Statistical Learning for Text Data Analytics
Unconstrained Optimization Techniques

Yangqiu Song

Hong Kong University of Science and Technology

yqsong@cse.ust.hk

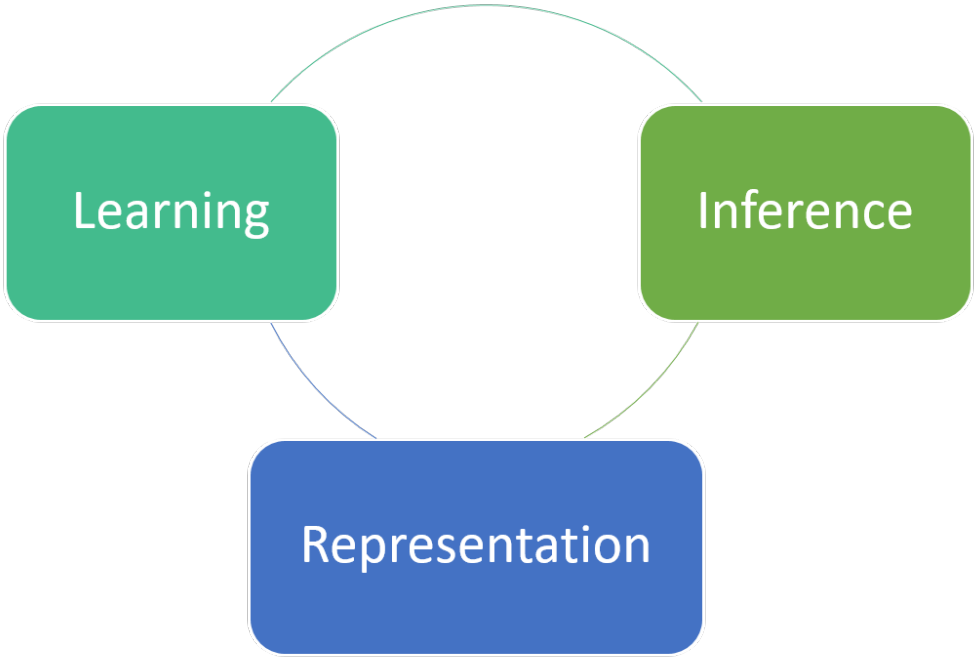
Spring 2018

∗Contents are based on materials created by Peter Richtérik, Mark Schmidt, Francis Bach, Tianbao Yang, Rong Jin, Shenghuo Zhu, and Qihang Lin

Reference Content

- Peter Richtérik and Mark Schmidt. ICML Tutorial on Modern Convex Optimization Methods for Large-scale Empirical Risk Minimization. https://icml.cc/2015/tutorials/2015_ICML_ConvexOptimization_I.pdf
- Francis Bach. NIPS 2016 Tutorial on Large-Scale Optimization: Beyond Stochastic Gradient Descent and Convexity. http://www.di.ens.fr/~fbach/fbach_tutorial_vr_nips_2016.pdf and http://www.di.ens.fr/~fbach/ssra_tutorial_vr_nips_2016.pdf
- Tianbao Yang, Qihang Lin, and Rong Jin. KDD Tutorial on Big Data Analytics: Optimization and Randomization. http://homepage.cs.uiowa.edu/~tyng/kdd15tutorial.html
- Tianbao Yang, Rong Jin and Shenghuo Zhu. SDM Tutorial on Stochastic Optimization for Big Data Analytics: Algorithms and Library. http://homepage.divms.uiowa.edu/~tyng/tutorial.html

Course Topics



- Representation: language models, word embeddings, topic models
- Learning: supervised learning, semi-supervised learning, sequence models, deep learning, optimization techniques
- Inference: constraint modeling, joint inference, search algorithms

NLP applications: tasks introduced in Lecture 1

Overview

Overview

Motivation: Sparse Regularization

- Recall the regularized empirical risk minimization problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{N} \underbrace{\sum_{i=1}^{N} \ell(\mathbf{w}, \mathbf{x}_i, y_i)}_{\text{Empirical Loss/Data Fitting}} + \underbrace{\lambda r(\mathbf{w})}_{\text{Regularization}}$$

- Often, regularizer r is used to encourage sparsity pattern in $\mathbf{w}$
- For example, ℓ_1-regularized least squares,

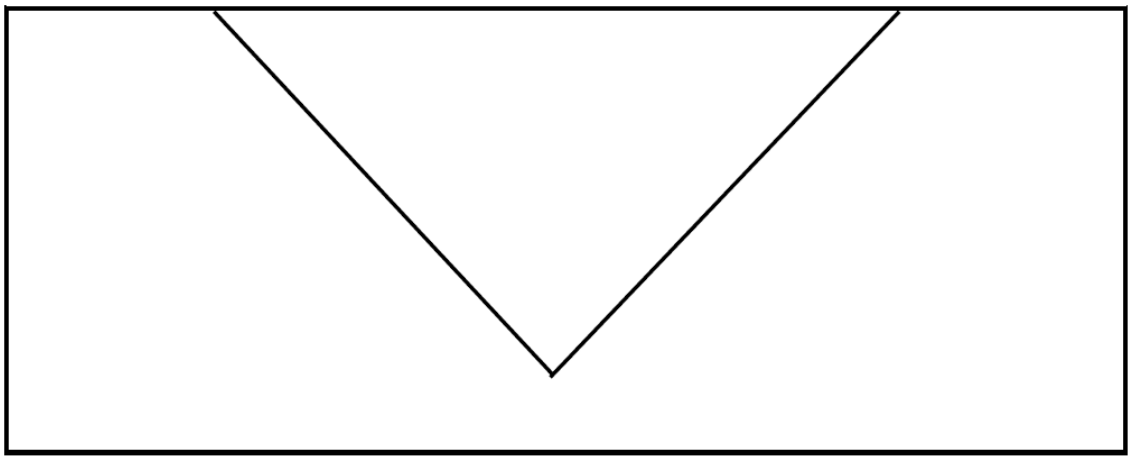
$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{N}\|\mathbf{X}\mathbf{w} - \mathbf{b}\|^2 + \lambda\|\mathbf{w}\|_1$$

- Regularizes and encourages sparsity in $\mathbf{w}$
- The objective is non-differentiable when any $\mathbf{w}_i = 0$
- Subgradient methods are optimal (slow) black-box methods
- Faster methods for specific non-smooth problems?

Smoothing Approximations of Non-Smooth Functions

- Smoothing: replace non-smooth f with smooth f_ξ
- Apply a fast method for smooth optimization
- Smooth approximation to the absolute value:

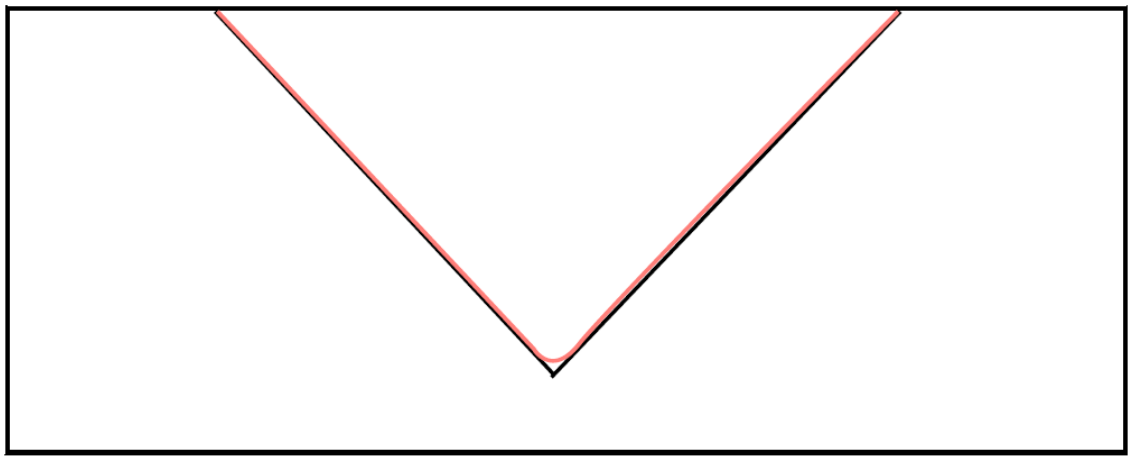
$$|x| \approx \sqrt{x^2 + \nu}$$



Smoothing Approximations of Non-Smooth Functions

- Smoothing: replace non-smooth f with smooth f_ξ
- Apply a fast method for smooth optimization
- Smooth approximation to the absolute value:

$$|x| \approx \sqrt{x^2 + \nu}$$



Smoothing Approximations of Non-Smooth Functions

- Smoothing: replace non-smooth f with smooth f_ξ
- Apply a fast method for smooth optimization
- Smooth approximation to the absolute value:

$$|x| \approx \sqrt{x^2 + \nu}$$

- Smooth approximation to the max function:

$$\max\{a, b\} \approx \log(\exp(a) + \exp(b))$$

- Smooth approximation to the hinge loss:

$$\max\{0, x\} \approx \begin{cases} 0 & x \geq 1 \\ 1 - x^2 & t < x < 1 \\ (1-t)^2 + 2(1-t)(t-x) & x \leq t \end{cases}$$

- Generic smoothing strategy: strongly-convex regularization of convex conjugate (Nesterov (2005))

Discussion of Smoothing Approach

- Nesterov (2005) shows that:
  - Gradient method on smoothed problem has $O(1/\sqrt{t})$ subgradient rate
  - Accelerated gradient method has faster $O(1/t)$ rate
- In practice:
  - Slowly decrease level of smoothing (often difficult to tune)
  - Use faster algorithms like L-BFGS, SAG, or SVRG

Converting to Constrained Optimization

- Re-write non-smooth problem as constrained problem
- The problem

$$\min_{\mathbf{w}} f(\mathbf{w}) + \lambda \|\mathbf{w}\|_1$$

is equivalent to the problem

$$\min_{\mathbf{w}^+ \geq 0, \mathbf{w}^- \geq 0} f(\mathbf{w}^+ - \mathbf{w}^-) + \lambda (\sum_i \mathbf{w}_i^+ + \mathbf{w}_i^-)$$

or

$$\min_{\forall i, -\mathbf{w'}_i \leq \mathbf{w}_i \leq \mathbf{w'}_i} f(\mathbf{w}) + \lambda \sum_i \mathbf{w'}_i$$

or

$$\min_{\|\mathbf{w}\|_1 \leq \gamma} f(\mathbf{w}) + \lambda\gamma$$

- These are smooth objective with "simple" constraints

$$\min_{\mathbf{w} \in \mathcal{C}} f(\mathbf{w})$$

Optimization with Simple Constraints

- Recall: gradient descent minimizes quadratic approximation:

$$f(\mathbf{w}') \leq f(\mathbf{w}) + \nabla f(\mathbf{w})^\top (\mathbf{w}' - \mathbf{w}) + \frac{1}{2\eta} \|\mathbf{w}' - \mathbf{w}\|^2$$

(Given the gradient is Lipschitz-continuous $\nabla^2 f(\mathbf{w}) \preceq LI$)

$$\mathbf{w}^{t+1} = \arg\min_{\mathbf{w}'} \{f(\mathbf{w}^t) + \nabla f(\mathbf{w}^t)^\top (\mathbf{w}' - \mathbf{w}^t) + \frac{1}{2\eta} \|\mathbf{w}' - \mathbf{w}^t\|^2\}$$

(Guaranteed progress; Projected Newton requires last term to be $\frac{1}{2\eta} \|\mathbf{w}' - \mathbf{w}^t\|^2_{\nabla^2 f(\mathbf{w}^t)}$)

- Consider minimizing subject to simple constraints:

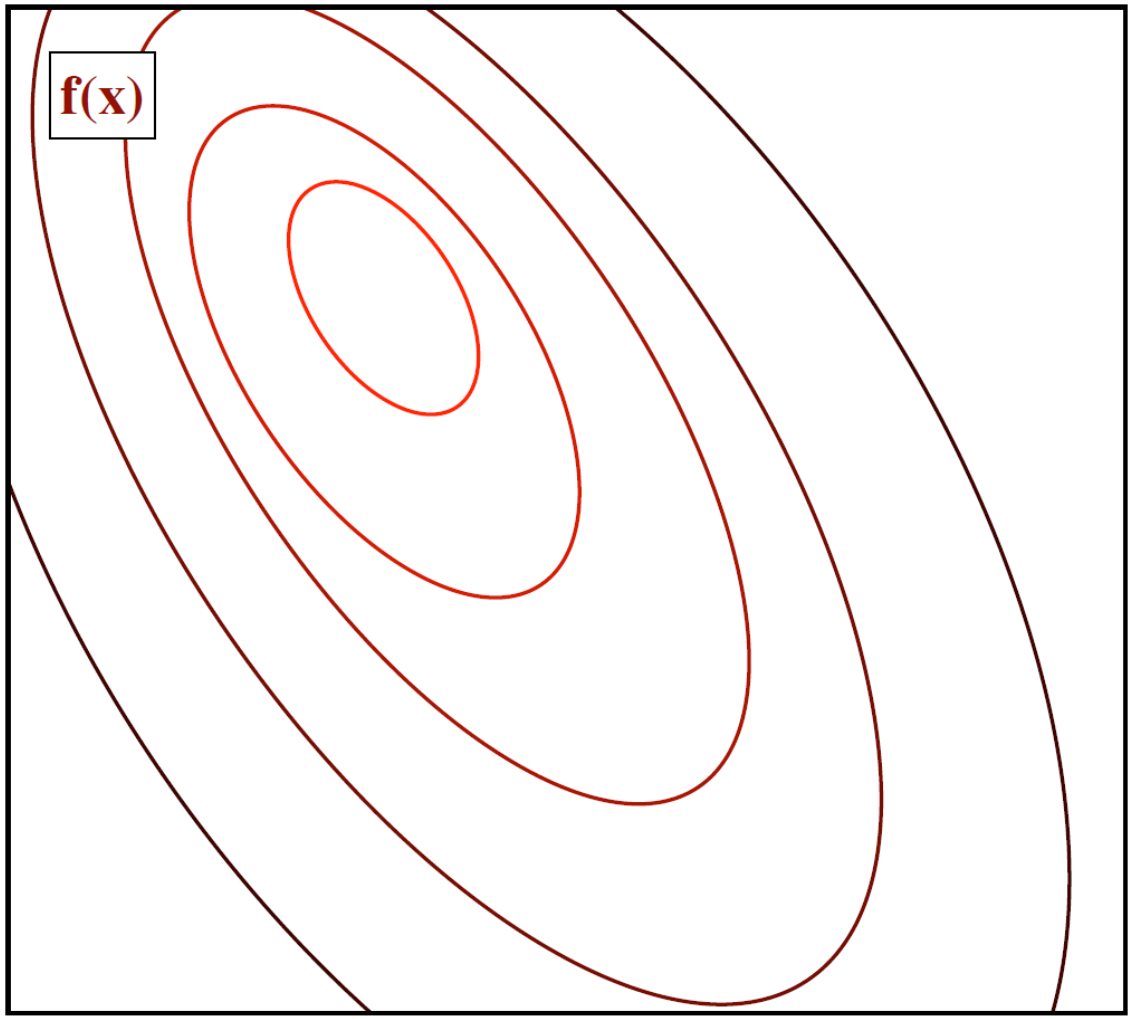
$$\mathbf{w}^{t+1} = \arg\min_{\mathbf{w}' \in \mathcal{C}} \{f(\mathbf{w}^t) + \nabla f(\mathbf{w}^t)^\top (\mathbf{w}' - \mathbf{w}^t) + \frac{1}{2\eta} \|\mathbf{w}' - \mathbf{w}^t\|^2\}$$

- Equivalent to projection of gradient descent:

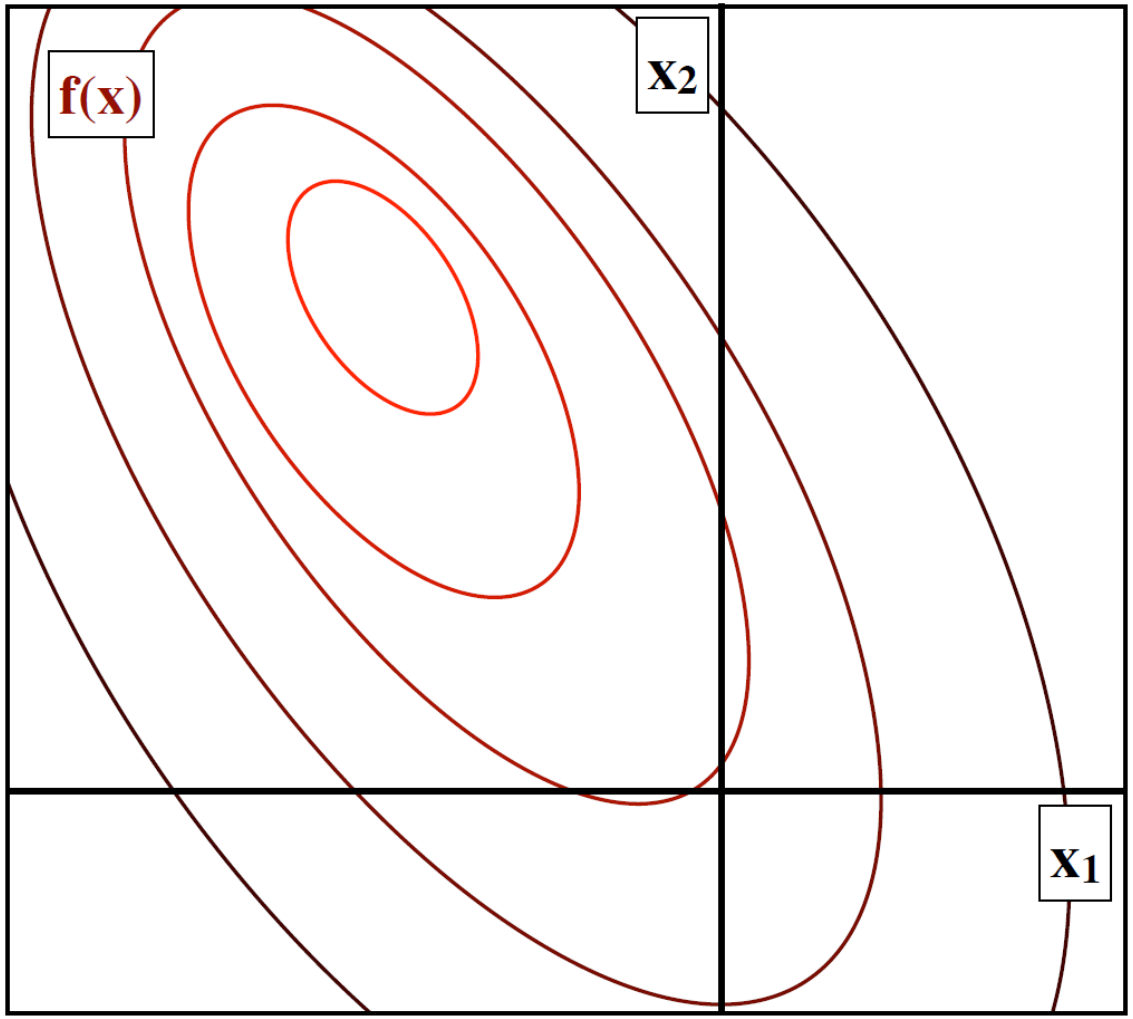
$$\mathbf{w}_t^{GD} = \mathbf{w}^t - \eta^t \nabla f(\mathbf{w}^t)$$

$$\mathbf{w}^{t+1} = \arg\min_{\mathbf{w}' \in \mathcal{C}} \|\mathbf{w}' - \mathbf{w}_t^{GD}\|$$

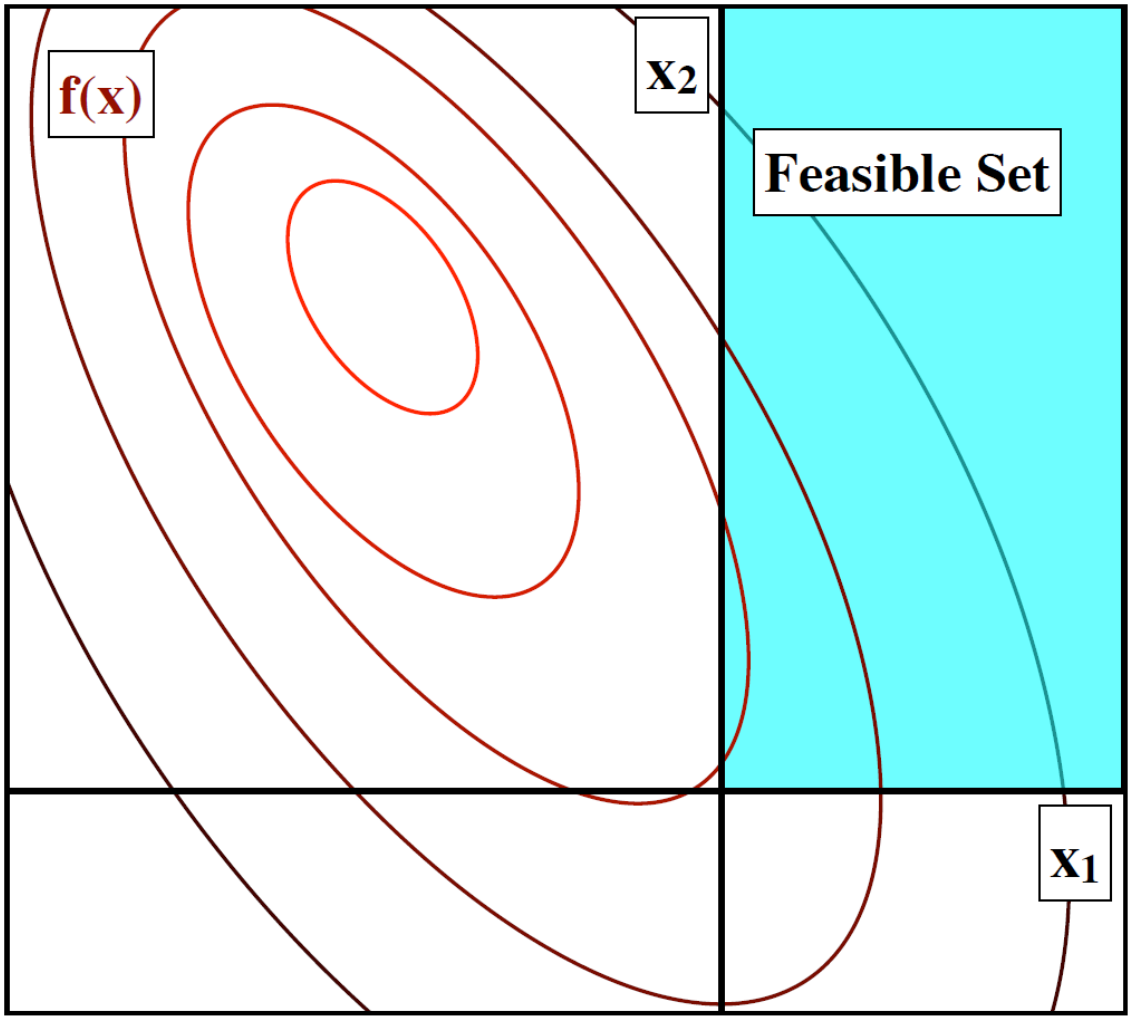
Gradient Projection



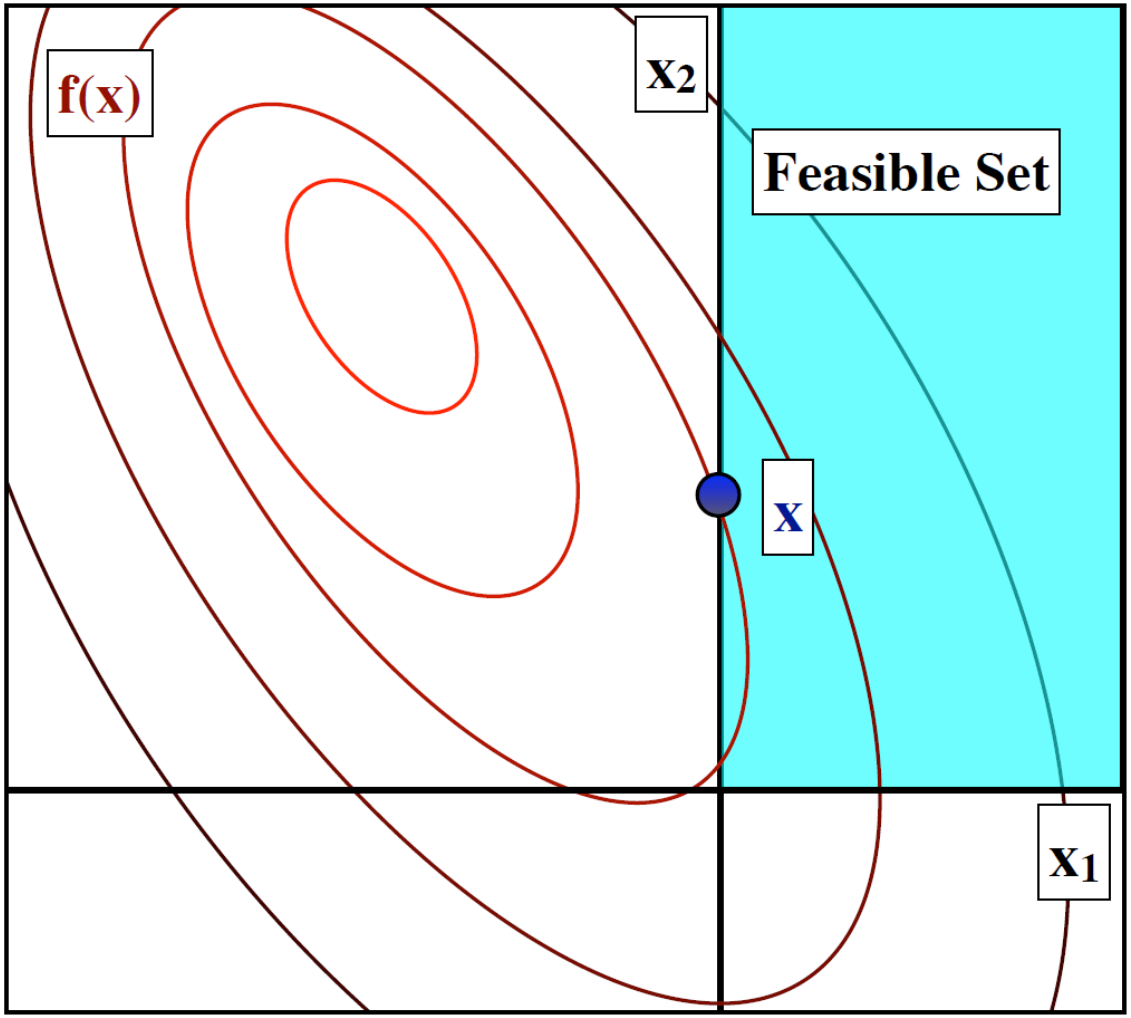
Gradient Projection



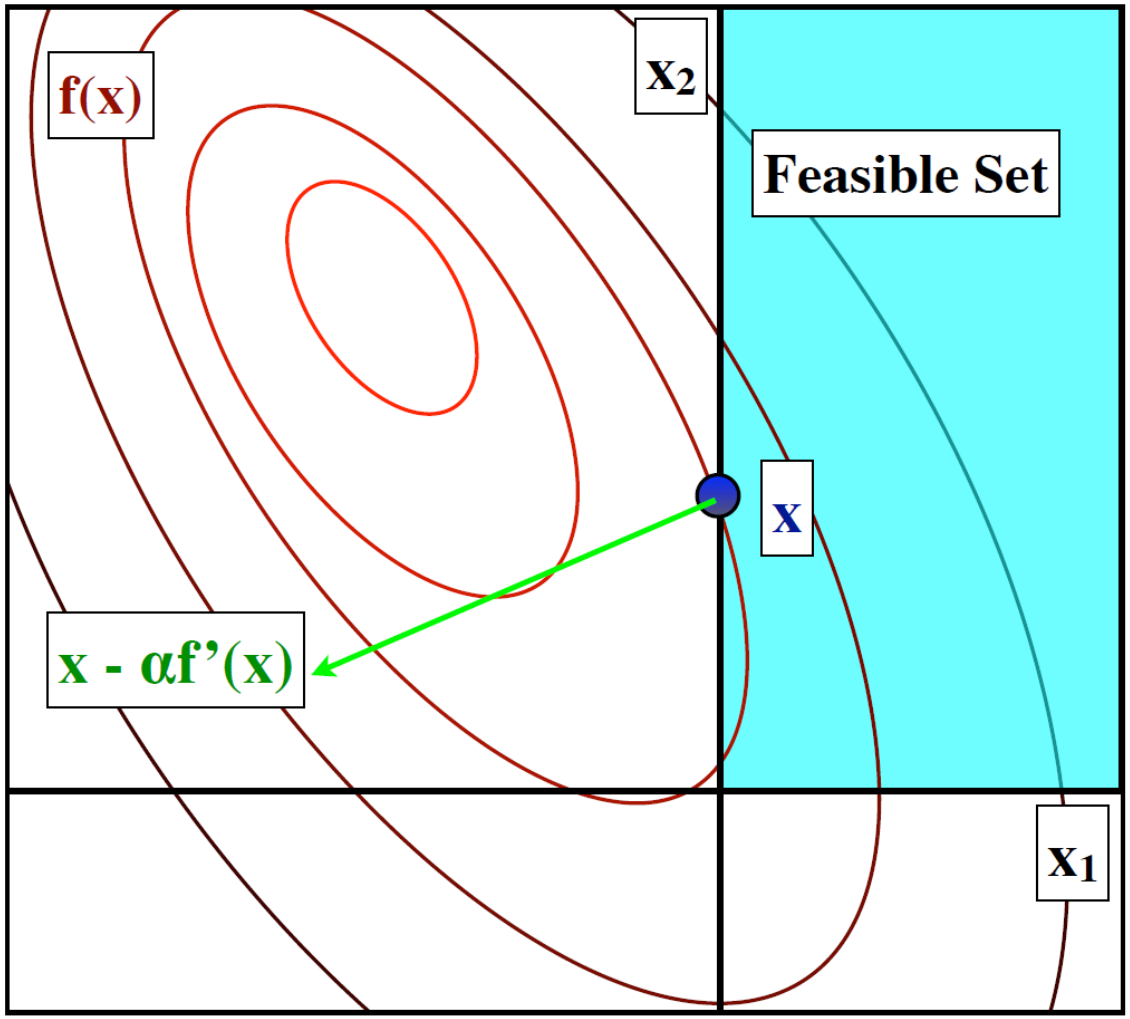
Gradient Projection



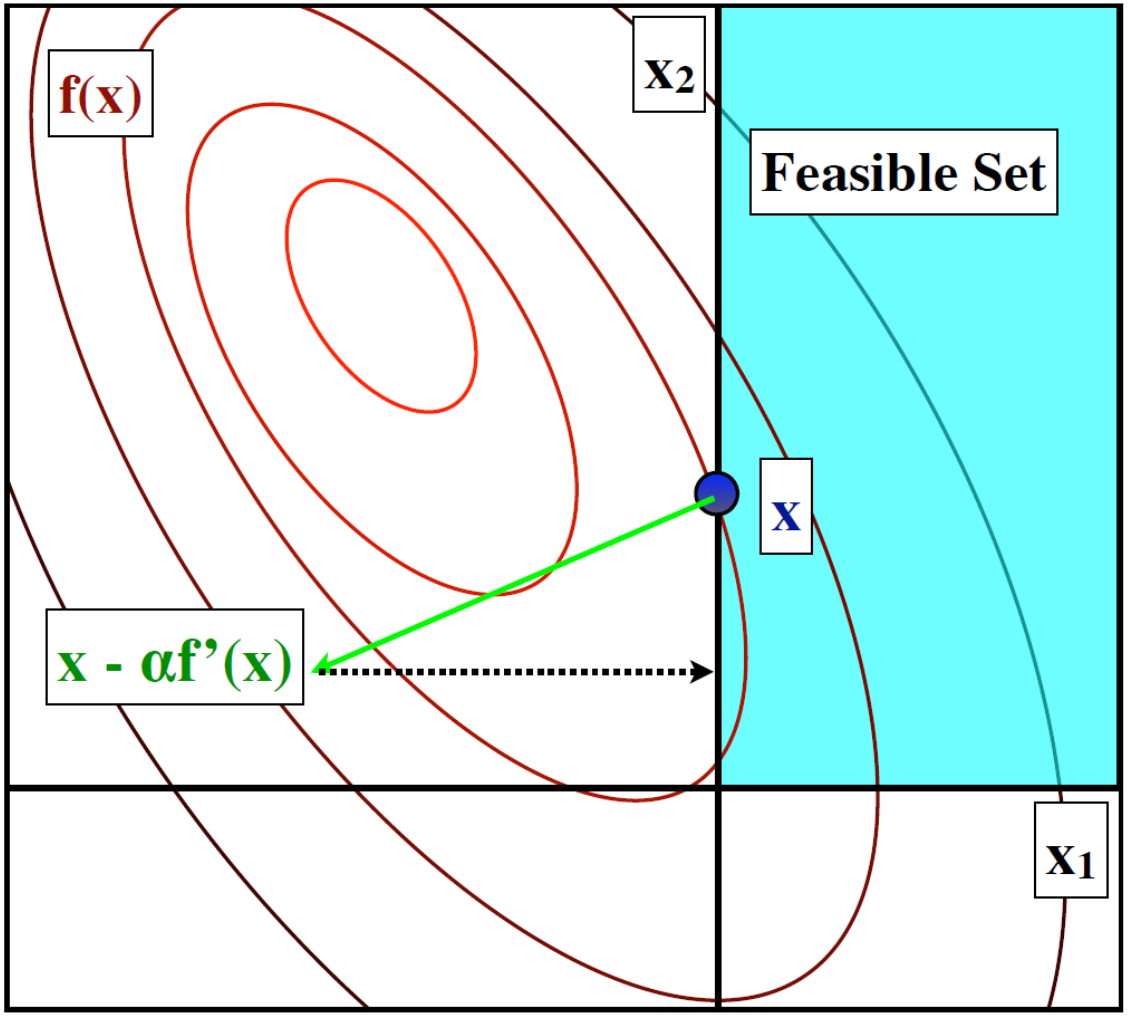
Gradient Projection



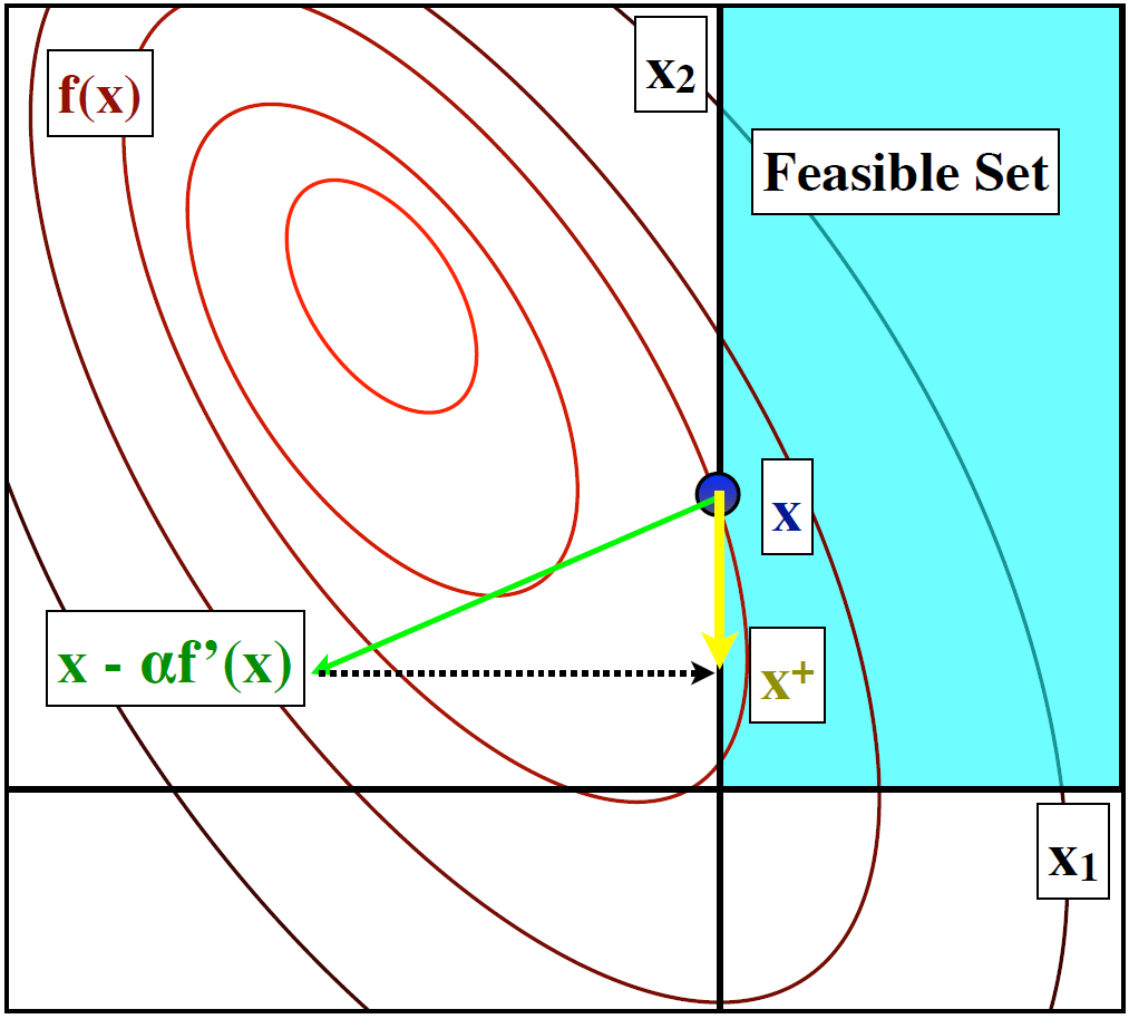
Gradient Projection



Gradient Projection



Gradient Projection



Discussion of Projected Gradient

- Projected gradient has same rate as gradient method!
- Can do many of the same tricks (i.e. line-search, acceleration, Barzilai-Borwein, SAG, SVRG)
- For projected Newton, you need to do an expensive projection under $\|\cdot\|_{\nabla^2 f(\mathbf{w}^t)}$

Proximal-Gradient Method

- A generalization of projected-gradient is Proximal-gradient
- The proximal-gradient method addresses problem of the form

$$\min_{\mathbf{w}} f(\mathbf{w}) + r(\mathbf{w})$$

where f is smooth but r is a general convex function

- Applies proximity operator of r to gradient descent on f:

$$\mathbf{w}_t^{GD} = \mathbf{w}^t - \eta^t \nabla f(\mathbf{w}^t)$$

$$\mathbf{w}^{t+1} = \arg\min_{\mathbf{w}'} \|\mathbf{w}' - \mathbf{w}_t^{GD}\|^2 + \eta r(\mathbf{w}')$$

- Equivalent to using the approximation

$$\mathbf{w}^{t+1} = \arg\min_{\mathbf{w}'} \{f(\mathbf{w}^t) + \nabla f(\mathbf{w}^t)^\top (\mathbf{w}' - \mathbf{w}^t) + \frac{1}{2\eta}\|\mathbf{w}' - \mathbf{w}^t\|^2 + r(\mathbf{w}')\}$$

- Convergence rates are still the same as for minimizing f

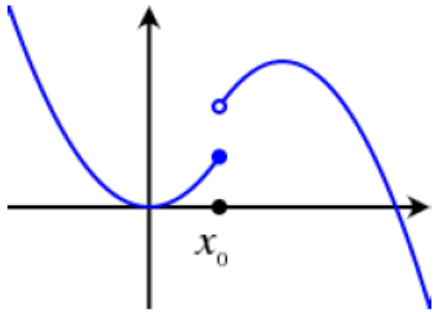
Proximal-Gradient Method

If L is known, we can do following update:

$$\begin{aligned}\mathbf{w}^{t+1} &= \arg\min_{\mathbf{w}'} \left\{ f(\mathbf{w}^t) + \nabla f(\mathbf{w}^t)^\top (\mathbf{w}' - \mathbf{w}^t) + \frac{L}{2}\|\mathbf{w}' - \mathbf{w}^t\|^2 + r(\mathbf{w}') \right\} \\ &= \arg\min_{\mathbf{w}'} \left\{ \left\| \mathbf{w}' - \left(\mathbf{w}^t - \frac{L}{2}\nabla f(\mathbf{w}^t)\right) \right\|_2^2 + \frac{1}{L} r(\mathbf{w}') \right\} \\ &= \text{Prox}_{\frac{1}{L}r}\left(\mathbf{w}^t - \frac{L}{2}\nabla f(\mathbf{w}^t)\right)\end{aligned}$$

If L is unknown, use line-search (Beck and Teboulle (2009))

- Most update to date assumptions
  - $f(\mathbf{w})$ is Lipschitz smooth, i.e., $\|\nabla f(\mathbf{w}) - \nabla f(\mathbf{w}')\|_2 \leq \|\mathbf{w} - \mathbf{w}'\|_2$ or the gradient is Lipschitz-continuous $\nabla^2 f(\mathbf{w}) \preceq LI$
  - $r(\mathbf{w})$ is lower semi-continuous (see the right figure)
  - $f(\mathbf{w}) + r(\mathbf{w})$ is bounded from below, i.e., $\inf f(\mathbf{w}) + r(\mathbf{w}) > -\infty$



Proximal Operator, Iterative Soft Thresholding

- For ℓ_1-regularization $r(\mathbf{w}) = \lambda \sum_i |\mathbf{w}_i|$, we obtain iterative soft-thresholding:

$$\mathbf{w}^{t+1} = \text{Prox}_{\eta,\lambda}[\mathbf{w}^t - \eta \nabla f(\mathbf{w}^t)] = \text{softThresh}_{\eta,\lambda}[\mathbf{w}^t - \eta \nabla f(\mathbf{w}^t)]$$

where

$$\text{softThresh}_\lambda(\mathbf{w}) = (\mathbf{w} - \lambda)_+ - (-\mathbf{w} - \lambda)_+ = \begin{cases} \mathbf{w}_i - \lambda & \mathbf{w}_i \geq \lambda \\ 0 & |\mathbf{w}_i| \leq \lambda \\ \mathbf{w}_i + \lambda & \mathbf{w}_i \leq -\lambda \end{cases}$$

Example (Example with $\lambda = 1$)

Input	Threshold	Soft-Threshold
$\begin{pmatrix} 0.6715 \\ -1.2075 \\ 0.7172 \\ 1.6302 \\ 0.4889 \end{pmatrix}$	$\begin{pmatrix} 0 \\ -1.2075 \\ 0 \\ 1.6302 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ -0.2075 \\ 0 \\ 0.6302 \\ 0 \end{pmatrix}$

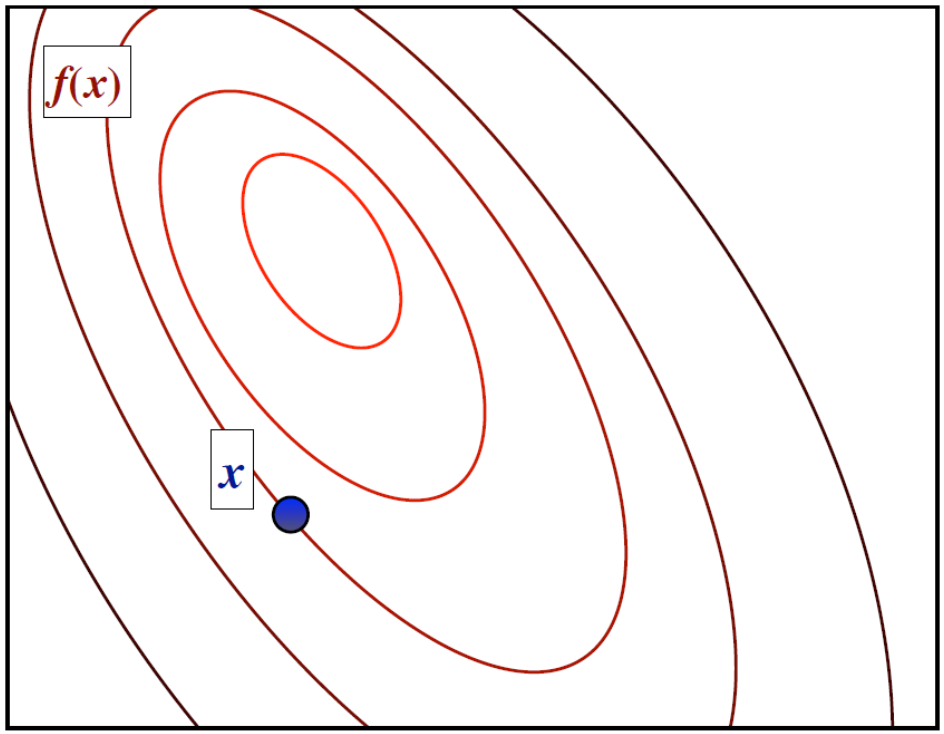
Special case of Projected-Gradient Methods

- Projected-gradient methods are another special case:

$$r(\mathbf{w}) = \begin{cases} 0 & \text{if } \mathbf{w} \in \mathcal{C} \\ \infty & \text{if } \mathbf{w} \notin \mathcal{C} \end{cases}$$

gives

$$\mathbf{w}^{t+1} = \text{Project}_{\mathcal{C}}[\mathbf{w}^t - \eta \nabla f(\mathbf{w}^t)]$$



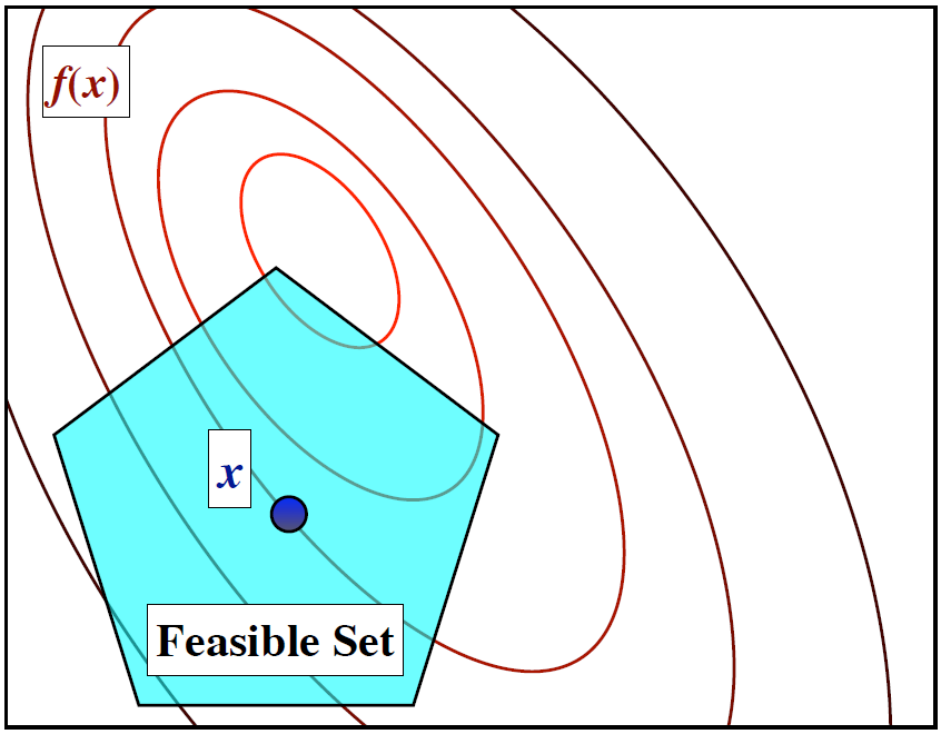
Special case of Projected-Gradient Methods

- Projected-gradient methods are another special case:

$$r(\mathbf{w}) = \begin{cases} 0 & \text{if } \mathbf{w} \in \mathcal{C} \\ \infty & \text{if } \mathbf{w} \notin \mathcal{C} \end{cases}$$

gives

$$\mathbf{w}^{t+1} = \text{Project}_{\mathcal{C}}[\mathbf{w}^t - \eta \nabla f(\mathbf{w}^t)]$$



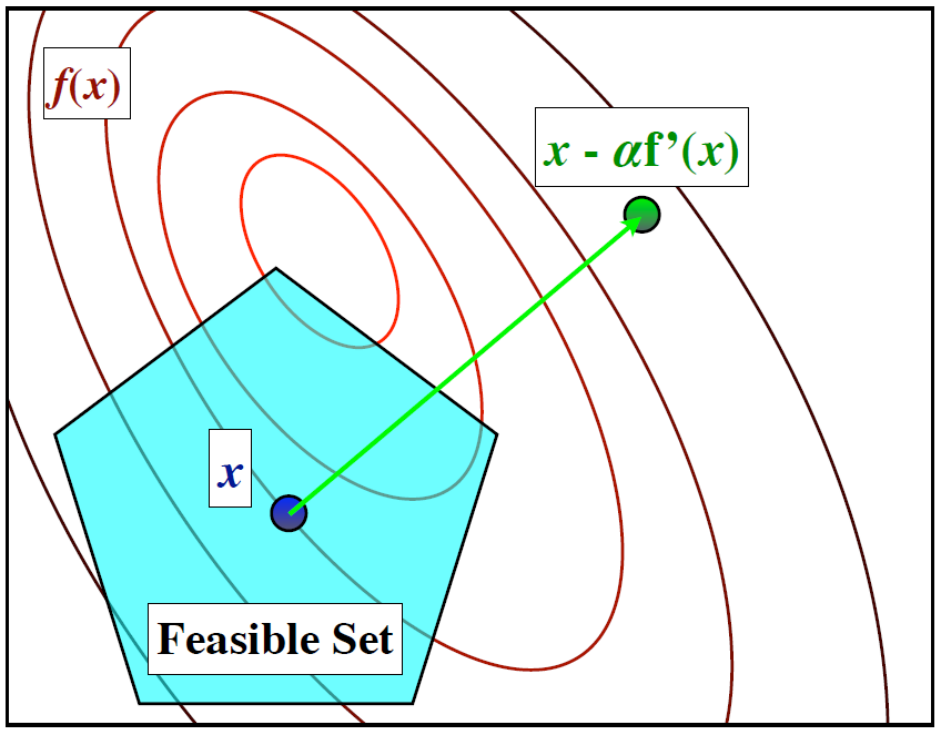
Special case of Projected-Gradient Methods

- Projected-gradient methods are another special case:

$$r(\mathbf{w}) = \begin{cases} 0 & \text{if } \mathbf{w} \in \mathcal{C} \\ \infty & \text{if } \mathbf{w} \notin \mathcal{C} \end{cases}$$

gives

$$\mathbf{w}^{t+1} = \text{Project}_{\mathcal{C}}[\mathbf{w}^t - \eta \nabla f(\mathbf{w}^t)]$$



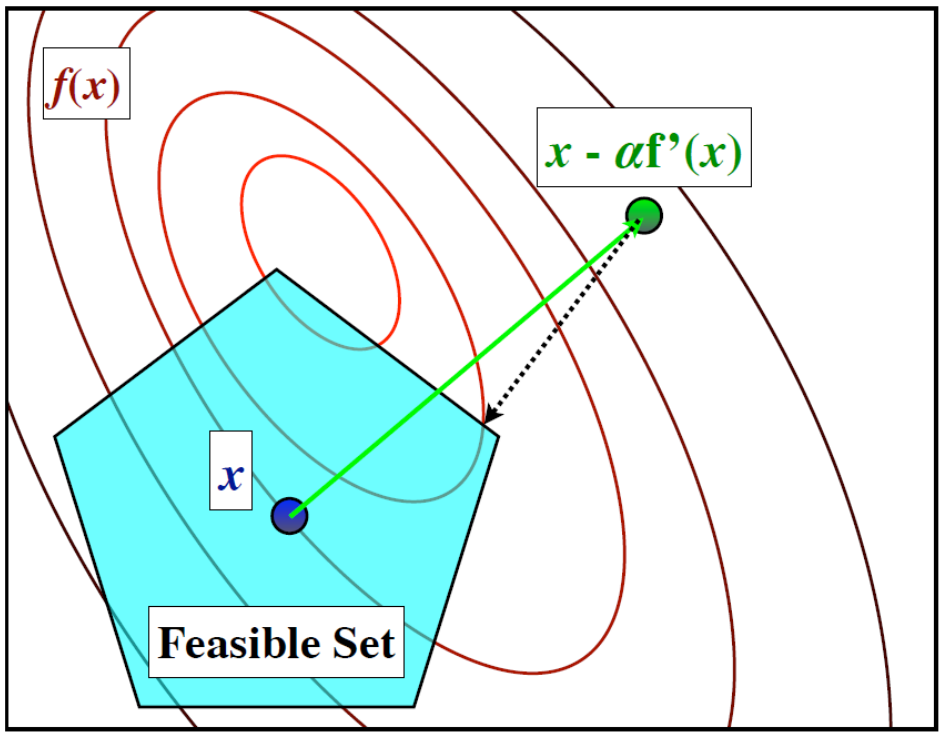
Special case of Projected-Gradient Methods

- Projected-gradient methods are another special case:

$$r(\mathbf{w}) = \begin{cases} 0 & \text{if } \mathbf{w} \in \mathcal{C} \\ \infty & \text{if } \mathbf{w} \notin \mathcal{C} \end{cases}$$

gives

$$\mathbf{w}^{t+1} = \text{Project}_{\mathcal{C}}[\mathbf{w}^t - \eta \nabla f(\mathbf{w}^t)]$$



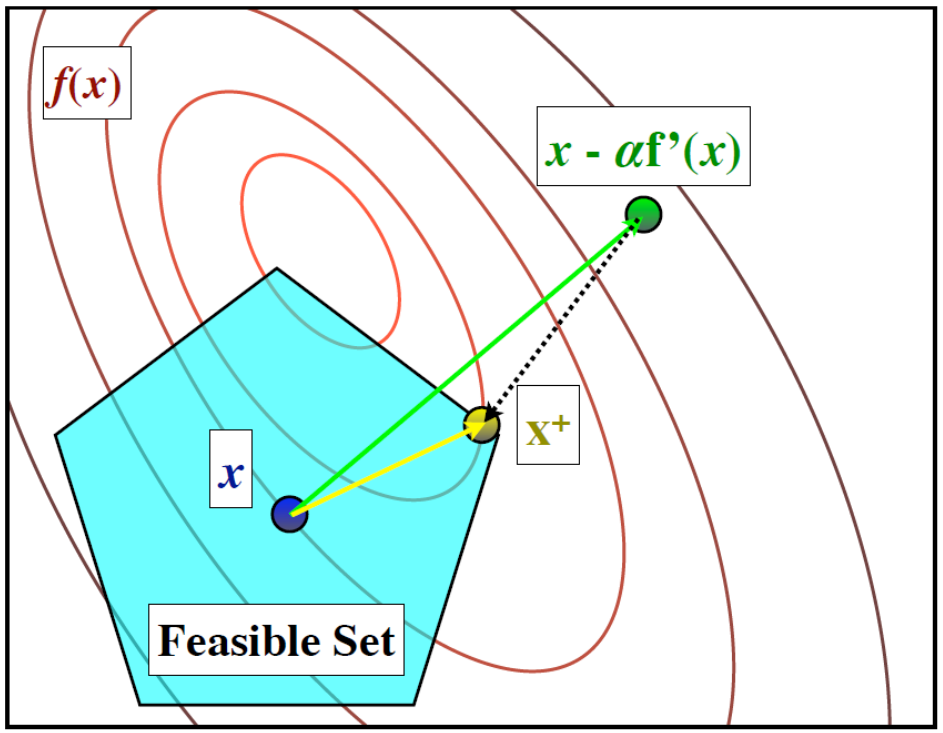
Special case of Projected-Gradient Methods

- Projected-gradient methods are another special case:

$$r(\mathbf{w}) = \begin{cases} 0 & \text{if } \mathbf{w} \in \mathcal{C} \\ \infty & \text{if } \mathbf{w} \notin \mathcal{C} \end{cases}$$

gives

$$\mathbf{w}^{t+1} = \text{Project}_{\mathcal{C}}[\mathbf{w}^t - \eta \nabla f(\mathbf{w}^t)]$$



Exact Proximal-Gradient Methods

- For what problems can we apply these methods?
- We can efficiently compute the proximity operator for:
  - ℓ_1-Regularization
  - Group ℓ_1-Regularization
  - Lower and upper bounds
  - Small number of linear constraint
  - Probability constraints
  - A few other simple regularizers/constraints
- Can solve these non-smooth/constrained problems as fast as smooth/unconstrained problems!
- We can again do many of the same tricks (line-search, acceleration, Barzilai-Borwein, two-metric projection, inexact proximal operators, SAG, SVRG)
  - e.g., Accelerated Proximal-Gradient Algorithm (Li and Lin (2015))

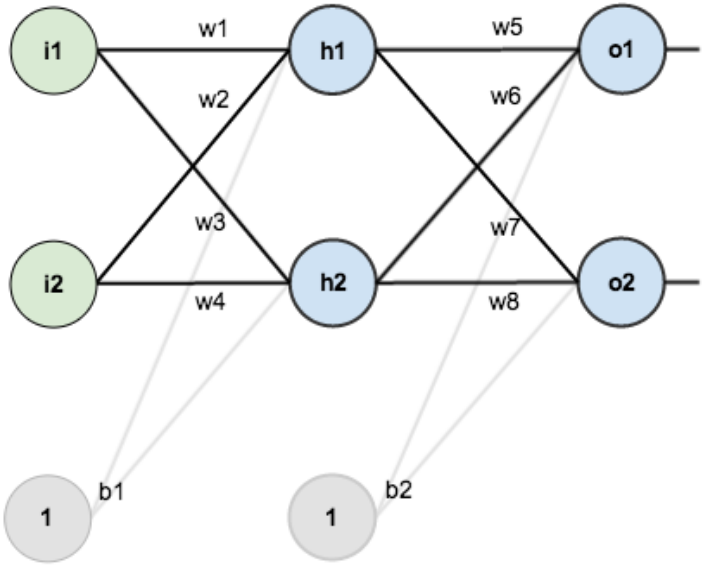
Summary

- Convex functions have special properties that allow us to efficiently minimize them
- Gradient-based methods allow elegant scaling with dimensionality of problem
- Stochastic-gradient methods allow scaling with number of training examples, at cost of slower convergence rate
- For finite datasets, SAG fixes convergence rate of stochastic gradient methods, and SVRG fixes memory problem of SAG
- These building blocks can be extended to solve a variety of constrained and non-smooth problems

Overview

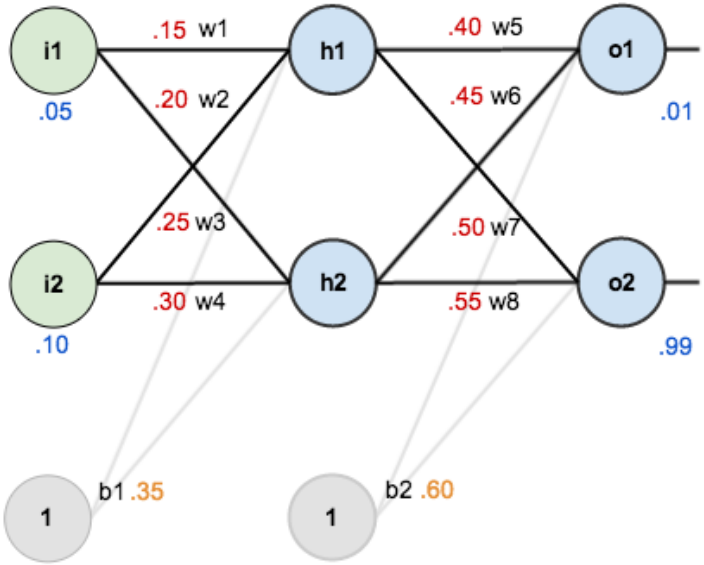
Neural Network: A Running Example

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/



Neural Network: A Running Example

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/



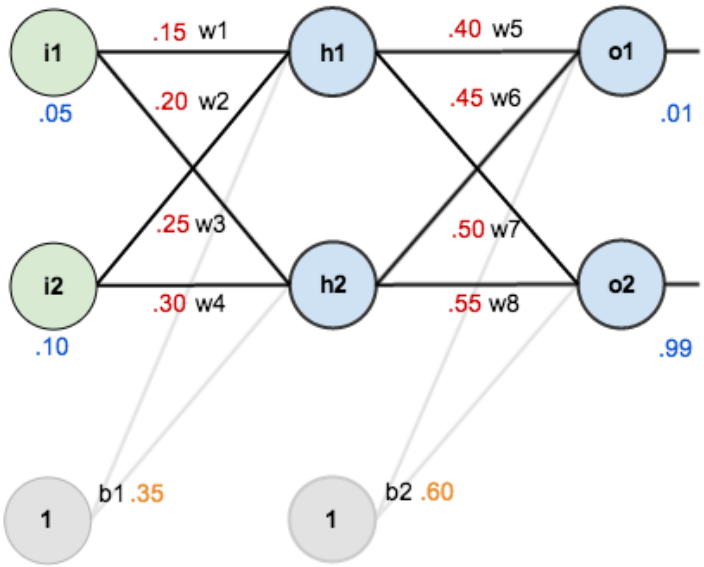
In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs

The forward pass:

- $net_{h_1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1 = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$
- $out_{h_1} = \frac{1}{1+e^{-net_{h_1}}} = \frac{1}{1+e^{-0.3775}} = 0.5933$
- Similarly, $out_{h_2} = 0.5969$
- $net_{o_1} = w_5 * out_{h_1} + w_6 * out_{h_2} + b_2 * 1 = 0.4 * 0.5933 + 0.45 * 0.5969 + 0.6 * 1 = 1.1059$
- $out_{o_1} = \frac{1}{1+e^{-net_{o_1}}} = \frac{1}{1+e^{-1.1059}} = 0.7514$
- Similarly, $out_{o_2} = 0.7729$

Neural Network: A Running Example

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/



In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs

Recall:

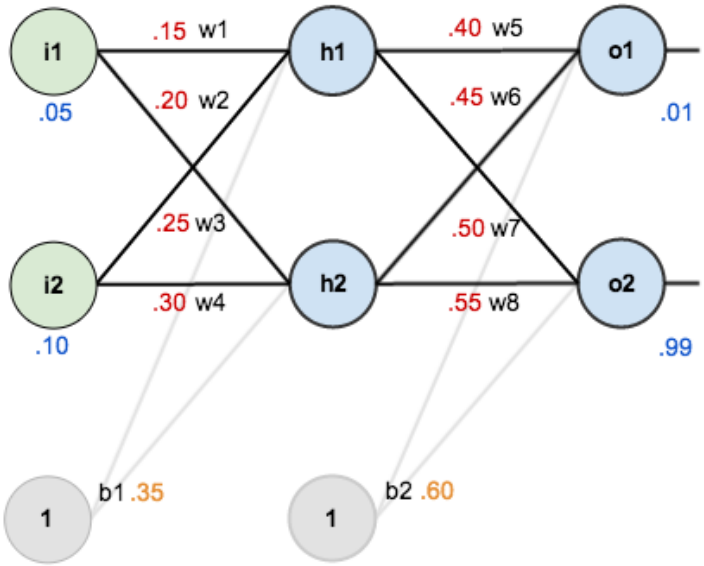
- $out_{o_1} = \frac{1}{1+e^{-net_{o_1}}} = \frac{1}{1+e^{-1.1059}} = 0.7514$
- Similarly, $out_{o_2} = 0.7729$

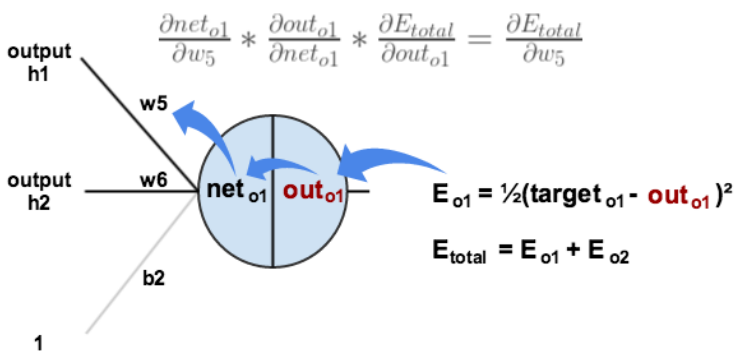
Consider square loss as total error
$E_{total} = \sum_i \frac{1}{2}(target - output)^2$

- $E_{o1} = \frac{1}{2}(target_{o1} - output_{o1})^2 = \frac{1}{2}(0.01 - 0.7514)^2 = 0.2748$
- Similarly $E_{o2} = 0.0236$
- $E_{total} = 0.2748 + 0.0236 = 0.2984$

Neural Network: A Running Example

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/





In the backward pass, we consider the chain rule for w_5:

- $\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} \frac{\partial out_{o1}}{\partial net_{o1}} \frac{\partial net_{o1}}{\partial w_5}$

Compute $\frac{\partial E_{total}}{\partial out_{o1}}$:

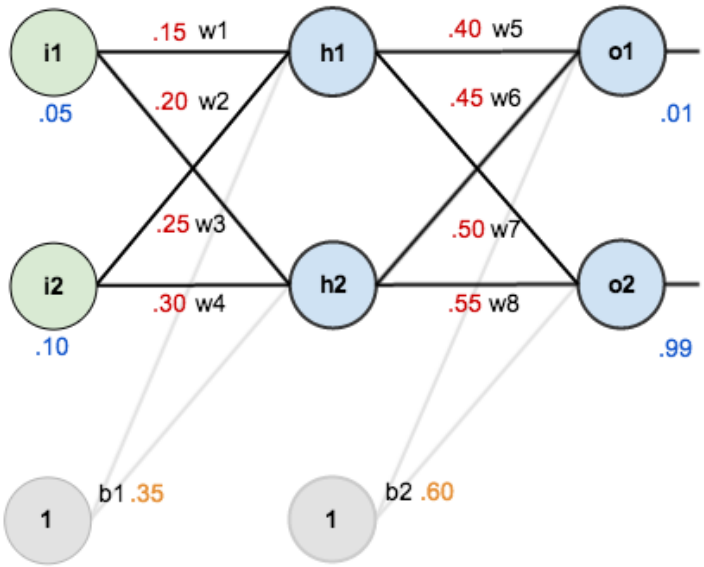
- $E_{total} = \frac{1}{2}(target_{o1} - output_{o1})^2 + \frac{1}{2}(target_{o2} - output_{o2})^2$
- $\frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{o_1}}{\partial out_{o1}} = (output_{o1} - target_{o1}) = 0.7514 - 0.01 = 0.7414$

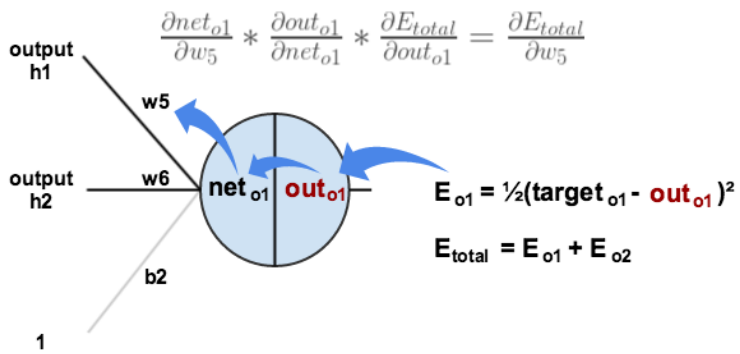
Compute $\frac{\partial out_{o1}}{\partial net_{o1}}$:

- $out_{o1} = \frac{1}{1+e^{-net_{o1}}} \doteq \sigma(net_{o_1})$
- $\frac{\partial out_{o1}}{\partial net_{o_1}} = \frac{e^{-net_{o1}}}{(1+e^{-net_{o1}})^2} = (1 - \sigma(net_{o_1}))\sigma(net_{o_1}) = 0.7514(1 - 0.7514) = 0.1868$

Neural Network: A Running Example

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/





Compute $\frac{\partial net_{o1}}{\partial w_5}$:

- $net_{o1} = w_5 * out_{h_1} + w6 * out_{h_2} + b_2 * 1$
- $\frac{\partial net_{o1}}{\partial w_5} = out_{h_1} = 0.5933$

Put all together:

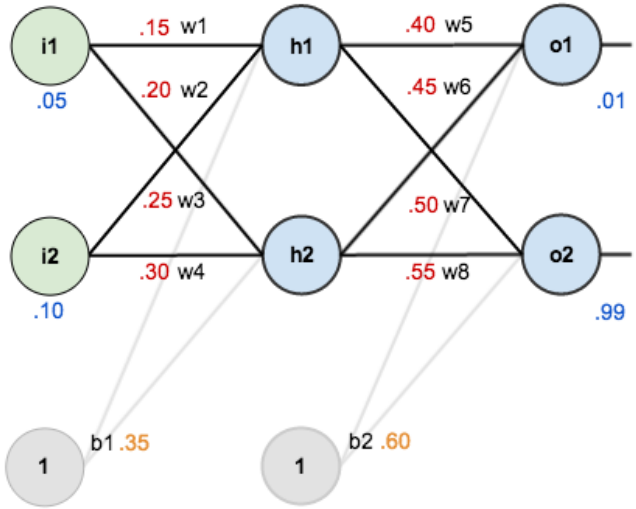
- $\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} \frac{\partial out_{o1}}{\partial net_{o1}} \frac{\partial net_{o1}}{\partial w_5} = 0.7414 * 0.1868 * 0.5932 = 0.0822$

We update w_5 as:

- $w_5' = w_5 - \eta \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.0822 = 0.3589$

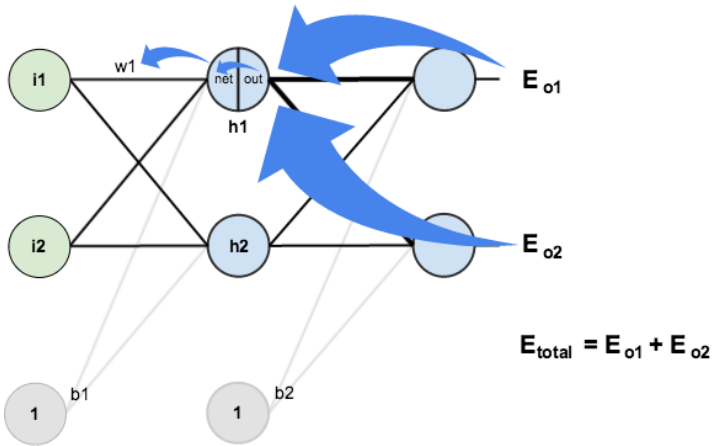
Neural Network: A Running Example

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/



$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

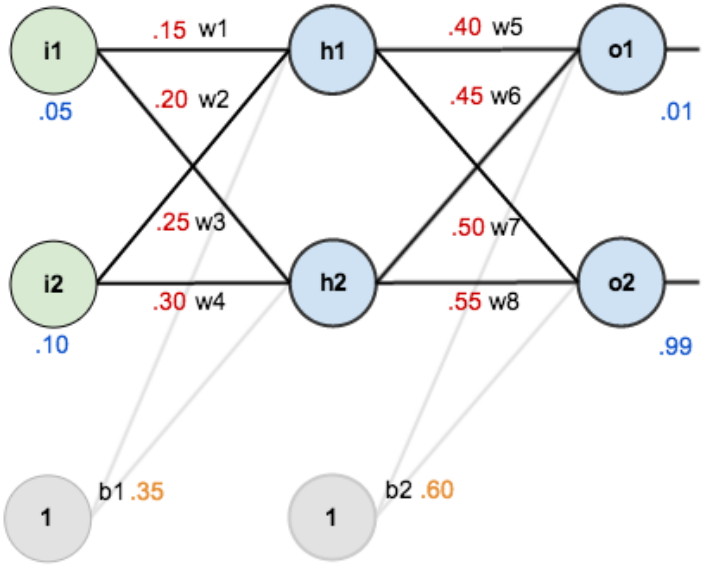


The chain rule for w_1:

- $\frac{\partial E_{total}}{\partial w_1} = (\frac{\partial E_{o_1}}{\partial out_{h1}} + \frac{\partial E_{o_2}}{\partial out_{h1}}) \frac{\partial out_{h1}}{\partial net_{h1}} \frac{\partial net_{h1}}{\partial w_1}$
  where
  $\frac{\partial E_{o_1}}{\partial out_{h1}} = \frac{\partial E_{o_1}}{\partial out_{o1}} \frac{\partial out_{o1}}{\partial net_{o1}} \frac{\partial net_{o1}}{\partial out_{h1}}$
  and
  $\frac{\partial E_{o_2}}{\partial out_{h1}} = \frac{\partial E_{o_2}}{\partial out_{o2}} \frac{\partial out_{o2}}{\partial net_{o2}} \frac{\partial net_{o2}}{\partial out_{h1}}$
- Note that here $\frac{\partial E_{o_1}}{\partial out_{o1}} \frac{\partial out_{o1}}{\partial net_{o1}}$ have been computed before and can be reused
- After simple calculation as before, we have $\frac{\partial E_{total}}{\partial w_1} = 0.0004$
- $w_1' = w_1 - \eta \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.0004 = 0.1498$

Neural Network: A Running Example

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/



Similarly we can compute all the weights:

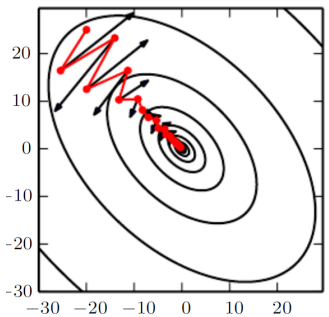
- $w_1' = 0.1498$
- $w_2' = 0.1996$
- $w_3' = 0.2498$
- $w_4' = 0.2995$
- $w_5' = 0.3589$
- $w_6' = 0.4087$
- $w_7' = 0.5113$
- $w_8' = 0.5614$

The total error is then: 0.2910 ($<$0.2984 as initial)

http://playground.tensorflow.org/

Neural Network Optimization: General Ideas

- We can in general use first-order stochastic gradient descent methods for neural network optimization
  - $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{1}{|\mathcal{B}^t|} \sum_{i \in \mathcal{B}^t}^{|\mathcal{B}^t|} \nabla f_i(\mathbf{w}^t)$
  - Usually called mini-batch in neural network context
- Momentum update (Polyak (1964); Goodfellow et al. (2016)):
  - Improve the condition of Hessian matrix and reduce variance in the stochastic gradient
  - $\mathbf{v}^{t+1} = \alpha \mathbf{v}^t - \eta \frac{1}{|\mathcal{B}^t|} \sum_{i \in \mathcal{B}^t}^{|\mathcal{B}^t|} \nabla f_i(\mathbf{w}^t)$
  - $\mathbf{w}^{t+1} = \mathbf{w}^t + \mathbf{v}^{t+1}$



  - $\alpha = 0.9$ corresponds to multiplying the maximum speed by 10 relative to the gradient descent algorithm

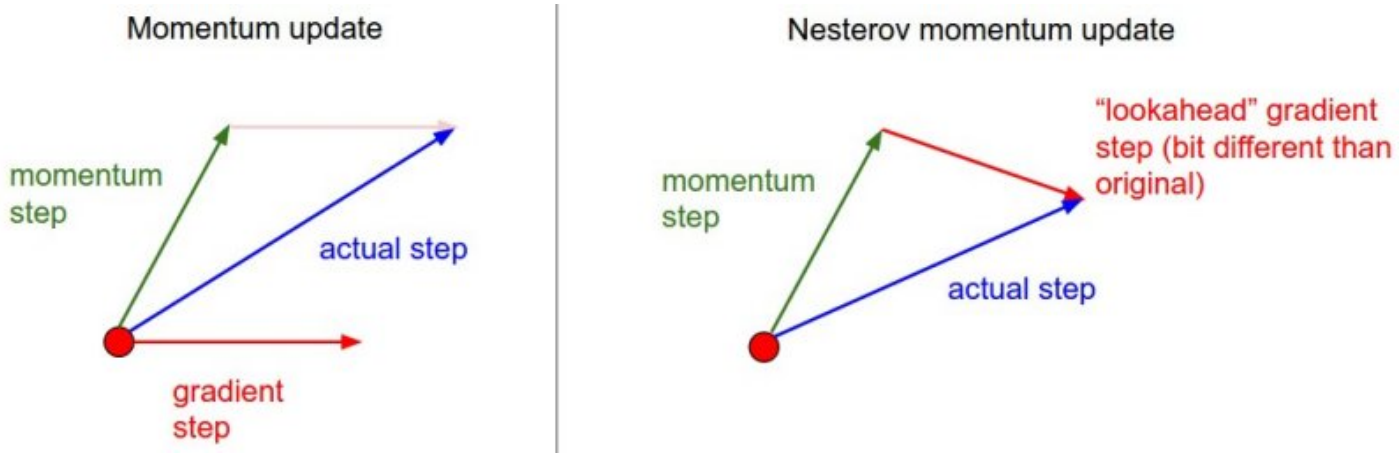
`http://www.deeplearningbook.org/contents/optimization.html`

Neural Network Optimization: General Ideas (Cont'd)

- Momentum update (Goodfellow et al. (2016)):
  - $\mathbf{v}^{t+1} = \alpha \mathbf{v}^t - \eta \frac{1}{|\mathcal{B}^t|} \sum_{i \in \mathcal{B}^t}^{|\mathcal{B}^t|} \nabla f_i(\mathbf{w}^t)$
  - $\mathbf{w}^{t+1} = \mathbf{w}^t + \mathbf{v}^{t+1}$
- Nesterov Momentum
  - Sutskever et al. (2013) introduced a variant of the momentum algorithm that was inspired by Nesterovs accelerated gradient method
    - $\mathbf{v}^{t+1} = \alpha \mathbf{v}^t - \eta \frac{1}{|\mathcal{B}^t|} \sum_{i \in \mathcal{B}^t}^{|\mathcal{B}^t|} \nabla f_i(\mathbf{w}^t + \alpha \mathbf{v}^t)$
    - $\mathbf{w}^{t+1} = \mathbf{w}^t + \mathbf{v}^{t+1}$
  - Unfortunately, in the stochastic gradient case, Nesterov momentum does not improve the rate of convergence

Neural Network Optimization: General Ideas (Cont'd)

- The difference between Nesterov momentum and standard momentum is where the gradient is evaluated
- Momentum update (Goodfellow et al. (2016)):
  - $\mathbf{v}^{t+1} = \alpha \mathbf{v}^t - \eta \frac{1}{|\mathcal{B}^t|} \sum_{i \in \mathcal{B}^t}^{|\mathcal{B}^t|} \nabla f_i(\mathbf{w}^t)$
  - $\mathbf{w}^{t+1} = \mathbf{w}^t + \mathbf{v}^{t+1}$
- Nesterov Momentum
  - $\mathbf{v}^{t+1} = \alpha \mathbf{v}^t - \eta \frac{1}{|\mathcal{B}^t|} \sum_{i \in \mathcal{B}^t}^{|\mathcal{B}^t|} \nabla f_i(\mathbf{w}^t + \alpha \mathbf{v}^t)$
  - $\mathbf{w}^{t+1} = \mathbf{w}^t + \mathbf{v}^{t+1}$

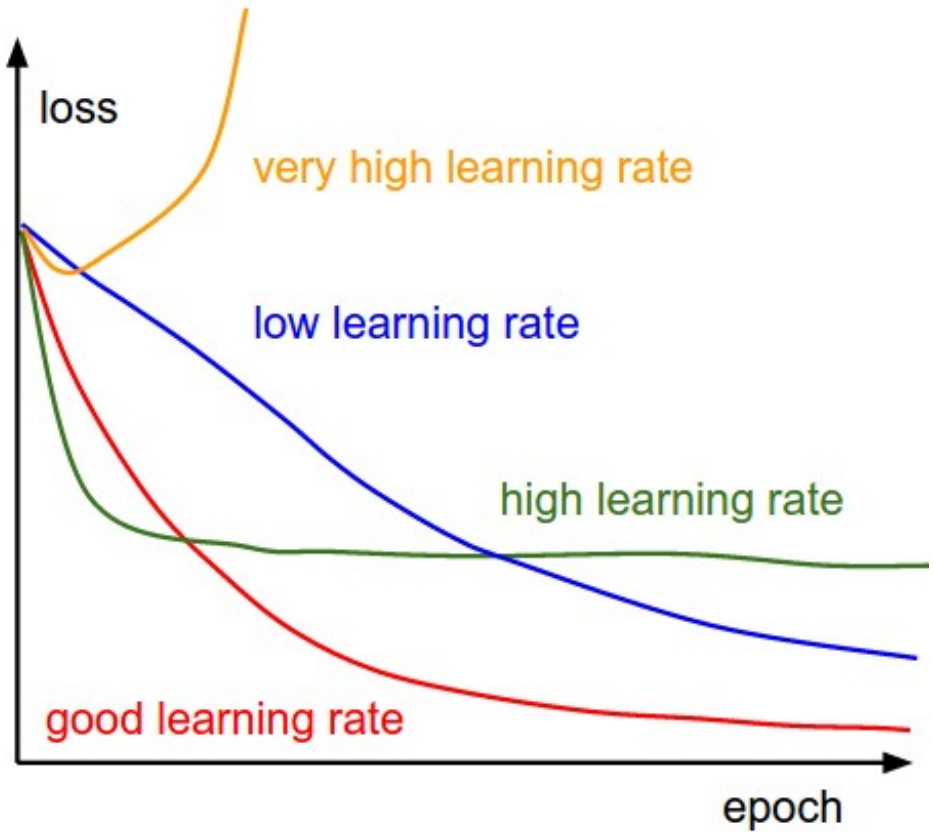


http://cs231n.github.io/neural-networks-3/

Neural Network Optimization: General Ideas (Cont'd)

- Momentum update (Goodfellow et al. (2016)):
  - $\mathbf{v}^{t+1} = \alpha \mathbf{v}^t - \eta \frac{1}{|\mathcal{B}^t|} \sum_{i \in \mathcal{B}^t}^{|\mathcal{B}^t|} \nabla f_i(\mathbf{w}^t)$
  - $\mathbf{w}^{t+1} = \mathbf{w}^t + \mathbf{v}^{t+1}$
- Nesterov Momentum (Sutskever et al. (2013))
  - $\mathbf{v}^{t+1} = \alpha \mathbf{v}^t - \eta \frac{1}{|\mathcal{B}^t|} \sum_{i \in \mathcal{B}^t}^{|\mathcal{B}^t|} \nabla f_i(\mathbf{w}^t + \alpha \mathbf{v}^t)$
  - $\mathbf{w}^{t+1} = \mathbf{w}^t + \mathbf{v}^{t+1}$
- Original Nesterov's accelerated gradient method (Nesterov (1983)):
  - $\mathbf{w}^{t+1} = \mathbf{v}^t - \eta^t \nabla f(\mathbf{v}^t)$ (regular gradient step)
  - $\mathbf{v}^{t+1} = \mathbf{w}^t + \beta^t(\mathbf{w}^{t+1} - \mathbf{w}^t) = \beta^t \mathbf{w}^{t+1} + (1-\beta)\mathbf{w}^t$ ("slide" in dir. of $\mathbf{w}$)

More about Learning Rates



Higher learning rates will decay the loss faster, but they get stuck at worse values of loss (green line). This is because there is too much "energy" in the optimization and the parameters are bouncing around chaotically, unable to settle in a nice spot in the optimization landscape.

Options to Tune Learning Rate

Global tuning methods

- Step decay: Reduce the learning rate by some factor every few epochs (Depend heavily on the type of problem and the model)
- Exponential decay: $\eta^t = \eta_0 e^{-kt}$, where η_0 and k are hyper-parameters
- $1/t$ decay: $\eta^t = \eta_0/(1+kt)$, where η_0 and k are hyper-parameters

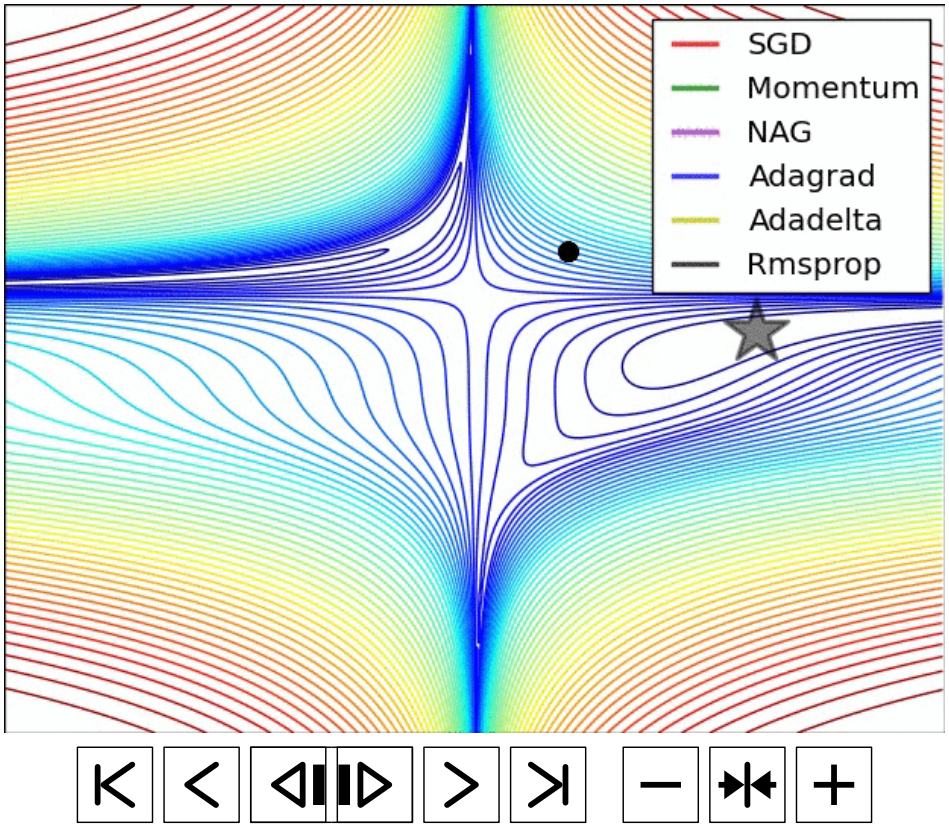
Options to Tune Learning Rate

Adaptive tuning (let $\mathbf{g}^t = \frac{1}{|\mathcal{B}^t|}\sum_{i\in\mathcal{B}^t}^{|\mathcal{B}^t|} \nabla f_i(\mathbf{w}^t)$)

- Adagrad (Duchi et al. (2011)) (larger partial derivative has a rapid decrease in learning rate)
  - $\mathbf{r}^{t+1} = \mathbf{r}^t + \mathbf{g}^t \odot \mathbf{g}^t$
  - $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{1}{\delta+\sqrt{\mathbf{r}^t}} \odot \mathbf{g}^t$ (Division and square root applied element-wise)
- RMSprop (Hinton (2012))
  - $\mathbf{r}^{t+1} = \rho\mathbf{r}^t + (1-\rho)\mathbf{g}^t \odot \mathbf{g}^t$ (exponentially weighted moving average)
  - $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{1}{\sqrt{\delta+\mathbf{r}^t}} \odot \mathbf{g}^t$
- Adam (Kingma and Ba (2014)) (adaptive moments)
  - $\mathbf{s}^{t+1} = \rho_1\mathbf{s}^t + (1-\rho_1)\mathbf{g}^t$ (gradient moments)
  - $\mathbf{r}^{t+1} = \rho_2\mathbf{r}^t + (1-\rho_2)\mathbf{g}^t \odot \mathbf{g}^t$ (partial derivative size moments)
  - $\hat{\mathbf{s}^{t+1}} = \mathbf{s}^{t+1}/(1-\rho_1^t)$ (ρ_1^t: ρ_1 to power t)
  - $\hat{\mathbf{r}^{t+1}} = \mathbf{r}^{t+1}/(1-\rho_2^t)$ (ρ_2^t: ρ_2 to power t)
  - $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{1}{\delta+\sqrt{\hat{\mathbf{r}^{t+1}}}} \odot \hat{\mathbf{s}^{t+1}}$
  - ρ_1 and ρ_2 are hyper-parameters (suggested defaults: 0.9 and 0.999 respectively)

Animations of the Learning Process Dynamics

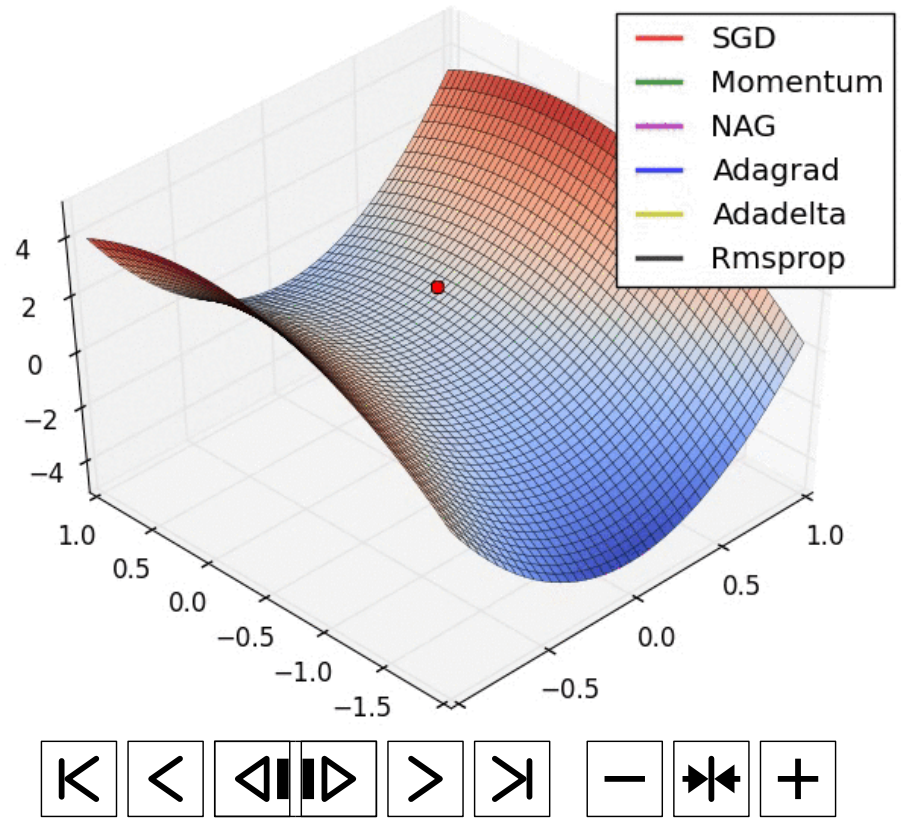
Images credit: Alec Radford



Contours of a loss surface and time evolution of different optimization algorithms. Notice the "overshooting" behavior of momentum-based methods, which make the optimization look like a ball rolling down the hill.

Animations of the Learning Process Dynamics

Images credit: Alec Radford



SGD has a very hard time breaking symmetry and gets stuck on the top. Conversely, algorithms such as RMSprop will see very low gradients in the saddle direction. Due to the denominator term in the RMSprop update, this will increase the effective learning rate along this direction, helping RMSProp proceed.

Further Reading

- Kingma and Ba (2014). Adam: A Method for Stochastic Optimization
- Reddi and Kumar (2018). On the Convergence of Adam and Beyond
- Smith et al. (2017). Don't Decay the Learning Rate, Increase the Batch Size

Brief Summary

- We have introduced stochastic gradient decent methods for
  - Convex optimization
  - Neural network optimization
- There is still less theoretical work on non-convex optimization, especially for deep learning
- But for most of the cases, people just try different adaptive SGD w/o momentum

References I

Beck, A. and Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sciences*, 2(1):183–202.

Duchi, J. C., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Hinton, G. (2012). Neural networks for machine learning. Technical report, University of Toronto.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Li, H. and Lin, Z. (2015). Accelerated proximal gradient methods for nonconvex programming. In *NIPS*, pages 379–387.

Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate o(1/k2). *Soviet Mathematics Doklady*, 27:372–376.

Nesterov, Y. (2005). Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152.

References II

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

Reddi, S. J. and Kumar, S. (2018). On the convergence of adam and beyond. In *ICLR*.

Smith, S. L., Kindermans, P., and Le, Q. V. (2017). Don't decay the learning rate, increase the batch size. *CoRR*, abs/1711.00489.

Sutskever, I., Martens, J., Dahl, G. E., and Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. In *ICML (3)*, volume 28, pages 1139–1147.