深層学習入門

Morita

2019 年 6 月 28 日

目次

1　はじめに

パーセプトロンから始め，ニューラルネットワークとその学習方法などの基本的な説明を扱う。正則化や重みの初期値の設定についても扱う。

2　Perceptron

パーセプトロンとは，入力として複数の数値を受け取り，1 つの数値を出力するものである。入力された数値には，それぞれ対応する重みがかけられ，その総和が取られる。それにバイアスと呼ばれる数値を足し，それが 0 よりも大きければ 1 を，0 以下であれば 0 を出力する。パーセプトロンのパラメータは重みとバイアスであり，これらを設定することで，例えば AND や OR などの論理回路を表現できる。

以上のことを数式で表すことを考える。入力される数値の個数を n，入力を $\boldsymbol{x}(\in \boldsymbol{R}^n)$，重みを $\boldsymbol{W}(\in \boldsymbol{R}^n)$，バイアスを $b(\in \boldsymbol{R})$，出力を $y(\in \boldsymbol{R})$ と表し，関数 f をステップ関数とすると，次のようになる。

$$y = f\left(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x} + b\right) \tag{1}$$

$$f(u) = \begin{cases} 1 & (u > 0) \\ 0 & (u \leq 0) \end{cases} \tag{2}$$

なお，ステップ関数の形は図 3a に示した。

例として，入力が 2 つの場合のパーセプトロンを図 1 に示す。この例では，

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \boldsymbol{W} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \tag{3}$$

$$y = f\left(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x} + b\right) = f(w_1 x_1 + w_2 x_2 + b) \tag{4}$$

$$= \begin{cases} 1 & (w_1 x_1 + w_2 x_2 + b > 0) \\ 0 & (w_1 x_1 + w_2 x_2 + b \leq 0) \end{cases} \tag{5}$$

となる。

パーセプトロンでは線形分離可能な問題しか解くことができず，例えば XOR は表現できない。しかし，これを多層にすることで線形分離できない問題でも解くことが可能になる。理論上，任意の数の中間層ノードを持った 2 層のパーセプトロンを用いれば，任意の関数を表現可能である。

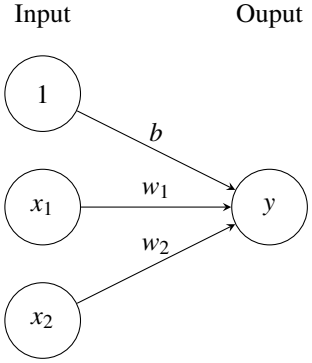


図 1: パーセプトロンの図

3 Neural Network

ニューラルネットワークとは，人間の脳内にある神経細胞（ニューロン）が繋がってできた神経回路網を，数式的なモデルで表現したものである。神経回路網においては，情報伝達は電気信号によって行われ，シナプスの結合強度によって伝達されやすさが異なる。この神経回路網は，信号をノード，シナプスの結合強度をエッジとしてモデル化することができる。

パーセプトロンにおいて，入力の重みづけ和とバイアスの和を出力に変換する関数はステップ関数であった。このような関数を活性化関数と呼ぶ。この活性化関数を他のものに変えたものがニューラルネットワークとなる。後述するように，活性化関数を非線形かつ微分した値が 0 でないものにすることで，誤差逆伝搬法によって重みとバイアスを学習することができるようになる。この活性化関数として，例えば logistic 関数が用いられる。logistic 関数はよく σ で表され，次で定義される。

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{6}$$

その他の活性化関数については 4 節にまとめる。

図 2 にニューラルネットワークの図を示す。図において，一番左の列が入力層，真ん中の層が隠れ層，一番右の層が出力層となる。この図の例はベクトルと行列を用いて次のように記述できる。

$$\boldsymbol{h} = f\left({\boldsymbol{W}^{(1)}}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{b}_1\right) \tag{7}$$

$$\boldsymbol{y} = f\left({\boldsymbol{W}^{(2)}}^{\mathrm{T}}\boldsymbol{h} + \boldsymbol{b}_2\right) \tag{8}$$

この例では隠れ層は 1 層だが，これを複数にすることもできる。隠れ層が複数ある，多層のニューラルネットワーク（Deep Neural Network: DNN）を用いた機械学習の手法を深層学習（Deep Learning）という。また以下で，パラメータという語は学習されるもの，主に重みとバイアスを指し，ハイパーパラメータは人手で設定されるもの，例えば学習率などを指す。

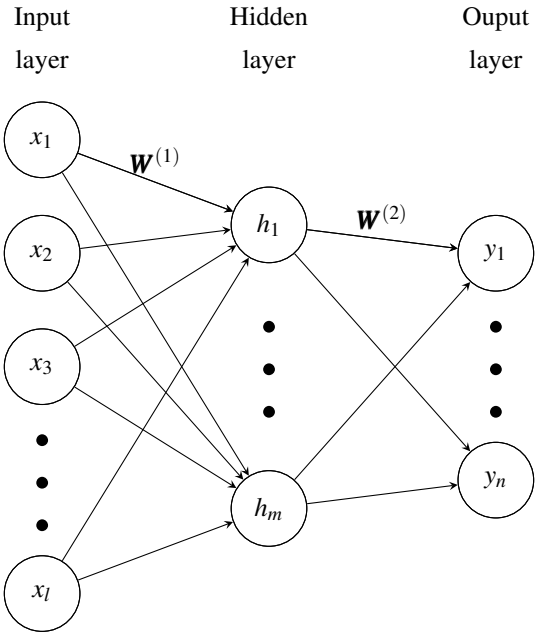


図 2: ニューラルネットワークの図

4　活性化関数

3 節で述べたように，入力の重みづけ和とバイアスの和を出力に変換する関数のことを活性化関数と呼ぶ。この活性化関数は非線形である必要がある。線形な活性化関数を用いて隠れ層のあるネットワークを構築しても，最終的に得られる出力は入力の線形結合にスカラー（バイアス）を足したものとなる。これは結局隠れ層のないネットワークによっても得られるようなものであるから，多層にする利点が失われる，つまり線形分離可能な問題しか解けなくなる。よって，活性化関数は非線形でなくてはならない。

活性化関数としてよく用いられるものに，logistic，tanh，ReLU が挙げられる。また，ReLU には負の値が入力されたときに活性化されない問題（dying ReLU）があるため，入力が負でも勾配が 0 にならないように考案された ReLU の派生形もしばしば用いられる。

以下に活性化関数の定義式とそれをプロットした図を列挙する。ReLU の派生形は代表的なもののみを挙げた。Parametric ReLU の図は Leaky ReLU の図と同じような形になるため省略した。また，softmax は 2 次元にプロットできないため，こちらも省略した。

step

$$f(x) = \begin{cases} 1 & (x > 0) \\ 0 & (x \leq 0) \end{cases} \tag{9}$$

logistic

$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}} \tag{10}$$

hyperbolic tangent

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{11}$$

softplus

$$f(x) = \log(1+e^{x}) \tag{12}$$

ReLU (Rectified Linear Unit)

$$f(x) = \max(0,\, x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \tag{13}$$

Leaky ReLU（α はハイパーパラメータ）

$$f(x) = \begin{cases} x & (x > 0) \\ \alpha x & (x \leq 0) \end{cases} \tag{14}$$

Parametric ReLU（a は学習されるパラメータの 1 つになる）

$$f(x) = \begin{cases} x & (x > 0) \\ ax & (x \leq 0) \end{cases} \tag{15}$$

Exponential Linear Units（α はハイパーパラメータ）

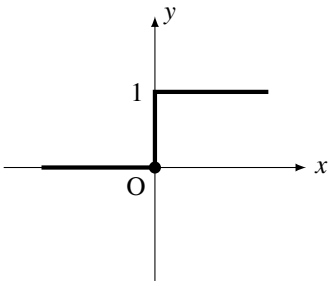
$$f(x) = \begin{cases} x & (x > 0) \\ \alpha(e^{x} - 1) & (x \leq 0) \end{cases} \tag{16}$$

恒等関数（回帰問題の出力層でよく用いられる）

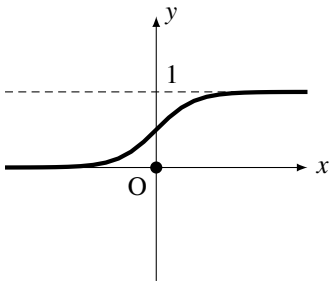
$$f(x) = x \tag{17}$$

softmax（多クラス分類問題の出力層でよく用いられる）

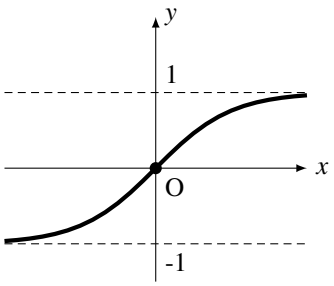
$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{18}$$



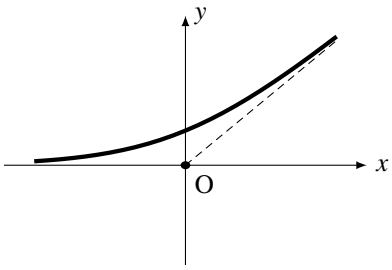
(a) ステップ関数



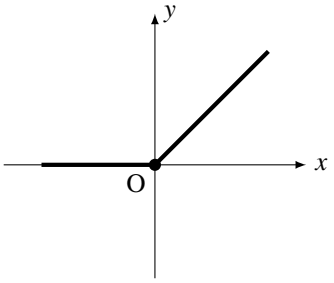
(b) sigmoid 関数



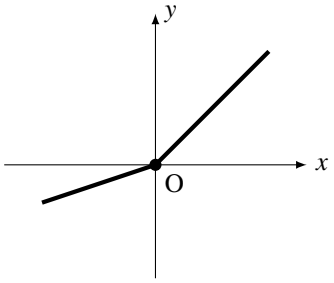
(c) tanh 関数



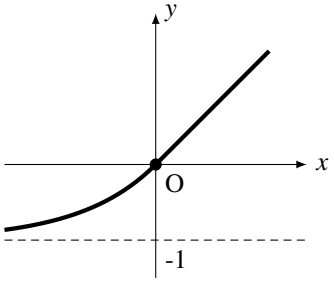
(d) softplus 関数



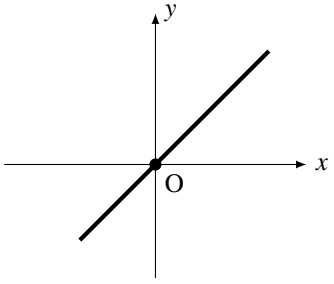
(e) ReLU 関数



(f) Leaky ReLU 関数



(g) ELU 関数



(h) 恒等関数

図 3: 活性化関数のプロット

5　学習とは

ネットワークを学習させるとは，入力 $\boldsymbol{x}$ に対し望ましい出力 $\boldsymbol{y}$ が得られるように重みとバイアスを調整するということである。この学習には大きく分けて教師あり学習と教師なし学習の 2 種類がある。教師あり学習は人間が用意した答えがある場合の学習のことを指し，教師なし学習はそうではないものを指す。

6　分類問題・回帰問題

分類問題は，入力データを予め決められたカテゴリに分けるタスクを指す。入力画像に猫が写っているか否か，センサからの入力が正常状態か異常状態かといった 2 値分類，入力画像には猫・犬・人・車のいずれかが写っているか，またはどれも写っていないかといった多クラス分類がある（この例においては 5 クラス）。2 値分類も多クラス分類の一部ではあるが，2 値分類では出力層の活性化関数として logistic 関数がよく用いられるのに対し，多クラス分類では softmax 関数が用いられることから分けて説明した。

回帰問題は，入力データから数値を予測するようなタスクを指す。株価や気温の予測等が当てはまる。また例えば物体検知では，画像中のどの位置に物体（例えば車や歩行者）があるかをピクセルレベルで予測しており，ここでも回帰問題が扱われている。出力層の活性化関数としては，よく恒等関数が用いられる。

6.1　分類問題の評価指標

分類問題におけるモデルの評価は，単純な正解率以外によっても行うことができる。極端な例として，例えば負例が正例に比べて非常に多いデータで二値分類を行うときを考える。このような問題に対しては，入力が何であれ False を出力するようなモデルでもその正解率が 9 割を超えてしまう。これはいいモデルとは言えず，正解率のみではモデルの評価としては不十分であるといえる。

以下では二値分類問題を扱う。多クラス分類問題においても同様の評価指標を導入できるが，ここでは省略する。二値分類問題では，真の結果が正負の 2 パターンあり，それに対する予測も正負の 2 パターンある。これを，例えば真の結果も予測結果も True であったものを TP (True Positive) などとして，表 1 のように分類する。予測結果と真の値が一致したものに True，そうでないものに False と，予測値が正だったものに Positive，そうでないものに Negative というラベルをつけている。この表を混同行列（confusion matrix）という。以下では，この混同行列を用いていくつかの評価指標を導入する。

正解率（accuracy）は，分類結果のうち，予測された値と真の値が一致したものの割合である。

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}} \tag{19}$$

適合率（precision）は，予測された値が正だったもののうち，真の値も正であったものの割合である。

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{20}$$

再現率（recall）は，真の値が正のもののうち，予測された値も正であったものの割合である。

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{21}$$

F 値（F-measure）は，適合率と再現率の調和平均である。適合率と再現率はトレードオフの関係にあるため，予測モデルの性能を評価しやすくするために導入されることが多い。

$$\mathrm{F} = \frac{2 \cdot \mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \tag{22}$$

表 1: 混同行列

		Predicted Class	
		Positive	Negative
Actual Class	Positive	TP	FN
	Negative	FP	TN

6.2 回帰問題の評価指標

回帰問題におけるモデルの評価指標にも様々なものがあり，やはり問題によってどれを重視するかは異なってくる。以下に代表的な指標を列挙する。t が実測値を，y が推定値を，N がデータの個数を表す。

平均二乗誤差（Mean Squared Error: MSE）

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2 \tag{23}$$

平均二乗平方根誤差（Root Mean Squared Error: RMSE）

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2} \tag{24}$$

平均絶対誤差（Mean Absolute Error: MAE）

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|t_i - y_i| \tag{25}$$

決定係数は実測値と推定値の相関係数を表す。値が 1 に近いほど性能が良い。なお，式中の $\bar{y}$ は推定値 y の平均値である。

$$\mathrm{R}^2 = 1 - \frac{\sum_{i=1}^{N}(t_i - y_i)^2}{\sum_{i=1}^{N}(t_i - \bar{y})^2} \tag{26}$$

Observed-Predicted Plot は，横軸に実測値，縦軸に推定値をプロットしたもので，原点を通り傾き 1 の直線付近に点が集まるほど性能が良いことを表す。

7 損失関数

入力 $\boldsymbol{x}$ に対し望ましい出力 $\boldsymbol{y}$ が得られていれば，ネットワークの学習はうまくいったといえる。つまり，出力がどれくらい望ましいかを評価する尺度が重要になる。この尺度を損失関数（loss function）と呼ぶ。損失関数は学習の進行度合いを表している。一般に，この損失関数は学習が進み出力が望ましいものになるほど小さくなるような関数となるため，Deep Learning における学習は損失関数を最小化する最適化問題となる。

損失関数を E と表すこととする。損失関数 E は，望ましい出力の値 t とネットワークの出力 y から計算される。損失関数は出力の望ましさの尺度であり，扱う問題によって出力の種類は異なるため，それぞれに対し適切な損失関数を設定する必要がある。例えば分類問題のときは，損失関数には基本的に交差エントロピー誤差が用いられる。回帰問題のときは 2 乗和誤差が用いられることが多い。以下に損失関数の代表例の定義式を列挙する。

2 乗和誤差

$$E = \frac{1}{2}\sum_k (y_k - t_k)^2 \tag{27}$$

交差エントロピー誤差

$$E = -\sum_k \{t_{nk} \ln y_{nk} + (1 - t_{nk}) \ln(1 - y_{nk})\} \tag{28}$$

$$= -\sum_k t_k \ln y_k \qquad (\text{when classification problem}) \tag{29}$$

ロジスティック損失（ロジスティック回帰）

$$E = \ln(1 + \exp(-ty)) \tag{30}$$

指数損失

$$E = \exp(-ty) \tag{31}$$

ヒンジ損失

$$E = \max(1 - ty,\ 0) \tag{32}$$

τ-分位損失（分位点回帰）

$$E = (1 - \tau)\max(y - t,\ 0) + \tau \max(t - y,\ 0) \tag{33}$$

Huber 損失（外れ値に大きく影響されるとき）

$$E = \begin{cases} (t-y)^2 & (y \in [t-\delta,\ t+\delta]) \\ 2\delta\left(|t-f| - \frac{\delta}{2}\right) & (\text{otherwise}) \end{cases} \qquad (\delta \in \boldsymbol{R}_{>0}) \tag{34}$$

ε-許容（感度）損失

$$E = \max(|t - y| - \varepsilon, 0) \qquad (\varepsilon \in \boldsymbol{R}_{>0}) \tag{35}$$

8 勾配降下法

7 節から，ネットワークを学習させるためには損失関数 E を最小化すればよいことがわかった。この節ではその最適化手法について考える。

8.1 勾配降下法

いま，ネットワークのパラメータ（主に重みとバイアス）をまとめて $\boldsymbol{\theta}$ と表すこととする。学習の最終的な目標は，

$$\boldsymbol{\theta} = \mathrm{argmin}_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) \tag{36}$$

となる $\boldsymbol{\theta}$ を設定することである。ここで，E は全データに対して計算した損失関数である。しかし，E は凸関数とは限らないため，極小値に至ったとき，そこが大域的な最小値であるとは限らない。しかし，大域的な最小値でなくとも，ある程度小さい値になっていれば，ある程度の性能を持つネットワークが学習されていると考えられる。よって，E に極小値を与えるような $\boldsymbol{\theta}$ を設定することを考える。

このような最適化手法は 1 つではないが，パラメータの数が多くなる深層学習のモデルにおいては，勾配（1 次微分）までしか用いずに計算できるという点で，勾配降下法（gradient descent method）が有利である。パラメータの数を M とすると，勾配は次のように表される。

$$\nabla_{\boldsymbol{\theta}} E \equiv \frac{\partial E}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \dfrac{\partial E}{\partial \theta_1} & \dfrac{\partial E}{\partial \theta_2} & \cdots & \dfrac{\partial E}{\partial \theta_M} \end{bmatrix}^{\mathrm{T}} \tag{37}$$

勾配が負の方向がすなわち極小値の方向であるため，その方向にパラメータを更新していけばよいというのが勾配降下法である。つまり，次のようにパラメータを更新する。

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} E \tag{38}$$

ここで，$\eta (\in \boldsymbol{R}_{>0})$ は学習率（learning rate）であり，更新の程度を表すハイパーパラメータである。

損失関数のパラメータによる偏微分は，例えば数値微分によって求めることができる。つまり，あるパラメータ θ に着目し，他のパラメータを固定して次のようにして計算できる。

$$\frac{\partial E}{\partial \theta} = \frac{E(\theta + \varepsilon) - E(\theta - \varepsilon)}{2\varepsilon} \tag{39}$$

ここで ε は十分小さい値である。これを全パラメータに関して行うことで，$\nabla_{\boldsymbol{\theta}} E$ を求めることができる。

8.2 学習率

学習率を十分小さくすれば確実に損失関数を減少させることができるため，十分な回数更新を繰り返せば確実に極小値に至らせることができる。しかし，これでは学習時間が増大してしまい，また局所的な極小値に捕われやすくもなってしまう。一方，学習率を大きくしすぎると E が増大して発散することも起こりうる。よって，学習率をちょうどよい値に設定することが，学習の効率的な進行に大きい影響を与えることになる。

8.3 確率的勾配降下法

勾配降下法では，全データ用いて計算した損失関数を用いてパラメータの更新を行っていた。このような学習法をバッチ学習という。一方，確率的勾配降下法（stochastic gradient descent: SGD）は，一つひとつのデータに対して同様の操作をしてパラメータの更新を行う手法である。バッチ学習をすると，局所的な極小値に捕らわれたときに抜け出すことができなくなる。しかし，一つひとつのデータに対して勾配を求めれば，それぞれの勾配は同じ方向を向いてはいないので，局所的な極小値には捕われにくくなる。

8.4 ミニバッチ学習

バッチ学習よりも SGD の方が深層学習における学習法としては有利な点が多い。しかし，データ一つひとつを用いる学習法では，並列計算をすることができず効率が悪い。そこで，両者の間を取って，全データのうち一部をひとまとめにし，それを 1 つの単位として学習させることを考える。このひとまとめのデータをミニバッチといい，この学習方法をミニバッチ学習という。ここで，損失関数の値はミニバッチに含まれるデータ数で平均する。この，ミニバッチに含まれるデータ数を，バッチサイズ（batch size）という。

9 誤差逆伝搬法

8 節より，勾配降下法を用いれば学習させられることが分かった。しかし，パラメータの各成分についての偏微分を数値微分で求めるのは計算コストが非常に高い。複雑な問題を解かせるときにはパラメータの数は爆発的に増えるため，その偏微分の数値微分の計算にかかる時間も大きく増加する。これを何度も繰り返して学習させなければならないため，現実的な計算時間ではなくなってしまう。誤差逆伝搬法を用いることで数値微分よりも計算コストを抑えて偏微分を計算することができる。

L 層のニューラルネットワークにおける，ある層 l を考える。変数やパラメータには上付きで (l) と付けることで l 層目であることを示す。また，隠れ状態 $\boldsymbol{h}^{(l)}$ の第 0 要素に 1 を追加し，$l-1$ 層と l 層の層間のパラメータ $\boldsymbol{\theta}^{(l)}$ は，第 0 行にバイアス $b^{(l)}$ を持ち，残りは重み $\boldsymbol{W}^{(l)}$ であるとする。このとき，次式が成立する。

$$\boldsymbol{u}^{(l)} = {\boldsymbol{\theta}^{(l)}}^{\mathrm{T}} \boldsymbol{h}^{(l-1)} \tag{40}$$

$$\boldsymbol{h}^{(l)} = f^{(l)}\left(\boldsymbol{u}^{(l)}\right) \tag{41}$$

となる。これを要素に分解して考えると，

$$u_j^{(l)} = \sum_i \theta_{ij}^{(l)} h_i^{(l-1)} \tag{42}$$

$$h_j^{(l)} = f^{(l)}\left(u_j^{(l)}\right) \tag{43}$$

である。損失関数 E の θ_{ij} による偏微分は，

$$\frac{\partial E}{\partial \theta_{ij}^{(l)}} = \frac{\partial E}{\partial u_j^{(l)}} \frac{\partial u_j^{(l)}}{\partial \theta_{ij}^{(l)}} \tag{44}$$

と書ける。ここで，

$$\frac{\partial u_j^{(l)}}{\partial \theta_{ij}^{(l)}} = h_i^{(l-1)} \tag{45}$$

であることから，

$$\delta_j^{(l)} \equiv \frac{\partial E}{\partial u_j^{(l)}} \tag{46}$$

とおくと，

$$\frac{\partial E}{\partial \theta_{ij}^{(l)}} = \delta_j^{(l)} h_i^{(l-1)} \tag{47}$$

$$\frac{\partial E}{\partial \boldsymbol{\theta}^{(l)}} = \boldsymbol{h}^{(l-1)} \boldsymbol{\delta}^{(l)\mathrm{T}} \tag{48}$$

と書ける。よって，損失関数の $\theta_{ij}^{(l)}$ による偏微分を δ と隠れ状態 h で表せた。

以下では δ を求めることを考える。連鎖率から，

$$\frac{\partial E}{\partial u_j^{(l)}} = \sum_k \frac{\partial E}{\partial u_k^{(l+1)}} \frac{\partial u_k^{(l+1)}}{\partial u_j^{(l)}} \tag{49}$$

ここで，

$$\frac{\partial u_k^{(l+1)}}{\partial u_j^{(l)}} = \frac{\partial u_k^{(l+1)}}{\partial h_j^{(l)}} \frac{\partial h_j^{(l)}}{\partial u_j^{(l)}} = \theta_{jk}^{(l+1)} f^{(l)\prime}\left(u_j^{(l)}\right) \tag{50}$$

となることから，結局，

$$\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} \theta_{jk}^{(l+1)} f^{(l)\prime}\left(u_j^{(l)}\right) \tag{51}$$

$$\boldsymbol{\delta} = \boldsymbol{\theta}^{(l+1)} \boldsymbol{\delta}^{(l+1)} \odot f^{(l)\prime}\left(\boldsymbol{u}^{(l)}\right) \tag{52}$$

という式を得る。この式から，ある層における $\boldsymbol{\delta}$ が求まれば，それより 1 つ浅い層の $\boldsymbol{\delta}$ も求まることがわかる。つまり，出力層における $\boldsymbol{\delta} = \boldsymbol{\delta}^{(L)}$ が求まれば，入力層側の $\boldsymbol{\delta}$ まで全て求まることになる。

出力層の $\delta^{(L)}$ を求める。望ましい出力を $\boldsymbol{t}$ とする。当然，損失関数 E と出力層の活性化関数 $f^{(L)}$ によって計算過程は異なる。例として，損失関数に 2 乗和誤差を，出力層の活性化関数に恒等関数を用いた場合（回帰問題）を考える。

$$\boldsymbol{y} = \boldsymbol{h}^{(L)} = \boldsymbol{u}^{(L)} \tag{53}$$

$$E = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{t}\|^2 = \frac{1}{2}\sum_i (y_i - t_i)^2 \tag{54}$$

式 (46) より，

$$\delta_i^{(L)} = \frac{\partial E}{\partial u_i^{(L)}} = \frac{\partial E}{\partial y_i} = y_i - t_i \tag{55}$$

よって，

$$\boldsymbol{\delta}^{(L)} = \boldsymbol{y} - \boldsymbol{t} \tag{56}$$

と求まる。なお，損失関数に交差エントロピー誤差を，活性化関数に logistic 関数を用いたとき（2 値分類問題），及び損失関数に交差エントロピー誤差を，活性化関数に softmax 関数を用いたとき（多クラス分類）の出力層における $\boldsymbol{\delta}$ も，式 (56) と同じ形で表すことができる。

以上まとまると，1 つのデータの集まりに対する学習の手順は次のようになる。

1. $\boldsymbol{x}$ を入力して各層の $\boldsymbol{u}$, $\boldsymbol{h}$ を計算する
2. 出力層の $\boldsymbol{\delta}^{(L)}$ を求める（式 (56) のようになることが多い）
3. 式 (52) を用いて各層の $\boldsymbol{\delta}$ を計算する
4. 式 (48) を用いて各パラメータによる損失関数の偏微分を計算する
5. 式 (38) を用いてパラメータを更新する

10 最適化手法

パラメータの更新は，式 (38) 以外の式によっても行うことができる。安定して極小値に収束させるために，様々な手法が提案されている。式 (38) を含め，以下にそれらを列挙する。ここで，$\odot$ は行列の要素ごとの積を表す。

SGD

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial E}{\partial \boldsymbol{\theta}} \tag{57}$$

Momentum（$\alpha \in [0,\, 1]$ はハイパーパラメータ）

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \eta \frac{\partial E}{\partial \boldsymbol{\theta}} \tag{58}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v} \tag{59}$$

AdaGrad

$$\boldsymbol{h} \leftarrow \boldsymbol{h} + \frac{\partial E}{\partial \boldsymbol{\theta}} \odot \frac{\partial E}{\partial \boldsymbol{\theta}} \tag{60}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{1}{\sqrt{\boldsymbol{h}}} \frac{\partial E}{\partial \boldsymbol{\theta}} \tag{61}$$

RMSprop（α, ε はハイパーパラメータ）

$$\boldsymbol{h} \leftarrow \alpha \boldsymbol{h} + (1-\alpha) \frac{\partial E}{\partial \boldsymbol{\theta}} \odot \frac{\partial E}{\partial \boldsymbol{\theta}} \tag{62}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{1}{\sqrt{\boldsymbol{h}} + \varepsilon} \frac{\partial E}{\partial \boldsymbol{\theta}} \tag{63}$$

Adam（β_1, β_2, ε はハイパーパラメータ，t はイタレーション回数）

$$\boldsymbol{m} \leftarrow \beta_1 \boldsymbol{m} + (1-\beta_1) \frac{\partial E}{\partial \boldsymbol{\theta}} \tag{64}$$

$$\boldsymbol{v} \leftarrow \beta_2 \boldsymbol{v} + (1-\beta_2) \frac{\partial E}{\partial \boldsymbol{\theta}} \odot \frac{\partial E}{\partial \boldsymbol{\theta}} \tag{65}$$

$$\hat{\boldsymbol{m}} = \frac{\boldsymbol{m}}{1-\beta_1^t} \tag{66}$$

$$\hat{\boldsymbol{v}} = \frac{\boldsymbol{v}}{1-\beta_2^t} \tag{67}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\hat{\boldsymbol{m}}}{\sqrt{\hat{\boldsymbol{v}}} + \varepsilon} \tag{68}$$

11　重みの初期化

学習の開始時には重みを適当な値で初期化する必要がある。1 層分の重みを全て同じ値で初期化すると，その出力は全て同じ値になる。このとき，誤差を逆伝搬させると，その層の重みは全て同じ値だけ更新される。従って，学習を進めてもその層の重みは全て同じ値のままとなり，その結果出力の値も全て同じ値のままになる。よって，重みを均一な値で初期化すると，学習ができないという状況になってしまう。以上のことから，重みは異なる値を持つように，例えば一様分布や正規分布等で初期化しなければならない。なお，バイアスにはこういった問題はないため，基本的には 0 で初期化する。

例えば，正規分布を用いて初期化するときを考えると，その分散をうまく設定する必要があることがわかる。分散が小さすぎるときは，重みの値は全てほぼ 0 となり，上記のように学習が進まないという状況に陥ってしまう。また，logistic 関数のような活性化関数では，出力の値が大きいと勾配が小さくなるため，分散が大きく値の大きい初期値である場合も学習がうまく進まない。よって，重みの初期化を適切に行えるような分散の値を選ぶ必要がある。

以下に重みの初期化に用いられる代表的な分布を列挙する。i 層目の重みを W_i，その入力の数を n_i とする。

Uniform Initialization

$$W_i \sim U[-1,\ 1] \tag{69}$$

Xavier (Glorot) Initialization（ReLU 以外の活性化関数を用いる場合）

$$W_i \sim N\left[0,\ \frac{1}{\sqrt{n_i}}\right] \tag{70}$$

He Initialization（ReLU を活性化関数として用いる場合）

$$W_i \sim N\left[0,\ \sqrt{\frac{2}{n_i}}\right] \tag{71}$$

Gauss Initialization

$$W_i \sim N[0,\ 1] \tag{72}$$

12　正則化

訓練に用いたデータに適応しすぎてしまうことを過学習（over fitting）という。機械学習ではパラメータの数が膨大になるため，訓練データを（ほぼ）完全に覚えてしまうということが起こる可能性がある。それが過学習である。過学習をすると，新しいデータに対しての予測性能，つまり汎化性能が低くなってしまう。

過学習をしているかどうかを確認するためには，パラメータの更新に用いない検証用のデータを用意し，検証データにおける性能を測りながら訓練を行う必要がある。性能の測り方としては，損失関数や精度をプロットするといった方法が挙げられる。訓練データと検証データの間の乖離が大きくなった場合，そのモデルは過学習しているといえる。また，最終的な成果物としてのモデルには，検証データにおける性能が最も高いものを採用するような形がしばしば取られる。しかし，これでは検証データに過適合する可能性があるため，また別の同規模のデータを用意して過適合していないことを補償することもある。このデータをテストデータと呼ぶ。つまり，データを訓練・検証・テストの 3 つに分割して学習を行う。これらの比は 8:1:1 や 6:2:2 とすることが多いが，検証・テストデータに関しては 1 万データあれば十分という考え方もある。

過学習の原因はデータの量に対してパラメータの数が多すぎることにある。従って，過学習を抑制するには，データの量を増やす，もしくはパラメータの数を減らすといったことがまず考えられる。しかし，モデルの性能を上げるためにはパラメータを増やす必要がある。よって，訓練データに対する精度が思うように出ていないにも関わらず過学習が発生した場合，パラメータの数を減らすことはできない。このとき，検証データにおける性能を上げるためには，まずデータの量を増やすことが考えられる。また，検証データに対する誤差が増加した場合，それ以上学習を進めても過学習をする一方なので，その時点で学習を止めるという方法を取ることもできる。これを早期終了（early stopping）という。

しかし，その他にも，重みに制限をかけるなどして過学習を防ぐことができる。これを正則化という。正則化によって，基本的に訓練データに対する性能は下がるが，汎化性能は上がる可能性がある。この節では，その正則化手法をいくつか紹介する。

12.1 L2 正則化

重み減衰（weight decay）とも呼ばれる。また，L2 正則化回帰のことを Ridge 回帰と呼ぶこともある。この正則化手法では，損失に重みの 2 乗の項を加える。つまり，

$$J = L(t,\, y) + \lambda |\boldsymbol{W}|^2 \tag{73}$$

ここで，λ はハイパーパラメータである。この項を損失に加えることにより，値の大きいパラメータを小さくするように学習が行われ，結果として過学習が防がれる場合がある。

12.2 Dropout

Dropout は，訓練時にニューロンのうち一部を確率的に不活性化させて学習させることにより，汎化性能を上げる手法である。つまり，Dropout を適用する層において，重みのうち一部を確率的に選び，選ばれたものの重みを 0 として順伝搬を行い，また逆伝搬時にも重みの更新をしないようにする。ここで，Dropout する確率はハイパーパラメータとなる。推論時には，全てのノードを用いて計算を行った後，訓練時に Dropout していた割合を出力にかけることで，用いているノードの数が訓練時より増加し大きくなった出力の値を修正している。このように学習することによって，複数のネットワークを独立に学習し，推論時にその出力を平均する，アンサンブル学習の近似になるといわれている。

12.3 Batch Normalization

Batch Normalization は単なる正則化手法というわけではないが，ここで解説することとする。第 i 層への入力を x_i，ミニバッチの大きさを m とする。この手法では，名前の通りミニバッチごとに正規化を行う。

$$\mu_{\mathrm{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{74}$$

$$\sigma_{\mathrm{B}}{}^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathrm{B}})^2 \tag{75}$$

$$\hat{x_i} = \frac{x_i - \mu_{\mathrm{B}}}{\sqrt{\sigma_{\mathrm{B}}{}^2 + \varepsilon}} \tag{76}$$

ここで，ε は 0 除算を回避するための定数であり，非常に小さい値である。このように正規化された入力 $\hat{x}_i$ に対し，次のような変換を行う。

$$y_i = \gamma \hat{x}_i + \beta \tag{77}$$

ここで，γ と β はパラメータであり，それぞれ 1 と 0 を初期値として学習される。なお，Batch Normalization を活性化関数の前と後ろのどちらかに置くかについては議論されている。

以上の過程が Batch Normalization 層における操作である。なお，推論時にはデータはミニバッチとしてではなく 1 つだけで入ってくることが想定されるため，μ_{B} と σ_{B} は学習時の移動平均を用いる。Batch Normalization には正則化の他に，損失が発散しにくくなるため学習率を大きくして学習にかかる時間を短くできる，初期値への依存性が下がるといった効果がある。