ラピッドチャレンジ「課題レポート (深層学習 Day2)」

佐藤晴一

2021 年 5 月 31 日

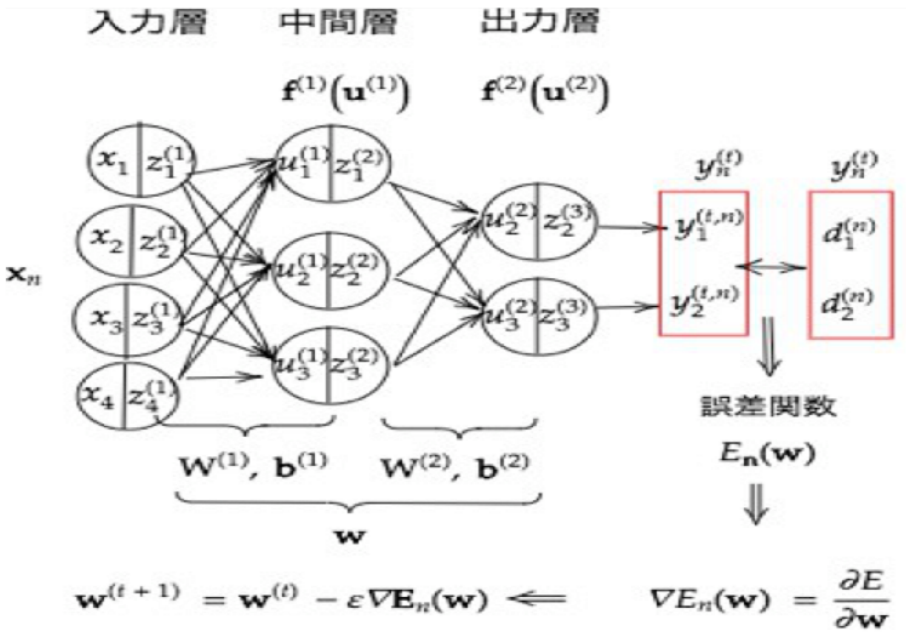
目次

第１章

勾配消失問題

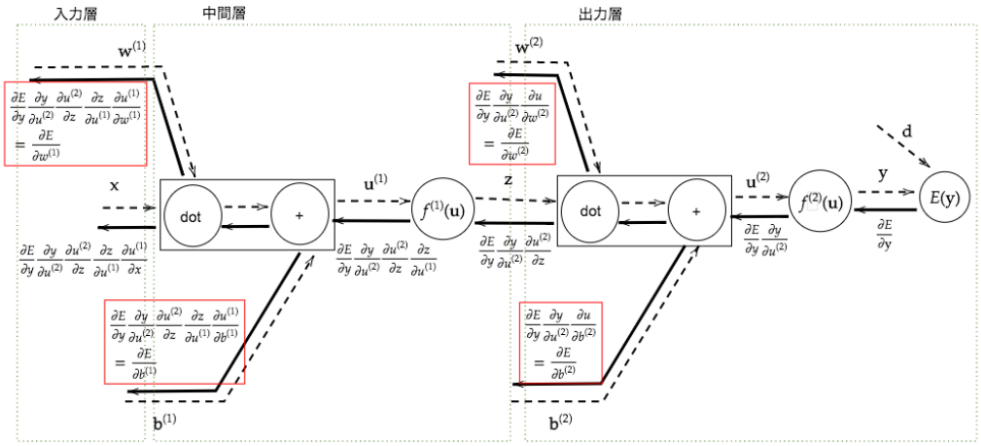
■ニューラルネットワーク全体像

1. 入力層に値を入力
2. 重み、バイアス、活性化関数で計算しながら値が伝わる
3. 出力層から値が伝わる
4. 出力層から出た値と正解値から、誤差関数を使って誤差を求める
5. 誤差を小さくするために重みやバイアスを更新する
6. 1～5 の操作を繰り返すことにより、出力値を正解値に近づけていく



■誤差逆伝播法

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。一方で、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]。



■確認テスト　連鎖律の原理を使い、dz/dx を求めよ。$z = t^2, \quad t = x + y$

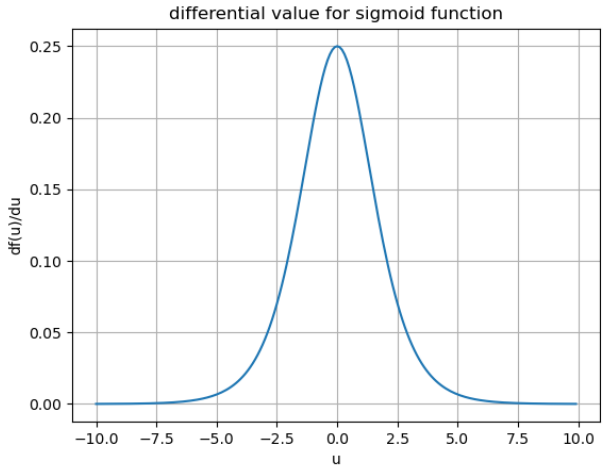
■答え

$$\frac{dz}{dx} = \frac{dz}{dt}\frac{dt}{dx} = \frac{d}{dt}(t^2)\frac{d}{dx}(x+y) = (2t) \times 1 = 2t = 2(x+y)$$

■確認テスト　シグモイド関数 $f(u)$ を微分した時、入力値が 0 の時に最大値をとる。その値？

■答え

$$\frac{d}{du}f(u) = (1 - f(u)) \cdot f(u) = 0.25 \quad (\text{Max}) \quad \text{for } u = 0$$



[*1] シグモイド関数の導関数は最大値が 0.25 であるため。1 より小さい数を掛け合わせると、どんどん小さくなり、入力層に近づくにしたがって更新量が消失していくことになる。

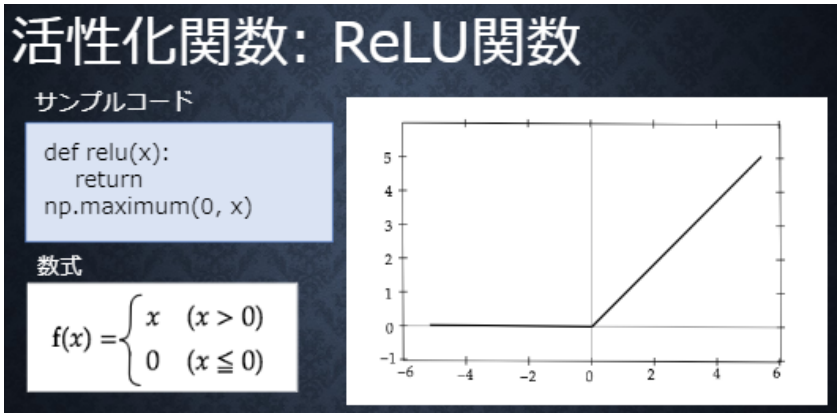
■勾配消失の解決法

1. 活性化関数の選択
2. 重みの初期値設定
3. バッチ正規化

1.1 活性化関数

先に誤差逆伝播法で説明したシグモイド関数は、$0-1$ 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。一方で、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。

そこで、導入されたのが次に示す Relu 関数である。これは、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

1.2.1 重みの初期値設定-Xavier(ザビエル)

一般的に初期化に用いられるのは、乱数であり、　この時の重みの初期値の設定方法としては、まず標準正規分布の乱数を発生させて、作りたい層における重みの要素を、前の層のノード数 n の平方根で除算した値として採用する。

```
1 # 入力層から中間層へ至るノードの重みの初期化
2 network[''W1''] = np.random.randn(input_layer_size, hidden_layer_size) / np.sqrt(
      input_layer_size)
3 # 中間層から出力層へ至るノードの重みの初期化
4 network[''W2''] = np.random.randn(hidden_layer_size, output_layer_size) / np.sqrt(
      hidden_layer_size)
```

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（tanh) のような S 字カーブの活性化関数に対して適用可能。

1.2.2 重みの初期値設定-He

次に、活性化関数が S 字カーブで与えられない（Relu 関数等）に対する初期値の設定方法 He（ヒー）について説明する。具体的には、途中までは先ほどと同様の手法であり、まずは標準席分布関数の乱数を発生させる。その次に、重みの要素を、前の層のノード数 n の平方根で除算した値に対し $\sqrt{2}$ をかけ合わせた値として採用する。

```
# 入力層から中間層へ至るノードの重みの初期化
network[''W1''] = np.random.randn(input_layer_size, hidden_layer_size) / np.sqrt(
    input_layer_size) * sqrt(2)
# 中間層から出力層へ至るノードの重みの初期化
network[''W2''] = np.random.randn(hidden_layer_size, output_layer_size) / np.sqrt(
    hidden_layer_size) * sqrt(2)
```

He の初期値を設定する際の活性化関数

Relu 関数

■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。

■答え

初期値がゼロの場合は、全ての重みの値が、均一に更新されるため多数の重みを持つ意味がなくなる。結果として、重みをゼロで初期化すると正しい学習が行えず、ニューラルネットワークの学習モデルとして不適となる。

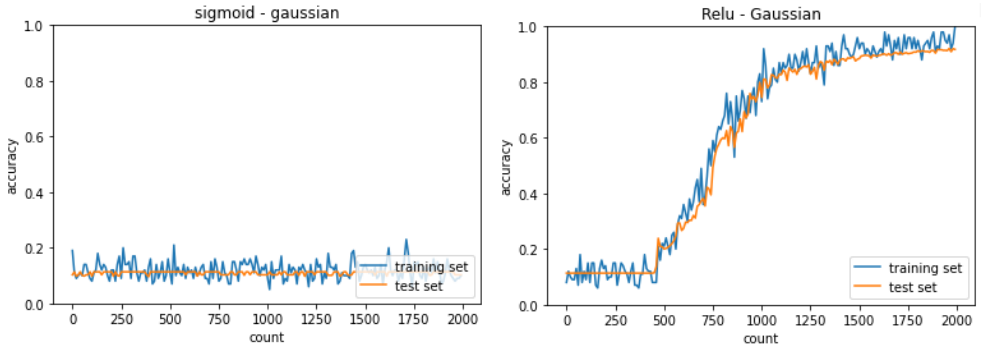
■実装演習成果（勾配消失問題の解決）

Listing 1.1 sigmoid - Xavier

```
# データの読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True, one_hot_label=True)

printデータ読み込み完了("")

network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20], output_size=10,
    activation='sigmoid', weight_init_std='Xavier')

iters_num = 2000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1

train_loss_list = []
```

```
accuracies_train = []
accuracies_test = []

plot_interval=10

for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    # 勾配
    grad = network.gradient(x_batch, d_batch)

    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(x_batch, d_batch)
    train_loss_list.append(loss)

    if (i + 1) % plot_interval == 0:
        accr_test = network.accuracy(x_test, d_test)
        accuracies_test.append(accr_test)
        accr_train = network.accuracy(x_batch, d_batch)
        accuracies_train.append(accr_train)

# print('Generation: ' + str(i+1) + '. 正答率トレーニング() = ' + str(accr_train))
# print(' : ' + str(i+1) + '. 正答率テスト() = ' + str(accr_test))


lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test, label="test set")
plt.legend(loc="lower right")
plt.title("sigmoid - Xavier")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
# グラフの表示
plt.show()
```

初期値（標準正規分布）



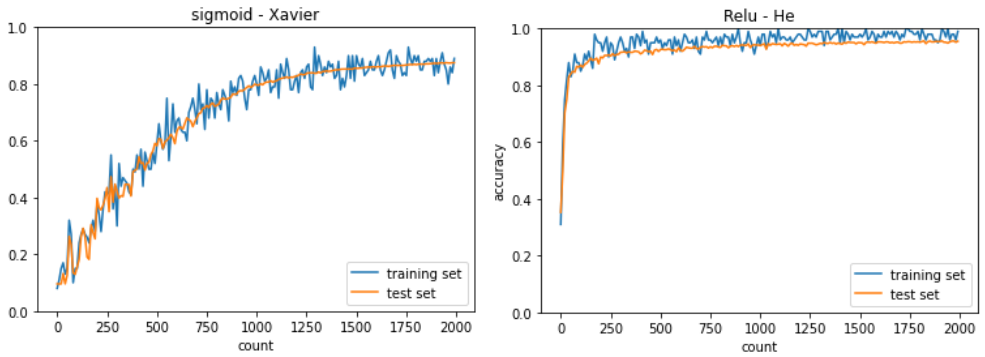
■実装演習成果（勾配消失問題の解決）

Listing 1.2　Relu - He

```
# データの読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True, one_hot_label=True)

printデータ読み込み完了("")

network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20], output_size=10,
    activation='relu', weight_init_std='He')

iters_num = 2000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10

for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    # 勾配
    grad = network.gradient(x_batch, d_batch)

    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(x_batch, d_batch)
```

```
31      train_loss_list.append(loss)
32
33      if (i + 1) % plot_interval == 0:
34          accr_test = network.accuracy(x_test, d_test)
35          accuracies_test.append(accr_test)
36          accr_train = network.accuracy(x_batch, d_batch)
37          accuracies_train.append(accr_train)
38
39  # print('Generation: ' + str(i+1) + '. 正答率トレーニング () = ' + str(accr_train))
40  # print(' : ' + str(i+1) + '. 正答率テスト () = ' + str(accr_test))
41
42
43  lists = range(0, iters_num, plot_interval)
44  plt.plot(lists, accuracies_train, label="training set")
45  plt.plot(lists, accuracies_test, label="test set")
46  plt.legend(loc="lower right")
47  plt.title("Relu - He")
48  plt.xlabel("count")
49  plt.ylabel("accuracy")
50  plt.ylim(0, 1.0)
51  # グラフの表示
52  plt.show()
```

初期値（勾配消失回避）



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である。例えば、ミニバッチの際は、機材によって変化する。CPU（汎用）、GPU（ゲーム用）だと 1〜64 枚、TPU（AI 専用）だとだと 1〜256 枚程度。コンピューター上では、一度の学習でこれくらいのデータしか処理できない。

バッチ正規化の使い所は、活性化関数に値を渡す前後に、バッチ正規化の処理を含んだ層を加える。
具体的には、バッチ正規化層への入力値は次のようになる。

$$\boldsymbol{u}^{(l)} = \boldsymbol{w}^{l} z^{(l-1)} + \boldsymbol{b}^{(l)}, \quad \text{or} \quad \boldsymbol{z}$$

■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。

■答え

1. 学習が安定化するため、結果として計算速度の高速化
2. 過学習を抑制することができる。（ミニバッチの過程で学習データを正規化するため）

バッチ正規化の数学的記述

$$\text{ミニバッチのデータ平均値}\quad \mu_y = \frac{1}{N_t}\sum_{i=1}^{N_t} x_{ni}$$

$$\text{ミニバッチのデータ分散}\quad \sigma_t^2 = \frac{1}{N_t}\sum_{i=1}^{N_t}(x_{ni}-\mu_i)^2$$

$$\text{ミニバッチの正規化}\quad \hat{x_{ni}} = \frac{x_{ni}-\mu_i}{\sigma_t^2+\theta}$$

$$\text{変倍・移動}\quad y_{ni} = \gamma x_{ni} + \beta$$

ここで、μ_i：ミニバッチ t 全体の平均、σ_t^2：ミニバッチ t 全体の標準偏差、N_t：ミニバッチのインデックス、$\hat{x_{ni}}$：0 に値を近づける計算（0 を中心とするセンタリング）と正規化を施した値、γ：スケーリングパラメータ、β シフトパラメタ、y_{ni}：ミニバッチのインデックス値とスケーリングの積にシフトを加算した値（バッチ正規化オペレーションの出力）である。

特に、4 番目のステップは、ニューラルネットワークにとって扱いが良いようにデータを最終的に補正するもの。

■実装演習成果

Listing 1.3　バッチ正規化

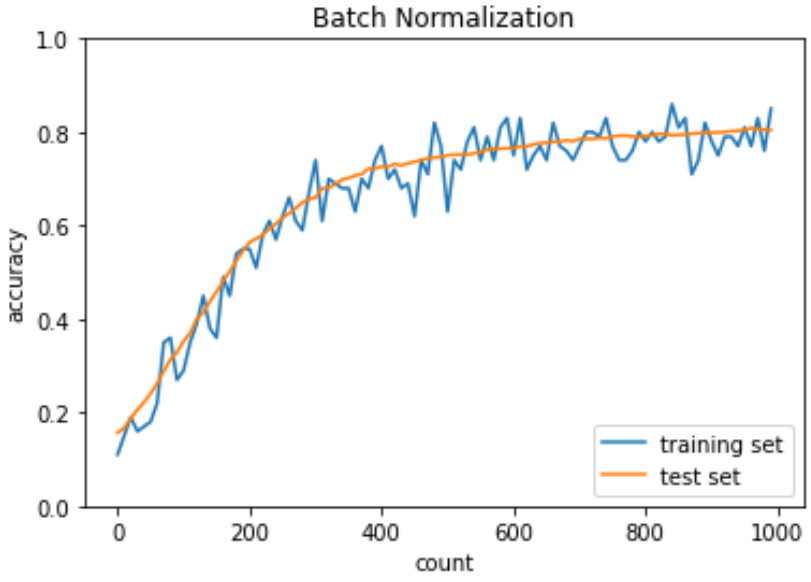
```
import numpy as np
from collections import OrderedDict
from common import layers
from data.mnist import load_mnist
import matplotlib.pyplot as plt
from multi_layer_net import MultiLayerNet
from common import optimizer

# バッチ正則化layer
class BatchNormalization:
    '''
    gamma: スケール係数
    beta: オフセット
    momentum: 慣性
    running_mean: テスト時に使用する平均
    running_var: テスト時に使用する分散
    '''
```

```
18      def __init__(self, gamma, beta, momentum=0.9, running_mean=None, running_var=None):
19          self.gamma = gamma
20          self.beta = beta
21          self.momentum = momentum
22          self.input_shape = None
23
24          self.running_mean = running_mean
25          self.running_var = running_var
26
27          # 時に使用する中間データ backward
28          self.batch_size = None
29          self.xc = None
30          self.std = None
31          self.dgamma = None
32          self.dbeta = None
33
34      def forward(self, x, train_flg=True):
35          if self.running_mean is None:
36              N, D = x.shape
37              self.running_mean = np.zeros(D)
38              self.running_var = np.zeros(D)
39
40          if train_flg:
41              mu = x.mean(axis=0) # 平均
42              xc = x - mu # をセンタリング x
43              var = np.mean(xc**2, axis=0) # 分散
44              std = np.sqrt(var + 10e-7) # スケーリング
45              xn = xc / std
46
47              self.batch_size = x.shape[0]
48              self.xc = xc
49              self.xn = xn
50              self.std = std
51              self.running_mean = self.momentum * self.running_mean + (1-self.momentum) *
                     mu # 平均値の加重
                    平均
52              self.running_var = self.momentum * self.running_var + (1-self.momentum) *
                    var 分散値の加重平均
                    #
53          else:
54              xc = x - self.running_mean
55              xn = xc / ((np.sqrt(self.running_var + 10e-7)))
56
57          out = self.gamma * xn + self.beta
58
59          return out
60
```

```
    def backward(self, dout):
        dbeta = dout.sum(axis=0)
        dgamma = np.sum(self.xn * dout, axis=0)
        dxn = self.gamma * dout
        dxc = dxn / self.std
        dstd = -np.sum((dxn * self.xc) / (self.std * self.std), axis=0)
        dvar = 0.5 * dstd / self.std
        dxc += (2.0 / self.batch_size) * self.xc * dvar
        dmu = np.sum(dxc, axis=0)
        dx = dxc - dmu / self.batch_size

        self.dgamma = dgamma
        self.dbeta = dbeta

        return dx
```

```
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)

printデータ読み込み完了("")


# の設定batch_normalization ========================
use_batchnorm = True
# use_batchnorm = False
# ====================================================

network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20], output_size=10,
                        activation='sigmoid', weight_init_std='Xavier', use_batchnorm=
                            use_batchnorm)

iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate=0.01

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10


for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]
```

```
30
31      grad = network.gradient(x_batch, d_batch)
32      for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
33          network.params[key] -= learning_rate * grad[key]
34
35          loss = network.loss(x_batch, d_batch)
36          train_loss_list.append(loss)
37
38      if (i + 1) % plot_interval == 0:
39          accr_test = network.accuracy(x_test, d_test)
40          accuracies_test.append(accr_test)
41          accr_train = network.accuracy(x_batch, d_batch)
42          accuracies_train.append(accr_train)
43
44  # print('Generation: ' + str(i+1) + '. 正答率トレーニング () = ' + str(accr_train))
45  # print(' : ' + str(i+1) + '. 正答率テスト () = ' + str(accr_test))
46
47
48  lists = range(0, iters_num, plot_interval)
49  plt.plot(lists, accuracies_train, label="training set")
50  plt.plot(lists, accuracies_test, label="test set")
51  plt.legend(loc="lower right")
52  plt.title("Batch Normalization")
53  plt.xlabel("count")
54  plt.ylabel("accuracy")
55  plt.ylim(0, 1.0)
56  # グラフの表示
57  plt.show()
```



■例題チャレンジ

深層学習では、一般に必要とするデータが多く、メモリなどの都合で全てまとめてバッチで計算することができない。そのため、データを少数のまとまりであるミニバッチにして計算を行う。次に示すコードは、特徴データ data_x, ラベルデータ data_t に対してミニバッチ学習を行うプログラムである。（き）にあてはまるのは何か

```
def train(data_x, data_t, n_epoch, batch_size):
    """
    data_x: training data (features)
    data_t: training data (labels)
    n_epoch: number of epochs
    batch_size: mini batch size
    """
    N = len(data_x)
    for epoch in range(n_epoch):
        shuffle_idx = np.random.permutation(N)
        for i in range(0, N, batch_size):
            i_end = i + batch_size
            batch_x, batch_t = （き）
            _update(batch_x, batch_t)
```

(1) data_x[i:i_end], data_t[i:i_end]
(2) data_x[i_end:i], data_t[i_end:i]
(3) data_x[i:], data_t[i:]
(4) data_x[:i_end], data_t[:i_end]

■答え

```
1        data_x[i:i_end], data_t[i:i_end]
```

※　range 関数（バッチサイズだけデータを取り出す処理）TPU などの AI 専用計算機は、8 倍ずつしか処理できないなどのハードウェア上の制限あるため、ミニバッチの数を 2 の階乗に指定することが多い。

■サマリ

1. 活性化関数で勾配消失問題が起こりにくいものを選ぶ。
2. 重みの初期値をこだわる。
3. バッチ正規化によりバッチ正規化手法により、ミニバッチ処理をする。

第 2 章

学習率最適化手法

■全体像　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である。具体的には、誤差 $\boldsymbol{E}(w)$ を最小化するパラメータ w を発見することにある。例えば、勾配降下法を利用してパラメータを最適化する際は、学習率 ϵ の大小によって収束するか否かの影響を与える。再掲になるが、学習率の値が大きい場合は、最適値にいつまでもたどり着かず発散し、逆に学習率の値が小さい場合は、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というデメリットも有していた（大域局所最適値に収束しづらくなる。）。

ここでは、学習率の決め方を次の指針に基づき最適化することを考える。これから 4 つの最適化手法 (optimizer) を紹介するが、優秀なのは Adam であり、今現在はこれがよく用いられている。

初期の学習率設定方法の指針

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　（これまでは、ϵ で固定）
2. パラメータ毎に学習率を可変させる

これらの指針に基づく「学習率最適化手法」を利用して、ニューラルネットワークにおける学習率を最適化を図っていく。

2.1　勾配降下法

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。

$$w^{(t+1)} = w^{(t)} - \epsilon \nabla E, \quad \text{where } \epsilon \text{（学習率）}$$

2.2　モメンタム

モメンタムとは、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。

$$V_t = \mu V_{t-1} - \epsilon \nabla E$$
$$w^{(t+1)} = w^{(t)} + V_t$$

ここで、V_{t-1}：前回の重みであり、このパラメータを新たに考慮することがポイント。μ：慣性とよぶ。
実際に、確率的勾配降下法（SGD）に適用した際は、SGD の振動を抑える、例えれば、株価の移動平均のよ

うな動きになる（勾配降下法は、ジグザグな変動だったのに対して)。【モメンタムのメリット】

1. 局所的最適解にはならず、大域的最適解となる。
2. 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。

■実装演習成果（数式とコード）

$$\boxed{V_t = \mu V_{t-1} - \epsilon \nabla E}$$

```
1        self.v[key] = self.momentum * self.v[key] - swlf.leraning_arte * grad[key]
```

$$\boxed{w^{(t+1)} = w^{(t)} + V_t}$$

```
1        params[key] += self.v[key]
```

2.3 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。これは、いままでの学習結果（重み更新量）を蓄積して、それを活用するもの。その際、新しいパラメータ h を導入する。これは、重み 1 つ 1 つに対して調整する作用を与える。

$$\begin{aligned} h_0 &= \theta \\ h_t &= h_{t-1} + (\nabla E)^2 \\ w^{(t+1)} &= w^{(t)} - \epsilon \frac{1}{\sqrt{h_t} + \theta} \nabla E \end{aligned}$$

ここで θ は、初期値として適当に与える。

1. メリット $\cdots$ 勾配の緩やかな斜面に対して、最適値に近づける。
2. 課題 $\cdots$ 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（大域的最適解の探索には不向き)。

■実装演習成果（数式とコード）

$$\boxed{h_0 = \theta}$$

```
1        \boxed{self.h[key] = np.zeros_like(val)}
```

ここで、zeros_like はランダムに勝手に適当な何かしらの値を与えてくれるもの。

$$\boxed{h_t = h_{t-1} + (\nabla E)^2}$$

```
1        self.h[key] += grad[key] * grad[key]
```

$$\boxed{w^{(t+1)} = w^{(t)} - \epsilon \frac{1}{\sqrt{h_t} + \theta} \nabla E}$$

```
1        params[key] -= self.learning_rate * grad[key] / (np.sqrt(self.h[key]) + 1e-7)
```

2.4 RMSProp

AdaGrad で課題にあげられていた、鞍点問題を回避するために改良されたもの。基本的な原理は AdaGrad と同様であり、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。

$$h_t = \alpha h_{t-1} + (1-\alpha)(\nabla E)^2$$
$$w^{(t+1)} = w^{(t)} - \epsilon \frac{1}{\sqrt{h_t} + \theta} \nabla E$$

ここで、h_t の式内の $\alpha(0 \sim 1$ の値)、どれくらい前回までの勾配の値を活かすかというパラメータ。第 1 項は、どれくらいの前回の経験を活かすか、第 2 項目は、どれくらい今回の経験を活かすかということになる。

1. 局所的最適解にはならず、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。

■実装演習成果（数式とコード）

$$\boxed{h_t = \alpha h_{t-1} + (1-\alpha)(\nabla E)^2}$$

```
        self.h[key] *= self.decay_rate
        self.h[key] *= (1 - self.decay_rate) * grad[key] * grad[key]
```

$$\boxed{w^{(t+1)} = w^{(t)} - \epsilon \frac{1}{\sqrt{h_t} + \theta} \nabla E}$$

```
        params[key] -= self.lerning_rate * grad[key] / (np.sqrt(self.h[key]) + 1e-7)
```

■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。

■答え

1. 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。局所的最適解にはならず、大域的最適解となるスピードが比較的早い。また、移動平均の概念を用いた計算であるため、SGD に比して振動が少なくなる。一方で変化が緩い学習の際は、鞍点に引っかかって学習が進まなくなる欠点も有している。
2. AdaGrad··· パラメータ更新の式で明らかであるが、更新量 (∇E) がそれなりの値を持つとき、h_t を通じて、学習率 ϵ に作用するとき、分母（$\sqrt{h_t} + \theta$）が効くため、学習率の減衰がその分だけ抑制されることになる。一方、鞍点問題には適用すると学習が進まない欠点も有している。
3. RMSprop··· 基本的に AdaGrad の改良版であり、その相違はパラメータ α(decay_rate) を導入し、過去と現在の更新量の軽重のバランスを取り直している点になる。これによって、比較的鞍点問題にも対応した学習が実行できるようになる。

2.5 Adam

Adam とは、モメンタム「過去の勾配の指数関数的減衰平均」と RMSProp「過去の勾配の 2 乗の指数関数的減衰平均」のこれら 2 つのメリットを含んだ最適化アルゴリズムである。その他にも一見よさそうな optimizer「Ftrl」などがあるが、これは誤差関数の値が激しく振動するため学習がうまく進展しているのか否か判断しにくい側面がある。こういった点から、今般の機械学習屋は、誤差関数の値が比較的安定して収束（減少）していく、「Adam optimizer」を信奉している。

$$
\begin{aligned}
m_0 &= \theta_0, \quad v_0 = \theta_1 \\
m_t &= m_{t-1} + (1-\beta_1)(\nabla E - m_{t-1}) = \beta_1 m_{t-1} + (1-\beta_1)\nabla E \\
v_t &= v_{t-1} + (1-\beta_2)((\nabla E)^2 - v_{t-1}) = \beta_2 v_{t-1} + (1-\beta_2)(\nabla E)^2 \\
w^{(t+1)} &= w^{(t)} - \epsilon \frac{m_t}{\sqrt{v_t}}
\end{aligned}
$$

ここで、ϵ（学習率）、$\beta_{1,2}$ はハイパーパラメータである。

■実装演習成果

Listing 2.1　SGD（確率的勾配降下法）

```
import sys, os
sys.path.append(os.pardir) # 親ディレクトリのファイルをインポートするための設定
import numpy as np
from collections import OrderedDict
from common import layers
from data.mnist import load_mnist
import matplotlib.pyplot as plt
from multi_layer_net import MultiLayerNet


# データの読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True, one_hot_label=True)

printデータ読み込み完了("")

# の設定batch_normalization ==================================
# use_batchnorm = True
use_batchnorm = False
# ====================================================


network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20], output_size=10,
    activation='sigmoid', weight_init_std=0.01,
                        use_batchnorm=use_batchnorm)

iters_num = 1000
train_size = x_train.shape[0]
```

```
batch_size = 100
learning_rate = 0.01

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10

for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    # 勾配
    grad = network.gradient(x_batch, d_batch)

    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        network.params[key] -= learning_rate * grad[key]

        loss = network.loss(x_batch, d_batch)
        train_loss_list.append(loss)


    if (i + 1) % plot_interval == 0:
        accr_test = network.accuracy(x_test, d_test)
        accuracies_test.append(accr_test)
        accr_train = network.accuracy(x_batch, d_batch)
        accuracies_train.append(accr_train)

        print('Generation: ' + str(i+1) + '. 正答率トレーニング
            () = ' + str(accr_train))
        print(' : ' + str(i+1) + '. 正答率テスト () = ' + str(accr_test))


lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test, label="test set")
plt.legend(loc="lower right")
plt.title("accuracy")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
# グラフの表示
plt.show()
```

Listing 2.2 Momentum

```
# データの読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True, one_hot_label=True)

printデータ読み込み完了("")

# の設定batch_normalization ================================
# use_batchnorm = True
use_batchnorm = False
# ====================================================

network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20], output_size=10,
    activation='sigmoid', weight_init_std=0.01,
                        use_batchnorm=use_batchnorm)

iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.01
# 慣性
momentum = 0.9

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10

for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    # 勾配
    grad = network.gradient(x_batch, d_batch)
    if i == 0:
        v = {}
    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        if i == 0:
            v[key] = np.zeros_like(network.params[key])
        v[key] = momentum * v[key] - learning_rate * grad[key]
        network.params[key] += v[key]

        loss = network.loss(x_batch, d_batch)
        train_loss_list.append(loss)

```

```
45      if (i + 1) % plot_interval == 0:
46          accr_test = network.accuracy(x_test, d_test)
47          accuracies_test.append(accr_test)
48          accr_train = network.accuracy(x_batch, d_batch)
49          accuracies_train.append(accr_train)
50
51          print('Generation: ' + str(i+1) + '. 正答率トレーニング
                () = ' + str(accr_train))
52          print(' : ' + str(i+1) + '. 正答率テスト () = ' + str(accr_test))
53
54
55  lists = range(0, iters_num, plot_interval)
56  plt.plot(lists, accuracies_train, label="training set")
57  plt.plot(lists, accuracies_test, label="test set")
58  plt.legend(loc="lower right")
59  plt.title("Momentum")
60  plt.xlabel("count")
61  plt.ylabel("accuracy")
62  plt.ylim(0, 1.0)
63  # グラフの表示
64  plt.show()
```

Listing 2.3　AdaGrad

```
1   # を作ってみよう AdaGrad
2   # データの読み込み
3   (x_train, d_train), (x_test, d_test) = load_mnist(normalize=True, one_hot_label=True)
4
5   printデータ読み込み完了 ("")
6
7   # の設定 batch_normalization ================================
8   # use_batchnorm = True
9   use_batchnorm = False
10  # ============================================================
11
12  network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20], output_size=10,
        activation='sigmoid', weight_init_std=0.01,
13                          use_batchnorm=use_batchnorm)
14
15  iters_num = 1000
16  # iters_num = 500 # 処理を短縮
17
18  train_size = x_train.shape[0]
19  batch_size = 100
20  learning_rate = 0.01
21
22  # では不必要 AdaGrad
```

```
23 # ==============================
24
25 momentum = 0.9
26
27 # ==============================
28
29 train_loss_list = []
30 accuracies_train = []
31 accuracies_test = []
32
33 plot_interval=10
34
35 for i in range(iters_num):
36     batch_mask = np.random.choice(train_size, batch_size)
37     x_batch = x_train[batch_mask]
38     d_batch = d_train[batch_mask]
39
40     # 勾配
41     grad = network.gradient(x_batch, d_batch)
42     if i == 0:
43         h = {}
44     for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
45
46         # 変更しよう
47         # ==========================================
48         if i == 0:
49 # Momentum
50 # h[key] = np.zeros_like(network.params[key])
51 # h[key] = momentum * h[key] - learning_rate * grad[key]
52 # network.params[key] += h[key]
53 # AdaGrad
54             h[key] = np.zeros_like(network.params[key])
55         h[key] += grad[key] * grad[key]
56         network.params[key] -= learning_rate * grad[key] / (np.sqrt(h[key]) + 1e-7)
57         # ==========================================
58
59         loss = network.loss(x_batch, d_batch)
60         train_loss_list.append(loss)
61
62     if (i + 1) % plot_interval == 0:
63         accr_test = network.accuracy(x_test, d_test)
64         accuracies_test.append(accr_test)
65         accr_train = network.accuracy(x_batch, d_batch)
66         accuracies_train.append(accr_train)
67
68         print('Generation: ' + str(i+1) + '. 正答率トレーニング
```

```
            () = ' + str(accr_train))
        print(' : ' + str(i+1) + '. 正答率テスト () = ' + str(accr_test))


lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test, label="test set")
plt.legend(loc="lower right")
plt.title("AdaGrad")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
# グラフの表示
plt.show()
```

Listing 2.4　RMSprop

```
# データの読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True, one_hot_label=True)

printデータ読み込み完了 ("")

# の設定 batch_normalization ===================================
# use_batchnorm = True
use_batchnorm = False
# ====================================================

network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20], output_size=10,
    activation='sigmoid', weight_init_std=0.01,
                        use_batchnorm=use_batchnorm)

iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.01
decay_rate = 0.99

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10

for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]
```

```
30
31      # 勾配
32      grad = network.gradient(x_batch, d_batch)
33      if i == 0:
34          h = {}
35      for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
36          if i == 0:
37              h[key] = np.zeros_like(network.params[key])
38          h[key] *= decay_rate
39          h[key] += (1 - decay_rate) * np.square(grad[key])
40          network.params[key] -= learning_rate * grad[key] / (np.sqrt(h[key]) + 1e-7)
41
42          loss = network.loss(x_batch, d_batch)
43          train_loss_list.append(loss)
44
45      if (i + 1) % plot_interval == 0:
46          accr_test = network.accuracy(x_test, d_test)
47          accuracies_test.append(accr_test)
48          accr_train = network.accuracy(x_batch, d_batch)
49          accuracies_train.append(accr_train)
50
51          print('Generation: ' + str(i+1) + '. 正答率トレーニング
                () = ' + str(accr_train))
52          print(' : ' + str(i+1) + '. 正答率テスト () = ' + str(accr_test))
53
54
55  lists = range(0, iters_num, plot_interval)
56  plt.plot(lists, accuracies_train, label="training set")
57  plt.plot(lists, accuracies_test, label="test set")
58  plt.legend(loc="lower right")
59  plt.title("RSMprop")
60  plt.xlabel("count")
61  plt.ylabel("accuracy")
62  plt.ylim(0, 1.0)
63  # グラフの表示
64  plt.show()
```

Listing 2.5 Adam

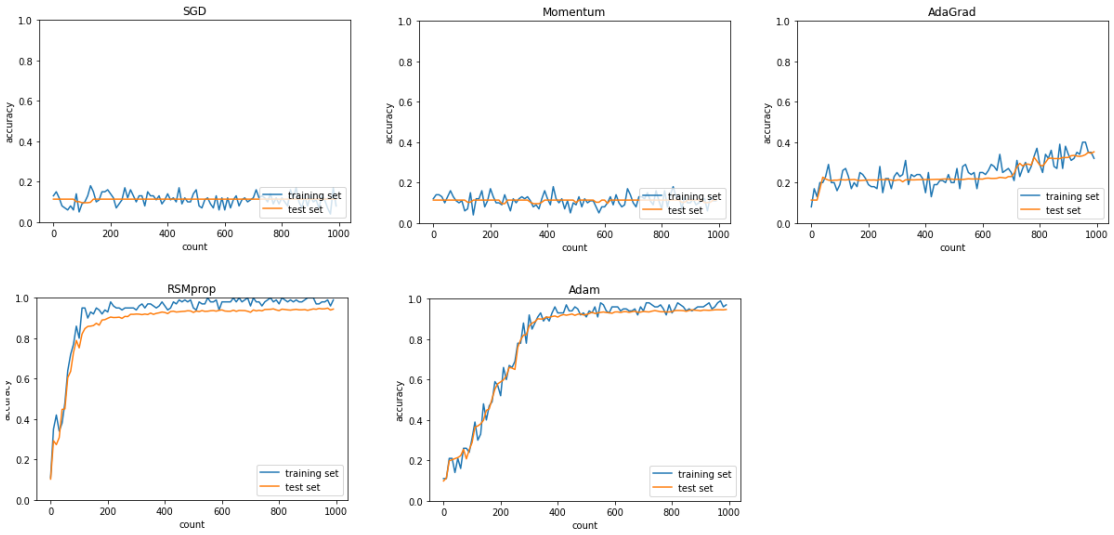
```
# データの読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True, one_hot_label=True)

printデータ読み込み完了("")

# の設定batch_normalization ================================
# use_batchnorm = True
use_batchnorm = False
# ====================================================

network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20], output_size=10,
    activation='sigmoid', weight_init_std=0.01,
                        use_batchnorm=use_batchnorm)

iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.01
beta1 = 0.9
beta2 = 0.999

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10

for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    # 勾配
    grad = network.gradient(x_batch, d_batch)
    if i == 0:
        m = {}
        v = {}
    learning_rate_t = learning_rate * np.sqrt(1.0 - beta2 ** (i + 1)) / (1.0 - beta1
        ** (i + 1))
    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        if i == 0:
            m[key] = np.zeros_like(network.params[key])
            v[key] = np.zeros_like(network.params[key])

        m[key] += (1 - beta1) * (grad[key] - m[key])
```

```
44          v[key] += (1 - beta2) * (grad[key] ** 2 - v[key])
45          network.params[key] -= learning_rate_t * m[key] / (np.sqrt(v[key]) + 1e-7)
46
47
48      if (i + 1) % plot_interval == 0:
49          accr_test = network.accuracy(x_test, d_test)
50          accuracies_test.append(accr_test)
51          accr_train = network.accuracy(x_batch, d_batch)
52          accuracies_train.append(accr_train)
53          loss = network.loss(x_batch, d_batch)
54          train_loss_list.append(loss)
55
56          print('Generation: ' + str(i+1) + '. 正答率トレーニング
                () = ' + str(accr_train))
57          print(' : ' + str(i+1) + '. 正答率テスト () = ' + str(accr_test))
58
59
60  lists = range(0, iters_num, plot_interval)
61  plt.plot(lists, accuracies_train, label="training set")
62  plt.plot(lists, accuracies_test, label="test set")
63  plt.legend(loc="lower right")
64  plt.title("Adam")
65  plt.xlabel("count")
66  plt.ylabel("accuracy")
67  plt.ylim(0, 1.0)
68  # グラフの表示
69  plt.show()
```

サマリ

第 1 章から第 2 章のここまで、勾配消失問題をいかに解決するかという各種手法を学んできた、第 1 章では、初期値の設定方法、第 2 章では、学習率最適化手法（重み更新の最適化）を学んだ。同じデータセット、検証データセットでも、Accuracy にこんなにも差が出ることは驚愕であった。

今後、実務で機械学習を適用させるときには、適切なデータ分割（訓練データと検証データ）、条件設定、アルゴリズム、ハイパーパラメータ、誤差関数の数値をモニター等により適正な判断をしつつオプティマイズしていくことが肝要である。



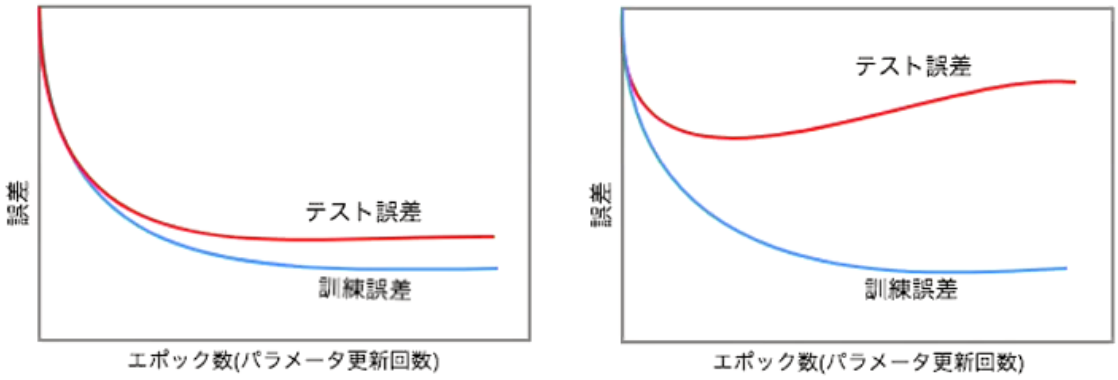
第 3 章

過学習

■全体像

過学習とは、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、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 etc... の多様な原因に起因する。※　ネットワークの自由度 (層数、ノード数、パラメータの値 etc...) が高い。

例えば、過学習の原因との 1 つである重みが大きい値をとることで、過学習が発生する時は、学習させていくと、重みにばらつきが発生する。重みが大きい値は、学習において重要な値であり、重みが大きいと過学習が起こることになる。



3.1　L1 正則化、L2 正則化

過学習の解決策としては、誤差に対して、正則化項を加算することで、重みを抑制することが考えらえる。つまり、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必要がある（荷重減衰, weight decay)。

正則化とは、ネットワークの自由度 (層数、ノード数、パラメータの値 etc...) を制約すること。この正則化手法を利用して過学習を抑制することが可能となる。

$$E_n(\boldsymbol{w}) + \frac{1}{p}||x||_p, \quad \text{where} \quad \text{Norm}： \quad ||x||_p = (|x_1|^p + \cdots + |x_n|^p)^{\frac{1}{p}}$$

例えば、$p = 1$ の場合、L1 正則化 (ラッソ回帰)、$p = 2$ の場合、L2 正則化（リッジ回帰）と呼び、それぞれの実際の計算した大きさ（距離）をそれぞれ「マンハッタン距離」、「ユークリッド距離」と呼ぶ。また、λ は人間が決めるハイパーパラメータである。

■確認テスト

正則化手法の 1 つにリッジ回帰 (L2 正則化) があるが、その特徴は何か？次の中から選べ。

■答え

1. 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ゼロに近づく
   （×）　ハイパーパラメータを大きくすことで、重み $\boldsymbol{w}$ の大きさが過学習しないように大きくならないように制約は与えるが、あくまでも出力層でのアウトプットと訓練データの誤差を最小化するために適正なノンゼロ成分の重み $\boldsymbol{w}$ は残る。
2. ハイパーパラメータをゼロに設定すると、非線形回帰となる。
   （×）　この場合、正則化のない単純な回帰となるだけ。
3. バイアス項についても正則化される。
   （○）　その通り。その定義に従って、重み $\boldsymbol{w}$ ならびにバイアス $\boldsymbol{b}$ について正則化がかかる。
4. リッジ回帰の場合、隠れ層に対して正則化項を加える。
   （×）　隠れ層のみならず、出力層から逆伝播して最終的に入力層まで回帰する。

■実装演習成果（数式とコード）

$$\boxed{||x||_p = (|x_1|^p + \cdots + |x_n|^p)^{\frac{1}{p}}}$$

```
1        np.sum(np.abs(network.params['W'+str(idx)]))
```

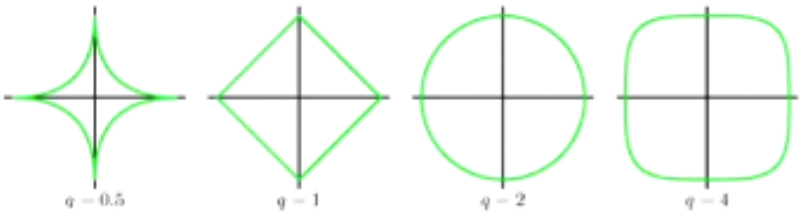
$$\boxed{E_n(\boldsymbol{w}) + \frac{1}{p}||x||_p}$$

```
1        weight_decay += weight_decay_lambda * np.sum(np.abs(network.params['W'+str(idx)
             ]))
2        loss = network.loss(x_batch, d_batch) + weight_decay
```

いろいろな正規化項のグラフ

$$\sum_{i=1}^{n} |w_i^q$$

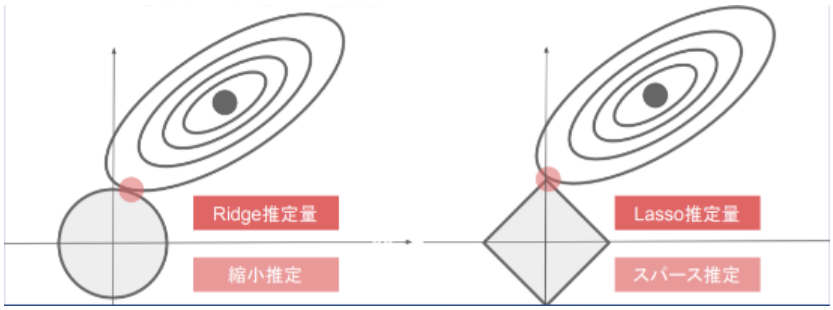
Rで描いた三次元グラフ



等高線の形（三次元グラフを横で切った形）

■確認テスト

下図について、L1 正則化を表しているグラフはどちらか答えよ。

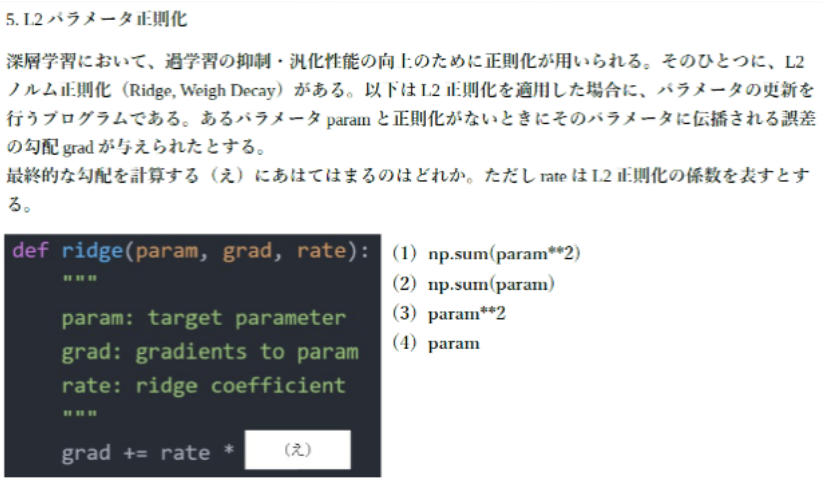


■答え

右側の Lasso 推定量である。等高線は、誤差関数を表し、円（L2 正則化）及びひし形（L1 正則化）を表しており、本正則化が加わることで、誤差関数だけの極小値（黒丸）ではなく、円又はひし形の交点部分が最終的に正則化を考慮した上で選択された最適解となる。

この正則化が加わることで、ラッソの角の部分に向かってパラメータ更新が進展するようになり。また、リッジ推定量に比して、スパース化も見込まれる　※　学習時の不効率性の除去。さらに、L1 正則化では、1 つのパラメータがゼロになるように選択される。

■例題チャレンジ



5. L2 パラメータ正則化

深層学習において、過学習の抑制・汎化性能の向上のために正則化が用いられる。そのひとつに、L2 ノルム正則化（Ridge, Weigh Decay）がある。以下は L2 正則化を適用した場合に、パラメータの更新を行うプログラムである。あるパラメータ param と正則化がないときにそのパラメータに伝播される誤差の勾配 grad が与えられたとする。
最終的な勾配を計算する（え）にあはてはまるのはどれか。ただし rate は L2 正則化の係数を表すとする。

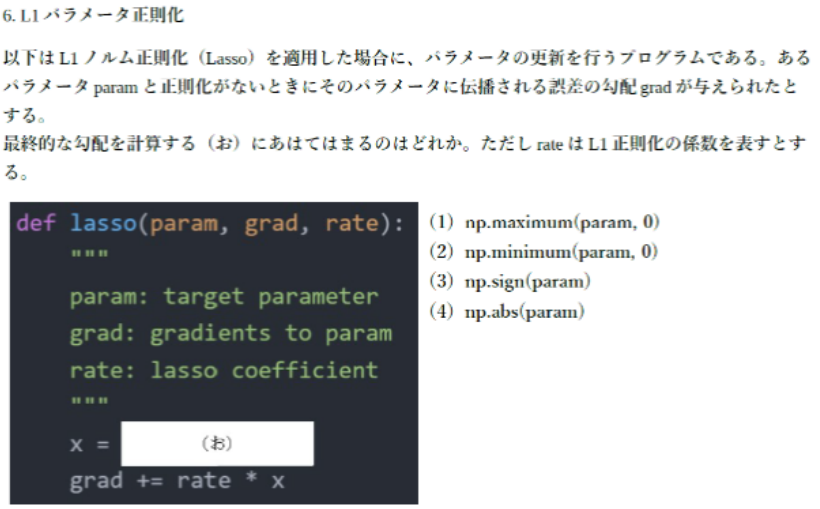
```
def ridge(param, grad, rate):
    """
    param: target parameter
    grad: gradients to param
    rate: ridge coefficient
    """
    grad += rate * （え）
```

(1) np.sum(param**2)
(2) np.sum(param)
(3) param**2
(4) param

■答え　param

L2 ノルムは、$||\text{param}||^2$ なので、その勾配が誤差の勾配に加えられる。つまり、$2 * \text{param}$ であるが、係数 2 は正則化の係数に吸収されても変わらないので param が正解となる。

■例題チャレンジ



6. L1 パラメータ正則化

以下は L1 ノルム正則化（Lasso）を適用した場合に、パラメータの更新を行うプログラムである。あるパラメータ param と正則化がないときにそのパラメータに伝播される誤差の勾配 grad が与えられたとする。
最終的な勾配を計算する（お）にあはてはまるのはどれか。ただし rate は L1 正則化の係数を表すとする。

```
def lasso(param, grad, rate):
    """
    param: target parameter
    grad: gradients to param
    rate: lasso coefficient
    """
    x = （お）
    grad += rate * x
```

(1) np.maximum(param, 0)
(2) np.minimum(param, 0)
(3) np.sign(param)
(4) np.abs(param)

■答え　sign(param)

L1 ノルムは、|param| なので、その勾配が誤差の勾配に加えられる。つまり、sign(param) である。ここで、sign は符号関数である。

$$sign(x) = \begin{cases} 1 & (x > 0) \\ 0 & (x = 0) \\ -1 & (x < 0) \end{cases}$$

3.2 ドロップアウト

過学習の課題の 1 つして、ノードの数が多いことがあげられる。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である。メリットとしては、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ことにある。

■実装演習成果

Listing 3.1 overfitting

```
import numpy as np
from collections import OrderedDict
from common import layers
from data.mnist import load_mnist
import matplotlib.pyplot as plt
from multi_layer_net import MultiLayerNet
from common import optimizer


(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)

printデータ読み込み完了("")

# 過学習を再現するために、学習データを削減
x_train = x_train[:300]
d_train = d_train[:300]

network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100,
    100], output_size=10)
optimizer = optimizer.SGD(learning_rate=0.01)

iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10


for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]
```

```
36
37      grad = network.gradient(x_batch, d_batch)
38      optimizer.update(network.params, grad)
39
40      loss = network.loss(x_batch, d_batch)
41      train_loss_list.append(loss)
42
43      if (i+1) % plot_interval == 0:
44          accr_train = network.accuracy(x_train, d_train)
45          accr_test = network.accuracy(x_test, d_test)
46          accuracies_train.append(accr_train)
47          accuracies_test.append(accr_test)
48
49          print('Generation: ' + str(i+1) + '. 正答率トレーニング
                () = ' + str(accr_train))
50          print(' : ' + str(i+1) + '. 正答率テスト () = ' + str(accr_test))
51
52  lists = range(0, iters_num, plot_interval)
53  plt.plot(lists, accuracies_train, label="training set")
54  plt.plot(lists, accuracies_test, label="test set")
55  plt.legend(loc="lower right")
56  plt.title("accuracy")
57  plt.xlabel("count")
58  plt.ylabel("accuracy")
59  plt.ylim(0, 1.0)
60  # グラフの表示
61  plt.show()
```

Listing 3.2　L2 正則化

```
1   from common import optimizer
2
3   (x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)
4
5   printデータ読み込み完了 ("")
6
7   # 過学習を再現するために、学習データを削減
8   x_train = x_train[:300]
9   d_train = d_train[:300]
10
11
12  network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100,
        100], output_size=10)
13
14
15  iters_num = 1000
16  train_size = x_train.shape[0]
```

```
17 batch_size = 100
18 learning_rate=0.01
19
20 train_loss_list = []
21 accuracies_train = []
22 accuracies_test = []
23
24 plot_interval=10
25 hidden_layer_num = network.hidden_layer_num
26
27 # 正則化強度設定=======================================
28 weight_decay_lambda = 0.1
29 # ====================================================
30
31 for i in range(iters_num):
32     batch_mask = np.random.choice(train_size, batch_size)
33     x_batch = x_train[batch_mask]
34     d_batch = d_train[batch_mask]
35
36     grad = network.gradient(x_batch, d_batch)
37     weight_decay = 0
38
39     for idx in range(1, hidden_layer_num+1):
40         grad['W' + str(idx)] = network.layers['Affine' + str(idx)].dW +
              weight_decay_lambda * network.params['W' + str(idx)]
41         grad['b' + str(idx)] = network.layers['Affine' + str(idx)].db
42         network.params['W' + str(idx)] -= learning_rate * grad['W' + str(idx)]
43         network.params['b' + str(idx)] -= learning_rate * grad['b' + str(idx)]
44         weight_decay += 0.5 * weight_decay_lambda * np.sqrt(np.sum(network.params['W' +
               str(idx)] ** 2))
45
46     loss = network.loss(x_batch, d_batch) + weight_decay
47     train_loss_list.append(loss)
48
49     if (i+1) % plot_interval == 0:
50         accr_train = network.accuracy(x_train, d_train)
51         accr_test = network.accuracy(x_test, d_test)
52         accuracies_train.append(accr_train)
53         accuracies_test.append(accr_test)
54
55         print('Generation: ' + str(i+1) + '. 正答率トレーニング
               () = ' + str(accr_train))
56         print(' : ' + str(i+1) + '. 正答率テスト () = ' + str(accr_test))
57
58
59 lists = range(0, iters_num, plot_interval)
```

```
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test, label="test set")
plt.legend(loc="lower right")
plt.title("L2(weight decay)")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
# グラフの表示
plt.show()
```

Listing 3.3　L1 正則化

```
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)

printデータ読み込み完了("")

# 過学習を再現するために、学習データを削減
x_train = x_train[:300]
d_train = d_train[:300]

network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100,
    100], output_size=10)


iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate=0.1

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10
hidden_layer_num = network.hidden_layer_num

# 正則化強度設定======================================
weight_decay_lambda = 0.005
# ================================================

for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    grad = network.gradient(x_batch, d_batch)
    weight_decay = 0
```

```
35
36      for idx in range(1, hidden_layer_num+1):
37          grad['W' + str(idx)] = network.layers['Affine' + str(idx)].dW +
                weight_decay_lambda * np.sign(network.params['W' + str(idx)])
38          grad['b' + str(idx)] = network.layers['Affine' + str(idx)].db
39          network.params['W' + str(idx)] -= learning_rate * grad['W' + str(idx)]
40          network.params['b' + str(idx)] -= learning_rate * grad['b' + str(idx)]
41          weight_decay += weight_decay_lambda * np.sum(np.abs(network.params['W' + str(
                idx)]))
42
43      loss = network.loss(x_batch, d_batch) + weight_decay
44      train_loss_list.append(loss)
45
46      if (i+1) % plot_interval == 0:
47          accr_train = network.accuracy(x_train, d_train)
48          accr_test = network.accuracy(x_test, d_test)
49          accuracies_train.append(accr_train)
50          accuracies_test.append(accr_test)
51
52          print('Generation: ' + str(i+1) + '. 正答率トレーニング
                () = ' + str(accr_train))
53          print(' : ' + str(i+1) + '. 正答率テスト () = ' + str(accr_test))
54
55  lists = range(0, iters_num, plot_interval)
56  plt.plot(lists, accuracies_train, label="training set")
57  plt.plot(lists, accuracies_test, label="test set")
58  plt.legend(loc="lower right")
59  plt.title("L1(weight decay)")
60  plt.xlabel("count")
61  plt.ylabel("accuracy")
62  plt.ylim(0, 1.0)
63  # グラフの表示
64  plt.show()
```

Listing 3.4 Dropout

```
1   class Dropout:
2       def __init__(self, dropout_ratio=0.5):
3           self.dropout_ratio = dropout_ratio
4           self.mask = None
5
6       def forward(self, x, train_flg=True):
7           if train_flg:
8               self.mask = np.random.rand(*x.shape) > self.dropout_ratio
9               return x * self.mask
10          else:
11              return x * (1.0 - self.dropout_ratio)
```

```
12
13      def backward(self, dout):
14          return dout * self.mask
15  ------------------------------------------------------
16  from common import optimizer
17  (x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)
18
19  printデータ読み込み完了("")
20
21  # 過学習を再現するために、学習データを削減
22  x_train = x_train[:300]
23  d_train = d_train[:300]
24
25  # ドロップアウト設定=====================================
26  use_dropout = True
27  dropout_ratio = 0.15
28  # ====================================================
29
30  network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100,
        100], output_size=10,
31                          weight_decay_lambda=weight_decay_lambda, use_dropout =
                                use_dropout, dropout_ratio = dropout_ratio)
32  optimizer = optimizer.SGD(learning_rate=0.01)
33  # optimizer = optimizer.Momentum(learning_rate=0.01, momentum=0.9)
34  # optimizer = optimizer.AdaGrad(learning_rate=0.01)
35  # optimizer = optimizer.Adam()
36
37  iters_num = 1000
38  train_size = x_train.shape[0]
39  batch_size = 100
40
41  train_loss_list = []
42  accuracies_train = []
43  accuracies_test = []
44
45  plot_interval=10
46
47
48  for i in range(iters_num):
49      batch_mask = np.random.choice(train_size, batch_size)
50      x_batch = x_train[batch_mask]
51      d_batch = d_train[batch_mask]
52
53      grad = network.gradient(x_batch, d_batch)
54      optimizer.update(network.params, grad)
55
```

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

    if (i+1) % plot_interval == 0:
        accr_train = network.accuracy(x_train, d_train)
        accr_test = network.accuracy(x_test, d_test)
        accuracies_train.append(accr_train)
        accuracies_test.append(accr_test)

        print('Generation: ' + str(i+1) + '. 正答率トレーニング
            () = ' + str(accr_train))
        print(' : ' + str(i+1) + '. 正答率テスト () = ' + str(accr_test))

lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test, label="test set")
plt.legend(loc="lower right")
plt.title("Dropout")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
# グラフの表示
plt.show()
```

Listing 3.5　Dropout + L1 正則化

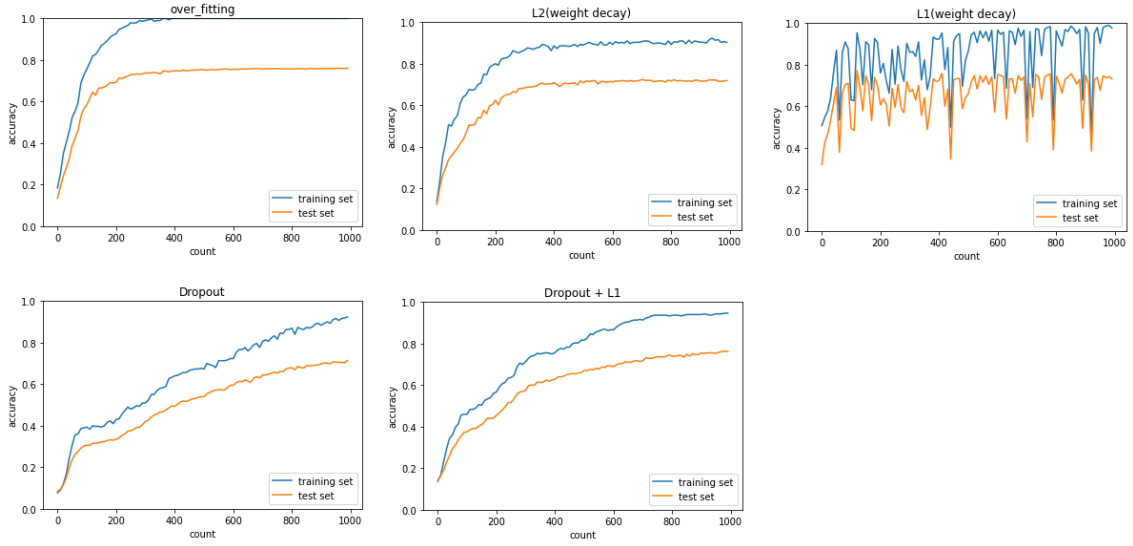
```
from common import optimizer
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)

printデータ読み込み完了("")

# 過学習を再現するために、学習データを削減
x_train = x_train[:300]
d_train = d_train[:300]

# ドロップアウト設定======================================
use_dropout = True
dropout_ratio = 0.08
# ====================================================

network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100,
    100], output_size=10,
                        use_dropout = use_dropout, dropout_ratio = dropout_ratio)

iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
```

```
learning_rate=0.01

train_loss_list = []
accuracies_train = []
accuracies_test = []
hidden_layer_num = network.hidden_layer_num

plot_interval=10

# 正則化強度設定======================================
weight_decay_lambda=0.004
# ================================================

for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    grad = network.gradient(x_batch, d_batch)
    weight_decay = 0

    for idx in range(1, hidden_layer_num+1):
        grad['W' + str(idx)] = network.layers['Affine' + str(idx)].dW +
            weight_decay_lambda * np.sign(network.params['W' + str(idx)])
        grad['b' + str(idx)] = network.layers['Affine' + str(idx)].db
        network.params['W' + str(idx)] -= learning_rate * grad['W' + str(idx)]
        network.params['b' + str(idx)] -= learning_rate * grad['b' + str(idx)]
        weight_decay += weight_decay_lambda * np.sum(np.abs(network.params['W' + str(
            idx)]))

    loss = network.loss(x_batch, d_batch) + weight_decay
    train_loss_list.append(loss)

    if (i+1) % plot_interval == 0:
        accr_train = network.accuracy(x_train, d_train)
        accr_test = network.accuracy(x_test, d_test)
        accuracies_train.append(accr_train)
        accuracies_test.append(accr_test)

        print('Generation: ' + str(i+1) + '. 正答率トレーニング
            () = ' + str(accr_train))
        print(' : ' + str(i+1) + '. 正答率テスト () = ' + str(accr_test))

lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test, label="test set")
```

```
64 plt.legend(loc="lower right")
65 plt.title("Dropout + L1")
66 plt.xlabel("count")
67 plt.ylabel("accuracy")
68 plt.ylim(0, 1.0)
69 # グラフの表示
70 plt.show()
```

サマリ

本章では、正則化の概念を導入し、誤差関数に正則項を加えて、もともとの訓練データと一致するニューラルネットワークの出力を出すようなパラメータを敢えてとらせないように制約を与えることだと認識した[*1]また、ドロップアウトは、途中の中間層ノードをランダムに削除して学習するさせる手法であり、実質的には仮想的に異なるデータ（モデル）を用いて学習させていることに相当する。そのため、Accuracy の上昇は、L1 正則化や L2 正則化に比して低調となる、かつより多くの訓練データが必要となる。



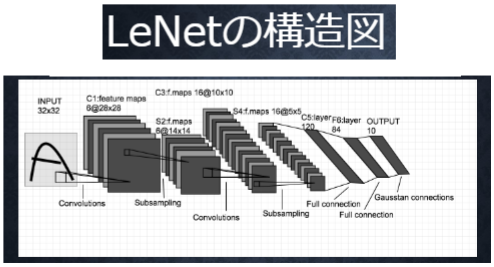
[*1] 機械学習（NN）においては、訓練データ 100%, 検証データ 100% の学習モデルは構築できないだろう（推測）

第 4 章

畳み込みニューラルネットワーク（CNN）の概念

■全体像

画像認識や画像識別でよく用いられる手法であるが、画像だけではなく、より汎用性の高いネットワークである。



CNN で扱えるデータの種類

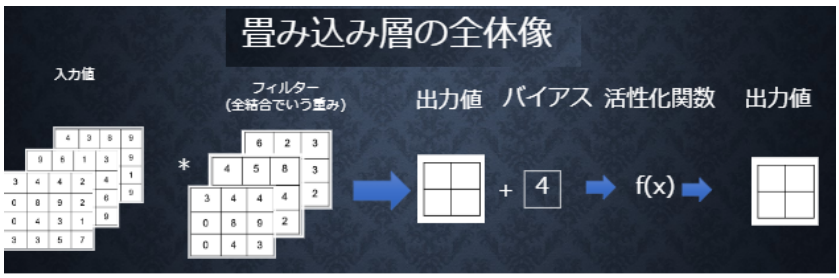
CNN では次元間で繋がりのあるデータを扱える。

	1次元	2次元	3次元
単一チャンネル	音声 [時刻, **強度**]	フーリエ変換した音声 [時刻, 周波数, **強度**]	CTスキャン画像 [x, y, z, **強度**]
複数チャンネル	アニメのスケルトン [時刻, (**腕の値**, **膝の値** …)]	カラー画像 [x, y, (**R**, **G**, **B**)]	動画 [時刻, x, y, (**R**, **G**, **B**)]]

LeNet の構造図

LeNet は 1998 年に考案された CNN の基本的なモデルである。入力層（画像データ）は、32pixel × 32pixel の比較的小さなデータ（現在の iPhone などでは、現在 3000 × 2000 程度が普通）。一方、出力層は 10 種類の識別を行うものとなっている。例えば数字の 0〜9 など。下の図は構造図。特に畳み込み (convolution) の説明において、美術館の感想を述べる人に例えたのは分かりやすかった (いろいろな人のいろいろな感想を取り込んでいくイメージ)。また全結合層 (Full connenction) も読書感想文を 1 冊にまとめるイメージで理解することができた。

(例題)

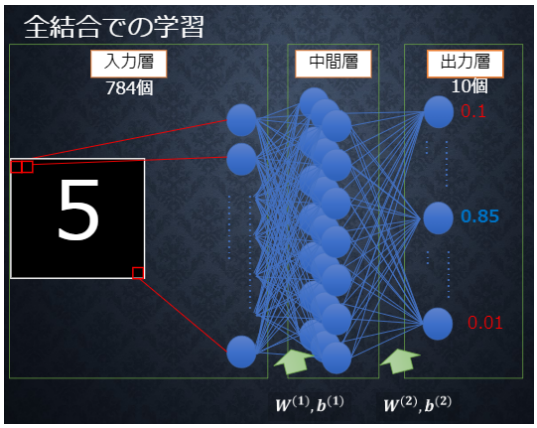


いま上図のように入力データとして 4 × 4 の 3ch のカラー画像がある時、3 × 3 のフィルター (これまでの全結合層 NN でいうところの重みに相当) を適用することを考える。このフィルターを通した結果、2 × 2 の出力値が得られる。また、入力データにフィルターを適用する際、フィルターをずらしながら逐一出力値を計算したが、この時に、入力データ内の隣接データとの繋がりを保ちつつフィルターをずらしていることがポイントである。これがまさに、畳み込みたるゆえんである。

また例題とは離れるが、音声データの時もまったく同じことが行われている。音声データは一般的に時間 t に対してデータが得られるが、この時もフィルター（読み取り範囲）をある時間間隔で区切ったものを、逐次時間発展方向へずらすことで時間的な繋がりを保ったまま処理を行うことができる。NN では、処理を行う、変換を行うということは、データから特徴を読み取ることに該当する。今の音声データの場合では、時間的な繋がりを持った特徴が得られることなる。

全結合層

全結合層は、これまでやってきた NN で既に用いてきた概念であり、ただ単に何個かの数字を入力として受け取り、重みとバイアスを通じて、新たな出力値を吐き出すものである。

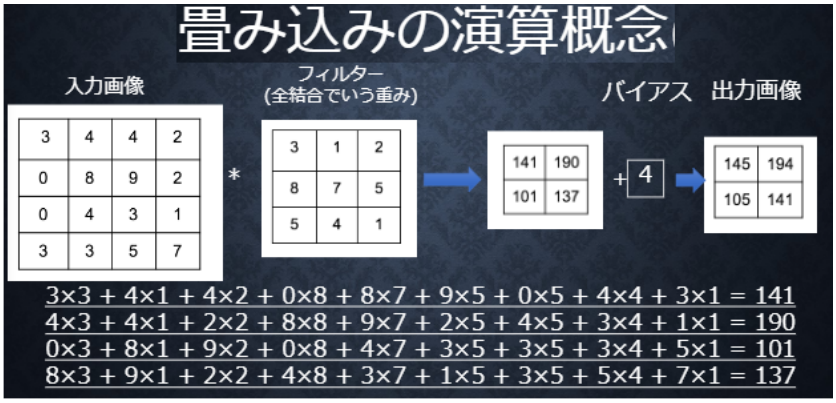


4.1　畳み込み層

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。

4.1.1　バイアス

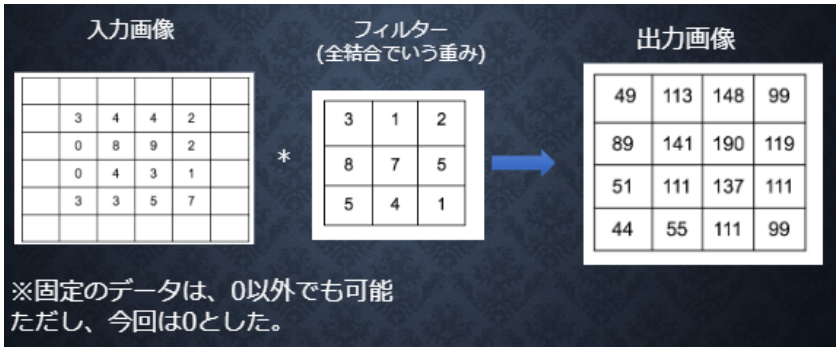
入力画像（4 × 4)、フィルター（3 × 3)、出力画像（2 × 2）となる。入力画像（4 × 4)、フィルター（2 × 2)、出力画像（3 × 3）となる。



4.1.2　パティング

畳み込み演算を繰り返すと、一般的に出力画像が小さくなっていく。コンピュータ演算では、2 進数はきれいな数（扱いやすい数）である。フィルターを通じて次元が落ちる際に、このパティングを前処理で行い、フィルター通過後においてもコンピュータ演算にとって、扱いやすい次元を保てるようになる。

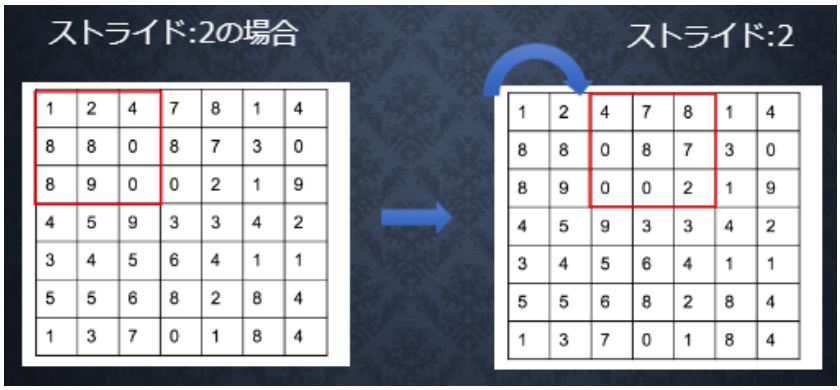
以下の例では、パティングを行って元入力画像 (4 × 4) の周囲に 0 データを入力し画像データを拡大 (6 × 6) させた[*1]。これによって、フィルター (3 × 3) を通しても出力画像の大きさ (4 × 4) が元の入力画像データを同じになり、小さくなることを防ぐことができた。



[*1] 一番近いところの数値を採用する場合もある。

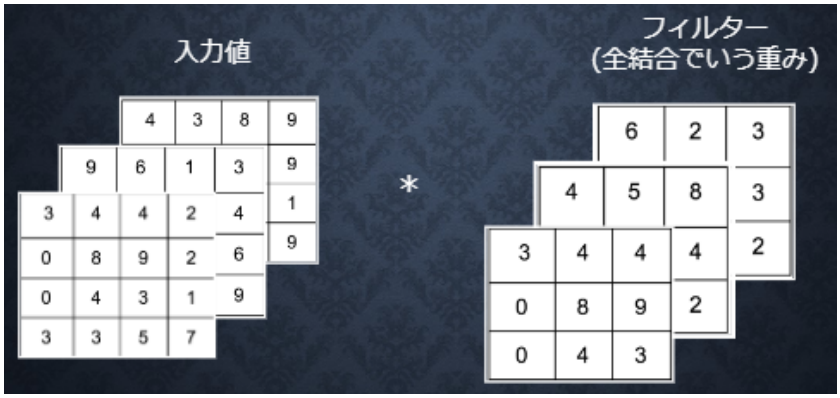
4.1.3　ストライド

フィルターが動く幅のこと。ストライドが 2 の場合は、入力画像に対してフィルターを 2pixel 分づつずらすことに相当する。この性質から、ストライドが大きいほど出力画像が小さくなることになる。



4.1.4　チャンネル

チャンネルはフィルターの数に相当する。



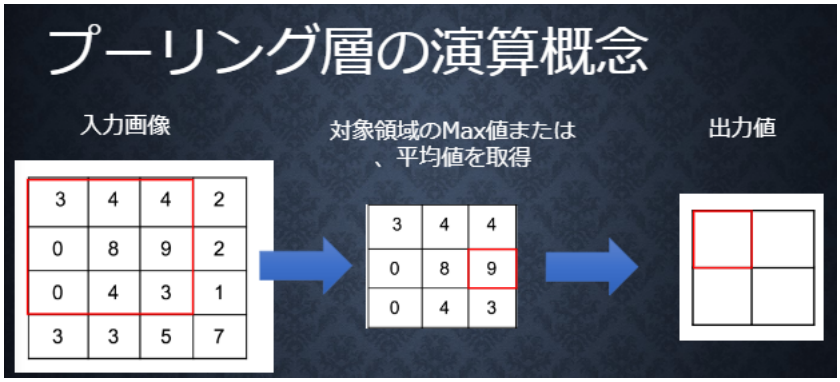
4.1.5　全結合で画像を学習した際の課題

通常の全結合層のデメリットは、画像の場合、縦、横、チャンネルの 3 次元データだが、結局は 1 次元のデータとして処理されることにある。このため、RGB の各チャンネル間の関連性が、学習には反映されないということになる。

本章で説明した CNN によって、入力画像に対してフィルターを少しずつずらすこと（畳み込み）で、次元間の繋がりを保ちつつ特徴量を抽出することができるようになったのである。

4.2　プーリング

畳み込み層と併用して用いられる層である。畳み込み層との違いは重みがないこと。唯一する演算は、Max Pooling（対象領域の最大値を取得）か Average Pooling（平均値を取得）のみ。このプーリング手法においても画像データ等の特徴量がつかめる特性がある。



■確認テスト

下サイズ 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。

■答え

$$O_H = \frac{\text{画像の高さ} + 2 \times \text{パティングの高さ} - \text{フィルターの高さ}}{\text{ストライド}} + 1 = \frac{6 + 2 \times 1 - 2}{1} + 1 = 7$$

$$O_W = \frac{\text{画像の幅} + 2 \times \text{パティングの幅} - \text{フィルターの幅}}{\text{ストライド}} + 1 = \frac{6 + 2 \times 1 - 2}{1} + 1 = 7$$

■実装演習成果

```
# coding: utf-8
# # simple convolution network
# ## image to column

# In[1]:
import sys, os
sys.path.append(os.pardir)
import pickle
import numpy as np
from collections import OrderedDict
from common import layers
from common import optimizer
from data.mnist import load_mnist
import matplotlib.pyplot as plt

# 画像データを２次元配列に変換
```

```
'''
input_data: 入力値
filter_h: フィルターの高さ
filter_w: フィルターの横幅
stride: ストライド
pad: パディング
'''
def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    # N: number, C: channel, H: height, W: width
    N, C, H, W = input_data.shape
    # 切り捨て除算
    out_h = (H + 2 * pad - filter_h)//stride + 1
    out_w = (W + 2 * pad - filter_w)//stride + 1

    img = np.pad(input_data, [(0,0), (0,0), (pad, pad), (pad, pad)], 'constant')
    col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))

    for y in range(filter_h):
        y_max = y + stride * out_h
        for x in range(filter_w):
            x_max = x + stride * out_w
            col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]

    col = col.transpose(0, 4, 5, 1, 2, 3) # (N, C, filter_h, filter_w, out_h, out_w)
        -> (N, filter_w, out_h, out_w, C, filter_h)

    col = col.reshape(N * out_h * out_w, -1)
    return col
# ------------------------------------------------------------------------------
# ## [try] の処理を確認しよう im2col
# ・関数内で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 transpose<br>
# ・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
------------------------------------------------------------------------------
# In[2]:

# の処理確認 im2col
input_data = np.random.rand(2, 1, 4, 4)*100//1 # number, channel, height, を表す width
print('========== input_data ==========\n', input_data)
print('==============================')
filter_h = 3
filter_w = 3
stride = 1
pad = 0
col = im2col(input_data, filter_h=filter_h, filter_w=filter_w, stride=stride, pad=pad)
print('============= col ==============\n', col)
print('==============================')
```

```
62
63 # ## column to image
64
65 # In[3]:
66
67 # ２次元配列を画像データに変換
68 def col2im(col, input_shape, filter_h, filter_w, stride=1, pad=0):
69     # N: number, C: channel, H: height, W: width
70     N, C, H, W = input_shape
71     # 切り捨て除算
72     out_h = (H + 2 * pad - filter_h)//stride + 1
73     out_w = (W + 2 * pad - filter_w)//stride + 1
74     col = col.reshape(N, out_h, out_w, C, filter_h, filter_w).transpose(0, 3, 4, 5, 1,
           2) # (N, filter_h, filter_w, out_h, out_w, C)
75
76     img = np.zeros((N, C, H + 2 * pad + stride - 1, W + 2 * pad + stride - 1))
77     for y in range(filter_h):
78         y_max = y + stride * out_h
79         for x in range(filter_w):
80             x_max = x + stride * out_w
81             img[:, :, y:y_max:stride, x:x_max:stride] += col[:, :, y, x, :, :]
82
83     return img[:, :, pad:H + pad, pad:W + pad]
84 # ------------------------------------------------------------------------------
85 # ## [try] の処理を確認しよう col2im
86 # ・の確認で出力したをに変換して確認しよう im2colcolimage
87 #
88 #
89 ------------------------------------------------------------------------------
90
91 # In[4]:
92
93 # ここにでの処理を書こう col2im
94 img = col2im(col, input_shape=input_data.shape, filter_h=filter_h, filter_w=filter_w,
       stride=stride, pad=pad)
95 print(img)
96
97 # ## convolution class
98
99 # In[5]:
100
101 class Convolution:
102     # W: フィルター, b: バイアス
103     def __init__(self, W, b, stride=1, pad=0):
104         self.W = W
105         self.b = b
```

```
106         self.stride = stride
107         self.pad = pad
108 
109         # 中間データ（時に使用）backward
110         self.x = None
111         self.col = None
112         self.col_W = None
113 
114         # フィルター・バイアスパラメータの勾配
115         self.dW = None
116         self.db = None
117 
118     def forward(self, x):
119         # FN: filter_number, C: channel, FH: filter_height, FW: filter_width
120         FN, C, FH, FW = self.W.shape
121         N, C, H, W = x.shape
122         # 出力値の height, width
123         out_h = 1 + int((H + 2 * self.pad - FH) / self.stride)
124         out_w = 1 + int((W + 2 * self.pad - FW) / self.stride)
125 
126         # を行列に変換 x
127         col = im2col(x, FH, FW, self.stride, self.pad)
128         # フィルターをに合わせた行列に変換 x
129         col_W = self.W.reshape(FN, -1).T
130 
131         out = np.dot(col, col_W) + self.b
132         # 計算のために変えた形式を戻す
133         out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
134 
135         self.x = x
136         self.col = col
137         self.col_W = col_W
138 
139         return out
140 
141     def backward(self, dout):
142         FN, C, FH, FW = self.W.shape
143         dout = dout.transpose(0, 2, 3, 1).reshape(-1, FN)
144 
145         self.db = np.sum(dout, axis=0)
146         self.dW = np.dot(self.col.T, dout)
147         self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)
148 
149         dcol = np.dot(dout, self.col_W.T)
150         # を画像データに変換 dcol
151         dx = col2im(dcol, self.x.shape, FH, FW, self.stride, self.pad)
```

```
152
153         return dx
154
155 # ## pooling class
156
157 # In[6]:
158
159 class Pooling:
160     def __init__(self, pool_h, pool_w, stride=1, pad=0):
161         self.pool_h = pool_h
162         self.pool_w = pool_w
163         self.stride = stride
164         self.pad = pad
165
166         self.x = None
167         self.arg_max = None
168
169     def forward(self, x):
170         N, C, H, W = x.shape
171         out_h = int(1 + (H - self.pool_h) / self.stride)
172         out_w = int(1 + (W - self.pool_w) / self.stride)
173
174         # を行列に変換 x
175         col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
176         # プーリングのサイズに合わせてリサイズ
177         col = col.reshape(-1, self.pool_h*self.pool_w)
178
179         #プーリング max
180         arg_max = np.argmax(col, axis=1)
181         out = np.max(col, axis=1)
182         out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)
183
184         self.x = x
185         self.arg_max = arg_max
186
187         return out
188
189     def backward(self, dout):
190         dout = dout.transpose(0, 2, 3, 1)
191
192         pool_size = self.pool_h * self.pool_w
193         dmax = np.zeros((dout.size, pool_size))
194         dmax[np.arange(self.arg_max.size), self.arg_max.flatten()] = dout.flatten()
195         dmax = dmax.reshape(dout.shape + (pool_size,))
196
197         dcol = dmax.reshape(dmax.shape[0] * dmax.shape[1] * dmax.shape[2], -1)
```

```
198         dx = col2im(dcol, self.x.shape, self.pool_h, self.pool_w, self.stride, self.pad
                )
199
200         return dx
201
202 # ## sinple convolution network class
203
204 # In[12]:
205
206 class SimpleConvNet:
207     # conv - relu - pool - affine - relu - affine - softmax
208     def __init__(self, input_dim=(1, 28, 28), conv_param={'filter_num':30, '
            filter_size':5, 'pad':0, 'stride':1},
209                  hidden_size=100, output_size=10, weight_init_std=0.01):
210         filter_num = conv_param['filter_num']
211         filter_size = conv_param['filter_size']
212         filter_pad = conv_param['pad']
213         filter_stride = conv_param['stride']
214         input_size = input_dim[1]
215         conv_output_size = (input_size - filter_size + 2 * filter_pad) / filter_stride
                + 1
216         pool_output_size = int(filter_num * (conv_output_size / 2) * (conv_output_size
                / 2))
217
218         # 重みの初期化
219         self.params = {}
220         self.params['W1'] = weight_init_std * np.random.randn(filter_num, input_dim[0],
                 filter_size, filter_size)
221         self.params['b1'] = np.zeros(filter_num)
222         self.params['W2'] = weight_init_std * np.random.randn(pool_output_size,
                hidden_size)
223         self.params['b2'] = np.zeros(hidden_size)
224         self.params['W3'] = weight_init_std * np.random.randn(hidden_size, output_size)
225         self.params['b3'] = np.zeros(output_size)
226
227         # レイヤの生成
228         self.layers = OrderedDict()
229         self.layers['Conv1'] = layers.Convolution(self.params['W1'], self.params['b1'],
                 conv_param['stride'], conv_param['pad'])
230         self.layers['Relu1'] = layers.Relu()
231         self.layers['Pool1'] = layers.Pooling(pool_h=2, pool_w=2, stride=2)
232         self.layers['Affine1'] = layers.Affine(self.params['W2'], self.params['b2'])
233         self.layers['Relu2'] = layers.Relu()
234         self.layers['Affine2'] = layers.Affine(self.params['W3'], self.params['b3'])
235
236         self.last_layer = layers.SoftmaxWithLoss()
```

```
237
238     def predict(self, x):
239         for key in self.layers.keys():
240             x = self.layers[key].forward(x)
241         return x
242
243     def loss(self, x, d):
244         y = self.predict(x)
245         return self.last_layer.forward(y, d)
246
247     def accuracy(self, x, d, batch_size=100):
248         if d.ndim != 1 : d = np.argmax(d, axis=1)
249
250         acc = 0.0
251
252         for i in range(int(x.shape[0] / batch_size)):
253             tx = x[i*batch_size:(i+1)*batch_size]
254             td = d[i*batch_size:(i+1)*batch_size]
255             y = self.predict(tx)
256             y = np.argmax(y, axis=1)
257             acc += np.sum(y == td)
258
259         return acc / x.shape[0]
260
261     def gradient(self, x, d):
262         # forward
263         self.loss(x, d)
264
265         # backward
266         dout = 1
267         dout = self.last_layer.backward(dout)
268         layers = list(self.layers.values())
269
270         layers.reverse()
271         for layer in layers:
272             dout = layer.backward(dout)
273
274         # 設定
275         grad = {}
276         grad['W1'], grad['b1'] = self.layers['Conv1'].dW, self.layers['Conv1'].db
277         grad['W2'], grad['b2'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
278         grad['W3'], grad['b3'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
279
280         return grad
281
282 # In[ ]:
```

```
283
284 from common import optimizer
285
286 # データの読み込み
287 (x_train, d_train), (x_test, d_test) = load_mnist(flatten=False)
288
289 printデータ読み込み完了("")
290
291 # 処理に時間のかかる場合はデータを削減
292 x_train, d_train = x_train[:5000], d_train[:5000]
293 x_test, d_test = x_test[:1000], d_test[:1000]
294
295 network = SimpleConvNet(input_dim=(1,28,28), conv_param = {'filter_num': 30, '
        filter_size': 5, 'pad': 0, 'stride': 1},
296                         hidden_size=100, output_size=10, weight_init_std=0.01)
297
298 optimizer = optimizer.Adam()
299
300 iters_num = 1000
301 train_size = x_train.shape[0]
302 batch_size = 100
303
304 train_loss_list = []
305 accuracies_train = []
306 accuracies_test = []
307
308 plot_interval=10
309
310 for i in range(iters_num):
311     batch_mask = np.random.choice(train_size, batch_size)
312     x_batch = x_train[batch_mask]
313     d_batch = d_train[batch_mask]
314
315     grad = network.gradient(x_batch, d_batch)
316     optimizer.update(network.params, grad)
317
318     loss = network.loss(x_batch, d_batch)
319     train_loss_list.append(loss)
320
321     if (i+1) % plot_interval == 0:
322         accr_train = network.accuracy(x_train, d_train)
323         accr_test = network.accuracy(x_test, d_test)
324         accuracies_train.append(accr_train)
325         accuracies_test.append(accr_test)
326
327         print('Generation: ' + str(i+1) + '. 正答率トレーニング
```

```
            () = ' + str(accr_train))
328         print(' : ' + str(i+1) + '. 正答率テスト () = ' + str(accr_test))
329
330 lists = range(0, iters_num, plot_interval)
331 plt.plot(lists, accuracies_train, label="training set")
332 plt.plot(lists, accuracies_test, label="test set")
333 plt.legend(loc="lower right")
334 plt.title("accuracy")
335 plt.xlabel("count")
336 plt.ylabel("accuracy")
337 plt.ylim(0, 1.0)
338 # グラフの表示
339 plt.show()
```

4.3 サマリ

CNN も原点は NN にあり、その相違はいかにして入力データの次元を保ったまま特徴量を抽出するかにある。具体的には、畳み込み演算（フィルター）やプーリング等を通じて演算を行い、最終的に人間が認識できるように全結合層、活性化関数を通じて出力層に導いていくことになる。

また入力データは、画像の他に音声などに対しても有効である。実装の段階で、処理速度を高速化するために入力データの配列を特徴的な並びに与えなおすことで実現できることが知られている (https://medium.com/@_init_/an-illustrated-explanation-of-performing-2d-convolutions-using-matrix-multiplications-1e8de8cd2544)。

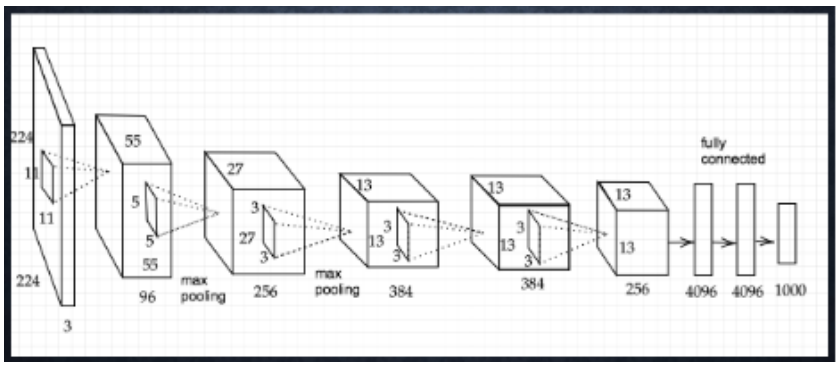
第 5 章

最新の CNN

5.1 AlexNet

2010 年から開催された ILSVRC における ImageNet のデータセットを用いた画像認識のコンペティション（精度を競争）において、2012 年に優勝したトロント大学（チーム名：SuperVision, ）が用いた CNN である。いわゆる深層学習の先駆けともいわれて、同大会では 2 位に 10% 以上の精度の差をつけた。

1. モデル構造：5 層の畳み込み層及びプーリング層を有するとともに、それに続く 3 層の全結合層
2. 過学習抑制施策：サイズ 4096 の全結合層の出力にドロップアウトと使用[*1]



[*1] 通常の NN では中間層に用いられることが多いが、CNN では全結合層に用いられることが多い。

第 6 章

Appendix

6.1 AI の弱点は外挿

内挿は、多項式などで近似して補間できる。一方、観測範囲外の値は信用できない（外挿は失敗しがち）。

AI も同様に、データの観測範囲内ならば、内挿で済むから性能が高い（凸包のデータセット）。逆に言えば、凸包を広げてくれないデータは、価値が少ないといえる。凸包を広げてくれるデータが、AI の性能を高める[*1]。

6.2 次元の呪い

しかし現実はそう甘くはなく、「次元の呪い」と呼ばれる事象が知られている。これは、高次元においては、凸包は簡単に広がらないことを意味している。高次元空間では、新規外れ値（一見すると高価値データ）を得ても凸包（高次元多面体）には、細い角のような領域しか加わらないため。

例えば、実務上低次元の特徴量データのデータセットならまんべんなく揃えられるが、仮に 7 次元（これでも高次元とは言えないが）だとしても、個々の特徴量の大小の組み合わせを網羅的に揃える場合には、$2^7 = 128$ 個必要。つまり 7 次元ですが、まんデータを取得するには、コストが莫大となる。

6.3 AI 巨大企業達のデータ争奪戦

1. 人々のデータを扱う
   データの項目が、そのまま次元になるので超高次元
2. データの多様性にこそ価値がある。
3. 変なデータを、とにかく集めろ
   （a）あらゆる国
   （b）あらゆる少数派
   （c）大量の会員数を持つ企業が強い

[*1] 中田亨,『多様性工学』（日科技連，2021）

(Ex.)　病気を予測する AI を作るには？

1. 既に収集できている多数派のデータよりも、未知の地域や少数派の属性の人のデータを集めるべき
2. 国境を超えた収集　⇒　世界競争
3. 少数派のデータの獲得　⇒　先人争い、会員囲み

6.4　Summary

1. AI 時代は、データの多様性の価値が重い
   かつては、外れ値は使い道がないので、むしろ邪魔。モデル制作では無視して使わないことが多かった。
2. データ項目数の多い高次元データを扱うならば、ますます範囲外のデータが貴重になる。
3. 多数派のデータばかり見ていても、ありがたくない。

参考文献

[1] 奥村晴彦，黒木裕介『LaTeX 2_ε 美文書作成入門 第 7 版』（技術評論社，2017）
[2] 加藤公一，『機械学習のエッセンス』（SB クリエイティブ，2018）
[3] 大重美幸，『Python 3 入門ノート』（ソーテック，2018）
[4] 大関真之，『機械学習入門』（オーム，2018）
[5] Andreas C. Müller et al.,『Python ではじめる機械学習』（オライリージャパン，2018）
[6] 加藤公一（監修），『機械学習図鑑』（翔泳社，2019）
[7] 岡谷貴之，『深層学習』（講談社，2015）
[8] 竹内一郎，『サポートベクトルマシン』、講談社，2015）
[9] 斎藤康毅，『ゼロから作る Deep Learning』（オライリージャパン，2019）
[10] Guido van Rossum，『Python チュートリアル（第 4 版)』（オライリージャパン，2021）
[11] 中田亨，『多様性工学』（日科技連，2021）