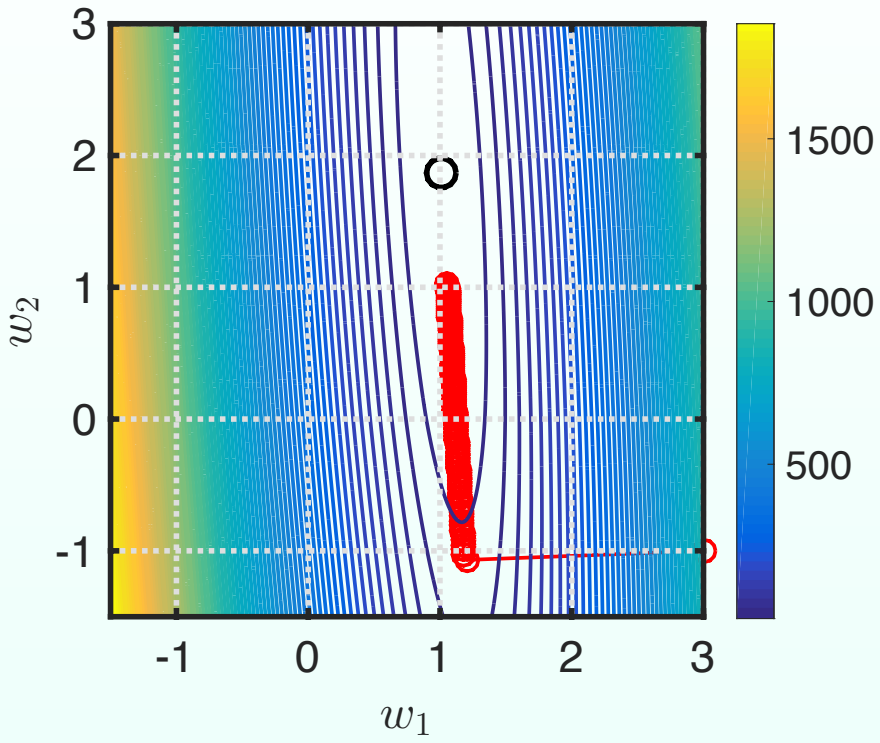
Actually, $\eta_t = \gamma^{-1}$ is not so efficient, but we could not control it automatically

GD in action with toy example

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \left((\boldsymbol{w}-\boldsymbol{\mu})^\top \boldsymbol{A} (\boldsymbol{w}-\boldsymbol{\mu}) + \lambda ||\boldsymbol{w}||_1 \right)$$

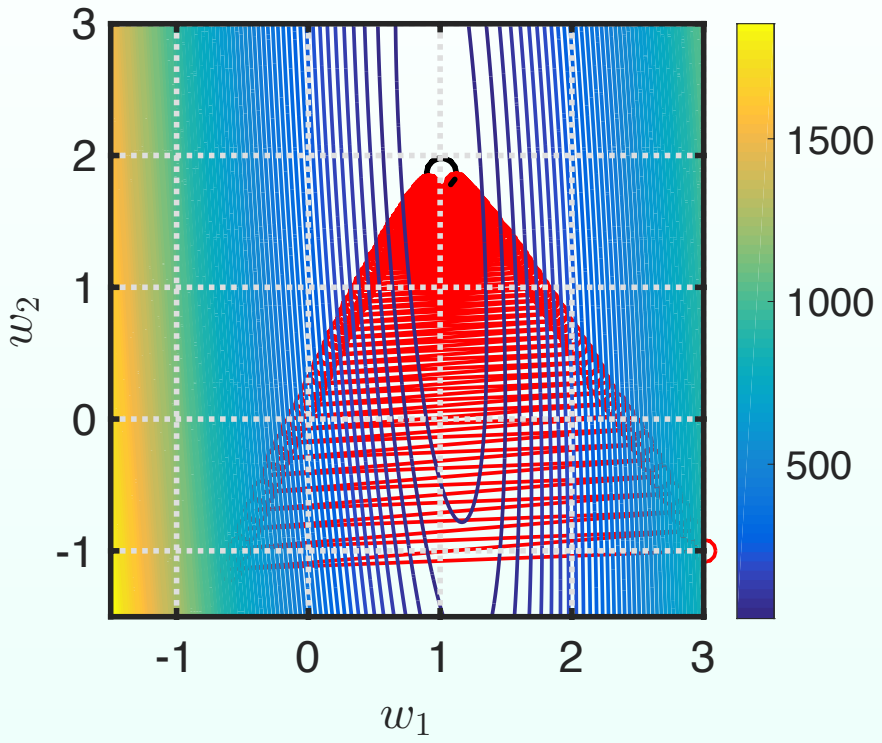
$$\boldsymbol{A} = \begin{pmatrix} 250 & 15 \\ 15 & 4 \end{pmatrix}, \boldsymbol{\mu}^\top = \begin{pmatrix} 1 & 2 \end{pmatrix}, \lambda = 0.89, \eta_t = 1/\gamma$$

Too slow to reach minima



$$\boldsymbol{A} = \begin{pmatrix} 250 & 15 \\ 15 & 4 \end{pmatrix}, \boldsymbol{\mu}^\top = \begin{pmatrix} 1 & 2 \end{pmatrix}, \lambda = 0.89, \eta_t = 2/\gamma$$

Too vibrated to reach minima



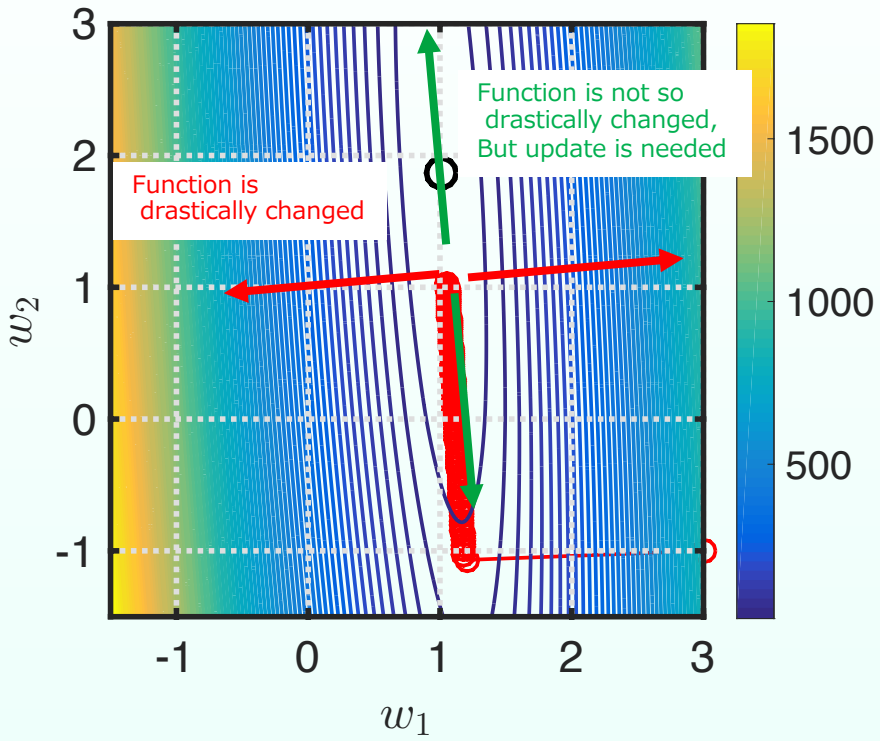
$$\gamma = \max \operatorname{eig}(2\boldsymbol{A})$$

GD in action with toy example

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \left((\boldsymbol{w}-\boldsymbol{\mu})^\top \boldsymbol{A} (\boldsymbol{w}-\boldsymbol{\mu}) + \lambda ||\boldsymbol{w}||_1 \right)$$

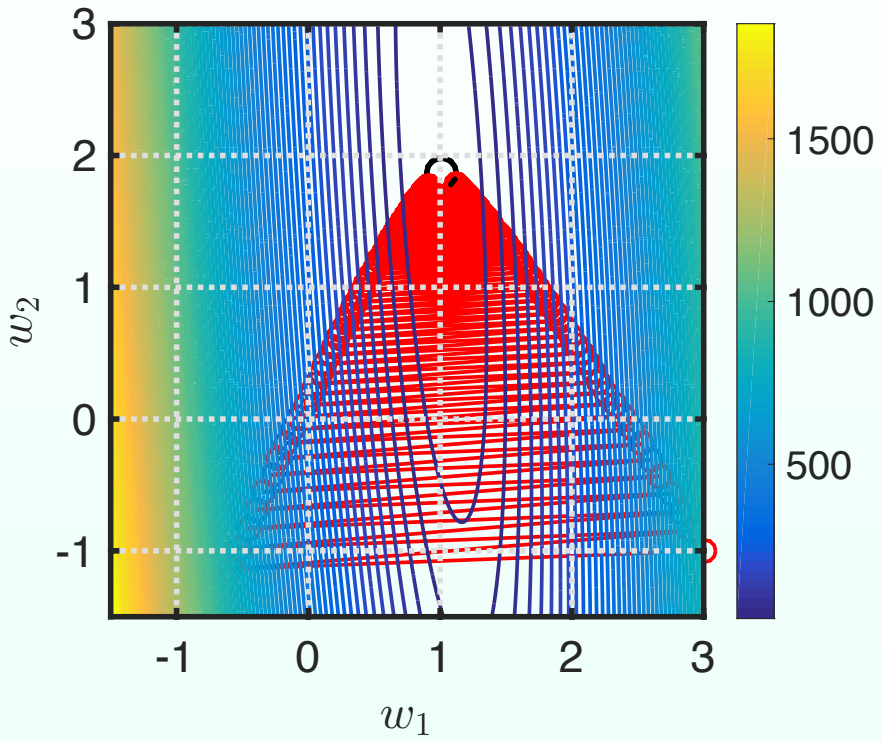
$$\boldsymbol{A} = \begin{pmatrix} 250 & 15 \\ 15 & 4 \end{pmatrix}, \boldsymbol{\mu}^\top = \begin{pmatrix} 1 & 2 \end{pmatrix}, \lambda = 0.89, \eta_t = 1/\gamma$$

Too slow to reach minima



$$\boldsymbol{A} = \begin{pmatrix} 250 & 15 \\ 15 & 4 \end{pmatrix}, \boldsymbol{\mu}^\top = \begin{pmatrix} 1 & 2 \end{pmatrix}, \lambda = 0.89, \eta_t = 2/\gamma$$

Too vibrated to reach minima



$$\gamma = \max \operatorname{eig}(2\boldsymbol{A})$$

We could not use info on ratio on 1st and 2nd eigen values

One approach is to use momentum (averaged of gradients)

Review: Newton method

Use 2nd order information for the optimization

$$\boldsymbol{w}^{(t)} = \boldsymbol{w}^{(t-1)} - \boldsymbol{H}(\boldsymbol{w}^{(t-1)})^{-1}\boldsymbol{g}(\boldsymbol{w}^{(t-1)})$$

- For batch training
- For smooth function optimization
- Let's consider toy problem $\underset{\boldsymbol{w}}{\operatorname{argmin}} \left((\boldsymbol{w}-\boldsymbol{\mu})^\top \boldsymbol{A}(\boldsymbol{w}-\boldsymbol{\mu}) + \lambda||\boldsymbol{w}||_1\right)$
- If λ = 0, $\boldsymbol{w}^{(t)} = \boldsymbol{w}^{(t-1)} - \eta_t(\boldsymbol{w}^{(t-1)} - \boldsymbol{\mu})$ The param is directly moved towards minima
- 2nd order info cancel the variation of scale of gradients
  - Like whitening with Mahalanobis distance
- But we could not use Newton method for non-smooth, stochastic case

Newton-like proximal operation

Proximal operation with gradient information

- Use quadratic function that covers objective around $\boldsymbol{w}^{(t-1)}$

$$\boldsymbol{w}^{(t)} = \mathrm{prox}_{\eta_t \lambda \Omega}(\boldsymbol{w}^{(t-1)} - \eta_t \boldsymbol{g}_t)$$

$$= \underset{\boldsymbol{w}}{\mathrm{argmin}} \left((\boldsymbol{w} - \boldsymbol{w}^{(t-1)})^\top \boldsymbol{g}_t + \lambda \Omega(\boldsymbol{w}) + \frac{1}{2\eta_t} ||\boldsymbol{w} - \boldsymbol{w}^{(t-1)}||_2^2 \right)$$

Common in two methods

Use Euclid distance

Approximating loss function with 2nd Taylor expansion

- Newton like approach

$$\boldsymbol{w}^{(t)} = \underset{\boldsymbol{w}}{\mathrm{argmin}} \left((\boldsymbol{w} - \boldsymbol{w}^{(t-1)})^\top \boldsymbol{g}_t + \lambda \Omega(\boldsymbol{w}) + \frac{1}{2} (\boldsymbol{w} - \boldsymbol{w}^{(t-1)})^\top \hat{\boldsymbol{H}}(\boldsymbol{w}^{(t-1)}) (\boldsymbol{w} - \boldsymbol{w}^{(t-1)}) \right)$$

Common in two methods

Use curvature of the objective

- It looks nice only if we can ignore computational cost

$$\hat{\boldsymbol{H}}(\boldsymbol{w}^{(t-1)}) \approx \nabla^2 \left(\sum_i l_i(\boldsymbol{w}^{(t-1)}) \right)$$

Obtaining pseudo Hessian in stochastic optimization

We consider to obtain Hessian like matrix H_t from sequence of gradient information

$$\boldsymbol{w}^{(t)} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left((\boldsymbol{w} - \boldsymbol{w}^{(t-1)})^\top \boldsymbol{g}_t + \lambda \Omega(\boldsymbol{w}) + \frac{1}{2} (\boldsymbol{w} - \boldsymbol{w}^{(t-1)})^\top \hat{\boldsymbol{H}}(\boldsymbol{w}^{(t-1)}) (\boldsymbol{w} - \boldsymbol{w}^{(t-1)}) \right)$$

Infeasible to obtain Hessian directly

$$\boldsymbol{w}^{(t)} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left((\boldsymbol{w} - \boldsymbol{w}^{(t-1)})^\top \boldsymbol{g}_t + \lambda \Omega(\boldsymbol{w}) + \frac{1}{2\eta_0} ||\boldsymbol{w} - \boldsymbol{w}^{(t-1)}||^2_{\boldsymbol{H}_t} \right)$$

$$\eta_t = \eta_0$$

$$||\boldsymbol{w} - \boldsymbol{w}^{(t-1)}||^2_{\boldsymbol{H}_t} = \left(\boldsymbol{w} - \boldsymbol{w}^{(t-1)} \right)^\top \boldsymbol{H}_t \left(\boldsymbol{w} - \boldsymbol{w}^{(t-1)} \right)$$

- A generalization form of proximal gradient method
- If we assume, $\boldsymbol{H}_t = \boldsymbol{I}, \eta_t = \gamma^{-1}$,
  the algorithm is equivalent to standard proximal gradient method

Design policy of pseudo Hessian

Use BFGS formula (quasi Newton)?

- It is not suitable to obtain pseudo Hessian in stochastic optimization

Design policy for creating pseudo Hessian

- Computationally efficient (calculation cost + memory storage)

Designing diagonal matrix from gradient history

$$\boldsymbol{H}_t = \boldsymbol{G}_t^{1/2} + \delta \boldsymbol{I}$$

$$\{\boldsymbol{G}_t\}_{i,j} = \begin{cases} 0 & \text{if } i \neq j \\ \sum_{\tau=1}^{t} \{\boldsymbol{g}_\tau\}_i^2 & \text{otherwise} \end{cases}$$

- The gradient is normalized w.r.t. the size of gradients
- The algorithm is known as AdaGrad

AdaGrad as adaptive proximal gradient

AdaGrad

- Sequential stochastic optimization problem with proximal gradients

$$\boldsymbol{w}^{(t)} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left((\boldsymbol{w} - \boldsymbol{w}^{(t-1)})^\top \boldsymbol{g}_t + \lambda \Omega(\boldsymbol{w}) + \frac{1}{2\eta_0} ||\boldsymbol{w} - \boldsymbol{w}^{(t-1)}||^2_{\boldsymbol{H}_t} \right)$$

- Use Hessian like information

$$\boldsymbol{H}_t = \boldsymbol{G}_t^{1/2} + \delta \boldsymbol{I}$$

$$\{\boldsymbol{G}_t\}_{i,j} = \begin{cases} 0 & \text{if } i \neq j \\ \sum_{\tau=1}^{t} \{\boldsymbol{g}_\tau\}_i^2 & \text{otherwise} \end{cases}$$

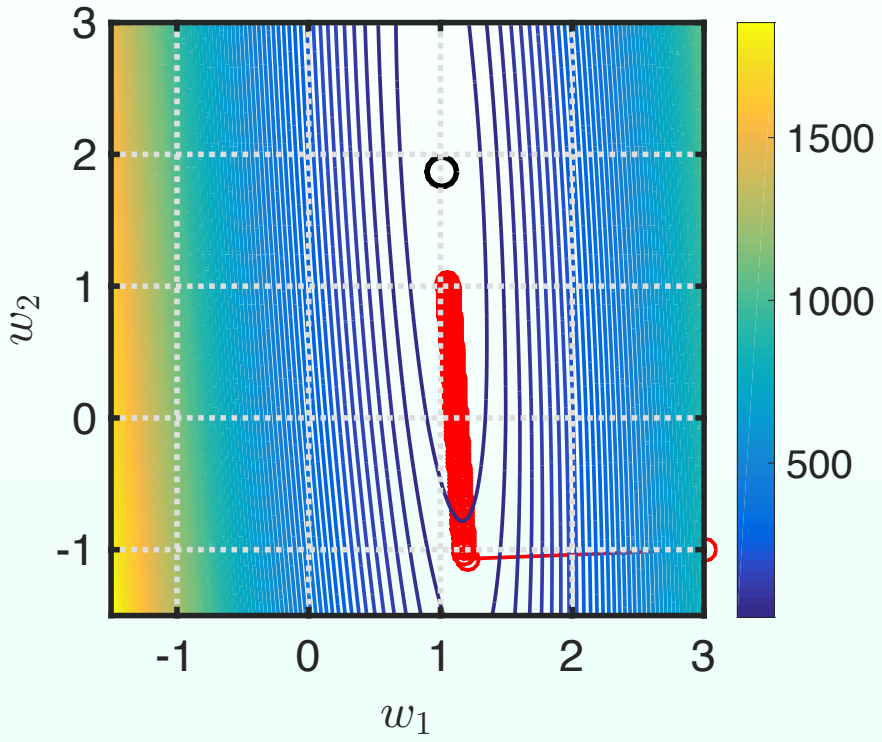
- Since $\boldsymbol{H}_t$ is diagonal, the parameter can be analytically solved by proximal operation. (operation cost is proportional to d)
  - Derive update formula
    for L1 case (with soft-thresholding), and square of L2 regularization(**a3)

$$\Omega(\boldsymbol{w}) = ||\boldsymbol{w}||_1 \qquad \Omega(\boldsymbol{w}) = ||\boldsymbol{w}||_2^2$$

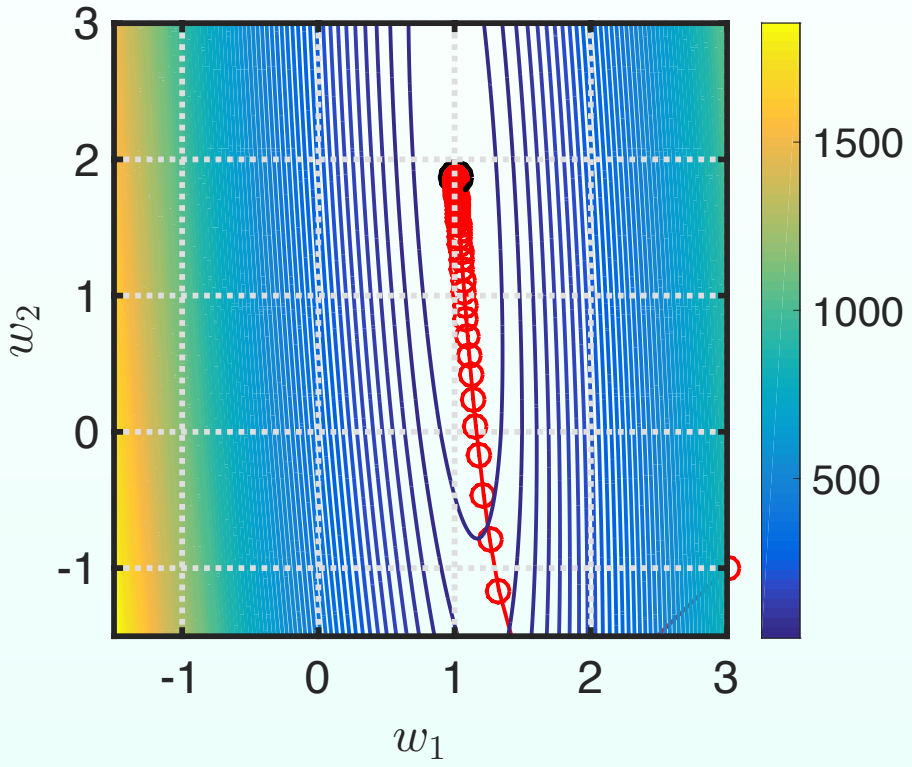
AdaGrad in action

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \left((\boldsymbol{w}-\boldsymbol{\mu})^\top \boldsymbol{A} (\boldsymbol{w}-\boldsymbol{\mu}) + \lambda ||\boldsymbol{w}||_1 \right) \quad \boldsymbol{A} = \begin{pmatrix} 250 & 15 \\ 15 & 4 \end{pmatrix}, \boldsymbol{\mu}^\top = \begin{pmatrix} 1 & 2 \end{pmatrix}, \lambda = 0.89$$

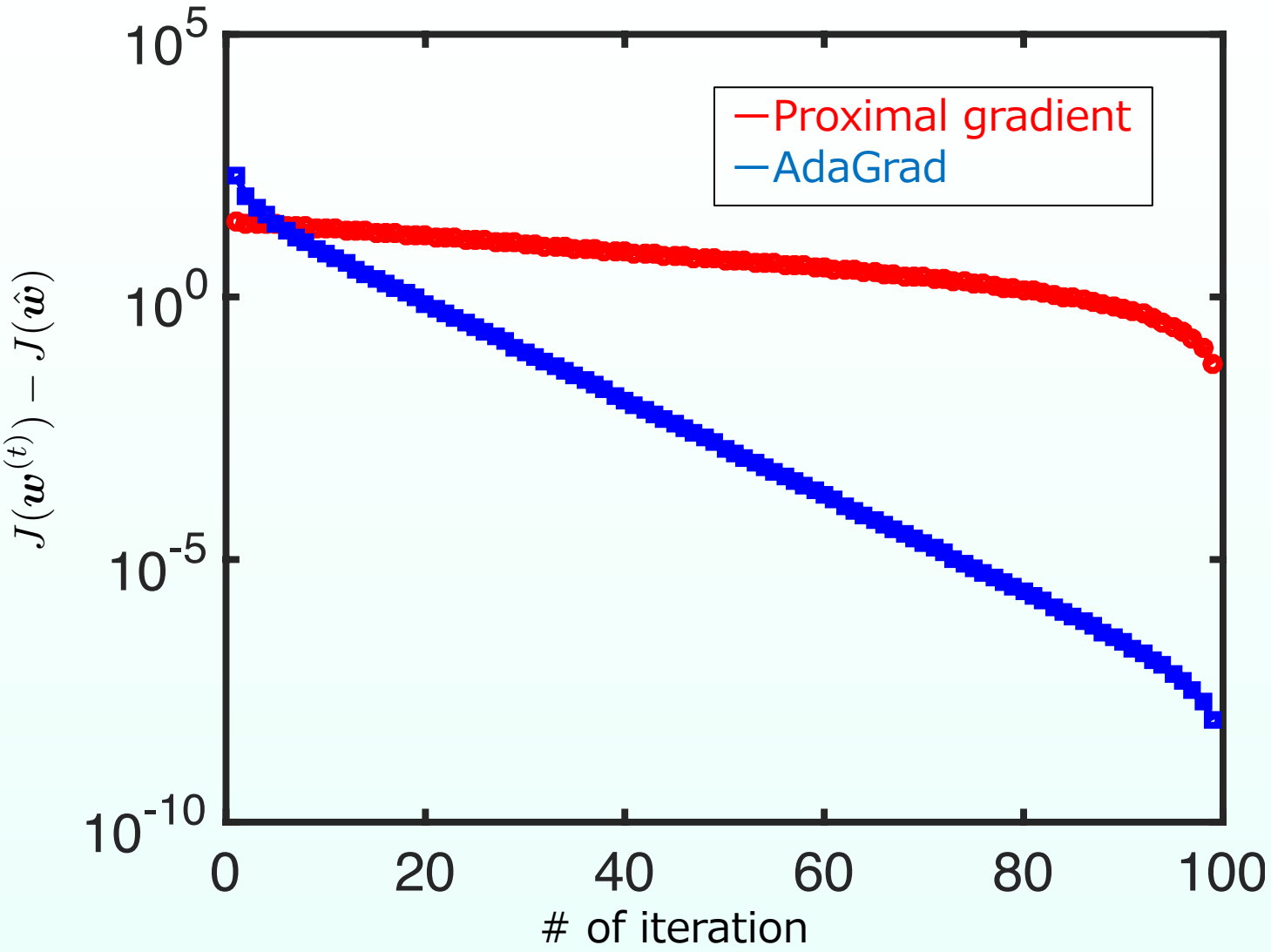
Proximal GD $\eta_t = \gamma^{-1}$



AdaGrad $\eta_t = 500\gamma^{-1}$ $\delta = 0.02$



AdaGrad in action (cont'd)



Advances in adaptive gradient methods

Adaptive gradient methods are actively explored in ML

- AdaGrad (2011)
- RMS Prop (2012)
- Ada Delta (2012)
- Adam (2015)
- Nadam (2016)
- …

Researchers want to create efficient optimization

- without hand tuning parameters (such as step size tuning)
- within computationally efficient manner

Adam: Adaptive moment estimation

Adam is similar to AdaGrad except that

- Adam uses momentum $\boldsymbol{\mu}_t$ rather than gradient
- Adam uses momentum $\boldsymbol{V}_t$ rather than sum of grad sq

AdaGrad $$\boldsymbol{w}^{(t)} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left(\left(\boldsymbol{w} - \boldsymbol{w}^{(t-1)} \right)^\top \boldsymbol{g}_t + \lambda \Omega(\boldsymbol{w}) + \frac{1}{2\eta_0} ||\boldsymbol{w} - \boldsymbol{w}^{(t-1)}||^2_{\boldsymbol{H}_t} \right)$$

Adam $$\boldsymbol{w}^{(t)} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left((\boldsymbol{w} - \boldsymbol{w}^{(t-1)})^\top \boldsymbol{\mu}_t + \lambda \Omega(\boldsymbol{w}) + \frac{1}{2\eta_0} ||\boldsymbol{w} - \boldsymbol{w}^{(t-1)}||^2_{\boldsymbol{V}_t} \right)$$

- Disclaimer
  The above formulation is just a variant of Adam.
  Though Adam paper does not present any issues related to proximal gradient.
  I would like to emphasize that proximal gradient could be used in adaptive gradient methods.

  Please refer the paper if you want to have original behavior.

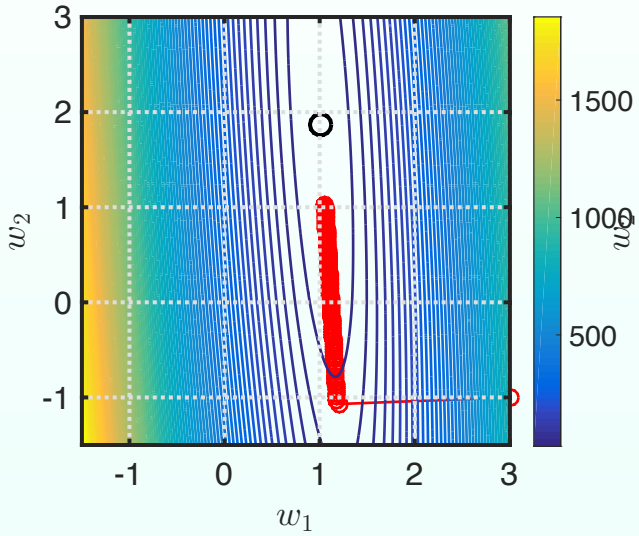
The paper provides theoretical guarantee

- Regret bound for convex function

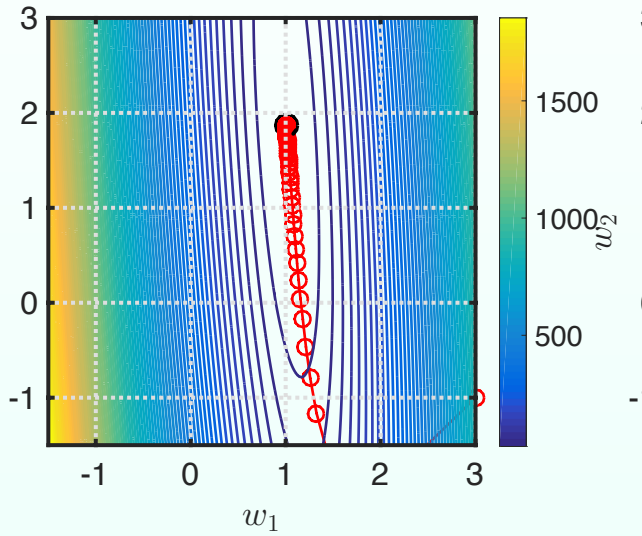
Adam in action

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \left((\boldsymbol{w}-\boldsymbol{\mu})^\top \boldsymbol{A} (\boldsymbol{w}-\boldsymbol{\mu}) + \lambda ||\boldsymbol{w}||_1 \right) \quad \boldsymbol{A} = \begin{pmatrix} 250 & 15 \\ 15 & 4 \end{pmatrix}, \boldsymbol{\mu}^\top = \begin{pmatrix} 1 & 2 \end{pmatrix}, \lambda = 0.89$$

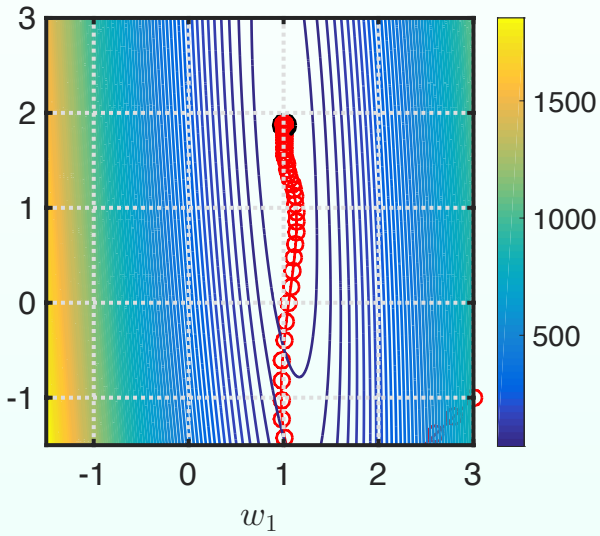
SGD



AdaGrad



Adam-like



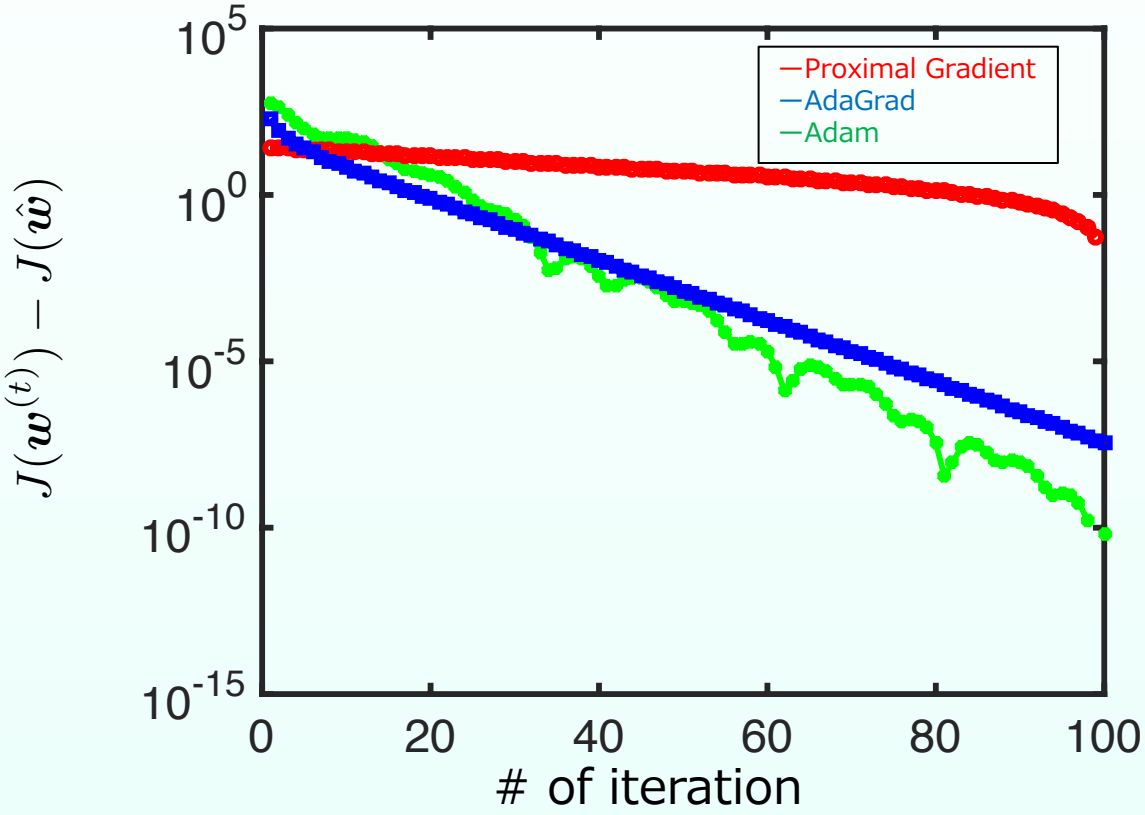
Adam in action(cont'd)

Adam, AdaGrad (vs SGD)

- Gets much better convergence with some condition $\sigma_1 \gg \sigma_2, \ldots$

Adam vs AdaGrad

- I could not give clear decision from this experiment…



Review of this lecture

Theoretical part: advances in gradient methods

- Accelerated proximal gradient methods
  - Convergence rate of proximal gradient methods
  - Acceleration, momentum, FISTA: Accelerated proximal gradient method
- Subgradient method
- Projected gradient method
- Stochastic optimization
  - From batch to stochastic way
  - vs. batch training in terms of convergence rate & computational cost
- Adaptive step size in gradient methods
  - AdaGrad, Adam

Bridge to deep learning

The optimization methods discussed today

- 1. Stochastic (vs batch)
- 2. Adaptive step-size (vs fixed rate or exact line search)
- 3. Momentum (vs standard gradient)
- 4. Proximal operation (vs smooth function minimization)

- verified by the convex programming theory with convergence rate

- Also used in deep learning community
  - The technique can be (practically) applied to non-convex programming
  - Researcher in ML tackle the theoretical analysis of non-convex programming

The issues for deep learning

- How to obtain **gradient** in deep layered model?
  - With **gradient,** you can use the optimization, such as AdaGrad, Adam!

Assignment “Shimosaka-6”

Solve the following problems & submit report via OCW-i

Format: pdf (scanned hand written report / tex)

Problem:

- Derive, or confirm the equations by your hand (**a1-3)

Preview of final assignment

Evaluation by me: 50pt in total

- Report in each lecture (I will not check the quality ☹)
  - 3pts x 6 times

- Final report (I will check the quality!)
  - Details will be announced soon
  - 30 pts in total.
  - You will be asked to select a couple of problems from 8–10 problems.

  - If you have questions about the final report,
    please send email to **titech-ml-course@miubiq.cs.titech.ac.jp**
    (DO NOT send email to this on any issues related to Okazaki-sensei's lecture)

- 2pts by your constructive criticism to my lecture (in final report)