ニューラルネットワークの学習の仕組み

高慎之助

前提知識

- 第3章までのニューラルネットワークについての知識
- 単純な微分の計算方法
- 行列の計算方法

著者

- 斎藤康毅

1984年長崎県対馬市生まれ。

東京工業大学卒、東京大学大学院修士課程修了。

現在は、株式会社Preferred Networksにて人工知能に関する研究開発に従事。

外部の既成品(ライブラリやツールなど)は極力使わずに、ディープラーニングを作り上げることで，ディープラーニングについてより深く理解してもらいたいと思っています。

目次

1節　そもそも学習とは？

2節　学習をするための事前準備

2.1　データを用意する　　2.2　損失関数を決める

3節　ニューラルネットワークが学習をする5つのステップ

3.1　ミニバッチを取得する　　3.3　勾配を算出する

3.4　パラメータを更新する　3.5　上記3つを繰り返す

実践　実際の学習の流れをコードを書いて学ぼう

・学習をするための事前準備は何をすればいいのか理解する
・ニューラルネットワークの学習の4ステップを覚える
・学習のアルゴリズムを数学的に理解する

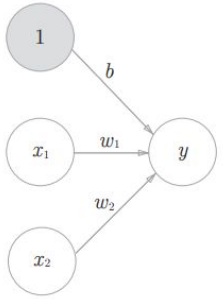
学習とは何か？

そもそも学習とは？

訓練データから最適な重みパラメータと，バイアスを自動で獲得すること

モデルの出力値と，正解ラベルとの誤差が小さいモデルを作り，ある問題を解くため

重みW,バイアスb
(乱数で初期化)



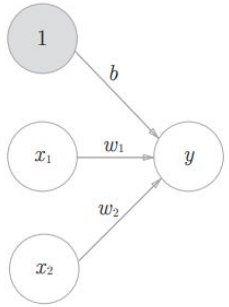
0.3の誤差

0.7 ⟷ 1.0

モデルの出力値　　正解ラベル

この誤差を小さくするように重みw，バイアスbを最適化する

重みW,バイアスb
(学習済み)



0.0の誤差

1.0 ⟷ 1.0

モデルの出力値　　正解ラベル

学習をするための事前準備

1.データを用意する

機械学習で行っているのは，データからあるパターンを見つけることだから

2.損失(誤差)関数を決める

学習させたモデルを定量的に評価するときに使う指標

モデルの良さ，悪さを評価するため

1.データを用意する(1)

種類は？

訓練(教師)データ　学習に使用するデータ

テストデータ　その訓練したモデルの実力を評価するデータ

なんで2種類に分けるの？

特定のデータだけに過度に対応するモデルを作らないようにするため

過学習
(overfitting)

例えば，手書き数字認識システムを作りたい場合

とある3人(A,B,Cさん)に数字を書いてもらいそれをデータとする.

Aさんが書いた数字だけを使って学習させると,Aさんのクセのある文字だけを学習している

可能性がある

1.データを用意する(2)

量は？

ニューラルネットワークの重みパラメーターの数に対して、最低限その10倍以上の訓練データ量が必要

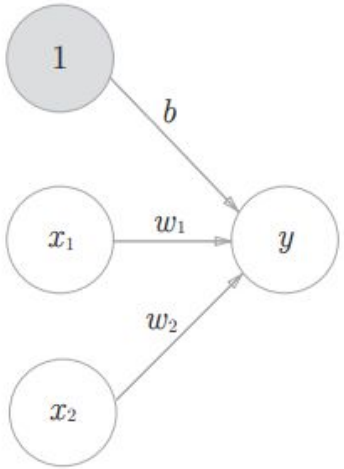
バーニーおじさんのルール

・モデルのパラメータ数の10倍のデータ数が必要という**経験則**

・機械学習のモデルの学習に必要なデータ量を測る明確な数字はない

・目安程度にはなる

例



調整するパラメータの数は3つなので，最低限30個の訓練データは必要

2.損失(誤差)関数を決める(1)

2乗和誤差

・モデルの出力値と，正解となるラベルの値の誤差を2乗し，その総和を求める関数

・最も有名な損失関数

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

交差エントロピー誤差

・正解ラベルの値が1に対応するモデルの出力の自然対数を求める関数

$$E = -\sum_k t_k \log y_k$$

2.損失(誤差)関数を決める(2)

例：MNISTデータセットを使って手書き数字認識器を作る
訓練データ：60000枚の28×28ピクセルのモノクロ画像
テストデータ：10000枚の28×28ピクセルのモノクロ画像
それぞれの画像データに対して，0,1,2,…,9のラベルが与えられている

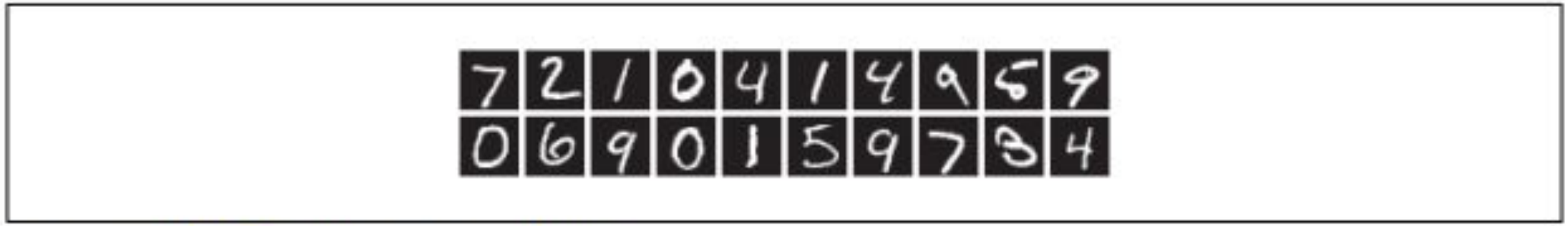


図3-24　MNIST画像データセットの例

2.損失(誤差)関数を決める(3)

```
>>> # 「2」を正解とする
>>> t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
>>>
>>> # 例 1：「2」の確率が最も高い場合（0.6）
>>> y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]
```

教師データ

ニューラルネットワークの出力

2乗和誤差

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

$$= \frac{1}{2}\{(0.1-0)^2 + (0.05-0)^2 + (0.6-1)^2 + \cdots + (0.0-0)^2\} \quad \fallingdotseq 0.0975$$

2.損失(誤差)関数を決める(4)

```
>>> # 「2」を正解とする
>>> t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
>>>
>>> # 例 1：「2」の確率が最も高い場合（0.6）
>>> y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]
```

教師データ

ニューラルネットワークの出力

交差エントロピー誤差

$$E = -\sum_k t_k \log y_k \qquad = 1 * \log 0.6 \qquad \fallingdotseq 0.51082$$

質疑応答

目次

1節　そもそも学習とは？

2節　学習をするための事前準備

2.1　データを用意する　　2.2　損失関数を決める

3節　ニューラルネットワークが学習をする5つのステップ

3.1　ミニバッチを取得する　　3.3　勾配を算出する

3.4　パラメータを更新する　3.5　上記3つを繰り返す

実践　実際の学習の流れをコードを書いて学ぼう

・学習をするための事前準備は何をすればいいのか理解する
・ニューラルネットワークの学習の4ステップを覚える
・学習のアルゴリズムを数学的に理解する

損失関数を最小化するには？

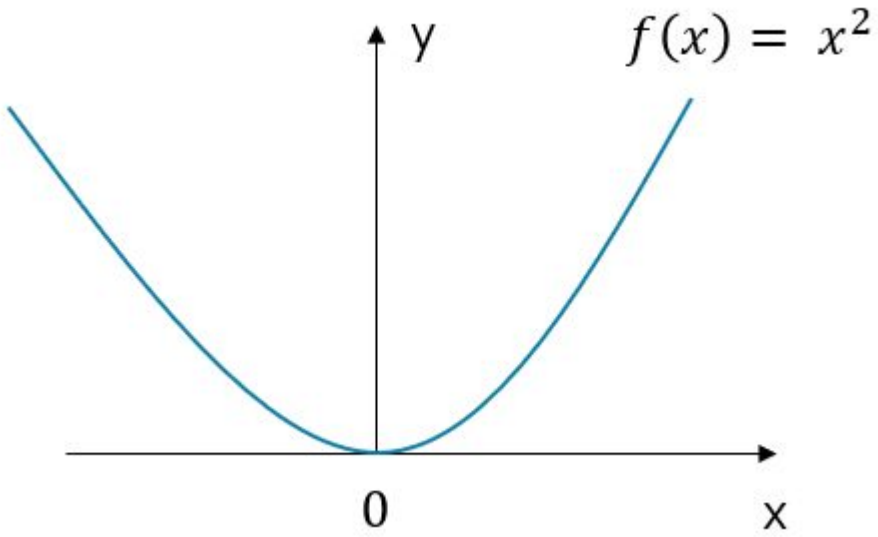
問題: $f(x) = x^2$

が最小値を取るようなxの値を求めよ

$$f'_{(x)} = 2x$$

$f'_{(x)} = 0$ となるようなxの値,

すなわち $X = 0$



ニューラルネットワークが学習をする 5つのステップ

第4章では，確率的勾配降下法(stochastic grandient descent)(通称SGD)を扱う

機械学習の最適化問題でよく使われる手法.

ニューラルネットワークの学習でよく用いられる.

1.ミニバッチを取得する(データを取得する)
2.ミニバッチを入力値として，現在のパラメータwによる出力値と，損失関数の値を求める
3.現在のパラメータWの勾配を算出する
4.パラメータを更新する
5.上記の4ステップを繰り返す

1.ミニバッチを取得する

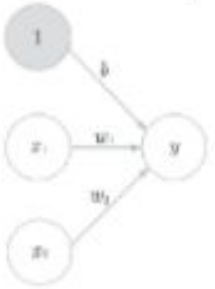
ミニバッチとは？

訓練データの中からランダムに選ばれたデータのこと

膨大な訓練データに対して，損失関数の和を求める手間を省くため

2.現在のパラメータwによる出力値と，損失関数の値を求める

重みW,バイアスb
(乱数で初期化)



$$E = \frac{1}{2}(1.0 - 0.2)^2 = 0.32$$

0.2 ↔ 1.0

出力値　　正解

3.勾配を算出する(1)

勾配とは？

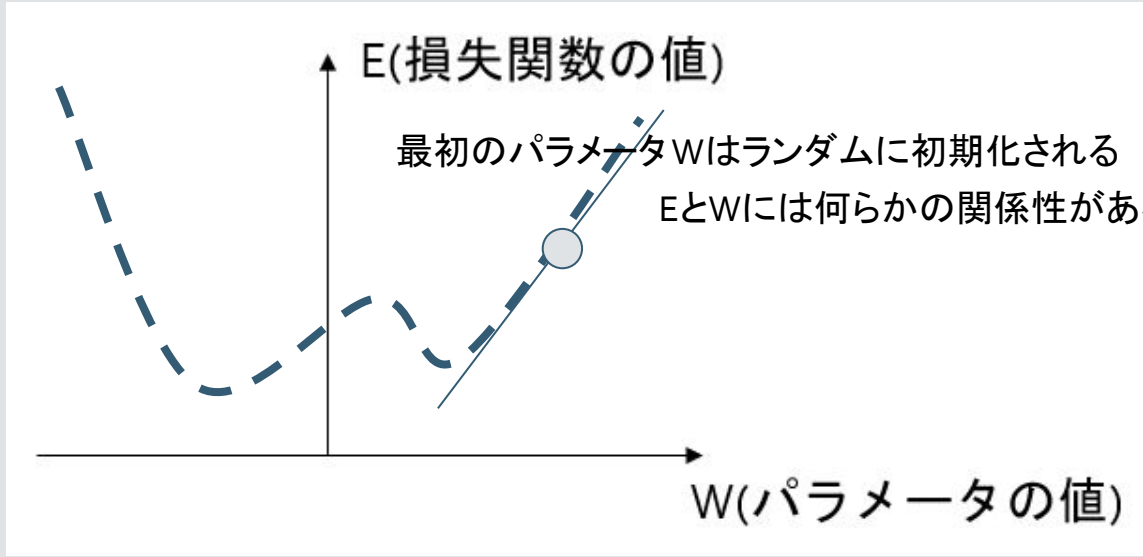
損失関数の値を最も減らすパラメータの方向を示すベクトル(傾き)

損失関数の値を最も減らす方向を知り，重みパラメータとバイアスの最適化をするため

求め方：**誤差関数をモデルのパラメータで数値微分する**

損失関数の値を小さくするには，
微分した結果が**正**の場合：
現在よりも**パラメータの値を小さくする**
微分した結果が**負**の場合：
現在よりも**パラメータの値を大きくする**

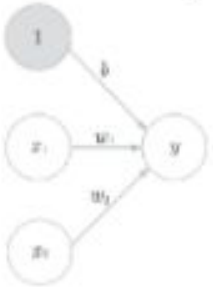
1次元だけの重みを横軸に取ったもの



3.勾配を算出する(2)

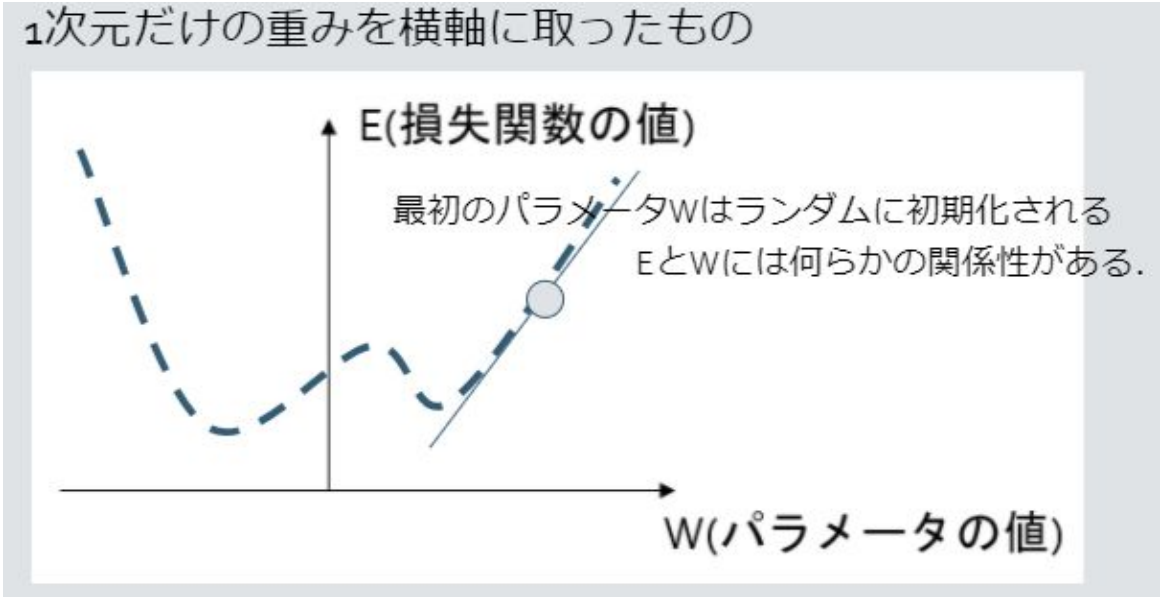
注意点

重みW,バイアスb
(乱数で初期化)



$$E = \frac{1}{2}(1.0 - 0.2)^2 = 0.32$$

0.2 ↔ 1.0
出力値　正解



b, w1, w2が決まることによって,
損失関数の値が1つに定まる
・変数が3つ

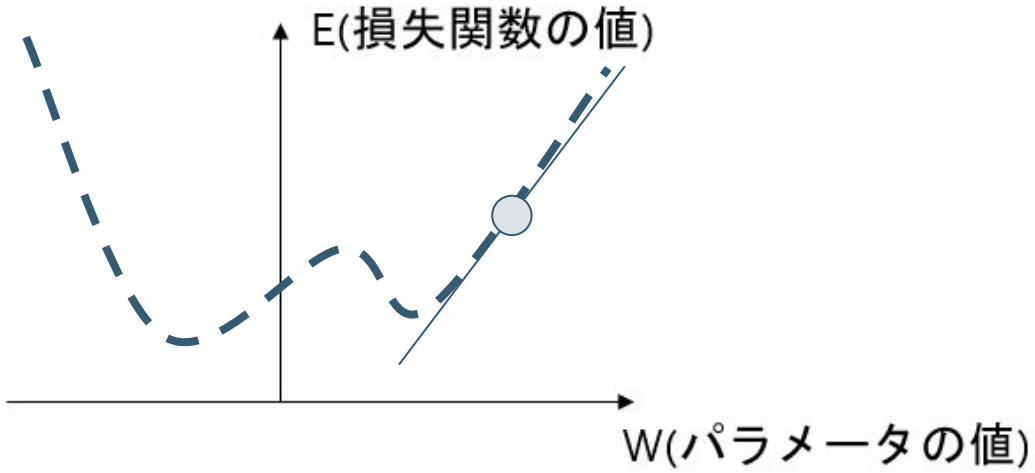
どの変数に対して
微分をするのか決める必要がある

Wが決まることによって,
損失関数の値が1つに定まる
・変数が1つ

偏微分を使う


予備校のノリで学ぶ
偏微分とは何か
$\frac{\partial}{\partial x} f(x,y)$
$\frac{\partial}{\partial y} f(x,y)$

4.パラメータを更新する



$$x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}$$

$$x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$$

学習率

1回の学習で，どれだけ学習すべきか，どれだけパラメータを更新するかを決める指標

局所最適解に陥らないようにするため．

5.今までの4ステップを繰り返す

1.ミニバッチを決める
2.現在のパラメータwによる出力値と，損失関数の値を求める
3.勾配を算出する
4.パラメータを更新する

質疑応答

目次

1節　そもそも学習とは？

2節　学習をするための事前準備

2.1　データを用意する　　2.2　損失関数を決める

3節　ニューラルネットワークが学習をする5つのステップ

3.1　ミニバッチを取得する　　3.3　勾配を算出する

3.4　パラメータを更新する　3.5　上記3つを繰り返す

実践　実際の学習の流れをコードを書いて学ぼう

・学習をするための事前準備は何をすればいいのか理解する
・ニューラルネットワークの学習の4ステップを覚える
・学習のアルゴリズムを数学的に理解する

環境構築

Google Colaboratoryを使用する

Google Colaboratoryとは？

・教育・研究機関向けの機械学習の普及を□的としたGoogleの 研究プロジェクトの□つ

・Googleアカウントさえあれば、クラウド上で機械学習のコードを実□可能で、GPUも利□できる。

Googleアカウント，Google Chromeがあれば使用可能

Google Colaboratoryのサイト

https:colab.research.google.com/

ノートブックを新規作成する

ファイル
↓
ノートブックを新規作成する

学習アルゴリズムの実装

```
import sys, os
sys.path.append(os.pardir)
from common.functions import *
from common.gradient import numerical_gradient

class TwoLayerNet:

    def __init__(self, input_size, hidden_size, output_size,
                 weight_init_std=0.01):
        # 重みの初期化
        self.params = {}
        self.params['W1'] = weight_init_std * \
                            np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * \
                            np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)

    def predict(self, x):
        W1, W2 = self.params['W1'], self.params['W2']
        b1, b2 = self.params['b1'], self.params['b2']

        a1 = np.dot(x, W1) + b1
        z1 = sigmoid(a1)
        a2 = np.dot(z1, W2) + b2
        y = softmax(a2)

        return y

    # x:入力データ, t:教師データ
    def loss(self, x, t):
        y = self.predict(x)

        return cross_entropy_error(y, t)

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        t = np.argmax(t, axis=1)

        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy

    # x:入力データ, t:教師データ
    def numerical_gradient(self, x, t):
        loss_W = lambda W: self.loss(x, t)

        grads = {}
        grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
        grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
        grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
        grads['b2'] = numerical_gradient(loss_W, self.params['b2'])

        return grads
```

二層のニュートラルネットワーク
をーつのクラスとしてTowLayerNet

```
class TwoLayerNet:

    def __init__(self, input_size, hidden_size, output_size,
                 weight_init_std=0.01):
        # 重みの初期化
        self.params = {}
        self.params['W1'] = weight_init_std * \
                            np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * \
                            np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)

    def predict(self, x):
        W1, W2 = self.params['W1'], self.params['W2']
        b1, b2 = self.params['b1'], self.params['b2']

        a1 = np.dot(x, W1) + b1
        z1 = sigmoid(a1)
        a2 = np.dot(z1, W2) + b2
        y = softmax(a2)

        return y
```

初期化

認識

TowLayerNetクラスで使用するクラス変数

params：ニューラルネットワークのパラメータを保持するディクショナリ変数（インスタンス変数）。
params['W1'] は 1 層目の重み、params['b1'] は 1 層目のバイアス。
params['W2'] は 2 層目の重み、params['b2'] は 2 層目のバイアス。

```
# x:入力データ, t:教師データ
def loss(self, x, t):
    y = self.predict(x)

    return cross_entropy_error(y, t)

def accuracy(self, x, t):
    y = self.predict(x)
    y = np.argmax(y, axis=1)
    t = np.argmax(t, axis=1)

    accuracy = np.sum(y == t) / float(x.shape[0])
    return accuracy

# x:入力データ, t:教師データ
def numerical_gradient(self, x, t):
    loss_W = lambda W: self.loss(x, t)

    grads = {}
grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
grads['b2'] = numerical_gradient(loss_W, self.params['b2'])

return grads
```

損失関数の値を求める

認識制度を求める

重みパラメータに対する勾配を求める

Grads:勾配を保持するディクショナリ変数
grads['W1'] は 1 層目の重みの勾配、grads['b1'] は 1 層目のバイアスの勾配。
grads['W2'] は 2 層目の重みの勾配、grads['b2'] は 2 層目のバイアスの勾配。

ミニバッチ学習

```
import numpy as np
from dataset.mnist import load_mnist
from two_layer_net import TwoLayerNet

(x_train, t_train), (x_test, t_test) = \
    load_mnist(normalize=True, one_hot_label=True)

train_loss_list = []

# ハイパーパラメータ
iters_num = 10000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

for i in range(iters_num):
    # ミニバッチの取得
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 勾配の計算
    grad = network.numerical_gradient(x_batch, t_batch)
    # grad = network.gradient(x_batch, t_batch) # 高速版!

    # パラメータの更新
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    # 学習経過の記録
    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)
```

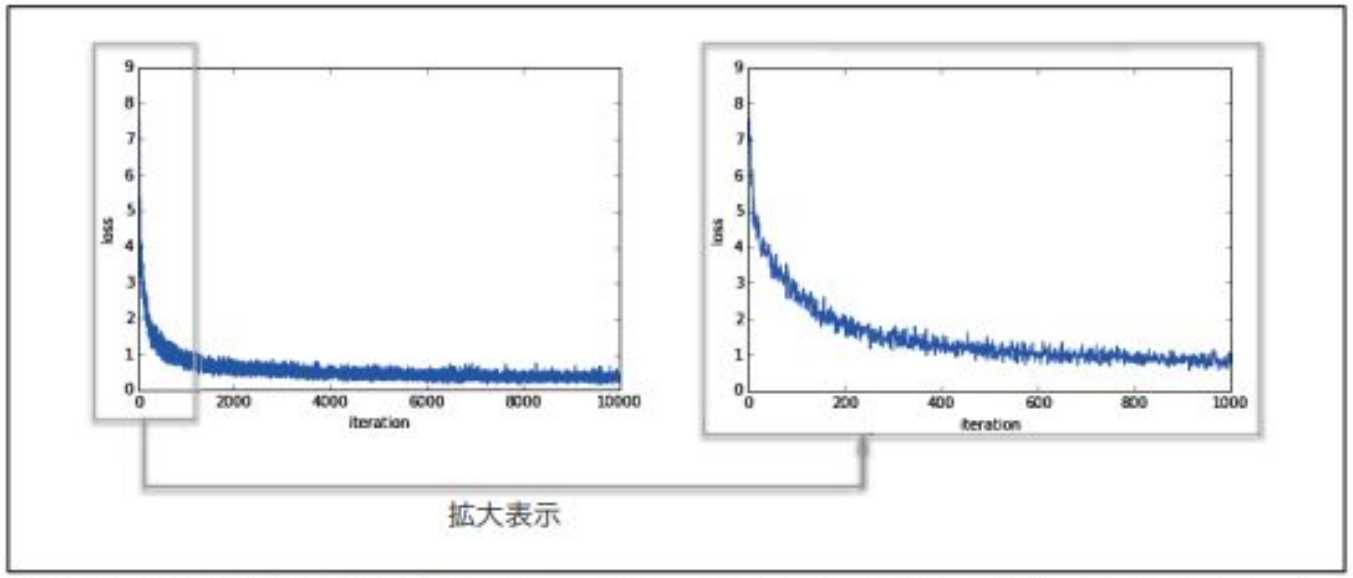


図 4-11　損失関数の推移：左図は 10,000 イテレーションまでの推移、右図は 1,000 イテレーションまでの推移

他のデータセットにも同じ程度の実力を発揮できるかどうかは定かではない

テストデータで評価

```
import numpy as np
from dataset.mnist import load_mnist
from two_layer_net import TwoLayerNet

(x_train, t_train), (x_test, t_test) = \
    load_mnist(normalize=True, one_hot_label=True)

train_loss_list = []
train_acc_list = []
test_acc_list = []
# 1エポックあたりの繰り返し数
iter_per_epoch = max(train_size / batch_size, 1)

# ハイパーパラメータ
iters_num = 10000
batch_size = 100
learning_rate = 0.1

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

for i in range(iters_num):
    # ミニバッチの取得
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 勾配の計算
    grad = network.numerical_gradient(x_batch, t_batch)
    # grad = network.gradient(x_batch, t_batch) # 高速版!

    # パラメータの更新
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)
```

```
# 1エポックごとに認識精度を計算
if i % iter_per_epoch == 0:
    train_acc = network.accuracy(x_train, t_train)
    test_acc = network.accuracy(x_test, t_test)
    train_acc_list.append(train_acc)
    test_acc_list.append(test_acc)
    print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))
```

1 エポックごとに、すべての訓練データとテストデータに対して
認識 精度を計算して、その結果を記録

1エポックごとに計算する理由は認識制度の推移がざっくり分かればいい

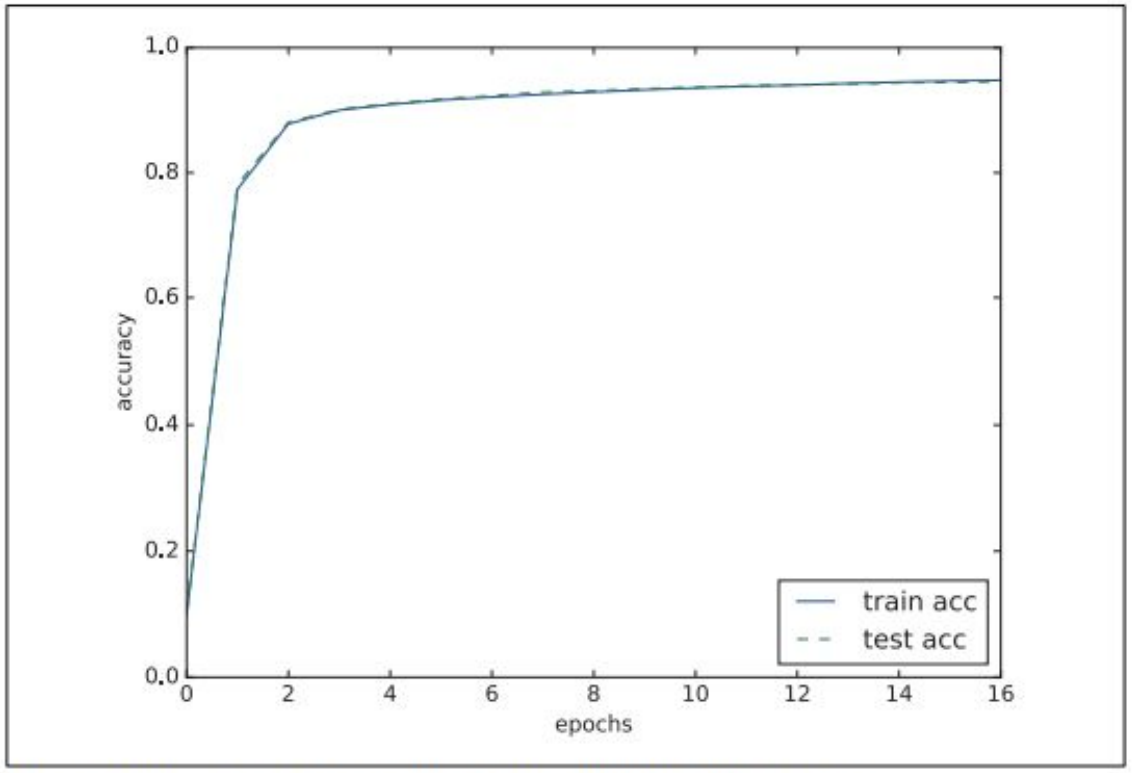


図4-12　訓練データとテストデータに対する認識精度の推移。横軸はエポック

エボックがすすむにつれて訓練データとテストデータを使って評価した認識精度は両方とも向上している

２つの認識精度には差がない

過学習が起きていない

まとめ

・学習とは，訓練データから**最適な重みパラメータと，バイアスを自動で獲得する**こと

・学習をするためには，そもそも**訓練データと，テストデータ，損失関数を準備することが必要**

・訓練データで学習を行い，学習したモデルの汎化能力をテストデータで評価する

・ニューラルネットワークの学習は，損失関数を指標として，**損失関数の値が小さくなるように，重みパラメータを更新する**

・重みパラメータを更新する際には，重みパラメータの勾配を利用して，**勾配方向に重みの値を更新する** 作業を繰り返す

まとめ

・微小な値を与えた時の差分によって微分を求めることを数値微分という

・数値微分によって，重みパラメータの勾配を求めることが出来る.

・数値微分による計算には時間がかかるが，その実装は簡単である.

次週は，第5章

第4週で扱った数値微分による勾配の計算には時間がかかる

重みパラメータの勾配の計算を効率よく行う手法である

誤差逆伝搬法についての話

ご清聴ありがとうございました.