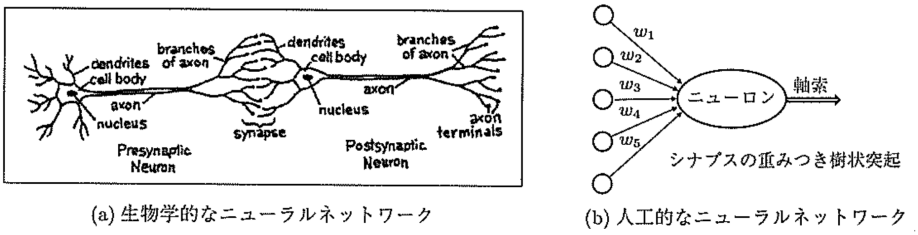
第 1 章

ニューラルネットワークとは

1.1　はじめに

機械学習とは明治的にプログラムしなくても学習する能力をコンピュータに与える分野とされるが, そのうちニューラルネットワークは, 生物の学習メカニズムを図 1.1 のように模倣した機械学習アルゴリズムである. より具体的に言えば, ある入力に対して重みをつけて出力情報を送り, それによる予測値 $\hat{y}$ と訓練データ y を比較することで, 再度重みを調整するという流れでそのネットワークを構成する.



(a) 生物学的なニューラルネットワーク　　(b) 人工的なニューラルネットワーク

図 1.1　ニューロン間のシナプス結合

従来の機械学習アルゴリズム (最小二乗回帰, ロジスティック回帰, etc.) と比べた場合, ニューラルネットワークの利点は

1. 計算グラフのアーキテクチャ設計の選び方によって, データの応用領域の意味論的な知見を表現できるような, 高レベルの抽象化が可能.
2. どの程度の訓練データや計算能力が利用できるかに応じて, アーキテクチャにニューロンを追加・消去し, モデルの複雑さを簡単に調整可能.

ということが上がる. なお, 小さなデータセットの場合については, 従来の機械学習アルゴリズムの方が優位になることもあるので注意が必要 (図 1.2).

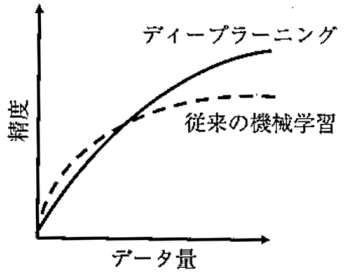


図 1.2

1.2　ニューラルネットワークの基本的なアーキテクチャ

最も単純なニューラルネットワークはパーセプトロンと呼ばれ, 図 1.3 のように, ネットワークの層が一層で, 入力の集合を直接出力に写像するようなものを指す. また, ニューロンが層状に配置され, 入力層と出力層が複数の隠れ層によって隔てられているようなものは, 多層ニューラルネットワーク及び順伝播型ネットワーク (feed-forward network) と呼ばれている.

1.2.1　単一の計算層: パーセプトロン

いま, 個々の訓練データが d 個の特徴量 $\boldsymbol{X} = [x_1, \cdots, x_d]$ と観測値 2 値のクラス変数 $y \in \{-1, +1\}$ の $(\boldsymbol{X}, y)$ で与えられる状況を考える. そして目標は訓練データ $(\boldsymbol{X}, y)$ を用いてニューラルネットワークを構成し, 未知の入力 x に対して適切な予測値 $\hat{y}$ を出力することである. ニューラルネットワークを構成するにあたって, 重みを $\boldsymbol{W} = [w_1, \cdots, w_d]$ のようにすれば, 各入力に対して $\boldsymbol{W} \cdot \boldsymbol{X} = \sum_{i=1}^{d} w_i x_i$ として重みづけがなされる. そして出力においては 2 値分類を行うため, 符号関数 sign を用いれば, 予測値 $\hat{y}$ は以下のように計算される.

$$\hat{y} = \mathrm{sign}\{\boldsymbol{W} \cdot \boldsymbol{X}\} = \mathrm{sign}\{\sum_{j=1}^{d} w_j x_j\} \tag{1.1}$$

そして観測値 y と予測値 $\hat{y}$ の間の誤差 $E(\boldsymbol{X}) = y - \hat{y}$ を最小にするように, つまり誤差勾配が負となるように重み更新を行っていき, ニューラルネットワークを構成する.

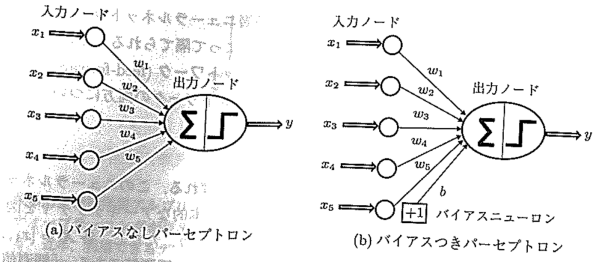


図 1.3

今回はクラスラベルに分類することが目標であったため, 入力に重みづけがされた値 $\boldsymbol{W}\cdot\boldsymbol{X}$ を符号関数で評価したが, これらは活性化関数と呼ばれ, 活性化関数の選び方によってニューラルネットワークの役割が決定されていく.

また, 多くの状況において予測にはバイアスと呼ばれる特徴量によらない不変な部分があり, 追加のバイアス b を組み込むことでこれを捉えることができる.

$$\hat{y} = \mathrm{sign}\{\boldsymbol{W}\cdot\boldsymbol{X} + b\} = \mathrm{sign}\{\sum_{j=1}^{d} w_j x_j + b\} \tag{1.2}$$

以上のことから, 特徴量とラベルのペアを含むデータセット $\mathcal{D}$ 内の全ての訓練データについて, パーセプトロンアルゴリズムの重み更新の目標を最小二乗形式

$$\mathrm{Minimize}_{\boldsymbol{W}} L, \quad L = \sum_{(\boldsymbol{X},y)\in\mathcal{D}} (y-\hat{y})^2 = \sum_{(\boldsymbol{X},y)\in\mathcal{D}} (y - \mathrm{sign}\{\boldsymbol{W}\cdot\boldsymbol{X}\})^2$$

で記述できる. これは損失関数 (loss function) ともよばれ, ほとんどのニューラルネットワークの学習アルゴリズムは損失関数を使って定式化されている. しかしながら今回のように活性化関数が符号関数の場合, 微分不可能でありさらに定義いきの大部分で一定値を取るために, 微分可能な点での勾配は 0 となってしまう. そのため, パーセプトロンアルゴリズムでは暗黙的に, 各データごとの目的関数の勾配を滑らかに近似した

$$\nabla L_{\mathrm{smooth}} = \sum_{(\boldsymbol{X},y)\in\mathcal{D}} (y-\hat{y})\boldsymbol{X} \tag{1.3}$$

を用いる.

では, 実際のニューラルネットワークの訓練アルゴリズムではどのように重みを更新しているかを考えると, それぞれの入力データ $\boldsymbol{X}$ をネットワークに 1 つずつ (または小さなバッチで) 与えて予測値 $\hat{y}$ を計算し, 誤差 $E(\boldsymbol{X}) = (y-\hat{y})$ が最小となるようにニューラルネットワークの学習率を α として

$$\boldsymbol{W} \Leftarrow \boldsymbol{W} + \alpha(y-\hat{y})\boldsymbol{X} \tag{1.4}$$

$$\boldsymbol{W} \Leftarrow \boldsymbol{W} + \alpha E(\boldsymbol{X})\boldsymbol{X} \tag{1.5}$$

で更新がなされている (確率的勾配降下法). なお, ランダムに選択された訓練データの部分集合 S からバッチを作って重みを更新するミニバッチ確率的勾配法は以下のように計算される.

$$\boldsymbol{W} \Leftarrow \boldsymbol{W} + \alpha \sum_{\boldsymbol{X}\in S} E(\boldsymbol{X})\boldsymbol{X} \tag{1.6}$$

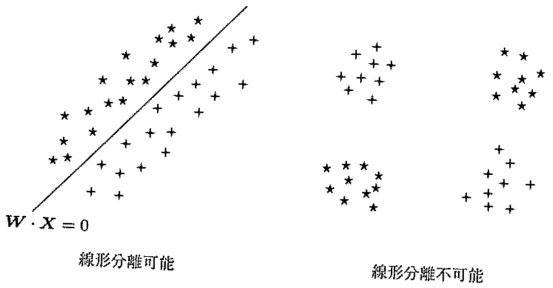


図 1.4

1.2.1.1　パーセプトロンはどのような目的関数を最適化しているか

2 値分類における誤分類数は, 訓練データ $(\boldsymbol{X}_i, y_i)$ ごとに 0/1 損失関数の形で以下のように書ける.

$$L_i^{(0,1)} = \frac{1}{4}\left(y_i - \mathrm{sign}\{\boldsymbol{W}\cdot\boldsymbol{X}_i\}\right)^2 = \frac{1}{2}\left(1 - y_i\cdot\mathrm{sign}\{\boldsymbol{W}\cdot\boldsymbol{X}_i\}\right) \tag{1.7}$$

しかしこれは微分不可能であり, ニューラルネットワークは勾配に基づく最適化によって定義されるため, パーセプトロンの更新に対応する滑らかな目的関数を定義する必要がある. そこでパーセプトロンの更新には暗黙的にパーセプトロン基準 (目的関数を滑らかなものに置き換えたもの) を最適化しているということが示されている. したがって, 式 (1.7) における符号関数を除去し, 正しい予測に対しては全ての損失が 0 となるようにすることで, 目的関数は以下のように定義することができる.

$$L_i = \max\{-y_i(\boldsymbol{W}\cdot\boldsymbol{X}_i), 0\} \tag{1.8}$$

では, 実際に計算してみると

$$\boldsymbol{W}\cdot\boldsymbol{X}_i = w_1x_1^i + \cdots + w_dx_d^i$$

であり

$$\begin{aligned}\frac{\partial L_i}{\partial w_1} &= -y_ix_1^i,\\ &\vdots\\ \frac{\partial L_i}{\partial w_d} &= -y_ix_d^i\\ \Rightarrow \nabla_W L_i &= -y_i\boldsymbol{X}\end{aligned}$$

となるから, $\boldsymbol{W} - \alpha\nabla_W L_i = \boldsymbol{W} - y_i\boldsymbol{X}$ となり, 式 (1.4) のパーセプトロン更新式と一致することが確かに確認できる.

1.2.1.2　サポートベクトルマシンとの関係

パーセプトロンは二章で扱う, サポートベクトルマシンで使われているヒンジ損失を平行移動したものとも捉えることができる. ヒンジ損失は

$$L_i^{SVM} = \max\{1 - y_i(\boldsymbol{W} \cdot \boldsymbol{X}_i), 0\} \tag{1.9}$$

で定義され, 最大値関数の中に定数を持って平行移動されている (図 1.6). この類似性をより理解するために式 (1.6) を

$$\boldsymbol{W} \Leftarrow \boldsymbol{W} + \alpha \sum_{(\boldsymbol{X},y)\in S^+} y\boldsymbol{X} \tag{1.10}$$

と書く (S^+ は $y(\boldsymbol{W} \cdot \boldsymbol{X}) < 0$ を満たすような全ての誤分類された点 $\boldsymbol{X} \in S$ の集合). ここで, S^+ における誤分類された点では, $\mathrm{sign}\{\boldsymbol{W} \cdot \boldsymbol{X}\} = 0$ となることはなく, 誤差値は $E(\boldsymbol{X}) = (y - \sin\{\boldsymbol{W} \cdot \boldsymbol{X}\}) \in \{-2, +2\}$ となることがわかる. 従って, 誤分類された点については必ず $E(\boldsymbol{X}) = 2y$ となり, 学習率に係数 2 を含めれば y に置き換えられる関係にあることがわかる. つまり, サポートベクトルマシン (SVM) では決定境界付近のほぼ正しい点も更新の対象にするのに対し, パーセプトロンで考えれば, 誤分類された点に対してのみ更新が行われるとみることができる.

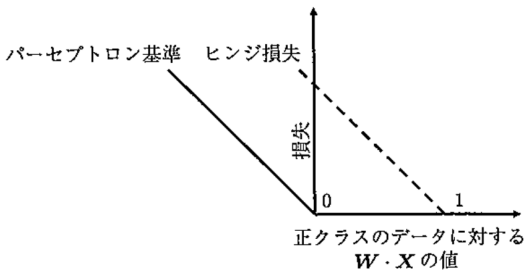


図 1.6　パーセプトロン基準とヒンジ損失

1.2.1.3　活性化関数と損失関数の選択

パーセプトロンの場合, 2 値のクラスラベルを予測したいという動機から, 活性化関数を符号関数に選んだが, 活性化関数を他のものにすることで, 様々な目的変数を予測できる. 以下に活性化関数

の例を示す.

$$
\begin{cases}
\Phi(v) = \mathrm{sign}(v) & (\text{符号関数}) \quad \leftarrow 2 \text{ 値出力への写像} \\
\Phi(v) = \dfrac{1}{1+e^{-v}} & (\text{シグモイド関数}) \quad \leftarrow \text{確率的解釈} \\
\Phi(v) = \dfrac{e^{2v}-1}{e^{2v}+1} & (\tanh \text{ 関数}) \quad \leftarrow \text{出力値が正と負の両方を取る確率的解釈} \\
\Phi(v) = \max\{v, 0\} & (\text{ReLu 関数}) \quad \leftarrow \text{多層ニューラルネットワーク} \\
\Phi(v) = \max\{\min[v, 1], -1\} & (\text{ハード } \tanh) \quad \leftarrow \text{多層ニューラルネットワーク} \\
\Phi(v)_i = \dfrac{\exp(v_i)}{\sum_{j=1}^{k} \exp(v_j)} & (\text{ソフトマックス関数}) \quad \leftarrow \text{出力を確率に変換する}
\end{cases}
$$

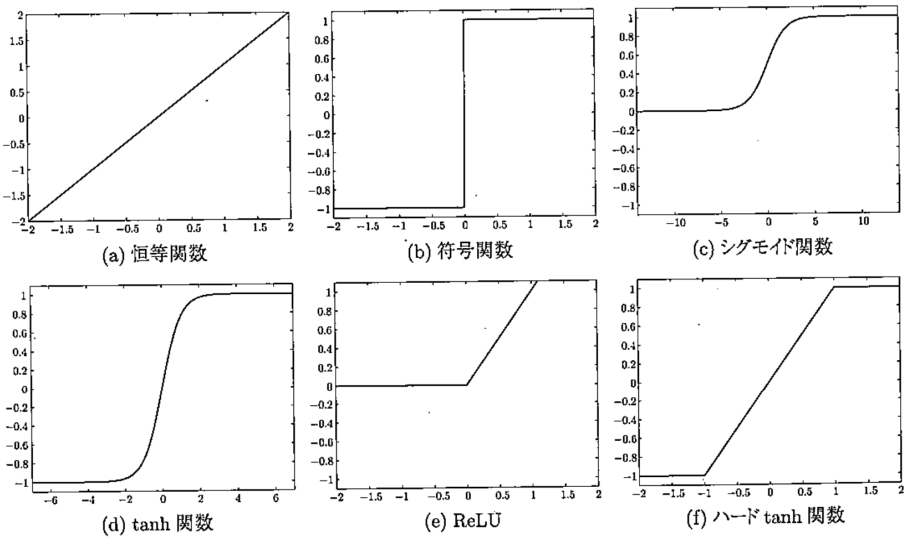


図 1.8

1.2.1.5 損失関数の選択

- 目的変数が 2 値の場合 (ロジスティック回帰)

$$L = -\log(1 + \exp(-y \cdot \hat{y})) \tag{1.14}$$

- 目的変数がカテゴリの場合

$$L = -\log(\hat{y}_r) \tag{1.15}$$

1.2.1.6 活性化関数の有用な導関数

ニューラルネットワークでは, 活性化関数を用いた勾配降下法が重要な役割を演じるため, それぞれの導関数を示しておく.

1. 恒等活性化関数と符号活性化関数
   恒等活性化関数は全ての場所で 1 であり, 符号活性化関数は $v = 0$ で不連続で微分不可能でそれ以外で 0 である. したがって, 導関数は常に 0. (損失関数としてつかわれることはほとんどない.)
2. シグモイド活性化関数
   o を v を引数とするシグモイド関数の出力とする.

$$o = \frac{1}{1 + \exp(-v)} \tag{1.16}$$

   この導関数は

$$\frac{\partial o}{\partial v} = \frac{\exp(-v)}{(1 + \exp(-v))^2} \tag{1.17}$$

$$= o(1 - o) \tag{1.18}$$

   となり, 出力を用いて記載できることがわかる.
3. tanh 活性化関数

$$o = \frac{\exp(2v) - 1}{\exp(2v) + 1} \tag{1.19}$$

   この導関数は

$$\frac{\partial o}{\partial v} = \frac{4\exp(2v)}{(\exp(2v) + 1)^2} \tag{1.20}$$

$$= 1 - o^2 \tag{1.21}$$

   となることがわかる.
4. ReLU 活性化関数とハード tanh 活性化関数
   ReLU の偏導関数は引数の値が非負であれば 1, そうでなければ 0. ハード tanh の導関数は引数の値が $[-1, +1]$ の場合は 1 をとり, それ例外では 0 を取る

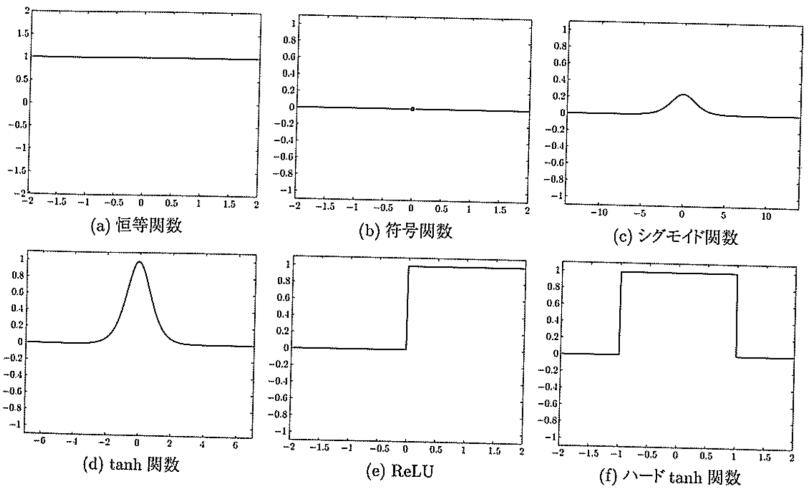


図 1.9　様々な活性化関数の導関数

1.2.2　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は, 図 1.10 のように, 入力層と出力層の間に複数の中間層をもち, パーセプトロンでは出力層のみで行なっていた計算を中間層でも行う. いま, ニューラルネットワークの k 個の層がそれぞれ $p_1, \cdots, p_k$ 個のユニットを含み, 入力が $x_1, \cdots, x_d$ の場合, 入力層と 1 番目の隠れ層の間の重みは $p_r \times p_{r+1}$ 行列 W_1 で表され, r 番目の隠れ層と $r+1$ 番目の隠れ層の間の重みは $p_r \times p_{r+1}$ 行列 W_r で表される. そして最後の出力層が o 個のノードを含む場合, 最後の行列 W_{k+1} のサイズは $p_k \times o$ となる. これを踏まえれば, d 次元の入力ベクトルは以下のように再帰的に出力に変換される.

$$\begin{aligned} \boldsymbol{h}_1 &= \Phi(\boldsymbol{W}_1^T \boldsymbol{x}) \\ \boldsymbol{h}_{p+1} &= \Phi(\boldsymbol{W}_{p+1}^T \boldsymbol{h}_p) \\ o &= \Phi(W_{k+1}^T \boldsymbol{h}_k) \end{aligned}$$

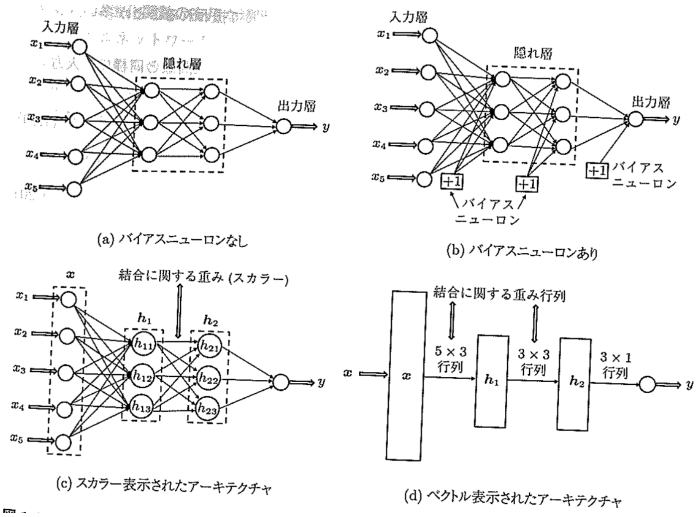


図 1.10

なお, ここでは順伝播型の古典的なアーキテクチャを示したが, 次元削減や畳み込みニューラルネットワークによって応用領域に沿った中間層アーキテクチャの設計がなされている.

1.3 誤差逆伝播法によるニューラルネットワークの訓練

1.3.1 誤差逆伝播法

重みの更新をするにあたって, 単一層ニューラルネットワークでは, 損失関数が直接重みの関数として計算できたことから, 学習の過程は比較的簡単であったが, 多層ネットワークの場合, 損失が前の層の重みの複雑な合成関数になるため, 更新が難しくなる. そこで用いられるのが誤差逆伝播法である.

誤差逆伝播法は

$$\frac{\partial L}{\partial w_{(h_{r-1},h_r)}} = \frac{\partial L}{\partial o} \cdot \left[\frac{\partial o}{\partial h_k} \Pi_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right] \frac{\partial h_r}{\partial w_{(h_{r-1},h_r)}} \tag{1.22}$$

で記述できる. これはネットワーク上に単一の経路しかないと仮定しているが, 複数の経路の勾配の計算には, 多変数連鎖率を用いて

$$\frac{\partial L}{\partial w_{(h_{r-1},h_r)}} = \underbrace{\frac{\partial L}{\partial o} \cdot \left[\sum_{[h_r,h_{r+1},\cdots,h_k,o]\in\mathcal{P}} \frac{\partial o}{\partial h_k} \Pi_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right]}_{\Delta(h_r,o)=\frac{\partial L}{\partial h_r}} \frac{\partial h_r}{\partial w_{(h_{r-1},h_r)}} \tag{1.23}$$

として一般化できる ($\mathcal{P}$ は h_r から o までの経路の集合).

では (1.23) をどのようにして計算するかを考えると, ニューラルネットワークの計算グラフには

巡回路がないことから, 全ての考えうる経路に対して

$$\Delta\left(h_r, o\right) = \frac{\partial L}{\partial h_r} = \sum_{h:h_r \Rightarrow h} \frac{\partial L}{\partial h} \frac{\partial h}{\partial h_r} = \sum_{h:h_r \Rightarrow h} \frac{\partial h}{\partial h_r} \Delta(h, o) \tag{1.25}$$

と計算できる. このとき, 活性化関数を適用する直前に隠れユニット h で計算された値を a_h とすれば

$$\frac{\partial h}{\partial h_r} = \frac{\partial h}{\partial a_h} \cdot \frac{\partial a_h}{\partial h_r} = \frac{\partial \Phi\left(a_h\right)}{\partial a_h} \cdot w_{(h_r,h)} = \Phi'\left(a_h\right) \cdot w_{(h_r,h)} \tag{1.26}$$

となり, 更新式は

$$\Delta(h_r, o) = \sum_{h:h_r \Rightarrow h} \Phi'(a_h) \cdot w_{(h_r,h)} \cdot \nabla(h, o) \tag{1.27}$$

となる. また, 式 (1.23) の末尾は

$$\frac{\partial h_r}{\partial w_{(h_{r-1},h_r)}} = h_{r-1} \cdot \Phi'(a_{h_r}) \tag{1.28}$$

と計算できる.