# 深層学習（day2）・要点まとめ

## Section0：深層学習全体像の復習　－学習概念

### ▣ 誤差逆伝搬法の復習

《確認テスト》：連鎖律の原理を使い、dz/dx を求めよ。

⇒ $dz/dx = (dz/dt)\cdot(dt/dx) = 2t\cdot 1 = 2t = 2(x+y)$

## Section1：勾配消失問題について

### ▣ 勾配消失問題

誤差逆伝搬法が下位層に進んでいくに連れて、勾配がどんどん緩やかになっていく。そのため、勾配降下法による更新では、下位層のパラメータはほとんど変わらず、訓練は最適値に収束しなくなる。

《確認テスト》：シグモイド関数を微分した時、入力値が 0 の時に最大値 x をとる。

シグモイド関数　$\sigma(u)=\frac{1}{1+e^{-u}} \rightarrow \frac{d}{du}\sigma(u)=\frac{1}{1+e^{-u}}\cdot\frac{e^{-u}}{1+e^{-u}}=\sigma(u)(1-\sigma(u))$

$$\therefore \frac{d}{du}\sigma(0)=\sigma(0)(1-\sigma(0)) = 0.5^2 = 0.25$$

【考察】

- シグモイド関数はその微分値が最大でも 0.25 しかなく、深層になるほど以下のように勾配にシグモイド関数の微分が多重になり勾配が微小になるため、**勾配消失問題**を引き起こす。

$$\frac{\partial E}{\partial y}\frac{\partial y}{\partial u^{(3)}}\frac{\partial u^{(3)}}{\partial z^{(2)}}\boxed{\frac{\partial z^{(2)}}{\partial u^{(2)}}}\frac{\partial u^{(2)}}{\partial z^{(1)}}\boxed{\frac{\partial z^{(1)}}{\partial u^{(1)}}}$$

［勾配消失問題の解決法］：

### 1-1 活性化関数の選択

今最も使われている関数　ReLU 関数：$f(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$

【考察】

- ReLU 関数はその微分値が$x > 0$では 1 となるため勾配消失問題の回避に繋がり、$x \leq 0$では 0 となるためスパース化（重みを 0 にして特徴量を減らす）に貢献することで良い成果をもたらす。

### 1-2 重みの初期値設定

A) **Xavier** の初期値：

設定方法：重みの要素を、前の層のノード数の平方根で除算した値

Xavier の初期値を設定する際の活性化関数：ReLU、シグモイド、双曲線正接（tanh）関数

```
# Xavier の初期値
network['W1'] = np.random.randn(input_layer_size,
                hidden_layer_1_size) / (np.sqrt(input_layer_size))
    network['W2'] = np.random.randn(hidden_layer_1_size,
                hidden_layer_2_size) / (np.sqrt(hidden_layer_1_size))
```

B) **He** の初期値：

設定方法： 重みの要素を、前の層のノード数の平方根で除算した値に、$\sqrt{2}$を掛けた値

Xavier の初期値を設定する際の活性化関数： ReLU 関数

```
# He の初期値
network['W1'] = np.random.randn(input_layer_size, hidden_layer_1_size)
              / np.sqrt(input_layer_size) * np.sqrt(2)
network['W2'] = np.random.randn(hidden_layer_1_size, hidden_layer_2_size)
              / np.sqrt(hidden_layer_1_size) * np.sqrt(2)
```

《確認テスト》： 重みの初期値に 0 を設定すると、どんな問題が発生するか？

⇒ w=0 となることで、総入力 $\boldsymbol{u}^{(l)}=\boldsymbol{w}^{(l)}\boldsymbol{z}^{(l-1)}+\boldsymbol{b}^{(l)}$が 0 となり、中間層の出力$\boldsymbol{z}^{(l)}$が活性化関数により 0 か 0.5 しかとらなくなる。従って、勾配の大部分が 0 となり学習が進まない。

## 1-3 バッチ正規化

ミニバッチ単位で、入力値のデータの偏りを抑制する手法 → 活性化関数に値を渡す前後に、バッチ正規化の処理を施した層を加える。

⇒ バッチ正規化層への入力値は、

$\boldsymbol{u}^{(l)}=\boldsymbol{w}^{(l)}\boldsymbol{z}^{(l-1)}+\boldsymbol{b}^{(l)}$（活性化関数に値を渡す前） または $\boldsymbol{z}$（活性化関数の処理後の値）

《確認テスト》： バッチ正規化の効果

①大きな学習係数が使える･･･伝播中パラメータの scale に影響を受けなくなる。結果的に学習係数を上げることができ、学習の収束速度が向上する。

②正則化効果がある･･･これまでの正則化テクニック（L2 正則化、Dropout）を不要にできる。Dropout は、過学習を抑える働きがあるが学習速度が遅くなり、Dropout を使わないことで学習速度を向上させることができる。

③初期値にそれほど依存しない･･･ニューラルネットワークの重みの初期値がそれほど性能に影響を与えなくなる。

＜参考資料＞ “Batch Normalization：ニューラルネットワークの学習を加速させる汎用的で強力な手法” https://deepage.net/deep_learning/2016/10/26/batch_normalization.html

【**勾配消失問題**の解決実装の考察（2_2_1_vanishing_gradient.ipynb）（2_2_1_vanishing_gradient2.ipynb）】：

- 確かにシグモイド関数で重み w に正規分布の初期値を与えた場合はほとんど学習できていない。それに対して、ReLU 関数は正規分布の初期値でも、学習速度は速くないが学習できている。シグモイド関数で Xavier の初期値の場合、学習速度は遅いが過学習なく学習できている。ReLU 関数で He の初期値の場合、学習速度も速く正解率も高い。
- オリジナルに対して、シグモイド関数に He、ReLU 関数に Xavier の初期値を設定 ⇒ ほとんどオリジナルと変わらず学習できている。実装上は、両方を試してみるのも良いと思われる。
- 中間層を 3 層（ノード数：40/20/20）に増やした場合 ⇒ 初期値が正規分布の場合はシグモイド関数、ReLU 関数のいずれも学習が全く進んでいない。シグモイド関数で Xavier の初期値の場合、ReLU 関数で He の初期値の場合共に、中間層 2 層の場合と傾向は同じ。中間層を 3 層にする効果はあまりないように考えられる。

# 深層学習（day2）・要点まとめ

## Section2：学習率最適化手法

### ▣ 学習率最適化手法

学習率の決め方： ①初期の学習率を大きく設定し、徐々に学習率を小さくしていく； ②パラメータ毎に学習率を可変させる

《確認テスト》： モメンタム・AdaGrad・RMSProp の特徴

Momentum＝以前に適用した勾配の方向を慣性項により現在のパラメータ更新にも影響させる

AdaGrad＝勾配を 2 乗した値を蓄積し既に大きく更新されたパラメータほど学習率を小さくする

RMSProp＝更新量が飽和した重みは更新されない AdaGrad の欠点を指数移動平均の蓄積で解決

Adam＝更に勾配の平均と分散をオンラインで推定し利用

| 最適化手法 | 内容説明 | 数式とコード |
|---|---|---|
| **モメンタム** | 誤差をパラメータで微分したものと学習率の積を減算した後、現在の重みに前回の重みを減算した値と慣性の積を加算<br>**メリット**＝①局所的最適解にならず、大域的最適解となる；②谷間に到達後、最適解に行くまでの時間が速い | $V_t = \mu V_{t-1} - \varepsilon \nabla E$ （$\mu$：慣性）<br>`v[key] = momentum * v[key] - learning_rate * grad[key]`<br>$W^{(t+1)} = W^{(t)} + V_t$<br>`network.params[key] += v[key]` |
| **AdaGrad** | 誤差をパラメータで微分したものと再定義した学習率の積を減算<br>**メリット**＝勾配の緩やかな斜面に対して、最適解に近づける<br>**課題**＝学習率が徐々に小さくなるので、鞍点問題を引き起こすことがある | $h_0 = \theta$<br>`h[key] = np.zeros_like(network.params[key])`<br>$h_t = h_{t-1} + (\nabla E)^2$<br>`h[key] += grad[key] * grad[key]`<br>$W^{(t+1)} = W^{(t)} - \varepsilon \frac{1}{\sqrt{h_t} + \theta} \nabla E$<br>`network.params[key] -= learning_rate * grad[key] / (np.sqrt(h[key]) + 1e-7)` |
| **RMSProp** | 誤差をパラメータで微分したものと再定義した学習率の積を減算<br>**メリット**＝①局所的最適解にならず、大域的最適解となる； ②ハイパーパラメータの調整が必要な場合が少ない | $h_t = \alpha h_{t-1} + (1-\alpha)(\nabla E)^2$<br>`h[key] *= decay_rate`<br>`h[key] += (1 - decay_rate) * np.square(grad[key])`<br>$W^{(t+1)} = W^{(t)} - \varepsilon \frac{1}{\sqrt{h_t} + \theta} \nabla E$<br>`network.params[key] -= learning_rate * grad[key] / (np.sqrt(h[key]) + 1e-7)` |
| **Adam** | **メリット**＝①モメンタムの、過去の勾配指数関数的減衰平均； ②RMSProp の、過去の勾配の二乗の指数関数的減衰平均⇒2 手法のメリットを取り込んだ最適化アルゴリズム | |

**[学習率最適化手法**の実装の考察（2_4_optimizer.ipynb）】：

- モメンタム・・・A)学習率を 0.8 に大きくすると正答率が高くなった。モメンタム係数を 0.9 から 0.5

に下げると正答率が下がった。 B) 活性化関数のシグモイドを Xavier 初期値にすることで正答率が向上した。更に、活性化関数を ReLU＋He 初期値にすることで正答率が向上した。 C) バッチ正規化により、更に正答率が向上した。

- AdaGrad･･･A)学習率を 0.1 に大きくすると正答率が高くなった。 B) 活性化関数のシグモイドを Xavier 初期値にすることで正答率が向上した。更に、活性化関数を ReLU＋He 初期値にすることで正答率が向上した。 C) バッチ正規化により、更に正答率が向上した。
- RMSprop･･･A)学習率を 0.8 に大きくすると正答率が高くなった。 B) 活性化関数のシグモイドを Xavier 初期値にすることでやや正答率が向上した。更に、活性化関数を ReLU＋He 初期値にすることで明らかに正答率が向上した。 C) バッチ正規化により、更に正答率が向上した。
- Adam･･･A)学習率を 0.8 に大きくすると正答率が高くなった。 B) 活性化関数のシグモイドを Xavier 初期値にしても正答率に大きな変化はなかった。更に、活性化関数を ReLU＋He 初期値にすることで明らかに正答率が向上した。 C) バッチ正規化により、更に正答率が向上した。

⇒ 学習率やモメンタム係数もハイパーパラメータとして事前検討が必要。活性化関数の種類と初期値の選択も重要である。学習率最適化手法としては、RMSprop か Adam を選択しておけば問題ないように思われる。バッチ正規化による正則化の効果は大きいようである。

## Section3：過学習について

### ▣ 過学習

過学習とは、特定の訓練サンプルに対して特化して学習が進むことで、テスト誤差と訓練誤差とで学習曲線が乖離すること。原因は、ネットワークの自由度（層数、ノード数等）が高いため、1）パラメータの数が多い、2）パラメータの値が適切でない、3）ノードが多い等である。

### ▣ 正則化

- 過学習の原因＝学習により重みにばらつきが発生する。大きな重みは学習において重要な値であり、過学習が起こる。
- 過学習の解決策＝過学習が起こりそうな重みの大きさ以下で重みをコントロールし、かつ重みの大きさにばらつきを出す。（＝**荷重減衰**）

#### 3-1. L1 **正則化、**L2 **正則化**

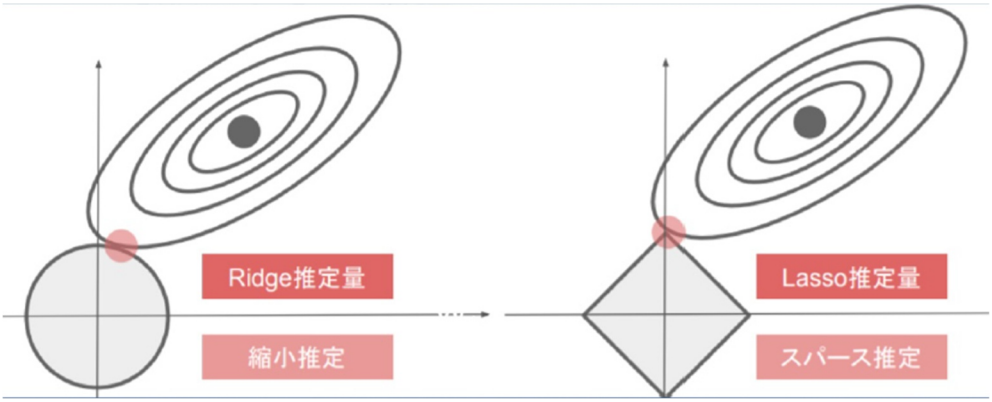
《確認テスト》： 線形モデルの正則化の中の、リッジ回帰の特徴として正しいものは？

⇒ (a)ハイパーパラメータを大きな値に設定すると、全ての重みが限りなく 0 に近づく･･･〇
(b)ハイパーパラメータを 0 に設定すると、非線形回帰となる･･･× ←元の線形回帰となる
(c)バイアス項についても正則化される･･･× ←重みが正則化され、バイアス項は正則化されない
(d)リッジ回帰の場合、隠れ層に対して正則化項を加える･･･× ←正則化の対象は誤差関数

《確認テスト》： L1 正則化を表しているグラフはどちらか？

⇒ L1 正則化は、重みの絶対値による荷重減衰であるため、その制約境界は下図の右側のようなひし形になる。従って、誤差関数の等高線が制約境界の端点と接することになり、重みが 0 となる重みが発生しスパース推定となる。

# 深層学習（day2）・要点まとめ




《例題チャレンジ》： L2 正則化、L1 正則化において、最終的な勾配を計算する式は？

1) L2 正則化 ⇒ grad += rate * param

(param：目標パラメータ、grad：パラメータの勾配、rate：リッジ係数)

L2 ノルムは$\|param\|^2$ なので、その勾配が誤差の勾配に加算される。従って、2*$param$ だが、係数 2 は正則化の係数に吸収されるため、勾配加算項は $param$ となる。

L2 正則化の誤差関数： $\boldsymbol{E_n(w)}+\frac{1}{2}\lambda\|\boldsymbol{x}\|^2$

2) L1 正則化 ⇒ x = sign(param) → grad += rate * x

(param：目標パラメータ、grad：パラメータの勾配、rate：ラッソ係数)

L1 ノルムは$|param|$なので、その勾配が誤差の勾配に加算される。従って、その勾配の値は、1 ($param>0$)、-1 ($param<0$)、0 ($param=0$) となるので、sign($param$)である。

L1 正則化の誤差関数： $\boldsymbol{E_n(w)}+\lambda|\boldsymbol{x}|$

《例題チャレンジ》： データ拡張における画像をランダムに切り取る処理に当てはまる式は？

⇒ image = image[top : bottom, left : right, : ] により、channel を保持しながら、画像を切り取る

## 3-2. ドロップアウト

### ▣ ドロップアウトとは

ノードの数が多いとの過学習の課題 ⇒ ランダムにノードを削除して学習させる

メリット＝データ量を変化させずに、異なるモデルを学習させている (アンサンブル学習の効果)

**【過学習解決手法の実装の考察 (2_5_overfiting.ipynb)】：**

- 荷重減衰 λ は、L2 正則化では 0.05 位の時に、L1 正則化では 0.003 位の時に正答率が最も高くなった。 ⇒ λ は小さくし過ぎると荷重減衰が働かず過学習が起こる。逆に、大きくし過ぎると正答率が上がらず学習が進まない。
- ドロップアウト比は、L2 正則化では 0.15 位の時に、L1 正則化では 0.07 位の時に正答率が最も高くなった。共に、大きくし過ぎると (0.5) 正答率が上がらず学習が進まない。また、学習率最適化手法を変えた場合、正答率は Adam＞モメンタム＞AdaGrad の順になった。 ⇒ モメンタムや AdaGrad はモデルによって出来不出来が左右されるようだが、Adam は概してどのようなモデルでも良い結果となることが多いようである。

# 深層学習（day2）・要点まとめ

## Section4：畳み込みニューラルネットワーク（CNN）

### ▣ プーリング層

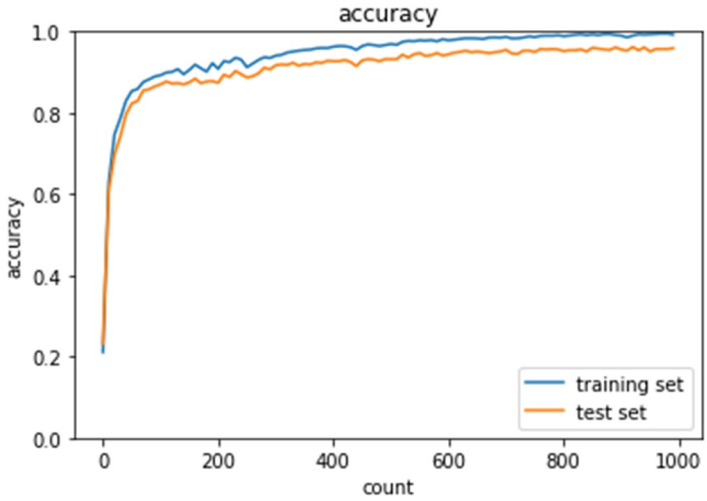
《確認テスト》： サイズ 6×6 の入力画像を、サイズ 2×2 のフィルターで畳み込んだ時の出力画像のサイズは（ストライドとパディングは 1）？

⇒ 出力画像のサイズ＝（入力画像サイズ＋ストライド×2－フィルタサイズ）／ストライド+1
＝（6＋1×2－2）／1＋1＝7

【畳み込み層プログラムの実装の考察（2_6_simple_convolution_network.ipynb）】：

- CNN では、畳み込みやプーリングでの numpy の行列計算の効率を考慮して、畳み込みやプーリングでは、元の 2 次元の画像データを、フィルターの大きさに合わせた 1 次元データを畳み込みの順に配列にする im2col 関数で変換してから順伝搬を実行する。逆に、逆伝播では求めた勾配データを col2im 関数で 2 次元データに戻す。
- SimpleConvNet：入力データ（28×28=784）、畳み込み層（フィルター：数 30、サイズ 5）、プーリング層、全結合層×2 で mnist の手書き数字を認識

```
-----------------------------------------------------------------------
Generation: 950. 正答率(トレーニング) = 0.9926 : 950. 正答率(テスト) = 0.961
Generation: 960. 正答率(トレーニング) = 0.9934 : 960. 正答率(テスト) = 0.951
Generation: 970. 正答率(トレーニング) = 0.9942 : 970. 正答率(テスト) = 0.957
Generation: 980. 正答率(トレーニング) = 0.995 : 980. 正答率(テスト) = 0.957
Generation: 990. 正答率(トレーニング) = 0.995 : 990. 正答率(テスト) = 0.957
Generation: 1000. 正答率(トレーニング) = 0.9928 : 1000. 正答率(テスト) = 0.959
```




⇒ オリジナルの im2col は処理イメージが判りにくかったため、自分なりに理解し易いように im2col_simple を作成し、結果が同じになることが確かめられた。オリジナルの実装は im2col_simple で行っているフィルター移動分の2重ループをせずに効率化するため、ストライド幅刻みでスライスすることで、フィルターを移動させて取得する分をまとめて取得・コピーしている。