

Deep Learning 輪読会 2017
第4章 数値計算

2017.11.06

東京大学工学部坂田・森研究室

B4 中元雪絵

4.1 オーバーフローとアンダーフロー

4.2 悪条件

4.3 勾配に基づく最適化

4.4 制約付き最適化

4.5 例：最小二乗法

4.1 オーバーフローとアンダーフロー

- 丸め誤差
 - 実数を有限のビットパターンで表現すると、近似による**丸め誤差**が生じる
 - アンダーフロー：0に近い数を0に丸めた時 → ゼロ除算、0の対数
 - オーバーフロー：巨大な数値を ∞ に近似した時

- 例：ソフトマックス関数

- アンダーフローやオーバーフローに対して安定的な動作が求められる

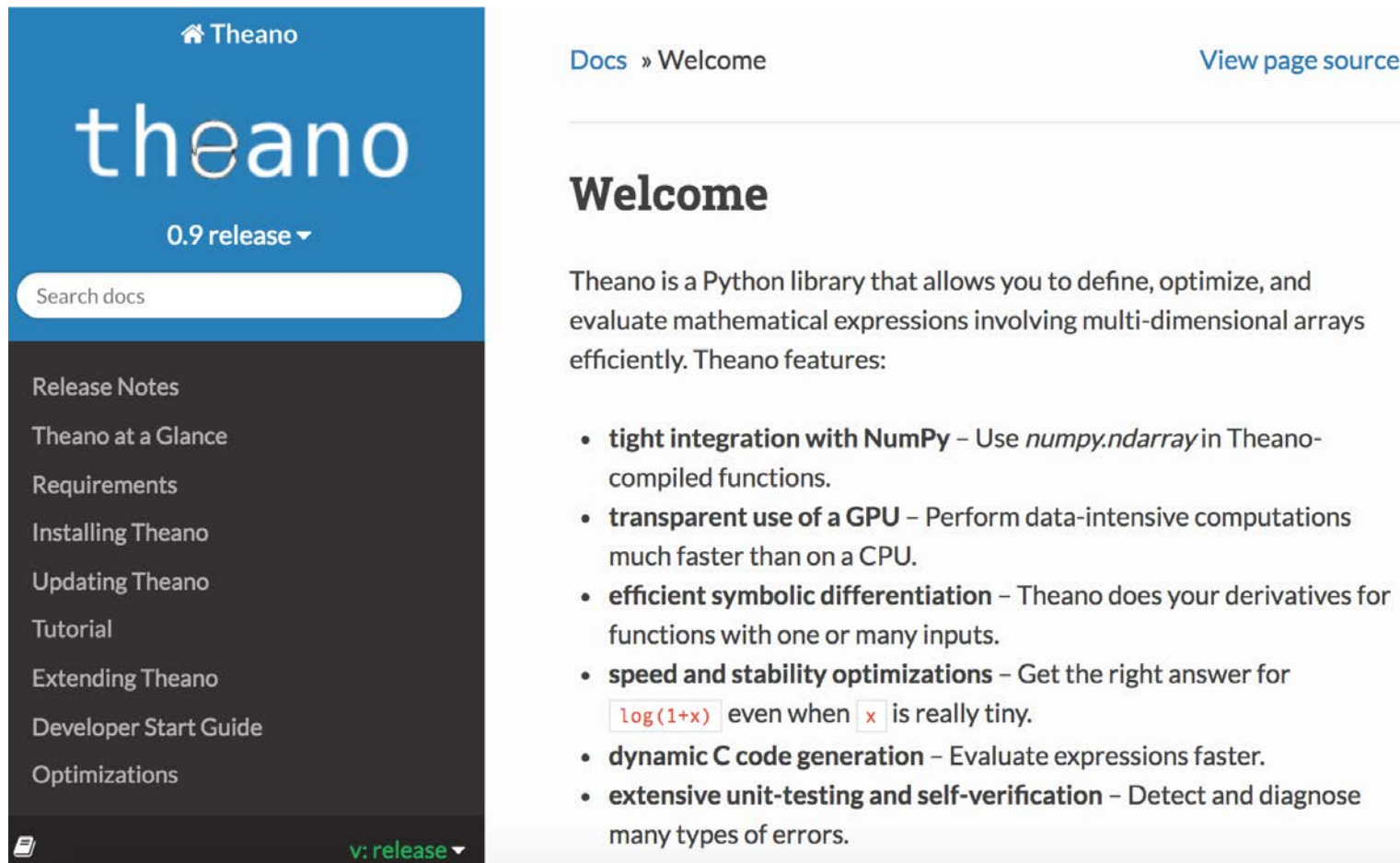
$$\text{softmax}(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)}.$$

- 全ての x_i がある定数 c に等しい場合
 - 解析的には出力は $1/n$ だが、 c の絶対値が非常に大きいとアンダーフローやオーバーフローが発生。Softmax(\mathbf{z})を使うことで解決できる。

$$\mathbf{z} = \mathbf{x} - \max_i x_i$$

4.1 オーバーフローとアンダーフロー

- Theano (Bergstra et al, 2010; Bastien et al, 2012)
 - Python用数値計算ライブラリ
 - 数値計算上不安定な計算式の多くを自動的に検出して安定化させる



The screenshot displays the Theano documentation website. The left sidebar contains a navigation menu with the following items: Release Notes, Theano at a Glance, Requirements, Installing Theano, Updating Theano, Tutorial, Extending Theano, Developer Start Guide, and Optimizations. The main content area features a 'Welcome' heading and a paragraph stating: 'Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. Theano features:'. Below this, a bulleted list highlights key features: tight integration with NumPy, transparent use of a GPU, efficient symbolic differentiation, speed and stability optimizations (illustrated with the example `log(1+x)`), dynamic C code generation, and extensive unit-testing and self-verification. The top of the page includes the Theano logo, version information (0.9 release), and links to documentation and source code.

Theano

theano

0.9 release ▾

Search docs

Release Notes

Theano at a Glance

Requirements

Installing Theano

Updating Theano

Tutorial

Extending Theano

Developer Start Guide

Optimizations

Docs » Welcome

[View page source](#)

Welcome

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. Theano features:

- **tight integration with NumPy** – Use `numpy.ndarray` in Theano-compiled functions.
- **transparent use of a GPU** – Perform data-intensive computations much faster than on a CPU.
- **efficient symbolic differentiation** – Theano does your derivatives for functions with one or many inputs.
- **speed and stability optimizations** – Get the right answer for `log(1+x)` even when `x` is really tiny.
- **dynamic C code generation** – Evaluate expressions faster.
- **extensive unit-testing and self-verification** – Detect and diagnose many types of errors.

v: release ▾

4.2 悪条件

- 条件(Condition)
 - 入力値の小さな変化に対して関数がどれだけ急激に変化するか
 - 関数 $f(\mathbf{x}) = \mathbf{A}^{-1}\mathbf{x}$ についてAが固有値分解できる場合、条件数を以下で表す

$$\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|$$

- この値が大きいと、逆行列が入力値の誤差に敏感となり、逆行列をかけると既存の誤差が増幅する

4.3 勾配に基づく最適化

- 用語
 - 最適化したい関数：目的関数(Objective Function) / 基準(criterion)
 - 最小化する場合のみ、次のように呼ぶこともある
 - コスト関数(cost function)
 - 損失関数(loss function)
 - 誤差関数(error function)
 - 関数を最小化または最大かすることを上付き文字*で表す

$$\boldsymbol{x}^* = \arg \min f(\boldsymbol{x}).$$

4.3 勾配に基づく最適化

- 微積分と最適化
 - 勾配降下法

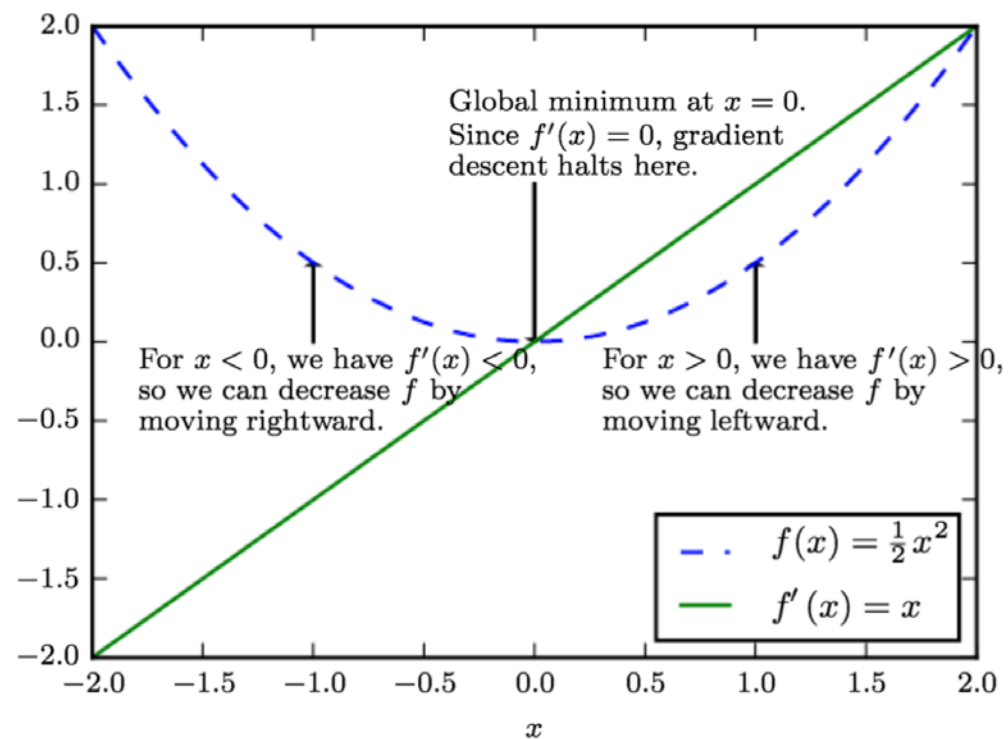


Figure 4.1: Gradient descent. An illustration of how the gradient descent algorithm uses the derivatives of a function can be used to follow the function downhill to a minimum.

4.3 勾配に基づく最適化

- 臨界点(critical points) / 停留点(stationery points) の種類
 - 左から極小値(local minimum)、極大値(local maximum)、鞍点(saddle points)

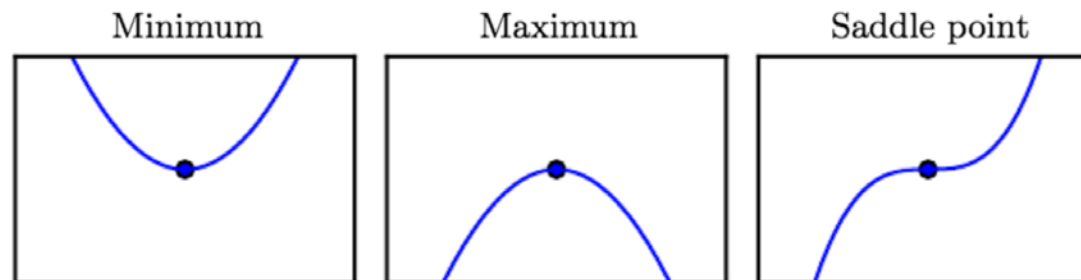


Figure 4.2: Types of critical points. Examples of the three types of critical points in one dimension. A critical point is a point with zero slope. Such a point can either be a local minimum, which is lower than the neighboring points; a local maximum, which is higher than the neighboring points; or a saddle point, which has neighbors that are both higher and lower than the point itself.

4.3 勾配に基づく最適化

- 複数の極小値の扱い → 大域的最適解でない値で手を打つこともある

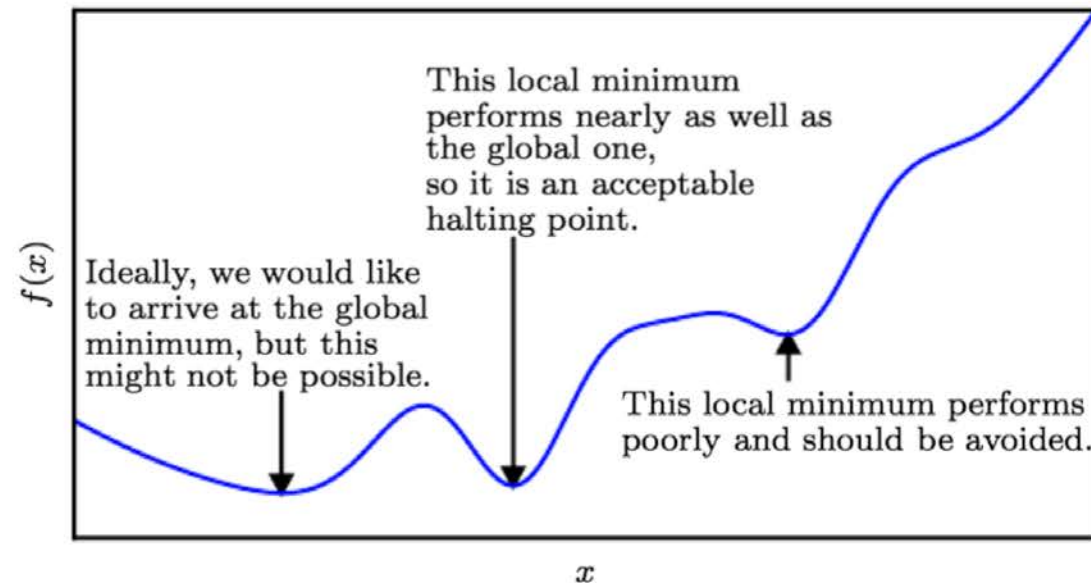


Figure 4.3: Approximate minimization. Optimization algorithms may fail to find a global minimum when there are multiple local minima or plateaus present. In the context of deep learning, we generally accept such solutions even though they are not truly minimal, so long as they correspond to significantly low values of the cost function.

4.3 勾配に基づく最適化

- 複数の入力を持つ関数
 - 勾配 $\nabla_{\mathbf{x}} f(\mathbf{x})$: f に関する偏微分全てを要素とするベクトル
 - 方向微分 : $f(\mathbf{x} + a\mathbf{u})$ の a に関する微分を $a=0$ で評価したもの

$$\begin{aligned} & \min_{\mathbf{u}, \mathbf{u}^\top \mathbf{u} = 1} \mathbf{u}^\top \nabla_{\mathbf{x}} f(\mathbf{x}) \\ &= \min_{\mathbf{u}, \mathbf{u}^\top \mathbf{u} = 1} \|\mathbf{u}\|_2 \|\nabla_{\mathbf{x}} f(\mathbf{x})\|_2 \cos \theta \end{aligned}$$

→ f が最も速く減少する方向は、勾配と反対方向

4.3 勾配に基づく最適化

- **最急降下法(method of steepest descent)**

- 次の式により点を更新する

$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

- 学習率 ϵ : ステップ幅を決める正のスカラー値
- 直線探索 : $f(\boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}))$ を ϵ に対して計算し、最小となるものを選択

- 参考デモ

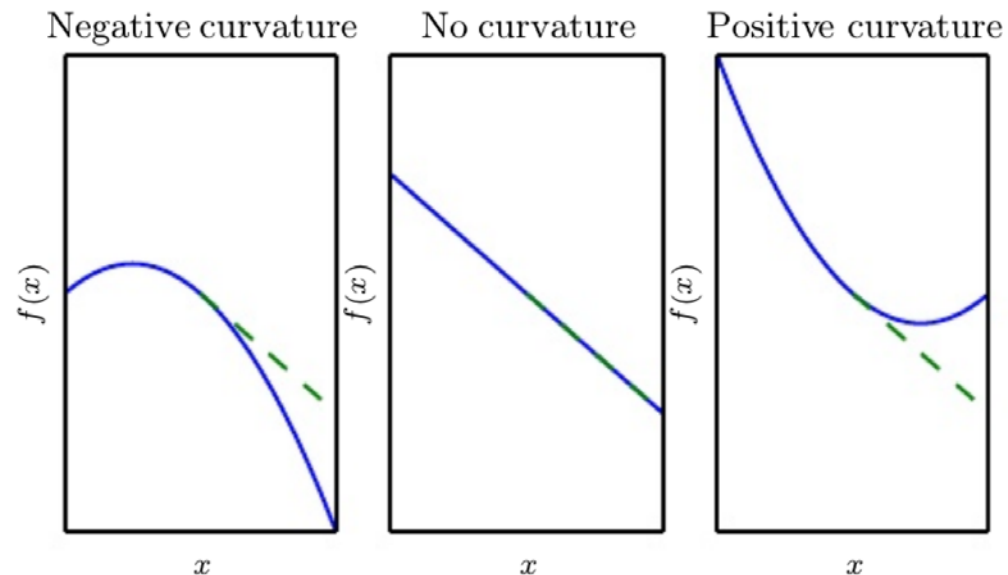
<http://www.benfrederickson.com/numerical-optimization/>

4.3.1 勾配を超えて：ヤコビ行列とヘッセ行列

- 入力も出力も共にベクトルである場合に一般化
 - ヤコビ行列(Jacobian matrix)：偏微分全てを要素としてもつ行列

$$J_{i,j} = \frac{\partial}{\partial x_j} f(\mathbf{x})_i$$

- 二階微分と曲率(curvature)
 - 曲率が負なら関数は下向きに曲がり、正なら上向きに曲がる



4.3.1 勾配を超えて：ヤコビ行列とヘッセ行列

- 入力も出力も共にベクトルである場合に一般化
 - ヘッセ行列(Hessian matrix)：二階微分全てを要素としてもつ行列

$$\mathbf{H}(f)(\mathbf{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\mathbf{x}).$$

4.3.1 勾配を超えて：ヤコビ行列とヘッセ行列

- 最急降下法の1ステップの効果を予測する

- 現在の点の周りでのテイラー展開

$$f(\mathbf{x}) \approx f(\mathbf{x}^{(0)}) + (\mathbf{x} - \mathbf{x}^{(0)})^\top \mathbf{g} + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(0)})^\top \mathbf{H}(\mathbf{x} - \mathbf{x}^{(0)}).$$

- 新しい点を代入：第1項が元の値、第2項が傾きからの予測される改善値、第3項が曲率による補正值

$$f(\mathbf{x}^{(0)} - \epsilon \mathbf{g}) \approx f(\mathbf{x}^{(0)}) - \epsilon \mathbf{g}^\top \mathbf{g} + \frac{1}{2} \epsilon^2 \mathbf{g}^\top \mathbf{H} \mathbf{g}$$

- $\mathbf{g}^\top \mathbf{H} \mathbf{g}$ が正の場合、最適なステップ幅は $\epsilon^* = \frac{\mathbf{g}^\top \mathbf{g}}{\mathbf{g}^\top \mathbf{H} \mathbf{g}}.$

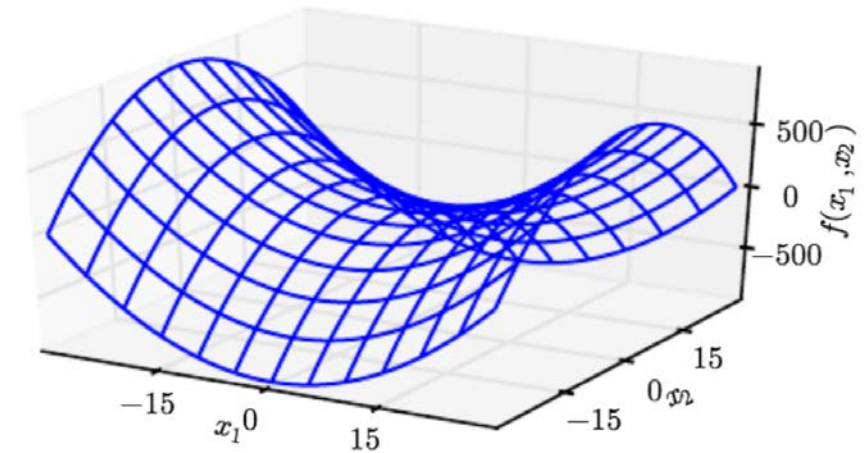
4.3.1 勾配を超えて：ヤコビ行列とヘッセ行列

- 二階微分による極値判定法

- 一次元の場合

$f''(x) > 0$: x で極小、 $f''(x) < 0$: x で極大、

$f''(x) = 0$: 判定できない



- 多次元の場合

- ヘッセ行列が正定（全ての固有値が正）なら**極小**

- 方向二階微分がどの方向に対しても正になるため

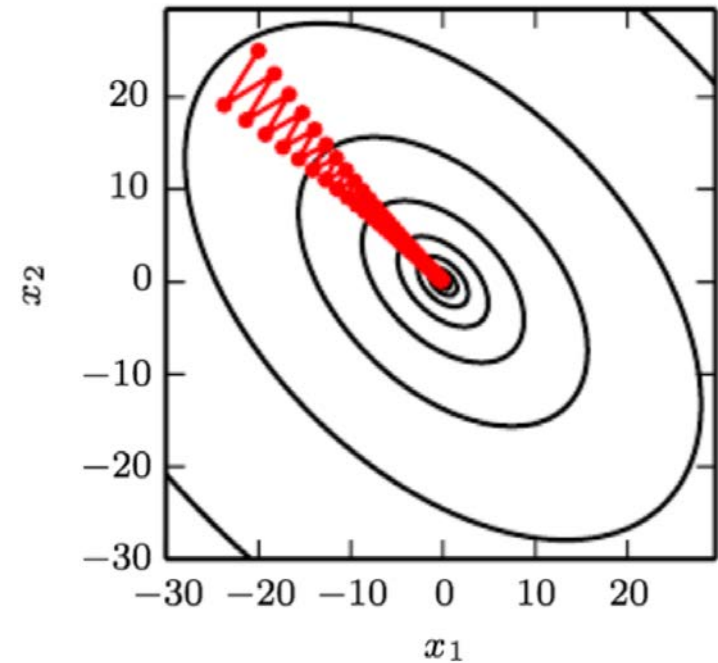
- ヘッセ行列が負定（全ての固有値が負）なら**極大**

- 少なくとも一つの固有値が正で、一つが負なら**鞍点**（図参照）

- 少なくとも一つの固有値が0でそれ以外の符号が全て同じなら**判定不能**

4.3.1 勾配を超えて：ヤコビ行列とヘッセ行列

- ヘッセ行列の条件数が悪い時の最急降下法
 - 勾配の変化量が方向によって大きく異なる
 - 長く負である方向を優先的に探索できない



4.3.1 勾配を超えて：ヤコビ行列とヘッセ行列

- ニュートン法(Newton's method)

- $f(x)$ の二次テイラー級数展開

$$f(x) \approx f(x^{(0)}) + (x - x^{(0)})^\top g + \frac{1}{2}(x - x^{(0)})^\top H(x - x^{(0)}).$$

- この関数の臨界点を求め、次の点とする

$$x^* = x^{(0)} - H(f)(x^{(0)})^{-1} \nabla_x f(x^{(0)}).$$

- ヘッセ行列の逆行列を計算するため、計算量が多い
- 最急降下法よりずっと早く臨界点を見つけることができる
- 極小値の近傍では有用だが、近くに鞍点がある場合は適さない

4.3.1 勾配を超えて：ヤコビ行列とヘッセ行列

- 最適化アルゴリズム
 - 勾配のみを利用するもの：一次最適化アルゴリズム ex)最急降下法
 - ヘッセ行列も利用するもの：二次最適化アルゴリズム ex)ニュートン法
- アルゴリズムを深層学習の関数に適用できる保証はない
 - リプシッツ連続(Lipschitz continuous)な関数に限定することで保証を得られる場合がある
$$\forall \mathbf{x}, \forall \mathbf{y}, |f(\mathbf{x}) - f(\mathbf{y})| \leq \mathcal{L} \|\mathbf{x} - \mathbf{y}\|_2.$$
 - 凸最適化(convex optimization)が最もうまくいくが、深層学習への適用は困難

4.4 制約付き最適化

- 制約付き最適化(constrained optimization)
 - 実現可能(feasible)点の集合に含まれる x に対して最適化を行う
- **カルーシュ・クーン・タッカー法(Karush-Kuhn-Tucker, KKT法)**
 - 一般化ラグランジアン(Lagrangian) / ラグランジュ関数(Lagrange function)

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\mathbf{x}) + \sum_i \lambda_i g^{(i)}(\mathbf{x}) + \sum_j \alpha_j h^{(j)}(\mathbf{x}).$$

$g(x)$: 等式制約、 $h(x)$: 不等式制約、 λ, α : KKT乗数

- 制約付き最小化問題は、次の制約なしの最適化問題に帰着できる

$$\min_{\mathbf{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha})$$

4.4 制約付き最適化

- **カルーシュ・クーン・タッカー法(Karush-Kuhn-Tucker, KKT法)**
 - KKT条件（必要だが十分ではない）
 - 一般化ラグランジアンの変分が0
 - x 及びKKT乗数の両方に関する制約が全て満たされている
 - 不等式制約が「相補性」を持つ $\alpha \odot h(x) = 0$.

4.5 例：線形最小二乗法

- 次の関数を最小化する \mathbf{x} の値を求める

$$f(\mathbf{x}) = \frac{1}{2} \|\mathbf{Ax} - \mathbf{b}\|_2^2.$$

- 勾配降下法

Algorithm 4.1 An algorithm to minimize $f(\mathbf{x}) = \frac{1}{2} \|\mathbf{Ax} - \mathbf{b}\|_2^2$ with respect to \mathbf{x} using gradient descent, starting from an arbitrary value of \mathbf{x} .

Set the step size (ϵ) and tolerance (δ) to small, positive numbers.

while $\|\mathbf{A}^\top \mathbf{Ax} - \mathbf{A}^\top \mathbf{b}\|_2 > \delta$ **do**

$\mathbf{x} \leftarrow \mathbf{x} - \epsilon (\mathbf{A}^\top \mathbf{Ax} - \mathbf{A}^\top \mathbf{b})$

end while

- ニュートン法：二次関数なので一回で最小値に収束

4.5 例：線形最小二乗法

- 次の関数を最小化する \mathbf{x} の値を求める

$$f(\mathbf{x}) = \frac{1}{2} \|\mathbf{Ax} - \mathbf{b}\|_2^2.$$

- 勾配降下法

Algorithm 4.1 An algorithm to minimize $f(\mathbf{x}) = \frac{1}{2} \|\mathbf{Ax} - \mathbf{b}\|_2^2$ with respect to \mathbf{x} using gradient descent, starting from an arbitrary value of \mathbf{x} .

Set the step size (ϵ) and tolerance (δ) to small, positive numbers.

while $\|\mathbf{A}^\top \mathbf{Ax} - \mathbf{A}^\top \mathbf{b}\|_2 > \delta$ **do**

$\mathbf{x} \leftarrow \mathbf{x} - \epsilon (\mathbf{A}^\top \mathbf{Ax} - \mathbf{A}^\top \mathbf{b})$

end while

- ニュートン法：二次関数なので一回で最小値に収束

4.5 例：線形最小二乗法

- 同じ問題を制約下で解く

$$f(\mathbf{x}) = \frac{1}{2} \|\mathbf{Ax} - \mathbf{b}\|_2^2, \quad \mathbf{x}^\top \mathbf{x} \leq 1$$

– ラグランジアンを導入

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda (\mathbf{x}^\top \mathbf{x} - 1). \quad \min_{\mathbf{x}} \max_{\lambda, \lambda \geq 0} L(\mathbf{x}, \lambda).$$

- 微分してxを求める

$$\mathbf{A}^\top \mathbf{Ax} - \mathbf{A}^\top \mathbf{b} + 2\lambda \mathbf{x} = 0. \quad \mathbf{x} = (\mathbf{A}^\top \mathbf{A} + 2\lambda \mathbf{I})^{-1} \mathbf{A}^\top \mathbf{b}.$$

- λ を決定する

$$\frac{\partial}{\partial \lambda} L(\mathbf{x}, \lambda) = \mathbf{x}^\top \mathbf{x} - 1.$$

- 最終的にxが正しいノルムを持ち、 λ の微分が0になるまで調整を繰り返す

参考文献

- Deep Learning

- Ian Goodfellow, Yoshua Bengio, Aaron Courville

- 日本語版

- <https://www.amazon.co.jp/%E6%B7%B1%E5%B1%A4%E5%AD%A6%E7%BF%92-Ian-Goodfellow/dp/4048930621>