かしこい機械の育て方

reyk(@reyk429)

目次

第 I 章

はじめに

こんにちは。68 回生の reyk と申します。宗教上の理由により部誌に寄稿するのは今年で初めてなのでほとんどの方は初めましてですね。最近になってハンドルネームを変えたので初めましてではない方も初めましてかもしれません。

さて、この記事のテーマは「機械学習」です。みなさんも、最近流行りの「ビッグデータ」というフレーズ、一度は耳にしたことがあると思います。Google の検索エンジンをはじめ、Amazon の商品推薦システムとか、NHK の twitter トレンドとか、ああいうやつです。しかし、耳にしたことがあっても、その中身・仕組みが何なのかよく知らない、という人も少なくはないと思います。この記事では、「機械学習」というものに少し足を踏み入れて、ビッグデータというものの裏でどのような技術が使われているのか、ということを紹介していきたいと思います。

第 II 章

機械学習入門

1　奇怪な機械

「機械学習」というワードが突然出てきましたが、学習する機械、と聞くと少し奇妙な感じがするかもしれません。機械学習という字面から何となく意味は分かりそうですが、その意味するところは何なのでしょうか。例によってとりあえず Wikipedia 先生 [11] に聞いてみましょう。

> 機械学習（きかいがくしゅう、英: machine learning）とは、人工知能における研究課題の一つで、人間が自然に行っている学習能力と同様の機能をコンピュータで実現しようとする技術・手法のことである。

... 字面以上の情報はあまり得られなかったみたいなので、具体例を挙げて説明します。

人間は、日頃から無意識のうちに多くのことを学習しています。例えば、普段何気なく行っている、歩く、つかむ、といった動作も、もちろん生まれてすぐできたわけではありません。繰り返し動作を試み、失敗、成功を積み重ねることで、次第に「コツ」のようなものを身につけ、失敗することなくできるようになるのです。これも一種の学習です。「文字を読む能力」や「音・声を聞き分ける能力」などの認識能力、「ボールの飛ぶ方向を予測する能力」や「相手の感情・反応を推測する能力」といった予測能力についても、認識・予測の「コツ」、なんとなくの「パターン」を学習によって見つけることで可能となっているのです。機械学習とは、このような人間の学習プロセスを真似ることで、機械に認識能力や予測能力を身につけさせよう、という試みなのです。

さて、人間の学習プロセスを真似する、ということですが、具体的にはどのように行うのでしょう。子供に勉強を教える、という例と比較して考えると、

1. 問題を与えて解かせる
2. (子供|機械) から返ってきた答えを正解と比較する

3. 答え合わせをもとに、(考え方|処理方法) を修正させる

という感じになります。

機械学習というのは基本的にこのようなプロセスで行われます[1]。

ただ、人間の高度な学習能力は未だ完全に解明されているわけではなく、それを機械で実現するのは難しいです。そのため、そこは機械の処理能力を生かして、才能を努力で補うように学習能力の低さをデータの量で補います[2]。ここがビッグデータと呼ばれる所以です。

まとめると、機械学習では、機械に与えるべき大量の問題集であるデータを集め、機械の問題に対する処理方法のモデルを作ることが課題となります。

ところで、このまま抽象的な感じで行くと終始ふわふわした説明だけになってしまいそうなので、以下では問題を「文字認識」ということに限定し、実際に文字認識のプログラムを作る際の流れに従って話を進めていきたいと思います。途中のソースコードは python で記述しています。

2　ネットで得たデータ

機械に問題を解かせることを考える前に、まずは解かせるための問題、つまりここでは文字の画像と、正解である文字の種類のラベルが必要です。自前で用意しよう、と思うかもしれませんが、いっぱい集めるのは一苦労です (特にラベル付け)。特に機械学習においては、前述の理由からデータは多ければ多いほど良く、またある程度の精度を実現するためには最低でも 10000 くらいのデータは必要です。

しかし、時代は 21 世紀、インターネット上には、文字画像や人の顔の画像、天気のデータから株式市場のデータに至るまで、様々なデータが保存されています。機械学習の際にはこうしたデータを利用すると便利です。今回は ETL 文字データベース [4] のひらがな約 6000 データ、カタカナ約 10000 データを使わせて頂きました。このデータセットにはその画像がどの文字であるか、というラベルもあらかじめ含まれています。

ところで、データを全て学習用に使ってしまっては、学習をさせた後にちゃんと学習が成功したのか、ということを評価することができなくなってしまいます。よって通常はデータを学習用のデータとテスト用のデータの二つに分け、学習後にテストを行うことで学習結果を評価します。この学習用のデータを**訓練用データ**、テスト用データを**評価用デー**

[1] 正確にはこのように正解と比較して処理方法を修正するような学習方法は「教師あり学習」と呼ばれ、それに対して正解の与えられないクラスタリングなどの「教師なし学習」という学習方法も存在します。

[2] 現在の機械であれば 1 秒間に 10^8 回程度の計算が可能なので、処理量にもよりますが 1 秒間に 100 ～ 100000 程度のデータは処理できます。

タと呼びます。今回は全データの 5% を評価用データとして使うこととします。

3　口調という特徴

特徴抽出―概略編

さて、次は問題の解き方、文字の認識方法について考えます。機械学習の定義に従い、人間の学習方法について考え、それを機械に適用することを考えます。

文字だと少し分かりづらいので、初めに顔認識について考えてみましょう。私たち人間は人の顔を見て、どのようにその人が誰であるかを認識しているのでしょうか。髪の色・長さ、眉毛の太さ、目の大きさ、鼻の高さ、顔の形... これらの判断材料を総合的に評価し、あらかじめ頭の中にたくさん保存してある顔のイメージのうち最も似ている顔を、その人の顔だと認識し、その顔に関連付けられている人をその人だと認識するのです。ここではこれらの判断材料のことをまとめて**特徴**と呼び、一つ一つの判断材料を特徴の成分と呼ぶことにします。

文字の話に戻りましょう。文字にはどのような特徴成分があるでしょうか。線の太さ、線がつながっているかなどは、書く人によって変わるため、あまり関係なさそうに思えます。そうすると、文字の特徴は文字の形ぐらいしかなさそうです。ただ、文字の形といってもそれは一言で表せるものではありません。それぞれの成分は機械に理解できる形式、つまり数値に変換しなければなりません。よって最終的に特徴はいくつかの数値の組として表されることになります。

単純化して数値に落とし込むために、とりあえず画像を小さい格子単位に分割します。さらにその中の線分を単純化して、その線分が大体どの向きであるか、というのを表すために、次の図のような単純な方向成分に分解することを考えます。

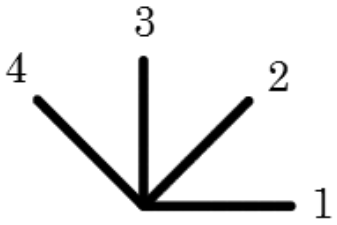

図 II.1

このそれぞれの方向成分の長さ (強さ) を数値化すれば、文字の形の情報を数値の組で表

すことができそうになってきました。

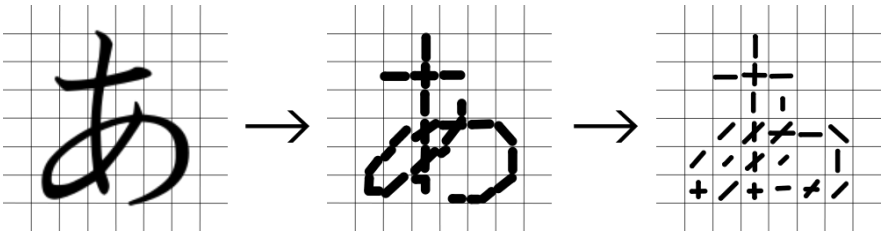

図 II.2　数値化の流れ

このようにして、元の文字の形の情報を (格子の数)×4 個の数値として表すことができました。このように元のデータから特徴を表す数値を計算する過程を**特徴抽出**と言います。

特徴抽出—実装編

文字認識におけるこのような特徴抽出の方式を**加重方向指数ヒストグラム方式**と呼びます。概略編では詳細を省略しましたが、実際の実装では参考文献 [6] に従い、以下のような手順で特徴抽出を行いました。

1. 画像の整形 (2 値化[3]、ノイズ除去[4])
2. 画像の大きさの正規化[5]
3. 輪郭の平滑化
4. 輪郭線抽出
5. 分割化と各格子内の各方向成分の抽出
6. 加重方向指数ヒストグラムの計算

図 II.3　画像の整形から輪郭線抽出までの流れ

形の情報は各格子内の方向成分として表現しているため、画像の大きさを正規化して文字の位置を合わせる必要があります。

[3] 画像の色を白と黒だけにする
[4] 文字以外の不要な情報の除去
[5] 文字が画像にぴったり収まるように調整し、大きさを揃える

また輪郭線以外の部分は方向を検出できないため、輪郭線のみについて方向成分を計算しています。これによって同時に太さの情報を無視することができます。

方向成分の抽出では、斜めも含めて隣接する黒画素についてその方向を方向成分として抽出しています。

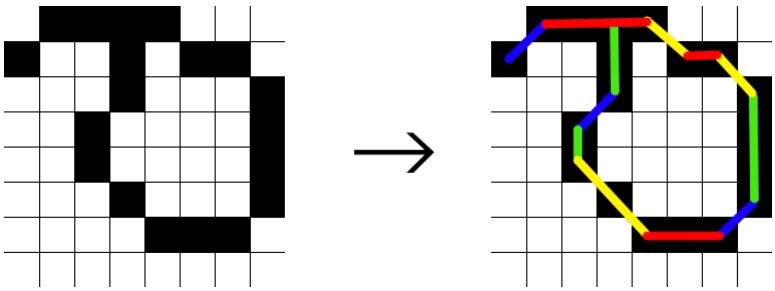

図 II.4　方向成分抽出。この図では図 II.1 の番号付けを使うと (成分 1, 成分 2, 成分 3, 成分 4) = (6, 3, 6, 4) となる

各格子 (ここでは 7×7) の方向成分を抽出した後は、適切な重みづけ[6]をして周辺の格子の方向成分を足し合わせることで、データごとに多少位置がずれても大幅な数値のずれが生じないようにします。

以上の操作をして代表のマスを 4×4 個選ぶことにより、最終的に $4 \times 4 \times 4$ 個の数値として文字の形を表現することができます。

4　顔の detail が似ている

類似度を定義する

前節で人間の認識方法について述べたことをもう少し詳しく説明すると、人間は頭の中でインスタンス[7]をいくつかのカテゴリーに分類していて、新しいインスタンスに出会った際には、似ているインスタンスが属しているようなカテゴリに分類している、と言うことができそうです。文字認識であれば各文字データがインスタンス、文字の種類がカテゴリに当たります。この分類方法を機械に適用することを考えます。(このように分類の仕方を学習する機械のことを特に**分類器**と呼びます。)

そこで、インスタンス同士が似ている度合い、類似度について考えてみます。もし、特徴を表す数値が 1 つなら、数値が近いほど似ていると言えるのは明らかです。この自然な拡張として、数値が 2 つ、3 つと増えていっても、特徴同士の近さ、距離によって類似度が

[6] 参考文献 [6] ではガウスフィルターを用いています。

[7] ある一人の顔や一つの文字など一つの具体的な認識対象をここではこう呼ぶことにします。

定義できそうに思えます。

ここで、2 つの N 個の数値の組 $(A_1, A_2, ..., A_N), (B_1, B_2, ..., B_N)$ の距離を次のように定義します。

$$Distance(A, B) = \sqrt{\sum_{k=1}^{N}(A_k - B_k)^2}$$

これも 1 次元、2 次元での距離の定義の自然な拡張といえます。以下ではインスタンスの類似度としてその特徴を A,B としたときの $Distance(A, B)$ を用いることとします[8]。

さて、インスタンス同士の類似度が定義できたところで、分類すべきカテゴリはどのように判断すればよいでしょうか。「似ているインスタンスが属しているようなカテゴリ」の解釈によって、色々な方法が考えられますが、ここでは

1. 新しいインスタンスとの類似度の小さい順から複数のインスタンスのうち、最も多くのインスタンスが属しているようなカテゴリに分類する

という考え方に従って文字の分類を行ってみたいと思います。

K 近傍法による分類

新しいインスタンスに対して、今までに入力された全てのインスタンスとの類似度を計算し、似ているものから K 個を取り出します。その中で最も多くのインスタンスが属しているようなカテゴリに、新しいインスタンスを分類します。なお、下のソースコードでは簡潔さのために一部実装の詳細を省略した部分があります。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from random import shuffle

#距離関数の定義
def Distance(a,b):
    s = 0
    for i in range(len(a)):
        s += pow(a[i] - b[i], 2)
    return s
```

[8] 類似度が小さいほど似ている、ということに注意してください。

```
    #カテゴリは [0,56) の数字で表されており、
    #それぞれ JIS X 0201 におけるカタカナの文字コードに対応している ([166,222))

    if __name__ == '__main__':
    #   feature_list = [(特徴のリスト, ラベル) in 全てのデータ]
        shuffle(feature_list) #ランダムに訓練用データと評価用データに分ける
        N = len(feature_list) #データセット数
        K = 20
        correct = 0 #正答数
        #番号 [0, N - N/20) のデータを訓練用に、
        #番号 [N - N/20, N) のデータを評価用に用いる
        for index in range(N - N/20, N):
            #全ての距離 (類似度) を計算
            distance = [(Distance(feature_list[index][0], feature), label)
                        for (feature, label) in feature_list[0, N - N/20]]
            distance.sort() #類似度の小さい順にソート
            appeared = [0] * 56 #各カテゴリの出現回数
            for i in range(K):
                appeared[dis[i][1]] += 1
            #最も多く現れたカテゴリを採用
            result = appeared.index(max(appeared))
            #正答数のカウント
            if result == feature_list[index][1]:
                correct += 1
        print correct, '/', N/20
```

実行結果は以下のようになりました。

データセット	正答サンプル数	全サンプル数	正答率
ひらがな (ETL4)	268.6	305	88.1%
カタカナ (ETL5)	513.6	530	96.9%

どちらのデータセットでもかなり高い正答率が実現されていることが分かります。

以上のような分類法を **K 近傍法**と呼びます。

K 近傍法の問題点

前節では距離によって類似度を定義し、それに基づいた K 近傍法という分類法を紹介しました。今回の場合はこの方法でうまくいきましたが、この K 近傍法にはいくつか問題点があります。

まず、K 近傍法は訓練データ全てとの距離の計算を行うため、計算量がかなり大きいです。適宜枝刈りをして計算量を減らすことは可能ですが、訓練データを全て保存しておく必要があるのは変わりません。これは訓練データが多いほど学習能力の上がる機械学習においては中々厳しい制約です。

次に、成分の重みづけが違う場合に、あまり良い結果が得られない、ということがあります。例えば、人の特徴として、身長と体重を考えた時に、身長を m、体重を kg で表したとします。すると、(身長, 体重) = $(1.7, 62.3), (1.7, 63.3), (2.7, 62.3)$ という 3 人の特徴を考えた時、1 人目と 2 人目、1 人目と 3 人目の類似度はどちらも 1.0 となり、同じだけ似ているということになってしまいます。これがおかしいのは明らかでしょう。

さらにもう一度人の特徴を例にとって考えてみます。(身長 (cm), 体重 (kg)) = $(170, 70), (170, 100), (170, 130)$ という 3 人の特徴について、距離で考えると 1 人目と 2 人目、2 人目と 3 人目の類似度は同じになりますが、人間なら肥満度という点から考えて 2 人目と 3 人目の方が似ていると考えるのが普通でしょう。このように、距離による類似度は成分同士の比率による類似度などをうまく表現できないという問題点があります。

では、どのようにすれば改善できるのでしょうか。一つには、データを正規化する[9]、コサイン類似度を用いることなどが考えられます。特徴空間を平面によって 2 つのグループに分割する SVM という手法も有名ですが、これらは今回は説明しません。次の節では、幅広い応用の効く**ニューラルネットワーク**というものを紹介して、この記事を締めくくることとします。

5　ニューロン導入論 ～ 脳 gain, no pain～

ニューラルネットワーク導入

ここで一度、機械学習の定義に戻って考えてみましょう。機械学習とは、人間の学習プロセスを機械に真似させることで、問題を解こうとする試みでした。これを突き詰めて、

[9] 数値を 0 ～ 1 に揃えること

学習プロセスにおいて人間の脳みその中で起こっていることを機械で表現してみよう、というのが、これから説明するニューラルネットワークの考え方です。

まず、人間の脳みその働きは、以下のようなニューロンという神経細胞によって成り立っている、ということが解明されています。

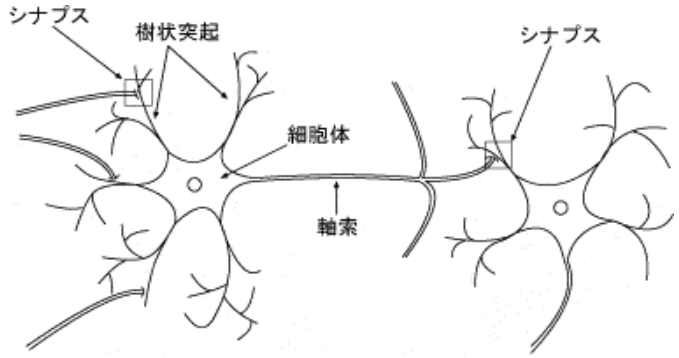

図 II.5　ニューロン

これをモデル化すると次のようになります。

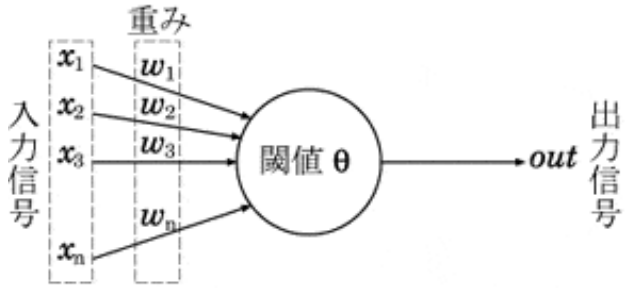

図 II.6　ニューロン模式図

この図は、一つのニューロンが辺の重みによって重みづけされた他のニューロンの出力を入力として受け取り、その合計がある閾値 θ を超えると、何らかの信号が出力される、という過程を表しています。このような繋がりが集まって一つの巨大なネットワークを形成しているのです。

これは機械においては、**辺 (edge)** によって繋がる**頂点 (node)** として表現することができます。信号の伝達については、辺によってつながった他の頂点の出力を $X_1, X_2, ..., X_n$、それぞれの入力辺の重みを $W_1, W_2, ..., W_n$ とし、頂点の内部関数を f として、

$$out = f(\sum_{k=1}^{n} X_k W_k)$$

として出力 out を定義します。ここで内部関数 f は、次のシグモイド関数と呼ばれるものとします。この理由は後述します。

$$f(x) = \frac{1}{1 + e^{-x}} \tag{II.1}$$

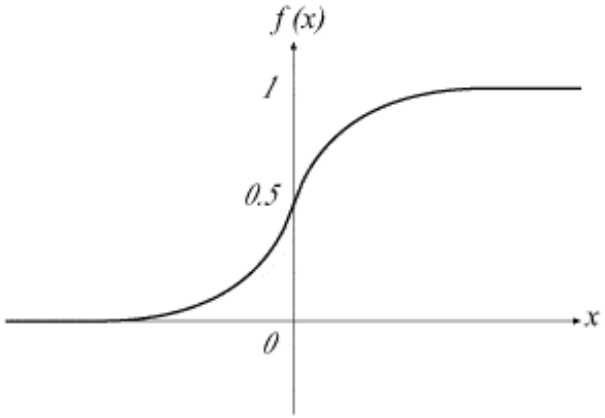

図 II.7　シグモイド関数

さらに、ニューロンの繋がりを単純化し、頂点たちが層をなしていると考え、それぞれの頂点を**入力層**、いくつかの**隠れ層 (中間層)**、**出力層**に分けます。また信号の伝わる方向は一方向のみで、内部ループは存在しません。このようなネットワークモデルを**多層パーセプトロン**と呼び、今回は特に一つの隠れ層と入力層、出力層からなる 3 層パーセプトロンを扱います。

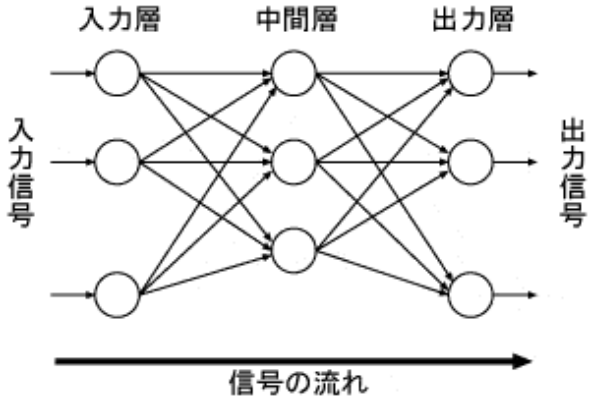

図 II.8　3 層パーセプトロン

入力層の頂点に数値を入力すると、上記の式に従ってそれぞれの頂点の入力値・出力値が計算され、出力層の出力値が全体としての出力となります。機械学習では、インスタンスの特徴の各成分を入力頂点に、各カテゴリを出力頂点に割り当て、特徴を入力すると各カテゴリのスコア[10]が算出されるようにします。この数値の最も高いカテゴリが、このインスタンスの属するカテゴリと判定される、ということになります。

最初の辺の重みづけはランダムに割り当てます。そのため、出力される数値に特に意味はありません。訓練データを入力する度に、出力された数値を正解 (そのインスタンスがどのカテゴリに属するか) と比較し、出力される数値が正解に近づいていくように辺の重みづけを変更します。これを繰り返すことで、最終的に正解に近い結果を出力するようなネットワークが構成されることになります。以上のような学習方法のことを**誤差逆伝搬法 (バックプロパゲーション)** と呼びます。

(図 II.5〜II.8 の引用元:[7])

誤差逆伝搬法

「出力される数値を正解に近づける」というのを言い換えると、「出力と正解との誤差を小さくする」ことが目的である、と言えます。これで、解くべき問題をより明確な問題に置き換えることができました。ニューラルネットワークにおけるこの問題について考える前に、基本的な最小化問題に対するアプローチを説明します。

[10] そのカテゴリに属している可能性、度合いを表す数値

最急降下法

次のような簡単な式を最小化することを考えます。

$$f(x) = x^2 - 4x + 5$$

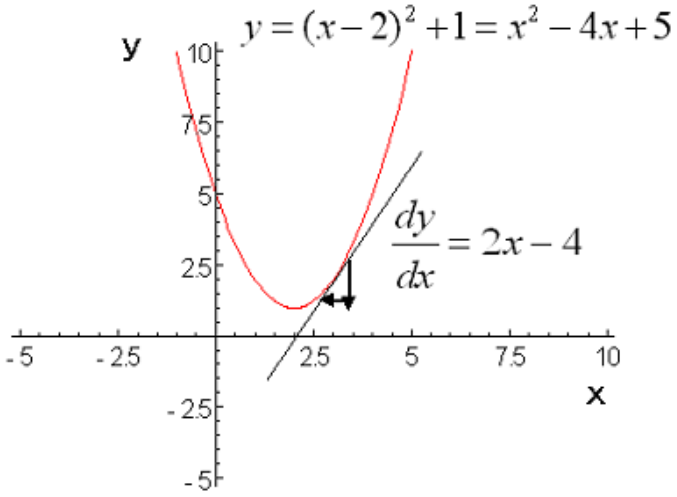

図 II.9　図の引用元:[9]

今、x の値を適当に決め、その点での接線の傾きが求まっているとします。そして、ここから x の値を修正することで、$f(x)$ の値を小さくしていくことを考えます。すると、接線の傾きが正なら x を少し小さく、負なら少し大きくすることで、$f(x)$ の値が小さくなっていきそうだということが分かります。そして、最小値付近では、接線の傾きは 0 に近づいていくことも分かります。

以上の考えをもとに、学習率 η (ある程度小さい数) というものを導入して、

$$x \leftarrow x - \eta f'(x)$$

という式によって x を更新していきます。$f'(x)$ が大きい時には x は大幅に変化します。逆に $f(x)$ がほぼ最小化されている場合には x はほぼ変化しないため、ほぼ一定に保たれます。

この方法によって $f(x)$ は次第に小さくなっていきますが、必ずしも最小化されるわけではなく、実際には極小値にしかなりません。しかし、今は値の最小化ということよりも値を小さくしていくことに関心があるため、このことはあまり問題にはなりません。ただ、機械学習において、初期値によっては悪い極小値に陥り、どれだけ訓練データを増やして

も良い結果が出ないということもあり得る、ということは頭に留めておく必要があります。

同様にして、n 変数の関数 $f(x_1, x_2, ..., x_n)$ についても、そのそれぞれの変数による偏微分[11]を考え、

$$x_1 \leftarrow x_1 - \eta \frac{\partial f(x_1, x_2, ..., x_n)}{\partial x_1}$$

$$x_2 \leftarrow x_2 - \eta \frac{\partial f(x_1, x_2, ..., x_n)}{\partial x_2}$$

$$...$$

という更新を繰り返すことで、f を最小値に近づけることができます。このように目的関数 f の値を小さくしていく方法を**最急降下法**と呼びます。

卑近な例で言うと、これは山の中で遭難してしまった際に、下の方に降りていけばいつか町に出るだろう、という考え方と同じです。先ほどの注意と同様、この方法は盆地が存在すると詰みます。

誤差関数の最小化

話を元に戻しましょう。これから話を進める上で、いくつかの記号を導入します。ここで関数 $f(x)$ は式 II.1 で定義したシグモイド関数です。

- 入力層、隠れ層、出力層の頂点数をそれぞれ N_{in}, N_{hid}, N_{out} とする。
- 入力 $(I_1, I_2, ..., I_{N_{in}})$ に対する隠れ頂点の出力を $(H_1, H_2, ..., H_{N_{hid}})$、出力頂点の出力を $(O_1, O_2, ..., O_{N_{out}})$ とする。
- 入力頂点 i から隠れ頂点 j への辺の重みを W_{ij} とし、隠れ頂点 j から出力頂点 k への辺の重みを W'_{jk} とする。
- $S_j = \sum_{i=1}^{N_{in}} I_i W_{ij}$ とすると、$H_j = f(S_j)$ である。
- $T_k = \sum_{i=1}^{N_{hid}} H_i W'_{jk}$ とすると、$O_k = f(T_k)$ である。
- 入力に対する正解を $(A_1, A_2, ..., A_{N_{out}})$ とする。これはこのインスタンスが属するカテゴリを cat としたとき $A_{cat} = 1$ であり、それ以外について $A_k = 0$ となる。

色々書いていますが、今までに述べたことを記号を導入して整理しただけです。

[11] 偏微分とは、関数をその変数だけに依存する関数だと考え、その他の変数を定数と考えて微分することです。これも、1 変数の微分と同様に他の変数が固定された状態での $f(x_i)$ という関数の接線の傾きと捉えることができます。

さて、今最小化[12]したいのは誤差関数です。これを E とおき、

$$E = \frac{1}{2}\sum_{k=1}^{N_{out}}(A_k - O_k)^2 \tag{II.2}$$

とします。$\frac{1}{2}$ は微分した際の係数を消すため、2 乗しているのは正負をまとめて扱うためです。このような誤差の定義の仕方は機械学習においてしばしば用いられます。

先ほど説明した最急降下法を用いることを考えましょう。E を最小化したいので、E を変化させたい変数で偏微分すれば、その変数の変化させるべき値が求まります。ここで A_k は定数なので、これは $O_1, O_2, ..., O_{N_{out}}$ の関数になっています。ところが、O_k は直接変化させることはできず、変化させるのは W_{ij}、W'_{jk} なので、E をこれらで微分した結果を求める必要があります。

まずは W'_{jk} の修正すべき差分 $\Delta W'_{jk}$ を求めましょう。微分の連鎖律[13]を用いると、

$$\Delta W'_{jk} = -\eta\frac{\partial E}{\partial W'_{jk}} = -\eta\frac{\partial E}{\partial O_k}\frac{\partial O_k}{\partial T_k}\frac{\partial T_k}{\partial W'_{jk}} \tag{II.3}$$

となります。それぞれの値を計算していきます。まず式 II.2 より

$$\frac{\partial E}{\partial O_k} = -(A_k - O_k)$$

です。O_k 以外の変数がすべて消えるのは、偏微分なので O_k 以外の変数を全て定数とみなすためです。

次に、簡単な計算により、式 II.1 のシグモイド関数について

$$f'(x) = f(x)(1 - f(x))$$

が成り立つことが分かります。よって、

$$\frac{\partial O_k}{\partial T_k} = \frac{\partial f(T_k)}{\partial T_k} = f(T_k)(1 - f(T_k)) = O_k(1 - O_k)$$

となります。頂点の内部関数にシグモイド関数を用いたのは、このように微分して元の関数の式で表せるという性質を用いると計算が簡単になるからです。

最後に、

$$\frac{\partial T_k}{\partial W'_{jk}} = \frac{\partial(\sum_{j=1}^{N_{hid}} H_j W'_{jk})}{\partial W'_{jk}} = H_j$$

[12] ここでの「最小化」は厳密な意味での最小化というより、目的関数を小さくしていくという意味合いで使っています。

[13] $z = f(y), y = g(x)$ と表されている時に、$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$ が成り立つという法則

以上から、式 (3) より

$$\Delta W'_{jk} = -\eta(-(A_k - O_k))O_k(1-O_k)H_j = \eta(A_k - O_k)O_k(1-O_k)H_j = \eta H_j \delta_k$$

$(\delta_k = O_k(1-O_k)(A_k - O_k)$ とおいた)

と表せ、きれいな式になりました。

次は ΔW_{ij} を求めます。先ほどは W'_{jk} が出力 O_k にしか関係していなかったため、$\frac{\partial E}{\partial O_k}$ のみを考えればよかったのですが、今度はそうはいきません。そこで、合成関数に関する次の公式を用いることになります。

関数 $f(y_1, y_2, ..., y_n)$ について、$y_j = g_j(x_1, x_2, ..., x_m)$ と表されているとき、

$$\frac{\partial f}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial y_j}\frac{\partial y_j}{\partial x_i}$$

が成り立つ。

証明は省略して認めることとします。E が $O_1, O_2, ..., O_{N_{out}}$ の関数であり、$O_k = f(T_k)$ と表されているので E は $T_1, T_2, ..., T_{N_{out}}$ の関数です。これと $T_k = \sum_{j=1}^{N_{hid}} H_j W'_{jk}$ より T_k が $H_1, H_2, ..., H_{N_{out}}$ の関数として表されていることに注意してこれを用いると

$$\begin{aligned}
\Delta W_{ij} = -\eta\frac{\partial E}{\partial W_{ij}} &= -\eta\frac{\partial E}{\partial H_j}\frac{\partial H_j}{\partial W_i j} \\
&= -\eta(\sum_{k=1}^{N_{out}} \frac{\partial E}{\partial T_k}\frac{\partial T_k}{\partial H_j})\frac{\partial H_j}{\partial S_j}\frac{\partial S_j}{\partial W_{ij}} \\
&= -\eta(\sum_{k=1}^{N_{out}} \frac{\partial E}{\partial O_k}\frac{\partial O_k}{\partial T_k}\frac{\partial T_k}{\partial H_j})\frac{\partial H_j}{\partial S_j}\frac{\partial S_j}{\partial W_{ij}}
\end{aligned}$$

となります。前二つの式はすでに計算済みで、その積は $-\delta_k$ であり、

$$\frac{\partial T_k}{\partial H_j} = \frac{\partial \sum_{j=1}^{N_{hid}} H_j W'_{jk}}{\partial H_j} = W'_{jk}$$

$$\frac{\partial H_j}{\partial S_j} = \frac{\partial f(S_j)}{\partial S_j} = H_j(1-H_j)$$

$$\frac{\partial S_j}{\partial W_{ij}} = \frac{\partial \sum_{i=1}^{N_{in}} I_i W_{ij}}{\partial W_{ij}} = I_i$$

です。以上から、

$$\Delta W_{ij} = \eta(\sum_{k=1}^{N_{out}} \delta_k W'_{jk}) H_j(1-H_j) I_i = \eta I_i H_j (1-H_j) \sum_{k=1}^{N_{out}} \delta_k W'_{jk}$$

これでようやく更新式が求められました。隠れ層が 2 つ以上になった際にも、2 つ目の更新式と同じ形の式で更新することができます。

5a　ニューラルネットワークまとめ

上で求めた更新式をもう一度書きます。

$$\delta_k = O_k(1-O_k)(A_k - O_k)$$

$$W'_{jk} \leftarrow W'_{jk} + \eta H_j \delta_k$$

$$W_{ij} \leftarrow W_{ij} + \eta I_i H_j (1-H_j) \sum_{k=1}^{N_{out}} W'_{jk} \delta_k$$

δ_k には、各カテゴリでのスコアの誤差と、シグモイド関数を微分した形が含まれています。シグモイド関数は入力の値が極端になるほど傾きの小さくなる関数です。スコアが極端 (ほとんど確定している) と修正分が小さくなり、そうでないとき (未確定) は修正分が大きくなる。そしてそれに誤差分がかかると考えれば、なんとなくこの式の直感的な理解もできるのではないでしょうか。

また、誤差を最小化したいのなら更新を止まるまで実行していかなくていいのか、と思う人がいるかもしれません。これは一見正しいようですが、このように一つのデータセットに対して過度に誤差を少なくしようとすることは、他のデータセットに対しての誤差を大きくすることにつながりかねません。一般に特定のデータに対してのみ良い精度の分類を行う分類器より、全てのデータに対しそこそこの結果を出す分類器の方が優れていると言えます。ニューラルネットワークの場合、全てのデータセットに対してそこそこの結果を出すためには各データセットに対し一回更新を実行するだけで十分です。このようなまんべんなくまともな結果を出すような分類器の性能のことを**汎化性能**と呼び、一つのデータセットに対して過度に学習して汎化性能が下がってしまうことを**過学習**と呼びます。

それでは、最後にニューラルネットワークを用いて実際に学習をさせてみます。下のソースコードにおいて、$feed\_forward$ という関数が入力から各頂点の出力を計算する関数、$back\_propagate$ という関数が誤差から辺の重みづけを変更する関数です。実際の訓練・評価の部分は K 近傍法のソースコードとほぼ同じなので省略します[14]。

[14] 訓練には $back\_propagate$ を、評価には $feed\_forward$ を用います。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from math import exp
from random import *

#シグモイド関数
def sigmoid(x):
    return 1 / (1 + exp(-x))

#シグモイド関数を微分した関数
def dif_sigmoid(y):
    return y * (1 - y)

def rand():
    # [-0.1,0.1) のランダムな数を返す
    return random() * 0.2 - 0.1

def randtable(n, m):
    #rand 関数を用いてランダムな (n,m) テーブルを作成

class Neuralnet:
    #初期化
    def __init__(self, n_input, n_hidden, n_output, rate):
        #入力層→隠れ層への辺の初期化
        self.in_edge = randtable(n_input, n_hidden)
        #隠れ層→出力層への辺の初期化
        self.out_edge = randtable(n_hidden, n_output)
        self.in_data = [0] * n_input # I_i
        self.hid_data = [0] * n_hidden # H_j
        self.out_data = [0] * n_output # O_k
        self.rate = rate #学習率 η
        self.n_in = n_input #入力層の頂点数
        self.n_hid = n_hidden #隠れ層の頂点数
        self.n_out = n_output #出力層の頂点数

    #前方向計算
```

```
    def feed_forward(self, data):
        #I_i への入力
        for i in range(self.n_in):
            self.in_data[i] = data[i]
        #H_j の計算
        for j in range(self.n_hid):
            s = 0
            for i in range(self.n_in):
                s += self.in_data[i]
                        * self.in_edge[i][j]
            self.hid_data[j] = sigmoid(s)
        #O_k の計算
        for k in range(self.n_out):
            s = 0
            for j in range(self.n_hid):
                s += self.hid_data[j]
                        * self.out_edge[j][k]
            self.out_data[k] = sigmoid(s)
        return self.out_data

    #逆方向誤差伝搬
    def back_propagate(self, data, answer):
        #各頂点の出力計算
        ret = self.feed_forward(data)
        #data : I_i, answer : A_k, ret : O_k
        out_dif = [] #δ_k
        for k in range(self.n_out):
            out_dif.append((answer[k] - ret[k])
                           * dif_sigmoid(ret[k]))
        hid_dif = [] #Δ W_ij の H_j * (1 - H_j) * Σ の部分
        for j in range(self.n_hid):
            s = 0
            for k in range(self.n_out):
                s += self.out_edge[j][k]
                            * out_dif[k]
            hid_dif.append(dif_sigmoid(self.hid_data[j]) * s)
        #各辺の重さを修正
```

```
        for i in range(self.n_in):
            for j in range(self.n_hid):
                self.in_edge[i][j]
                    += self.rate * hid_dif[j] * data[i]
        for j in range(self.n_hid):
            for k in range(self.n_out):
                self.out_edge[j][k]
                    += self.rate * out_dif[k] * self.hid_data[j]
```

学習の結果は以下のようになりました。

データセット	正答サンプル数	全サンプル数	正答率
ひらがな (ETL4)	220.0	305	72.1%
カタカナ (ETL5)	508.3	530	96.0%

カタカナのデータセットでは高い正答率を実現していますが、ひらがなのデータセットの結果はいまいちです。これはデータセットの数が少ないことが関係していると思われます。一般にニューラルネットワークは大体の問題に応用できる代わりに、比較的収束が遅い (良い結果を得るために必要なデータセットの量が多い) ことが知られています。

第 III 章

今後の展望

第 II 章では、画像から加重方向指数ヒストグラムという方式を用いて特徴を抽出し、その情報と K 近傍法・ニューラルネットワークを用いて文字認識を行いました。では、特徴を抽出せず、画像データをそのまま[1]ニューラルネットワークに渡すとどうなるのでしょうか。画像を 10*10 ピクセルに圧縮し、200 個の隠れ頂点を持つニューラルネットワークに渡してみます。(カタカナデータセットのみ)

データセット	正答サンプル数	全サンプル数	正答率
カタカナ (ETL5)	384.3	530	72.5%

高い精度とは言えませんが、特徴抽出の手間なしでこの精度が実現できる、というのは注目に値します。実は、今回紹介したニューラルネットワークの階層を増やし、さらに改良を重ねることで、元のデータから自動で特徴抽出のようなことが行われ、かなり高い精度での識別が実現される、ということが分かってきています。

以前は今回紹介した通り、元のデータからある意味手動で特徴を抽出し、それを学習モデルに渡すことで機械学習を行ってきました。しかし、この方法では、特徴抽出の技術によって結果が大きく左右され、しかも今回見たように特徴抽出には分野ごとに異なる洗練されたアプローチが必要とされるため、機械学習には職人技とも呼ばれる特徴抽出技術が必要でした。

そのため、その特徴の抽出すらも機械に丸投げすることができる、というのは非常に画期的であり、Google や Facebook などの大企業をはじめ、世界中の企業・研究者がこの方法に注目しています。この深い階層のニューラルネットワークによる**ディープラーニング (Deep Learning)** という分野は、今後大きく発展していくことが期待されています。今回はこのような軽い紹介に留まってしまいましたが、非常に魅力的な分野であることは間違いないので、興味のある方は是非勉強してみることをお勧めします。

[1] 実際は各ピクセルで黒い画素を 1、白い画素を 0 とする特徴を入力として与えています

第 IV 章

おわりに

お疲れ様でした。今回は機械学習の大まかな流れ、K 近傍法、ニューラルネットワークなどについて紹介しましたが、クラスタリングなどの教師なし学習、自然言語処理、分類器であるベイズ分類器・決定木・SVM(サポートベクトルマシン)、そして第 Ⅲ 章で紹介した Deep Learning など、今回扱えなかった話題はたくさんあるので、より詳しく知りたい方は参考文献を参照してみてください。Deep Learning に関しては中々まだ勉強しやすい環境は整っていないようですが、多くの研究グループが Deep Learning を行うためのソフトウェアをオープンソースにしているそう [10] なので、そちらを試してみるのも良いと思います。

また、この記事についてのご質問、ご指摘などありましたら、以下の連絡先までご連絡頂けると幸いです。私自身まだ機械学習の勉強を始めて間もないので、どんな内容でも参考にさせて頂きます。

最後になりましたが、私の稚拙な文章をここまで読んで下さり、ありがとうございました。もしこの記事によって機械学習に興味を持って頂け、この記事が皆さんの理解の一助となれたのならば、これ以上嬉しいことはありません。

連絡先
Gmail: ryunosuke.iwai@gmail.com
Twitter: @reyk429

参考文献

[1] Toby Segaran. (2008).『集合知プログラミング』(當山 仁健・鴨澤 眞夫 訳) オライリー・ジャパン

[2] 高村 大也. (2010).『言語処理のための機械学習入門』(奥村 学 監修) コロナ社

[3] Bishop, C.M. (2012).『パターン認識と機械学習』(元田 浩・栗田 多喜夫・樋口 知之・松本 裕治・村田 昇 監訳) 丸善出版

[4] WordPress. (2014).『ETL 文字データベース』 http://etlcdb.db.aist.go.jp/?lang=ja

[5] Media Drive Corporation. http://mediadrive.jp/technology/techocr10.html

[6] 木村 拓也. (2011).『複数人の類似筆記者の文字特徴を利用したひらがな文字認識』

[7] 村上, 泉田研究室.『ニューラルネットワーク』 http://ipr20.cs.ehime-u.ac.jp/column/neural/index.html

[8] Akira Iwata, Toshiyuki Matubara. (1996).『ニューラルネットワーク入門』http://www-ailab.elcom.nitech.ac.jp/lecture/neuro/menu.html

[9] 金久保　正明. http://www.sist.ac.jp/∼kanakubo/index.html

[10] Hatena blog『Deep Learning でラブライブ！キャラを識別する』http://christina.hatenablog.com/entry/2015/01/23/212541

[11] Wikipedia『機械学習』 http://ja.wikipedia.org/wiki/機械学習