5.2〜5.4 2層パーセプトロン

平成 28 年 9 月 11 日

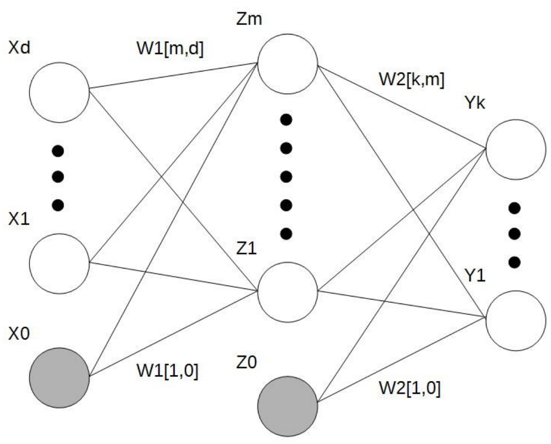
概 要

PRML の「5.1 フィードフォワードネットワーク関数」「5.2 ネットワーク訓練」「5.3 誤差逆伝播」についての実装と考察.

目 次

1　問題設定

以下のような多層パーセプトロンについて考える.



これを用いることで, 回帰, 分類を行える.

2　アルゴリズム

2.1　回帰

簡単のため単一出力のニューラルネットワーを考える.
目標変数 t が, x に依存する平均をを持つガウス分布に従うとする.

$$p(t|x, \mathbf{w}) = N(t|y(x, \mathbf{w}), \beta^{-1}) \quad (5.12)$$

平均は, ニューラルネットワークの出力, β は精度である.
N 個の独立同分布に従う観測値 $X = \{x_1, \ldots, x_N\}$ と対応する目標値 $\boldsymbol{t} = \{t_1, \ldots, t_N\}$ が与えられたとき, 尤度関数は

$$p(\mathbf{t}|X, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|x_n, \mathbf{w}, \beta)$$

となるため, 負の対数をとることで誤差関数は

$$\frac{\beta}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2} + \frac{N}{2} \ln(2\pi) \quad (5.13)$$

から二乗和誤差

$$E(\mathbf{w}) = \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2$$

が出てくる. これを微分することで W_{ML} を得る (誤差関数の極小点が得られる). また, これが求まった後, β についても最適化でき

$$\frac{1}{\beta_{ML}} = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}_{ML}) - t_n\}^2 \quad (5.14)$$

2.2 分類

簡単のため 2 クラス分類, 単一出力のニューラルネットワークを考える.
$t=1$ で $C_1, t=0$ で C_2 を表す. ここではニューラルネットワークの出力を

$$y = \sigma(a) = \frac{1}{1+exp(-a)} \quad (5.19)$$

とする. これは $p(C_1|x)$ と解釈でき, $p(C_2|x)$ は $1-y(x,W)$ となる. このとき目標の確率分布は

$$p(t|x,\mathbf{w}) = y(x,\mathbf{w})^t\{1-y(x,\mathbf{w})\}^{1-t} \quad (5.20)$$

ここからは, 交差エントロピー誤差関数

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\} \quad (5.21)$$

が出てくる.

2.3 5.2.4 勾配降下最適化

単純なアプローチとして, 勾配降下法があり.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E(\mathbf{w}^{(\tau)}) \quad (5.41)$$

がある. これのオンライン版として逐次的勾配降下法

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E_n(\mathbf{w}^{(\tau)}) \quad (5.43)$$

があるが, こちらのほうが性能がいい.

2.4 5.3.1 誤差関数微分の評価

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \quad (5.44)$$

とすると,

$$E_n(\mathbf{w}) = \frac{1}{2}\sum_k (y_{nk}-t_{nk})^2 \quad (5.46)$$

となる.
一般のフィードフォワードネットワークでは, 各ユニットの入力は

$$a_j = \sum_i w_{ij} z_i \quad (5.48)$$

で出力は

$$z_j = h(a_j) \quad (5.49)$$

の形であらわされる. ここで

$$\frac{\partial E_n}{\partial w_{ij}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} \quad (5.50)$$

であるため.

$$\delta_j = \frac{\partial E_n}{\partial w_{ij}} = \quad (5.51)$$

とすると,

$$\frac{\partial a_j}{\partial w_{ij}} = z_i \quad (5.52)$$

より,

$$\frac{\partial E_n}{\partial w_{ij}} = \delta_j z_i \quad (5.53)$$

となる.
正準連結関数を活性化関数に用いた出力ユニットでは

$$\delta_k = y_k - t_k \quad (5.54)$$

となることから, 隠れユニットの δ を評価するには

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \quad (5.55)$$

となるので, これを計算すると

$$\delta_j = h^{'}(a_j)\sum_k w_{kj}\delta_k \quad (5.56)$$

となり, 逆伝播公式が得られる.

ニューラルネットワーク

1. 入力ベクトル $\mathbf{x}_n$ をネットワークに入れ, ネットワーク上を順伝播させ, すべての隠れユニットと出力ユニットの出力を求める.
2. すべての出力ユニットの δ_k を評価する.
3. 2 の δ を逆伝播させ, ネットワークのすべての隠れユニットの δ_j を得る.
4. 2,3 の δ を用いて, 誤差関数の微分を計算する.
5. パラメータ $\mathbf{w}$ を更新する.
6. 1 に戻る. ただし, 収束条件を満たせばこれを終了する.

3 回帰

3.1 コード

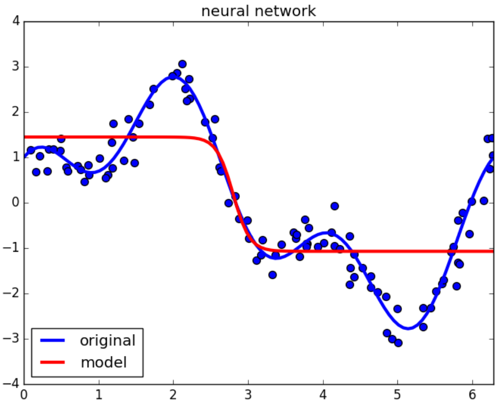
逐次的勾配降下法 (kaiki.py)

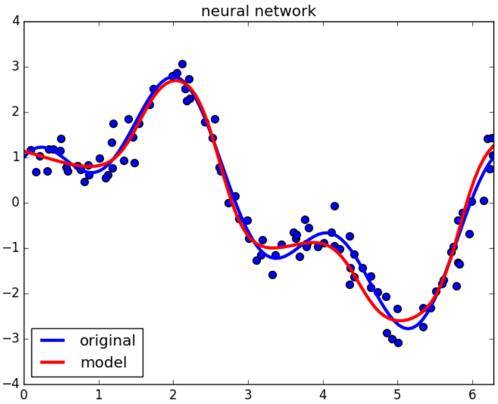
```
"""ニューラルネットワーク構築"""
#入力層3
X=np.zeros(2)
#入力層3->隠れユニットM
M=10
W1=np.random.rand(M,2)
#隠れユニットM
def h(z):
```

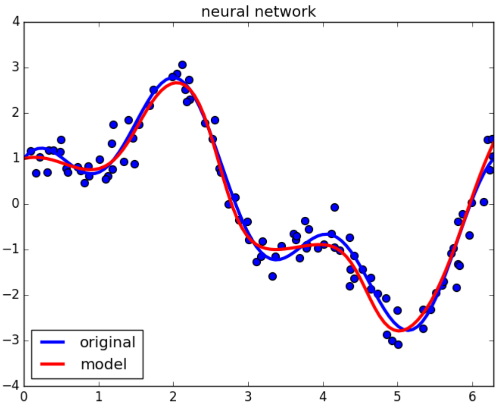
```
  return np.tanh(z)
Z=np.zeros(M)
#隠れユニットM->出 力 層1
W2=np.random.rand(M)
#出力層1
Y=0

"""パラメータの決定"""
delta_2=0
def dh(z):
  return (1-h(z)**2)
delta_1=np.zeros(M)

roop=0
eta=10**(-2)
while roop<500000:
  n=roop%N
  #X=(1,x[n])
  X=x[n,:]
  a=dot(W1,X)
  Z=h(a)
  #バイアス項
  Z[0]=1
  Y=dot(W2,Z)

  delta_2=Y-t[n]
  delta_1=dh(a)*W2*delta_2

  W1-=eta*outer(delta_1,X)
  W2-=eta*delta_2*Z
  roop+=1

def model(z):
  X=z
  a=dot(W1,X)
  Z=h(a)
  #バイアス項
  Z[0]=1
  Y=dot(W2,Z)
  return Y
```

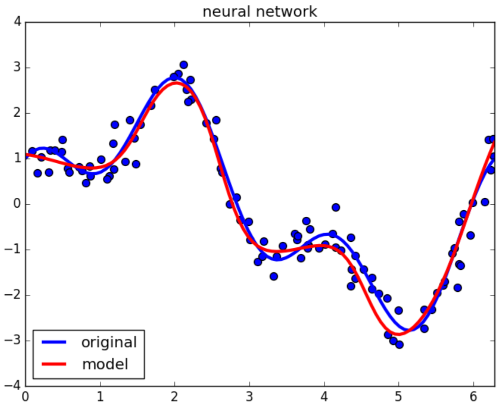
3.2 結果

隠れ層のノード数を 2,10,20,100 と変えて実験してみた.









ノード数が 2 のときは回帰が成功しなかった．逆に，ノード数が 100 のときは過学習を起こさなかった．これは，ループを 500000 回で切っている (早期終了となっている) せいかもしれないが，よい結果である．

4 分類

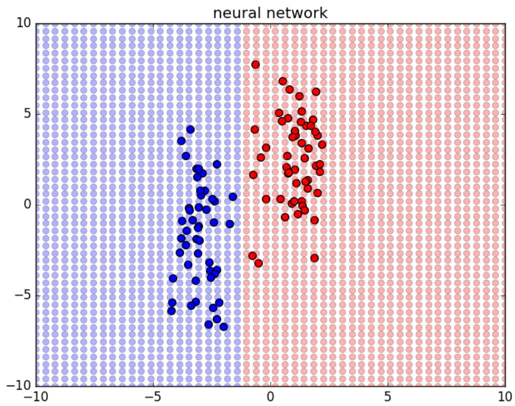
4.1 コード

逐次的勾配降下法 (bunnrui.py)

```
"""ニューラルネットワーク構築"""
#初期化
#入力層3
X=np.zeros(3)
#入力層3->隠 れ ユ ニ ッ ト M
M=10
W1=np.random.rand(M,3)
#隠れユニットM
def h(z):
  return np.tanh(z)
Z=np.zeros(M)
#隠れユニットM->出 力 層 1
W2=np.random.rand(M)
#出力層1
def sig(z):
  return 1/(1+np.exp(-z))
Y=0

"""パラメータの決定"""
delta_2=0
def dh(z):
  return (1-h(z)**2)
delta_1=np.zeros(M)

roop=0
eta=10**(-2)
while roop<100000:
  n=roop%N
  X=x[n,:]
  a=dot(W1,X)
  Z=h(a)
  #バイアス項
  Z[0]=1
  Y=sig(dot(W2,Z))
```

```
  delta_2=Y-t[n]
  delta_1=dh(a)*W2*delta_2

  W1-=eta*outer(delta_1,X)
  W2-=eta*delta_2*Z
  roop+=1

def model(z):
  X=z
  a=dot(W1,X)
  Z=h(a)
  #バイアス項
  Z[0]=1
  Y=sig(dot(W2,Z))
  return Y
```

4.2 結果



5 まとめ

ニューラルネットワークは今までのベイズ線形回帰と比べて, 過学習を起こしにくい感じがする. ただ, 確率的な答えを出すわけではない点が残念である. ただ, 拡張は簡単な気もする.