Machine Learning - MT 2017
9. Optimisation II

Christoph Haase

University of Oxford
October 27, 2017

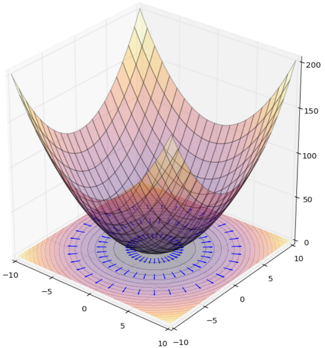
Calculus Background: Gradients

$$z = f(w_1, w_2) = \frac{w_1^2}{a^2} + \frac{w_2^2}{b^2}$$

$$\frac{\partial f}{\partial w_1} = \frac{2w_1}{a^2}$$

$$\frac{\partial f}{\partial w_2} = \frac{2w_2}{b^2}$$

$$\nabla_{\mathbf{w}} f = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \end{bmatrix} = \begin{bmatrix} \frac{2w_1}{a^2} \\ \frac{2w_2}{b^2} \end{bmatrix}$$



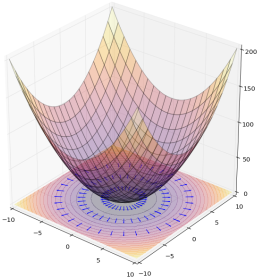
- Gradient vectors are orthogonal to contour curves
- Gradient points in the direction of steepest increase

Calculus Background: Hessians

$$z = f(w_1, w_2) = \frac{w_1^2}{a^2} + \frac{w_2^2}{b^2}$$

$$\nabla_{\mathbf{w}} f = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \end{bmatrix} = \begin{bmatrix} \frac{2w_1}{a^2} \\ \frac{2w_2}{b^2} \end{bmatrix}$$

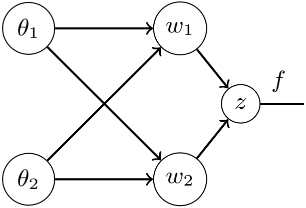
$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial w_1^2} & \frac{\partial^2 f}{\partial w_1 \partial w_2} \\ \frac{\partial^2 f}{\partial w_2 \partial w_1} & \frac{\partial^2 f}{\partial w_2^2} \end{bmatrix} = \begin{bmatrix} \frac{2}{a^2} & 0 \\ 0 & \frac{2}{b^2} \end{bmatrix}$$



- As long as all second derivates exist, the Hessian H is symmetric
- Hessian captures the curvature of the surface

Calculus Background: Chain Rule

$$z = f(w_1(\theta_1, \theta_2), w_2(\theta_1, \theta_2))$$



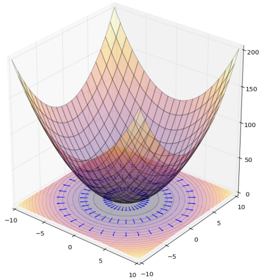
$$\frac{\partial f}{\partial \theta_1} = \frac{\partial f}{\partial w_1} \cdot \frac{\partial w_1}{\partial \theta_1} + \frac{\partial f}{\partial w_2} \cdot \frac{\partial w_2}{\partial \theta_1}$$

We will use this a lot when we study neural networks and back propagation

General Form for Gradient and Hessian

Suppose $\mathbf{w} \in \mathbb{R}^D$ and $f : \mathbb{R}^D \to \mathbb{R}$

The gradient vector contains all first order partial derivatives



$$\nabla_{\mathbf{w}} f(\mathbf{w}) = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \\ \vdots \\ \frac{\partial f}{\partial w_D} \end{bmatrix}$$

Hessian matrix of f contains all second order partial derivatives.

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial w_1^2} & \frac{\partial^2 f}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 f}{\partial w_1 \partial w_D} \\ \frac{\partial^2 f}{\partial w_2 \partial w_1} & \frac{\partial^2 f}{\partial w_2^2} & \cdots & \frac{\partial^2 f}{\partial w_2 \partial w_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial w_D \partial w_1} & \frac{\partial^2 f}{\partial w_D \partial w_2} & \cdots & \frac{\partial^2 f}{\partial w_D^2} \end{bmatrix}$$

Gradient Descent Algorithm

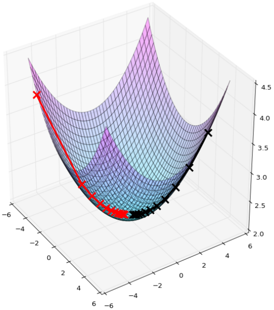
Gradient descent is one of the simplest, but very general algorithm for optimization

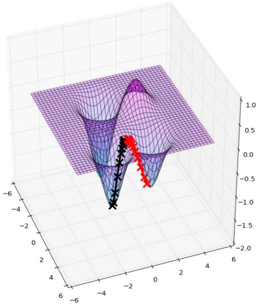
It is an iterative algorithm, producing a new $\mathbf{w}_{t+1}$ at each iteration as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \mathbf{g}_t = \mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t)$$

We will denote the gradients by $\mathbf{g}_t$

$\eta_t > 0$ is the learning rate or step size





Gradient Descent for Least Squares Regression

$$\mathcal{L}(\mathbf{w}) = (\mathbf{X}\mathbf{w} - y)^\mathsf{T}(\mathbf{X}\mathbf{w} - \mathbf{y}) = \sum_{i=1}^{N}(\mathbf{x}_i^\mathsf{T}\mathbf{w} - y_i)^2$$

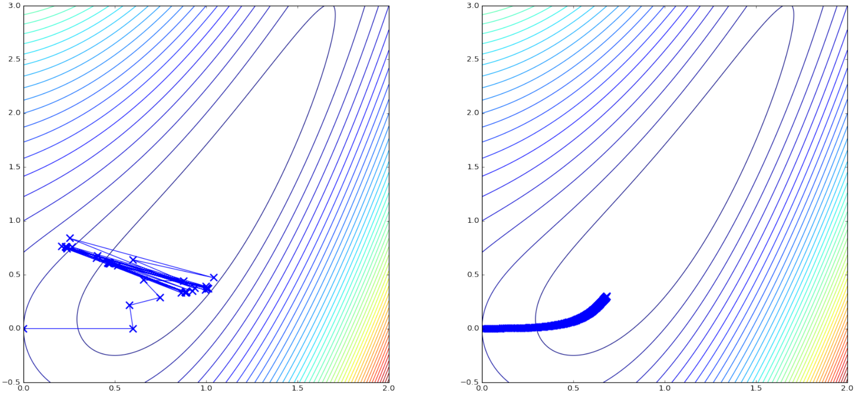
We can compute the gradient of $\mathcal{L}$ with respect to $\mathbf{w}$

$$\nabla_{\mathbf{w}}\mathcal{L} = 2\left(\mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{w} - \mathbf{X}^\mathsf{T}\mathbf{y}\right)$$

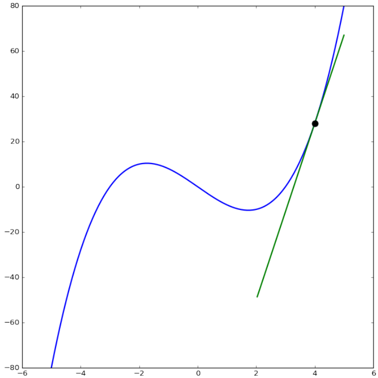
- Why would you want to use gradient descent instead of directly plugging in the formula?
- If N and D are both very large
  - Computational complexity of matrix formula $O\left(\min\{N^2 D, N D^2\}\right)$
  - Each gradient calculation $O(ND)$

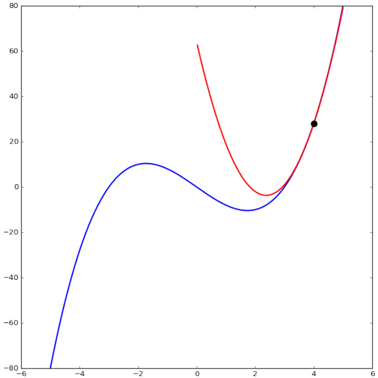
Choosing a Step Size

- Choosing a good step-size is important
- It step size is too large, algorithm may never converge
- If step size is too small, convergence may be very slow
- May want a time-varying step size



Newton's Method (Second Order Method)





- Gradient descent uses only the first derivative
- Local linear approximation

- Newton's method uses second derivatives
- Degree 2 Taylor approximation around current point

Newton's Method in High Dimensions

The updates depend on the gradient $\mathbf{g}_t$ and the Hessian $\mathbf{H}_t$ at point $\mathbf{w}_t$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}_t^{-1}\mathbf{g}_t$$

Approximate f around $\mathbf{w}_t$ using second order Taylor approximation

$$f_{\text{quad}}(\mathbf{w}) = f(\mathbf{w}_t) + \mathbf{g}_t^\mathsf{T}(\mathbf{w} - \mathbf{w}_t) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_t)^\mathsf{T}\mathbf{H}_t(\mathbf{w} - \mathbf{w}_t)$$

We move directly to the (unique) stationary point of f_{quad}

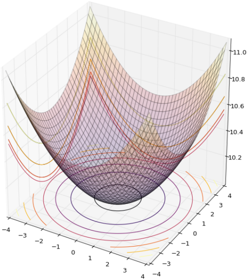
The gradient of f_{quad} is given by:

$$\nabla_{\mathbf{w}} f_{\text{quad}} = \mathbf{g}_t + \mathbf{H}_t(\mathbf{w} - \mathbf{w}_t)$$

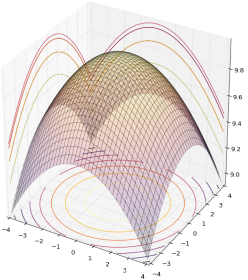
Setting $\nabla_{\mathbf{w}} f_{\text{quad}} = 0$, to get $\mathbf{w}_{t+1}$, we have

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}_t^{-1}\mathbf{g}_t$$

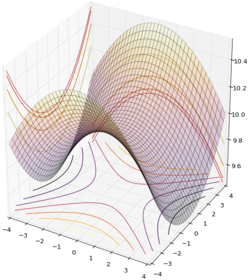
Newton's Method gives Stationary Points



$\mathbf{H}$ has positive eigenvalues



$\mathbf{H}$ has negative eigenvalues



$\mathbf{H}$ has mixed eigenvalues

Hessian will tell you which kind of stationary point is found

Newton's method can be computationally expensive in high dimensions.
Need to compute and invert a Hessian at each iteration

Minimising the Lasso Objective

For the Lasso objective, *i.e.*, linear model with ℓ_1-regularisation, we have

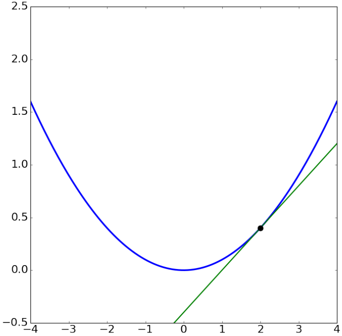
$$\mathcal{L}_{\text{lasso}}(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w}^{\mathsf{T}} \mathbf{x}_i - y_i)^2 + \lambda \sum_{i=1}^{D} |w_i|$$

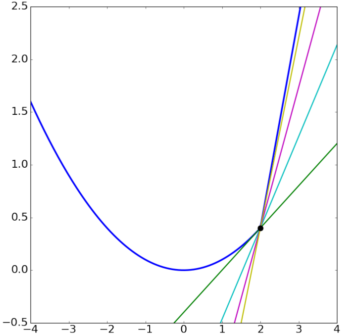
- Quadratic part of the loss function can't be framed as linear programming
- Lasso regularization does not allow for closed form solutions
- Must resort to general optimisation methods
- We still have the problem that the objective function is not differentiable!

Sub-gradient Descent

Focus on the case when f is convex,

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y) \qquad \text{for all } x, y, \alpha \in [0,1]$$





$$f(x) \geq f(x_0) + g(x - x_0) \qquad \text{where } g \text{ is a sub-derivative}$$

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + \mathbf{g}^{\mathsf{T}}(\mathbf{x} - \mathbf{x}_0) \qquad \text{where } \mathbf{g} \text{ is a sub-gradient}$$

Any $\mathbf{g}$ satisfying the above inequality will be called a sub-gradient at $\mathbf{x}_0$

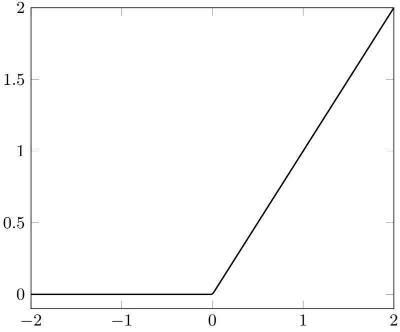
Sub-gradient Descent

$f(\mathbf{w}) = |w_1| + |w_2| + |w_3| + |w_4|$ for $\mathbf{w} \in \mathbb{R}^4$

What is a sub-gradient at the point $\mathbf{w} = [2, -3, 0, 1]^\mathsf{T}$?

$$\mathbf{g} = \nabla_{\mathbf{w}} f = \begin{bmatrix} 1 \\ -1 \\ \gamma \\ 1 \end{bmatrix}$$

for any $\gamma \in [-1, 1]$



The sub-derivative of $f(x) = \max(x, 0)$ at $x = 0$ is $[0, 1]$.

Optimization Algorithms for Machine Learning

We have data $\mathcal{D} = \langle(\mathbf{x}_i, y_i)\rangle_{i=1}^{N}$. We are minimizing the objective function:

$$\mathcal{L}(\mathbf{w}; \mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} \ell(\mathbf{w}; \mathbf{x}_i, y_i) + \underbrace{\lambda\mathcal{R}(\mathbf{w})}_{\text{Regularisation Term}}$$

The gradient of the objective function is

$$\nabla_{\mathbf{w}}\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \nabla_{\mathbf{w}}\ell(\mathbf{w}; \mathbf{x}_i, y_i) + \lambda\nabla_{\mathbf{w}}\mathcal{R}(\mathbf{w})$$

For Ridge Regression we have

$$\mathcal{L}_{\text{ridge}}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^{\mathsf{T}}\mathbf{x}_i - y_i)^2 + \lambda\mathbf{w}^{\mathsf{T}}\mathbf{w}$$

$$\nabla_{\mathbf{w}}\mathcal{L}_{\text{ridge}} = \frac{1}{N}\sum_{i=1}^{N} 2(\mathbf{w}^{\mathsf{T}}\mathbf{x}_i - y_i)\mathbf{x}_i + 2\lambda\mathbf{w}$$

Stochastic Gradient Descent

As part of the learning algorithm, we calculate the following gradient:

$$\nabla_{\mathbf{w}}\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\nabla_{\mathbf{w}}\ell(\mathbf{w};\mathbf{x}_i, y_i) + \mathcal{R}(\mathbf{w})$$

Suppose we pick a random datapoint $(\mathbf{x}_i, y_i)$ and evaluate $\mathbf{g}_i = \nabla_{\mathbf{w}}\ell(\mathbf{w};\mathbf{x}_i, y_i)$

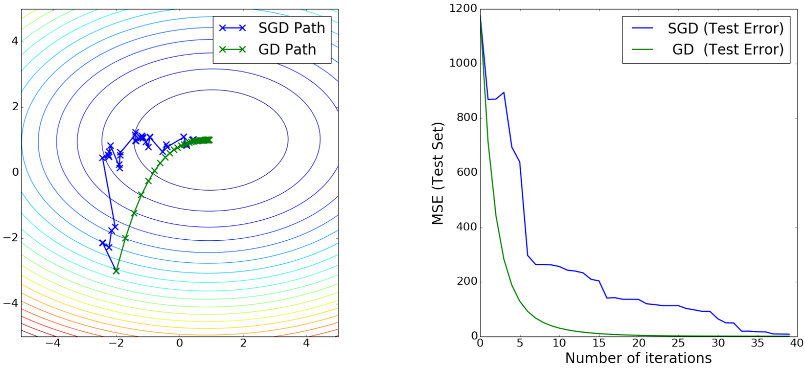
What is $\mathbb{E}[\mathbf{g}_i]$?

$$\mathbb{E}[\mathbf{g}_i] = \frac{1}{N}\sum_{i=1}^{N}\nabla_{\mathbf{w}}\ell(\mathbf{w};\mathbf{x}_i, y_i)$$

Instead of computing the entire gradient, we can compute the gradient at just a single datapoint!

In expectation $\mathbf{g}_i$ points in the same direction as the entire gradient (except for the regularisation term)

Online Learning: Stochastic Gradient Descent



- Using stochastic gradient descent it is possible to learn "online", *i.e.*, we get data little at a time
- Cost of computing the gradient in 'Stochastic Gradient Descent (SGD)' is significantly less compared to the gradient on the full dataset
- Learning rates should be chosen by (cross-)validation

Batch/Offline Learning

$$w_{t+1} = w_t - \frac{\eta}{N} \sum_{i=1}^{N} \nabla_{\mathbf{w}} \ell(\mathbf{w}; \mathbf{x}_i, y_i) - \lambda \nabla_{\mathbf{w}} \mathcal{R}(\mathbf{w})$$

Online Learning

$$w_{t+1} = w_t - \eta \nabla_{\mathbf{w}} \ell(\mathbf{w}; \mathbf{x}_i, y_i) - \lambda \nabla_{\mathbf{w}} \mathcal{R}(\mathbf{w})$$

Minibatch Online Learning

$$w_{t+1} = w_t - \frac{\eta}{b} \sum_{i=1}^{b} \nabla_{\mathbf{w}} \ell(\mathbf{w}; \mathbf{x}_i, y_i) - \lambda \nabla_{\mathbf{w}} \mathcal{R}(\mathbf{w})$$

Many Optimisation Techniques (Tricks)

First Order Methods/(Sub) Gradient Methods

- Nesterov's Accelerated Gradient
- Line-Search to Find Step-Size
- Momentum-based Methods
- AdaGrad, AdaDelta, Adam, RMSProp

Second Order/Newton/Quasinewton Methods

- Conjugate Gradient Method
- BGFS and L-BGFS

Adagrad: Example Application for Text Data

Heathrow: Will Boris Johnson lie down in front of the bulldozers? He was happy to lie down the side of a bus.

. . .

On his part, Johnson has already sought to clarify the comments, telling Sky News that what he in fact said was *not* that he would lie down *in front* of the bulldozers, but that he would lie down *the side*. And he never actually said *bulldozers*, he said *bus*.

y	x_1	x_2	x_3	x_4
1	1	0	0	1
-1	1	1	0	0
-1	1	1	1	0
1	1	1	0	0
1	1	0	0	0
-1	1	1	1	0
1	1	1	0	0
1	1	1	0	1
1	1	1	0	0

Adagrad Update

$$w_{t+1,i} \leftarrow w_{t,i} - \frac{\eta}{\sqrt{\sum_{s=1}^{t} g_{s,i}^2}} g_{t,i}$$

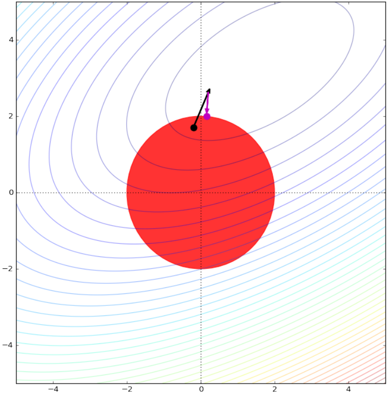
Rare features (which are 0 in most datapoints) can be most predictive

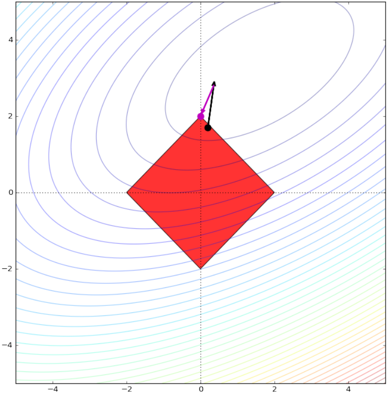
Constrained Convex Optimization

Often we want to look for a solution in a constrained set (not all of $\mathbb{R}^D$)

For example, minimise $(\mathbf{X}\mathbf{w} - \mathbf{y})^\mathsf{T}(\mathbf{X}\mathbf{w} - \mathbf{y})$ in the sets $\mathbf{w}^\mathsf{T}\mathbf{w} < R$, or $\sum_{i=1}^{D} |w_i| < R$

Gradient step is followed by a projection step





Summary

Convex Optimization

- Convex Optimization is ‘efficient’ (*i.e.*, polynomial time)
- Try to cast learning problem as a convex optimization problem
- Many, many extensions exist: Adagrad, Momentum-based, BGFS, L-BGFS, Adam, *etc.*
- Books: Boyd and Vandenberghe, Nesterov’s Book

Non-Convex Optimization

- Encountered frequently in deep learning
- Stochastic Gradient Descent gives local minima
- Nonlinear Programming - Dimitri Bertsekas