実装演習レポート【深層学習_前編】 Section1 入力層～中間層

要点のまとめ

- 入力層ではデータを入力するところ。入力を受ける部分をノードという。
  入力されたデータをどのくらい使うべきかということを重みwで表現する。
  重要度が高いものは重みが大きい。
  入力全体をずらしたいときはバイアスb（一次関数でいう切片）を使用して、学習する。
  重みｗとバイアスbを上手く決めることが大事。
  重みとバイアスを計算して中間層に入力するものを総入力という。

入力層から中間層の計算式

重み

$$\mathbf{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_I \end{bmatrix}$$

入力

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_I \end{bmatrix}$$

総入力

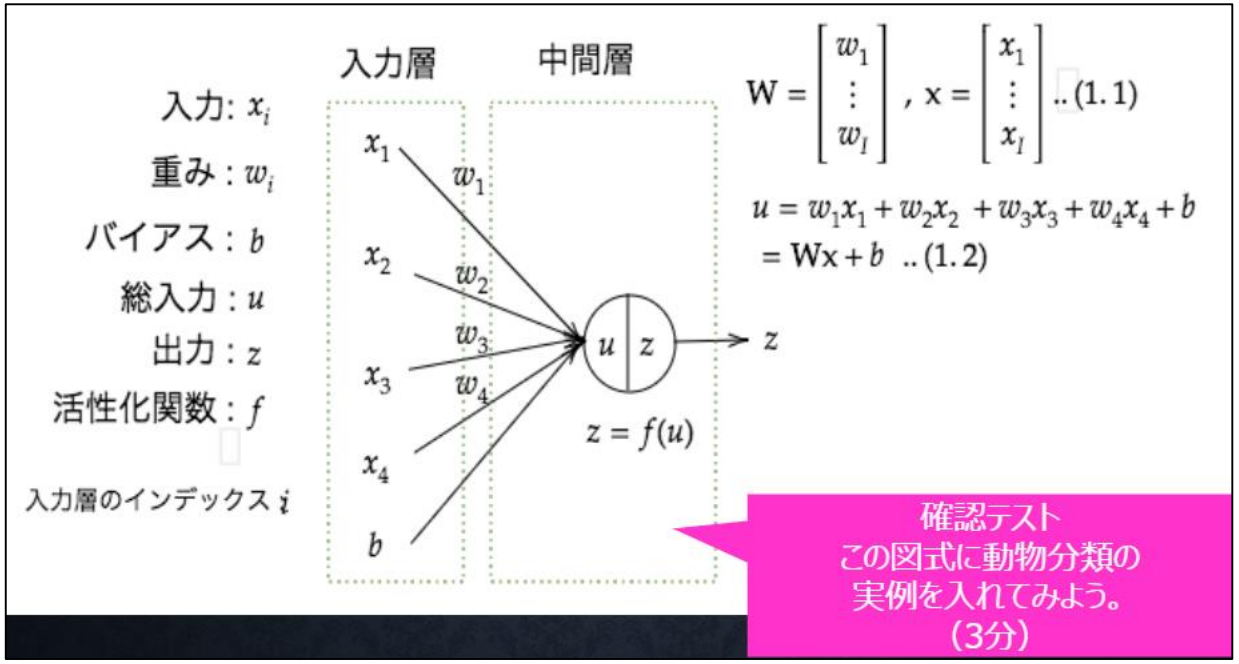
$$u = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + b$$

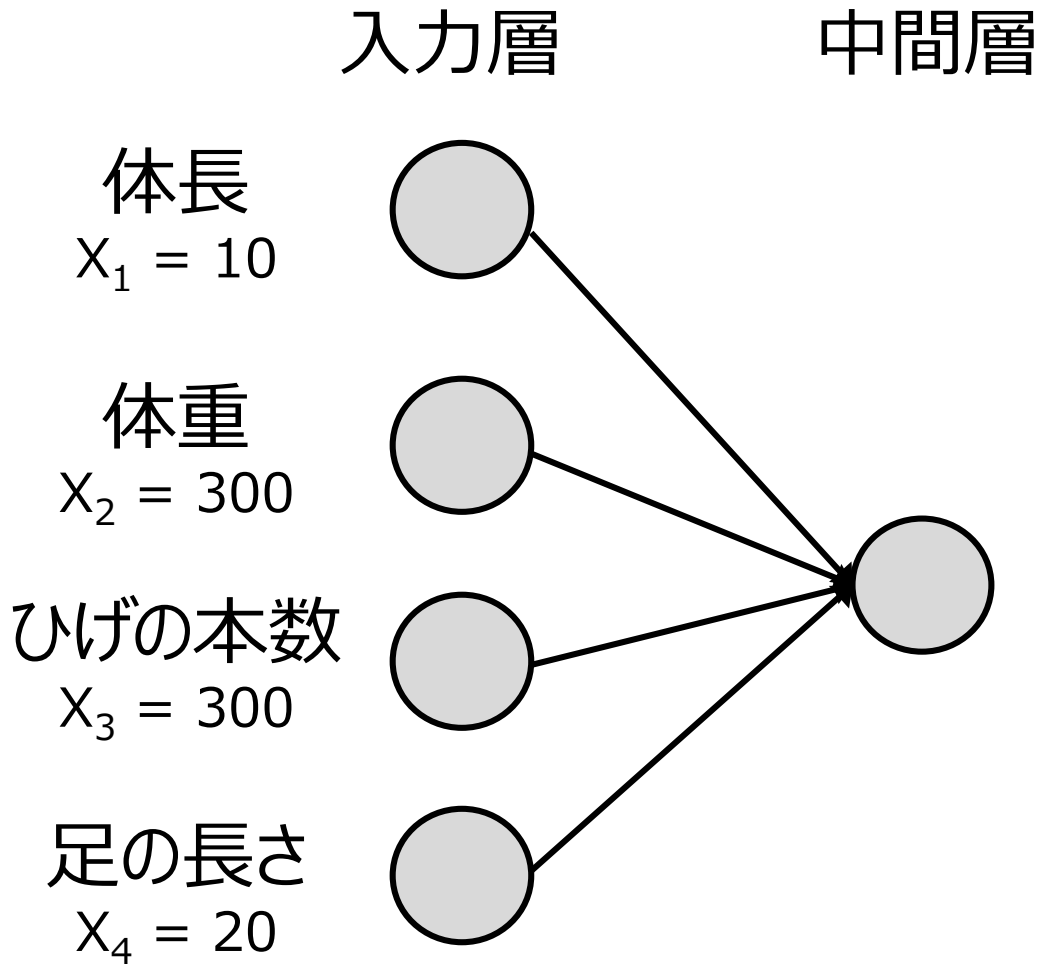
$$= \mathbf{W}\mathbf{x} + b$$

バイアス

実装演習レポート【深層学習_前編】 Section1 入力層～中間層

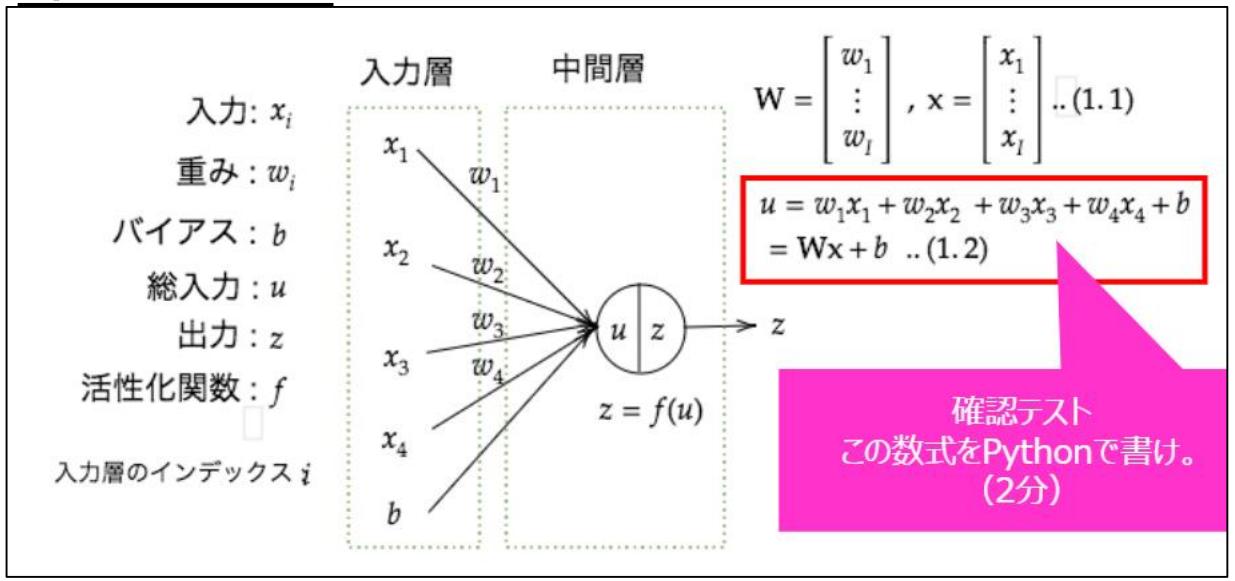
確認テスト





実装演習レポート【深層学習_前編】 Section1 入力層～中間層

確認テスト

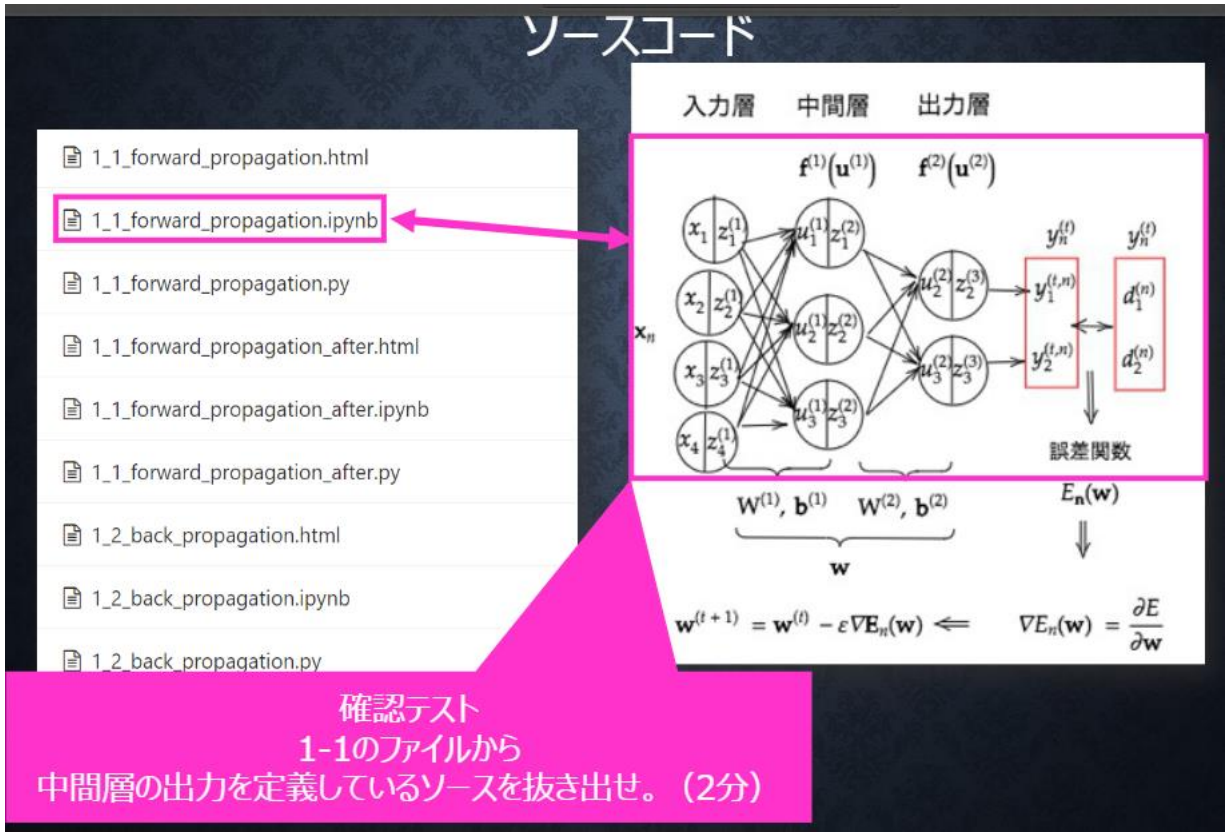



```
u = np.dot(x,W) + b
```

行列同士の内積計算は
np.dotを使用する。

実装演習レポート【深層学習_前編】 Section1 入力層～中間層

確認テスト



中間層の出力定義

```
# 2層の総出力
z2 = functions.relu(u2)
```

実装演習レポート【深層学習_前編】 Section1 入力層～中間層

参考図書レポート

ニューラルネットワークは、３種類のユニット層（layer）から成り立つ。
最初の部分にあるのは入力層で、個々のノードに観測値の個々の
特徴量が与えられる。
例えば、観測値に100の特徴があるなら、入力層には100のノードが必要。
中間層は、入力層への入力を連続的に変換し、出力する。
この変換は入力値に対して、パラメータであるw（重み）をかけて
b（バイアス）を足し合わせることで実施される。

実装演習レポート【深層学習_前編】 Section2 活性化関数

要点のまとめ

・活性化関数には中間層用と出力層用の２種類ある。

中間層用の活性化関数

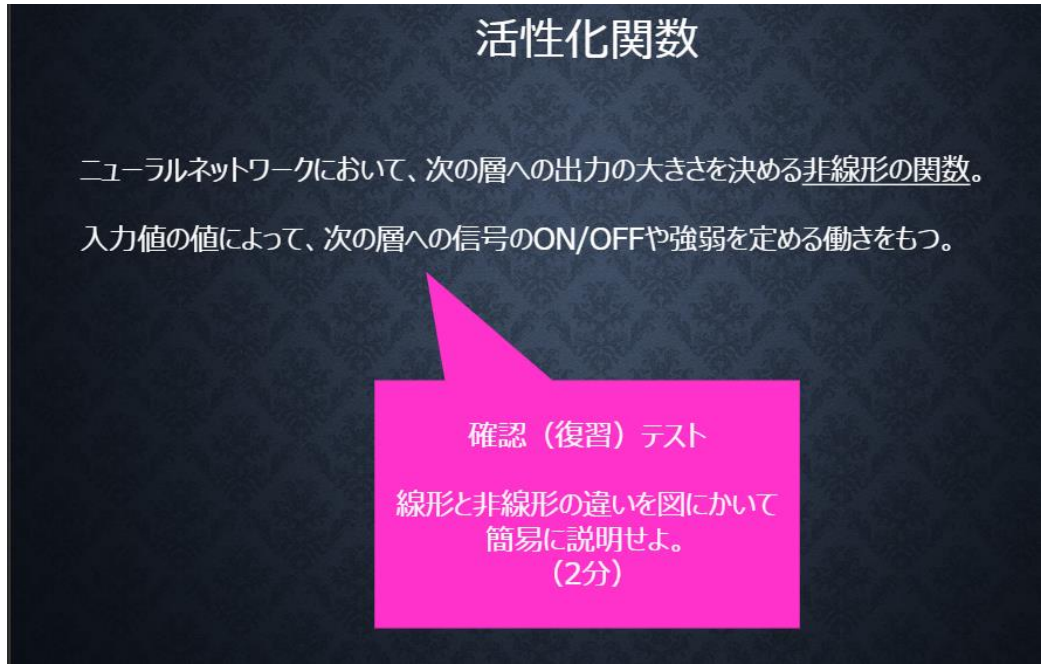
① ステップ関数 ： 閾値を超えたら発火する関数であり、出力は常に1か0。
現在はあまり使用されていない。

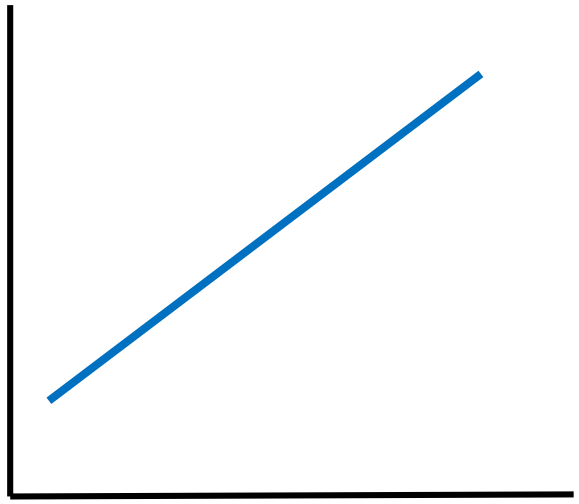
② シグモイド関数 ： 0～1の間を緩やかに変化する関数。信号の強弱を伝えられるようになりニューラルネットワーク普及のきっかけとなった。
但し、勾配消失問題を引き起こすことが課題。

③ RELU関数 ： 今、最も使われている活性化関数。
勾配消失問題の回避とスパース化に貢献することで良い成果を出す。

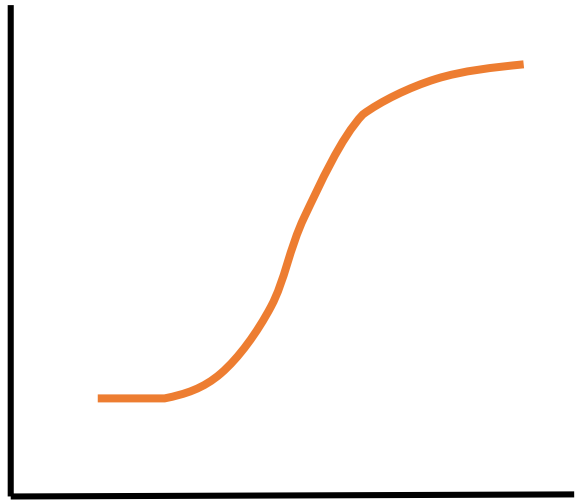
実装演習レポート【深層学習_前編】 Section2 活性化関数

確認テスト





線形な関数



非線形な関数

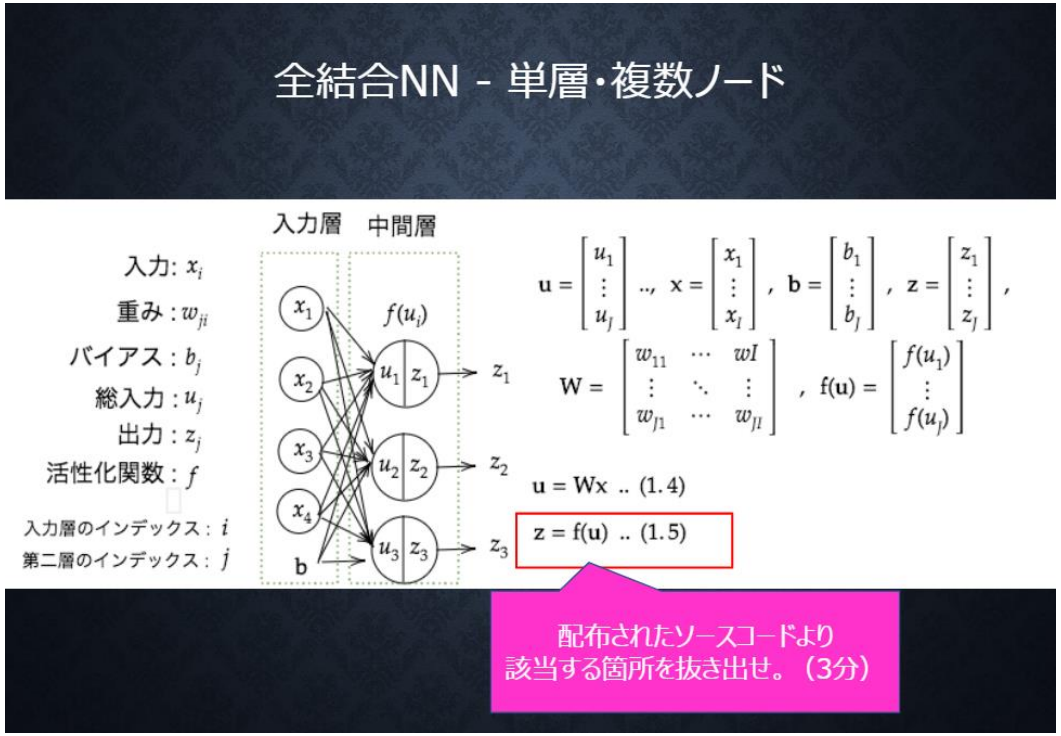
線形な関数は加法性・斉次性を満たすが、
非線形な関数は加法性・斉次性を満たさない。

加法性：$f(x+y) = f(x) + f(y)$

斉次性：$f(kx) = kf(x)$

実装演習レポート【深層学習_前編】 Section2 活性化関数

確認テスト



活性化関数

```
# 1層の総出力
z1 = functions.relu(u1)
```

```
# 2層の総出力
z2 = functions.relu(u2)
```

RELU関数を定義
具体的な計算式はfunctionsに
記述してある。

実装演習レポート【深層学習_前編】 Section2 活性化関数

参考図書レポート

活性化関数は非線形関数の為、モデルの表現力を増すために使用される。
特に、中間層では勾配消失問題を防ぐことができる非線形関数が望ましい。
シグモイド関数では、勾配消失問題が起こってしまうため、現在では
勾配消失問題が発生しにくいRELU関数が使われている。

※シグモイド関数では微分したときの最大値が0.25（＜1）の為、
連鎖率の原理で微分が掛け算されると誤差がどんどん小さくなって、
勾配消失問題が発生する。
これに対して、RELU関数での微分はxが0より大きいときでは、
常に1となるので、勾配消失問題が発生しにくくなる。

実装演習レポート【深層学習_前編】 Section3 出力層

要点のまとめ

・出力層の役割は入力層、中間層から受け取ったデータを自分たちが
求める情報を出力する。
入力層、中間層、出力層でニューラルネットワークを構成、これを学習していく。

ニューラルネットワーク全体を学習するときは、訓練データ(入力データと
正解値のセット）を準備する。
ニューラルネットワークから出力した結果と正解値を照らし合わせて、
どのくらい合っているかを 計算する。数値で示すために誤差関数を使用する。

誤差関数には二乗和誤差を用いる。

$$E_n(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{J}(y_j - d_j)^2 = \frac{1}{2}\|(\mathbf{y}-\mathbf{d})\|^2$$

コード：loss = functions.mean_squared_error(d,y)

実装演習レポート【深層学習_前編】 Section3 出力層

要点のまとめ

・出力層の活性化関数
出力層と中間層で利用される活性化関数が異なる。

【出力層に使われる活性化関数の種類】
・ソフトマックス関数　：　多クラス分類に使用する。
・恒等写像 :回帰問題に使用する
・シグモイド関数 ： 二値分類に使用する

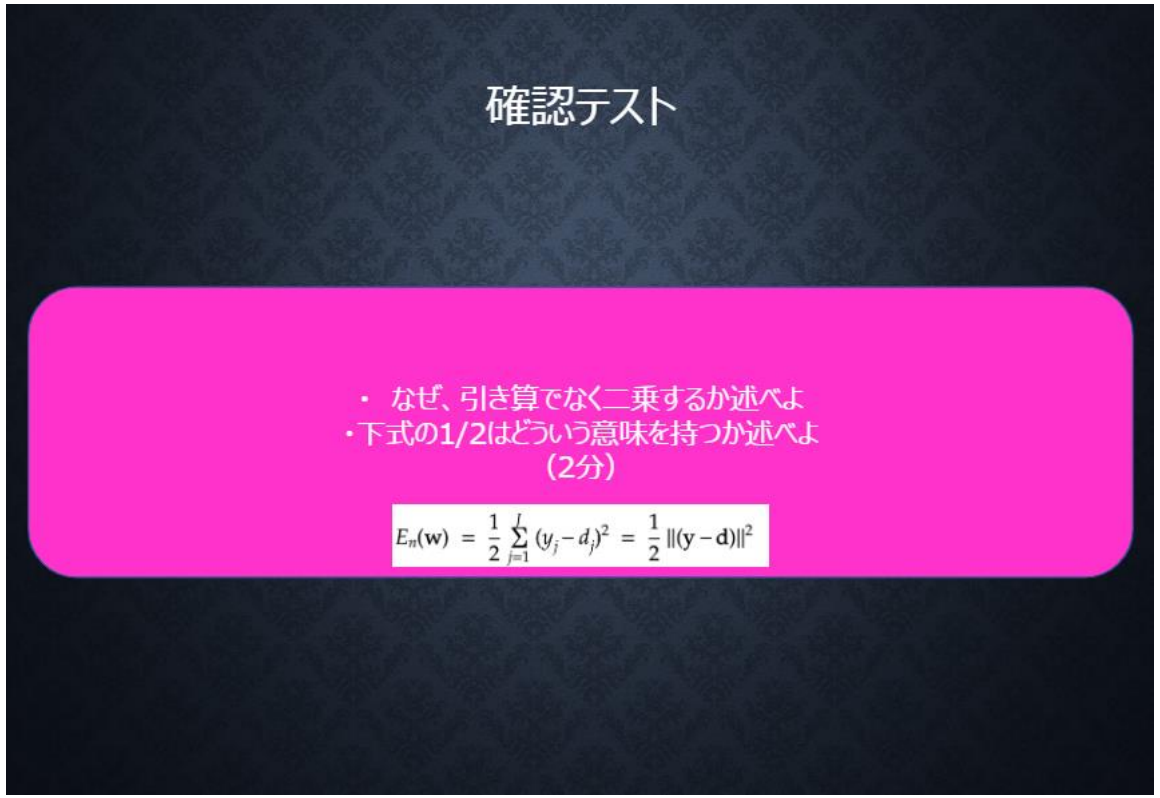
・誤差関数は回帰では二乗誤差、分類では交差エントロピーを使用する。

$$E_n(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{I}(\mathbf{y}_n - \mathbf{d}_n)^2 \quad .. \text{ 二乗誤差}$$

$$E_n(\mathbf{w}) = -\sum_{i=1}^{I} d_i \log y_i \quad .. \text{ 交差エントロピー}$$

実装演習レポート【深層学習_前編】 Section3 出力層

確認テスト



・ なぜ、引き算でなく二乗するか述べよ。

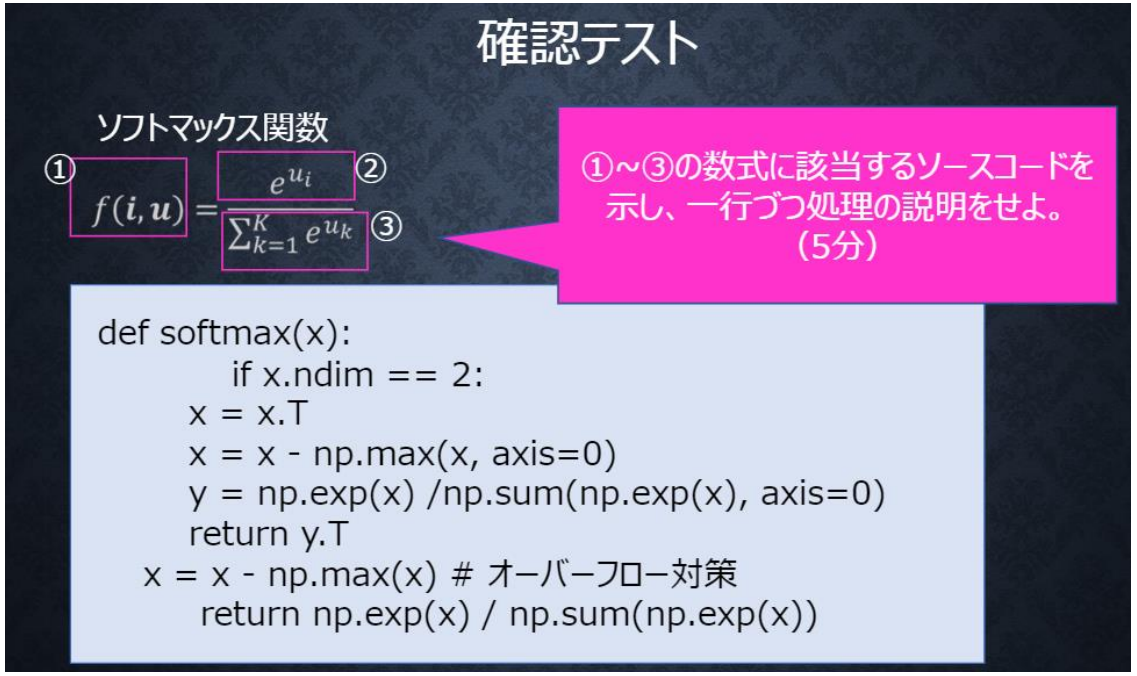
引き算のみだと正の値のみでなく
負の値をとる場合もあり、全体の誤差を
正しく捉えることができない。
これを防ぐために、2乗して全ての値を
正の値にする。

・ 下式の1/2はどういう意味を持つか述べよ。

計算の便宜上で1/2を使う。
誤差関数を微分したときの数式を
簡単に示すために使用される。

実装演習レポート【深層学習_前編】 Section3 出力層

確認テスト



- ①~③の数式に該当するソースコードを示し、一行づつ処理の説明せよ

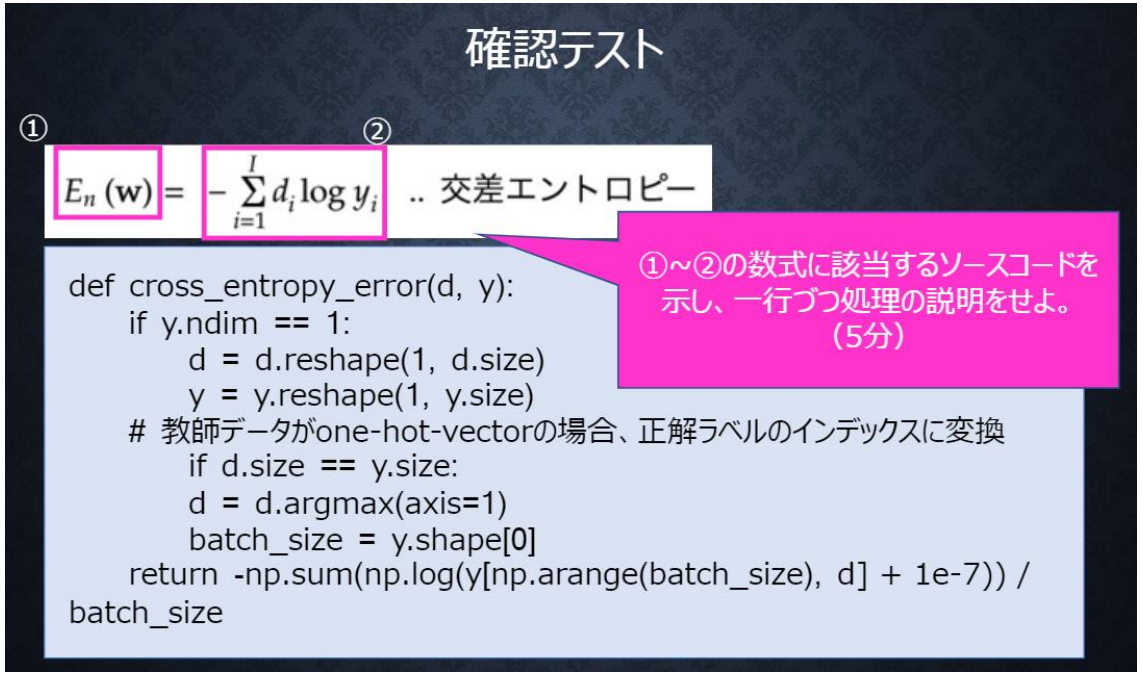
①：def softmax(x)
ソフトマックス関数の出力値を示す。

②：np.exp(x)
入力値xの指数関数

③：np.sum(np.exp(x))
全ての入力値xの指数関数の和を算出する。

実装演習レポート【深層学習_前編】 Section3 出力層

確認テスト



- ①~②数式に該当するソースコードを示し、一行づつ処理の説明せよ

①：cross_entropy_error(d,y):
交差エントロピーの出力値を示す。

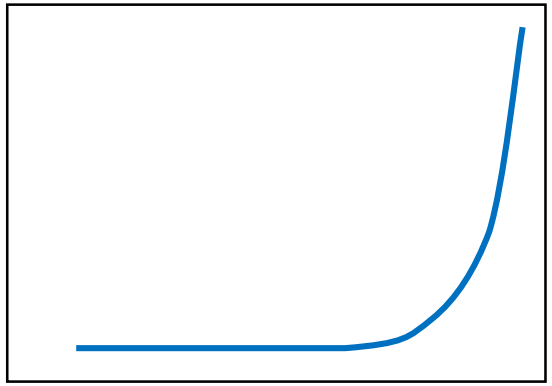
②：-np.sum(np.log(y[np.arange(batch_size),d] + 1e-7))
入力値yの指数関数と正解値dを掛け合わせた値の合計値に-1を掛ける。

実装演習レポート【深層学習_前編】 Section3 出力層

参考図書レポート

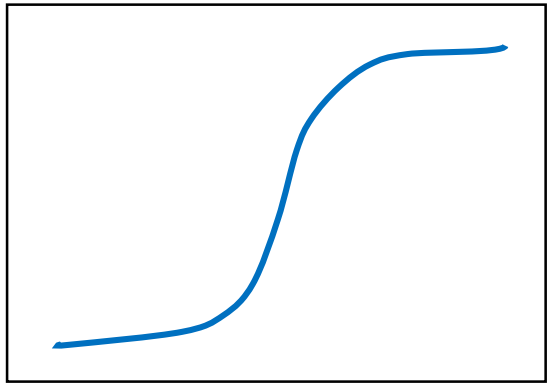
【出力層に使われる活性化関数の種類】

ソフトマックス関数



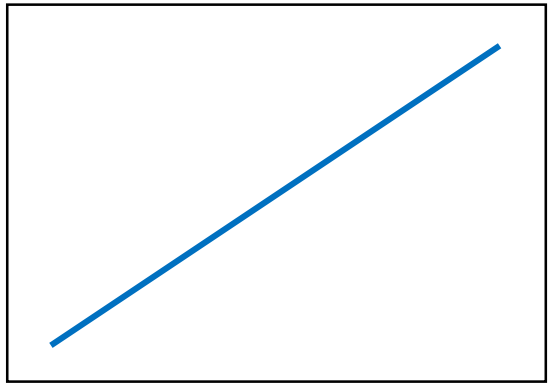
- 出力値の合計が1.0
- 各出力値の範囲は0.0~1.0
- 多クラス分類に使用

シグモイド関数



- 入力値が大きいほど1に近づき、入力値が小さいほど0に近づくように変換して出力
- 2クラス分類に使用

恒等関数



- 入力した値をそのまま出力値に置き換え
- 回帰問題に使用

実装演習レポート【深層学習_前編】 Section4 勾配降下法

要点のまとめ

・深層学習の目的は学習を通して誤差を最小にするネットワークを作成すること
　⇒ 誤差を最小化するためにパラメータを最適化させる。
　　パラメータを最適化する為に勾配降下法を利用する。

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \varepsilon \nabla E$$

ε ：学習率

学習を効率に進めることができる。
但し、適切な値を指定することが重要。

・勾配降下法には下記の３種類がある。

① 勾配降下法

② 確率的勾配降下法

③ ミニバッチ勾配降下法

実装演習レポート【深層学習_前編】 Section4 勾配降下法

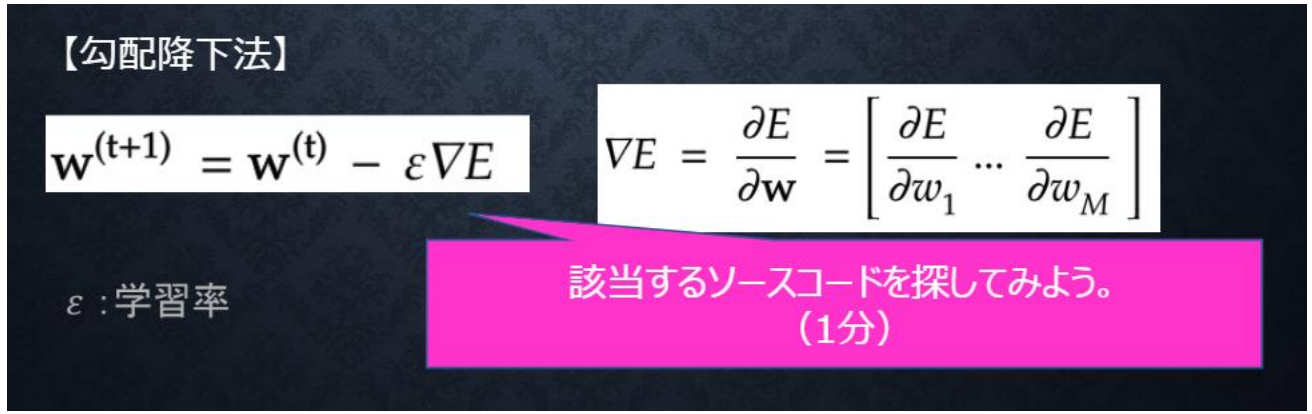
要点のまとめ

・確率的勾配降下法（SGD）
全データから一部分のデータを使用して学習する方法。エポック毎で使用するデータが異なる。
【メリット】
・計算コスト低減　・望まない局所極小解に収束するリスク低減
・オンライン学習ができる（モデルに少しづつデータを与えて学習を進める）

・ミニバッチ勾配降下法
ランダムに分割したデータを使用して学習する方法。
この方法が一般的に使用される。
【メリット】
・確率的勾配降下法のメリットを損なわず、計算機を有効利用できる。
CPUを利用したスレッドの並列化やGPUを利用したSIMD並列化。

実装演習レポート【深層学習_前編】 Section4 勾配降下法

確認テスト



$$w^{(t+1)} = w^{(t)} - \varepsilon \nabla E$$

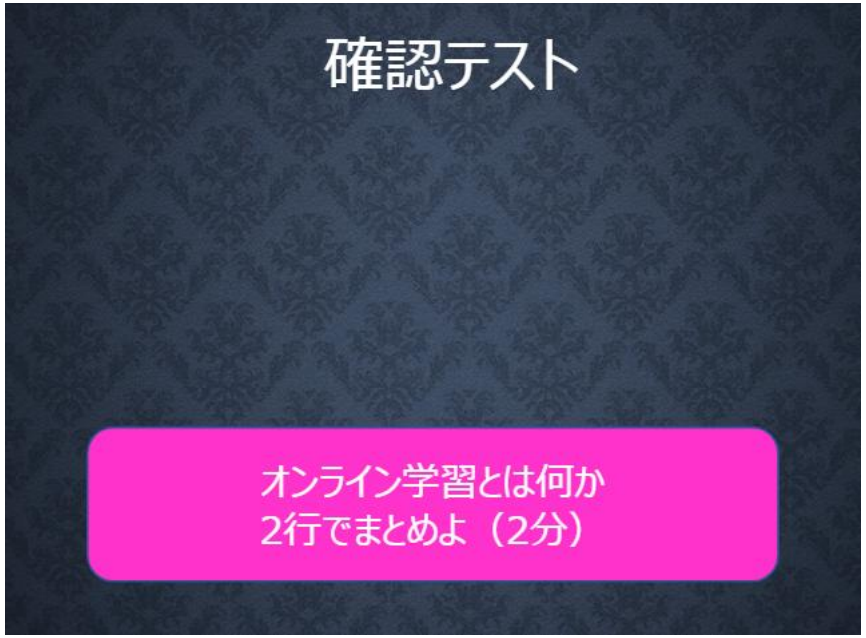
```
Network[key] -= learning_rate * grad[key]
```

$$\nabla E = \frac{\partial E}{\partial w} = \left(\frac{\partial E}{\partial w_1} \cdots \frac{\partial E}{\partial w_M} \right)$$

```
grad = backward(x, d, z1, y)
```

実装演習レポート【深層学習_前編】 Section4 勾配降下法

確認テスト

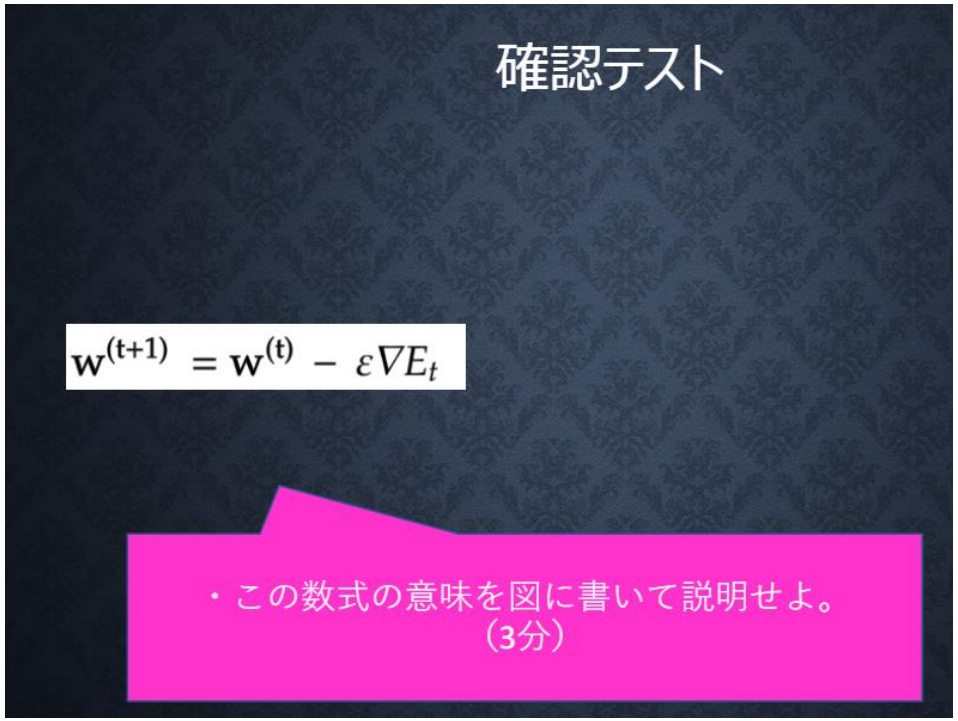


オンライン学習とは何か

オンライン学習とは、データを少しづつ与えながらパラメータを更新し学習を進めること。
一方、始めから全てのデータでパラメータを更新し学習を進めることをバッチ学習という。

実装演習レポート【深層学習_前編】 Section4 勾配降下法

確認テスト



この数式の意味を図に書いて説明せよ。

エポック　t　t+1　t+2

重み　w^t → w^{t+1} → w^{t+2}

$-\varepsilon \nabla E_t$　$-\varepsilon \nabla E_{t+1}$

既存の重みから学習率と誤差勾配を掛けた値を引いた値を算出。この算出した値に重みを更新。これを各エポックで実施し、繰り返して重みを更新していく。

実装演習レポート【深層学習_前編】 Section4 勾配降下法

参考図書レポート

勾配降下法の比較表

手法	利用データ	計算時間	メリット	デメリット
バッチ勾配降下法	全てのデータを利用	大	・解への到達が速いことが多い ・解が安定する	・メモリの使用量が多い ・局所解にはまりやすい
ミニバッチ勾配降下法	一部のデータを利用	中	・バッチ勾配降下法と確率的勾配降下法のそれぞれのメリットがある。	・バッチ勾配降下法と確率的勾配降下法のそれぞれのデメリットがある。
確率的勾配降下法（SGD）	1つのデータを利用	小	・メモリの使用量が少ない ・オンライン学習が可能 ・局所解を回避できる可能性がある	・解への到達が遅いことがある ・外れ値の影響を受ける。

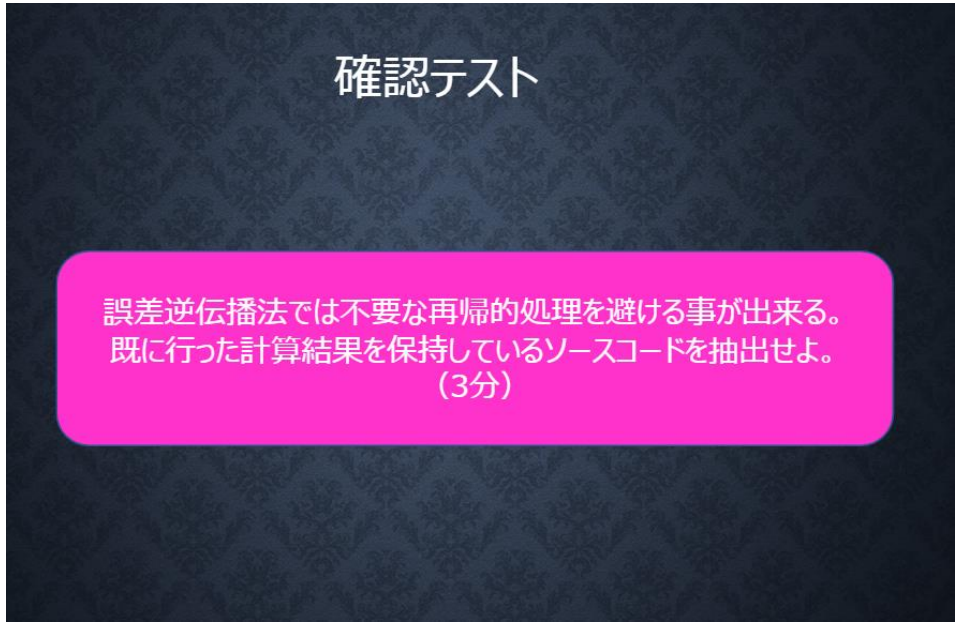
実装演習レポート【深層学習_前編】 Section5 誤差逆伝播法

要点のまとめ

- 誤差勾配の計算で、数値微分を使用すると計算を繰り返す為、負荷が大。その為、誤差逆伝播法を活用する。
- 誤差逆伝播法とは算出された誤差を、出力層側から順に微分し、前の層前の層へと伝播。最小限の計算で各パラメータでの微分値を解析的に計算する手法。
- 計算結果（＝誤差）から微分を逆算することで、不要な再帰的計算を避けて微分を算出できる。

実装演習レポート【深層学習_前編】 Section5 誤差逆伝播法

確認テスト



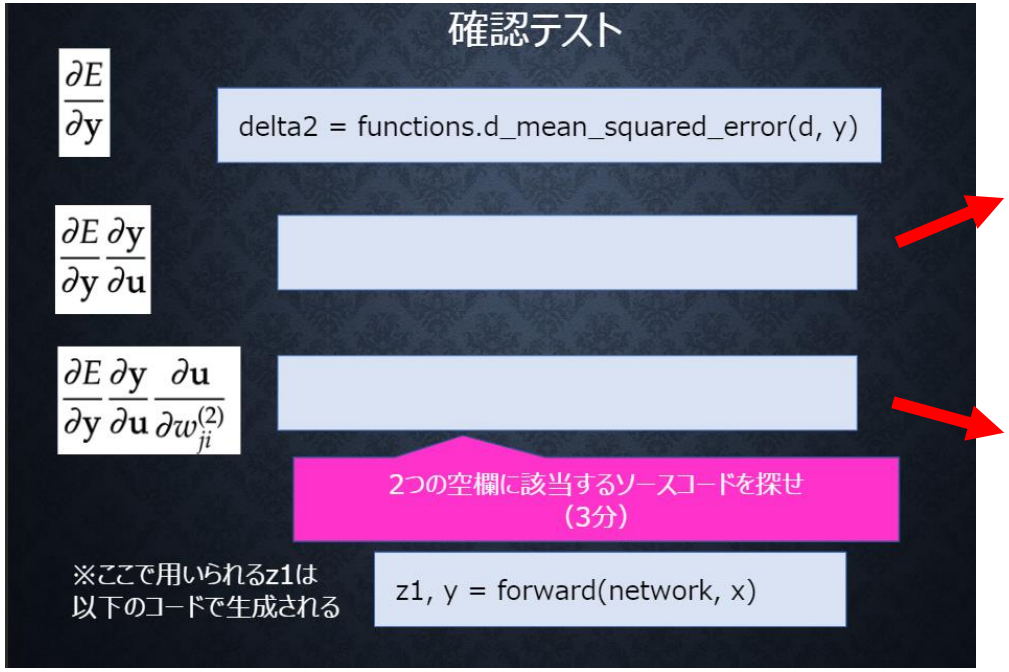
既に行った計算結果処理を保持しているソースコードを抽出せよ。

```
#出力層のデルタ
Delta2 = function.d_mean_squared_error(d,y)

ここで計算した微分値が前の層での微分計算にも使用される。
##試してみよう
delta1 = np.dot(delta2,W2,T) * function.d_sigmoid(z1)
このようにすることで、計算負荷を抑えることができる。
```

実装演習レポート【深層学習_前編】 Section5 誤差逆伝播法

確認テスト



2つの空欄に該当するソースコードを探せ。

```
delta1 = np.dot(delta2,W2.T) *
                functions.d_sigmoid(z1)
```

```
grad['W1'] = np.dot(x.T , delta1)
```

実装演習レポート【深層学習_前編】 Section5 誤差逆伝播法

参考図書レポート

誤差逆伝播法とは正解値と出力値の誤差から、重みやバイアスを修正していく学習方法。精度を向上させる。
この計算をするときには、重みやバイアスの変化による誤差の変化量が大事。
その為、下記に示す微分計算事項が重要となる。

偏微分：1つの変数に着目し、他は定数として扱う

連鎖律：$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial u}\frac{\partial u}{\partial x}$

シグモイド関数の微分：$\sigma(x)' = (1-\sigma(x))\sigma(x)$

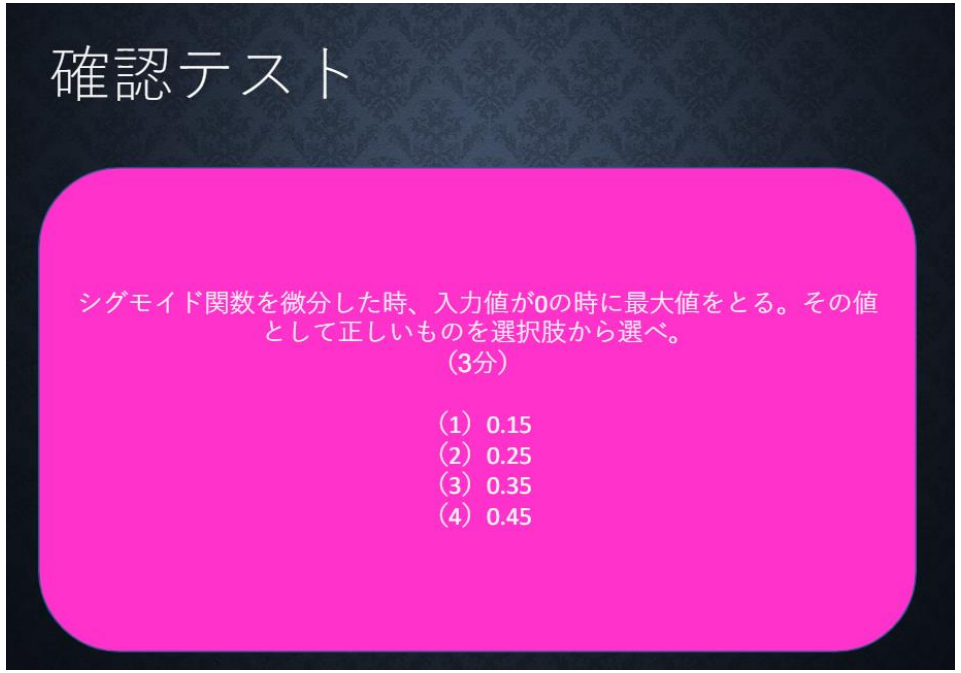
実装演習レポート【深層学習_前編】 Section1 勾配消失問題

要点のまとめ

- 勾配消失問題とは、学習が上手くいかなくなる原因の一つ。
  中間層が増えていくと、徐々に逆伝播で伝搬される情報が少なくなり、
  重みが更新されなくなり、学習が上手くいかなくなる。
- シグモイド関数は勾配消失問題を起こしやすい。
  （微分値が最大でも0.25と、1より小さい）
- 勾配消失問題の解決方法は以下の３つになる。
  ① 活性化関数の選択
  　　勾配消失問題を解消されるReLU関数を使用する。(微分値が0か1)
  ② 重みの初期値設定
  　　XavierやHeで重みの要素を前のノード数を用いて計算したものを設定
  ③ バッチ正規化
  　　ミニバッチ単位で、入力前のデータの偏りを抑制する手法

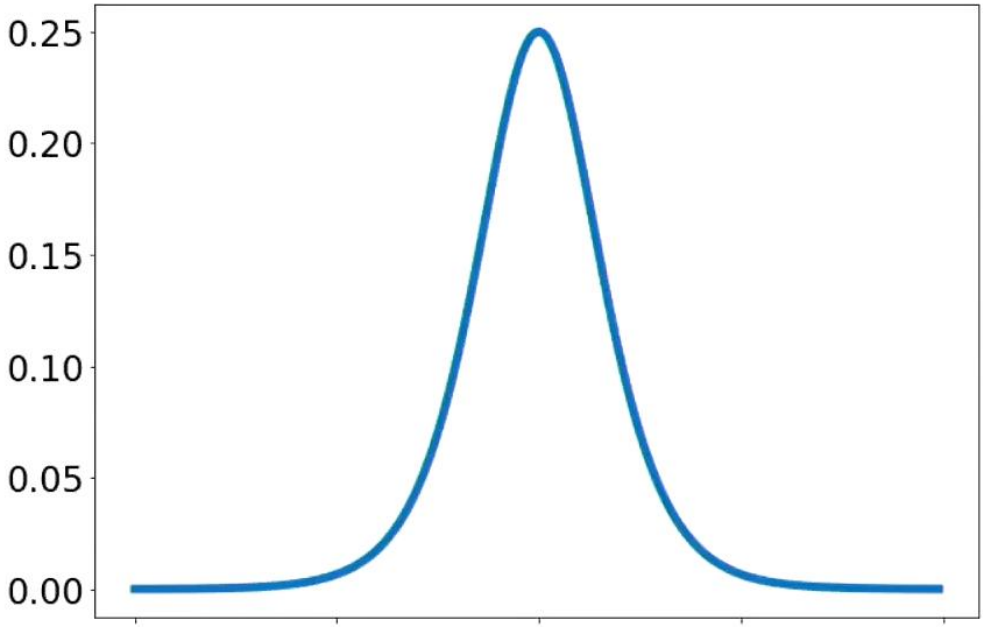
実装演習レポート【深層学習_前編】 Section1 勾配消失問題

確認テスト



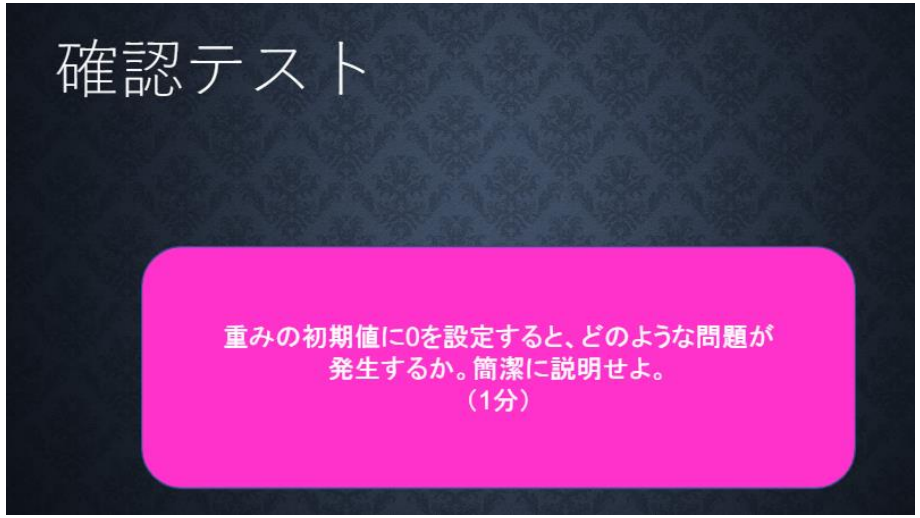
シグモイド関数を微分した時、入力値が0の時に最大値をとる。
その値として正しいものを選択肢から選べ。

(2) 0.25



実装演習レポート【深層学習_前編】 Section1 勾配消失問題

確認テスト

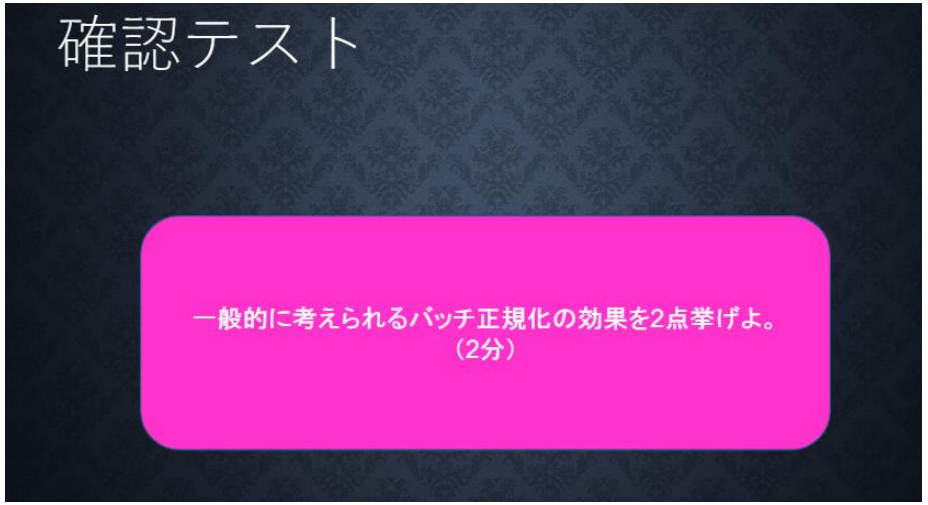


重みの初期値に0を設定すると、どのような問題が発生するか、簡潔に説明せよ。

全ての重みの値が均一に更新される為、
多数の重みを持つ意味がなくなる。
その為、正しい学習ができない。

実装演習レポート【深層学習_前編】 Section1 勾配消失問題

確認テスト



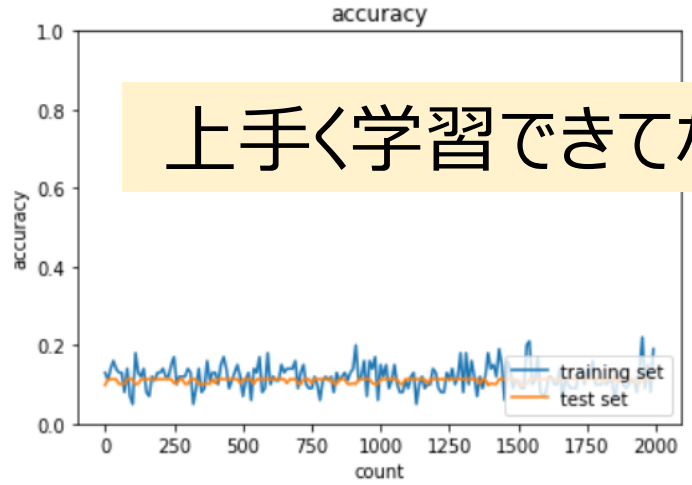
一般的に考えられるバッチ正規化の効果を
２点挙げよ。

- 中間層の重みの更新が安定し、
  計算が速くなる。
- 学習データの極端なバラツキがなくなり、
  過学習が起きにくくなる。

実装演習レポート【深層学習_前編】 Section1 勾配消失問題

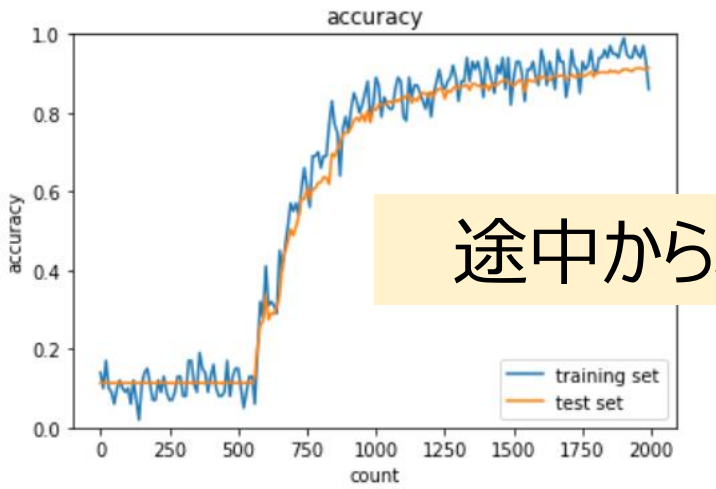
実装演習（2_2_2_vanishing_gradient_modified.ipynb）

- sigmoid - gauss



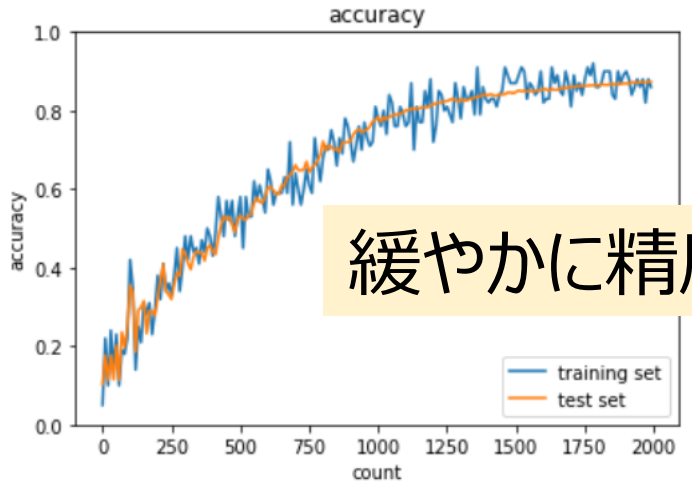
上手く学習できてない

- ReLU - gauss



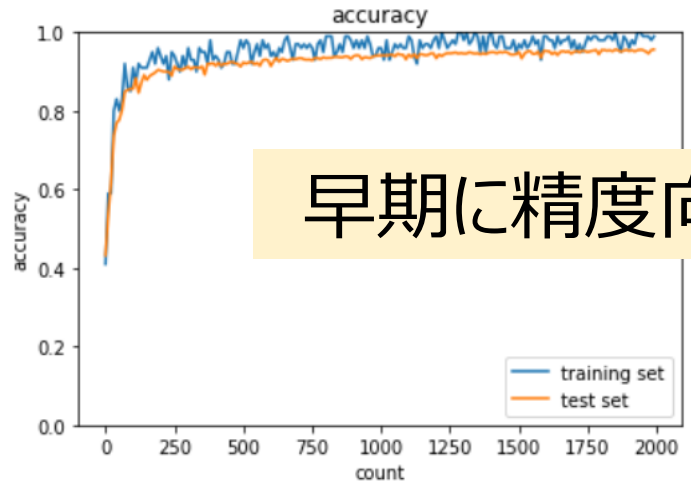
途中から精度向上

- sigmoid - Xavier



緩やかに精度向上

- ReLU - He



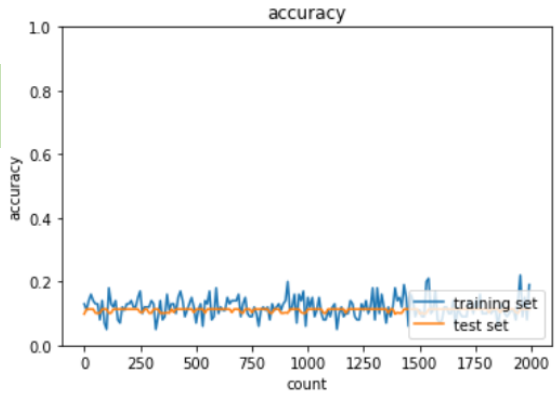
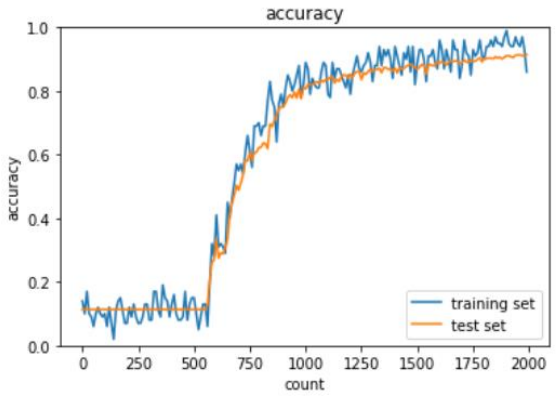
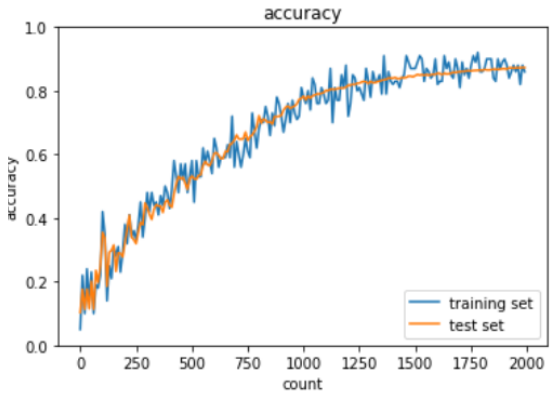
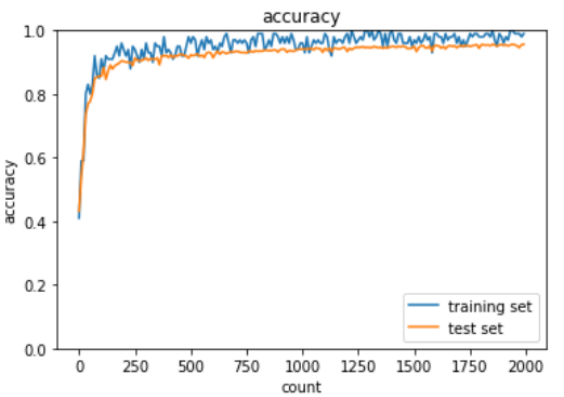
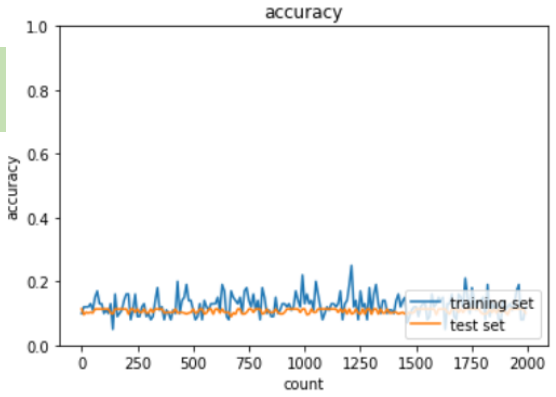
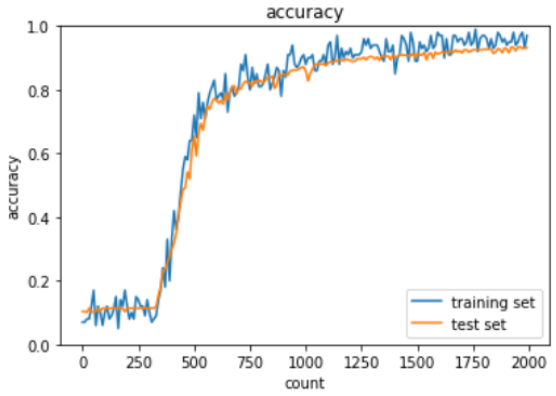
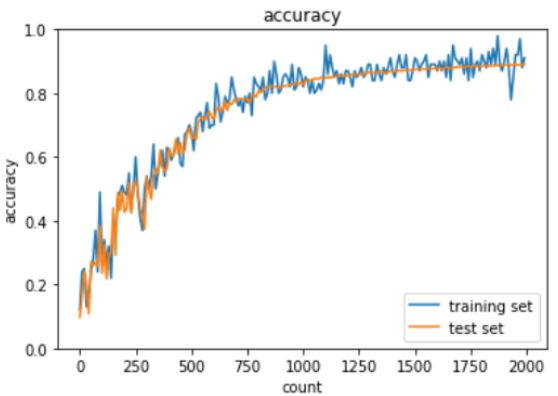
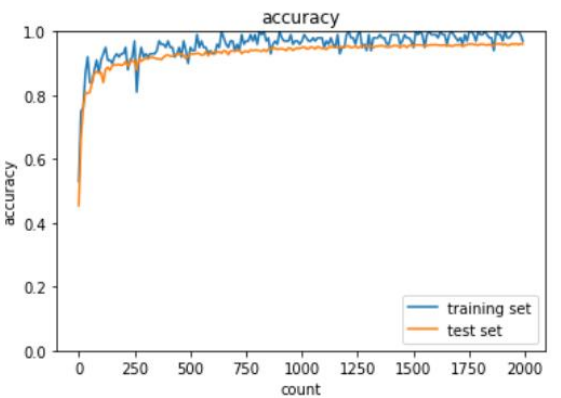
早期に精度向上

実装演習レポート【深層学習_前編】 Section1 勾配消失問題

実装演習（2_2_2_vanishing_gradient_modified.ipynb）

- [try] hidden_size_listの数字を変更してみよう
  （隠れ層のノード数のリスト）

[40,20]から[80,40]に変更。
若干、早期に精度が向上する傾向。

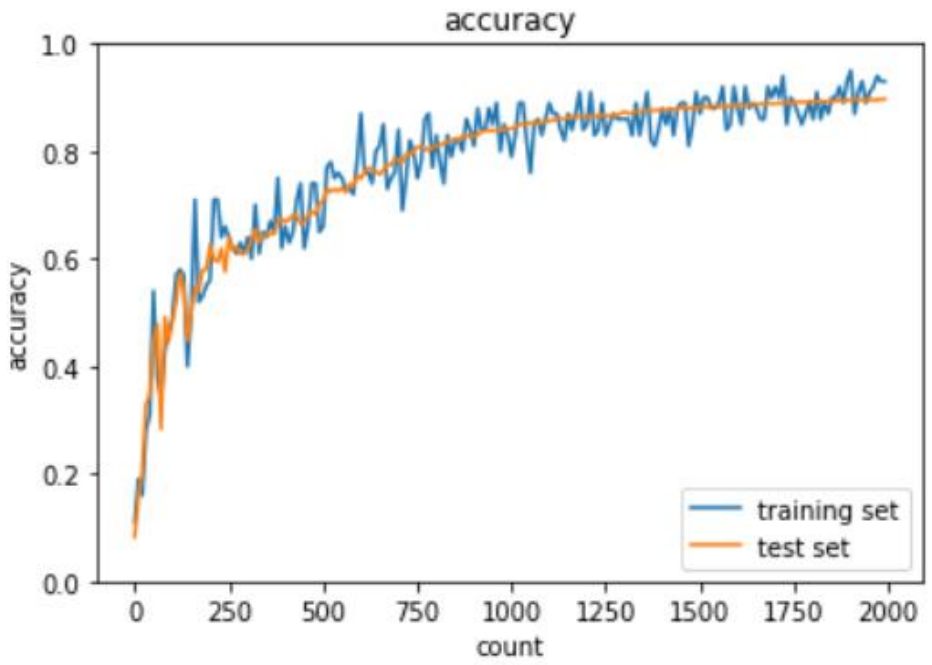
	sigmoid - gauss	ReLU - gauss	sigmoid - Xavier	ReLU - He
[40,20]				
[80,40]				

実装演習レポート【深層学習_前編】 Section1 勾配消失問題

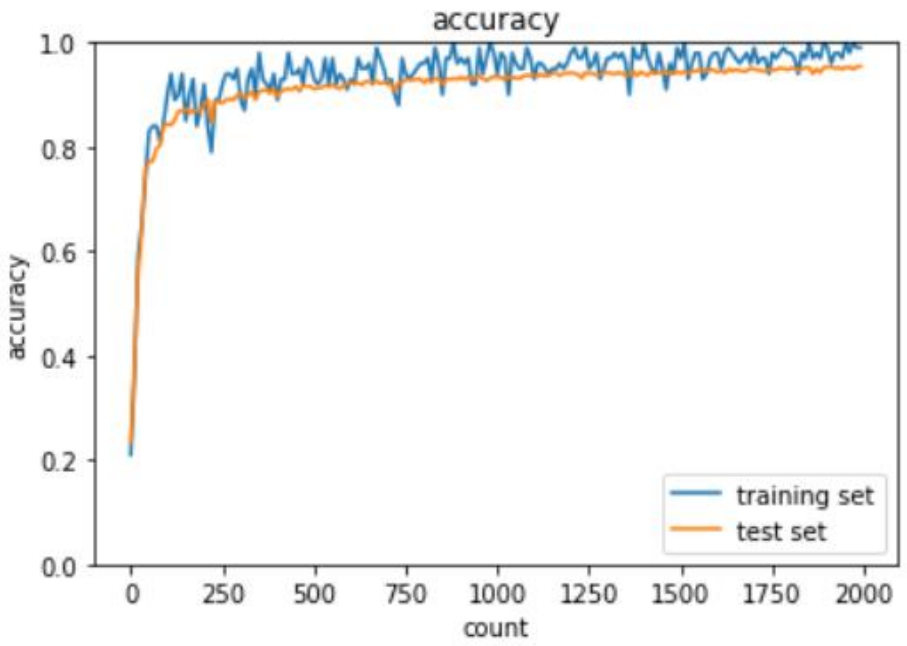
実装演習（2_2_2_vanishing_gradient_modified.ipynb）

- [try] sigmoid - He と relu - Xavier についても試してみよう

sigmoid - He



relu - Xavier



実装演習レポート【深層学習_前編】 Section1 勾配消失問題

参考図書レポート

- 勾配消失問題とは、ニューラルネットワークの設計において、勾配が消失することで学習が進まなくなる技術的な問題のこと。
- ニューラルネットワークによる学習をする際、非線形分離が必要となる問題では、パーセプトロンを多層化する必要がある。
  このことで、予測値と実際の値の差分である誤差を最小化する、最適化問題が複雑化してしまう。
  この最適化問題を解くために、誤差逆伝播法が用いられるようになったが、
  ここで勾配消失問題が発生。
  活性化関数の変更などにより勾配消失問題が解消されたことで、
  多層ニューラルネットワークは深層化に成功し、2010年代にはとりわけイメージ知覚問題で分類精度が飛躍的に向上しました。
  深層化した多層ニューラルネットワークはディープラーニングと呼ばれる。

実装演習レポート【深層学習_前編】 Section2 学習率最適化手法

要点のまとめ

・深層学習では、勾配降下法を利用してパラメータを最適化する。
その時に使用するのが、学習率。これを最適化手法を利用して最適化する。
最適化手法には下記の４つがある。

① モメンタム

前回の重みと慣性を使って学習量を調整する。よりスピーディに学習が進む。
局所的最適解でなく、大域的最適解を取りやすい。

② AdaGrad

大局的最適解を取りにくい

これまでの学習を覚えながら、重みの更新量を決定していく。
勾配の緩やかな斜面に対して、最適値に近づける。但し、鞍点問題を引き起すことがある。

③ RMSProp

AdaGradの進化系、鞍点問題を起こしにくい。
局所最適解でなく、大域最適解を取りやすい。

④ Adam

①、②、③のメリットを取り込んだ手法。

実装演習レポート【深層学習_前編】 Section2 学習率最適化手法

確認テスト

モメンタム・AdaGrad・RMSPropの特徴を
それぞれ簡潔に説明せよ。

モメンタムは慣性を利用する。
勾配が急になる程、学習が加速する。
大域的最適解を取りやすい。

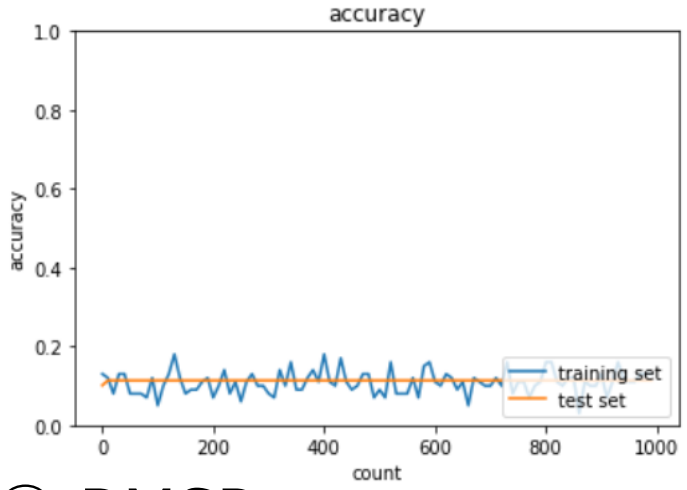
AdaGradは前回の重みを利用しながら
更新量を決めていく。鞍点問題が起きやすい。

RMSPropはAdaGradの鞍点問題に対応した
進化系。大域的最適解を取りやすい。

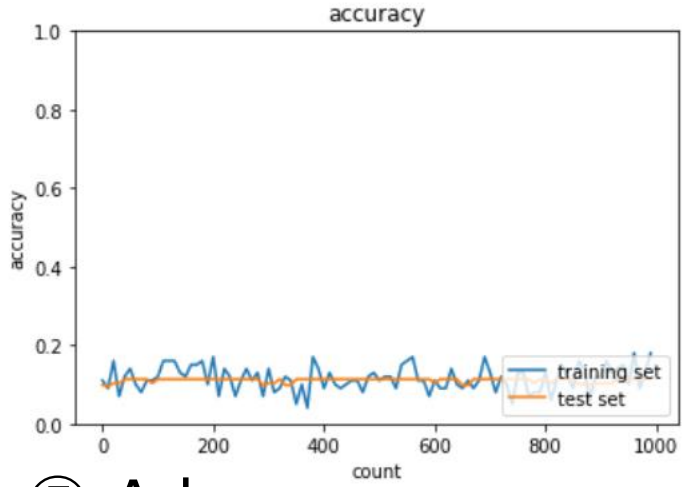
実装演習レポート【深層学習_前編】 Section2 学習率最適化手法

実装演習（2_4_optimizer.ipynb）

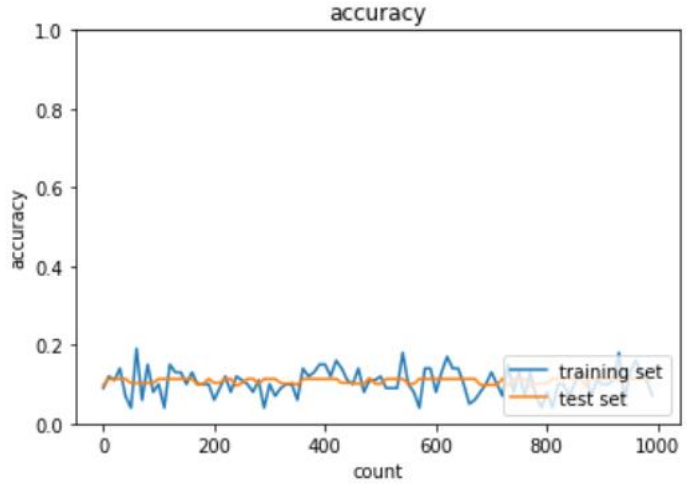
① SDG



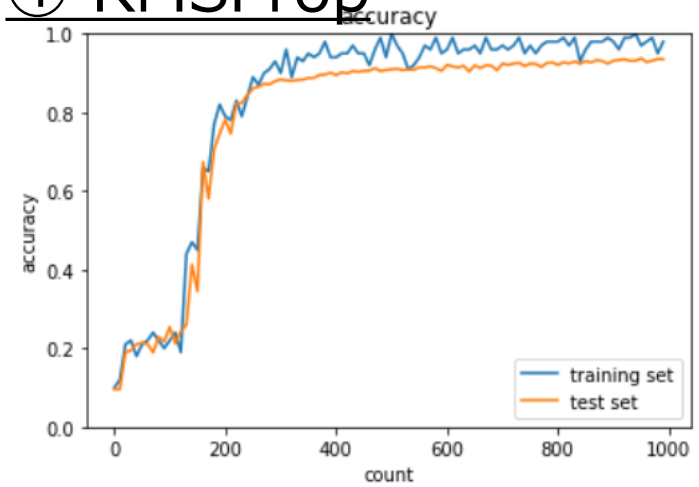
② モメンタム



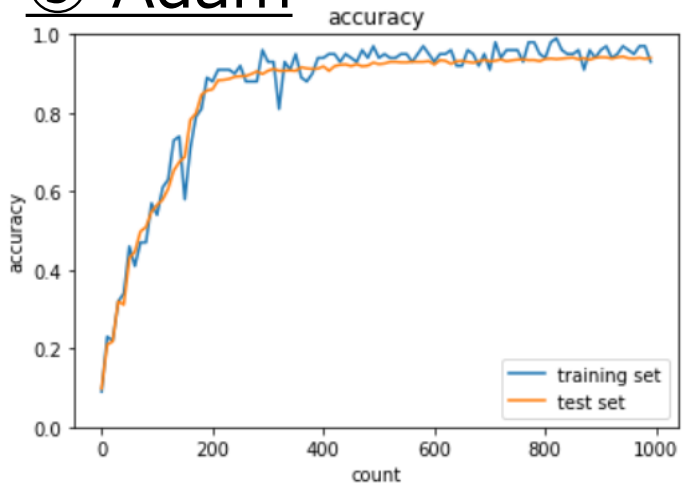
③ AdaGrad



④ RMSProp



⑤ Adam



①～③は学習ができてない。
④、⑤は学習を進めると
精度が向上。

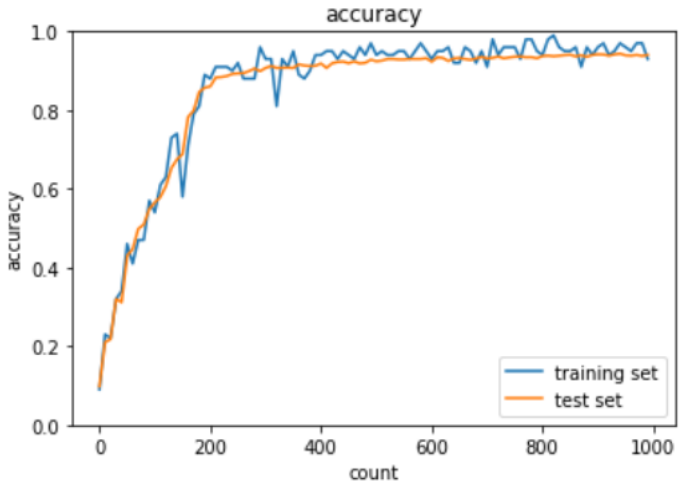
実装演習レポート【深層学習_前編】 Section2 学習率最適化手法

実装演習（2_4_optimizer.ipynb）

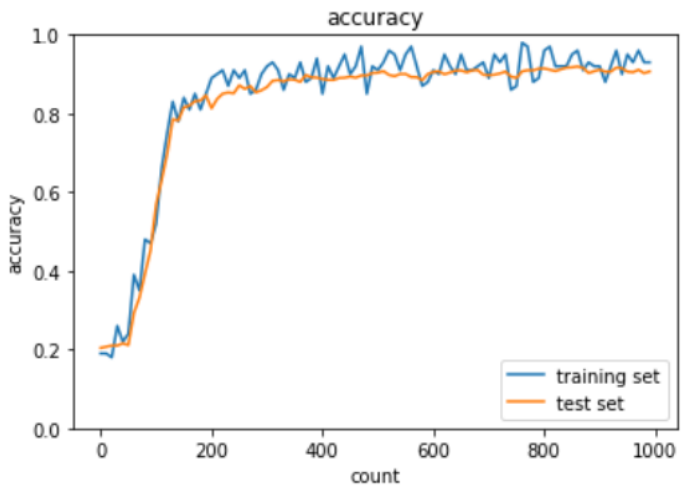
[try] 学習率を変えてみよう

・Adamで検証

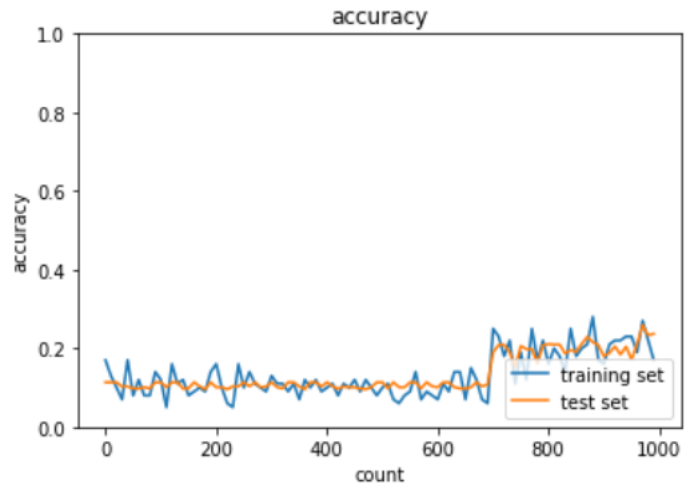
学習率 = 0.01



学習率 = 0.05



学習率 = 0.10



学習率を変えると、学習傾向が異なる。
また、学習率を大きく変化させると、学習できなくなってしまう。

実装演習レポート【深層学習_前編】 Section2 学習率最適化手法

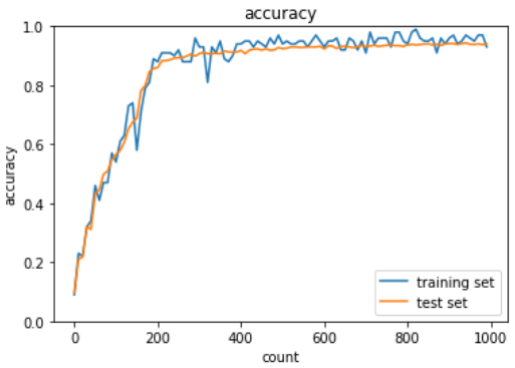
実装演習（2_4_optimizer.ipynb）

[try] 活性化関数と重みの初期化方法を変えてみよう

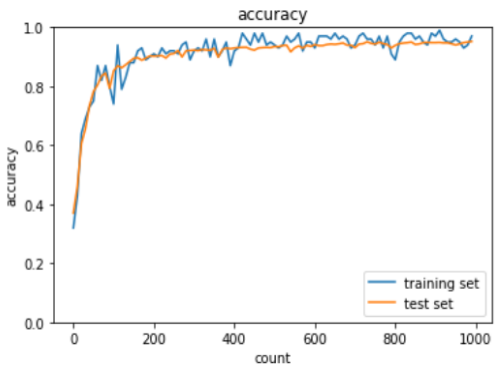
初期状態ではsigmoid - gauss
activationはReLU、weight_init_stdは別の数値や'Xavier'・'He'に変更可能

・Adamで検証

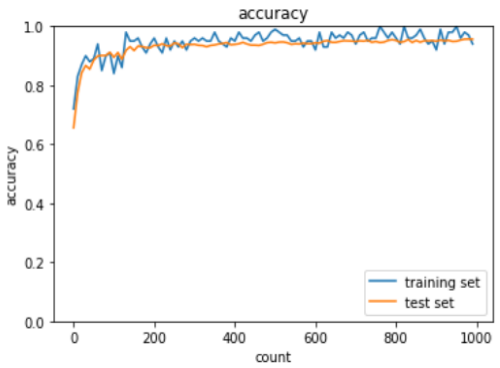
初期状態
(sigmoid-gauss 0.01)



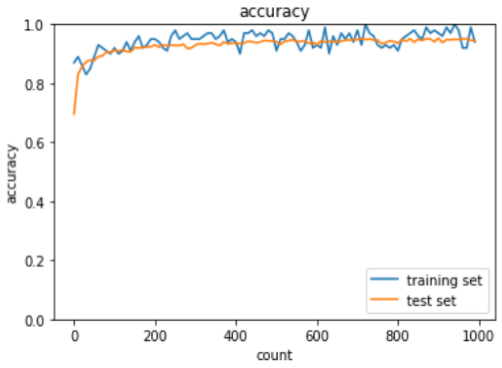
変更①
(ReLU-gauss 0.01)



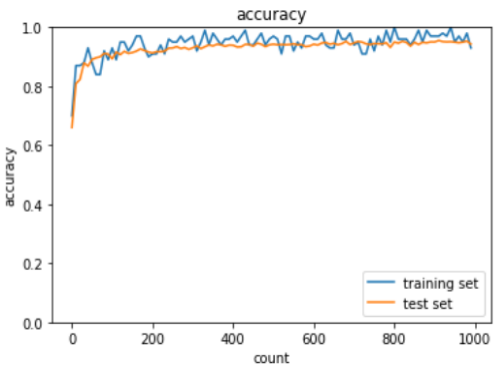
変更②
(ReLU-gauss 0.10)



変更③
(ReLU-Xavier)



変更④
(ReLU-He)



活性化関数をsigmoidからReLUに変更すると、初期の精度が高くなる。
weight_init_stdを変更すると、更に初期の精度が高くなる。

実装演習レポート【深層学習_前編】 Section2 学習率最適化手法

実装演習（2_4_optimizer.ipynb）

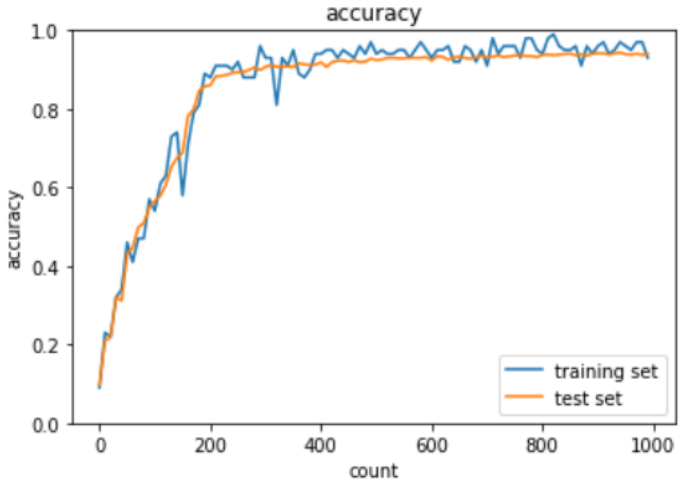
[try] バッチ正規化をしてみよう

use_batchnormをTrueにしよう

・Adamで検証

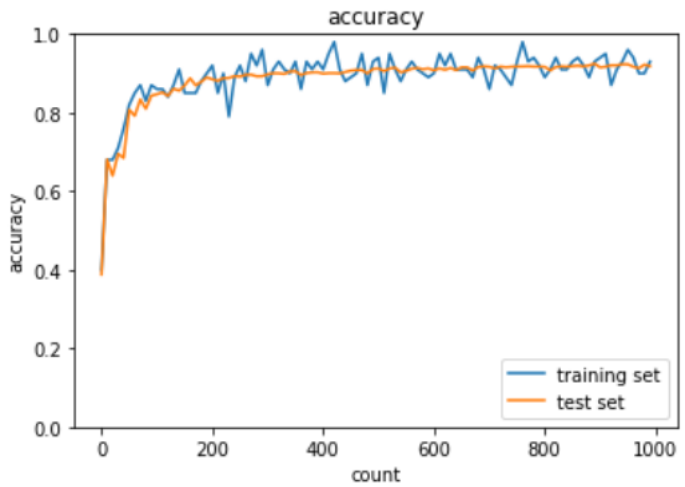
初期状態

（ use_batchnorm = False）



バッチ正規化

（ use_batchnorm = True）



バッチ正規化すると、初期の精度が高くなる。

実装演習レポート【深層学習_前編】 Section2 学習率最適化手法

参考図書レポート

- SGDは最も基本となるアルゴリズムであり、勾配の情報のみを使って
  パラメータを動かす
- モメンタムはSGDに「速度」と「慣性」の概念を追加したもの
- AdaGradは「学習が進むほど学習率を小さくしていく」という機能をパラメータ
  ごとに独立して行う。
- RMSPropはAdaGradを最近の勾配ほど強く影響するように改良したもの
- AdamはモメンタムとRMSPropを組み合わせたようなもの

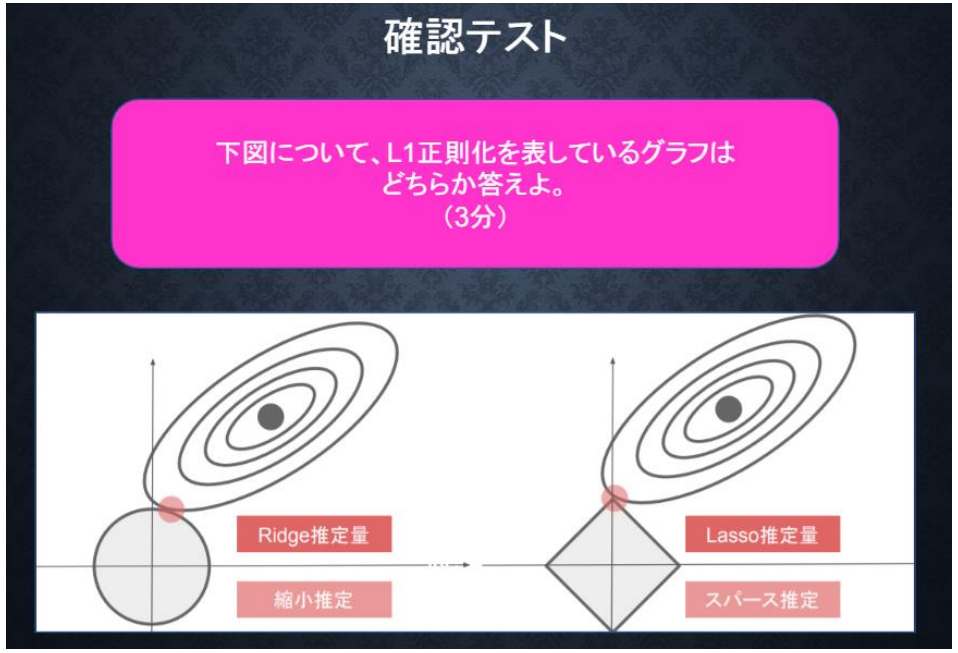
実装演習レポート【深層学習_前編】 Section3 過学習

要点のまとめ

- 過学習とは訓練データに特化して学習してしまい、検証データに対しては
  十分な学習ができてないこと。
  原因としては、パラメータの数が多い。パラメータの値が適切でない、
  ノードが多い等が挙げられる。ネットワークの自由度が高い時に過学習になる。
- 過学習を抑える方法として正則化手法がある。(ネットワークの自由度を抑える)
  L1 , L2正則化がある。大きすぎる重みを抑制し、過学習を防止する。
- また、過学習を抑える方法として、ドロップアウトがある。
  ランダムにノードを削除して学習させる。
  シンプルな方法だが、効果があるのでよく用いられる。

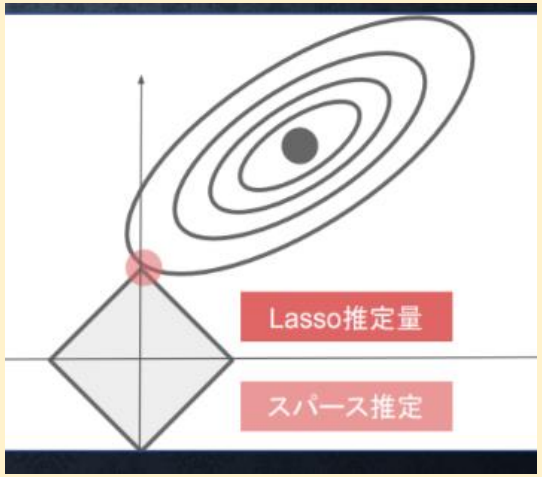
実装演習レポート【深層学習_前編】 Section3 過学習

確認テスト



L1正則化を表しているグラフはどちらか答えよ。

L1正則化グラフ



一つの重みが0になるところを探す。

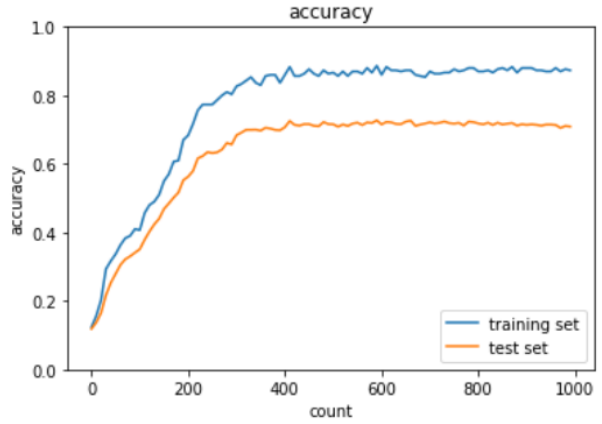
実装演習レポート【深層学習_前編】 Section3 過学習

実装演習（2_5_overfiting.ipynb）

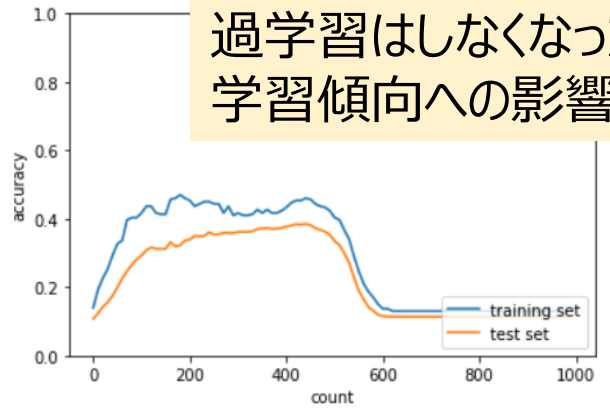
[try] weigth_decay_lambdaの値を変更して正則化の強さを確認しよう

L2
正則化

weight_decay_lambda = 0.1



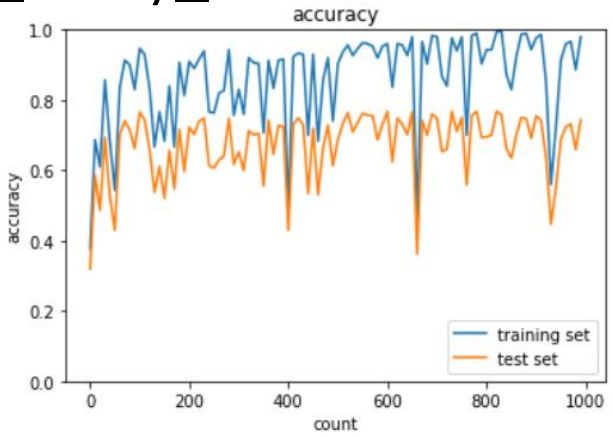
weight_decay_lambda = 0.2



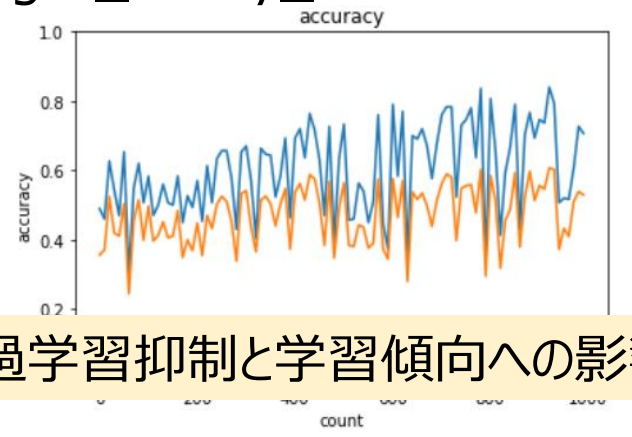
過学習はしなくなったが、
学習傾向への影響は大きい。

L1
正則化

weight_decay_lambda = 0.005



weight_decay_lambda = 0.01



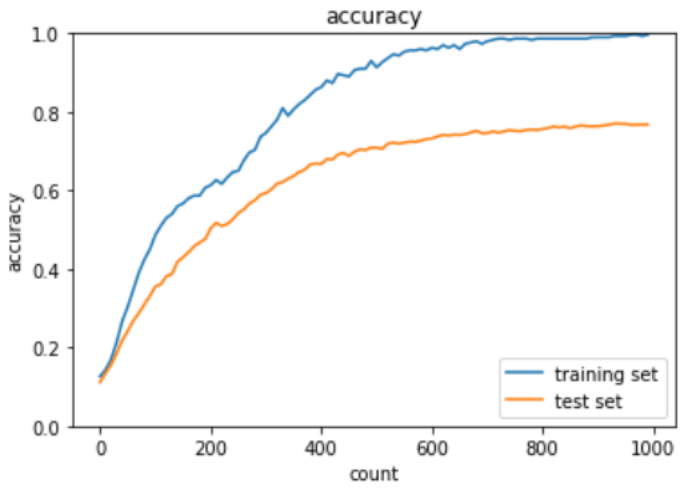
L2に比べると、過学習抑制と学習傾向への影響は小さい。

実装演習レポート【深層学習_前編】 Section3 過学習

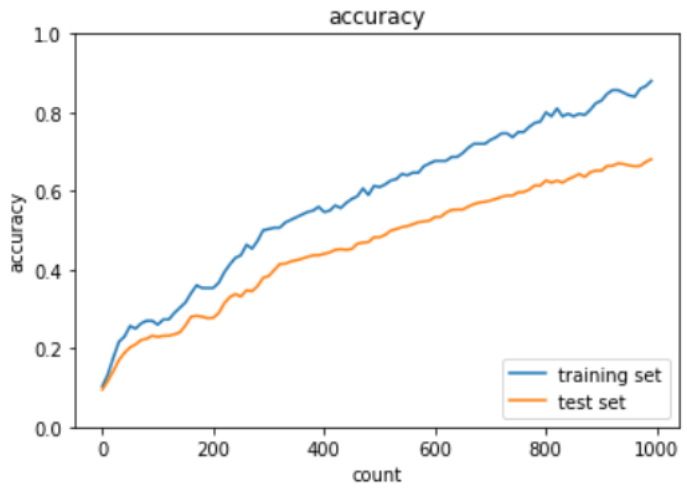
実装演習（2_5_overfiting.ipynb）

[try] dropout_ratioの値を変更してみよう

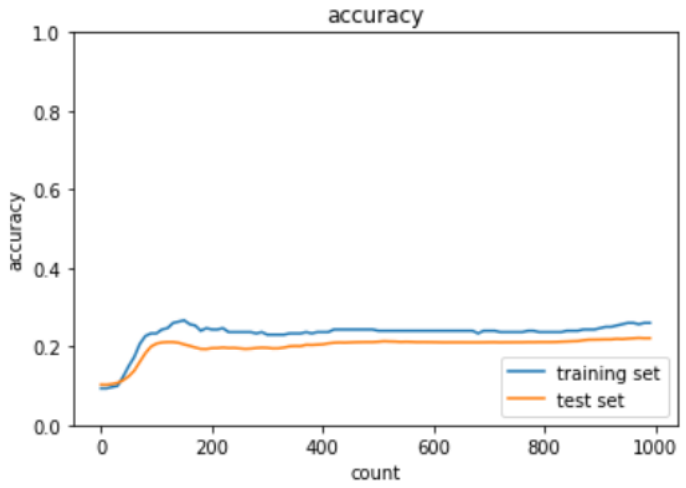
dropout_ratio = 0.075



dropout_ratio = 0.15



dropout_ratio = 0.30



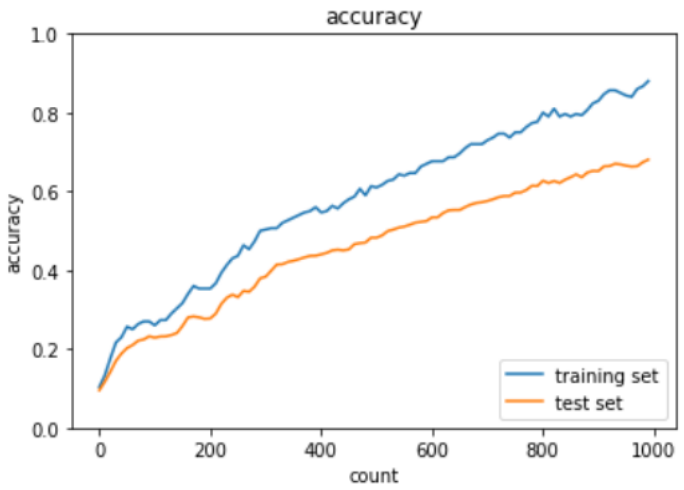
Dropout_ratioの値を小さくすると、精度は高くなるが、過学習しやすくなる。
Dropout_ratioの値を大きくすると、精度は小さくなるが、過学習はしにくくなる。

実装演習レポート【深層学習_前編】 Section3 過学習

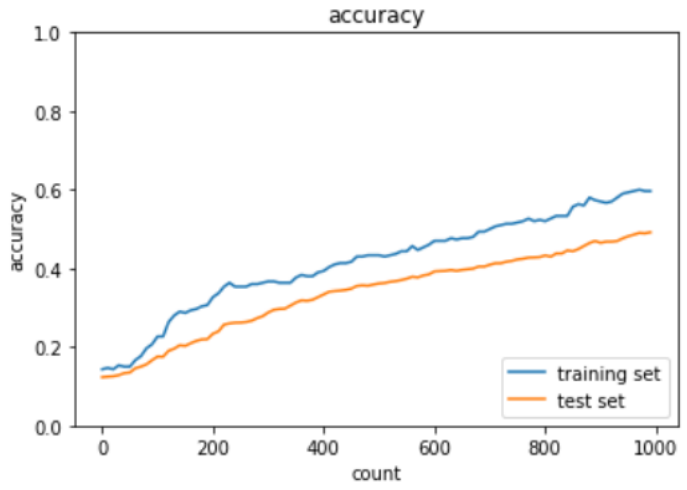
実装演習（2_5_overfiting.ipynb）

[try] optimizerとdropout_ratioの値を変更してみよう

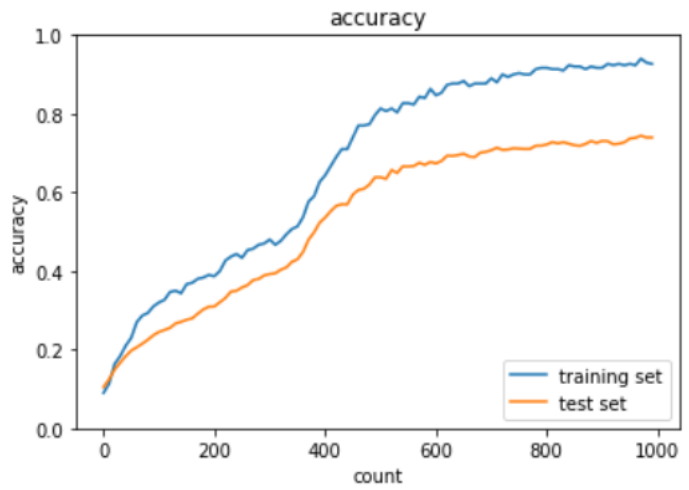
Dropout
dropout_ratio = 0.15



Dropout + L1
dropout_ratio = 0.15



Dropout + L1
dropout_ratio = 0.08



DropoutからL1正則化を加えると、過学習は若干抑えられるが精度は低くなる。
そこから、dropout_ratioを小さくすると、精度は高くなるが、過学習もしやすくなる。

実装演習レポート【深層学習_前編】 Section3 過学習

参考図書レポート

- 過学習はバリアンス・バイアスと関係性が深い。
  バリアンスとは予測結果のばらつき、バイアスは予測結果と実測値の誤差。

  バイアスが低いモデルの方がモデルとしては優秀であるが、バイアスを
  低くしすぎるとノイズに対しても適合することになるので予測結果のばらつきが
  大きくなってくる。すなわち、バリアンスが大きくなってくる。

  つまり、「バリアンスが大きい状態」＝「過学習している可能性がある」。

  バリアンスとバイアスは基本トレードオフの関係になっており、
  これらの**バランスがとれたモデルを選択することが重要**となる。

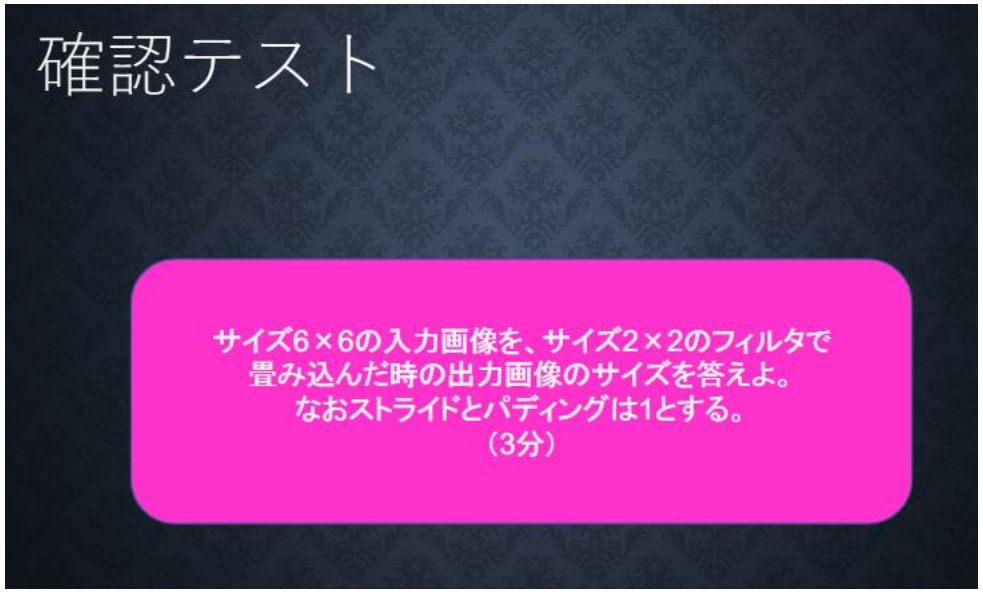
実装演習レポート【深層学習_前編】 Section4 CNNの概念

要点のまとめ

・CNN（畳み込みニューラルネットワーク）とは画像の識別・画像の認識を処理するのに主に使われる。
次元間で繋がりのあるデータを扱える。
・CNNの代表的なものにLeNetがある。
入力層、畳み込み層、プーリング層、全結合層、出力層からなる。
畳み込み層ではフィルターを使って入力データを処理して、出力する。
畳み込み層ではバイアス、パディング、ストライド、チャンネルの処理をする。
最後は全結合層で処理する。重みとバイアスを使って数字の変換をする。
全結合層では画像の場合、縦、横、チャンネルの3次元データだが、
1次元のデータとして処理される。その為、RGBの各チャンネルの関連性が、
学習に反映されない。
プーリング層とは重みを使わずにデータを処理する。対象領域のMax値もしくは平均値を取得する。

実装演習レポート【深層学習_前編】 Section4 CNNの概念

確認テスト



サイズ6×6の入力画像を、サイズ2×2のフィルタで畳み込んだ時の出力画像のサイズを答えよ。
なおストライドとパディングは1とする。

出力画像のサイズは、『7×7』

$$縦 = \frac{画像の高さ \ + \ 2 \times パディング高さ \ - \ フィルター高さ}{ストライド} + 1 \ = 7$$

$$横 = \frac{画像の幅 \ + \ 2 \times パディング幅 \ - \ フィルター幅}{ストライド} + 1 \ = 7$$

実装演習レポート【深層学習_前編】 Section4 CNNの概念

実装演習（2_6_simple_convolution_network_after.ipynb）

[try] im2colの処理を確認しよう

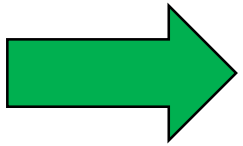
・関数内でtransposeの処理をしている行をコメントアウトして下のコードを実行してみよう

コメントアウト

```
#col = col.transpose(0, 4, 5, 1, 2, 3)
```

```
========== input_data ===========
 [[[[48.  4.  4. 89.]
   [33. 27. 21. 88.]
   [40. 83. 81. 17.]
   [64. 10. 44. 79.]]]


 [[[35. 42. 29. 70.]
   [31. 91. 46. 90.]
   [21. 27.  9. 85.]
   [46. 53. 50.  1.]]]]
==============================
============= col ==============
 [[48.  4.  4. 33. 27. 21. 40. 83. 81.]
 [ 4.  4. 89. 27. 21. 88. 83. 81. 17.]
 [33. 27. 21. 40. 83. 81. 64. 10. 44.]
 [27. 21. 88. 83. 81. 17. 10. 44. 79.]
 [35. 42. 29. 31. 91. 46. 21. 27.  9.]
 [42. 29. 70. 91. 46. 90. 27.  9. 85.]
 [31. 91. 46. 21. 27.  9. 46. 53. 50.]
 [91. 46. 90. 27.  9. 85. 53. 50.  1.]]
==============================
```



```
========== input_data ===========
 [[[[91. 19. 56. 44.]
   [14. 45. 22. 38.]
   [ 1. 11.  5.  6.]
   [ 7. 39. 25. 68.]]]


 [[[80.  4. 33. 71.]
   [29. 78. 19. 23.]
   [28. 30. 62. 63.]
   [57. 63.  3. 10.]]]]
==============================
============= col ==============
 [[91. 19. 14. 45. 19. 56. 45. 22. 56.]
 [44. 22. 38. 14. 45.  1. 11. 45. 22.]
 [11.  5. 22. 38.  5.  6.  1. 11.  7.]
 [39. 11.  5. 39. 25.  5.  6. 25. 68.]
 [80.  4. 29. 78.  4. 33. 78. 19. 33.]
 [71. 19. 23. 29. 78. 28. 30. 78. 19.]
 [30. 62. 19. 23. 62. 63. 28. 30. 57.]
 [63. 30. 62. 63.  3. 62. 63.  3. 10.]]
==============================
```

transpose処理をコメントアウトしたので、各項目の順番が異なり、違ったデータが表示される。

実装演習レポート【深層学習_前編】 Section4 CNNの概念

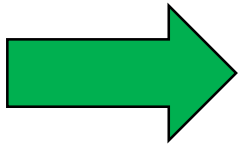
実装演習（2_6_simple_convolution_network_after.ipynb）

[try] im2colの処理を確認しよう

・input_dataの各次元のサイズやフィルターサイズ・ストライド・パディングを変えてみよう

```
========== input_data ===========
 [[[[48.  4.  4. 89.]
   [33. 27. 21. 88.]
   [40. 83. 81. 17.]
   [64. 10. 44. 79.]]]


 [[[35. 42. 29. 70.]
   [31. 91. 46. 90.]
   [21. 27.  9. 85.]
   [46. 53. 50.  1.]]]]
==============================
============= col ==============
 [[48.  4.  4. 33. 27. 21. 40. 83. 81.]
 [ 4.  4. 89. 27. 21. 88. 83. 81. 17.]
 [33. 27. 21. 40. 83. 81. 64. 10. 44.]
 [27. 21. 88. 83. 81. 17. 10. 44. 79.]
 [35. 42. 29. 31. 91. 46. 21. 27.  9.]
 [42. 29. 70. 91. 46. 90. 27.  9. 85.]
 [31. 91. 46. 21. 27.  9. 46. 53. 50.]
 [91. 46. 90. 27.  9. 85. 53. 50.  1.]]
==============================
```



```
========== input_data ===========
 [[[[44. 13. 53. 97. 65.]
   [23. 84. 65. 43.  4.]
   [13. 88. 60. 26. 21.]
   [36. 16. 74. 84. 81.]
   [82. 82. 69. 97. 17.]]]


 [[[21. 63.  1.  0. 97.]
   [85. 97. 34. 97.  4.]
   [23. 17. 48. 11. 49.]
   [89. 26. 38. 77. 85.]
   [95. 91. 66. 38. 72.]]]]
==============================
============= col ==============
 [[44. 13. 53. 23. 84. 65. 13. 88. 60.]
 [13. 53. 97. 84. 65. 43. 88. 60. 26.]
 [53. 97. 65. 65. 43.  4. 60. 26. 21.]
 [23. 84. 65. 13. 88. 60. 36. 16. 74.]
 [84. 65. 43. 88. 60. 26. 16. 74. 84.]
 [65. 43.  4. 60. 26. 21. 74. 84. 81.]
 [13. 88. 60. 36. 16. 74. 82. 82. 69.]
 [88. 60. 26. 16. 74. 84. 82. 69. 97.]
 [60. 26. 21. 74. 84. 81. 69. 97. 17.]
 [21. 63.  1. 85. 97. 34. 23. 17. 48.]
 [63.  1.  0. 97. 34. 97. 17. 48. 11.]
 [ 1.  0. 97. 34. 97.  4. 48. 11. 49.]
 [85. 97. 34. 23. 17. 48. 89. 26. 38.]
 [97. 34. 97. 17. 48. 11. 26. 38. 77.]
 [34. 97.  4. 48. 11. 49. 38. 77. 85.]
 [23. 17. 48. 89. 26. 38. 95. 91. 66.]
 [17. 48. 11. 26. 38. 77. 91. 66. 38.]
 [48. 11. 49. 38. 77. 85. 66. 38. 72.]]
 ==============================
```

サイズを４×４から５×5に変更。

実装演習レポート【深層学習_前編】 Section4 CNNの概念

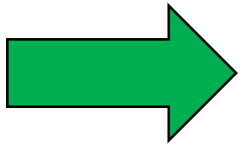
実装演習（2_6_simple_convolution_network_after.ipynb）

[try] im2colの処理を確認しよう

・input_dataの各次元のサイズやフィルターサイズ・ストライド・パディングを変えてみよう

```
========== input_data ===========
 [[[[48.  4.  4. 89.]
   [33. 27. 21. 88.]
   [40. 83. 81. 17.]
   [64. 10. 44. 79.]]]


 [[[35. 42. 29. 70.]
   [31. 91. 46. 90.]
   [21. 27.  9. 85.]
   [46. 53. 50.  1.]]]]
=============================
============= col ==============
 [[48.  4.  4. 33. 27. 21. 40. 83. 81.]
 [ 4.  4. 89. 27. 21. 88. 83. 81. 17.]
 [33. 27. 21. 40. 83. 81. 64. 10. 44.]
 [27. 21. 88. 83. 81. 17. 10. 44. 79.]
 [35. 42. 29. 31. 91. 46. 21. 27.  9.]
 [42. 29. 70. 91. 46. 90. 27.  9. 85.]
 [31. 91. 46. 21. 27.  9. 46. 53. 50.]
 [91. 46. 90. 27.  9. 85. 53. 50.  1.]]
=============================
```



```
========== input_data ===========
 [[[[58. 91. 99. 48.]
   [57. 96. 44. 24.]
   [14. 51. 89. 51.]
   [ 7.  2. 91. 67.]]]


 [[[10. 72. 92. 65.]
   [64. 21. 92. 42.]
   [12. 64. 90. 15.]
   [20. 90. 44. 73.]]]]
=============================
============= col ==============
 [[58. 91. 99. 48. 57. 96. 44. 24. 14. 51. 89. 51.  7.  2. 91. 67.]
 [10. 72. 92. 65. 64. 21. 92. 42. 12. 64. 90. 15. 20. 90. 44. 73.]]
=============================
```

フィルターサイズを３×３から４×４に変更。

実装演習レポート【深層学習_前編】 Section4 CNNの概念

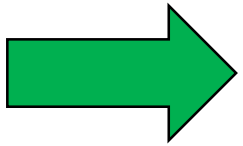
実装演習（2_6_simple_convolution_network_after.ipynb）

[try] im2colの処理を確認しよう

・input_dataの各次元のサイズやフィルターサイズ・ストライド・パディングを変えてみよう

```
========== input_data ===========
 [[[[48.  4.  4. 89.]
   [33. 27. 21. 88.]
   [40. 83. 81. 17.]
   [64. 10. 44. 79.]]]


 [[[35. 42. 29. 70.]
   [31. 91. 46. 90.]
   [21. 27.  9. 85.]
   [46. 53. 50.  1.]]]]
==============================
============= col ==============
 [[48.  4.  4. 33. 27. 21. 40. 83. 81.]
 [ 4.  4. 89. 27. 21. 88. 83. 81. 17.]
 [33. 27. 21. 40. 83. 81. 64. 10. 44.]
 [27. 21. 88. 83. 81. 17. 10. 44. 79.]
 [35. 42. 29. 31. 91. 46. 21. 27.  9.]
 [42. 29. 70. 91. 46. 90. 27.  9. 85.]
 [31. 91. 46. 21. 27.  9. 46. 53. 50.]
 [91. 46. 90. 27.  9. 85. 53. 50.  1.]]
==============================
```



```
========== input_data ===========
 [[[[56. 84.  9. 35.]
   [12. 97. 27. 91.]
   [62. 27. 54. 21.]
   [91. 94. 87. 24.]]]


 [[[93. 63. 15. 17.]
   [61.  1. 44. 69.]
   [42. 48. 51. 84.]
   [75. 26. 82. 24.]]]]
==============================
============= col ==============
 [[56. 84.  9. 12. 97. 27. 62. 27. 54.]
 [93. 63. 15. 61.  1. 44. 42. 48. 51.]]
==============================
```

ストライドを１から２に変更。

実装演習レポート【深層学習_前編】 Section4 CNNの概念

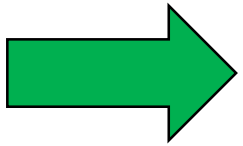
実装演習（2_6_simple_convolution_network_after.ipynb）

[try] im2colの処理を確認しよう

・input_dataの各次元のサイズやフィルターサイズ・ストライド・パディングを変えてみよう

```
========== input_data ===========
 [[[[48.  4.  4. 89.]
   [33. 27. 21. 88.]
   [40. 83. 81. 17.]
   [64. 10. 44. 79.]]]


 [[[35. 42. 29. 70.]
   [31. 91. 46. 90.]
   [21. 27.  9. 85.]
   [46. 53. 50.  1.]]]]
==============================
============= col ==============
 [[48.  4.  4. 33. 27. 21. 40. 83. 81.]
 [ 4.  4. 89. 27. 21. 88. 83. 81. 17.]
 [33. 27. 21. 40. 83. 81. 64. 10. 44.]
 [27. 21. 88. 83. 81. 17. 10. 44. 79.]
 [35. 42. 29. 31. 91. 46. 21. 27.  9.]
 [42. 29. 70. 91. 46. 90. 27.  9. 85.]
 [31. 91. 46. 21. 27.  9. 46. 53. 50.]
 [91. 46. 90. 27.  9. 85. 53. 50.  1.]]
==============================
```



```
========== input_data ===========
 [[[[50. 15. 55.  6.]
   [96. 17. 50. 29.]
   [11. 61. 50. 44.]
   [89. 29. 66. 49.]]]


 [[[75. 91. 80. 17.]
   [44. 25. 48. 90.]
   [43. 81. 20. 38.]
   [52. 33.  4. 76.]]]]
==============================
```

パディングを0から１に変更

```
============ col =============
 [[ 0.  0.  0.  0. 50. 15.  0. 96. 17.]
 [ 0.  0.  0. 50. 15. 55. 96. 17. 50.]
 [ 0.  0.  0. 15. 55.  6. 17. 50. 29.]
 [ 0.  0.  0. 55.  6.  0. 50. 29.  0.]
 [ 0. 50. 15.  0. 96. 17.  0. 11. 61.]
 [50. 15. 55. 96. 17. 50. 11. 61. 50.]
 [15. 55.  6. 17. 50. 29. 61. 50. 44.]
 [55.  6.  0. 50. 29.  0. 50. 44.  0.]
 [ 0. 96. 17.  0. 11. 61.  0. 89. 29.]
 [96. 17. 50. 11. 61. 50. 89. 29. 66.]
 [17. 50. 29. 61. 50. 44. 29. 66. 49.]
 [50. 29.  0. 50. 44.  0. 66. 49.  0.]
 [ 0. 11. 61.  0. 89. 29.  0.  0.  0.]
 [11. 61. 50. 89. 29. 66.  0.  0.  0.]
 [61. 50. 44. 29. 66. 49.  0.  0.  0.]
 [50. 44.  0. 66. 49.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. 75. 91.  0. 44. 25.]
 [ 0.  0.  0. 75. 91. 80. 44. 25. 48.]
 [ 0.  0.  0. 91. 80. 17. 25. 48. 90.]
 [ 0.  0.  0. 80. 17.  0. 48. 90.  0.]
 [ 0. 75. 91.  0. 44. 25.  0. 43. 81.]
 [75. 91. 80. 44. 25. 48. 43. 81. 20.]
 [91. 80. 17. 25. 48. 90. 81. 20. 38.]
 [80. 17.  0. 48. 90.  0. 20. 38.  0.]
 [ 0. 44. 25.  0. 43. 81.  0. 52. 33.]
 [44. 25. 48. 43. 81. 20. 52. 33.  4.]
 [25. 48. 90. 81. 20. 38. 33.  4. 76.]
 [48. 90.  0. 20. 38.  0.  4. 76.  0.]
 [ 0. 43. 81.  0. 52. 33.  0.  0.  0.]
 [43. 81. 20. 52. 33.  4.  0.  0.  0.]
 [81. 20. 38. 33.  4. 76.  0.  0.  0.]
 [20. 38.  0.  4. 76.  0.  0.  0.  0.]]
============================
```

実装演習レポート【深層学習_前編】 Section4 CNNの概念

実装演習（2_6_simple_convolution_network_after.ipynb）

[try] col2imの処理を確認しよう

・im2colの確認で出力したcolをimageに変換して確認しよう

col2imでcolからimageに変換。

```
[14] # ここにcol2imでの処理を書こう
     img = col2im(col, input_shape=input_data.shape, filter_h=filter_h, filter_w=filter_w, stride=stride, pad=pad)
     print(img)
```

```
[[[[ 30. 112.  90.  95.]
   [186.   8. 160.  80.]
   [186.  96. 228.  62.]
   [ 82.  88.  44.  48.]]]


 [[[ 80. 114.  66.  90.]
   [184. 292. 156.  38.]
   [184. 208. 132.  38.]
   [ 99.  32.  46.  19.]]]]
```

実装演習レポート【深層学習_前編】 Section4 CNNの概念

参考図書レポート

- CNNは大きく分けて2つのパートに分けることができる。
  １つ目は、画像を読込み、畳み込みやプーリングによって特徴マップを作成する特徴量抽出パート
  ２つ目は、全結合層を繰り返すことで最終的な出力を得る識別パート

  最終的な出力は目的によって設定する必要があるが、例えば図中の物体が何であるか判別したい場合は、判別対象のクラスの確率になるようにする。この確率に変換するときはソフトマックス関数などの活性化関数を使用する。

- CNNの注意点
  CNNはあくまで畳み込み処理を利用したニューラルネットワークであり、モデルの精度がよくなるように、ネットワークを設計者が考えたり検証したりする必要がある。この指針、検証方法を理解することが必要。

実装演習レポート【深層学習_前編】 Section5 最新のCNN

要点のまとめ

- AlexNet

  5 層の畳み込み層およびプーリング層など、それに続く 3 層の全結合層から構成される。
  入力には224×224の画像を使用する。
  11×11のフィルターで畳み込み演算をして、96チャンネルに増やす。
  5×5のフィルターでマックスプーリングして縮め込む。256チャンネルまで増やす。
  3×3のフィルターでマックスプーリングして縮め込む。384チャンネルまで増やす。
  パディング処理をして畳み込み演算をする。
  最終的に13×13のサイズ、256チャンネルとなる。
  flattenの作業で一列の数字に並べ替える。
  過学習を防ぐ施策として、サイズ4096の全結合層の出力にドロップアウトを使用している。

実装演習レポート【深層学習_前編】 Section5 最新のCNN

参考図書レポート

- AlexNetは、2012年のイメージネット画像認識コンテスト（ILSVRC）で、従来手法のサポートベクターマシンに替わりディープラーニングに基づくモデルとして初めて優勝した。

  畳み込み層５層にプーリング層３層の構成。
  1400万以上のカラー画像を1万カテゴリに分類するというコンペティション向けに開発されたため、高い認識能力を誇るが、学習時にチューニングすべきパラメータ数も多く、実装には高いスペックをもつハードウェアが必要となる。