ラビットチャレンジstudyai0027863

深層学習　前半

実装演習レポート北村裕斗　hiroto7018@gmail.com

ビデオ視聴学習者提出区分け	科目	章タイトル	1点100文字以上で要点のまとめ	実装演習結果キャプチャー又はサマリーと考察	「確認テスト」など、自身の考察結果	演習問題や参考図書、修了課題など関連記事レポートによる加点
【c】 1つのURLで提出	深層学習day1 （基準点：15点）	Section1：入力層～中間層	1点	1点	1点	1点
		Section2：活性化関数	1点	1点	1点	1点
		Section3：出力層	1点	1点	1点	1点
		Section4：勾配降下法	1点	1点	1点	1点
		Section5：誤差逆伝播法	1点	1点	1点	1点
	深層学習day2 （基準点：15点）	Section1：勾配消失問題	1点	1点	1点	1点
		Section2：学習率最適化手法	1点	1点	1点	1点
		Section3：過学習	1点	1点	1点	1点
		Section4：畳み込みニューラルネットワークの概念	1点	1点	1点	1点
		Section5：最新のCNN	1点	1点	1点	1点

- 講義動画視聴およびソース実装演習を確実に実施しているかを審査します。
  - レポートが講義本筋に沿ってまとめてあるか。受講者自身の言葉で課題や気づきが記載されているか。
- 必須要件を満たさない場合、他の受講者のレポートのコピーなど不正が発覚した場合、差し戻しとします。
- 数式やコード演習結果の個々の間違いは審査に影響しません。なお、講師からのフィードバックはありませんのでご了承ください。
- 各科目以下の場合は差し戻し
  - 基準点以下
  - 実装演習を全く行なっていない場合（応用数学以外）

下記の要件の通り、区分ごとに単元レポートを作成、ご提出ください。

1）各章につき100文字以上で要点をまとめ、実装演習結果、確認テストについての自身の考察等を取り入れたレポートとする。

2）各科目の基準点が足りない場合、実装演習が不足する場合は差し戻しとする。

※各章は講義動画および講義資料（PDF）でご確認ください。

- **レポート掲載場所：自身のGitHub、SlideShareなどのBlogやWeb媒体で作成。お持ちでない方は新規アカウント（無料）を作成。**
- **レポート提出方法：レポート掲載ページのURLを【学習システム（LMS）内の指定フォーム】より提出。**

DAY1

Section1) 入力層～中間層

入力xに重みをかけた値と、それにバイアスを足した値を中間層に出力する。そこで、活性化関数を通ってzが出力される。

動物種類分類ネットワーク

例　体重、体長、ひげ、毛の長さ、耳の大きさ、足の長さ。これらの入力層を一つにまとめ次の層へ情報を伝播する。

確認テスト2

u = x1w1 + x2w2 + x3w3.... + b の数式をPythonで書け。

解答

u1 =np.dot(x, W1) + b1

確認テスト3

1-1のファイルから中間層の出力を定義しているソースを抜き出せ。NN全体像ソースコード

```
# 中間層出力
z = functions.relu(u)
print_vec("中間層出力", z)
```

Jupyter演習

入力 [01,0.2] バイアス 0.5

活性化関数 : relu

numpy使い方

np.zeros(2) → zero初期化

np.ones(2) → 1で初期化

np.random.rand(2) → ランダム値(0〜1)

np.random.randint(5, size=(2)) → ランダム値(任意の整数: ここでは5)

Section2) 活性化関数

活性化関数とはニューラルネットワークにおいて、次の層への出力の大きさ決める非線形の関数である。また入力値の値によって、次の層への信号のON/OFFや強弱を定める働きをもつ。

確認（復習）テスト線形と非線形の違いを図にかいて簡易に説明せよ。

中間層用の活性化関数

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もたらしている。

シグモイド（ロジスティック）関数

→0 ~ 1の間を緩やかに変化する関数で、ステップ関数ではON/OFFしかない状態に対し、信号の強弱を伝えられるようになり、予想ニューラルネットワーク普及のきっかけとなった。課題大きな値では出力の変化が微小なため、勾配消失問題を引き起こす事があった。

ステップ関数

→しきい値を超えたら発火する関数であり、出力は常に1か0。パーセプトロン（ニューラルネットワークの前身）で利用された関数。課題0 -1間の間を表現できず、線形分離可能なものしか学習できなかった

確認テスト

配布されたソースコードより該当する箇所を抜き出せ

$$z = f(u) \ .. \ (1.5)$$

解答

z1 = functions.sigmoid(u)

Section3) 出力層

3-1 誤差関数

全結合NN-誤差関数

誤差の計算誤差関数= 二乗誤差の場合

確認テスト

なぜ、引き算でなく二乗するか述べよ・下式の1/2はどういう意味を持つか述べよ

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

loss = functions.mean_squared_error(d, y)

確認テスト

1なぜ2乗しているのか？

→ マイナス値を出さないため

2なぜ 1/2 しているのか？

→ 微分したときに計算しやすいため

3-2 出力層の活性化関数

出力層の中間層との違い

【値の強弱】

•**中間層** しきい値の前後で信号の強弱を調整

•**出力層** 信号の大きさ（比率）はそのままに**変換**

【確率出力】

•分類問題の場合、出力層の出力は0 ~ 1 の範囲に限定し、総和を1とする必要がある

→出力層と中間層で利用される活性化関数が異なる

出力層用の活性化関数

ソフトマックス関数

恒等写像

シグモイド関数（ロジスティック関数）

```
def   sigmoid(x):return 1/(1 + np.exp(-x))
```

確認テスト

数式に該当するソースコードを示し、一行ずつ処理の説明をせよ。

```
def softmax(x):
        if x.ndim == 2:
        x = x.Tx = x -np.max(x, axis=0)
        y = np.exp(x) /np.sum(np.exp(x), axis=0)
        return y.T
x = x -np.max(x) #オーバーフロー対策
        return np.exp(x) / np.sum(np.exp(x))
```

数式とコード

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

```
def   mean_squared_error(d, y):return np.mean(np.square(d -y)) / 2)
```

確認テスト

数式に該当するソースコードを示し、一行ずつ処理の説明をせよ。

```
def  cross_entropy_error(d, y):
        if  y.ndim== 1:
        d= d.reshape(1, d.size)
        y= y.reshape(1, y.size)
# 教師データがone-hot-vectorの場合、正解ラベルのインデックスに変換
        ifd.size== y.size:
        d=d.argmax(axis=1)batch_size=y.shape[0]
return-np.sum(np.log(y[np.arange(batch_size), d] + 1e-7)) / batch_size
```

Section4) 勾配降下法

•勾配降下法

深層学習の目的

→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勾配降下法を利用してパラメータを最適化すること。

該当するソースコードを探してみよう。

解答

```
  grad = backward(x, d, z1, y)
  for key in ('W1', 'W2', 'b1', 'b2'):
     network[key]  -= learning_rate * grad[key]
```

パラメータを発見する

学習率の値によって学習の効率が大きく異なる

学習率

学習率が大きすぎた場合、最小値にいつまでもたどり着かず発散してしまう。

学習率が小さい場合発散することはないが、小さすぎると収束するまでに時間がかかってしまう。

勾配降下法の学習率の決定、収束性向上のためのアルゴリズムについて複数の論文が公開され、よく利用されている。

•Momentum

•AdaGrad

•Adadelta

•Adam

•確率的勾配降下法

確率的勾配降下法のメリット

•データが冗長な場合の計算コストの軽減

•望まない局所極小解に収束するリスクの軽減

•オンライン学習ができる

確認問題

オンライン学習とは何か2行でまとめよ

　オンライン学習とは学習データが入ってくるたびにその都度、新たに入ってきたデータのみを使って学習を行うもの。そのデータの学習で今あるモデルのパラメータを随時更新していく

•ミニバッチ勾配降下法

ミニバッチ勾配法のメリット

•確率的勾配降下法のメリットを損なわず、計算機の計算資源を有効利用できる→CPUを利用したスレッド並列化やGPUを利用したSIMD並列化

確認テスト

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

この数式の意味を図に書いて説明せよ

【数値微分】

プログラムで微小な数値を生成し擬似的に微分を計算する一般的な手法

数値微分のデメリット

順伝播の計算を繰り返し行う必要があり負荷が大きい

→誤差逆伝播法を利用する

Section5) 誤差逆伝播法

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。

確認テスト

誤差逆伝播法では不要な再帰的処理を避ける事が出来る。既に行った計算結果を保持しているソースコードを抽出せよ。

確認テスト

2つの空欄に該当するソースコードを探せ

解答

delta2 = functions.d_mean_squared_error(d, y)

delta2 = functions.d_mean_squared_error(d, y)

grad['W2'] = np.dot(z1.T, delta2)

DAY2

Section1) 勾配消失問題について全体像（前回の流れと課題全体像のビジョン）

勾配消失問題の復習

誤差逆伝播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。

1-1活性化関数

活性化関数: シグモイド関数

```
def   sigmoid(x):
        return 1/(1 + np.exp(-x))
```

サンプルコード数式0 ~ 1の間を緩やかに変化する関数で、ステップ関数ではON/OFFしかない状態に対し、信号の強弱を伝えられるようになり、予想ニューラルネットワーク普及のきっかけとなった。課題大きな値では出力の変化が微小なため、勾配消失問題を引き起こす事があった。

勾配消失の解決方

•活性化関数の選択

•重みの初期値設定

•バッチ正規化

活性化関数:ReLU関数

```
def relu(x):
        return np.maximum(0, x)
```

サンプルコード数式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もたらしている。

1-2初期値の設定方法

重みの初期値設定-Xavier

Xavierの初期値を設定する際の活性化関数

ReLU関数

シグモイド（ロジスティック）関数

双曲線正接関数

初期値の設定方法

n重みの要素を、前の層のノード数の平方根で除算した値

重みの初期値設定-He

Heの初期値を設定する際の活性化関数

Relu関数

初期値の設定方法

n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対し√2をかけ合わせた値

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。

解答

→ すべての値が同じ値で伝わるためパラメータのチューニングが行われなくなる

1-3バッチ正規化

バッチ正規化とは

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

バッチ正規化の使い所とは

活性化関数に値を渡す前後に、バッチ正規化の処理を孕んだ層を加える

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。

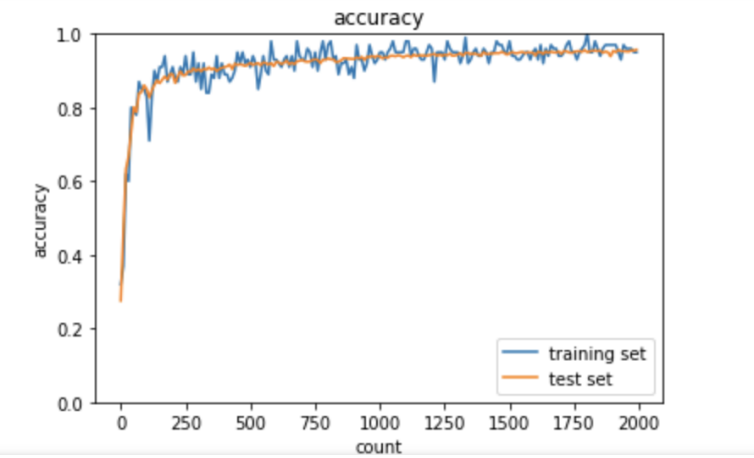
1. 計算の高速化

2. 勾配消失が起きづらくなる

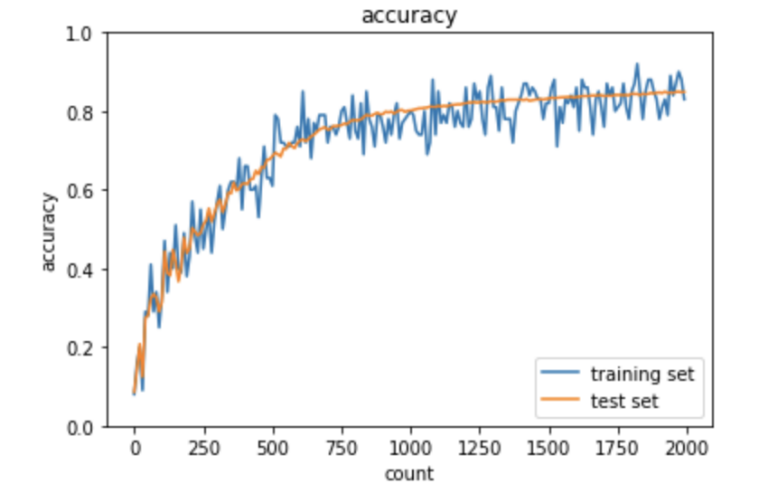
Jupyter

※ 結果のグラフ化、見える化が大事。感覚をつかみやすい

ReLU / Xavier



Sigmoid / He



2-1モメンタム

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する方法

モメンタムのメリット

・局所的最適解にはならず、大域的最適解となる。

・谷間についてから最も低い位置(最適値)にいくまでの時間が早い。

2-2AdaGrad

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する方法

AdaGradのメリット

・勾配の緩やかな斜面に対して、最適値に近づける

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事があった。

2-3RMSProp

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する誤差をパラメータで微分したものと学習率の積を減算する方法

RMSPropのメリット

・局所的最適解にはならず、大域的最適解となる。・ハイパーパラメータの調整が必要な場合が少ない

2-4Adam

AdamAdamとは ？

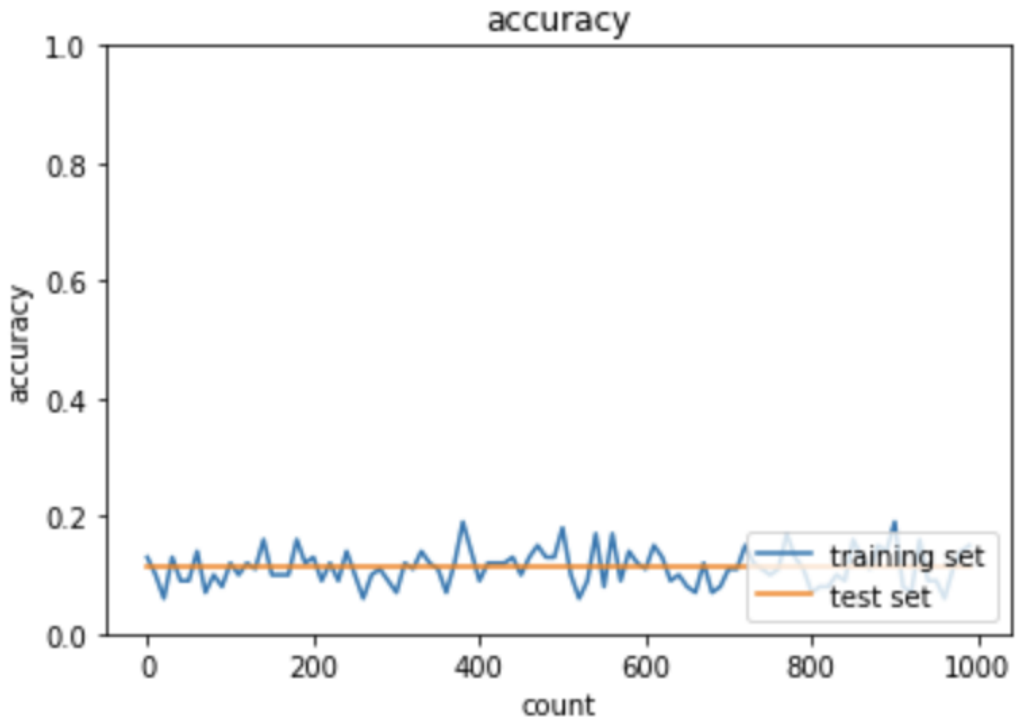
・モメンタムの、過去の勾配の指数関数的減衰平均・RMSPropの、過去の勾配の2乗の指数関数的減衰平均上記をそれぞれ孕んだ最適化アルゴリズムである。

Adamのメリットとは ？

・モメンタムおよびRMSPropのメリットを孕んだアルゴリズムである。

Jupiter演習

SGD

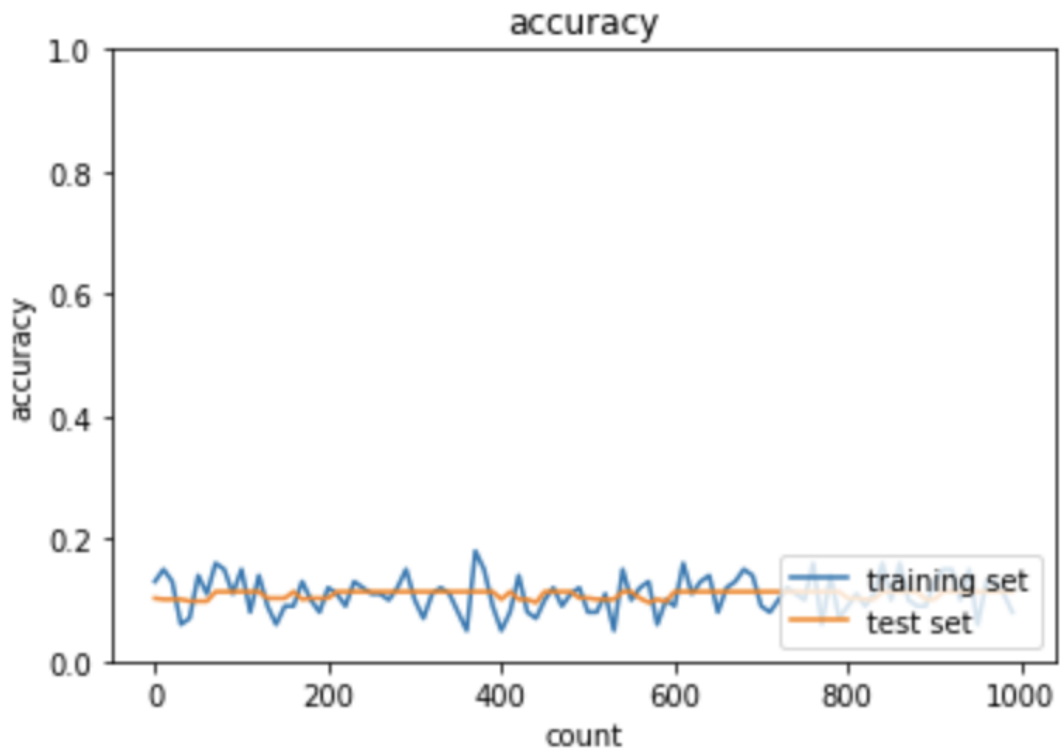


モメンタム

※ パラメータとしては0.5～0.9くらいしか設定されているのを見たことがない

慣性

momentum = 0.9



MomentumをもとにAdaGradを作ってみよう

変更しよう

if i == 0:

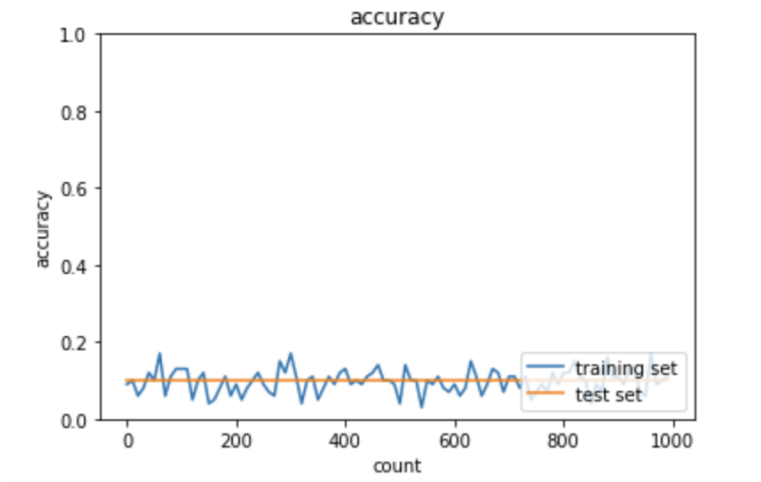
  h[key] = np.full_like(network.params[key], 1e-4)

else:

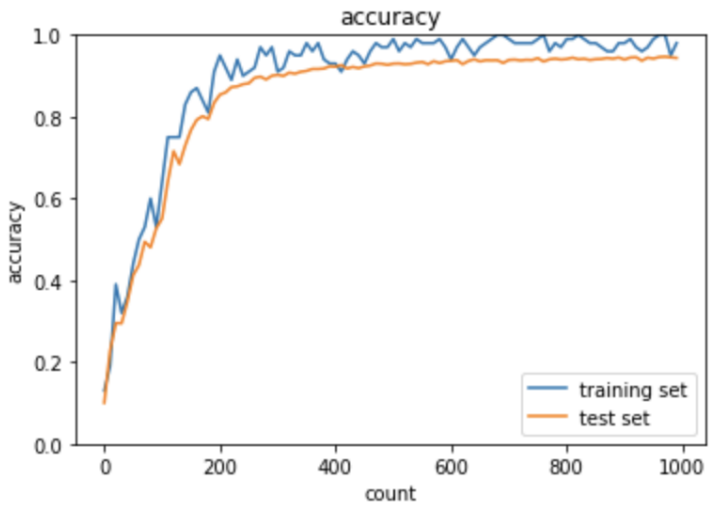
  h[key] += np.square(grad[key])

h[key] = momentum * h[key] - learning_rate * grad[key]

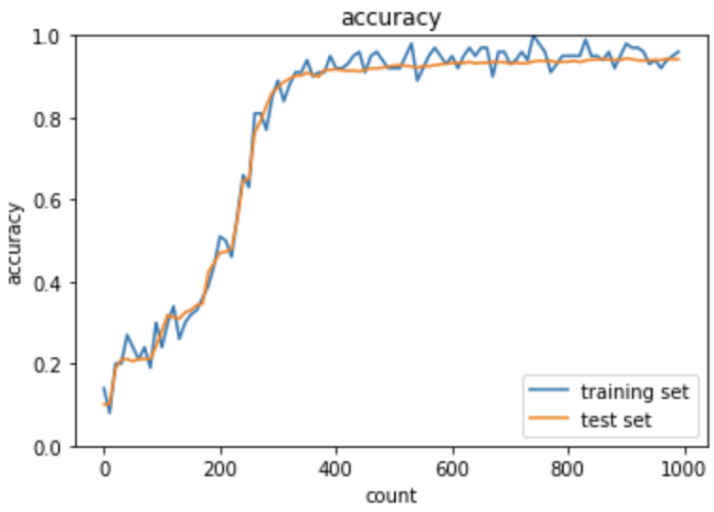
network.params[key] -= learning_rate * grad[key] / (np.sqrt(h[key]))



RSMProp



Adam(実装例)


accuracy
1.0
0.8
0.6
0.4
0.2
0.0
accuracy
0
200
400
600
800
1000
count
training set
test set

Section3)過学習について全体像（前回の流れと課題全体像のビジョン）

過学習の復習

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

•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

ネットワークの自由度(層数、ノード数、パラメータの値.)が高い特定の訓練サンプルに対して、特化して学習する。

3-1 L1正則化、L2正則化

正則化

ネットワークの自由度(層数、ノード数、パラメータの値)を制約すること正則化手法を利用して過学習を抑制する

正則化手法について

•L1正則化、L2正則化

•ドロップアウト

p = 1の場合、L1正則化と呼ぶ。p = 2の場合、L2正則化と呼ぶ。

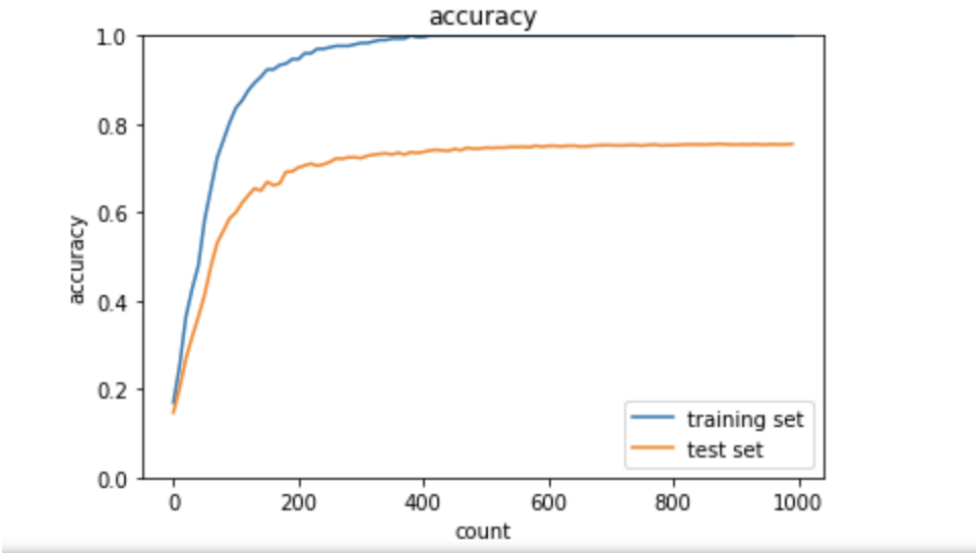
3-2ドロップアウト

過学習の課題・ノードの数が多いドロップアウトとは ？ランダムにノードを削除して学習させること

メリットとして・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と解釈できる

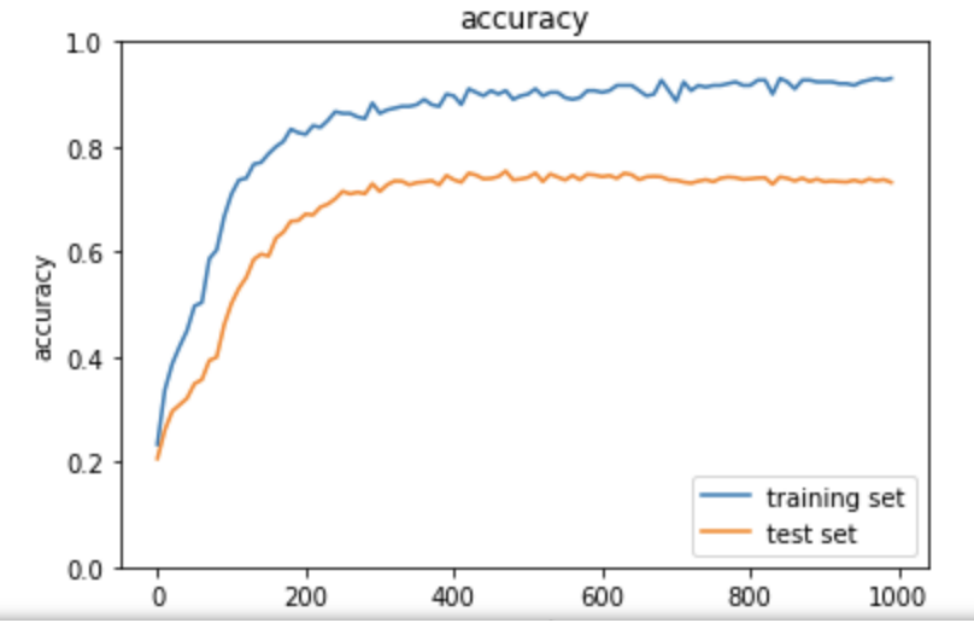
ソースコード演習

overfitting



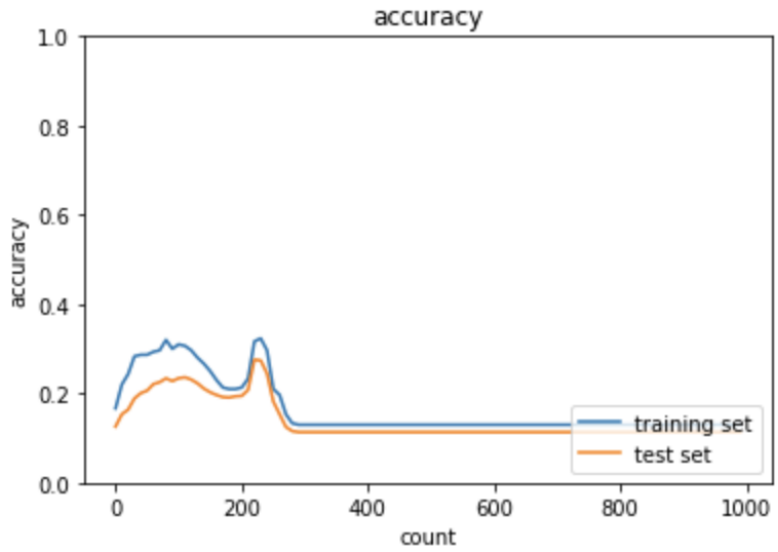
Weight decay(L2)

weight_decay_lambda = 0.1



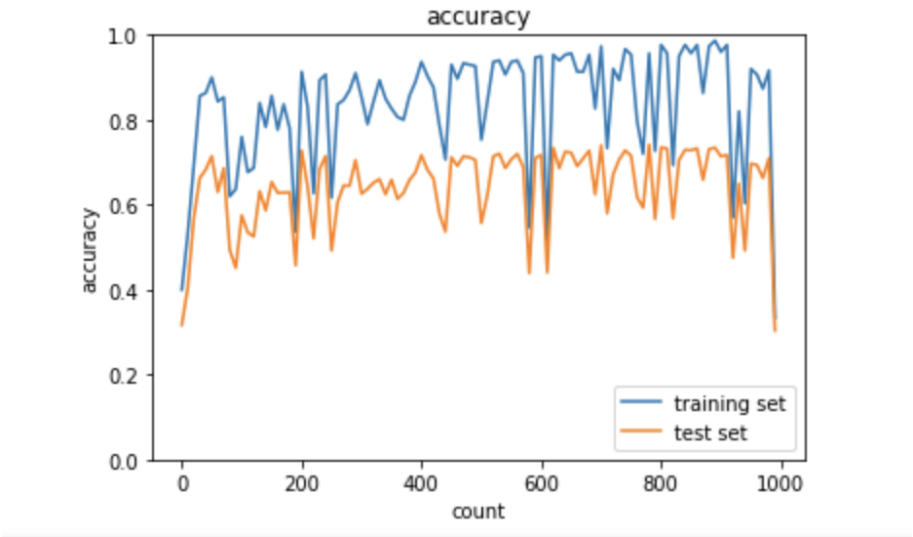
Weight decay(L2)

weight_decay_lambda = 0.4



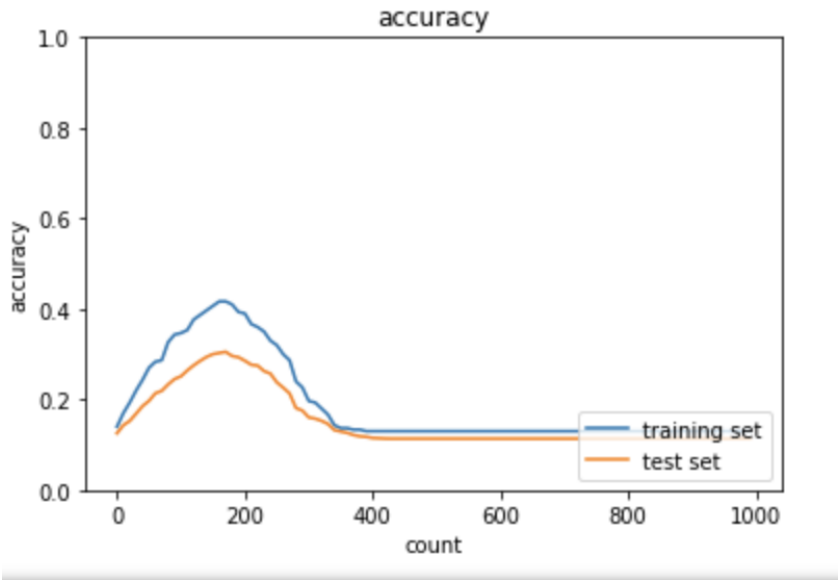
weight_decay_lambda が小さい → 過学習が抑制されない weight_decay_lambdaが大きい → 過学習はされないが精度が出ない

Weight decay(L1)

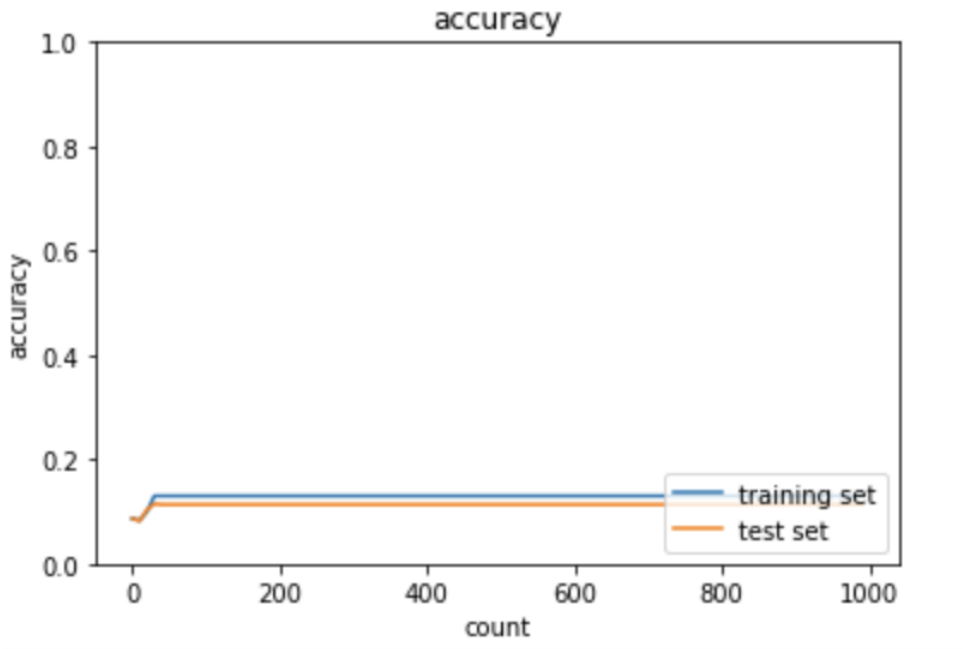


Jupyter演習(Dropout)

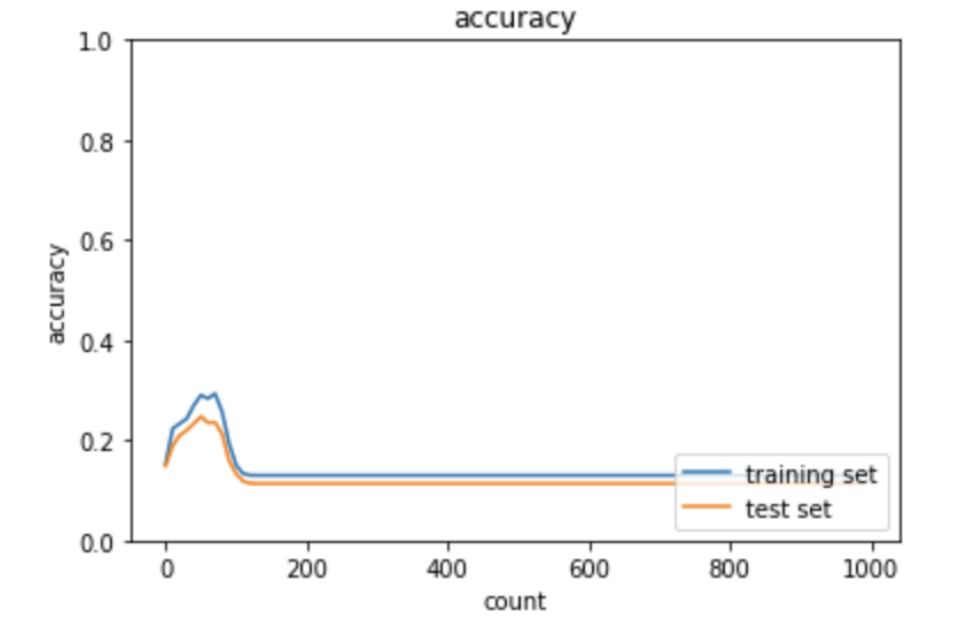
Dropout



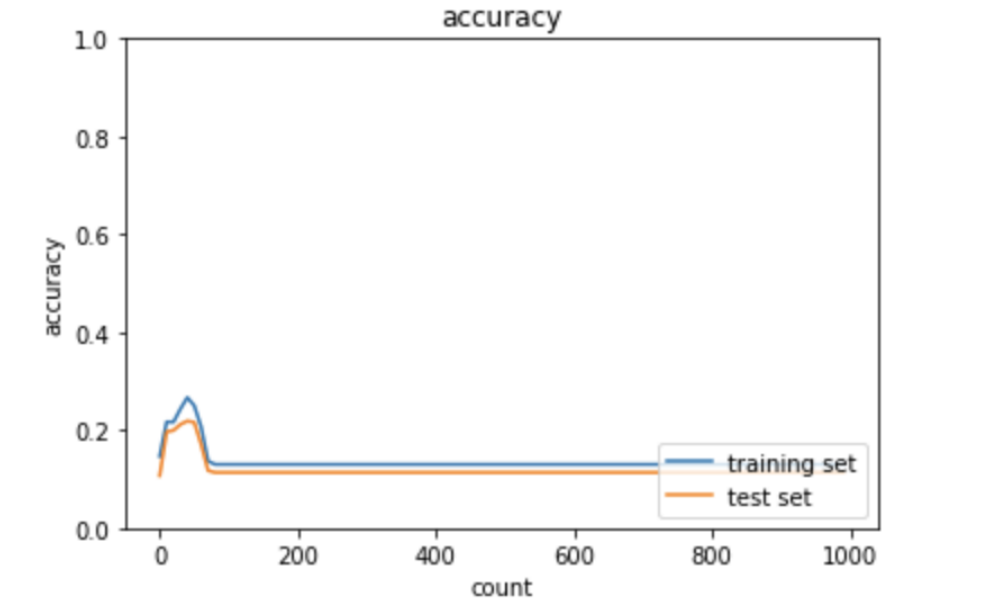
Dropout(ratio=0.4)



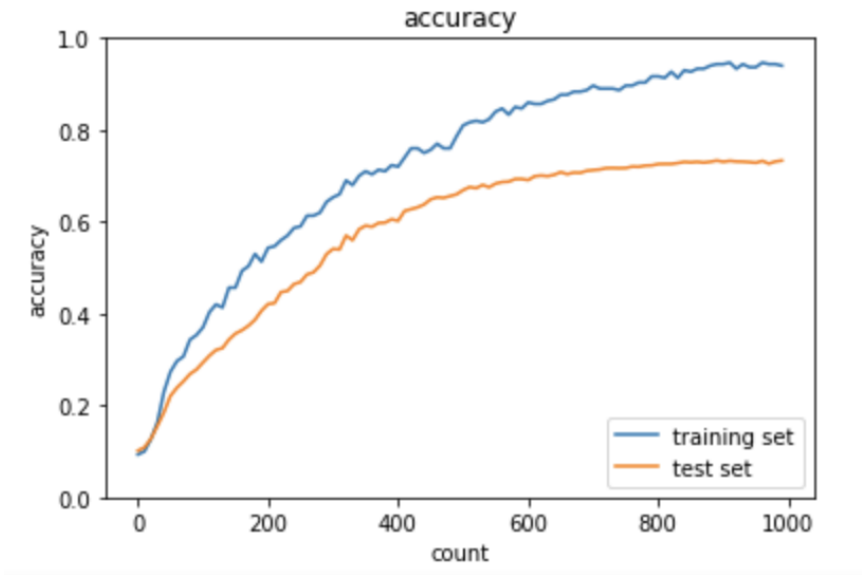
Dropout(ratio=0.02)



Dropout(Adam, ratio=0.15)



Dropout + L1



■畳み込みニューラルネットワークについて

Section4)畳み込みニューラルネットワークの概念全体像（CNNの構造図）

4-1畳み込み層

4-1-1バイアス

畳み込み層では、画像の場合、縦、横、チャンネルの3次元のデータをそのまま学習し、次に伝えることができる。結論:3次元の空間情報も学習できるような層が畳み込み層である。

概念

入力画像→フィルター(全結合でいう重み)→出力画像

例　3×3 + 4×1 + 4×2 + 0×8 + 8×7 + 9×5 + 0×5 + 4×4 + 3×1 = 141S4) 1-182

4-1-2パディング

空欄

固定のデータは0以外でも可能。

入力画像と出力画像を合致させるために入力画像に固定データで枠をつけるイメージ

4-1-3ストライド

演算式の図を挿入

ずらす幅

4-1-4チャンネル

複数の入力値と複数のフィルター　CNNの味噌

4-2プーリング層

入力画像→対象領域のMax値または、平均値を取得→出力値

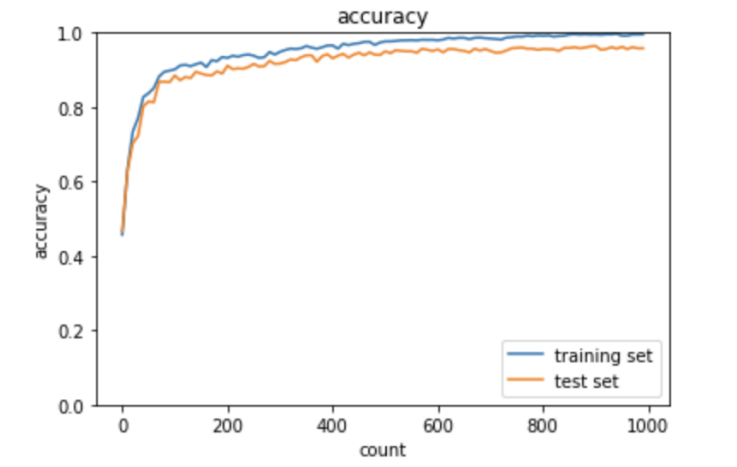
確認テスト

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。

→ 7×7

Jupyter演習

Simple Convolution Network



Section5)最新のCNN

5-1 AlexNe

過学習を防ぐ施策・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