ニューラルネットワークの歴史と手法2

HubelとWeiselからLeNetまで

公立小松大学

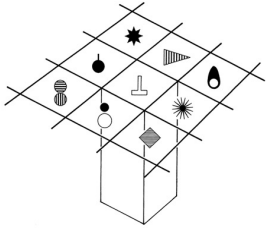
藤田 一寿

記述が不十分なところがあります.
Boltzmann machineとrecurrent neural networkのスライドはまだありません.

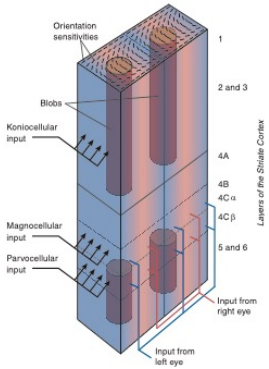
かなりマニアックな部分があります.
必ず元論文をチェックしましょう！！

神経科学の発展とニューラルネットワーク研究

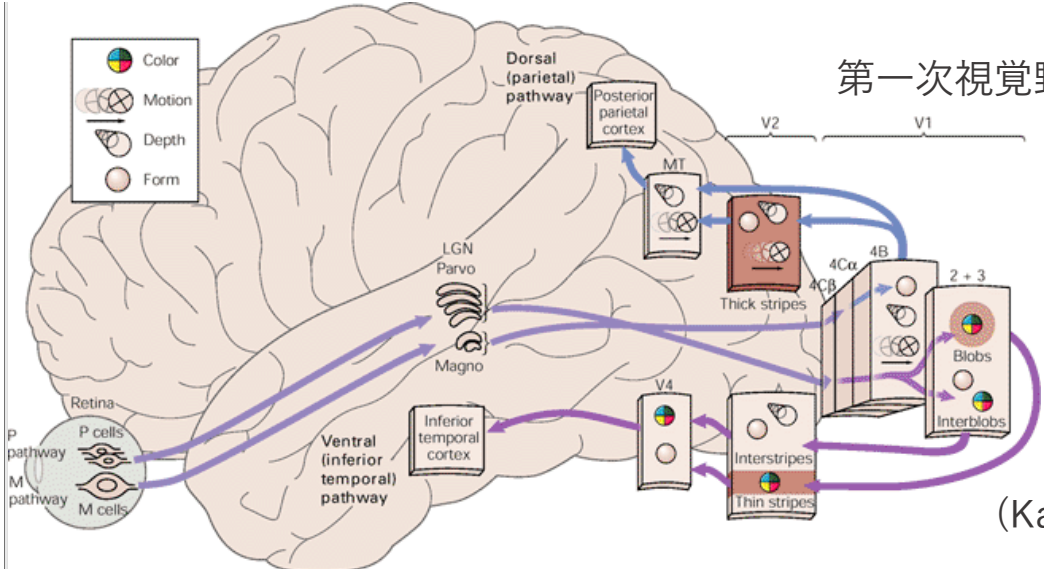
人間の視覚処理



ITの構造 (Tanaka, 2003)



V1の構造 (Carlson)



第一次視覚野：V1

(Kandel)

下側頭葉：IT (inferior temporal cortex)

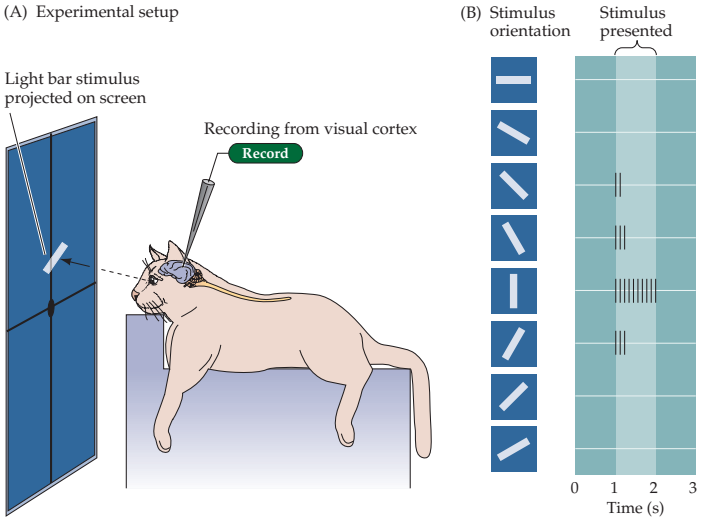
腹側経路

網膜センサ → LGN → V1 角度 → V2 組み合わせ → V4 組み合わせ → IT 基本画像

第一次視覚野の衝撃的な研究

- HubelとWieselの実験（1968年）
  - 第一次視覚野の神経細胞は方向に選択性を持つ．．

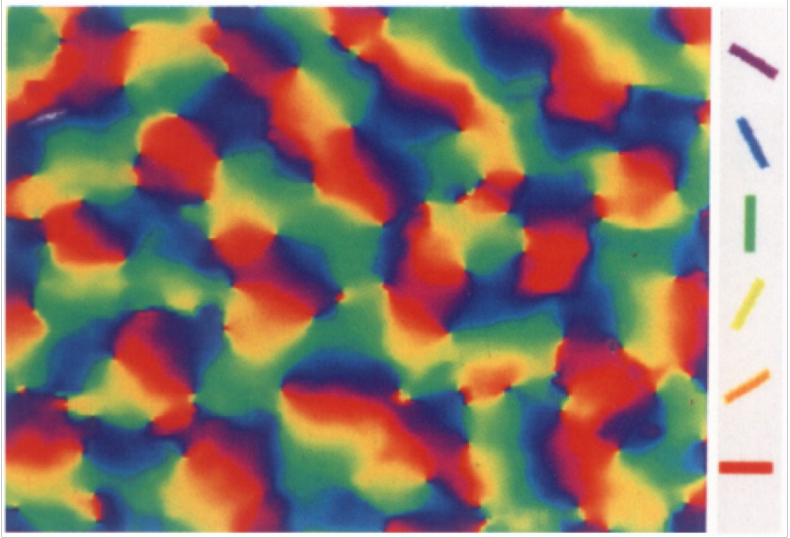
Wiesel　　Hubel



(Purves et al., Neuroscience)

第1次視覚野

- 第1次視覚野では，似た角度を抽出する細胞が隣り合っている．
- さらに，それが見た目の位置が保存された状態で並んでいる（retinotopical map）．
- このマップは，生後，自己組織的に形成されている（生得的ではない）．



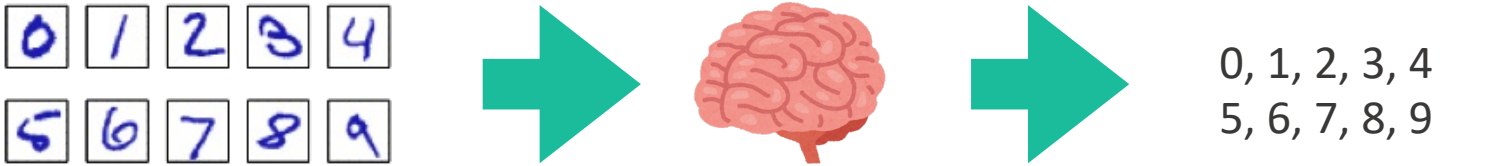
サルの脳の方位選択性

(Blasdel, 1992)

脳の理論的研究とそのアプローチ

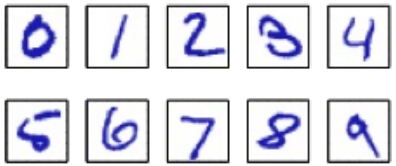
- トップダウン的手法
  - 脳を情報処理機械とみなし，その計算理論を明らかにすることで脳を理解する．
  - 機能から考える．

- ボトムアップ的手法
  - 脳を構成する構成要素とその相互作用により，どのようにして脳の情報処理機能が実現しているのか明らかにする．
  - 部品から考える．



David Marrの３つのレベル（Marr, 1982）

- 計算論
  - 計算の目的はなにか，なぜそれが適切なのか，そしてその実行可能な方略の理論はなにか.
- 表現とアルゴリズム
  - この計算理論はどのようにして実現することができるか，特に入出力の表現はなにか，そして，変換のためのアルゴリズムはなにか.
- ハードウェアによる実現
  - 表現とアルゴリズムがどのようにして物理的に実現されるか.



画像

脳

0, 1, 2, 3, 4
5, 6, 7, 8, 9

数字だとわかる

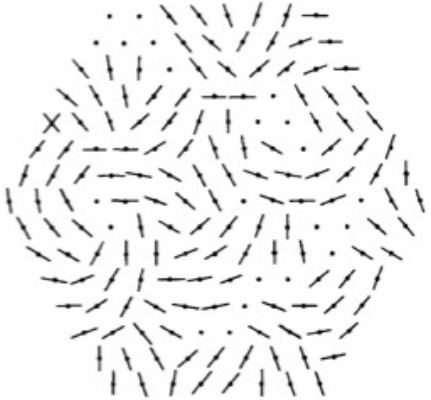
HubelとWieselの研究に関係

- Self-organizing map (von der Marsberg, 1976; Kohonen, 1982)
  - 入力されたベクトルを類似度に応じてd次元マップに射影する.
  - V1の角度マップを再現できる.
  - 似た入力をまとめる能力があるため，量子化などで使われる.
  - また，さらにd次元マップに射影できる特徴から，次元削減にも使われる.
- Neocognitron (Fukushima, 1980)
  - 英語ではBiological cyberneticsの論文が初出である.
  - 畳み込みニューラルネットワークの元祖と言われる.
  - Neocognitronについては深層ニューラルネットワークで取り扱う.
- Sparse coding (Olshausen, 1996)

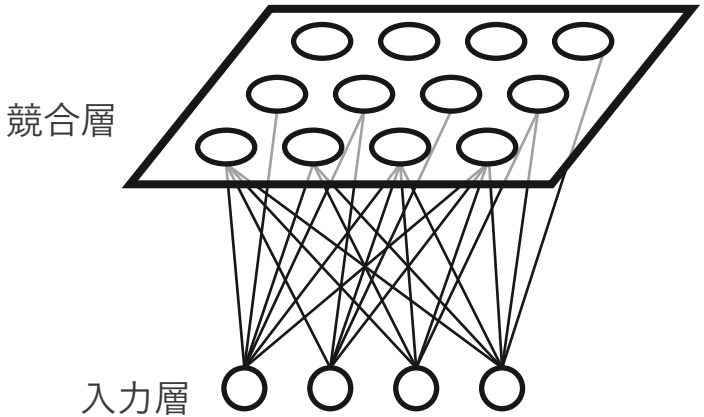
自己組織化マップ

自己組織化マップ (Self organizing map)

- ニューラルネットワークの一種である.
- 第1次視覚野のモデルから出発した(von der Malsburg, 1976).
- Kohonenの自己組織化マップが広く普及した(Kohonen, 1982).
- 入力層と競合層の2層からなる.

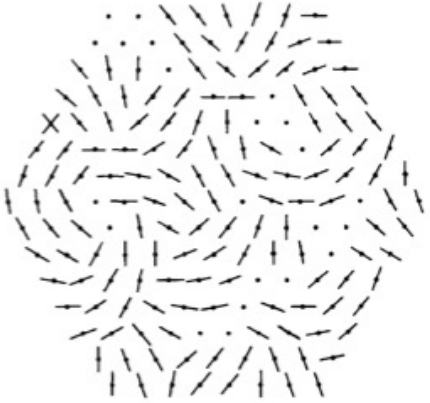


von der Marburgは数理モデルで第1次視覚野の方位選択性細胞のマップを再現した (von der Malsburg, 1973). 線は抽出した線の角度を表す.

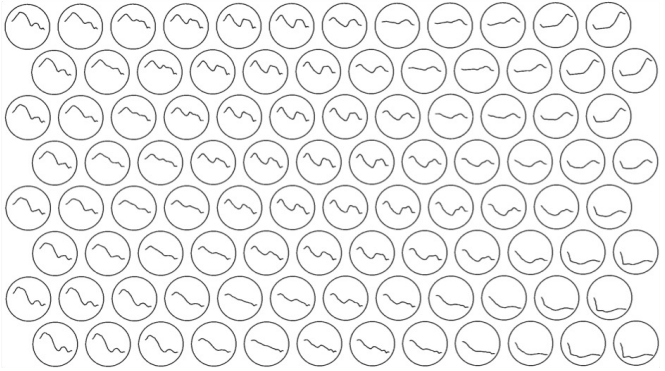


自己組織化マップ (Self organizing map)

- 競合層のニューロンは入力の特徴を抽出する.
- 例えば斜め45度の入力に対し応答するニューロンは，斜め45度の画像が入ったとき発火する.
- 斜め45度の入力に対し応答するニューロンに隣接するニューロンは斜め45度に近い角度，たとえば40度や50度に応答するニューロンが並ぶ.



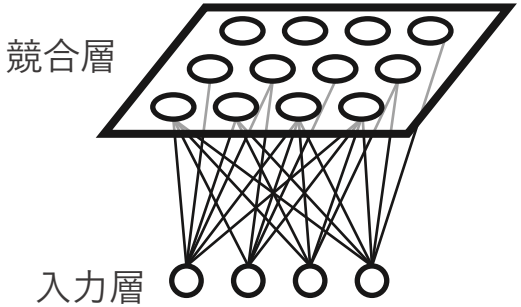
(von der Malsburg, 1973)



波形を自己組織化マップに入れると，似た波形を抽出するニューロン群ができる (Kohonen)

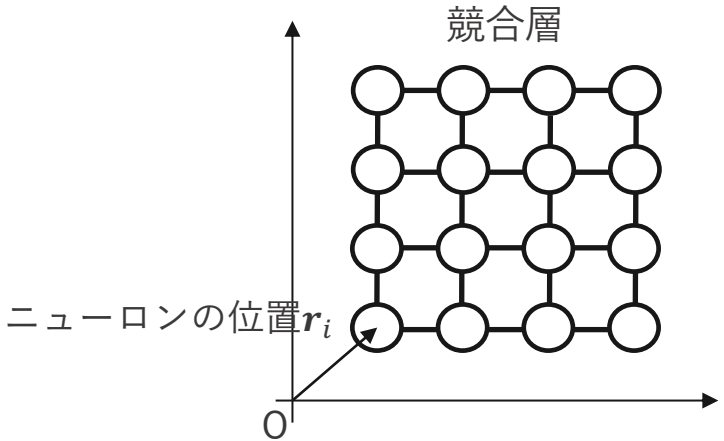
Kohonenの自己組織化マップの構造

- Kohonenの自己組織化マップは入力層と競合層の2層からなる.
- ネットワークはN個のニューロン（ユニット）で構成されており，それぞれが重み$\boldsymbol{w}_i$を持っている.
  - 重みはreference vector, codebook vectorなどとも呼ばれる.
- 競合層では，入力に対し最も近い重みを持つユニットが発火する.
  - winner-take-all（勝者総取り）と呼ぶ.
  - 入力に対し最も近い重みを持つユニットを勝ちユニット（勝ちニューロン）と呼ぶ.



Kohonenの自己組織化マップの構造

- 競合層ではユニットが格子状に配置されている.
- ユニットの隣接関係は格子上での隣接関係で決まる.
  - 重みが最も近い2つのユニットが格子上で隣接でなければのそれらのユニットは隣接関係にない.
- ユニットは競合層上の位置$\boldsymbol{r}_i$に配置されている.



ユニット
重み：$\boldsymbol{w}_i$
競合層上での場所：$\boldsymbol{r}_i$
ユニットの隣接関係：格子で決まる

Kohonenの自己組織化マップのアルゴリズム

1. 入力$\boldsymbol{x}(t)$を提示する.
2. 入力に最も近いユニットを勝者ユニットC_wとする.

$$c = \mathrm{argmin}_i \|\boldsymbol{w}_i - \boldsymbol{x}\|$$

3. 勝者ユニットとそれに隣接するユニットの重みを入力に近づける.

$$\boldsymbol{w}_i^{t+1} \leftarrow \boldsymbol{w}_i^t + h_{ci}(t)\big(\boldsymbol{x}(t) - \boldsymbol{w}_i(t)\big)$$

$h_{ci}(t)$は隣接関数である.

$$h_{ci}(t) = h_0(t) \exp\left(-\frac{\|\boldsymbol{r}_i - \boldsymbol{r}_c\|^2}{\sigma^2(t)}\right)$$

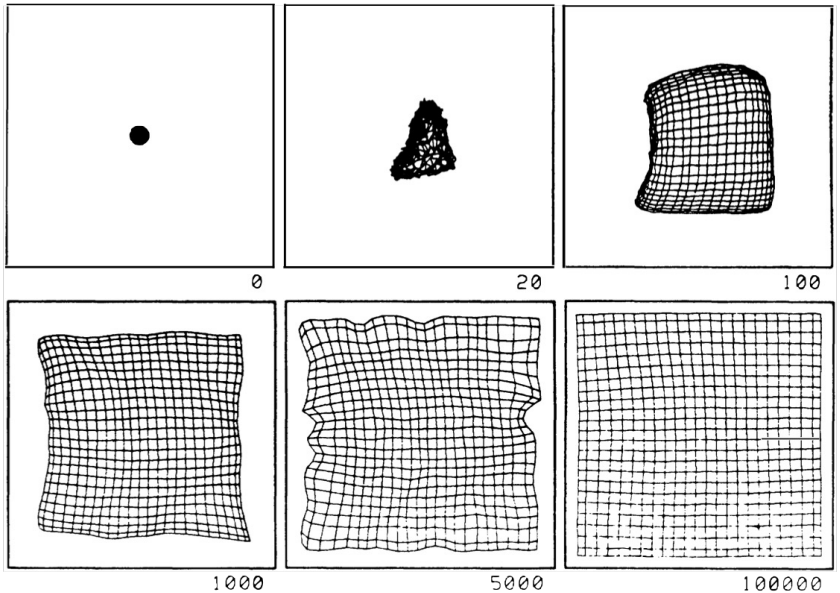
h_0とσは時間的に学習が進むにつれて減衰する.

4. ステップ数tがある値以下ならば1に戻る.

勝者ユニット(winner)は
best matching unitとも言う

Kohonenの自己組織化マップの学習の様子

- 空間に一様に分布する入力を与えたところ，重みは空間全体に広がる．
- ユニットの隣接関係は格子で固定されているので，競合層上で隣り合うユニットは，近い入力に対し応答するようになる．



(Kohonen)

自己組織化マップで何ができるか

- 次元削減
- ベクトル量子化
- データの可視化
- クラスタリング

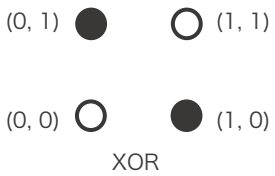
ニューラルネットワークから人工ニューラルネットワークへ

多層パーセプトロン

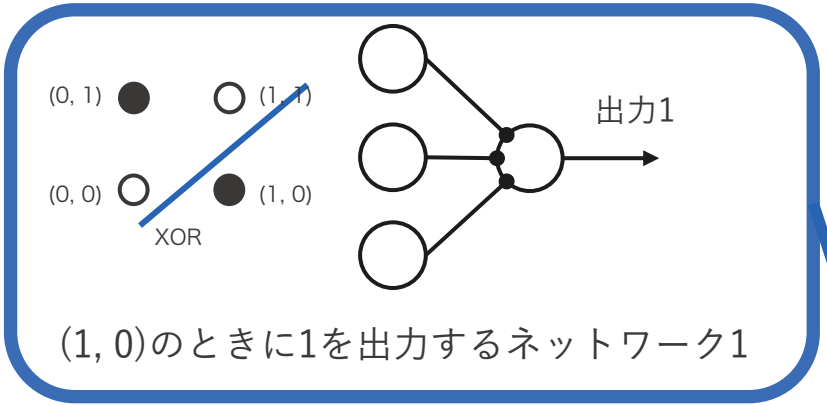
多層パーセプトロン

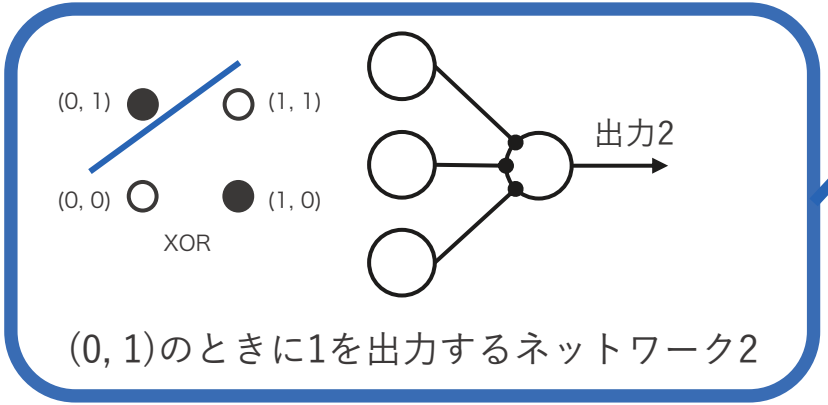
- いわゆるパーセプトロンは線形分離不可能な問題を解けない.
- 解くための一つの方法として多層化がある.
  - Rosenblattはパーセプトロンを多層化することで線形分離不可能な問題を解けるようにした.

多層パーセプトロンでXORを解く

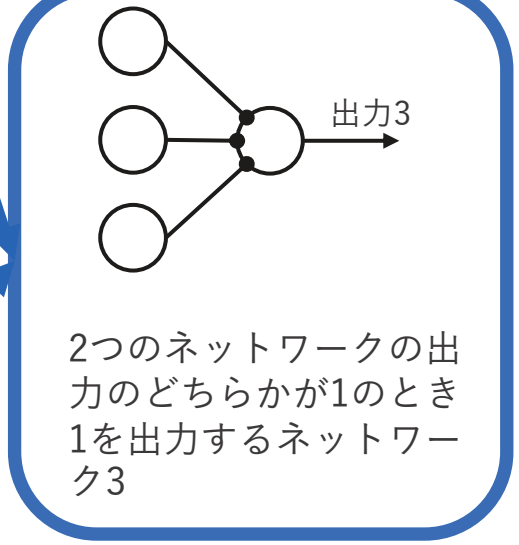


これは線形分離不可能な問題



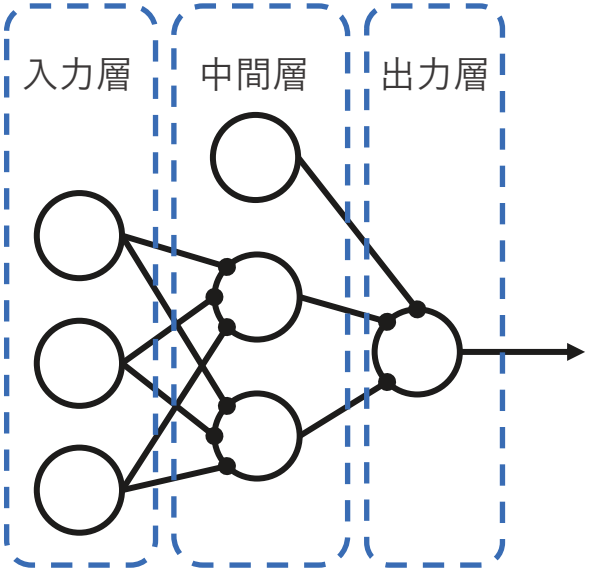


入力	出力1	出力2	出力3
(0, 0)	0	0	0
(0, 1)	0	1	1
(1, 0)	1	0	1
(1, 1)	0	0	0



XORを解くための多層パーセプトロン

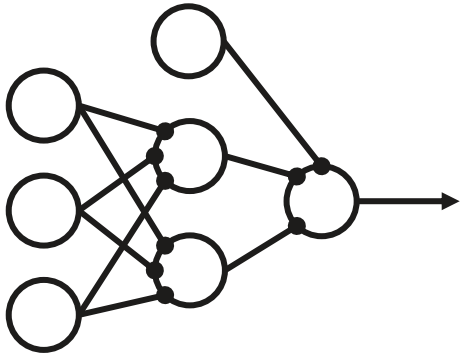
- XORを解くには図のような3層のパーセプトロンを作れば良いことが分かる.
- 入力を生成する層を入力層，出力を生成する層を出力層，入力層と出力層の間の層を中間層もしくは隠れ層という.



入力層を数えず2層ということもある.
ニューロンのことをユニットとも言う.

多層化の問題

- 多層化により線形分離不可能な問題が解ける.
- しかし，多層化したパーセプトロンをどう学習すればよいかわからない.
  - Rosenblattはランダム接続を用いることで，運任せであるが入力を線形分離可能な状態に変換しようとしている.

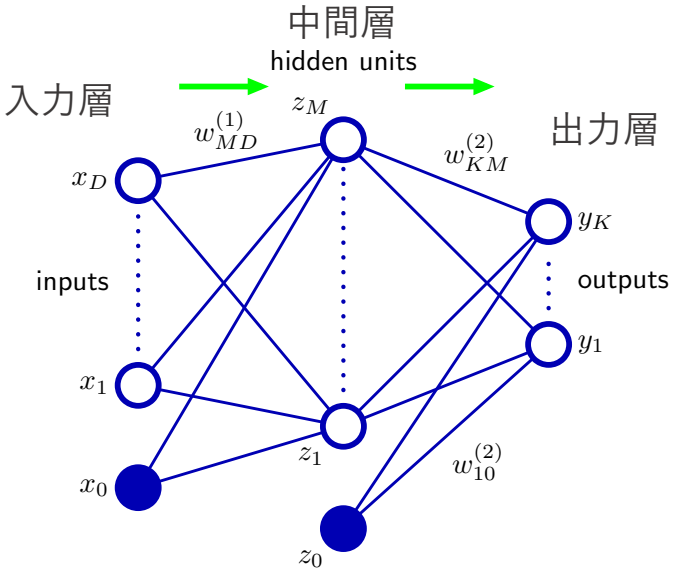


このネットワークを使えばXOR問題は解けるが，どのように自動で重みを決めればよいかわからない.

多層パーセプトロンの数式表現

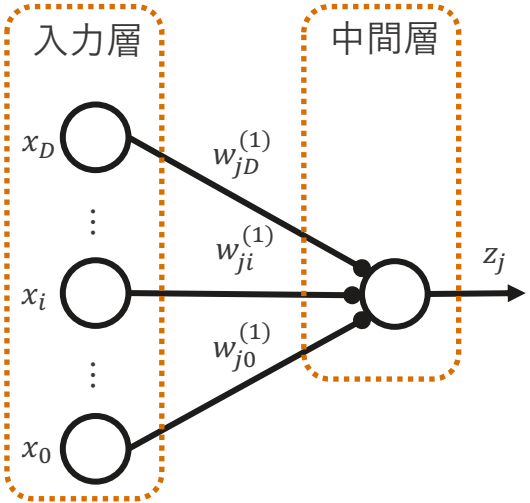
多層パーセプトロンの数式表現

- 図のような3層（2層）のニューラルネットワークを考える.
- このネットワークは入力はD次元ベクトル，出力はK次元ベクトルである.
- 一度に全体を考えるのは難しいので，入力層と中間層から考えていこう.



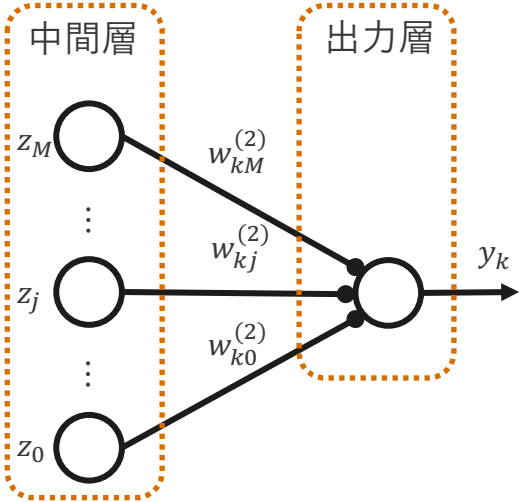
中間層のユニットの出力

- 入力はD次元であるとすると，入力ユニットはD個ある．
- 入力 $\boldsymbol{x}$ はD次元ベクトルで表現できるので入力は次のように表される．
- $\boldsymbol{x} = (x_0, x_1, \ldots, x_i \ldots, x_D)^{\mathrm{T}}$
- x_iが入力ユニットiの出力である．
- x_0は常に1である．これをバイアスという．
- 中間層のユニットjと入力層のユニット間の重みを
- $\boldsymbol{w}_j^{(1)} = \left(w_{j0}, w_{j1}, \ldots, w_{jD}\right)^{\mathrm{T}}$
- とすると，中間層のユニットjの入力 a_j は次のように書ける．
- $a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i = \boldsymbol{w}_j^{(1)^{\mathrm{T}}} \cdot \boldsymbol{x}$
- つまり，入力と重みをかけたものの総和が入力になる．
- 中間層のユニットjの入力 z_jは次のように書ける．
- $z_j = h\left(a_j\right)$
- $h(\cdot)$は活性化関数である．



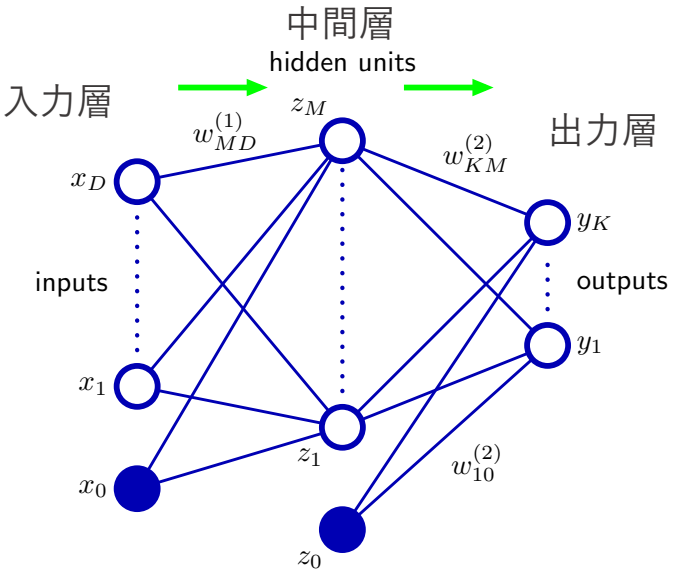
出力層のユニットの出力

- 中間層のユニットがM個あるとする.
- よって，中間層の出力 $\mathbf{z}$ はM次元ベクトルで表現できるので次のように表される.
- $\boldsymbol{z} = \left(z_0, z_1, \ldots, z_j \ldots, z_D\right)^{\mathrm{T}}$
- z_jが中間層のユニットjの出力である.
- z_0は常に1である.
- 中間層のユニットjと入力層のユニット間の重みを
- $\boldsymbol{w}_k^{(2)} = (w_{k0}, w_{k1}, \ldots, w_{kM})^{\mathrm{T}}$
- とすると， 出力層のユニットkの入力 a_k は次のように書ける.
- $a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j = \boldsymbol{w}_k^{(2)^{\mathrm{T}}} \cdot \boldsymbol{z}$
- 中間層のユニットjの入力 z_jは次のように書ける.
- $y_k = \sigma(a_k)$
- $h(\cdot)$は活性化関数である.



多層パーセプトロンの数式表現まとめ

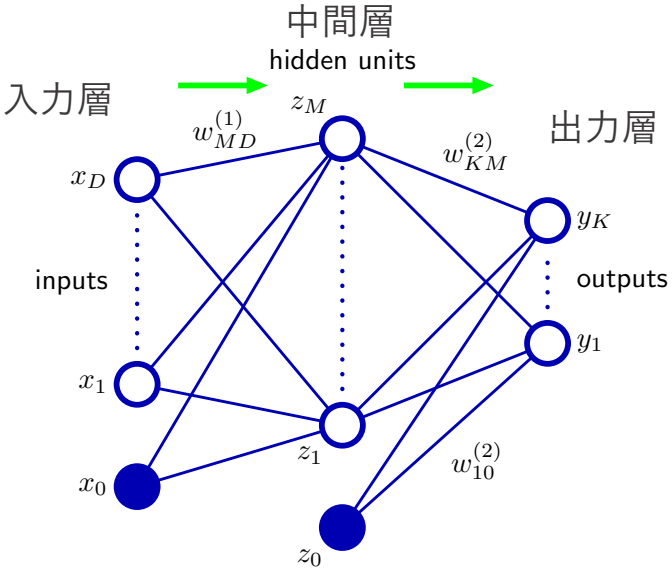
- 図のような3層（2層）のニューラルネットワークを考える.
- 出力層のユニットkの出力 y_kは次のように書ける.
- $a_j = \sum_{i=0}^{M} w_{ji}^{(1)} x_i = \boldsymbol{w}_j^{(1)\mathrm{T}} \cdot \boldsymbol{x}$
- $z_j = h(a_j)$
- $a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j = \boldsymbol{w}_k^{(2)\mathrm{T}} \cdot \boldsymbol{z}$
- $y_k = \sigma(a_k)$



$\boldsymbol{x} = (x_0, x_1, \ldots, x_D)^{\mathrm{T}}$：入力
$\boldsymbol{w}_j^{(1)} = (w_{j0}, w_{j1}, \ldots, w_{jD})^{\mathrm{T}}$：中間層のユニットjと入力層のユニット間の重み
$\boldsymbol{z} = (z_0, z_1, \ldots, z_D)^{\mathrm{T}}$：中間層のユニットの出力
$\boldsymbol{w}_k^{(2)} = (w_{k0}, w_{k1}, \ldots, w_{kM})^{\mathrm{T}}$：出力層のユニットkと中間層のユニット間の重み

出力を1つの式で書いてみる

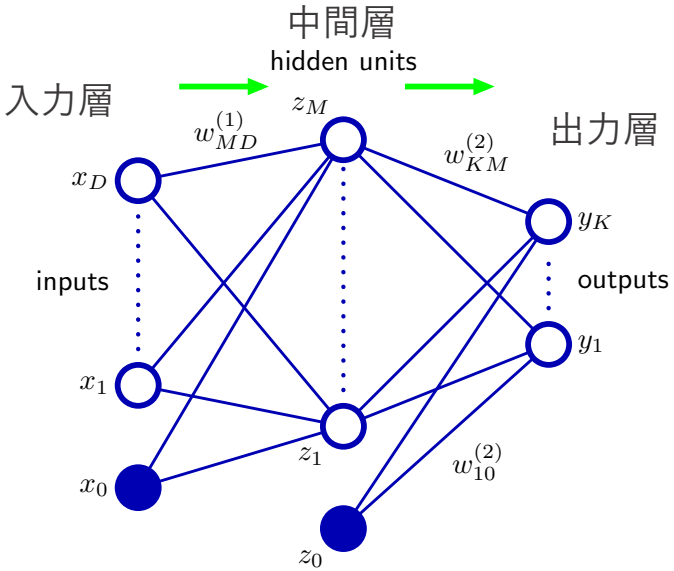
- 出力層のユニットkの出力 y_k 1つの式で書くと次のようになる.
- $y_k = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{M} w_{ji}^{(1)} x_i\right)\right)$
- すべての重みをまとめたベクトルを $\boldsymbol{w}$とすると，出力は $\boldsymbol{w}$と $\boldsymbol{x}$の関数 $y_k(\boldsymbol{w}, \boldsymbol{x})$である.



多層パーセプトロンの誤差関数

多層パーセプトロンの誤差関数

- 多層パーセプトロンは識別機である.
- つまり，多層パーセプトロンは入力に対しラベルyを付ける.
- しかし，多層パーセプトロンは正しいラベルをつけるとは限らない.

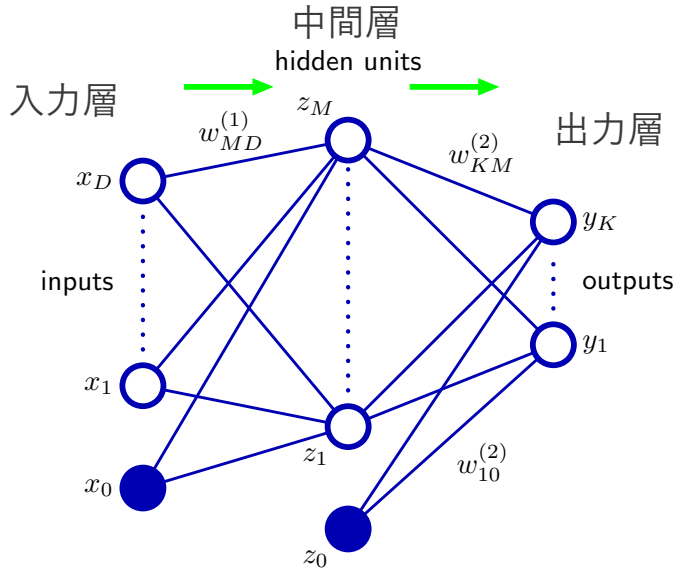


多層パーセプトロンの誤差関数

- ラベル $\boldsymbol{t}_n$ が付いた入力 $\boldsymbol{x}_n$が多層パーセプトロンに入力されたとき，出力が $\boldsymbol{y}(\boldsymbol{w}, \boldsymbol{x}_n) = (y_1, .., y_k, \ldots, y_K)$であったとする．
- このとき，出力がラベルにどれほど近いかを評価したい．
- 近い遠いなので，出力とラベル間の距離の2乗を評価基準としたい．数式で書くと
- $E_n(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{y}(\boldsymbol{w}, \boldsymbol{x}_n) - \boldsymbol{t}_n\|^2$
- となる．

$\frac{1}{2}$は後々数式を格好良くするためのものである．

- これを2乗誤差という．



$\|\cdot\|$はL2ノルムであり，ベクトルの大きさを表す．つまりユークリッド距離を計算している．
$\|(a,b)\|^2 = (a,b)\cdot(a,b)^{\mathrm{T}} = a^2 + b^2$
L2ノルム（ユークリッド距離）でなくても，例えばマンハッタン距離でも良いのではと考える人がいるかも知れない．もちろんどんな距離でも良い．
では，なぜユークリッド距離なのか考えてみると面白い．

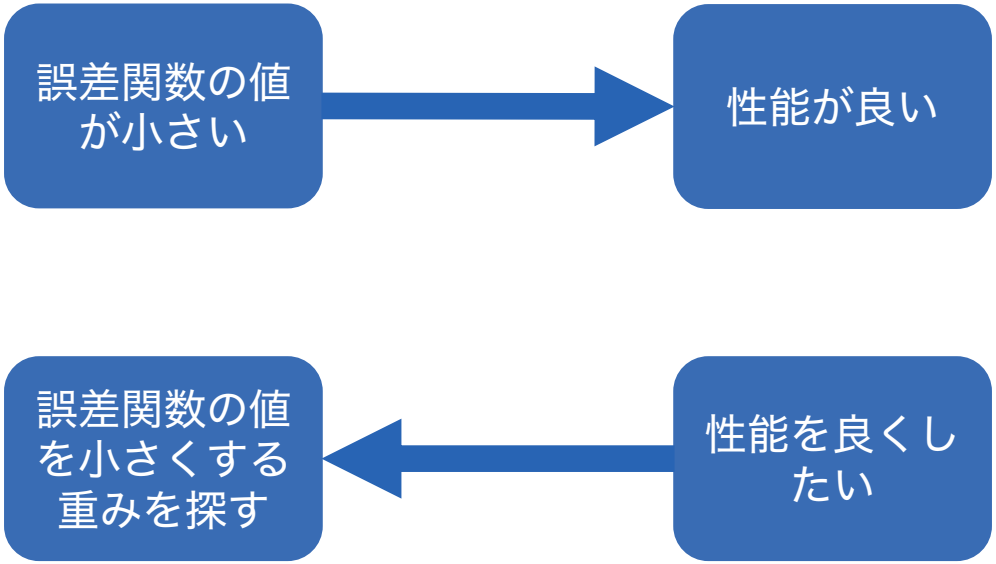
多層パーセプトロンの誤差関数

- $\boldsymbol{x}_n$がN個あるとする.
- N個すべての入力に対する出力とラベルの2乗誤差は
- $E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\|\boldsymbol{y}(\boldsymbol{w}, \boldsymbol{x}_n) - \boldsymbol{t}_n\|^2$
- となる.
- これを誤差関数と呼ぶ.
- 誤差関数の値が小さければ小さいほど，出力とラベルの差が小さい.
- つまり，　誤差関数の値が小さければ小さいほど性能が良い.

ニューラルネットワークの学習とバックプロパゲーション

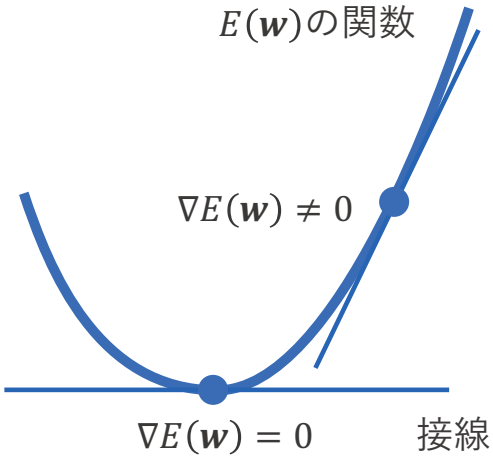
誤差関数の最小化

- 良いニューラルネットワークを作るには，誤差関数の値を小さくする重み $\boldsymbol{w}$を探せば良いということである．



誤差関数の最小化

- 誤差関数の値を小さくする重み $\boldsymbol{w}$をどう探すか.
- 高校数学では関数の最小値を求めるとき，極小値（微分が0）を求めた.
- 重み $\boldsymbol{w}$のとき誤差関数$E(\boldsymbol{w})$が最小となるには
- $\nabla E(\boldsymbol{w}) = 0$
- を満たすはずである.
- しかし，解析的に求めるのは無理である.

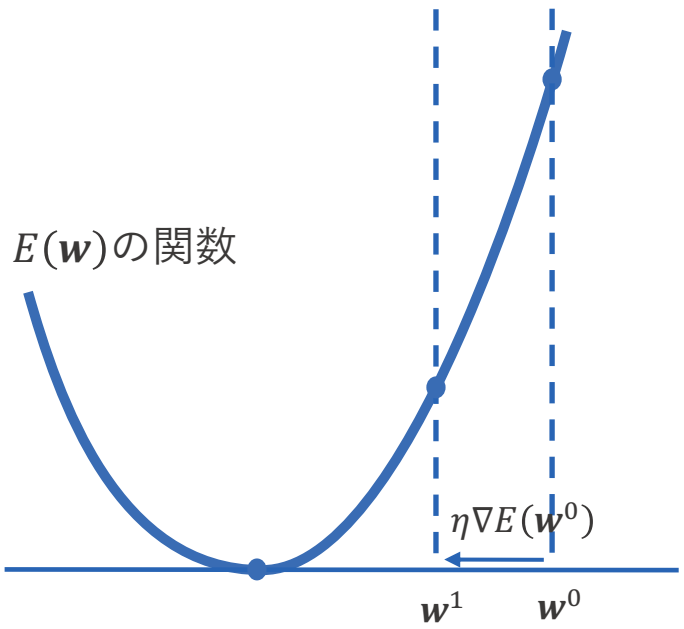


$\nabla E(\boldsymbol{w})$は $E(\boldsymbol{w})$の勾配だが，よくわからなければ微分と思っておく.

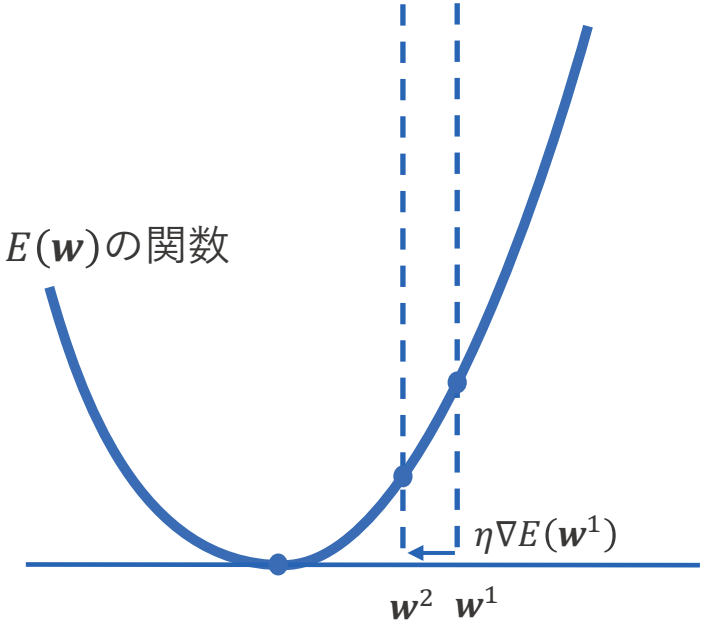
勾配法

- 勾配を用いて繰り返し計算で$\nabla E(\boldsymbol{w}) = 0$となる$\boldsymbol{w}$を求める方法に勾配法がある.
- τ 回繰り返し計算し求めた重みを $\boldsymbol{w}^{\tau}$ としたとき，次の τ +1回目の計算で求まる重み $\boldsymbol{w}^{\tau+1}$ は
- $\boldsymbol{w}^{\tau+1} = \boldsymbol{w}^{\tau} - \eta \nabla E(\boldsymbol{w}^{\tau})$
- と書ける.
- η は学習率と呼ばれるパラメタである.

図による勾配法の理解



$\boldsymbol{w}^0$ から$\eta\nabla E(\boldsymbol{w}^0)$移動する.



$\boldsymbol{w}^1$ から$\eta\nabla E(\boldsymbol{w}^1)$移動する. $\eta\nabla E(\boldsymbol{w}^1)$ は$\nabla E(\boldsymbol{w}^0)$より小さいため移動距離は小さい.

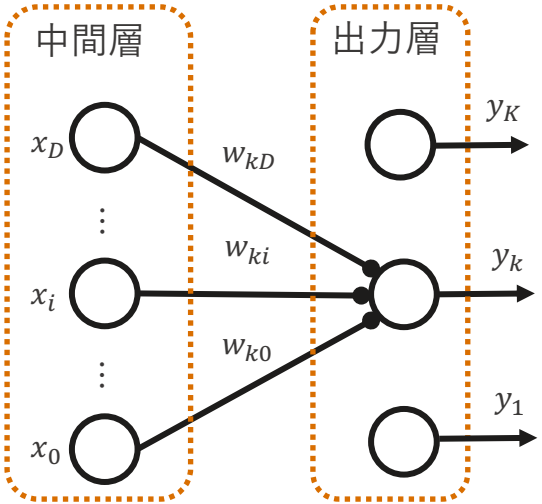
最小値に近づけば近づくほど勾配が小さく移動距離が小さくなる.
最小値まで移動すれば, 移動距離は0となる.

Online learning, batch learning, mini-batch learning

- Online learning
  - データ一つづつ用い学習する.
  - $\boldsymbol{w}^{\tau+1} = \boldsymbol{w}^{\tau} - \eta \nabla E_n(\boldsymbol{w}^{\tau})$
  - $E_n(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{y}(\boldsymbol{w}, \boldsymbol{x}_n) - \boldsymbol{t}_n\|^2$
- Batch learning
  - データすべて用い学習する.
  - $\boldsymbol{w}^{\tau+1} = \boldsymbol{w}^{\tau} - \eta \nabla E(\boldsymbol{w}^{\tau})$
  - $E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\|\boldsymbol{y}(\boldsymbol{w}, \boldsymbol{x}_n) - \boldsymbol{t}_n\|^2 = \sum_{n=1}^{N} \boldsymbol{E}_n$
- Mini-batch learning
  - データをいくつかに分け，それぞれについて学習する.
  - 現在の主流
  - $\boldsymbol{w}^{\tau+1} = \boldsymbol{w}^{\tau} - \eta \nabla E_b(\boldsymbol{w}^{\tau})$
  - $E_b(\boldsymbol{w}) = \frac{1}{2}\sum_{\boldsymbol{x}_n \in \boldsymbol{X}_{\mathrm{batch}}}\|\boldsymbol{y}(\boldsymbol{w}, \boldsymbol{x}_n) - \boldsymbol{t}_n\|^2$
  - $\boldsymbol{X}_{\mathrm{batch}}$にはbatch size個のデータ点 $\boldsymbol{x}_n$ が入っている.

勾配の計算

- 勾配法を使うには誤差関数の勾配を求める必要がある.
- まず図のような2層のニューラルネットワークを考える.
- ラベル $\boldsymbol{t}_n = (t_{n1}, \cdots, t_{ni}, \cdots, t_{nK})^{\mathrm{T}}$ が付いたデータ $\boldsymbol{x}_n = (x_{n0}, \cdots, x_{ni}, \cdots, x_{nD})^{\mathrm{T}}$をネットワークに入力したときの出力を$\boldsymbol{y}_n = (y_{n1}, \cdots, y_{ni}, \cdots, y_{nK})^{\mathrm{T}}$ とする.
- このときの誤差関数は
- $E_n(\boldsymbol{w}) = \frac{1}{2}\sum_k (y_{nk} - t_{nk})^2$



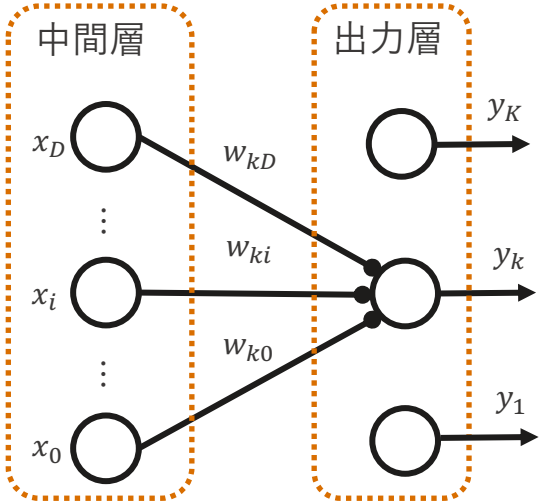
ラベルが持つ数値の個数K
1つの入力が持つ数値の個数D
出力が持つ数値の個数K

勾配の計算

- 出力y_{nk}は次のように書けるとする.
- $y_{nk} = \sum_i w_{ki} x_{ni}$
- $E_n(\boldsymbol{w})$を w_{ki}で微分すると

$$\frac{\partial E_n}{\partial w_{ki}} = \frac{\partial}{\partial w_{ki}}\left(\frac{1}{2}\sum_k (y_{nk} - t_{nk})^2\right)$$
$$= \frac{\partial}{\partial w_{ki}}\left(\frac{1}{2}\sum_k \left(\sum_i w_{ki} x_{ni} - t_{nk}\right)^2\right) = \left(\sum_i w_{ki} x_{ni} - t_{nk}\right) x_{ni}$$
$$= (y_{nk} - t_{nk}) x_{ni}$$

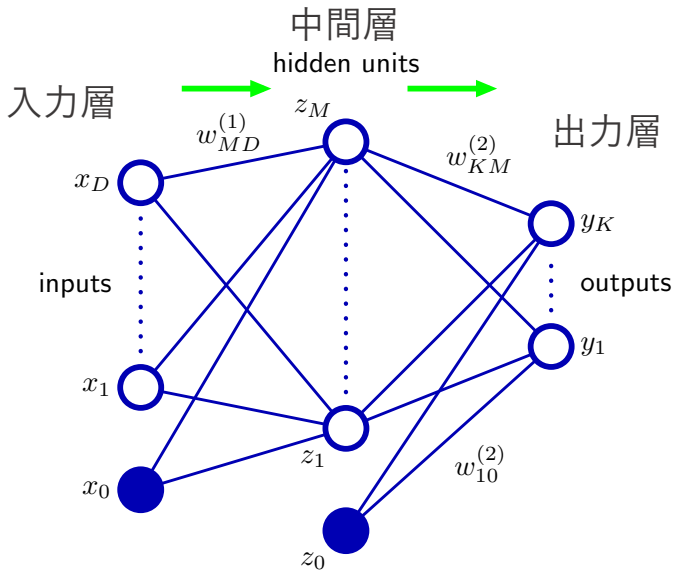
- となる.
- これを勾配法で使えば，最適な$\boldsymbol{w}$が求まる.



多層パーセプトロンの数式表現の振り返り

- 図のような3層（2層）のニューラルネットワークを考える.
- 出力層のユニットkの出力 y_kは次のように書ける.

- $a_j = \sum_{i=0}^{M} w_{ji}^{(1)} x_i = \boldsymbol{w}_j^{(1)\mathrm{T}} \cdot \boldsymbol{x}$
- $z_j = h(a_j)$
- $a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j = \boldsymbol{w}_k^{(2)\mathrm{T}} \cdot \boldsymbol{z}$
- $y_k = h(a_k)$



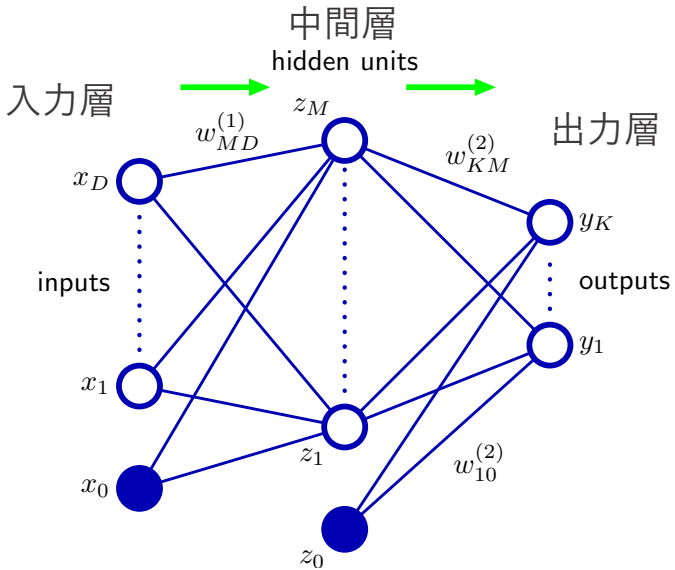
$\boldsymbol{x} = (x_0, x_1, \ldots, x_D)^{\mathrm{T}}$：入力
$\boldsymbol{w}_j^{(1)} = (w_{j0}, w_{j1}, \ldots, w_{jD})^{\mathrm{T}}$：中間層のユニットjと入力層のユニット間の重み
$\boldsymbol{z} = (z_0, z_1, \ldots, z_D)^{\mathrm{T}}$：中間層のユニットの出力
$\boldsymbol{w}_k^{(2)} = (w_{k0}, w_{k1}, \ldots, w_{kM})^{\mathrm{T}}$：出力層のユニットkと中間層のユニット間の重み

多層パーセプトロンの誤差関数の微分

- ラベル $\boldsymbol{t}_n = (t_{n1}, \cdots, t_{ni}, \cdots, t_{nK})$ が付いたデータ $\boldsymbol{x}_n = (x_{n0}, \cdots, x_{ni}, \cdots, x_{nD})$をネットワークに入力したときの出力を$\boldsymbol{y}_n = (y_{n1}, \cdots, y_{ni}, \cdots, y_{nK})$ とする.
- このときの誤差関数は

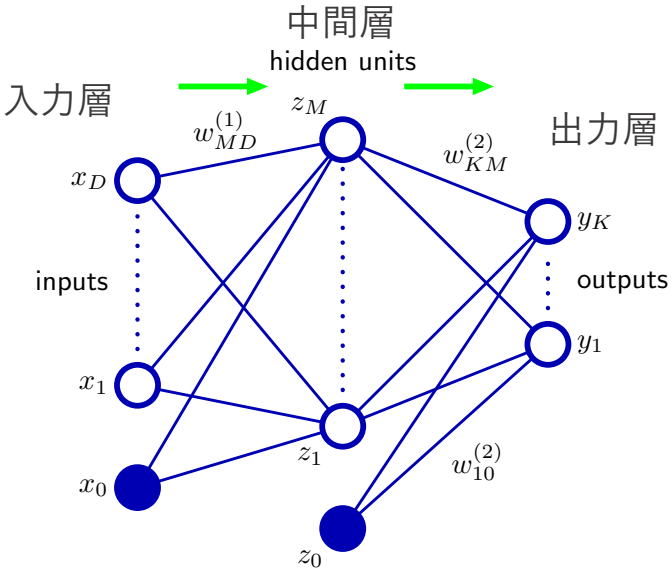
$$E_n(\boldsymbol{w}) = \frac{1}{2}\sum_k (y_{nk} - t_{nk})^2$$
$$= \frac{1}{2}\sum_k \left(h\left(a_k\left(\boldsymbol{w}_k^{(2)}\right)\right) - t_{nk}\right)^2$$

- と書ける.
- w_{kj} について誤差関数の微分をとると
- $\dfrac{\partial E_n}{\partial w_{ki}^{(2)}} = \dfrac{\partial E_n}{\partial a_k}\dfrac{\partial a_k}{\partial w_{ki}^{(2)}}$
- となる. この式はチェインルールを用い導いている.



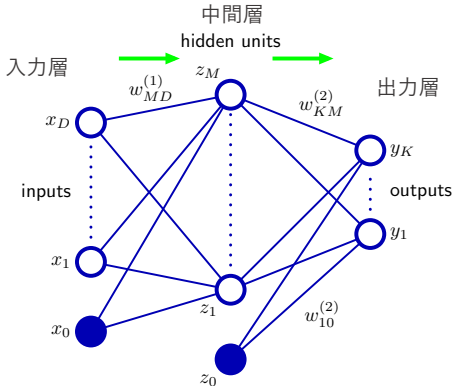
多層パーセプトロンの誤差関数の微分

- $\frac{\partial E_n}{\partial w_{ki}^{(2)}} = \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial w_{ki}^{(2)}}$
- $\frac{\partial a_k}{\partial w_{ki}^{(2)}} = \frac{\partial}{\partial w_{ki}^{(2)}}\sum_{j=0}^{M} w_{kj}^{(2)} z_j = z_j$
- $\delta_k \equiv \frac{\partial E_n}{\partial a_k}$, とすると
- $\frac{\partial E_n}{\partial w_{ki}^{(2)}} = \delta_k z_j$
- と書ける.
- ここで δ_k を誤差という.



多層パーセプトロンの誤差関数の微分

- $\delta_j \equiv \frac{\partial E_n}{\partial a_j}$とすると
- $\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \frac{\partial}{\partial a_j}\frac{1}{2}\sum_k (h(a_k) - t_{nk})^2$
- $a_k = \sum_{j=0}^{M} w_{kj}^{(2)} h(a_j)$だから a_kは a_jの関数である.
- $\delta_j = \frac{\partial}{\partial a_j}\frac{1}{2}\sum_k \left(h\left(a_k(a_j)\right) - t_{nk}\right)^2 = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}$
- これが中間層の誤差となる.
  $\delta_j = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial}{\partial a_j}\left(\sum_{j=0}^{M} w_{kj}^{(2)} h(a_j)\right) = h'(a_j)\sum_k \delta_k w_{kj}^{(2)}$



- この式をみると，出力層の誤差 δ_kが中間層の誤差に出てきている.
- これは誤差が中間層に伝播したと考えられるので，誤差の逆伝播という.

誤差逆伝播法（バックプロパゲーション法）

- 中間層jの誤差は$\delta_j = h'(a_j)\sum_k \delta_k w_{kj}^{(2)}$のように書けた.
- この式は，隣接する中間層lとmの間でも成り立つ.（mの方が出力に近いとする）
- $\delta_l = h'(a_l)\sum_m \delta_m w_{ml}$
- このように，誤差は出力層から入力層へ伝播していく.
- 出力層から入力層へ伝播することを逆伝播という.

- この誤差を使い，最適な重みを求める方法を誤差逆伝播法（バックプロパゲーション法）という.

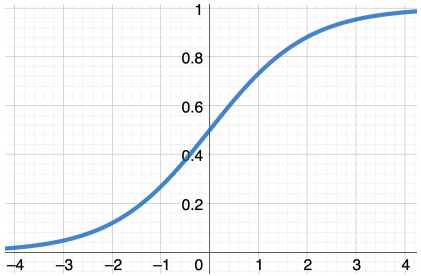
様々な活性化関数

- 誤差逆伝播法を使い多層パーセプトロンを学習可能にするためには，誤差が微分できなければならないし，微分が値を持たなければならない．
- つまり，パーセプトロンで用いたステップ関数の微分は，0以外のとき 0 となり，誤差逆伝播法では使えない．
  - 微分が 0 だと，誤差 δ_l が 0 となる．
- 多層パーセプトロンでは次に紹介する活性化関数がよく用いられる．

様々な活性化関数

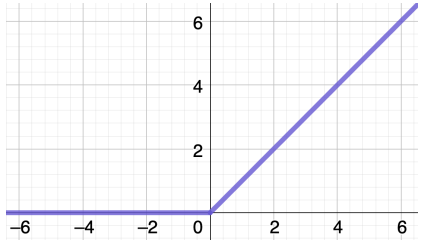
昔の主流

Sigmoid関数



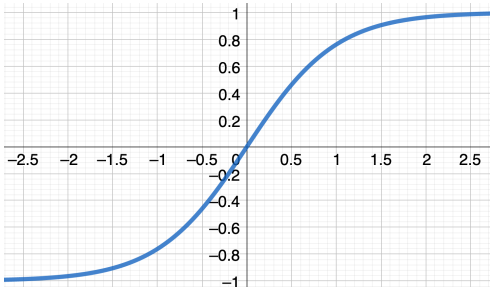
$$h(x) = \frac{1}{1 + \exp(-x)}$$

今の主流　Rectified linear関数 (Rectified Linear Unit: ReLU)



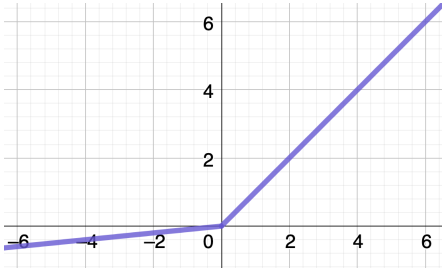
$$h(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

tanh関数



$$h(x) = \tanh x$$

Leaky rectified linear関数

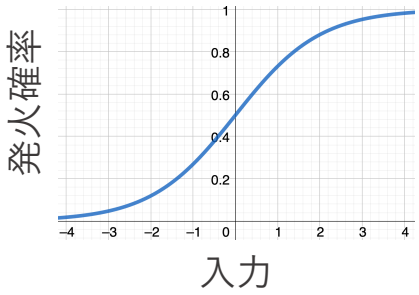


$$h(x) = \begin{cases} x & \text{if } x \geq 0 \\ ax & \text{otherwise} \end{cases}$$

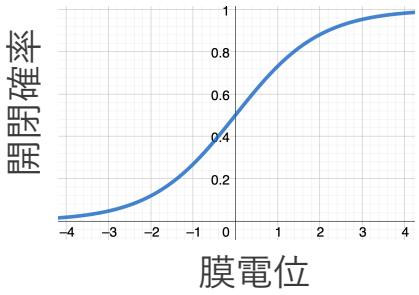
なぜSigmoid関数なのか

- Sigmoid関数は式展開しやすい．
- 神経細胞の応答と親和性が高い．
  1. 神経細胞の発火は確率的である．入力を大きくすればするほど発火確率が上がる．
  2. イオンチャネルの開閉確率は膜電位の大きさに依存する．
  3. 神経細胞の発火率は入力を大きくすればするほど大きくなる．しかし，発火率は一定以上には上がらない．
  4. 神経細胞集団内の発火する神経細胞の数は入力を増やせば増やすほど大きくなる．しかし，神経細胞の数の上限は決まっている．

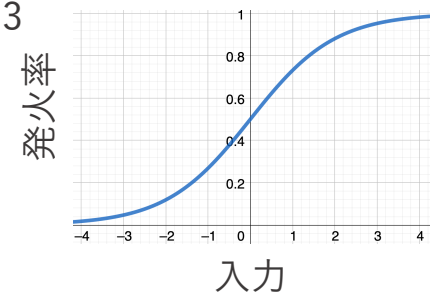
1



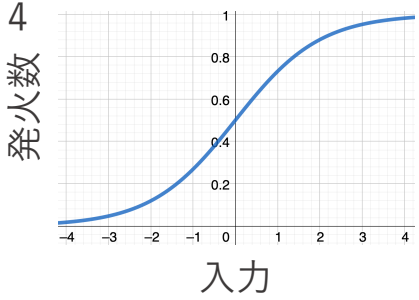
2



3

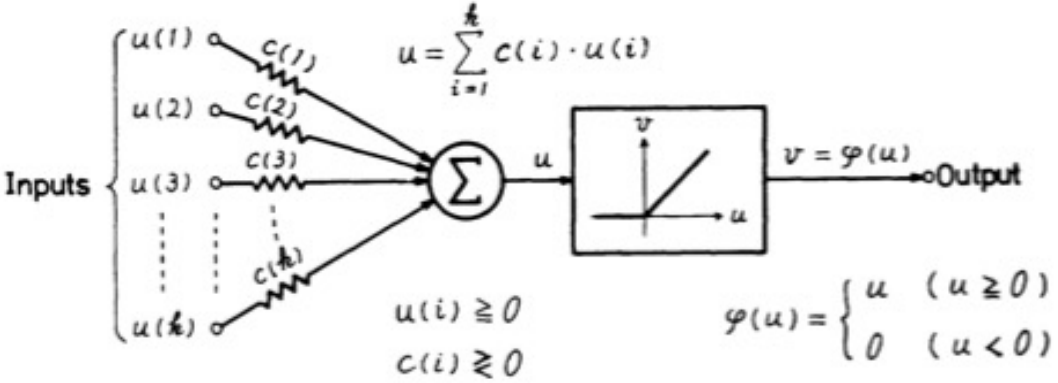


4



Rectified linearはFukushimaが初めて使った？

- 現在主流のReLUはFukushimaが1969年の論文ですでに使っている.



(Fukushima 1969より)

なぜこのような神経生理学とは異なる発想ができたのだろうか

- ステップ関数やSigmoid関数の方が神経細胞の応答と対応させられる．
  - 神経細胞の発火率は入力が強ければ強いほど高くなるが，発火率の高さには限界がありいずれ飽和する．
- しかし，Fukushimaはなぜ神経細胞の応答と対応させにくいReLUを発想できたのか？

- Fukushimaは「生理学からはヒントをもらうが，開発時には実際の**脳はいったん忘れて研究を進めることが重要だ．**ただ，それだけではいずれ限界が来る．その時はもう一度，生理学に戻って考える．これを繰り返すことで，前進していけるだろう」(NikkeiBPnet, 2015)と語っている．
  - これはKanadeの「素人発想，玄人実行」にも通じる．

多層パーセプトロンの限界

- 深層化に限界があった.
  - ネットワークを深くしても意味がない.
  - 勾配消失問題
- よく分からない.
  - ニューラルネットワークはブラックボックスになってしまう.

- この問題とサポートベクタマシン（SVM）の登場により，ニューラルネットワークの冬の時代が訪れる.
  - 識別機はニューラルネットワークからSVMへ.

深い意味がない

- 通常，層間の接続はランダムな値で初期化される．
- ランダムなネットワークは，入力を何らかのパターンに変換する．
- 深いネットワークでは，入力層に近いランダムなネットワークが入力を変換し，そのランダムネットワークの出力パターンを出力層に近いネットワークが学習するという現象が起こる．
- ランダムネットワークの出力パターンを用いればデータを識別できてしまうため，入力に近い層は学習する必要がない．
- つまり，深い多層パーセプトロンではランダムネットワークと識別ネットワークの2重構造になる．
  - この構造は，Rosenblattのパーセプトロンと同じである．
- 以上のような現象が起こるため，無闇にネットワークを深くしてもランダムネットワークが大きくなるだけで意味がないかもしれない．

勾配が消える（勾配消失問題）

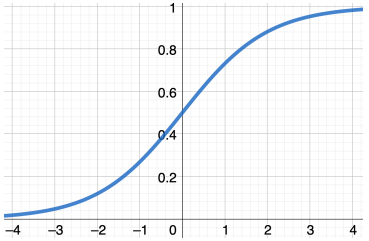
- Sigmoid関数を使った場合，入力層に近い中間層の誤差が小さくなる．
- その結果，勾配も小さくなり，入力層に近い中間層の学習が進まない．
- 詳細はつぎのスライド

Sigmoid関数とReLUの比較

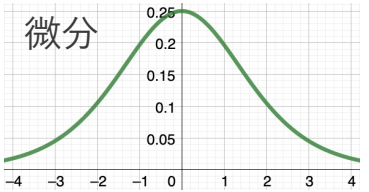
- 誤差逆伝播法に出てくる誤差$\delta_l = h'(a_l)\sum_m \delta_m w_{ml}$を見ると活性化関数の微分が出てくる．

- Sigmoid関数の微分はxが0から離れると小さくなり0に漸近する．つまり，Sigmoid関数を使うと a_l が0から離れると誤差が0に近くなる．その場合誤差も0に近づき学習が進まない．
- Sigmoid関数の微分の最大値は0.25である．誤差の式をみると誤差 δ_m と微分 $h'(a_l)$ の積になっている．誤差δ_mも誤差と微分の積になっている．この積算は再帰的に行われるため，層が増えれば増えるほど誤差がかけられることになる．つまり，ネットワークの層が多い場合，入力に近いそうの誤差が 0 に近づく．

Sigmoid関数



$$h(x) = \frac{1}{1+\exp(-x)}$$

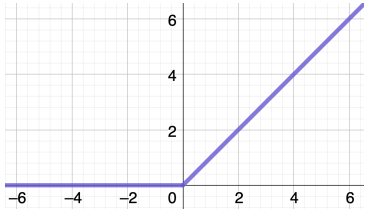


Sigmoid関数とReLUの比較

- 誤差逆伝播法に出てくる誤差$\delta_l =$ $h'(a_l)\sum_m \delta_m w_{ml}$を見ると活性化関数の微分が出てくる.

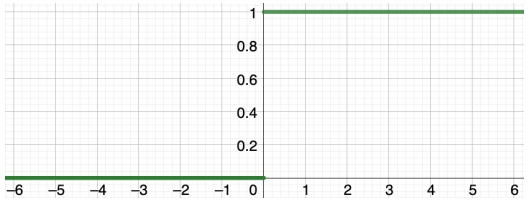
- ReLUは $x \geq 0$のとき微分が1である.
  - a_lが極めて大きくても誤差が 0 にならない.
  - 微分の掛け算がいくら繰り返されても $a_l > 0$ならば1のままである.
  - ネットワークが多層になっても入力に近い層の誤差が消えることなく入力に近い層も学習が進む.

Rectified linear関数
(Rectified Linear Unit: ReLU)



$$h(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

微分



Backpropergation法とはなにか

- 意地悪くBackpropagationを見れば
  - 誤差を単にchain ruleで微分しているだけと言える.
  - 先行研究の再発見なだけではないのか.

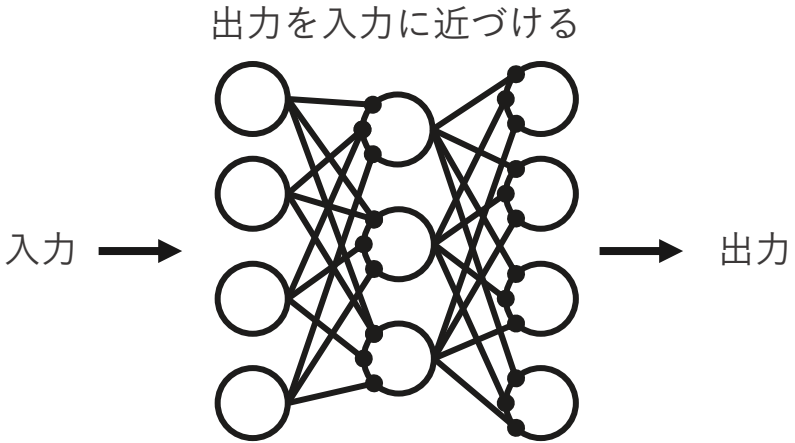
- 神経科学の立場から見れば, Backpropagationを使ったネットワークは脳モデルからの逸脱しているのではないか.
  - 脳でBackpropagationが起こっているかどうか分かっていない.
    - Back-prop netは脳が行っているかどうかで言えば，現実的なものではない(Crick, 1989).

- ニューラルネットワークは神経科学の縛りはなくても良いのでは.
  - 使用する数式やアーキテクチャは神経科学を気にすることなく作ることができる.
    - 活性化関数はステップ関数やシグモイド関数でなくても良い.
    - Rectified linear関数やleaky rectified linear関数は神経細胞の活性を表現していない.
- もはやニューラルネットワークは人工ニューラルネットワークという脳のモデルではなく機械学習の一分野になった.

Autoencoder

Autoencoder

- Autoencoderは砂時計型をしたニューラルネットワークである.
- Autoencoderは入力と出力を同じにするように学習する.
- 中間層は入力層よりもユニットが少ないため，Autoencoderは入力よりも小さなユニットで入力を再現しなければならない.
- 入力を再現するために，中間層は再現するために必要な重要な特徴のみ保存するように動作する.
- 入力の次元圧縮に使える.

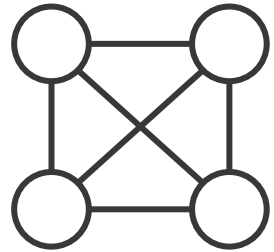


Hopfield network

Hopfield network (Hopfield, 1982, 1984)

- 相互結合ネットワーク
- 対象結合である.
  - ユニットiからjへの重みとjからiへの重みは等しい.
  - 完全に脳と関係ないニューラルネットワークといえる.
- 次のルールでユニットの応答が決まる.

$$u_i \leftarrow \begin{cases} 1 & \text{if } \sum_j w_{ji}u_j + s_i - \theta_i > 0 \\ 0 & \text{if } \sum_j w_{ji}u_j + s_i - \theta_i < 0 \\ u_i & \text{if } \sum_j w_{ji}u_j + s_i - \theta_i = 0 \end{cases}$$

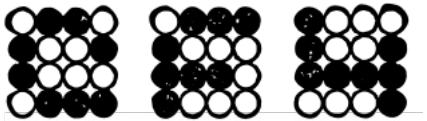


相互結合ネットワーク

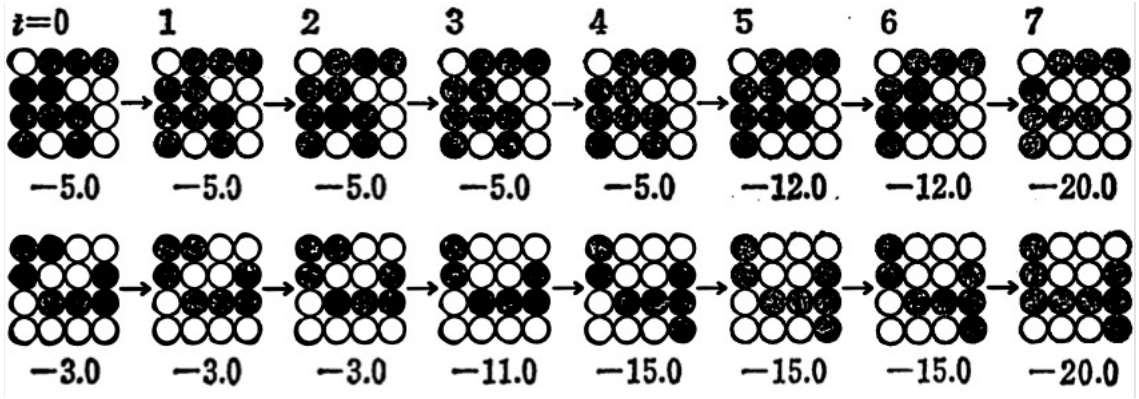
- ユニットの出力をu_i，iからjへの重みをw_{ji}，入力をs_i，閾値をθ_iとする.

Hopfield network (Hopfield, 1982, 1984)

- 先のルールでユニットの出力を更新するとある応答パターンが出てくる.
- パターンは覚えさせる事ができる.
- ユニットの出力を更新していくと勝手にパターンが出てくる.
  - 自己想起



覚えさせたパターン



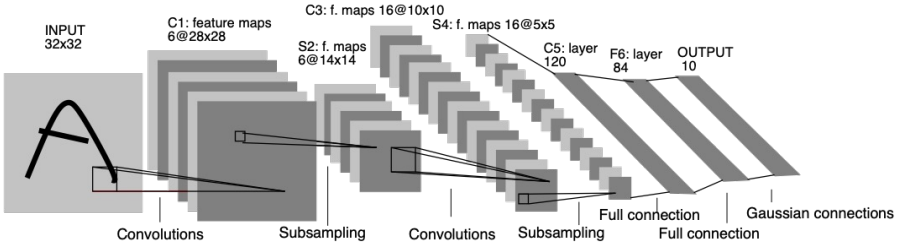
ユニットの出力を更新していったときのネットワークの応答

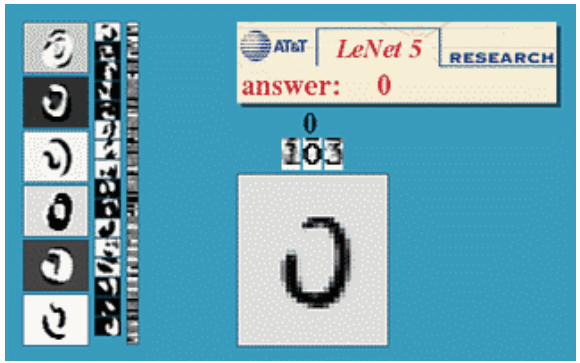
(麻生, ニューラルネットワーク情報処理)

LeNet

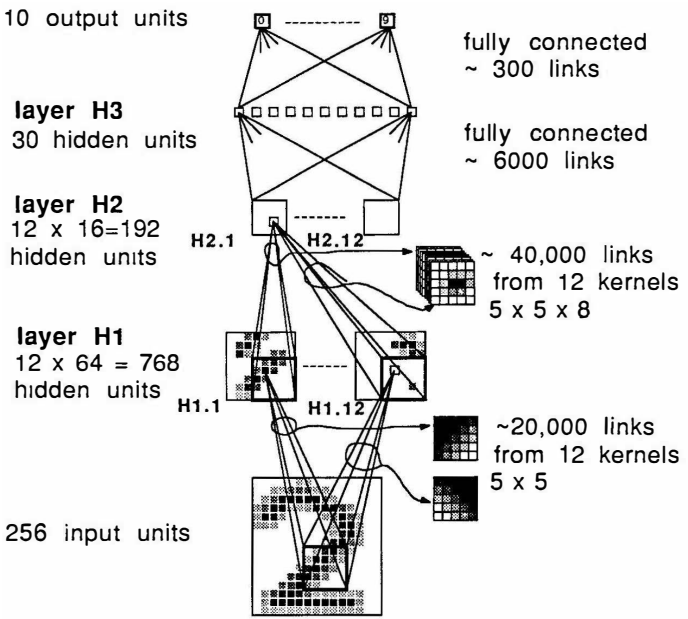
LeNet

- LeCunらにより提案された畳み込みニューラルネットワーク
- 1989年US Post Serviceから提供された手書き数字の識別に成功





LeNet-5 (LeCun et al., 1998)



(LeCun et al., 1989)

ニューラルネットワークから人工ニューラルネットワークへ

- バックプロパゲーションは脳でやっていない？
- Hopfield networkは脳構造から逸脱している.
  - 対象の接続はありえない.
- ニューラルネットワークは脳の機能・構造を反映しなくてよいのではないか.

- 生物からの離脱を果たすことで，脳のニューラルネットと人工ニューラルネットワークが分かれる.
- 人工ニューラルネットワークは神経科学や認知科学の分野から機械学習の分野に移行する.
- しかし，人工ニューラルネットワークは脳とは異なるが，脳からアイディアをもらう.

人工ニューラルネットワークの技術は充実する

- 1980年代までに人工ニューラルネットワークの技術はかなりそろってくる.
  - Percetron (Rosenbratt, 1958)
  - Backprobagation (Rumenhert et al, 1985)
  - Autoencoder (Rumelhart et al. 1986; Baldi and Hornik, 1988)
  - Convolutional neural network (Fukushima, 1980; LeCun, 1989)
  - ReLU (Fukushima, 1968)
  - SOM (von der Marsberg, 1976; Kohonen, 1982)
  - Hopfield network (Hopfield, 1982)
  - Rcurrent neural network (Rumelhart et al, 1985; Jordan, 1986)

- 1990年代以降の技術
  - Long term short term memory: LSTM (Hochreiter and Schmidhuber, 1997)
  - Leaky ReLU
  - Dropout
  - Redisual neural network (He et al., 2016)
  - Random erasing
  - Generative adversarial network: GAN (Goodfellow et al., 2014)
  - 深層強化学習