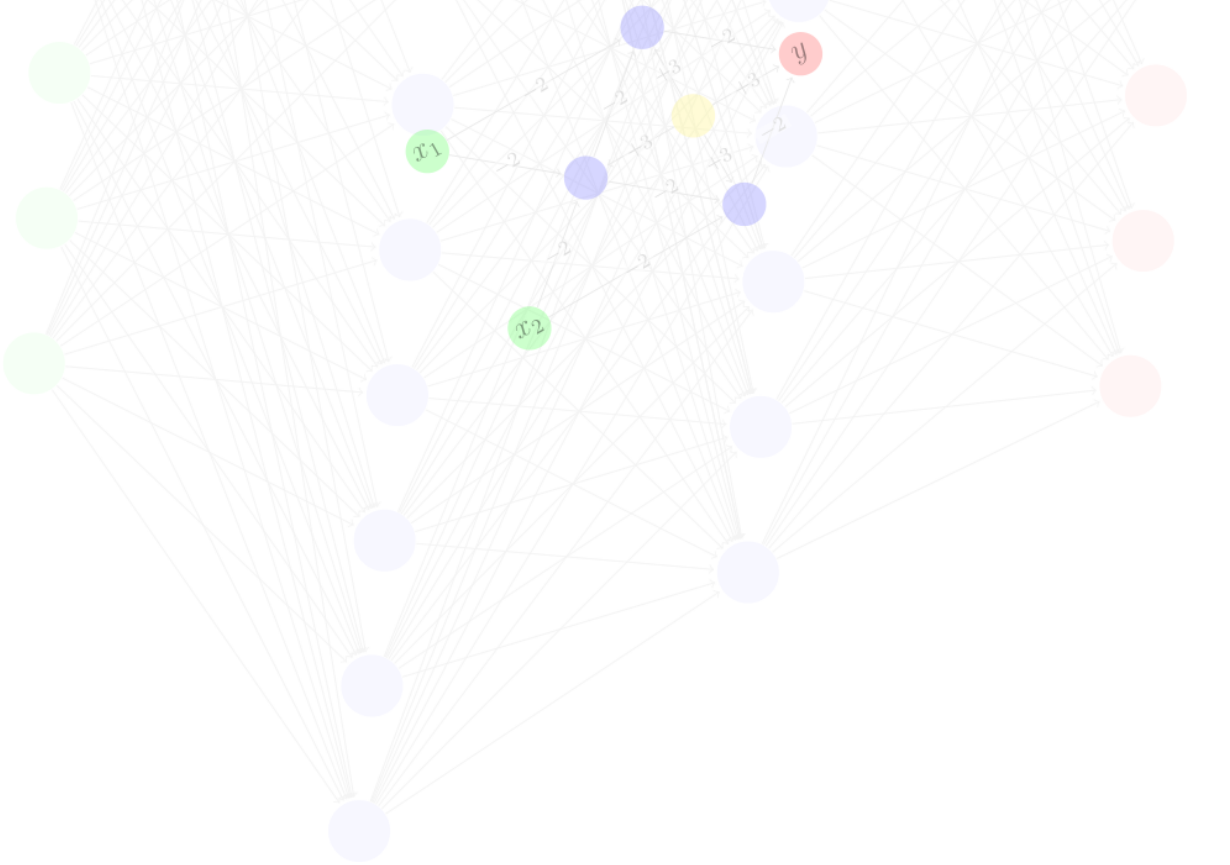
# Machine Learning

Mark K Cowan



本書は、アンドリュー・ング（スタンフォード大学）が運営するオンライン機械学習コースの2013年セッションの内容の一部に基づいている。このオンラインコースは、Courseraプラットフォーム（www.coursera.org）を通じて無料で提供されている。著者はCoursera、スタンフォード大学、Andrew Ngとは一切関係がない。

DRAFT

14/08/2013

# Contents

# Viewing

この電子書籍は本のように閲覧できるように設計されており、一度に2ページ(見開き)の別カバーペー and a
ジがある。

# Notation

行列は太字、ベクトルは矢印で示す。˜ 複数のデータを一度に処理するために行列に展開できるベクトルを太字で示す。

行列／ベクトルの関数は要素ごとに適用され、その結果はパラメータと同じ型になる。ハダマード行列積はA◦Bと表記される。行列Aのノルムは、行列を展開して生成されるベクトルのノルムである。

$$\left\|\mathbf{A}\right\|^2 = \sum_{i,j} \mathbf{A}_{ij}^2$$

# Preface

TODO

# 1 Regression

回帰は、データセットの傾向を特定し、新しいデータポイントの特性を予測するための、シンプルでありながら強力なテクニックである。これは、より複雑で強力な他の多くのテクニックのビルディングブロックを形成する[1]。

**Objective:**
Create a model:
$\mathbf{\Theta} \in \mathbb{R}^{p\times q}$, $\mathrm{g}\colon \mathbb{R} \to \mathbb{R}$
that predicts output:
$\vec{y} \in \mathbb{R}^{q}$
given input:
$\vec{x} \in \mathbb{R}^{p}$.

**Usage:**
出力~yは連続値、離散値(例えばブール値)、またはそのような値のベクトルである。入力 ~x はベクトルであり、離散値でも連続値でもよい。データ行の個々の特性を特徴と呼ぶ。通常、一度に多くの入力を処理するため、特徴ベクトルは行列を形成するように補強され、モデルはm個のデータ行に対して次のように書き換えることができる：

$$\begin{aligned} \text{Model:} \quad & \mathbf{\Theta} \in \mathbb{R}^{p\times q},\ \mathrm{g}\colon \mathbb{R} \to \mathbb{R} \\ \text{Predicts:} \quad & \mathbf{Y} \in \mathbb{R}^{m\times q} \\ \text{For input:} \quad & \mathbf{X} \in \mathbb{R}^{m\times p} \end{aligned}$$

**Discrete:**
離散的な問題では、出力のさまざまな「レベル」の間で閾値／が選択されます。これらのレベルによって定義される位相空間サーフェスは、決定境界と呼ばれる。カテゴライズの問題は、カテゴリが必ずしも論理的な順序を持たない（<と>の比較演算子に関して）ので、少し複雑です。予測されるカテゴリーc のインデックスは、$\mathrm{c} = \operatorname{argmax}_i(\tilde{\mathrm{y}}_i)$ で与えられ、~y はカテゴリー数に等しい長さのベクトルである。

## 1.1 Hypothesis

予測は仮説関数 $\mathrm{h}_\Theta(\tilde{\mathrm{x}})$ によって行われます。この関数は一般的に次のような形です：

$$\mathrm{h}_\Theta(\vec{x}) = \mathrm{g}(\vec{x}\mathbf{\Theta})$$

ここでΘは入力パラメータの線形変換を定義する行列、xは入力値の行ベクトル、g(k)はオプションの非線形伝達関数である。一般的な非線形

[1] For example, artificial neural networks (chapter 2)

伝達関数の選択肢はシグモイド関数と逆双曲線正接関数です。

$$\mathrm{g}(k) = \mathrm{sigmoid}(k) = \frac{1}{1+e^{-k}} \qquad 0 \le \mathrm{g}(k) \le +1$$
$$\mathrm{g}(k) = \mathrm{artanh}(k) \propto \ln\tfrac{1+k}{1-k} \qquad -1 \le \mathrm{g}(k) \le +1$$

y が連続的な場合，線形回帰を行うために線形伝達関数が使われます．出力が離散的な場合（すなわち、分類問題や意思決定）、シグモイド伝達関数が典型的に使用され、ロジスティック回帰を提供する。gのパラメータkがベクトルまたは行列の場合、gはパラメータに要素ごとに適用されます。つまり、g = g(k) の場合、 $g_{i,j}$ = g($k_{i,j}$ ) となります。

## 1.2 Learning

学習 ”はm個の例、つまり対応するyの値が既知である˜xのm個の値によって駆動される。仮説は$h_\Theta$ (X) = g(XΘ)（gは要素ごとに適用される）となり、学習段階はコスト関数を最小化するプロセスと定義できる：

$$\mathrm{J}_\Theta(\varepsilon), \text{using } \varepsilon = \left\|\mathrm{h}_\Theta(\mathbf{X}) - \mathbf{Y}\right\|^2$$

この非負値関数は、学習データセット[Y, X]における現在の仮説$h_\Theta$の精度を示すものであり、学習段階の目的はこの$J_\Theta$の値を最小化することである。

### 1.2.1 通常の方程式

万が一X行列が正方かつ非特異行列の場合、逆行列を使ってΘを求めることができます。その他の場合は、左の擬似逆行列を使用することができる：

$$\mathbf{Y} = \mathbf{X\Theta} \Longrightarrow \mathbf{\Theta} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y}$$

2 これはXΘ − Yを最小化する(つまり$h_\Theta$ (X) = XΘと仮定する)。逆行列は対称行列であり、一般に正半定値であるため、コレスキー分解を用いると、安定した方法で高速に逆行列を求めることができます。特異値分解(svd)も実用的である。

### 1.2.2 勾配降下

コスト関数の最小値の逐次近似は、コスト関数で定義されるランドスケープを「歩く 」ことで得られる。これは以下の反復によって達成される：

$$\mathbf{\Theta}_{i+1} = \mathbf{\Theta}_i - \alpha\vec{\nabla}\mathrm{J}_\Theta$$

[2] See figure 1.1 on page 5

ここで$\alpha$は学習率であり、学習アルゴリズムが収束する速度を定義する。$\alpha$の値が小さいと学習が遅くなり、$\alpha$の値が大きいと最小値で振動する（したがって収束しない）。$\alpha$の定数の代わりに定常的に減少する関数を用いると、最小値への収束が速くなるが、アルゴリズムが振動した場合に最小値からの最終的な距離を小さくすることができる。初期推定値$\Theta_0$は、通常、平均$\mu$ = 0の正規分布ランダム値に初期化されます。勾配降下法は、常にコスト関数の大域的最小値を見つけるわけではなく、局所的最小値に引っかかる可能性があることに注意してください。

### 1.3 データの正規化

異なる特徴がかなり異なる大きさのスケールにある場合，まず正規化が必要かもしれません．そうしないと、大きなスケールの特徴がコスト関数を支配し、他の特徴の 「学習 」を妨げてしまいます。すべての特徴を同じようなスケールに変換する簡単な方法の1つは、次のとおりである：

$$x_{normalised} = \frac{x_{raw} - \mu}{\sigma}$$

ここで、N 個の値を含むリスト x の平均と標準偏差はそれぞれ # である：

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (x_i - \mu)^2}$$

TODO
なぜうまくいくかを示すグラフ例

TODO
ティホノフ正則化について言及し、画像／フーリエ演算子の威力を示すために博士の例を使う。進化的アルゴリズムのような他のAI技術をミックスに加えたときに発生する可能性

### 1.4 正則化

これまでのソルバーは残差$h_\Theta$（X）- Yを最小化するが、その結果、学習データのノイズや誤差もフィットしようとする可能性がある。これを防ぐために、他の

のあるクールなものについて言及する。最小化するための簡単な追加項は、学習値Θの大きさである：

$$\left\|\mathrm{h}_{\Theta}(\mathbf{X}) - \mathbf{Y}\right\|^2 + \lambda\left\|\boldsymbol{\Theta}\right\|^2$$

非負の変数λは正則化パラメータであり、学習過程が主にパラメータベクトル(Θ)の大きさを最小化するかどうかを決定する。

パラメータ(ΘのΘ$_1$)の1つが定数オフセット(例：$\mathrm{X}_1^{def}$ = ˜ 1 )を表す場合、このパラメータ(および関連する重み付け)は正則化[3]から除外される：

$$\left\|\mathrm{h}_{\Theta}(\mathbf{X}) - \mathbf{Y}\right\|^2 + \lambda\left\|\boldsymbol{\Theta}_{2..N}\right\|^2$$

TODO
なぜ／バイアス特徴量が必要なのか、なぜバイアス特徴量は正則化を免れることが多いのかを説明しなさい。

正規方程式の正則化：

$$\boldsymbol{\Theta} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{Y}$$

* Remember to set $\mathbf{I}_1$ to zero if $\boldsymbol{\Theta}_1$ is an offset.

TODO
*Prove it*

勾配降下法の正則化：

$$\mathrm{J}_{\Theta} = \left\|\mathrm{h}_{\Theta}\left(\mathbf{X}\right) - \mathbf{Y}\right\|^2 + \lambda\left\|\boldsymbol{\Theta}\right\|^2$$

$\Theta_1$ がオフセットである場合、ノルムの計算から $\Theta_1$ を除外することを忘れないでください。

TODO
*Prove it*

[3] ニューラルネットワーク(第2章)のバイアスノードは定数オフセット ではなく、余分な特徴であることに注意。

## 1.5 Linear regression

伝達関数をg(k) = kとすることで、モデルは単に入力値の線形結合を表し、Θ行列によってパラメータ化される。これは（XとYに関して）線形仮説関数$h_\Theta$をもたらしますが、出力は必ずしも実際の基礎となる問題のパラメータに関して線形である必要はありません - 回帰パラメータ（x値）は、問題パラメータの非線形関数かもしれません。たとえば、2つの物理パラメータ a と b を含む問題があるとすると、a と bに関して非線形である回帰の特徴量の集合を構築することができます：

$$\vec{x} = \left(a^2,\, ab,\, b^2,\, a^3/b\right)$$

## 1.6 Logistic regression

非線形の伝達関数(例えばシグモイド関数: $g(k) = \frac{1}{1+e^{-k}}$ )を使って、非線形の仮説を生成することができる。これは、分類問題や決定問題のような離散的な出力を持つ問題で特に有用である。出力は、例えば離散値[4]を生成するために閾値関数を通すことができる：

$$\text{threshold}\,(y) = \begin{cases} \text{true} & \text{if } y \geq 0.5, \\ \text{false} & \text{if } y < 0.5 \end{cases}$$

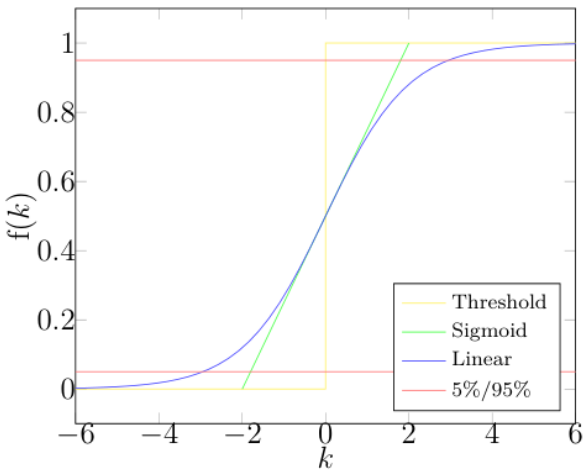



図1.1： 図1.1: シグモイド伝達関数のプロット

シグモイド関数は、[0, 1]の範囲を持ち、カテゴライズのようなyes/noの出力を持つ2値問題に最適です。正／中立／負の出力を持つ3値問題には、[-1, +1]の範囲を持つ関数がより適用可能です。そのような関数の1つがartanh関数ですが、シグモイド関数もこの新しい範囲にスケーリングすることができます。

[4] For example, yes/no

## 1.7 多クラス分類

入力ベクトル ˜x をあるカテゴリ c に分類するために，各カテゴリに対して個別の分類器を学習することができる．各個別分類器 $C_i$ は，（入力ベクトル ˜x で表現される）オブジェクト o がカテゴリ $c_i$ にあるかどうかを予測する，つまり，各個別分類器は真偽値式 $C_i$（˜x ）→ $o \in c_i$ の結果を予測する．個々の分類器は、オブジェクトが関連するカテゴリに属するか属さないかだけを予測するので、このアプローチはしばしば1対全分類と呼ばれます。多クラス分類器のコスト関数は、個々の1対全分類器のコスト関数の和です。その結果，コスト関数の勾配は，個々の分類器のコスト関数のすべての勾配の合計となる．

TODO
*Simplify and shorten, add examples*



## 1.8 Formulae

線形回帰

$$\begin{aligned} \mathrm{g}\left(k\right) &= k \\ \mathrm{J}_\Theta &= \tfrac{1}{2m}\left\|\mathrm{h}_\Theta\left(\mathbf{X}\right) - \mathbf{Y}\right\|^2 + \tfrac{\lambda}{2m}\left\|\mathbf{\Theta}\right\|^2 \\ \vec{\nabla}\mathrm{J}_\Theta &= \tfrac{1}{m}\mathbf{X}^T\left(\mathrm{h}_\Theta\left(\mathbf{X}\right) - \mathbf{Y}\right) + \tfrac{\lambda}{m}\mathbf{\Theta} \end{aligned}$$

ロジスティック回帰

$$\begin{aligned} \mathrm{g}\left(k\right) &= \frac{1}{1+e^{-k}} \text{ for yes/no} \\ \text{or} &= \tanh^{-1}\left(k\right) \text{ for positive/neutral/negative} \\ \mathrm{J}_\Theta &= -\tfrac{1}{m}\left(\vec{y}\cdot\log\left(\mathrm{h}_\Theta\left(\vec{x}\right)\right) + \left(1-\vec{y}\right)\cdot\log\left(1-\mathrm{h}_\Theta\left(\vec{x}\right)\right)\right) + \tfrac{\lambda}{2m}\left\|\mathbf{\Theta}\right\|^2 \\ \vec{\nabla}\mathrm{J}_\Theta &= \frac{1}{m}\mathbf{X}^T\left(\mathrm{h}_\Theta\left(\mathbf{X}\right) - \mathbf{Y}\right) + \frac{\lambda}{m}\mathbf{\Theta} \end{aligned}$$

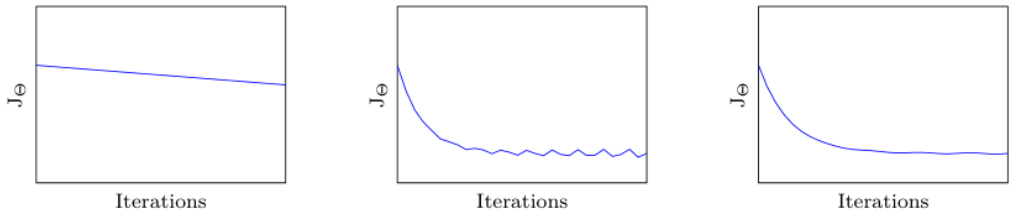
### 1.9 学習プロセスの微調整

勾配降下のような反復学習プロセスの結果は，様々な手法によって評価することができ，学習率や正則化パラメータをインテリジェントに調整するために利用できる情報を提供します．

学習率$\alpha$の不適切な選択から生じる問題は、学習曲線を見ることで特定できる。正則化パラメータ$\lambda$の不適切な値は，バイアスと分散を識別するクロスバリデーションによって診断することができる．

1.9.1 $\alpha$: 学習、収束、振動

学習プロセスが反復するにつれて、コスト関数を反復回数に対してプロットすることで、学習曲線を作成することができる。学習過程におけるコスト関数の推移は、学習率が高すぎるか、遅すぎるかを示します。理想的には、学習プロセスが完了したとき（図1.2c）、コスト関数はある値に収束するはずであり、その結果、関数は局所最小値に達する。収束しない場合は、学習プロセスが完了から程遠いことを示し（図1.2a）、振動は学習プロセスが完了しそうにないことを示す（図1.2b）。




(a) No convergence
Increase $\alpha$

(b) Oscillation
Decrease $\alpha$

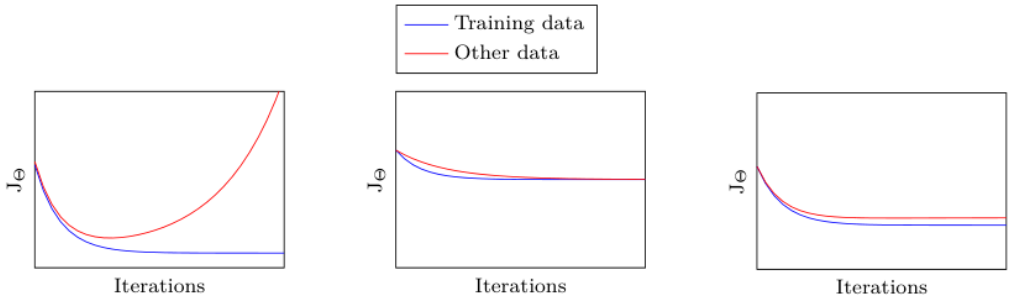
(c) Convergence
Good $\alpha$

図1.2: 様々な学習率に対する学習曲線

簡単な最適化は、$\alpha$を高い値で開始し、振動のたびに値を下げることです。これによって、コスト関数の最小値に近づくにつれて高い確率で振動が発生しますが、学習率を下げるにつれて振動が減少し、結果としてコスト関数の最小値に近い値に収束します。

### 1.9.2 λ: Bias and variance

機械学習アルゴリズムの訓練に使われるデータセットは， アルゴリズムが解決しようとする問題を完全に表現しているわけではないので， アルゴリズムが望ましい傾向に加えてデータセットの不完全さも学習することで， データに過剰にフィットする可能性がある．この場合、アルゴリズムが訓練データによくフィットするため、訓練データで評価するとコスト関数は非常に低くなる。しかしアルゴリズムを他のデータに適用すると、誤差が大きくなることがある（図1.3a）。オーバーフィットは正則化によって防ぐことができるが、これは別の問題、アンダーフィットにつながり、アルゴリズムが傾向を十分に詳細に学習しない結果、コスト関数が訓練データセットでも他のデータでも高くなる（図1.3b）。理想的には、アルゴリズムはトレーニングデータでも他のデータでも低いが同程度のエラー値で実行されるべきである（図1.3c）。

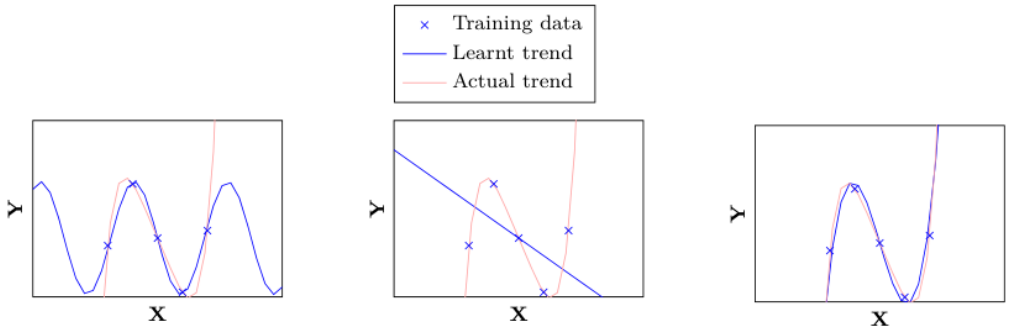



(a) Over-fit/Variance Increase $\lambda$ (b) Under-fit/Bias Decrease $\lambda$ (c) Good

Figure 1.3: Learning curves for various learning rates



**Example:**
5次の多項式を例にとり、多項式中の4[5]個の既知点を用いて(勾配降下を 用いて)線形回帰システムを学習する。多くの特徴量（例えば10次までの単項式）を使って訓練すると、図1.4aに示すように訓練データと完全に一致するか、それに近いものを見つけることができますが、この学習された傾向は実際の傾向と大きく異なる可能性があります。逆に、特徴量が少なすぎたり、正則化が高すぎたりする学習では、図1.4bに示すように、学習データセットまたは将来のデータのいずれにも等しく一致しない学習傾向が得られます。図1.4cの良い適合は、5次の多項式を記述するには4点だけでは情報が不十分であるため、完全にトレンドに沿ったものではありませんが、妥当な一致です。もちろん、実際のトレンドが正弦波であれば、「良い」フィットは高い誤差を持つでしょう。例えば、正弦波特徴、単項式特徴、指数関数特徴、またはそれらの組み合わせを使用するのでしょうか？




(a) Variance: many features and low regularisation (b) Bias: few features or high regularisation (c) Good

図1.4： 図1.4: オーバーフィット、アンダーフィット、良いフィットの例

クロスバリデーション：

If the actual trend were known, then there would be no need for machine learning. In

[5] これは多項式を一意に記述するのに必要な数より2つ少ない。

実際の問題では、学習されたトレンドは実際のトレンドと直接比較することはできません。その代わりに、トレーニングデータセットをトレーニングセットとクロスバリデーションセットの2つのサブセットに分割することができる。そして学習アルゴリズムを訓練集合に適用し、正則化パラメータを変化させて、クロスバリデーション集合の誤差[6]を最小化する（下記参照）。したがって、正則化の量は、当てずっぽうではなく、インテリジェントに決定することができる。図1.3では、「他のデータ 」系列がクロス・バリデーション・データセットのコスト関数曲線になる。

$$\boldsymbol{\Theta}_{good} = \underset{\Theta}{\operatorname{argmin}}\ \mathrm{J}_{\Theta}\left(\lambda, \mathbf{X}_{train}\right)$$
$$\lambda_{good} = \underset{\lambda}{\operatorname{argmin}}\ \mathrm{J}_{\Theta}\left(\lambda, \mathbf{X}_{cross}\right)$$

TODO
検証データセットと例

訓練セットの成長

バイアス／分散を診断するもう1つのテクニックは、訓練セットと固定クロスバリデーションセットのコスト関数を、訓練セットのサイズを変えながらプロットすることです。訓練データを増やしても、バイアスに起因する誤差は減りませんが、分散に起因する誤差は減ります。したがって、システムが高いバイアスに悩まされている場合、トレーニングセットとクロスバリデーションセットの誤差はともに高い値に収束する（図1.5a）。その代わりにシステムが高い分散に悩まされている場合、誤差は2つの別々の値に収束し、その間に大きなギャップがあるように見えますが（図1.5b）、十分な大きさと多様性のある訓練セットが使用されていれば、最終的には収束します。したがって、正則化パラメータ(λ)を増加させる代わりに、可能な限り多くのトレーニングデータを取得することで、オーバーフィットを排除することができる。


(a) Variance (b) Bias (c) Good

Figure 1.5: 訓練セットのサイズを変えることによるバイアスと分散の識別

**Summary:**
オーバーフィットは、システムが少なすぎる訓練データから多すぎる推論を行おうとし ていると解釈できる。一方、アンダーフィットは、システムが訓練データを非効率的に利用 している、またはシステムが有用な仮説を作成するのに十分な訓練データがない と解釈できる。

[6]The value of the cost-function, $\mathrm{J}_{\Theta}$

したがって、正則化の量を調整するか、利用可能な訓練データの量を変更することで、どちらも修正することができる。学習データ量を増やすには、新しいデータ／特徴量[7]を収集するか、既存の要素に指数演算（$\tilde{x}_7 = \tilde{x}_1^2$ ）、乗算、除算（$\tilde{x}_8 = \tilde{x}_4 \ / \tilde{x}_5$ ）などの非線形演算を適用して新しい特徴量を生成する。

| **High bias** | **High variance** |
|---|---|
| Decrease $\lambda$<br>より多くの特徴を得る<br>より多くの特徴を作成する | Increase $\lambda$<br>より少ない特徴を使う<br>訓練データセットを成長させる |

Table 1.1: Cheat-sheet for handling bias and variance

### 1.9.3 精度、精度、F値

TODO
*This section is high priority*

[7]Recall that features are properties of a data item

# 2 Neural networks

ニューラルネットワークは複雑な非線形モデルであり，個々のコンポーネントは回帰モ デルと同様の振る舞いをします．グラフ[1]として可視化することができ、論理ゲート[2]に似た振る舞いをする部分グラフも存在する。ニューラルネットワークの構造はあらかじめ明示的に設計されていますが、仮説を生成するためにネットワークが行う処理（したがって、ネットワーク内のさまざまな論理ゲートやその他の処理構造）は、学習プロセス中に進化します[3] 。このため、ニューラルネットワークは、明示的に設計してコード化しなければならない典型的なアルゴリズムとは対照的に、「自らプログラムする」ソルバーとして使用できる。
これは、入力ノードを設定し、すべての出力ノードが完全に計算され るまで、ネットワーク内の接続を通じて値を伝播させることであ る。学習は勾配降下を使って達成することができ、出力ノードの誤差は、隠れノードの誤差を推定するために、バックプロパゲーションによってネットワークを通して押し戻され、コスト関数の勾配を計算することができる。

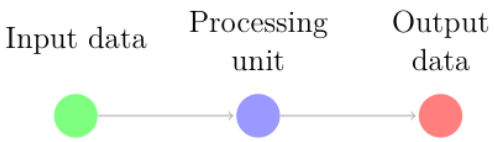



図2.1： グラフとして視覚化された情報処理ユニット

## 2.1 ビルディングブロックとして可視化された回帰

Linear regression may be visualised as a graph. The output is simply the weighted sum of the inputs:

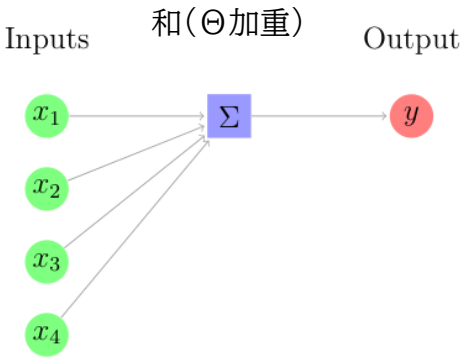



Figure 2.2: Linear regression

[1]For example, figure 2.6
[2]For example, figure 2.10
人工ニューラルネットワークの概念が派生した生物学的ニューラルネットワークに類似している。

同様に，ロジスティック回帰は，伝達関数を表現する1つのノードが追加されたグラフとして可視化できる．ロジスティック回帰要素は，ロジスティック回帰要素の最初の2つのステージが線形回帰要素を形成することを認識することによって，線形回帰要素と伝達ノードを用いて記述することもできる．

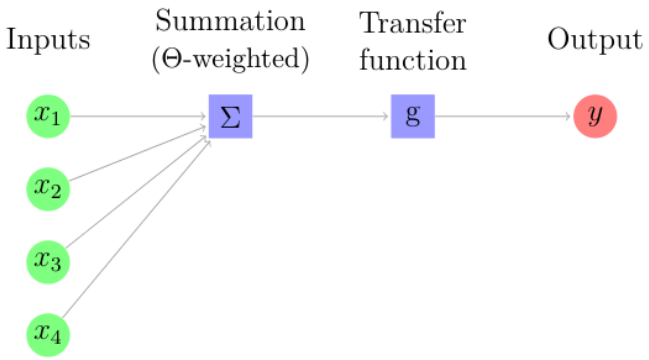



Figure 2.3: Logistic regression



パイプラインの最後の3つのステージは、最初のステージにのみ依存するので、出力の1つの非線形混合操作に凝縮する：

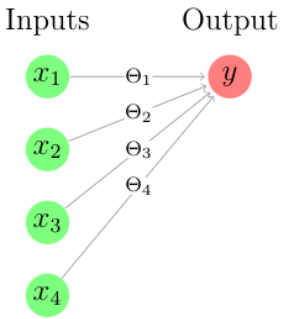



図2.4: ロジスティック回帰プロセスの単純化された解剖学

この表記法を使用すると、複数の1対全分類器を介して分類を実行するネットワークは、次のような形になる。ここで、パラメータベクトルΘは、出力ベクトルの各列を生成するために別々の列を持つパラメータ行列Θを形成するように結合されている：

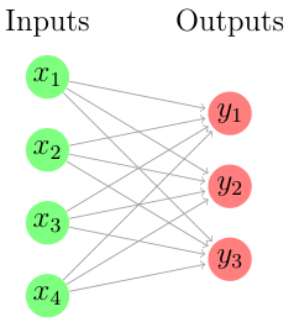



図2.5: 多クラス分類ネットワークの単純化された構造

## 2.2 隠れ層

ロジスティック回帰は強力なツールであるが、入力値の線形結合（できるだけ早く非線形関数を適用するとはいえ）で動作するため、単純な仮説しか形成できない。ニューラルネットワークは、そのような非線形混合要素の層から構築され、より複雑な仮説の開発を可能にする。これは、より複雑な振る舞いを生み出すために、[4]ロジスティック回帰ネットワークを積み重ねることによって達成される。入力ノードと出力ノードの間に余分な非線形混合ステージを含めると、ネットワークの複雑さが増し、より高度な仮説を開発できるようになる。これは比較的簡単である：

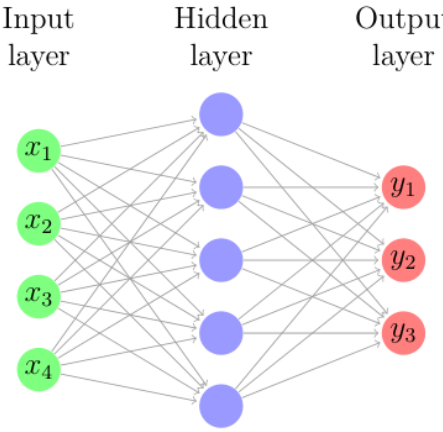



図2.6: 隠れ層が1つの単純なニューラルネットワーク

各ロジスティック回帰要素は入力の線形結合を変換し、線形結合の線形結合はそれ自体が線形結合[5] であるためである。

## 2.3 ニューラルネットワーク要素の表記法

L 層からなるネットワークの l 層目の j 番目のノード(またはニューロン)への入力値を $z_j^l$ とし、ノードの出力値(または活性化)を a $l_j$ = g($z_j^l$ ) とする。第 l 層（$a^{l-1}$ から $z^l$ を生成）のパラメータ行列を $\Theta^{l-1}$ とする。第1層（入力層）の活性化は、ネットワークの入力値によって与えられる： $a^1$ = ˜x . 最後の層(出力層)の活性化はネットワークの出力である: $a^L$ = ˜y .

すなわち、複数の線形結合層を組み合わせて、1つの線形結合要素を与えることができる。

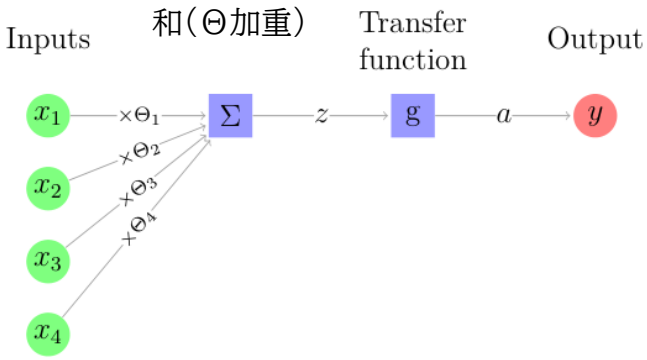



Figure 2.7: Θ, g, $z$ and $a$ in a neural network element

## 2.4 Bias nodes



通常、各レイヤーはオフセット項を含み、これはある定数値（例えば1）に設定される。便宜上、$a_0^l = 1$となるように、インデックス0を与える。各層には別々のパラメータ・ベクトルがあるので、これでΘ行列のセットができたことになる。このようなネットワークの構造と表記法を説明するために、いくつかの隠れ層を持つバイアスされたネットワークを以下に示す：

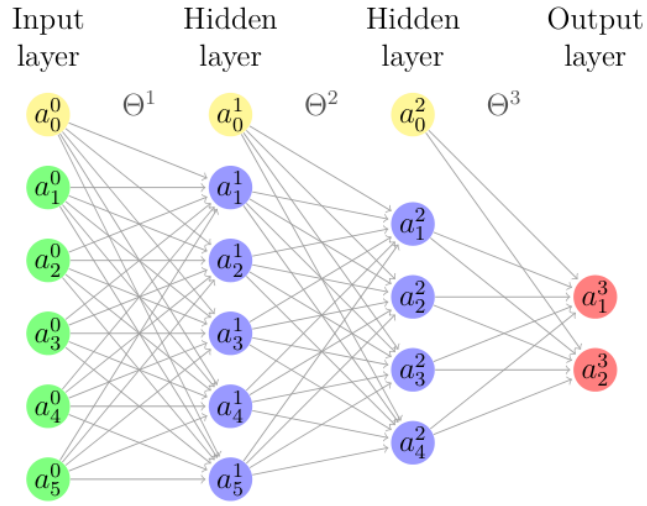
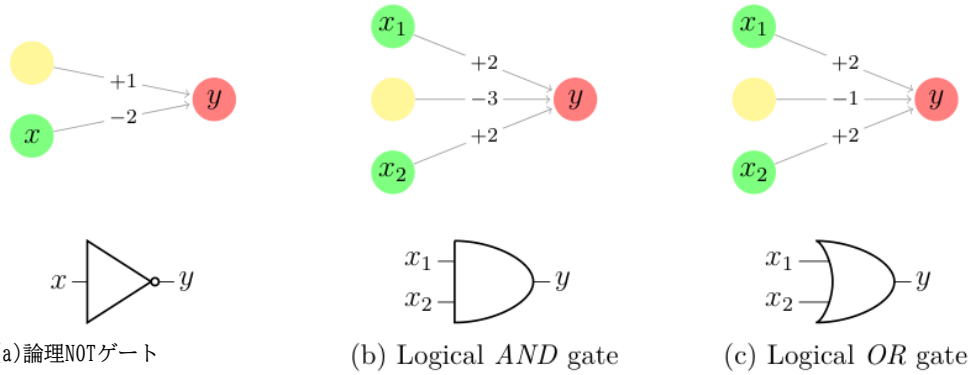



Figure 2.8: Example of the notation used to number the nodes of a neural network

## 2.5 Logic gates

複数の層が存在することで、すべての初等論理ゲートを構成することができる。そしてこの構築は学習段階で自動的に行われる。以下にいくつかの例を示す。0／1を入力とし、真なら正の出力、偽なら非正の出力を返す：




(a)論理NOTゲート (b) Logical *AND* gate (c) Logical *OR* gate

図2.9： ニューラルネットワークとしての初等論理ゲート

これらから他のゲートを構成するのは簡単である。Θの値を否定すると反転ゲートが生成され、これらを使ってより複雑なゲートを構築することができる。したがって、ニューラルネットワークは、デジタル処理とアナログ処理の両方が可能な「自己設計マイクロチップ」と理解することができる。

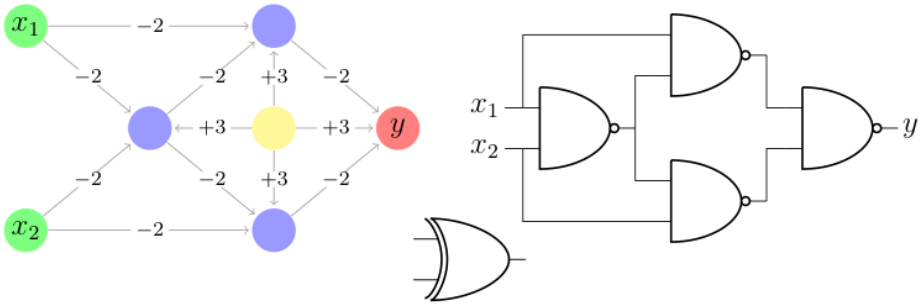



図2.10: 4つのNANDゲートから構成される論理XORゲート

## 2.6 Feed-forward

入力データ˜xに対して仮説$h_\Theta$（˜x ）を評価するために、データは各層を通過する：

$$\begin{aligned}
\vec{z}^{l+1} &= \vec{a}^{l}\boldsymbol{\Theta}^{n} && \text{— input value of neuron}\\
\vec{a}^{l+1} &= \mathrm{g}\left(\vec{z}^{l+1}\right) && \text{— activation of neuron}\\
\vec{a}^{1} &= \vec{x} && \text{— network input}\\
\vec{a}^{L} &= \mathrm{h}_\Theta\left(\vec{x}\right) && \text{— network output}
\end{aligned}$$

## 2.7 Cost-function

コスト関数はロジスティック回帰と同じである：

$$\mathrm{J}_\Theta = -\tfrac{1}{m}\left(\vec{y}\cdot\log\left(\mathrm{h}_\Theta\left(\vec{x}\right)\right)+\left(1-\vec{y}\right)\cdot\log\left(1-\mathrm{h}_\Theta\left(\vec{x}\right)\right)\right)+\tfrac{\lambda}{2m}\sum_{l=1}^{L}\left\|\boldsymbol{\Theta}^{l}\right\|^{2}$$

これは個々の1対全分類器のコスト関数の合計を表す。ニューラルネットワークは入力ノードと出力ノードの間に隠れ層を持つので、学習プロセスはロジスティック回帰の多クラスネットワークよりも若干複雑である。

## 2.8 バックプロパゲーションによる勾配

様々なパラメータ行列に対する）コスト関数の勾配は，出力の誤差をネットワークに逆伝播することで計算される．

$$
\begin{aligned}
e^L &= \mathrm{h}_\Theta(\mathbf{X}) - \mathbf{Y} && \text{— error at the final layer} \\
e^l &= \left(e^{l+1}\,\boldsymbol{\Theta}^{l^T}\right) \circ \mathrm{g}'(z^l) && \text{— error at each prior layer} \\
\mathrm{g}(k) &= \tfrac{1}{1+e^{-k}} \rightarrow \mathrm{g}'(k) = \tfrac{\mathrm{g}(k)}{1-\mathrm{g}(k)} && \text{— transfer function and derivative}
\end{aligned}
$$

各ノードにおける誤差が計算されることで，各ノードが全体的なコストに寄与する量が推定され，勾配が導かれます．

$$
\begin{aligned}
\Delta^n &= a^{n^T} e^{n+1} && \text{— error contributed to the next layer} \\
\vec{\nabla}_{\Theta^n} \mathrm{J}_\Theta &= \tfrac{1}{m}\Delta^n + \tfrac{\lambda}{m}\boldsymbol{\Theta}^n && \text{— gradient } \tfrac{\partial \mathrm{J}_\Theta}{\partial \Theta^n}
\end{aligned}
$$

ここで示されている勾配には正則化項が含まれていることに注意してください。また、バイアス・ノードの活性値は一定であるが、バイアス・ノードの重み($\Theta^l_{0,j}$)は一定ではないので、バイアス・ノードが寄与する誤差は必ずしもゼロ[6]ではない。従って、バイアスの量を最小化することが有効であるような稀なケースを除き、1.4節で説明したように、バイアスノードの重みは正則化から除外されるべきである。

TODO
*Real-world examples*

[6]Therefore the gradient of the cost-function with respect to the weights of these nodes may also be non-zero

# List of Figures