ニューラルネットワークと誤差逆伝搬法の概要

Pythonによる手書き文字判定モデルの作成

研究者：下吉 賢信　　指導教員：濱川 恭央

あらまし　このレポートは本科５年にて実施される卒業研究への先駆けとなるレポートである。ニューラルネットワークへの理解を深めることを目的とし、理論などを学ぶ。理論を学んだ後、MNISTの手書き文字データを判定するモデルを作成し、実験を行った。学習回数を変動させることによる、正解率への影響を調べた。

キーワード：　ニューラルネットワーク，誤差逆伝搬法，Python2.7，MNIST,手書き文字認識

1. 目的

　ニューラルネットワークに対する理解を深め、誤差逆伝搬法での学習モデルの作成を目的とする。

1. 理論
   1. 単層パーセプトロン
      1. パーセプトロンとは

　　パーセプトロンとは、複数の信号を入力として受け取り、一つの信号を出力するものである。その信号は１か０の二値である。図１の、２つの信号を入力として受け取るパーセプトロンを示す。x1,x2は入力信号yは出力信号、w1,w2は重みを表す。図１の○は「ニューロン」や「ノード」などと呼ばれる。図１に表されるパーセプトロンは特に単層パーセプトロンと呼ばれる。

　入力信号は、ニューロンに送られる際、x1,x2にそれぞれの固有の重みw1,w2が乗算され、ニューロンで和が求められる。その総和がある限界値を超えた場合に１を出力する。これを「ニューロンが発火する」と表現することもある。その限界値を閾値と呼び、θで表す。

数式で表すと、式⑴のようになる。

|  |  |  |
| --- | --- | --- |
|  |  |  |

* + 1. ANDゲートの実装

　パーセプトロンを用いてANDゲートについて考える。(w1,w2,θ)の選び方は多様にあり、(0.5,0.5,0.8)や(0.9,0.8,1.0)でも書く入力に対し、同じ出力を確認できる。これらのパラメータを設定すればx1,x2が両方1のときだけ、重み付き信号の総和が閾値を上回る。

* + 1. 重みとバイアス

式⑴のθを-bとして、パーセプトロの動作を式⑵に示す。

|  |  |  |
| --- | --- | --- |
|  |  |  |

　ここでbをバイアスと呼ぶ。式⑵で表されるように、パーセプトロンでは、入力信号に重みが乗算された値とバイアスの和が計算され、その値が０を上回ると１を出力し、そうでなければ０を出力する。

　bが-0.1であれば、入力信号の重み付き和が0.1を上回るだけでニューロンが発火する。しかし、もしbが-20であれば、入力信号の重み付き和が20を上回らなければ発火しない。このことから、バイアスbはニューロンの発火のしやすさの度合いであると言える。

* + 1. XORゲート

　XORゲートは、x1,x2のどちらかが1だったときのみ1を出力する論理回路である。この論理回路は、単層パーセプトロンでは実装できない。それを以下に示す。

　まずORゲートの挙動を視覚的に考える。たとえば重みパラメータ(b,w1,w2) = (-0.5,1.0,1.0)のとき、パーセプトロンは式⑶で表される。

|  |  |  |
| --- | --- | --- |
|  |  |  |

　式⑶で表されるパーセプトロンは、-0.5 + x1 + x2 = 0の直線で分断された二つの領域を作る。直線で分けられた片方の領域は１を出力し、もう片方は０を出力する。これを図２に示す。図では、0を○、1を△で表している。ORゲートを作るには図２の○と△を二つの領域に分ける必要がある。実際に、式⑶の直線は4つの点を正しく分断できている。

|  |
| --- |
| 画像/新規ドキュメント%202017-01-31_2.jpg |

図 2 ORゲートの出力

それでは、XORゲートはどうだろう。XORゲートの出力を図３に示す。

|  |
| --- |
| 画像/新規ドキュメント%202017-01-31_3.jpg |

図 3 XORゲートの出力

図３の○と△を直線によって分断することは不可能である。よって、単層パーセプトロンでは実装できないことがわかる。

* + 1. 線形と非線形

　図３の○と△は直線で分断することができない。だが、直線という制限を無くせば、分断できる線はいくらでもあるだろう。

　単層パーセプトロンの限界は、一本の直線で分けた領域しか表現できない点にある。領域を分断する曲線を単層パーセプトロンでは表現できない。ちなみに、図２のような直線による領域を線形な領域といい、曲線による領域を非線形な領域という。この非線形な領域を表現するのが、多層パーセプトロンである。

* 1. 多層パーセプトロン

パーセプトロンでは実装できなかったXORゲートを実現するのが多層パーセプトロンである。多層パーセプトロンとは層を重ねることで非線形な領域を表現しようとしたものだ。

* + 1. XORの実装

　それでは、多層パーセプトロンを用いてXORを実装する。XORゲートは、AND,NAND,ORゲートの組み合わせによって実装することができる。論理ゲートの組み合わせによって実装されたXORを図４に示す。

|  |
| --- |
| ../../Desktop/新規ドキュメント%202017-02-15_1.jpg |

図 4 XORの論理ゲート

　ANDゲートのみをパーセプトロンで表現したがORもNANDも実装可能である。図４の各論理ゲートをパーセプトロンに置き換えた図を図５に示す。図４と図５のx,s,yはそれぞれ対応している。

|  |
| --- |
| ../../Desktop/新規ドキュメント%202017-02-15_2.jpg |

図 5 多層パーセプトロンでのXOR

単層パーセプトロンでは表現できなかったことが、層を一つ増やすことによって表現できるようになった。つまり、パーセプトロンは層を増やすことで、より柔軟な表現が可能となる。

* 1. ニューラルネットワーク

　ニューラルネットワークは、前節で説明したパーセプトロンと共通する点が多くある。前節のパーセプトロンと異なる点を中心に、ニューラルネットワークの仕組みを解説する。

* + 1. ニューラルネットワークの例

　ニューラルネットワークを図で表すと図６のようになる。

|  |
| --- |
| ../../Desktop/新規ドキュメント%202017-02-15_3.jpg |

図 6 ニューラルネットワークの例

ここで、一番左の層を入力層、一番右の列を出力層、中間の列を中間層または隠れ層と呼ぶ。図を見る限り、ニューロンのつながり方に関しては、パーセプトロンとの違いはない。パーセプトロンは出力が二値であったのに対し、実数となっている。

* + 1. 活性化関数とは

　活性化関数は入力信号の総和がどのように発火するかというこsとを決定する役割がある。活性化関数をで表すとすると、式⑵は以下の式に書き換えられる。

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

　これを明示的に示すとすれば、図７のようになる。

|  |
| --- |
| ../../Desktop/新規ドキュメント%202017-02-15_4.jpg |

図 7 活性化関数

図に示される通りこれまでのニューロンの○の中に、活性化関数によるプロセスがある。つまり、重み付き信号の和の結果がというノードになり、活性化関数によってというノードに変換されることが示されている。

* 1. 活性化関数

　活性化関数の例をいくつか示す。

* + 1. ステップ関数

　式⑹に表される活性化関数は、閾値を境にして出力が切り替わる関数であり「ステップ関数」や「階段関数」と呼ばれる。

* + 1. シグモイド関数

　ニューラルネットワークでよく用いられる活性化関数のひとつは、式⑺で表されるシグモイド関数である。

|  |  |  |
| --- | --- | --- |
|  |  |  |

　はを意味する。グラフは以下のようになる。

|  |
| --- |
| *../../Desktop/figure_1.png* |

図８　シグモイド関数

* + 1. ソフトサイン

下記関数をソフトサインと呼ぶ。

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |
| --- |
| ../../Desktop/figure_1.png |

図 9 ソフトサイン

* + 1. ソフトプラス

　下記関数をソフトプラスと呼ぶ。

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |
| --- |
| ../../Desktop/figure_1.png |

図 10 ソフトプラス

* 1. 訓練とは

　機械学習の問題では、訓練データとテストデータの２つに分けて、学習や実験などを行うのが一般的である。その場合、まず訓練データのみを用いて学習を行い、最適なパラメータを探索する。そして、テストデータを使って、その訓練したモデルに実力を評価する。訓練データとテストデータを分けるのにはある理由がある。それは、我々が求めているものは、モデルの汎用的な能力であるからである。この汎化能力を正しく評価したいがために、訓練データとテストデータを分離する必要があるのである。なお、訓練データを教師データと呼ぶ。

* 1. 損失関数

　損失関数はニューラルネットワークの性能の悪さを示す指標である。現在のニューラルネットワークが教師データに対してどれだけ適合していないか、教師データに対してどれだけ一致していないかということを表す。損失関数にマイナスをかけた値は、どれだけ性能が良いかという指標として解釈できるため、どちらを指標としたとしても行うことは本質的に同じである。損失関数として用いられる関数をいくつか以下に示す。

* + 1. ２乗和誤差

　次の数式で表される。ここで、はニューラルネットワークの出力、は教師データを表し、はデータの次元数を表す。

|  |  |  |
| --- | --- | --- |
|  |  |  |

* + 1. 交差エントロピー誤差

　次の数式で表される。

|  |  |  |
| --- | --- | --- |
|  |  |  |

* 1. 勾配法

　機械学習の問題の多くは、学習の際に最適なパラメータを探索する。ニューラルネットワークも同様に最適なパラメータを学習時に見つけなければならない。最適なパラメータというのは、損失関数が最小値をとるときのパラメータの値である。しかし、一般的に損失関数は複雑で、パラメータ空間は広大であるため、どこに最小値を取るのか見当がつかない。そこで、勾配をうまく利用して関数の最小値を探すことを目指したものが勾配法である。

勾配法では、現在の場所から勾配方向に一定の距離だけ進む。そして、移動した先でも同様に勾配を求め、また、その勾配方向へ進むというように、繰り返し勾配方向へ移動する。このように勾配方向へ進むことを繰り返すことにより、関数の値を徐々に減らすのが勾配法である。ニューラルネットワークの学習では勾配法がよく用いられる。例えば、という式についての勾配法を数式で表すと次の式のように書くことができる。

|  |  |  |
| --- | --- | --- |
|  |  |  |

　式(12)のηは更新の量を表す。これは、ニューラルネットワークの学習においては、学習率と呼ばれる。一回の学習でどれだけパラメータを更新するかを決めるのが学習率である。

　式（12）は一回の更新式を示しており、このステップを繰り返し行う。つまり、ステップごとに、変数の値を更新していき、そのステップを何度か繰り返すことによって徐々に関数の値を減らしていく。ここでは、変数が２つの場合を示しているが、変数の数が増えても、同じような式によって更新される。

勾配法は最小値を探す場合を「勾配降下法」、最大値を探す場合を「勾配上昇法」という。

* 1. 誤差逆伝播法

　ニューラルネットワークの重みパラメータの勾配は、数値微分によって求められる。誤差逆伝播法とは、重みパラメータの勾配の計算を効率良く行う手法である。以下に誤差逆伝播の手順を示す。

* + 1. ネットワーク

　まず、想定するニューラルネットワークと、記号を対応付けする。ここでは、隣接する３つの層を考え、入力に近い順からi、j、k、で表す。３層のニューラルネットワークであれば、iが入力層、jが隠れ層、kが出力層となるような構成である。また、iからjへの結合荷重の重みを、jからkへの結合の重みをとして表すこととする。このとき、p番目のパターンを入力とした時、ユニットiからの出力を、ユニットjへの入力をとすると、

|  |  |  |
| --- | --- | --- |
|  |  |  |

さらに、ユニットｊからの出力を非線形関数を用いて、

|  |  |  |
| --- | --- | --- |
|  |  |  |

と表すことにする。

　活性化関数はシグモイド関数、誤差関数は２乗和誤差を用いる。式(10)をこのニューラルネットワークに置き換えると、

|  |  |  |
| --- | --- | --- |
|  |  |  |

となる。ここで、は教師データとなる正解値は出力値、は出力層のユニットを表すインデックスである。

* + 1. 勾配降下法での重みの計算

　勾配降下法を用いた重みの更新は

|  |  |  |
| --- | --- | --- |
|  |  |  |

と求められる。また、勾配は連鎖率を用いて次のように計算できる。

|  |  |  |
| --- | --- | --- |
|  |  |  |

右辺の第一項をで表すことにすると、

|  |  |  |
| --- | --- | --- |
|  |  |  |

第二項は式(13)を代入し、さらに偏微分で残るのはの時だけなので、

|  |  |  |
| --- | --- | --- |
|  |  |  |

として求められる。これを式(16)に代入すると、

|  |  |  |
| --- | --- | --- |
|  |  |  |

を得ることができる。

* + 1. 個々のユニットに対するの計算

　同じように連鎖率を使って計算を行っていくことが可能である。

|  |  |  |
| --- | --- | --- |
|  |  |  |

第一項の計算だが、ユニットｊが出力層かもしくは隠れ層かによって異なる。出力層にある場合は、

|  |  |  |
| --- | --- | --- |
|  |  |  |

一方中間層にある場合は、

|  |  |  |
| --- | --- | --- |
|  |  |  |

さらに、第一項と第二項は、

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

つまり、中間層における計算式は以下のように求められる。

|  |  |  |
| --- | --- | --- |
|  |  |  |

* + 1. シグモイド関数の微分式

　シグモイド関数の微分式は以下のように表される。

|  |  |  |
| --- | --- | --- |
|  |  |  |

よって、はシグモイド関数であるため、

|  |  |  |
| --- | --- | --- |
|  |  |  |

* + 1. 重みの更新式

　勾配降下法による重みの更新は、jが出力層の時は、

|  |  |  |
| --- | --- | --- |
|  |  |  |

Jが隠れ層の時は、

|  |  |  |
| --- | --- | --- |
|  |  |  |

となる。

1. 実験
   1. 概要

　M NISTの手書き文字データを使用し、手書き文字を判別するニューラルネットワークを作成する。MNISTの手書き文字データは学習データ60000個、テストデータ10000個で構成される。各画像は28ピクセル×28ピクセルであり784列のベクトルである。つまり、学習データセットは[60000, 784]の行列となる。

　作成するニューラルネットワークは、入力層784個、中間層100ユニット、出力層10ユニットである。Pythonでの実装を行う。

　全学習データセット６００００個の学習を１セットとする。学習を１セットから20セットまで行い、それぞれのテストデータの正答率を算出し、同一データにおいて学習回数は正答率への影響があるのかを計測する。

* 1. 実行環境

OS X El Capitan 10.11.6

プロセッサ　1.6GHz Intel Core i5

メモリ 8GB

Intel HD Graphics 6000

python 2.7.10

* 1. 実装

　作成したプログラム図12,図13,図14を示す。

　今回はメインのプログラムを簡素なものにするため、メインプログラム、学習データを取り込むプログラム、三層のニューラルネットワークを作成するプログラムの３つに分けて作成した。

* 1. 結果

　実験結果を図１１に示す。

1. 考察

学習回数540000回を超えたあたりから正解率が横ばいになり、ニューラルネットワークの改善が見られなくなったが95%の精度はでていたので、ネットワーク自体は完成していると思われる。学習時間は学習回数に比例して単純で線形的な増加傾向にあるので、どの程度まで精度を出すのかしっかりと見極めなければ時間の無駄になってしまう。

　また、任意の人の書いた任意の文字を判別するので、どの手書き文字でも正確な出力を出せるとは限らない。手書き文字判別のニューラルネットワークに関して言えば、完全な正解はないのだと感じられる。もし、入力者の意思と異なる出力を出した場合、何かしらのUIで取り消させるとするならば95%程度の正答率でも実用できる範囲であると考えられる。

1. あとがき

　今回は学習回数を変数に実験を行った。次の機会には中間層のユニット数や層の数を変更できるようなプログラムを作成し出力にどのような影響がでるのか、実験を行いたい。

1. 参考文献
2. 斎藤　康毅, [初版] ゼロから作るDeep Learning –Pythonで学ぶディープラーニングの理論と実装, 2016
3. Pythonで多層パーセプトロンの実装例,

<http://stmind.hatenablog.com/entry/2014/06/16/145524>, 2014

1. 数学的基礎から学ぶ

Deep Learningその２（受講中）, http://qiita.com/watarukato/items/98584e3bcb8e091be96b,2016

1. 付録

|  |  |  |
| --- | --- | --- |
| 学習回数（回） | 学習時間（秒） | 正答率（％） |
| 60000 | 16.794 | 89.61 |
| 120000 | 32.621 | 92.20 |
| 180000 | 49.968 | 93.20 |
| 240000 | 61.740 | 93.86 |
| 300000 | 85.238 | 94.18 |
| 360000 | 98.536 | 94.35 |
| 420000 | 111.222 | 94.79 |
| 480000 | 123.669 | 94.86 |
| 540000 | 141.635 | 95.18 |
| 600000 | 157.430 | 95.29 |
| 660000 | 176.411 | 95.23 |
| 720000 | 194.846 | 95.48 |
| 780000 | 209.719 | 95.40 |
| 840000 | 227.684 | 95.51 |
| 900000 | 236.858 | 95.66 |
| 960000 | 254.260 | 95.44 |
| 1020000 | 271.936 | 95.61 |
| 1080000 | 289.302 | 95.65 |
| 1140000 | 297.469 | 96.10 |
| 1200000 | 314.783 | 96.15 |

図 11 実験結果

|  |
| --- |
| # coding: utf-8  #WMNIST.py  import os  import struct  from array import array  class MNIST(object):  def \_\_init\_\_(self):  self.train\_img\_fname = 'train-images-idx3-ubyte'  self.train\_lbl\_fname = 'train-labels-idx1-ubyte'  self.train\_images = []  self.train\_labels = []  self.test\_images = []  self.test\_labels = []  def load\_training(self):  ims, labels = self.load(('./mnist/train-images-idx3-ubyte'),  ('./mnist/train-labels-idx1-ubyte'))  self.train\_images = ims  self.train\_labels = labels  return ims, labels  def load\_test(self):  tims, tlabels = self.load(('./mnist/t10k-images-idx3-ubyte'),('./mnist/t10k-labels-idx1-ubyte'))  self.test\_images = tims  self.test\_labels = tlabels  return tims, tlabels  @classmethod  def load(cls, path\_img, path\_lbl):  with open(path\_lbl, 'rb') as file:  magic, size = struct.unpack(">II", file.read(8))  if magic != 2049:  raise ValueError('Magic number mismatch, expected 2049,'  'got {}'.format(magic))  labels = array("B", file.read())  with open(path\_img, 'rb') as file:  magic, size, rows, cols = struct.unpack(">IIII", file.read(16))  if magic != 2051:  raise ValueError('Magic number mismatch, expected 2051,'  'got {}'.format(magic))  image\_data = array("B", file.read())  images = []  for i in range(size):  images.append([0] \* rows \* cols)  for i in range(size):  images[i][:] = image\_data[i \* rows \* cols:(i + 1) \* rows \* cols]  return images, labels |
| 図 12 WMNIST.py |

|  |
| --- |
| # make\_3Layered\_perceptron.py  import numpy as np  class make\_perceptron(object):  """  3 Layered Perceptron  """  def \_\_init\_\_(self, n\_input\_units, n\_hidden\_units, n\_output\_units):  self.nin = n\_input\_units  self.nhid = n\_hidden\_units  self.nout = n\_output\_units  self.v = np.random.uniform(-1.0, 1.0, (self.nhid, self.nin+1))  self.w = np.random.uniform(-1.0, 1.0, (self.nout, self.nhid+1))  print ''  print '--- NN configuration ---'  print 'Num of input layer units: %d' % (self.nin)  print 'Num of hidden layer units: %d' % (self.nhid)  print 'Num of output layer units: %d' % (self.nout)  print 'Shape of first layer weight(v): (%d, %d)' % (self.nhid, self.nin+1)  print 'Shape of second layer weight(w): (%d, %d)' % (self.nout, self.nhid+1)  print ''  def \_\_add\_bias(self, x, axis=None):  return np.insert(x, 0, 1, axis=axis)  def \_\_sigmoid(self, u):  return (1.0 / (1.0 + np.exp(-u)))  def \_\_sigmoid\_deriv(self, u):  return (u \* (1 - u))  def fit(self, inputs, targets, learning\_rate=0.05, epochs=1000):  print 'number of learning : %d' %epochs  print 'learning rate : %f' %learning\_rate  inputs = self.\_\_add\_bias(inputs, axis=1)  targets = np.array(targets)  for loop\_cnt in xrange(epochs):  #randomise the order of the inputs  """  p = np.random.randint(inputs.shape[0])  xp = inputs[p]  bkp = targets[p]  """  for p in xrange(inputs.shape[0]):  xp = inputs[p]  bkp = targets[p]  #forward phase  gjp = self.\_\_sigmoid(np.dot(self.v, xp))  gjp = self.\_\_add\_bias(gjp)  gkp = self.\_\_sigmoid(np.dot(self.w, gjp))  #backward phase(back prop)  eps2 = self.\_\_sigmoid\_deriv(gkp) \* (gkp - bkp)  eps = self.\_\_sigmoid\_deriv(gjp) \* np.dot(self.w.T, eps2)  gjp = np.atleast\_2d(gjp)  eps2 = np.atleast\_2d(eps2)  self.w = self.w - learning\_rate \* np.dot(eps2.T, gjp)  xp = np.atleast\_2d(xp)  eps = np.atleast\_2d(eps)  self.v = self.v - learning\_rate \* np.dot(eps.T, xp)[1:, :]  def predict(self, inputs):  inputs.insert(0, 1)  u1 = np.dot(inputs, self.v.T)  z1 = self.\_\_sigmoid(u1)  z1 = np.insert(z1, 0, 1)  u2 = np.dot(z1, self.w.T)  y = self.\_\_sigmoid(u2)  return y |
| 図 13 make\_3Layered\_perceptron.py |

|  |
| --- |
| # -\*- coding: utf-8 -\*-  # mnist\_test.py  from WMNIST import MNIST  import make\_3Layered\_perceptron  import sys  import numpy as np  import time  mndata = MNIST()  train\_img, train\_label = mndata.load\_training()  test\_img, test\_label = mndata.load\_test()  train\_img = np.array(train\_img)  train\_img = train\_img / 255.0  test\_img = np.array(test\_img)  test\_img = test\_img / 255.0  trdata\_num = 60000  tedata\_num = 10000  mlp = make\_3Layered\_perceptron.make\_perceptron(n\_input\_units=784, n\_hidden\_units=50, n\_output\_units=10)  inputs = np.empty((0, 784), float)  targets = np.empty((0, 10), int)  start = time.time()  inputs = train\_img[0:trdata\_num]  targets = [np.insert(np.zeros(9), train\_label[i], 1) for i in xrange(trdata\_num)]  lrn\_num = input("Number of learning? -> ")  set\_time = time.time()  # training  print '---training---'  print 'Number of training data : %d' %trdata\_num  mlp.fit(inputs, targets, learning\_rate=0.05, epochs=lrn\_num)  learning\_time = time.time()  # predict  print '---predict---'  print ''  correct\_ary = [np.nanargmax(mlp.predict(list(test\_img[i]))) == test\_label[i] for i in xrange(tedata\_num)]  predict\_time = time.time()  print '---result---'  print 'correct : %d' %sum(correct\_ary)  print 'data set time(s) : %f' %(set\_time - start)  print 'learning time(s) : %f' %(learning\_time - set\_time)  print 'predict time(s) : %f' %(predict\_time - learning\_time)  print '\007' |
| 図 14 mnist\_test.py |