目的

このレポートでは、ニューラルネットワークの論理的なことは置いておき、NN を使うときどういうことが自明であるか?また pybrain を使ってどうすればうまく行くか?などについて調べた結果を述べる。主に調べる際に参考文献[1]を使用にしている。

ニューラルネットワークの構造とパラメータの与え方

誤差逆伝播法を用いるときには、ニューラルネットワークの構造(構造そのものや中間層は何層かなど)やパラメータ(学習係数・教師データの個数)の値を決める必要がある。この作業は、対象とする問題に応じて行う必要があり、設計理論が確立しているわけでない。以下の内容について経験的な指針について述べる。

(a)階層、各層のニューロン数
(b)繰り返し学習回数
(c)興奮性の信号値、抑制性の信号値
(d)学習係数
(e)その他

(a)階層、各層のニューロン数

誤差逆伝播法では、最急降下法を用いて誤差関数の大域的最小値(global minimum)の探索を試みている。本質的には、局所的最小値(local minimum)を見つける方法であり、大域的最小値を見つける保証はない。ニューラルネットワークの階層が多いほど、また各層のニューロン数が多いほど、局所的最小値が多くなり、誤差関数の大域的最小値探索の際に局所的最小値に陥る可能性が高まる。しかしながら、一般に、総数が多くニューロン数が多いほどパターン識別能力が高い。実際には試行錯誤で決められている。2層のニューラルネットワークは、単純パーセプトロンと同様に超平面で線形分離可能な問題しか識別できない。3層,4層と層数を増すにつれて識別能力が高まり、4層では任意の特徴ベクトル空間でパターン認識可能となる。例えば2層では識別できない排他的論理和の演算は3層で認識できることが知られている。

＜出力層＞

出力層のニューロン数は、分類すべきパターン数であり、問題に応じて決定する。また、排他的論理和のような出力が0と1しか無いような論理演算の場合には、出力層のニューロン数は1個でよい。

＜入力層＞

入力層のニューロン数は、文字などのパターン認識の場合は各パターンの識別に必要な特徴ベクトルの次元数である。一般に次元数が多いほど、パターン識別能力は向上する。その反面、誤差関数の局所的最小値に陥る可能性は増す。そこで、教師パターンを安定してパターン認識できるようになるまで次元数を少しづつ増やすというのが定石である。

＜中間層＞

中間層の数や中間層のニューロン数についても、まずは中間層を1つにして、中間層のニューロン数を少しづつ増やし、教師信号を安定してパターン識別できる条件を見つけてみる。それで、パターン識別が安定しなければ、中間層を2層にして、中間層のニューロン数を少しづつ増やし、教師信号が安定してパターン識別できる条件を見つければいい。中間層が2つの場合は、2つの中間層のニューロン数は同程度でパターン識別される例が多い。中間層を2層にしてうまく行かない場合、特徴ベクトルが適切か再検討してみるとよい。(逆に、層数・ニューロン数を高めに設定し逐次減らしていく方法も提案されている)

(b)繰り返し学習回数

繰り返し学習回数は、解くべき問題や学習係数などの他のパラメータとの関係で適正な値が決まるので、理論値は存在しないが、1000～10万回の範囲が一般的である。誤差がどの程度になれば計算を打ち切るかを決め、繰り返し学習回数は1万回くらいの大きな数にしておくのが無難である。問題は、繰り返し学習させても誤差の打ち切り条件を満たす範囲に収束しない場合である。この場合、特徴ベクトルの次元を増やしてみると良い。

(c)興奮性の信号値、抑制性の信号値

入出力関数として、シグモイド関数を用いるため、各入力の出力値がそれぞれ0, 1になるためには、入力の荷重和が、-∞～∞となる必要があるが、これはありえない。そこで教師信号として、出力層のニューロンの出力値に0と1の代わりに、0.1 と 0.9 の値が用いられる場合が多い。

(d)学習係数

学習係数は大きいほど学習速度が速くなるのが一般的である。しかしながら、誤差をシナプス結合荷重の関数を見た場合、最急降下法で最小値の探索を行う誤差逆伝播法では、学習係数が大きすぎると、次の探索点が行きすぎてしまうこともありえる。この場合、誤差が増えたり、振動する。

(e)その他

学習係数が小さいほど振動などの問題が起こりにくい、しかしながら、修正量が小さいと学習速度が遅くなる。そこで探索点の動きが振動しにくい程度に学習速度を上げるために慣性項を付与してシナプス結合荷重の修正を行う方法が用いられている。この方法をモーメント法(moment method)と呼んでいる。この項の係数として pybrain では momentum が実装されており、この値は1程度が好ましいとされる。

誤差逆伝播学習法の問題点

誤差逆伝播法の問題点を整理してみる。

1. シナプス結合荷重の初期値などの計算条件によっては誤差関数の値が大域的最小値で収束せず、局所的最小値へ収束する。この問題はネットワークの規模が大きいほど生じやすい。
2. シナプス結合荷重の修正に伴う誤差関数の値の単調減少性が保証されていない。
3. 最急降下法を用いているので、解が最適店に近づくと修正量が少なくなり学習速度が遅くなる。この問題もネットワークの規模が大きいほど生じやすい。
4. 層の数・各層のニューロン数、その他のパラメータを試行錯誤的に求める必要がある。
5. 中間層のニューロンの活性・不活性の意味の解釈が困難なため、中間層をブラックボックス的に扱わざるをえない場合が多い。

Pybrain での実験

実験1＜0と1での AND 演算＞

ここでは入力として、0か1かの値を入れ、論理演算 AND を学習させている。

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

#################Create Network ################
from pybrain.tools.shortcuts import buildNetwork
net = buildNetwork(2, 2, 1)
print net

################# Set Trainning DataSet ################
from pybrain.datasets import SupervisedDataSet
ds = SupervisedDataSet(2, 1)
ds.addSample((0.0,0.0),0.0)
ds.addSample((1.0,0.0),0.0)
ds.addSample((0.0,1.0),0.0)
ds.addSample((1.0,1.0),1.0)

################# Trainning ################
from pybrain.supervised.trainers import BackpropTrainer
trainer = BackpropTrainer(net, ds, learningrate=0.01, momentum=0.99)

for i in range (2000):
   trainer.train()
   if abs(float(net.activate([1.0,1.0]))-1.0) <=0.0001:
      print "["+str(i)+"]ERROR="+str(float(net.activate([1.0,1.0]))-1.0)
      break

################## Result ##################
print net.activate([0.0,0.0])
print net.activate([1.0,0.0])
print net.activate([0.0,1.0])
print net.activate([1.0,1.0])
```

[結果]

```
foo@foo:~/python/NN$ python pbtest2.py
[323]ERROR=-4.97581384725e-05
[-0.06265254]
[ 0.01817703]
[ 0.07224294]
[ 0.99995024]
```

実験2＜1と100での AND 演算＞

実験1のソースの「0」を「1」に、「1」を「100」にして実験を行った。その結果以下のようになり、適切に学習されていないことを示す。

[結果]

[24.55424829]

[24.55424829]

[24.55424829]

[24.55424829]

また、NN は0～1の範囲でしか出せないのでは？という疑問に関しては pybrain のネットワークレイヤが関係している。今回使用したコマンドは net = buildNetwork(2, 2, 1)で、これにより出来たネットワークは以下の通り。

FeedForwardNetwork-8

Modules:

[<BiasUnit 'bias'>, <LinearLayer 'in'>, <SigmoidLayer 'hidden0'>, <LinearLayer 'out'>]

Connections:

[<FullConnection 'FullConnection-4': 'bias' -> 'out'>,

<FullConnection 'FullConnection-5': 'bias' -> 'hidden0'>,

<FullConnection 'FullConnection-6': 'hidden0' -> 'out'>,

<FullConnection 'FullConnection-7': 'in' -> 'hidden0'>]

このように入力層と出力層はリニアレイヤになっているため、出力値が1を超えたり、入力に1以上の値を入れても受け付けていると考えられる。しかし、一般的なニューラルネットワークでは0～1の範囲で認識を行うものらしいので、この範囲に収まるように値を正規化して使うべきだと考える。また、今回の例では出力層直前の隠れ層のニューロン数が2つであり、これはシグモイド関数を使ったニューロンであることから、結果として出力層に現れるパターンは、両方活性（1通り）・片方活性（2通り）・両方不活性（1通り）の4通りのはずである。これを実験1と同じように、入力を0と1にし確かめてみる。

実験3＜1と0での AND 演算の出力範囲＞

様々な値を入れてみてどうなるかを実験した。

[結果]

[431]ERROR=-8.26213096752e-05

net.activate([0.0,0.0]) [-0.03916597]

net.activate([1.0,0.0]) [0.05218795]

net.activate([0.0,1.0]) [0.01631141]

net.activate([1.0,1.0]) [0.99991738]

****

net.activate([100.0,100.0]) [0.61020139]

net.activate([100.0,1.0]) [0.06516616]

net.activate([1.0,100.0]) [0.61020139]

net.activate([1.0,1.0]) [0.99991738]

****

net.activate([0.3,0.3]) [0.11717904]

net.activate([0.3,0.7]) [0.30937375]

net.activate([0.7,0.3]) [0.19031173]

net.activate([0.7,0.7]) [0.53093559]

****

net.activate([0.6,0.6]) [0.40218351]

net.activate([0.5,0.5]) [0.29048132]

net.activate([0.4,0.4]) [0.19580215]

net.activate([0.3,0.3]) [0.11717904]

****

net.activate([10.0,10.0]) [0.92490158]

net.activate([100.0,100.0]) [0.61020139]

net.activate([1000.0,1000.0]) [0.61020139]

net.activate([10000.0,10000.0]) [0.61020139]

net.activate([100000.0,100000.0]) [0.61020139]

****

net.activate([-10.0,-10.0]) [-0.10659097]

net.activate([-100.0,-100.0]) [-0.10558466]

net.activate([-1000.0,-1000.0]) [-0.10558466]

net.activate([-10000.0,-10000.0]) [-0.10558466]

net.activate([-100000.0,-100000.0]) [-0.10558466]

以上のことから、出力値はおおよそ-0.10558466～0.99991738 の範囲にあることが予想される。また、予想では4通りの値しか出ないと思っていたが、シグモイド関数は連続であり厳密なしきい値でないため、このような値が出ていると考えられる。今回の実験では、出力値の範囲は、教師信号の範囲とほぼ等しいがこれは、たまたまであると思う。学習結果によっては、教師信号の範囲を超えること、すなわち誤差増加や振動があることに注意したい。入力をかなり大きくした結果 net.activate([100000.0,100000.0]) [0.61020139] が最大で無いことを考えると、マイナスの重みが存在している可能性が高い。

---

[1]吉富康成,シリーズ非線形科学入門ニューラルネットワーク,朝倉書店(2002)