ニューラルネットワーク

はじめに

今(2023.3)から 5 年ほど前の一時期, DL(Deep Learning)に興味を持ち, 専門書を何冊か読破したことがある[1]〜 [6]. 筆者が学生(30 年以上昔)のころの記憶[4]にはない項目が興味を引いた.

＊事前学習, 自己符号化
　恒等写像による重みの最適化
＊畳み込み層, 正規化層, プーリング層による機能の細分化
＊活性化関数 ReL(中間層), Softmax(出力層)
＊対数尤度(KL-divergence)

筆者の場合, Python, MATLAB といったライブラリを使うことはなく, 基本的に自作である. 長い間の習慣で, ライブラリはアルゴリズムを理解するためのツールである. 当時, 10 層の CNN を C++で実装し, MINIST の手書き文字の認識を試行した(主に[1]を参照). プログラムはあっさりと動作し, DL の性能(特に活性化関数)に感銘を受けた次第である.

とはいえ, 第一に線形な手法, 次に確率・統計, それでもダメならそれら以外の非線形な手法, というのが筆者の姿勢である. そもそも一意な解, 解の収束, 安定性, そして計算コストにおいておよそ線形な手法が有利というのは間違いない. 筆者が従事してきた計測機器開発(特に医療器)では解の確度, 安全性を求められる. DL はこのような環境には厳しい. 結果に対する理由, 原因の特定が難しく, それこそが最大の弱点かと感じた.

本稿ではニューラルネットワークのプログラムを自作したい向きを対象とし, 手順を記述しようと思う. また, これに併せて**筆者が提案する学習速度を向上させる手法**についても述べる. 当時, 片手間で論文にまとめわけだが査読は通らなかった. その理由として, 数理での証明がなく, ない場合は大量のデータによる確度を示せ, とのご指摘をいただいた. 数理での証明は難しく, かといって専門家ではない筆者にとり, 下記が課題である.

＊どのようなネットワーク構成
＊どのようなデータ?
＊どの程度の量？

実のところ, 筆者の興味案件リストからは除外, 放置となっていた.

誤差逆伝搬

誤差逆伝搬のフローを示す(**図 1**).

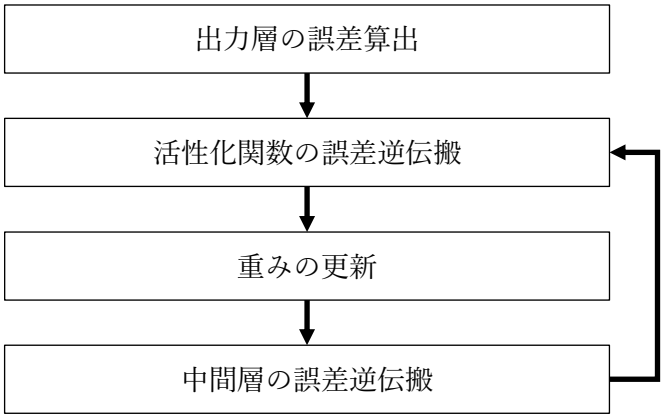

図 1　誤差逆伝搬のフロー

出力層の誤差算出

教師信号との誤差を算出する(1) (**図 2**).

$$E = \frac{1}{2}\left(z_p - t_p\right)^2$$

$$\frac{dE}{dz_p} = z_p - t_p \quad (1)$$

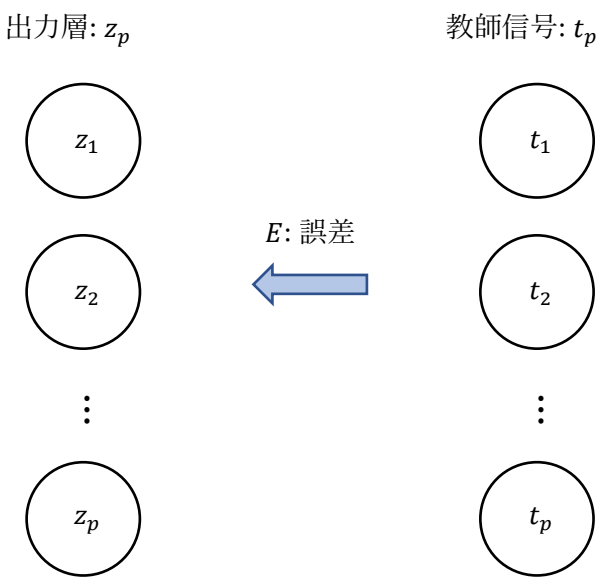

図 2　出力層の誤差算出

活性化関数の誤差逆伝搬

活性化関数の誤差逆伝搬を算出する(2)(**図 3**).

$$\frac{dE}{du_p} = f' \frac{dE}{dz_p} \quad (2)$$

f': 活性化関数

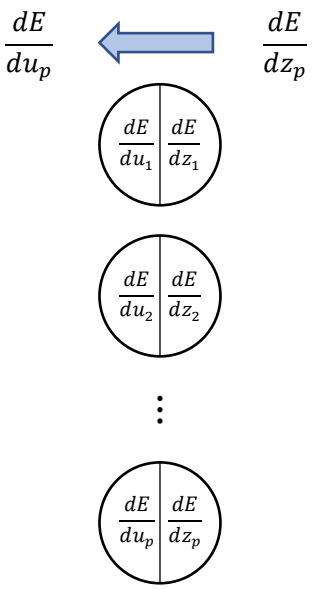

図 3　活性化関数の誤差逆伝搬

重みの更新

重みを更新する(3)(**図 4**).

$$w_{pq} = w_{pq} - \eta \frac{dw_{pq}}{du_q}$$

$$= w_{pq} - \eta \frac{dE}{du_q} z_p \quad (3)$$

η: 学習係数

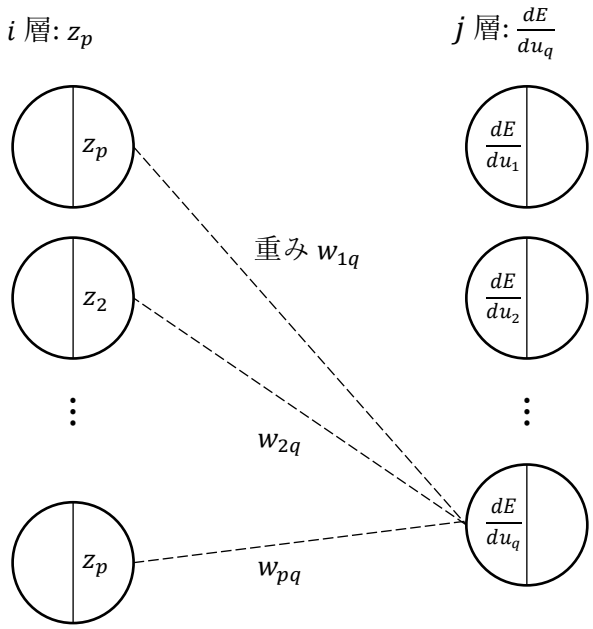

図 4　重みの更新(層間全結合)

なお, モメンタムを付す場合, (4)である.

$$w_{pq} = w_{pq} - \eta \frac{dE}{du_q} z_p + \alpha \Delta w_{pq} \quad (4)$$

α: 安定化係数

Δw_{pq}: 前回の重みの更新量

中間層の誤差逆伝搬

中間層の誤差逆伝搬を算出する(5)(**図 5**).

$$\frac{dE}{dz_p} = \sum_q w_{pq} \frac{dE}{du_q} \quad (5)$$

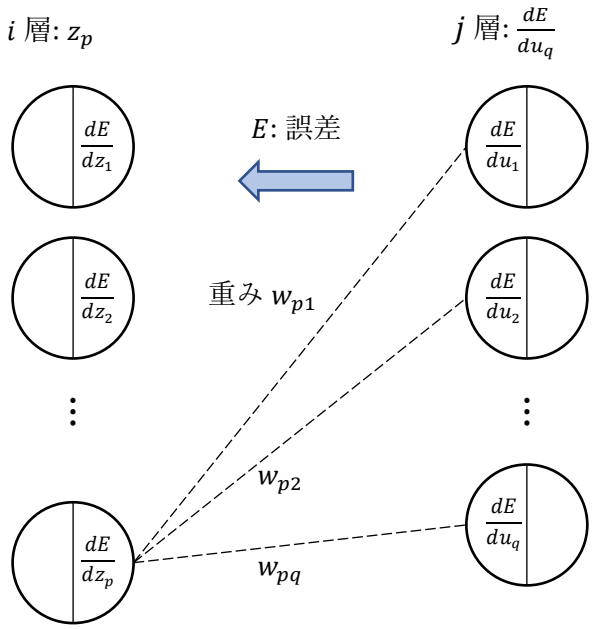

図 5　中間層の誤差逆伝搬

順伝搬

筆者が提案する学習速度を向上させる手法を含めた順伝搬が(6)～(8)である(**図 6**).

$$u_q = \sum_p w_{pq} z_p + b_q \quad (6)$$

$$v_q = g_q u_q \quad (7)$$

$$z_q = f(v_q) \quad (8)$$

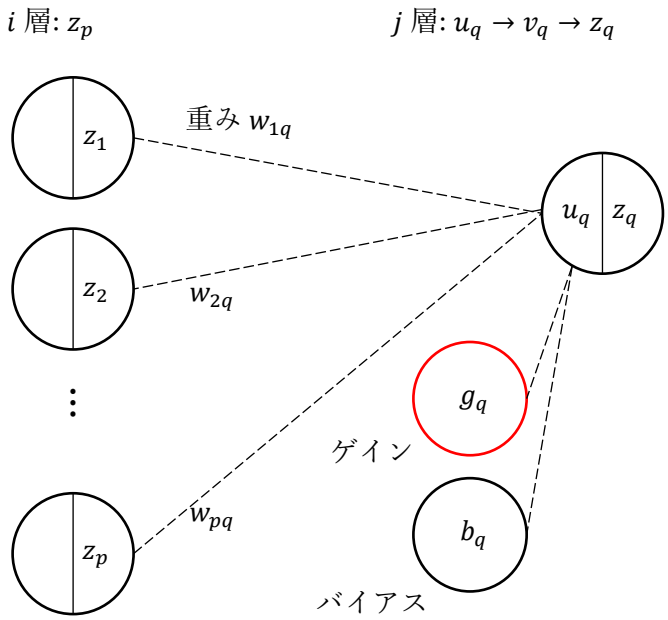

図 6　順伝搬

通常のユニット入力は(6)であるが, 提案手法では(7)によりゲインを乗じ, (8)の入力とする. なお, 筆者はバイアスというものを使用したことがない.

誤差逆伝搬の勾配

通常, 誤差逆伝搬の勾配は(9)である. 順伝搬で$f(v_q)$としたが(**図 6**), 提案手法の**要点**として(9) のままとする.

$$f'(u_q) \quad (9)$$

学習後のネットワーク

提案手法では, 学習後, ゲインを残したままでもよいが, (7)より(10)とできる.

$$v_q = g_q \left(\sum_p w_{pq} z_p + b_q \right) = \sum_p g_q w_{pq} z_p + g_q b_q \quad (10)$$

学習の重みとバイアスに関して, $w_{pq} \Rightarrow g_q w_{pq}$, $b_q \Rightarrow g_q b_q$と更新する. これによりゲインを除去して通常のネットワーク構成とすることができる.

まとめ

ニューラルネットワークのプログラミングに十分な内容は述べたかと思う. 提案手法については, 筆者が確認したいくつかの小規模なネットワーク(3〜4 層, ノード数 8 ケ以下)と 10 層の CNN で学習回数を短縮する効果があることを確認している. 提案手法は**順伝搬で誤差を増幅させる手法**, と筆者は考えている. これは筆者の感覚であるが, スポーツ等で, わざと大きな動作をして失敗してみる, ということはないだろうか? 大きな失敗をすることで修正の方向性が明確になるということがある. 提案手法は, これが発端である. そして, 既存の専門書からは発生してしまった誤差の修正に議論が集中しているように思われた. 冒頭で述べたように, 大量のデータでの検証は筆者には厳しい. 本稿をご覧になり, 思うところがあれば, ご助力, あるいは協力いただけないかという次第である.

参考文献

[1] 岡谷貴之. 深層学習. 講談社. 2015.

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville. 深層学習. ASCII ドワンゴ. 2018.

[3] 藤田毅. C++で学ぶディープラーニング. マイナビ出版. 2017.

[4] 平野廣美. C でつくるニューラルネットワーク. パーソナルメディア. 1991.

[5] Sebastian Raschka. Python 機械学習プログラミング. インプレス. 2016.

[6] 小高知宏. 機械学習と深層学習 オーム社. 2016.