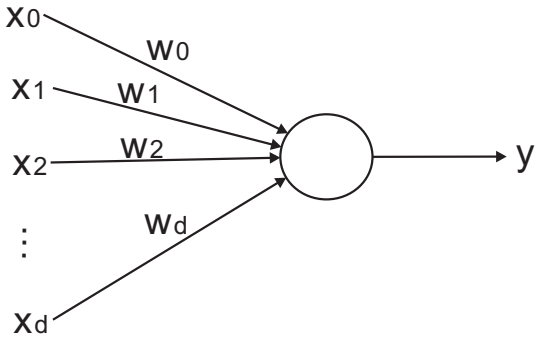
パターン認識
Pattern Recognition

佐藤真一
Shin'ichi Satoh

国立情報学研究所
National Institute of Informatics

May 30, 2023

線形識別関数



$$g(x) = w_0 + \sum_{j=1}^{d} w_j x_j.$$

線形識別関数: まとめ

識別器	損失関数	ソルバー
Perceptron	$J = \sum_{x \in \tilde{\mathcal{X}}} (-w^T x)$	SGD
MSE	$J_p = \frac{1}{2}(\hat{w}^T \hat{x_p} - b_p)^2$	Pinv
Widrow-Hoff	$J_p = \frac{1}{2}(\hat{w}^T \hat{x_p} - b_p)^2$	SGD
Neural network	$J_p = \frac{1}{2}(f(\hat{w}^T \hat{x_p}) - b_p)^2$	SGD

パーセプトロン (再掲)

Rosenblatt, 1957[1]
全学習サンプルについて $w^T x > 0$ となるように w を決定したい
パーセプトロン損失関数:

$$J(w) = \sum_{x \in \tilde{\mathcal{X}}} (-w^T x)$$

ただし $\tilde{\mathcal{X}}$ は識別誤りとなっているすべてのサンプル
J は常に非負であり、これをゼロにしたい (すなわち、$\tilde{\mathcal{X}}$ を空集合としたい)

[1] F. Rosenblatt, The perceptron - A perceiving and recognizing automaton, Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York, January, 1957.

パーセプトロン (再掲)

J の勾配 (gradient) は

$$\nabla J = \sum_{x \in \tilde{\mathcal{X}}} (-x)$$

なので、最急降下法 (勾配降下法, Gradient Descent, もしくは batch Gradient Descent) により重みベクトルを更新可能

$$w(k+1) = w(k) - \rho \nabla J = w(k) + \rho \sum_{x \in \tilde{\mathcal{X}}} x.$$

ρ は学習率 (learning rate) と呼ばれる

最小二乗誤差法と疑似逆行列 (再掲)

以下とすると

$$X = [\hat{x_1}\ \hat{x_2}\ \cdots\ \hat{x_n}]^T$$
$$b_i = [b_{i1}\ b_{i2}\ \cdots\ b_{in}]^T\ (i = 1, \cdots, c)$$

損失関数は

$$J(\hat{w_1}, \hat{w_2}, \cdots, \hat{w_c}) = \frac{1}{2}\sum_{i=1}^{c} \|X\hat{w_i} - b_i\|^2$$
$$\frac{\partial J}{\partial \hat{w_i}} = X^T(X\hat{w_i} - b_i) = 0$$
$$X^T X\hat{w_i} = X^T b_i$$
$$\hat{w_i} = (X^T X)^{-1} X^T b_i$$

これは $\|Xw_i - b_i\|^2$ の最小二乗 (MSE) 解を与える

Widrow-Hoff 法/LMS 法 (再掲)

ここで最急降下法を考える

$$\hat{w}_i(k+1) = \hat{w}_i(k) - \rho \frac{\partial J}{\partial \hat{w}_i} = \hat{w}_i(k) - \rho \nabla_i J$$

$$\Delta \hat{w}_i = -\rho \nabla_i J$$

確率的最急降下法を考えてもよい

$$\Delta \hat{w}_i = -\rho \frac{\partial J_p}{\partial \hat{w}_i}$$

Widrow-Hoff 法/LMS 法 (再掲)

$g_i(x_p)$ を g_{ip} と記載すると

$$\frac{\partial J_p}{\partial \hat{w}_i} = \frac{\partial J_p}{\partial g_{ip}} \frac{\partial g_{ip}}{\partial \hat{w}_i}$$

ただし

$$\frac{\partial J_p}{\partial g_{ip}} = g_{ip} - b_{ip} = \varepsilon_{ip} \quad (\because J_p = \frac{1}{2} \sum_{i=1}^{c} (\hat{w}_i{}^T \hat{x_p} - b_{ip})^2)$$

$$\frac{\partial g_{ip}}{\partial \hat{w}_i} = \hat{x_p} \quad (\because g_{ip} = g_i(x_p) = \hat{w}_i{}^T \hat{x_p})$$

従って

$$\frac{\partial J_p}{\partial \hat{w}_i} = (g_{ip} - b_{ip})\hat{x_p} = \varepsilon_{ip}\hat{x_p}$$

Widrow-Hoff 法/LMS 法 (再掲)

更新式は以下の通り

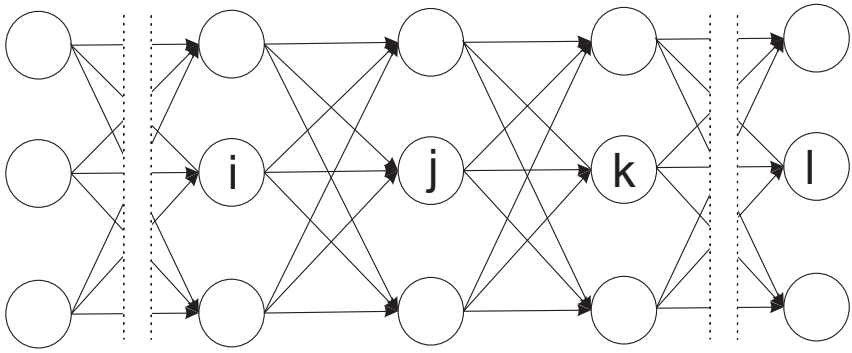
$$\begin{aligned}\Delta \hat{w}_i &= \frac{\partial J_p}{\partial \hat{w}_i} \\ &= -\rho \varepsilon_{ip} \hat{x_p} \\ &= -\rho (g_{ip} - b_{ip}) \hat{x_p} \\ &= -\rho (\hat{w_i}^T \hat{x_p} - b_{ip}) \hat{x_p}.\end{aligned}$$

これを Widrow-Hoff 法 もしくは LMS 法 (least-mean-squared) と呼ぶ

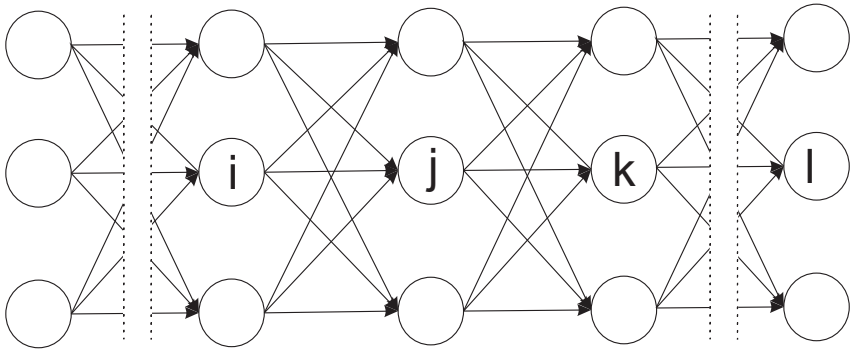
ニューラルネットワーク

- 脳は膨大な数の神経細胞のネットワークによる情報処理システムであり、パターン認識等の知的処理を極めて柔軟に実現可能
- (人工) ニューラルネットワークは、ソフトウェアなどで人工的に実現した神経細胞とネットワークによりパターン認識等を実現を目指した計算モデル
- ニューロンや脳の機能や動作原理の解明という生理学的側面とニューラルネットワークにより実現される並列分散処理の原理の解明という情報科学的側面
- McCulloch-Pitts model (1943), Hebb 則 (1949)
- フィードフォワードネットワーク, リカレント型ネットワーク
- 教師あり学習、教師なし学習

多層ニューラルネットワーク



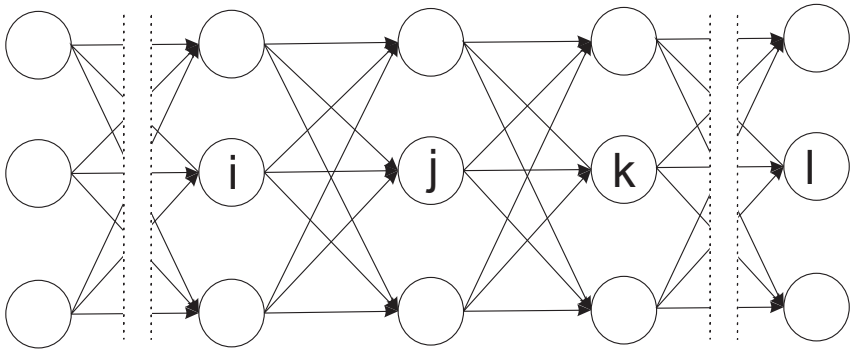
多層ニューラルネットワーク



$$y = [w_{kl}]\,[w_{jk}]\,[w_{ij}]\,x$$

$$= \left[\sum_{j,k} w_{ij} w_{jk} w_{kl}\right] x$$

多層ニューラルネットワーク



$$y = f\left([w_{kl}]\, f\left([w_{jk}]\, f\left([w_{ij}]\, x\right)\right)\right)$$

多層ニューラルネットワーク

単純に線形識別関数・線形識別器を積み重ねるだけではうまくない: 複数の線形関数を何回組み合わせても結果は一つの線形関数になってしまうため
そこで非線形関数 $f(\cdot)$ を導入する

$$h_{jp} = \sum_i w_{ij} g_{ip}$$
$$g_{jp} = f(h_{jp})$$

損失関数は以下の通り

$$J_p = \frac{1}{2} \sum_l (g_{lp} - b_{lp})^2$$
$$J = \sum_p J_p$$

これを勾配降下法で最小化する

$$\frac{\partial J_p}{\partial w_{ij}} = \frac{\partial J_p}{\partial h_{jp}} \cdot \frac{\partial h_{jp}}{\partial w_{ij}}$$

$$\varepsilon_{jp} \stackrel{\text{def}}{=} \frac{\partial J_p}{\partial h_{jp}}$$

$$\frac{\partial h_{jp}}{\partial w_{ij}} = g_{ip} \quad (\because h_{jp} = \sum_i w_{ij} g_{ip})$$

$$\frac{\partial J_p}{\partial w_{ij}} = \varepsilon_{jp} g_{ip}$$

$$\Delta w_{ij} = -\rho \frac{\partial J_p}{\partial w_{ij}} = -\rho \varepsilon_{jp} g_{ip}$$

各ユニットの ε_{jp} の決定法について考える

$$\begin{aligned}\varepsilon_{jp} &= \frac{\partial J_p}{\partial h_{jp}} = \frac{\partial J_p}{\partial g_{jp}} \cdot \frac{\partial g_{jp}}{\partial h_{jp}} \\ &= \frac{\partial J_p}{\partial g_{jp}} \cdot f'(h_{jp}) \quad (\because g_{jp} = f(h_{jp}))\end{aligned}$$

もしユニット j が出力層ならば

$$\frac{\partial J_p}{\partial g_{jp}} = g_{jp} - b_{jp} \quad (\because J_p = \frac{1}{2}\sum_l (g_{lp} - b_{lp})^2).$$

もしユニット j が中間層ならば

$$\frac{\partial J_p}{\partial g_{jp}} = \sum_k \frac{\partial J_p}{\partial h_{kp}} \cdot \frac{\partial h_{kp}}{\partial g_{jp}} \quad (\because h_{kp} = \sum_j w_{jk} g_{jp}).$$

ただし

$$\frac{\partial J_p}{\partial h_{kp}} = \varepsilon_{kp}$$

$$h_{kp} = \sum_j w_{jk} g_{jp}$$

$$\frac{\partial h_{kp}}{\partial g_{jp}} = w_{jk}$$

従って

$$\frac{\partial J_p}{\partial g_{jp}} = \sum_k \frac{\partial J_p}{\partial h_{kp}} \cdot \frac{\partial h_{kp}}{\partial g_{jp}}$$

$$= \sum_k \varepsilon_{kp} w_{jk}$$

多層ニューラルネットワーク

非線形関数 $f(\cdot)$ について考える
閾値関数を考えることはできるが、微分不可能
その代わりに以下のシグモイド関数を考える

$$f(u) = \frac{1}{1 + e^{-u}}$$
$$f'(u) = f(u)(1 - f(u))$$

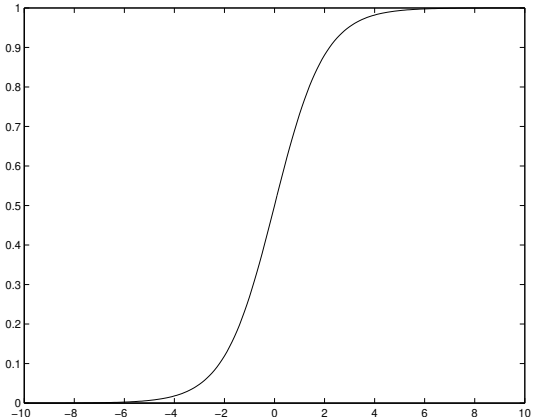
従って

$$f'(h_{jp}) = g_{jp}(1 - g_{jp}) \quad (\because f(h_{jp}) = g_{jp})$$

よって

$$\varepsilon_{jp} = \begin{cases} (g_{jp} - b_{jp})g_{jp}(1 - g_{jp}) & (j \text{ ユニットが出力層の場合}) \\ (\sum_k \varepsilon_{kp} w_{jk})g_{jp}(1 - g_{jp}) & (j \text{ ユニットが中間層の場合}) \end{cases}$$

シグモイド関数



誤差逆伝播法 (Backpropagation)

全体のネットワークはどのように学習するか

- 学習ベクトルが入力されたら、教師信号とネットワークの出力との差分に基づき ε_{jp} を計算する
- これによりまず出力層とその直前の層の間の重みを更新する
- 次いでこれによりその前の層の ε_{jp} を計算し、これにより前の層の重みを更新する
- これを入力層に至るまで繰り返す
- 全体をすべてのデータについて繰り返す
- さらに全体を複数回繰り返す (epoch と呼ばれる)

誤差が出力層から入力層に向けて徐々に伝播していくため、このアルゴリズムは誤差逆伝播法 (backpropagation) と呼ばれる

深層学習 (Deep Learning)

深層学習は多層ニューラルネットワークの一種であり、特に極めて多数の層を有する
理論的には多層ニューラルネットワークは区分線形識別器とほぼ等価であり、多数のユニットと層を利用することにより、任意に複雑な識別境界を学習可能である
しかしながら、多層になるほど学習は困難となる
いくつかの最近の技術革新が深層学習を可能としている

- 事前学習、オートエンコーダ、restricted Boltzmann machine (RBM)
- Dropout
- 高速計算機 (特に GPU)

深層畳み込みニューラルネットワークも極めて盛んに使われている

- 畳み込み層
- プーリング層
- 全結合層

確率的最急降下法と畳み込み層という二つの極めて重要な技術革新は日本人による

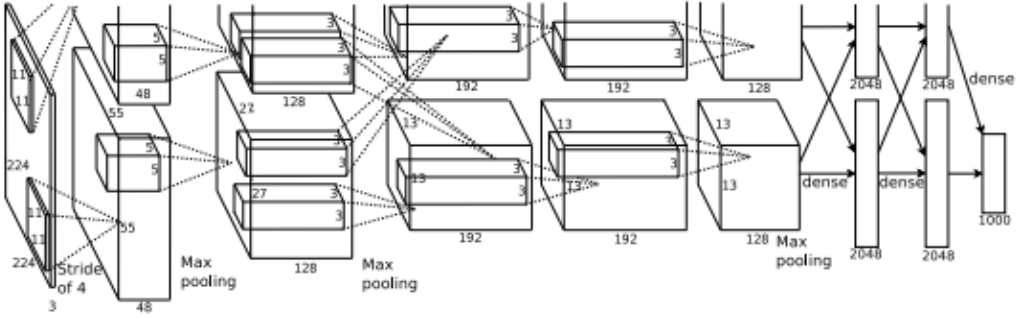


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

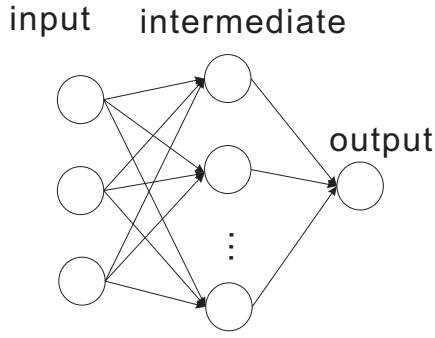
Krizhevsky, Sutskever, and Hinton, NIPS 2012.

宿題

- プログラミング課題と非プログラミング課題を課する
- プログラミング課題か非プログラミング課題かのいずれかを解いて提出すること
- プログラミング課題を解くことを推奨する
- もちろん両方解いてもらえると嬉しい
- 締切は 6/13

宿題

プログラミング課題 1 以下の構造を持つ多層ニューラルネットワークを実装せよ



ネットワークを前回配布した三種類のデータで誤差逆伝播法により学習せよ。正しく識別するために必要な中間層のユニット数について検討せよ。

注意！: 既存のニューラルネットワークツールやライブラリは使わないこと。ユニットの出力、シグモイド関数、重み行列などを自分で定義して実装せよ。例えば

```
outp = sigmoid(sigmoid(inp.dot(Wim)).dot(Wmo))
```

宿題

プログラミング課題 2 (中級) MNIST を識別する MSE 識別器を実装せよ
(私の実装では 86%)

プログラミング課題 3 (余力問題) MNIST を識別するニューラルネットワークを実装せよ。
宿題 1 と同等の構造を利用せよ。
(97.7%)

宿題

非プログラミング課題以下の深層学習に関する技術用語について説明せよ。これらを本講義で説明した多層ニューラルネットワークに組み込む方法について検討せよ。

- dropout
- momentum
- data augmentation

宿題 1

```
import numpy as np
import matplotlib.pyplot as plt
from nonlinear import *

inp = 3  # input vector dim
mid = 10  # num of units of intermediate layer
out = 1  # num of units of output layer
rho = 1  # learning rate
wim = np.random.rand(inp, mid) * 2 - 1  # weights from input to intermediate
wmo = np.random.rand(mid, out) * 2 - 1  # weights from intermediate to output
ax = np.concatenate((np.ones((1, n)), x))  # extended vector
sigmoid = lambda x: 1. / (1. + np.exp(-x))
```

宿題 1

```
xx, yy = np.meshgrid(np.linspace(-1, 1), np.linspace(-1, 1))
xf = xx.flatten()[:, np.newaxis]
yf = yy.flatten()[:, np.newaxis]
axy = np.concatenate((np.ones(xf.shape[0])[:, np.newaxis], xf, yf),
                     axis=1).T
l = (l == 1)
plt.figure()
plt.ion()
```

宿題 1

```
while True:  # loop for each epoch
    for i in range(n):  # loop for each data
        gm = <<compute output of the intermediate layer>>
        go = <<compute output of the output layer>>
        eo = <<compute error of the output layer>>
        em = <<compute error of the intermediate layer>>
        wim = wim - rho * <<update W_{im}>>
        wmo = wmo - rho * <<update W_{mo}>>
```

宿題 1

```
plt.clf()
plt.xlim([-1, 1])
plt.ylim([-1, 1])
plt.plot(x[0, np.where(l == 1)], x[1, np.where(l == 1)], 'bo')
plt.plot(x[0, np.where(l == 0)], x[1, np.where(l == 0)], 'bx')
p = sigmoid(axy.T.dot(wim)).dot(wmo)  # compute classification results
cs = plt.contour(xx, yy, np.reshape(p, xx.shape),
                 levels=[-5, 0, 5], colors='g')
plt.clabel(cs)
# plt.show()
plt.pause(0.01)
```