

機械学習の整理帳

tomixy

2025 年 8 月 4 日

目次

第 I 部	機械学習の概念	3
第 1 章	学習の原理	4
	意思決定のプロセス	4
	モデルとパラメータによる学習	5
	パラメータ数とモデルの表現力	6
	汎化能力	6
	過学習	7
第 2 章	学習の手法	10
	教師あり学習	10
	教師あり学習の流れ	11
	教師なし学習	11
	強化学習	12

第 II 部 教師あり学習の構成	13
第 3 章 損失関数	14
損失関数の設計	14
分類器の-marginと損失関数	15
分類問題：0/1 損失関数	16
分類問題：クロスエントロピー損失関数	16
回帰問題：二乗損失・絶対損失	16
第 4 章 目的関数	18
訓練誤差と目的関数	18
二乗損失・絶対損失による目的関数	19
正則化	19
目的関数の変更による正則化	20
正則化項	21
正則化パラメータ	21
ハイパーパラメータ	22
第 5 章 関数の最適化	23
解析解と数値解	23
勾配降下法	24

第 I 部

機械学習の概念

第 1 章

学習の原理



意思決定のプロセス

経験に基づいて意思決定を行うために人間が用いるプロセスは記憶・定式化・予測フレームワークと呼ばれ、次の 3 つのステップで構成されている。

1. 記憶：過去の同じような状況を思い出す
2. 定式化：全般的なルールを定式化する
3. 予測：このルールを使って将来起こるかもしれないことを予測する


コンピュータに「記憶・定式化・予測」フレームワークを使わせることで、コンピュータに私たちと同じように考えさせることができる。

1. 記憶：巨大なデータテーブルを調べる
2. 定式化：さまざまなルールや式を調べてデータに最適なモデルを作成する
3. 予測：モデルを使って未来（未知）のデータについて予測を行う



モデルとパラメータによる学習


コンピュータはデータを使って**モデル** (model) を構築するという方法で問題を解く。

 **モデル** データを表すルール集まりであり、予測を行うために使うことができる

モデルは、対象の問題で獲得したい分類器や予測器、生成器などであり、入力（具体的なデータ）から出力（予測結果）を計算する関数とみなすことができる。

パラメータ

モデルの挙動を調整する「つまみ」となる変数を**パラメータ** (parameter) という。
パラメータ θ によって関数の挙動が決まることを、関数が θ によって「特徴づけられた」という。


 **パラメトリックモデル** パラメータによって特徴づけられたモデル

パラメータ θ によって挙動が決まるモデルを $f(x; \theta)$ と表記する。


「;」以降の変数は、この関数の入力ではないことを表している。

学習の定義

モデルのパラメータを調整して、最適なモデルを構築することを**学習**という。

 **学習** 「データから**学習**する」とは、データからモデルの最適なパラメータを推定すること

人間の学習も、脳内にある神経回路のパラメータ（シナプスの重みなど）を調整して実現されている。



パラメータ数とモデルの表現力

モデルは、成り立つかもしれない**仮説**を表すものであり、パラメータの値によって、異なる仮説を表現することができる。

つまり、パラメータの推定は、複数存在する仮説の中から一つを選択することとみなすことができる。

たとえば、モデルのパラメータ θ が $\{-1, 0, 1\}$ のいずれかの値をとる場合は、

- $\theta = -1$ の場合のモデルが表す仮説
- $\theta = 0$ の場合のモデルが表す仮説
- $\theta = 1$ の場合のモデルが表す仮説

の中から選択しているとみなせる。

さまざまな仮説の中から選ぶことができる場合、「モデルの**表現力**が高い」という。

単純には、パラメータ数が多いモデルほど仮説数が多く、表現力が高いといえる。



汎化能力

計算機は多くの情報を誤りなく大量に記憶することができる。

そのため、起きうる事象を十分網羅できるようにデータを用意できれば、わざわざモデルを作らなくても、過去の似たような値をそのまま使えばよいのではないかと考えることもできる。

学習時のデータをすべてそのまま記憶し、それを予測時に利用するアプローチを**丸暗記** (**memorization**) という。

しかし、世の中の多くの問題では、すべてのケースを前もって列挙したり、それらを記憶しておくことはできない。

特に、入力値が**高次元データ**（画像、音声、言語、時系列、etc.）や**連続値**である場合、すべての事例を網羅することは不可能である。

丸暗記できない場合は機械学習が有効

機械学習は、有限の「訓練データ」を用いて、無限ともいえる「未知のデータ」に対してもうまく動くようなモデルを作る手法といえる。

未知のデータに対してもうまく動く能力を**汎化能力**（**generalization ability**）という。

機械学習では、単に訓練データでうまくいくようなモデルを見つけるだけでなく、「未知のデータでどれだけうまくいくか」を表す汎化能力をどのように獲得するかが重要な課題となる。



過学習

訓練データではうまくいっているのに、訓練時には見なかった未知のデータではうまくいかない状態を**過学習**（**overfitting**）という。

過学習が起こる原理

たくさんのデータを集めれば、仮説が本当に成立するかどうかを高い確率で検証することができる。

データ数に対して仮説数が多い場合、正しい仮説よりも、たまたま成り立ってしまう誤った仮説が含まれる可能性が高くなる。これが過学習が起こる場合である。

- 検証する仮説数が少ない場合：事例をすべて満たしている仮説が本当に成り立つ可能性が高い
- 検証する仮説数が多い場合：事例をすべて満たしている仮説も、たまたま成り立っているだけの可能性が高い

機械学習は、多くの仮説（パラメータがある値をとる時のモデル）の中から、多くの訓練事例を説明する仮説がどれかを探す問題ともいえる。

このとき、過学習は、たまたま多くの訓練事例でうまくいくようなモデルが見つかってしまうことで起こる。

過学習を防ぐ原理

訓練データ数に比べて、検証する仮説数が少ない状況であれば、見つかった仮説がたまたま成り立ったものでなく、実際に関係がある可能性が高くなる。

そのため、過学習を防止するには、

$$\text{訓練データ数} > \text{検証する仮説数}$$

という状況を目指すことが有効となる。

過学習の防止策：訓練データを増やす


訓練データを増やすことができれば、多くの事例で仮説を検証できるので、たまたま仮説が成り立つ可能性を小さくすることができる。

しかし、訓練データを増やすことは困難である場合も多い。

そこで、訓練データに意味を変えないような変換を加えて、人工的にデータを水増しする **データオーグメンテーション**（**data augmentation**）という手法が有効となる。

過学習の防止策：仮説数を少なくする

訓練データ数が同じであれば、その中で仮説数を少なくすることも過学習の防止として有効である。

 **オッカムの剃刀** ある事柄を説明するためには、必要以上に多くを仮定すべきでない

仮説数を少なくするには、単純にはモデルのパラメータ数を少なくすればよい。

モデルに制約を加えることで、可能な限り単純なモデルを使うことで、仮説数を少なくする

ことができる。

しかし、過学習を抑える別の仕組みがある場合は、必ずしもパラメータ数が少ない方が汎化するとは限らない。

たとえば、ニューラルネットワーク (neural network) は、パラメータ数が膨大であるにも関わらず、高い汎化能力を持っている。

第 2 章

学習の手法



教師あり学習

教師あり学習（**supervised learning**）では、入力 x と推定したい出力 y からなるペア (x, y) を訓練データとして利用し、



入力 x から望ましい出力 y を予測できるような

モデル $y = f(x; \theta)$ を学習すること



を目標とする。

訓練データは、**教師ありデータ**や**学習データ**と呼ばれることもある。

$$D = \{(x_1, y_1), \dots, (x_n, y_n)\}$$

学習と推論

教師あり学習は、**学習**（**training**）と**推論**（**inference**）という 2 つのフェーズに分けられる。

学習フェーズでは、訓練データをうまく推定できるように、すなわち

$$y_i = f(x_i; \theta)$$

となるように、モデルのパラメータを調整していく。

推論フェーズでは、学習によって得られたパラメータ $\hat{\theta}$ を使ったモデル $f(x; \hat{\theta})$ を使い、新しいテストデータ \tilde{x} の出力を

$$\tilde{y} = f(\tilde{x}; \hat{\theta})$$

として求める。



教師あり学習の流れ

教師あり学習では、**訓練データ**、**モデル**、**損失関数**、**目的関数**、**最適化**をそれぞれ設計して組み合わせることで、学習を実現する。

1. **訓練データ**を用意する： $(x_i, y_i)_{i=1}^n$
2. 学習対象の**モデル**を用意する： $y = f(x; \theta)$
3. **損失関数**を設計する： $l(y, y')$
4. **目的関数**を導出する： $L(\theta) = \sum_i l(y_i, f(x_i; \theta)) + R(\theta)$
5. **最適化**問題を解く (**学習**)： $\theta^* = \underset{\theta}{\operatorname{argmin}} L(\theta)$
6. 学習して得られたモデルを**評価**する

[Note 1: それぞれの章へのリンクを貼る]



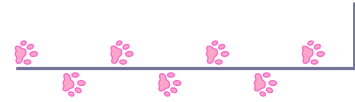
教師なし学習

教師なし学習 (**unsupervised learning**) は、教師（正解）がつけられていないデータ

$$D = \{x_1, \dots, x_n\}$$

を利用した学習であり、

データの特徴を捉え、データの最適な表現や
データ間の関係を獲得すること



を目標とする。

たとえば、教師なし学習では分類する目標がデータとして与えられないため、画像分類は学習できない。

代わりに、画像をどのようにデータとして表現できれば、後続のタスクがうまく処理できるのかを学習する。

教師なし学習の代表例

[Todo 1: book: ディープラーニングを支える技術 p63~]



強化学習

[Todo 2: book: ディープラーニングを支える技術 p66~]



第 II 部

教師あり学習の構成

第 3 章

損失関数



損失関数の設計

損失関数 (loss function) は、訓練データで与えられる正解に対し、



予測がどれだけ間違っているのか



を表す関数である。

コスト関数 (cost function) や誤差関数 (error function) と呼ばれることもある。

損失関数は、入力 \mathbf{x} 、正解の出力 \mathbf{y} 、モデルパラメータ θ を引数としてとり、0 以上の値を返す。

$$l(\mathbf{x}, \mathbf{y}; \theta) \geq 0$$

予測が正しければ 0 を返し、予測が間違っていれば正の値を返すようにする。

学習は、損失関数によって表される「現在の予測の間違っている度合い」を最小化するようなパラメータを求める最適化問題を解くことによって実現される。

損失関数の設計の重要性

損失関数は、学習の目的に応じて自由に設定することができる。

どのような損失関数を使うかによって、学習結果のモデルがどのような性質を持つかが決まる。

- 汎化性能
- ノイズに強いかわ弱いかわ
- 平均的な性能が優れているか、最悪の場合の性能が優れているか

たとえば、損失関数が大きく間違っているサンプルを重視するなら、大きな間違いはしないようになる。その反面、訓練データのノイズ（間違ったラベルがある場合など）に弱くなる。

損失関数の微分の形の重要性


損失関数の微分の性質によって、どのような解に収束するかが変わる。



分類器の-marginと損失関数

分類器が高い確信度で予測したにもかかわらず間違えた場合は、損失関数は大きな正の値を返すようにする。

確信度は、margin (margin) によって表される。

 margin 予測結果が境界面からどれだけ離れているか

境界面より離れている（marginが大きい）分類結果は、確信を持って「これは XXX である」と予測していることを表す。

逆に、境界面に近い（marginが小さい）分類結果は、「これはどちらかといえば XXX だが、YYY かもしれない」といったように、確信度が低いことを表している。

marginが大きいにもかかわらず予測が外れている場合は、今の予測や境界面が適切ではないことを示しており、大きな更新が必要となる。



分類問題：0/1 損失関数

0/1 損失関数は、分類問題で用いられる損失関数であり、モデルによる分類が間違っていたら 1、正しければ 0 を返す関数である。

0/1 損失関数

$$l_{0/1}(\mathbf{x}, y; \theta) = \begin{cases} 0 & \text{if } f(\mathbf{x}; \theta) = y \\ 1 & \text{if } f(\mathbf{x}; \theta) \neq y \end{cases}$$

この損失関数を使えば分類精度を評価できるが、微分がほとんどの位置で 0 になるため、**勾配法**を用いた学習では利用できない。

分類問題：クロスエントロピー損失関数

[Todo 3:]



回帰問題：二乗損失・絶対損失

回帰問題の場合は、**二乗損失**や**絶対損失**を損失関数として使うことが一般的である。

二乗損失は、正解データと予測値の差の二乗をとった値である。

二乗損失

$$l_{SE}(\mathbf{x}, y; \theta) = \|f(\mathbf{x}; \theta) - y\|^2$$

絶対損失は、差の絶対値をとった値である。

絶対損失

$$l_{AE}(\mathbf{x}, y; \theta) = \|f(\mathbf{x}; \theta) - y\|$$

二乗や絶対値をとることで、差が負になっても損失として正にできる。

最小値からのズレを許しやすいか

原点付近の様子を見ると、二次関数は最小値からずれても関数の値が急激には増加しない。

絶対値関数の方が、少しでもズレが生じると、関数の値が急に増える。

つまり、二乗損失より絶対損失の方が、最小値からのズレを許しづらくなる。

誤差の大きさを重視するか

二乗損失と絶対損失では、間違えた場合に重視する部分が異なる。

2 次関数は、入力が大きくなるにつれ、結果が急激に大きくなる関数である。

そのため、二乗損失は、大きく間違っている場合の誤差が大きくなる。

これに対し、絶対値関数は原点以外は直線（比例）であり、入力が大きくなっても結果は一定の割合で大きくなる関数である。

これらの性質から、

- 二乗損失は、誤差が大きい場合を重視して学習する
- 絶対損失は、相対的に誤差が小さい場合を重視して学習する

ということがいえる。

第 4 章

目的関数

訓練誤差と目的関数

訓練事例に対する損失関数を定義したら、訓練データ全体にわたって、損失関数の値の平均を計算する。

これを訓練誤差 (**training error**) と呼ぶ。

$$\frac{1}{N} \sum_{i=1}^N l(\mathbf{x}_i, y_i; \theta)$$

パラメータ θ を入力とし、訓練誤差を返す関数を、最適化問題の目的関数 (**objective function**) とする。

$$L(\theta) = \frac{1}{N} \sum_{i=1}^N l(\mathbf{x}_i, y_i; \theta)$$

この目的関数の値を最小化するパラメータ θ を求めることで、多くの訓練データの損失関数の値を小さくできるようなパラメータが求まる。

つまり、最適なパラメータを求める手順は、次のようになる。

1. 損失関数を定義する
2. その損失関数によって目的関数を定義する
3. 目的関数のパラメータ θ に関する最小化問題を解く

二乗損失・絶対損失による目的関数

回帰問題の場合は、**二乗損失**や**絶対損失**を損失関数として使うことが一般的である。

平均二乗誤差と最小二乗法

二乗損失を損失関数として用いた場合、目的関数は次のようになる。

$$L_{SE}(\theta) = \frac{1}{N} \sum_{i=1}^N \|f(\mathbf{x}_i; \theta) - y_i\|^2$$

これは「二乗の平均」を表す式になっているため、**平均二乗誤差**（**MSE: mean square error**）と呼ばれることがある。

平均二乗誤差を最小化する手法は、**最小二乗法**と呼ばれる。

平均二乗平方根誤差

また、二乗すると単位も二乗されてしまうため、もとのデータと単位を揃えるため、平方根をとることもある。

平均二乗誤差の平方根をとったものは、**平均二乗平方根誤差**（**RMSE: root mean square error**）と呼ばれる。

平均絶対誤差

絶対損失を損失関数として用いた場合、目的関数は次のようになる。

$$L_{AE}(\theta) = \frac{1}{N} \sum_{i=1}^N \|f(\mathbf{x}_i; \theta) - y_i\|$$

これは「絶対値の平均」を表す式になっているため、**平均絶対誤差**（**MAE: mean absolute error**）と呼ばれることがある。




正則化

複雑なモデルは多くのパラメータを持ち、柔軟に調整することができる。

しかし、モデルを柔軟にしすぎると過学習を起こす可能性がある。

そこで、パラメータに制限（条件）をつけることで過学習を防ぐ手法として、正則化（regularization）がある。

 **正則化** 学習時に行う、訓練誤差の最小化に加えて汎化性能を上げるための操作

過学習とは、訓練データではうまくいくが、未知のデータではうまくいかない状態のことだった。

過学習を防ぐということは、未知のデータに対してもうまく動くような「汎化能力を与える」ことだといえる。

目的関数の変更による正則化

正則化では、モデルを訓練する際に、次の 2 つの最適化を試みる。

- モデルの性能を向上させる（訓練誤差の最小化）
- モデルの複雑さを減らす（汎化性能の向上）

そのために、性能の指標と複雑度の指標を数値化し、それらを組み合わせた上で最適化問題を解くという方法をとる。

- 性能の指標：訓練誤差
- 複雑度の指標：正則化項（regularization term）

つまり、目的関数を次のように正則化項を加えた形に変更する。

$$\text{目的関数} = \text{訓練誤差} + \lambda \cdot \text{正則化項}$$

ここで、 λ は正の値であり、正則化パラメータと呼ばれるものである。

正則化項

正則化項は、モデルの複雑度を数値化したものである。

パラメータの数が多いモデルや、パラメータの値が大きいモデルは、複雑になる傾向がある。

そこで、次のような正則化項を用いることが多い。

- **L1 ノルム**：パラメータの絶対値の合計
- **L2 ノルム**：パラメータの二乗の合計

絶対値や二乗を使うのは、負のパラメータをなくすためである。

そうしないと、大きな負の値によって大きな正の値が相殺され、非常に複雑なモデルでもこれらの値が小さくなってしまうリスクがある。

L1 ノルムを使った正則化は **L1 正則化**、L2 ノルムを使った正則化は **L2 正則化**と呼ばれる。



正則化パラメータ

正則化を用いても、モデルの性能を向上させようとするすると複雑さが増し、モデルを単純化しようとするすると性能が低下する…という綱引き状態に陥ることがある。

その場合は、正則化パラメータ λ によって、性能と複雑度の間で調整を行う。

正則化パラメータ λ は、訓練誤差に対し、正則化項をどれだけ重視するのかを決めるパラメータである。

- λ が大きければ、正則化項を重視する
- λ が小さければ、訓練誤差を重視する

つまり、正則化パラメータの目的は、モデルの訓練プロセスにおいて、性能と単純さのどちらを重視すべきかを決めることにある。



ハイパーパラメータ

正則化パラメータのように、学習中に最適化するのではなく、学習時に前もって決める必要のあるパラメータをハイパーパラメータ (**hyperparameter**) という。

第 5 章

関数の最適化



解析解と数値解

目的関数が設定できたら、その目的関数の値を最小化するようなパラメータを求める。

解析的なアプローチ

目的関数の値を小さくできるようなパラメータを求めるには、「解析的に解けるか」をまず調べる。

たとえば、二次方程式の解の公式のように、解を得られる式が存在する場合は、「解析的に解ける」という。このように直接求められる解を **解析解** という。

逐次的なアプローチ

一方、解析解が使えない場合は、**数値解** を求める。

適当な初期値から始め、それを逐次的に更新していくことで、解を探索する。

解析的に解ける場合であっても、計算量が多すぎる場合は、このように逐次的に解くアプローチが用いられる。



勾配降下法

逐次的にパラメータを更新していくアプローチの中で、**勾配**の情報を使ってパラメータを逐次的に更新する方法を**勾配降下法**（GE: **gradient descent**）という。

勾配と関数の値

目的関数 $L(\boldsymbol{\theta})$ の、 $\boldsymbol{\theta}$ に関する勾配

$$\nabla L(\boldsymbol{\theta}) = \frac{\partial L}{\partial \boldsymbol{\theta}} = \left(\frac{\partial L}{\partial \theta_1}, \dots, \frac{\partial L}{\partial \theta_n} \right)$$

は、 $\boldsymbol{\theta}$ を変化させたときに関数の値が最も急激に増加する方向を表している。

そして、その逆 $-\nabla L(\boldsymbol{\theta})$ が、関数の値を最も急激に下げる方向を表している。



勾配がなぜ関数の値を最も急激に増加させる方向を表すのかは、拙著「[微分の整理帳](#)」で解説している。

勾配降下法のアルゴリズム

勾配降下法では、現在のパラメータに関する勾配 $\boldsymbol{v} = \nabla L$ を計算し、その逆方向にパラメータを更新する。

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \eta \boldsymbol{v} \quad (\eta > 0)$$

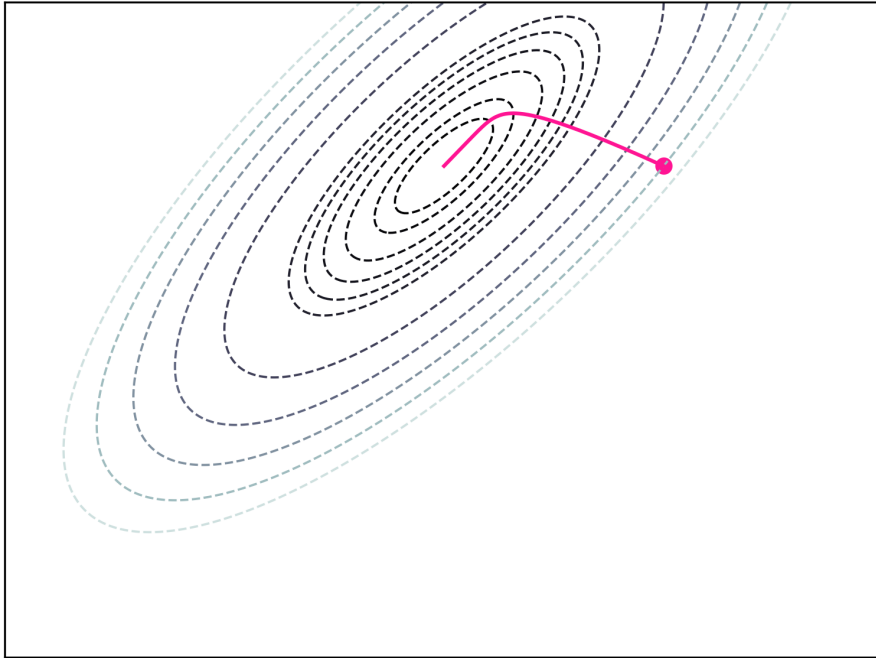
ここで、 η は**学習率**と呼ばれるハイパーパラメータであり、更新の大きさを微調整するために用いる。

そして、更新後のパラメータ $\boldsymbol{\theta}'$ を新たなパラメータとして、再び勾配を計算し、同様に更新を繰り返していく。あらかじめ決めた回数に達するまで、もしくは、目的関数の改善幅が閾値を下回るまで、更新を続ける。

学習率と収束

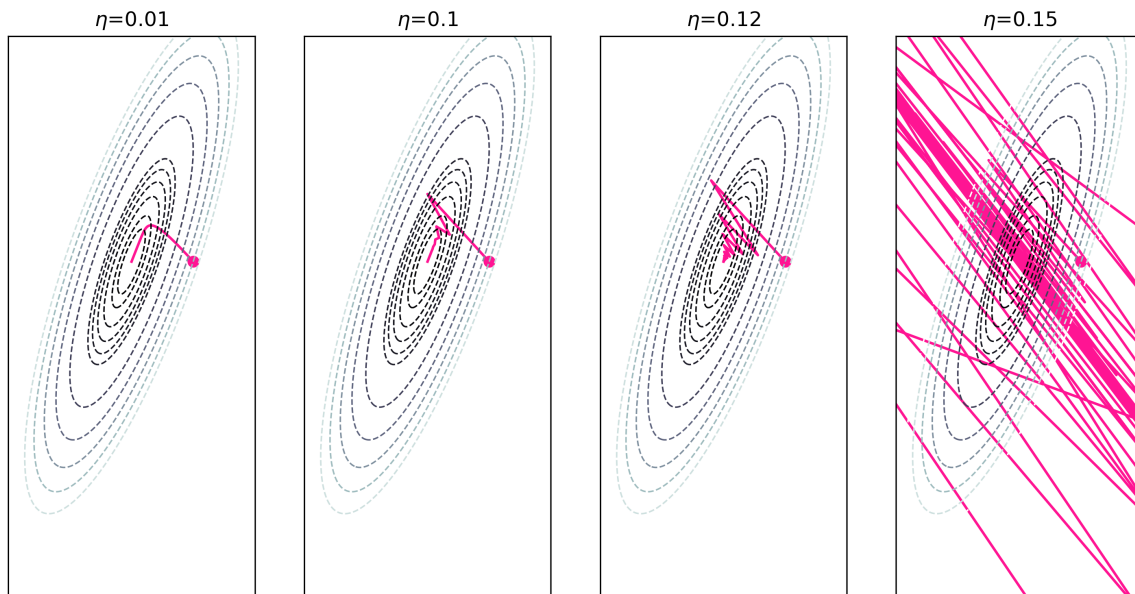
勾配は局所的な関数値の変化を表しており、遠くの地形の情報はわからない。

そのため、少しずつ山を下っていくのが勾配降下法の基本的な考え方である。



ここで、学習率 η を大きくしすぎると、一気に谷を飛び越えてしまう。

この場合、無駄な移動が発生して収束に時間がかかったり、最適解にたどり着けなくなる（収束しない）可能性がある。



一方で、学習率 η が小さければ小さいほど、狭い歩幅で山を下っていくようなもので、その分収束に時間がかかってしまう。

勾配降下法の計算コスト

勾配降下法は、すべてのパラメータ $\theta_1, \dots, \theta_n$ をまとめて更新できるため、効率が良い。

しかし、勾配を計算するためには、毎回訓練データ全体を走査する必要がある。

なぜなら、次のように、現在のパラメータについての勾配は、各訓練データについての勾配の和であるからだ。

$$\nabla L(\boldsymbol{\theta}) = \frac{\partial L}{\partial \boldsymbol{\theta}} = \frac{\partial}{\partial \boldsymbol{\theta}} \left(\frac{1}{N} \sum_{i=1}^N l(\mathbf{x}_i, y_i; \boldsymbol{\theta}) \right) = \frac{1}{N} \sum_{i=1}^N \frac{\partial l(\mathbf{x}_i, y_i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

そのため、訓練事例数が多い場合、勾配降下法の計算コストは非常に大きくなってしまう。

.....

Zebra Notes

Type	Number
todo	3
note	1