

計算機科学実験及演習4：画像認識

－ ニューラルネットワークによる画像認識 －

担当：飯山将晃 (TA: 世古雄大, 東 晃平)

2017 年度版 (Ver1.1)

目 次

1	はじめに	2
2	演習にあたって	2
3	Python	3
3.1	学生実験室環境での Python 利用	3
3.1.1	pip による外部ライブラリのインストール	3
3.1.2	PyCharm	4
3.2	Python チュートリアル	5
3.3	NumPy チュートリアル	5
4	3 層ニューラルネットワークを用いた MNIST 手書き数字データ認識	5
4.1	MNIST 手書き数字データ	5
4.2	3 層ニューラルネットワークを用いた多クラス識別	7
4.2.1	3 層ニューラルネットワーク	7
4.2.2	[課題 1] 3 層ニューラルネットワークの構築 (順伝播)	9
4.3	ニューラルネットワークの学習	9
4.3.1	損失関数	10
4.3.2	ミニバッチ	10
4.3.3	[課題 2] ミニバッチ対応&クロスエントロピー誤差の計算	10
4.3.4	誤差逆伝播法による損失関数の勾配の計算	11
4.3.5	[課題 3] 誤差逆伝播法による 3 層ニューラルネットワークの学習	13
4.3.6	[課題 4] 手書き数字画像識別器の構築	13
5	発展課題 (以降, 追加する可能性があります)	13
5.1	[発展課題] 手書き数字画像認識コンテスト	13
5.2	[発展課題] 活性化関数	14
5.3	[発展課題] Dropout	14
5.4	[発展課題] Batch Normalization	15

5.5	[発展課題] 最適化手法の改良	16
5.6	[発展課題] カラー画像への拡張	18
5.6.1	CIFAR-10 一般画像物体認識用画像データ	18
5.7	[発展課題] 畳み込み層	19
5.7.1	入力層	19
5.7.2	画像の畳み込み	19
5.7.3	多チャンネル画像の畳み込み	20
5.7.4	畳み込み層の出力の変換	21
5.7.5	畳み込み層の計算（順伝播）と実装	21
5.7.6	畳み込み層の計算（逆伝播）	22
5.8	[発展課題] プーリング層	23

1 はじめに

本演習では、計算機によるパターン情報処理の一例として、画像に対する認識技術、特にニューラルネットワークを用いた画像認識技術をプログラミング演習を通して学習する。具体的には、手書き数字の認識・物体認識を題材とし、データの取り扱い、ニューラルネットワークによる多クラス識別器の学習と利用に必要な基礎技術を習得することを目指す。

2 演習にあたって

MNIST の手書き数字データ [1] (図 1(a)) と CIFAR-10 画像データ [2] (図 1(b)) を題材に、画像を入力として受け取り、その画像に対応するラベル (MNIST の場合は「0」～「9」の数字、CIFAR10 の場合は「bird」や「ship」といった 10 種類の物体名称) を出力する多クラス識別器を構築することが本演習のゴールである。画像とその画像に対応するラベルからなる**教師データ**を用いた**教師付学習**により識別器を構築する。識別器を構築する方法として、サポートベクターマシンや決定木 (やその拡張版である Random Forests) などがあるが、本演習ではクラス識別を行う代表的手法のひとつであるニューラルネットワークを題材として用いる。

ニューラルネットワークについては、caffe や tensorflow, chainer など整備されたフレームワークが存在し、多層のニューラルネットワークを用いた学習 (深層学習) についても比較的短い学習コストで利用することができるようになっている。しかしながら、本演習ではあえてそれらのフレームワークを用いず、1 からニューラルネットワークを実装することで、その仕組みを学ぶ。とはいえ、行列演算などのローレベルの処理まで 1 から実装することは限られた演習時間では困難であるので、基本的な演算については既存のライブラリ (numpy) を利用することとする。後述する one hot vector 表記やクロスエントロピー誤差算出については機械学習ライブラリ (scikit-learn など) で用意されているが、あえて機械学習のライブラリは使わずに実装して頂きたい。

実装には、近年機械学習・パターン認識で主流となっている Python を用いる。それ以外のプログラミング言語を用いて実装することを妨げることはないが、以降の説明は Python

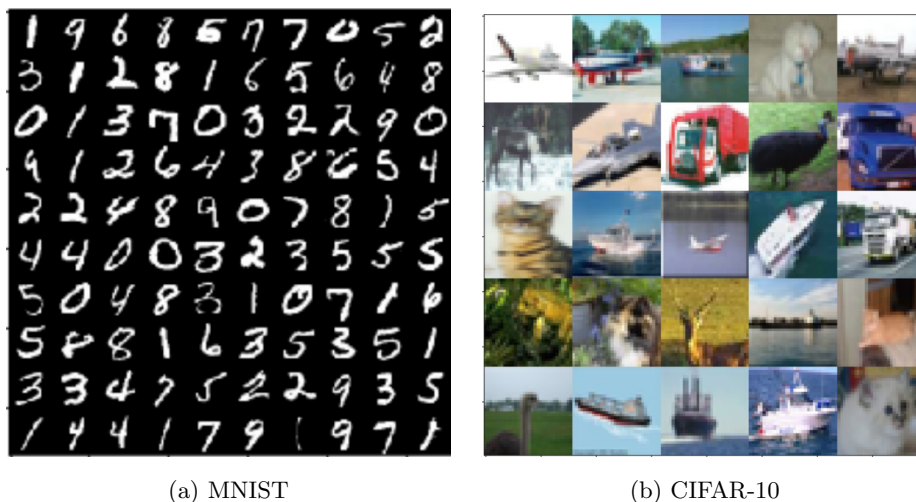


図 1: MNIST と CIFAR-10 データセット

(と Python で使えるライブラリ) での実装を想定した記述となっているため, Python 以外の言語での実装は at your own risk で行って頂きたい.

演習は必修課題 4 つと, 発展課題からなる. 発展課題については受講者の状況に応じて追加する可能性がある. 〆切・提出方法については Web サイト上でアナウンスするが, 早めの提出を強く薦める.

3 Python

3.1 学生実験室環境での Python 利用

学生実験室 PC の Python は 2.7 系である.

以下のようなファイルを適当なテキストエディタで作成し, `python helloworld.py` で実行する.

```
helloworld.py
print "Hello World"
```

3.1.1 pip による外部ライブラリのインストール

初期状態では, `numpy` など演習に必要なライブラリがインストールされていない. そこで, `pip` コマンドを用いて必要なライブラリをインストールする. 端末で以下のコマンド¹を順に実行すること.

¹個人 PC にインストールする場合はおそらく `-user --ignore-installed` オプションは不要です.

pip でインストールするライブラリ

```
pip install numpy --user --ignore-installed
pip install jupyter --user --ignore-installed
pip install matplotlib --user --ignore-installed
pip install python-mnist --user --ignore-installed
```

3.1.2 PyCharm

Python の IDE (統合開発環境) として, PyCharm を用いることもできる. <https://www.jetbrains.com/pycharm/> より “Community” 版をダウンロードすると, 「ダウンロード」フォルダに pycharm-community-2017.2.4.tar.gz がダウンロードされているはずなので, 端末で以下のコマンドを順に実行しインストールする.

PyCharm のインストール

```
cd ~/ダウンロード
tar xzf pycharm-community-2017.2.4.tar.gz
rm pycharm-community-2017.2.4.tar.gz
cd pycharm-community-2017.2.4/bin
./pycharm.sh
```

すると, インストールウィザードが開くのであとはそのまま OK 押していけばインストールが完了し, PyCharm が起動する. 次回以降は, アプリケーション一覧から起動できるようになる.

PyCharm で Hello world を表示させるまで

1. “Create New Project” もしくは “File ⇒ New Project” でプロジェクトを作成する.
2. “File ⇒ New ⇒ Python File” で新たに Python ファイルを作成する.
3. 適当にコードを書いたあと, “Run ⇒ Run...” を選んで, 実行したいソースコードを選択して実行.

なお, ひとつのプロジェクト内で複数のソースコードを編集できる. 実行の際は, “Run ⇒ Run...” で実行するファイルを選択すること.

[おまけ] PyCharm で Jupyter Notebook Jupyter Notebook はインタラクティブな Python 実行環境である. 逐次実行しながら結果を確認できるので, Python の学習やコードの動作確認に役に立つ. PyCharm 上でも Jupyter Notebook を利用することができ, “File ⇒ New ⇒ Jupyter Notebook” で新たな Notebook を作成できる. セル内にコードを記述し, 実行ボタン (横向き緑の三角形) かもしくは Shift-Enter でセル内のコードを実行することができる. 実行時に “Please, enter your Jupyter Notebook URL and authentication token” との表示がでた場合は, 一旦 「Cancel」 ボタンを押して, 直後に吹き出しで表示

される”Cannot connect to Jupyter Notebook. Run Jupyter Notebook”をクリックして Jupyter Notebook を起動すれば良い。

3.2 Python チュートリアル

Python を用いたプログラミングについては、以下のチュートリアル [3] を利用して自習して頂きたい。チュートリアルの全てを学習する必要は無く、一通り目次に目を通して機能を把握し、残りは必要に応じて参照すれば本演習の範囲を実装するには十分である。

Python チュートリアル <https://docs.python.jp/2.7/tutorial/index.html>

3.3 NumPy チュートリアル

NumPy は科学計算用の Python ライブラリで、N 次元配列（行列）や行列演算、乱数生成など基本的な科学計算用のメソッドが用意されている。本演習では NumPy の N 次元配列 (numpy.ndarray) を多用してニューラルネットワークを構築する。

NumPy の使い方については、以下のチュートリアル [4] を参照のこと。

NumPy Quickstart Tutorial <https://docs.scipy.org/doc/numpy-dev/user/quickstart.html>

4 3層ニューラルネットワークを用いた MNIST 手書き数字データ認識

本演習の必修課題（最低ライン）は、MNIST の手書き数字データを学習データとして、10 クラス識別器を 3 層ニューラルネットワークを用いて構築することである。本章では、これを構築するために必要な知識と基本的な方針を解説する。

4.1 MNIST 手書き数字データ

MNIST 手書き数字データセットは、60000 枚の教師用画像と 10000 枚のテスト用画像からなる手書き数字認識用のデータセットである。画像はすべて 28×28 画素の 1 チャンネル画像であり、各画素には 0～255 までの整数値が格納されている。各画像には「0」～「9」までの数字のうちいずれか 1 つが描かれている。

データの準備 <http://yann.lecun.com/exdb/mnist/> より、4つのファイルをダウンロードする。

データのダウンロード

```
mkdir -p ~/le4nn
cd ~/le4nn
wget http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
wget http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
wget http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
wget http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
```

これらのファイルは gzip 形式で圧縮されているので、gunzip コマンドで解凍する。

```
cd ~/le4nn
gunzip *.gz
```

Python でのデータの読み込み

sample1.py

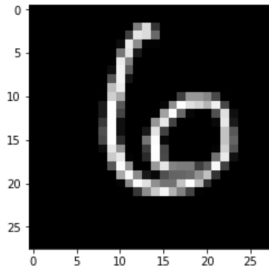
```
import numpy as np
from mnist import MNIST
mnndata = MNIST("/export/home/*****/le4nn/")
# ""の中はtrain-images-idx3-ubyteとtrain-labels-idx1-ubyteを置いたディレクトリ名とすること
X, Y = mnndata.load_training()
X = np.array(X)
X = X.reshape((X.shape[0], 28, 28))
Y = np.array(Y)
```

上のサンプルコードにおいて、変数 X には MNIST の画像データ全てが格納されており、 X は $60000 \times 28 \times 28$ の `numpy.ndarray` の 3 次元配列である。 i 番目の画像は $X[i]$ で取り出すことができる。画像 1 枚は `numpy` の 2 次元配列で表現されており、画像の左上を原点として、左から右を x 軸、上から下を y 軸としている。また、各画像に描かれている数字については変数 Y に格納されており、 i 番目の画像に描かれている数字は $Y[i]$ で取り出すことができる。なお、`mnndata.load_training()` の代わりに `mnndata.load_testing()` を呼び出すことにより、テスト用の画像データを読み込むことができる。

[練習] 画像の表示 `sample1.py` に続けて以下のコードを実行すると、`idx` 番目の画像とそれに対する正解ラベルを表示することができる。

sample1.py の続き

```
import matplotlib.pyplot as plt
from pylab import cm
idx = 100
plt.imshow(X[idx], cmap=cm.gray)
plt.show()
print Y[idx]
```



4.2 3層ニューラルネットワークを用いた多クラス識別

1枚の画像を d 次元ベクトル \mathbf{x} で表す. MNIST 画像は 28×28 の2次元配列であるがこれを左上から x 軸, y 軸の順に並べかえることにより $d = 784$ 次元のベクトルとする. 画像 \mathbf{x} を入力として, \mathbf{x} に対応する数字 $y \in \{0, \dots, 9\}$ を出力する関数 $y = f(\mathbf{x})$ を構築する. 関数 f の構築にあたり, \mathbf{x} と y とのペアの集合 $\{(\mathbf{x}_i, y_i)\}$ ($i = 1, \dots, N$) を教師データとして用い, 関数 f として, 課題1, 課題2では3層のニューラルネットワークを用いる.

4.2.1 3層ニューラルネットワーク

3層ニューラルネットワークの模式図を図2に示す. (多クラス識別における) 3層ニューラルネットワークは, 入力次元数 d と同じ数のノードからなる入力層, クラス数 (MNIST の場合は0~9までの10クラス) と同じ数のノードからなる出力層, および入力層と出力層の間にある中間層の3層で構成される.

入力層の各ノードは入力 \mathbf{x} の各要素 x_i ($i = 1, \dots, d$) に対応し, x_i を入力として x_i をそのまま出力する.

中間層は M 個のノードからなり, 中間層の各ノード j ($j = 1, \dots, M$) は入力層の線形和を入力として受け取り次式で表される出力 $y_j^{(1)}$ を返す.

$$y_j^{(1)} = a \left(\sum_{i=1}^d w_{ji}^{(1)} x_i + b_j^{(1)} \right) \quad (1)$$

ここで, 関数 $a(t)$ は**活性化関数**と呼ばれ, **シグモイド関数**

$$a(t) = \frac{1}{1 + \exp(-t)} \quad (2)$$

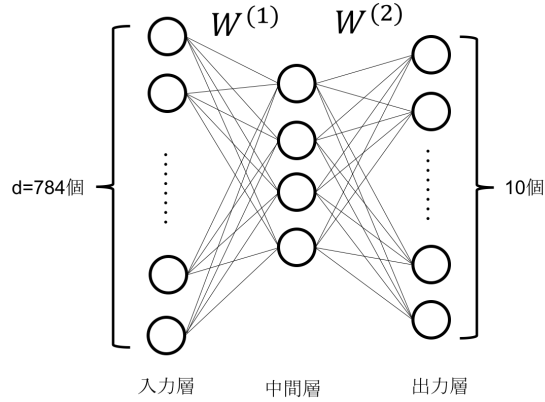


図 2: 3 層ニューラルネットワーク

が良く用いられている。線形和の重みを $\mathbf{w}_j^{(1)} = (w_{1j}^{(1)}, \dots, w_{dj}^{(1)})^T$ とすれば式 1 は $\mathbf{w}_j^{(1)}$ と \mathbf{x} との内積を用いて以下のようにシンプルに書ける。

$$y_j^{(1)} = a(\mathbf{w}_j^{(1)} \cdot \mathbf{x} + b_j^{(1)}) \quad (3)$$

さらに、線形和の重み $\mathbf{w}_j^{(1)}$ を j 行目の成分とする M 行 d 列の行列 $W^{(1)}$ と M 次元ベクトル $\mathbf{b}^{(1)} = (b_1^{(1)}, \dots, b_d^{(1)})^T$ を用いて中間層への M 個の入力を $W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$ と書くことができる。

出力層は C 個 (C はクラス数) のノードからなり、中間層の出力 $y_j^{(1)}$ の線形和を入力とする。

$$a_k = \mathbf{w}_k^{(2)} \cdot \mathbf{y}^{(1)} + b_k^{(2)} \quad (4)$$

ここで、 $\mathbf{y}^{(1)} = (y_1^{(1)}, \dots, y_M^{(1)})^T$ である。また、 C 個の入力 $\mathbf{a} = (a_1, \dots, a_C)^T$ を、線形和の重み $\mathbf{w}_k^{(2)}$ を k 行目の成分とする C 行 M 列の行列 $W^{(2)}$ と C 次元ベクトル $\mathbf{b}^{(2)} = (b_1^{(2)}, \dots, b_d^{(2)})^T$ を用いて、 $\mathbf{a} = W^{(2)}\mathbf{y}^{(1)} + \mathbf{b}^{(2)}$ と書くことができる。

出力層における活性化関数として**ソフトマックス関数**が用いられる。 C 個の入力を a_i ($i = 1, \dots, C$) とし、ソフトマックス関数を用いて出力層の出力 $y_i^{(2)}$ ($i = 1, \dots, C$) を次式で得る²。

$$y_i^{(2)} = \frac{\exp(a_i - \alpha)}{\sum_{j=1}^C \exp(a_j - \alpha)} \quad (5)$$

$$\alpha = \max a_i \quad (6)$$

出力層の出力 $y_i^{(2)}$ ($i = 1, \dots, C$) は入力 \mathbf{x} がクラス i に属する**尤度**を表し、 $y_i^{(2)}$ が最大となる i を認識結果 y として出力する。

²ソフトマックス関数の定義では α は存在しませんが、数値計算時のオーバーフローに対処するため α を導入しています。

4.2.2 [課題 1] 3 層ニューラルネットワークの構築 (順伝播)

MNIST の画像 1 枚を入力とし、3 層ニューラルネットワークを用いて、0~9 の値のうち 1 つを出力するプログラムを作成せよ。

- キーボードから 0~9999 の整数を入力 i として受け取り、0~9 の整数を標準出力に出力すること。
- MNIST のテストデータ 10000 枚の画像のうち i 番目の画像を入力画像として用いる。
- MNIST の画像サイズ (28×28)、画像枚数 (10000 枚)、クラス数 ($C = 10$) は既知とする。ただし、後々の改良のため変更可能な仕様にしておくことを薦める。
- 中間層のノード数 M は自由に決めて良い。
- 重み $W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}$ については乱数で決定すること。ここでは、手前の層のノード数を N として $1/N$ を分散とする平均 0 の正規分布で与えることとする。適切な重みを設定しないため、課題 1 の段階では入力に対してデタラメな認識結果を返す。ただし、実行する度に同じ結果を出力するよう乱数のシードを固定すること³。

(参考) 実装の方針 後の課題のために、処理を、前処理・入力層・中間層への入力を計算する層 (全結合層)・シグモイド関数・出力層への入力を計算する層 (全結合層)・ソフトマックス関数・後処理、に分割して実装することをお勧めする。

- 「入力層」では MNIST の画像を 784 次元のベクトルに変換する
- 「中間層への入力を計算する層」と「出力層への入力を計算する層」はパラメータは違えど処理は同じ (全結合層)。多次元の入力を受け取ってその線形和を出力する。
- 「シグモイド関数」の実装にあたっては、多次元ベクトルを入力とできるようにすること。python では for 文を用いると極端に処理速度が遅くなるので、for 文を使わずに済む工夫をすること。

4.3 ニューラルネットワークの学習

教師データを用いて、重み $W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}$ を学習する手法を解説する。基本的な学習アルゴリズムを以下に示す。

1. 適当な値で重みを初期化 (課題 1 で実装済み)
2. 定められた繰り返し回数に達するまで 3~5 を繰り返す
3. 教師データに対する出力を計算 (課題 1 で実装済み)
4. 3 の結果と正解との誤差を計算 (**損失関数**)
5. **誤差逆伝播法**により重みを修正

³乱数生成前に `numpy.random.seed(好きな数字)` を実行すれば良い。

4.3.1 損失関数

one-hot vector 教師データにおいて x に対する正解 y はクラスラベル (0~9 までの値) なのに対し, ニューラルネットワークの出力層の出力は C 次元のベクトル $y^{(2)}$ である。そこで, 教師データの正解を正解ラベルを 1, 残りを 0 とした C 次元ベクトルで表記することを考える。例えば, ある入力に対する正解が $y = 3$ である場合, これを 10 次元ベクトル $(0, 0, 0, 1, 0, 0, 0, 0, 0, 0)$ で表記する。この表記法を **one-hot vector 表記** と呼ぶ。

クロスエントロピー誤差 多クラス識別における損失関数として, 次式で表されるクロスエントロピー誤差がよく用いられる。

$$E = \sum_{k=1}^C -y_k \log y_k^{(2)} \quad (7)$$

ここで, y_k は one-hot vector 表記における正解の k 番目の要素, $y_k^{(2)}$ は出力層における k 番目の要素の出力である。

4.3.2 ミニバッチ

教師データを用いて学習を行う際, 全ての教師データに対して損失関数の値を計算するのは計算量の観点から効率的ではない。そこで, 教師データの中からランダムに一部のデータを取り出し, そのデータに対してクロスエントロピー誤差を計算してその平均を全ての教師データに対する誤差の近似として用いる方法がとられる。これを**ミニバッチ学習**と呼び, 一度に取り出すデータの個数を**バッチサイズ**と呼ぶ。

$$E_n = \frac{1}{B} \sum_{i \in \text{ミニバッチ}} \sum_{k=1}^C -y_{i,k} \log y_{i,k}^{(2)} \quad (8)$$

B はバッチサイズ, $y_{i,k}$ と $y_{i,k}^{(2)}$ はそれぞれ i 番目のデータにおける y_k , $y_k^{(2)}$ である。

4.3.3 [課題 2] ミニバッチ対応&クロスエントロピー誤差の計算

[課題 1] のコードをベースに, ミニバッチ (=複数枚の画像) を入力可能とするように改良し, さらにクロスエントロピー誤差を計算するプログラムを作成せよ。

- MNIST のテスト画像 10000 枚の中からランダムに B 枚をミニバッチとして取り出すこと。
- クロスエントロピー誤差の平均を標準出力に出力すること。
- ニューラルネットワークの構造, 重みは課題 1 と同じでよい。
- バッチサイズ B は自由に決めて良い (100 程度がちょうどよい)。
- ミニバッチを取り出す処理はランダムに行う⁴。

⁴numpy.random.choice() 関数を利用すれば比較的簡単に実現できる。

4.3.4 誤差逆伝播法による損失関数の勾配の計算

重み $W^{(1)}, W^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}$ を修正するために、損失関数 E_n の勾配 ($\frac{\partial E_n}{\partial W^{(1)}}$ など) を誤差逆伝播法により計算する。勾配が計算できれば、手順5における重みの修正を

$$W^{(1)} \leftarrow W^{(1)} - \eta \frac{\partial E_n}{\partial W^{(1)}} \quad (9)$$

のように更新することができる。(η は後述する**学習率**である)

連鎖律 ある合成関数 $f(g(x))$ が与えられた場合、 x に対する f の偏微分 $\partial f / \partial x$ は、

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x} \quad (10)$$

によって与えられる。

3層ニューラルネットワークにおいて、入力から出力までの処理は順に、入力信号 \Rightarrow 全結合層 $W^{(1)}, \mathbf{b}^{(1)} \Rightarrow$ シグモイド関数 \Rightarrow 全結合層 $W^{(2)}, \mathbf{b}^{(2)} \Rightarrow$ Softmax 関数 \Rightarrow 損失関数 E_n の順に処理される。この一連の処理を合成関数であると考えれば、勾配 $\partial E_n / \partial W^{(1)}$ や $\partial E_n / \partial W^{(2)}$ の計算を、Softmax 関数の出力に対する損失関数の偏微分、Softmax 関数の偏微分、全結合層の偏微分、... を順に計算することによって実現することができる。

Softmax 関数+クロスエントロピー誤差の逆伝播 Softmax 関数層は C 個の入力 $\mathbf{a} = (a_1, \dots, a_C)^T$ を受け取り、 C 個の出力 $y_k^{(2)}$ ($k = 1, \dots, C$) を返し、クロスエントロピー誤差 E は $y_k^{(2)}$ を受け取り、誤差 E を出力する関数である。勾配 $\partial E / \partial a_k$ は、次式で与えられる (証明は割愛)。

$$\frac{\partial E}{\partial a_k} = y_k^{(2)} - y_k \quad (11)$$

また、 E_n は E の平均であるため、 $\partial E_n / \partial E = 1/B$ (B はバッチサイズ) となり、勾配 $\partial E_n / \partial a_k$ は、

$$\frac{\partial E_n}{\partial a_k} = \frac{y_k^{(2)} - y_k}{B} \quad (12)$$

となる。

全結合層の逆伝播 ベクトル \mathbf{x} を入力とし、行列 W 、ベクトル \mathbf{b} をパラメータとして、線形和 $\mathbf{y} = W\mathbf{x} + \mathbf{b}$ を出力する関数を考える。 $\partial E_n / \partial \mathbf{y}$ を用いて、

$$\frac{\partial E_n}{\partial \mathbf{x}} = W^T \frac{\partial E_n}{\partial \mathbf{y}} \quad (13)$$

$$\frac{\partial E_n}{\partial W} = \frac{\partial E_n}{\partial \mathbf{y}} \mathbf{x}^T \quad (14)$$

$$\frac{\partial E_n}{\partial \mathbf{b}} = \frac{\partial E_n}{\partial \mathbf{y}} \quad (15)$$

となる (証明は割愛)。ミニバッチの場合、すなわち B 個のベクトル $\mathbf{x}_1, \dots, \mathbf{x}_B$ とそれに対する出力 $\mathbf{y}_1, \dots, \mathbf{y}_B$ 、および出力に対する損失関数の勾配 $\partial E_N / \partial \mathbf{y}_i$ ($i = 1, \dots, B$) が

与えられた場合、 W や \mathbf{b} に対する損失関数の勾配は、 B 個のベクトル $\mathbf{x}_1, \dots, \mathbf{x}_B$ を各列に持つ行列を X , $\partial E_N / \partial \mathbf{y}_i$ を各列として持つ行列を $\partial E_N / \partial Y$ として、

$$\frac{\partial E_n}{\partial X} = W^T \frac{\partial E_n}{\partial Y} \quad (16)$$

$$\frac{\partial E_n}{\partial W} = \frac{\partial E_n}{\partial Y} X^T \quad (17)$$

$$\frac{\partial E_n}{\partial \mathbf{b}} = \text{rowSum} \left(\frac{\partial E_n}{\partial Y} \right) \quad \text{※行列} \frac{\partial E_n}{\partial Y} \text{の行ごとの和} \quad (18)$$

で与えられる。

シグモイド関数の逆伝播 シグモイド関数 $a(t) = 1/(1 + \exp(-t))$ の微分は、

$$a(t)' = (1 - a(t))a(t) \quad (19)$$

で与えられる。

処理手順 これまでの説明を踏まえて、ニューラルネットワークの学習の手続き 3~5 をまとめる

1. ミニバッチを作成
2. ミニバッチに対する出力を順伝播によって計算。
3. 損失関数の値 E_n を算出
4. 式 12 を用いて $\frac{\partial E_n}{\partial a_k}$ を計算。 $\frac{\partial E_n}{\partial a_k}$ は全部で $B \times C$ 個得られる。
5. 式 60, 61, 62 を用いて $\frac{\partial E_n}{\partial X}$, $\frac{\partial E_n}{\partial W}$, $\frac{\partial E_n}{\partial \mathbf{b}}$ を計算。式 60, 61, 62 内の $\frac{\partial E_n}{\partial Y}$ として、前の手順で得られた $\frac{\partial E_n}{\partial a_k}$ を用いる。ここで計算した、 $\frac{\partial E_n}{\partial W}$, $\frac{\partial E_n}{\partial \mathbf{b}}$ が、それぞれ $\frac{\partial E_n}{\partial W^{(2)}}$, $\frac{\partial E_n}{\partial \mathbf{b}^{(2)}}$ となる。
6. 式 19 を $\frac{\partial E_n}{\partial X}$ の各要素に適用。
7. 式 60, 61, 62 を用いて $\frac{\partial E_n}{\partial X}$, $\frac{\partial E_n}{\partial W}$, $\frac{\partial E_n}{\partial \mathbf{b}}$ を計算。式 60, 61, 62 内の $\frac{\partial E_n}{\partial Y}$ として、前の手順で得られたシグモイド関数の微分値を用いる。ここで計算した、 $\frac{\partial E_n}{\partial W}$, $\frac{\partial E_n}{\partial \mathbf{b}}$ が、それぞれ $\frac{\partial E_n}{\partial W^{(1)}}$, $\frac{\partial E_n}{\partial \mathbf{b}^{(1)}}$ となる。
8. 学習率 η を用いて、パラメータを更新。

$$W^{(1)} \leftarrow W^{(1)} - \eta \frac{\partial E_n}{\partial W^{(1)}} \quad (20)$$

$$W^{(2)} \leftarrow W^{(2)} - \eta \frac{\partial E_n}{\partial W^{(2)}} \quad (21)$$

$$\mathbf{b}^{(1)} \leftarrow \mathbf{b}^{(1)} - \eta \frac{\partial E_n}{\partial \mathbf{b}^{(1)}} \quad (22)$$

$$\mathbf{b}^{(2)} \leftarrow \mathbf{b}^{(2)} - \eta \frac{\partial E_n}{\partial \mathbf{b}^{(2)}} \quad (23)$$

4.3.5 [課題 3] 誤差逆伝播法による 3 層ニューラルネットワークの学習

[課題 2] のコードをベースに、3 層ニューラルネットワークのパラメータ $W^{(1)}, W^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}$ を学習するプログラムを作成せよ。

- ネットワークの構造、バッチサイズは課題 2 と同じで良い。
- 学習には MNIST の学習データ 60000 枚を用いること。
- 繰り返し回数は自由に決めて良い。教師データの数を N 、ミニバッチのサイズを B としたとき、 N/B 回の繰り返しを 1 **エポック** と呼び、通常はエポック数とミニバッチサイズで繰り返し回数を指定する。
- 学習率 η は自由に決めて良い (0.01 あたりが経験的に良さそう)
- 各エポックの処理が終了する毎に、クロスエントロピー誤差を標準出力に出力すること。(これにより学習がうまく進んでいるかどうかを確認することができる)
- 学習終了時に学習したパラメータをファイルに保存する機能を用意すること。フォーマットは自由。パラメータを numpy の配列 (ndarray) で実装している場合は `numpy.save()` や `numpy.savez()` 関数が利用できる。
- ファイルに保存したパラメータを読み込み、再利用できる機能を用意すること。(`numpy.load()` 関数が利用できる)

4.3.6 [課題 4] 手書き数字画像識別器の構築

MNIST のテスト画像 1 枚を入力とし、3 層ニューラルネットワークを用いて、0~9 の値のうち 1 つを出力するプログラムを作成せよ。

- 重みパラメータ $W^{(1)}, W^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}$ が 課題 3 で計算された重みであること以外は課題 1 と同じ仕様である。重みパラメータについてはファイルから読み込むようにすること。

5 発展課題 (以降、追加する可能性があります)

[発展課題] 以下の追加機能を実装せよ。どの順番で取り組んでも良い。

5.1 [発展課題] 手書き数字画像認識コンテスト

PandA の「リソース」に 10000 枚の新しい手書き数字画像データを用意している。(ファイル名は "le4MNIST_X.dump") 以下のコードより、numpy の 3 次元配列 (10000*28*28) として読み込むことができる。

```
import numpy as np
import cPickle # python 3 系の場合は import pickle としてください.
with open("le4nnMNIST_X.dump","rb") with f:
X = cPickle.load(f)
# python 3 系の場合 ⇒ X = pickle.load(f, encoding='bytes')
X = X.reshape((X.shape[0], 28,28))
```

これら 10000 枚の画像に対し、認識を行いその結果を出力せよ。出力ファイルのフォーマットは 10000 行のテキストファイルで、第 n 行に n 番目の画像に対する認識結果 (0~9 の数字のうちのひとつ) を記載すること。

本課題については提出を PandA ではなく、メールによる提出とする。le4dip@mm.media.kyoto-u.ac.jp 宛に

- 結果の出力ファイル (テキスト形式でメールに添付)
- ソースコード (メールに添付)
- 氏名 (メール本文に記述)
- ニックネーム (メール本文に記述)

を送ること。

認識精度をこちらで計算し <https://goo.gl/1YTrKu> にて公表する。なお、再提出は何度でも受けつけるが、認識精度の公表は 1 日に一度程度しか行わないので慎重に行うこと。(精度の計算、ランキングの更新は手動で行います…)

5.2 [発展課題] 活性化関数

活性化関数としてシグモイド関数の他に次式で挙げる ReLU もよく用いられる。

$$a(t) = \begin{cases} t & (t > 0) \\ 0 & (t \leq 0) \end{cases} \quad (24)$$

ReLU の微分は、

$$a(t)' = \begin{cases} 1 & (t > 0) \\ 0 & (t \leq 0) \end{cases} \quad (25)$$

で与えられる。

5.3 [発展課題] Dropout

Dropout は学習時に中間層のノードをランダムに選び、その出力を無視 (出力 = 0) して学習する手法である。無視するノードの選択は学習データ毎にランダムに行い、中間層全体のノード数 $\times \rho$ 個のノードの出力を無視する。

テスト時は、全てのノードの出力を無視せず、代わりに元の出力に $(1 - \rho)$ 倍したものを出力として用いる。このように、Dropout は学習時とテスト時でふるまいが異なるので、学習時かテスト時かを判定するフラグを用意しておく必要がある。

Dropout を活性化関数の一種と考えると、

$$a(t) = \begin{cases} t & (\text{ノードが無視されない場合}) \\ 0 & (\text{ノードが無視された場合}) \end{cases} \quad (26)$$

となり、Dropout の微分は、

$$a(t)' = \begin{cases} 1 & (\text{ノードが無視されない場合}) \\ 0 & (\text{ノードが無視された場合}) \end{cases} \quad (27)$$

で与えられる。

5.4 [発展課題] Batch Normalization

Batch Normalization はミニバッチに対する各ノードの出力が分散 1、平均 0 となるように正規化する処理である。ミニバッチサイズを B 、ミニバッチ内の各データに対するあるノードの出力を x_i ($i = 1, \dots, B$) とした場合、以下の式により x_i を y_i に変換する。

$$\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^B x_i \quad \text{※ミニバッチの平均} \quad (28)$$

$$\sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^B (x_i - \mu_B)^2 \quad \text{※ミニバッチの分散} \quad (29)$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \text{※ } x_i \text{ の正規化} \quad (30)$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \quad (31)$$

ここで、 γ と β の初期値はそれぞれ 1、0 であり、学習が進むにつれて適切な値に学習されていく。

Batch Normalization における逆伝播は以下の式を用いて計算できる。

$$\frac{\partial E_n}{\partial \hat{x}_i} = \frac{\partial E_n}{\partial y_i} \cdot \gamma \quad (32)$$

$$\frac{\partial E_n}{\partial \sigma_B^2} = \sum_{i=1}^B \frac{\partial E_n}{\partial \hat{x}_i} \cdot (x_i - \mu_B) \cdot \frac{-1}{2} (\sigma_B^2 + \epsilon)^{-3/2} \quad (33)$$

$$\frac{\partial E_n}{\partial \mu_B} = \left(\sum_{i=1}^B \frac{\partial E_n}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \right) + \frac{\partial E_n}{\partial \sigma_B^2} \cdot \frac{\sum_{i=1}^B -2(x_i - \mu_B)}{m} \quad (34)$$

$$\frac{\partial E_n}{\partial x_i} = \frac{\partial E_n}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial E_n}{\partial \sigma_B^2} \cdot \frac{2(x_i - \mu_B)}{m} + \frac{\partial E_n}{\partial \mu_B} \cdot \frac{1}{m} \quad (35)$$

$$\frac{\partial E_n}{\partial \gamma} = \sum_{i=1}^B \frac{\partial E_n}{\partial y_i} \cdot \hat{x}_i \quad (36)$$

$$\frac{\partial E_n}{\partial \beta} = \sum_{i=1}^B \frac{\partial E_n}{\partial y_i} \quad (37)$$

μ_B , σ_B^2 はミニバッチ毎に計算されるが、十分な数のミニバッチを用いて μ_B と σ_B^2 の期待値をそれぞれ求め、 $E[x_i]$, $Var[x_i]$ とする。

テスト時には、

$$y_i \leftarrow \frac{\gamma}{\sqrt{Var[x_i] + \epsilon}} \cdot x_i + \left(\beta - \frac{\gamma E[x_i]}{\sqrt{Var[x_i] + \epsilon}} \right) \quad (38)$$

を入力 x_i に対する出力として用いる。

5.5 [発展課題] 最適化手法の改良

4.3 節で述べた手法では重みの修正を学習率 η を用いて

$$W \leftarrow W - \eta \frac{\partial E_n}{\partial W} \quad (39)$$

で実現していた。この手法は**確率的勾配降下法 (SGD: Stochastic Gradient Descent)** と呼ばれている。これに対し、いくつかの最適化手法が提案されている。

慣性項 (Momentum) 付き SGD 慣性項 (Momentum) 付き SGD では、パラメータ W の更新量に前回の更新量の α 倍を加算する手法である。事前に設定する必要のあるパラメータは学習率 η と α である。($\eta = 0.01$, $\alpha = 0.9$ くらいがおすすめ)

$$\Delta W \leftarrow \alpha \Delta W - \eta \frac{\partial E_n}{\partial W} \quad (40)$$

$$W \leftarrow W + \Delta W \quad (41)$$

なお、 ΔW の初期値は $\mathbf{0}$ とする。

AdaGrad AdaGrad では、学習率を繰り返し毎に自動で調整する。最初は大きめの学習率からスタートし、学習が進むにつれて小さな学習率を用いるようになる。

$$h \leftarrow h + \frac{\partial E_n}{\partial W} \circ \frac{\partial E_n}{\partial W} \quad (42)$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial E_n}{\partial W} \quad (43)$$

ここで、 \circ はアダマール積（2つの行列の要素毎の積）を表す。事前に設定する必要があるパラメータは学習率の初期値 η と h の初期値 h_0 である。（ $\eta = 0.001$, $h_0 = 10^{-8}$ くらいがおすすめ）

RMSProp RMSProp は AdaGrad の改良版で、AdaGrad が学習開始時から現在までの勾配の二乗和を h として用いていたのに対し、RMSProp では勾配の二乗の指数移動平均を h として用いる。

$$h \leftarrow \rho h + (1 - \rho) \frac{\partial E_n}{\partial W} \circ \frac{\partial E_n}{\partial W} \quad (44)$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{h} + \epsilon} \frac{\partial E_n}{\partial W} \quad (45)$$

h の初期値は 0 である。事前に設定する必要があるパラメータは学習率の初期値 η , ρ , ϵ である。（ $\eta = 0.01$, $\rho = 0.99$, $\epsilon = 10^{-8}$ くらいがおすすめ）

AdaDelta AdaDelta は RMSProp や AdaGrad の改良版で、学習率の初期値が設定不要となっている。

$$h \leftarrow \rho h + (1 - \rho) \frac{\partial E_n}{\partial W} \circ \frac{\partial E_n}{\partial W} \quad (46)$$

$$\Delta W \leftarrow - \frac{\sqrt{s} + \epsilon}{\sqrt{h} + \epsilon} \frac{\partial E_n}{\partial W} \quad (47)$$

$$s \leftarrow \rho s + (1 - \rho) \Delta W \circ \Delta W \quad (48)$$

$$W \leftarrow W + \Delta W \quad (49)$$

h と s の初期値は 0, 事前に設定する必要があるパラメータは ρ と ϵ の 2 つである。（AdaDelta の論文中では $\rho = 0.95$, $\epsilon = 10^{-6}$ を推奨）

Adam Adam は AdaDelta の改良版で 2 つの慣性項を用いる。

$$t \leftarrow t + 1 \quad (50)$$

$$m \leftarrow \beta_1 m + (1 - \beta_1) \frac{\partial E_n}{\partial W} \quad (51)$$

$$v \leftarrow \beta_2 v + (1 - \beta_2) \frac{\partial E_n}{\partial W} \circ \frac{\partial E_n}{\partial W} \quad (52)$$

$$\hat{m} \leftarrow m / (1 - \beta_1^t) \quad (53)$$

$$\hat{v} \leftarrow v / (1 - \beta_2^t) \quad (54)$$

$$W \leftarrow W - \alpha \hat{m} / (\sqrt{\hat{v}} + \epsilon) \quad (55)$$

t, m, v の初期値は 0, 事前に設定する必要のあるパラメータは $\alpha, \beta_1, \beta_2, \epsilon$ である. (Adam の論文中での推奨値は $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$)

5.6 [発展課題] カラー画像への拡張

MNIST の画像は 1 チャンネル (モノクロ) の画像であった, これをカラー画像に拡張する. カラー画像は R (赤), G (緑), B (青) の 3 チャンネルの画像で構成され, 3 次元配列 $3 \times d_x \times d_y$ の形式をとる.

5.6.1 CIFAR-10 一般画像物体認識用画像データ

CIFAR-10 は物体認識用のデータセットで, 10 クラス 60,000 枚 (1 クラスあたり 6000 枚) の 32×32 カラー画像からなる. そのうち 50000 枚を教師データ, 残り 10000 枚をテストデータとして用いる. 10 個のクラスは { airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck } からなり, 順番に 0~9 の番号で表されている.

CIFAR-10 の Web サイト <https://www.cs.toronto.edu/~kriz/cifar.html> より, CIFAR-10 python version (<https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz>) をダウンロードすること.

ダウンロードしたファイル (cifar-10-python.tar.gz) を以下のコマンドで展開し, 学習・テストに用いる.

CIFAR-10 の準備

```
cd ~/ダウンロード
tar xzf cifar-10-python.tar.gz
mv cifar-10-batches-py ~/
```

cifar-10-batches-py の下の data_batch_1~data_batch_5 にそれぞれ 10000 枚の訓練用画像が, test_batch に 10000 枚のテスト用画像がある.

これらの画像は以下のサンプルコードを用いて読み込むことができる.

CIFAR-10 の訓練用画像 10000 枚を読み込んで、1 枚表示する。サンプルコード

```
import numpy as np
import cPickle

def unpickle(file):
    with open(file, 'rb') as fo:
        dict = cPickle.load(fo)
    X = np.array(dict['data'])
    X = X.reshape((X.shape[0],3,32,32))
    Y = np.array(dict['labels'])
    return X,Y

X,Y = unpickle("./data_batch_1")

import matplotlib.pyplot as plt
idx = 1000
plt.imshow(X[idx].transpose(((1,2,0))) # X[idx] が (3*32*32) になっている
のを (32*32*3) に変更する.
plt.show() # トラックの画像が表示されるはず
print Y[idx] # 9 番 (truck) が表示されるはず
```

5.7 [発展課題] 畳み込み層

ここまではニューラルネットワークへの入力をベクトル x としてきたが、本来入力画像 (2 次元配列) であり、画像が持つ形状をうまく扱うことができていない。畳み込み層は画像の形状をうまく扱うための層である。

5.7.1 入力層

これまでの入力層は d 次元ベクトルを入力として受け取りそれをそのまま出力していたが、これを $d_x \times d_y$ の 2 次元配列を入力として受け取り、それをそのまま出力するように変更する。MNIST の場合 $d_x = d_y = 28$ である。

5.7.2 画像の畳み込み

2 次元配列 $I(i, j)$ ($i = 1, \dots, d_x$) ($j = 1, \dots, d_y$) に対し、**フィルタ**と呼ばれる 2 次元配列 $f(i, j)$ ($i, j = 1, \dots, R$) を畳み込むことで 2 次元配列 $I'(i, j)$ を得る。

$$I'(i, j) = \sum_{s=1}^R \sum_{t=1}^R I(i+s-\lceil R/2 \rceil, j+t-\lceil R/2 \rceil) f(s, t) \quad (56)$$

ここで、 I の範囲外 ($i < 1$ または $i > d_x$, $j < 1$ または $j > d_y$) については $I(i, j) = 0$ とする (パディング). $I'(i, j)$ のサイズを元の画像サイズ $I(x, y)$ と同一にしたい場合は, 画像の上下左右にそれぞれ幅 $\lfloor R/2 \rfloor$ だけの 0 で埋められた画素を追加すればよい.

これにさらにバイアス b を加えた,

$$I'(i, j) = \sum_{s=1}^R \sum_{t=1}^R I(i + s - \lfloor R/2 \rfloor, j + t - \lfloor R/2 \rfloor) + b \quad (57)$$

を畳み込み層における処理とする. 学習すべきパラメータはフィルタ $f(i, j)$ とバイアス b である. フィルタのサイズ R をフィルタサイズと呼ぶ.

また, 畳み込みを行う際, 全ての $I'(i, j)$ を計算するのではなく, s 個おきに $I'(i, j)$ を計算することもある. この s を**ストライド**と呼び, 例えば $s = 2$ の時は元の入力の半分のサイズ ($\frac{d_x}{s} \times \frac{d_y}{s}$) の 2 次元配列が出力される.

5.7.3 多チャンネル画像の畳み込み

カラー画像のように複数のチャンネルからなる画像は, チャンネル数を ch とすると 3 次元配列 $I(k, i, j)$ ($k = 1, \dots, ch$) で表現できる. これに対するフィルタも 3 次元配列 $f(k, i, j)$ ($k = 1, \dots, ch$) となり, 畳み込み処理は,

$$I'(i, j) = \sum_{c=1}^{c=ch} \sum_{s=1}^R \sum_{t=1}^R I(i + s - \lfloor R/2 \rfloor, j + t - \lfloor R/2 \rfloor) + b \quad (58)$$

で計算される. 多チャンネルの場合も出力は 1 チャンネルの 2 次元配列となることに注意すること.

通常畳み込み層では 1 つのフィルタだけを学習するのではなく, 複数の (同じフィルタサイズの) フィルタを学習する. 1 つのフィルタに対して畳み込みにより 1 チャンネルの 2 次元配列が出力されるので, 複数のフィルタを用いる場合, 畳み込み層の出力は 3 次元配列となる. 例えば ch チャンネルの画像 ($d_x \times d_y$) を入力とし, フィルタサイズ 3×3 のフィルタ K 枚をパディング $p = \lfloor 3/2 \rfloor = 1$, ストライド $s = 2$ で畳み込む場合, 学習すべきパラメータは $K * (3 * 3 * ch + 1)$ 個, 畳み込み層の出力は $K \times \frac{d_x}{2} \times \frac{d_y}{2}$ の 3 次元配列となる.

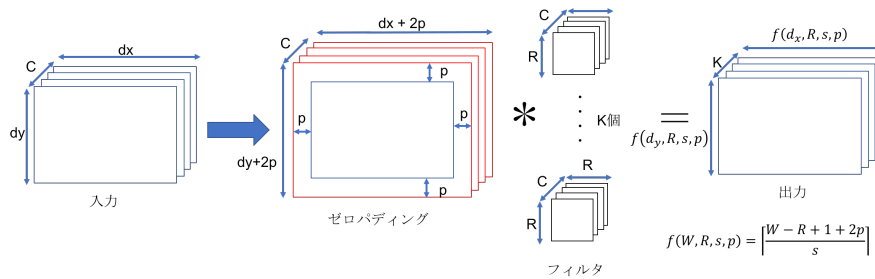


図 3: 畳み込み層 (バイアス項 b 無しの例)

5.7.4 畳み込み層の出力の変換

上で述べたように、畳み込み層の出力は3次元配列となる。この出力結果を全結合層の入力として用いる場合は、単に3次元配列を1次元配列に並び替えればよい。これは後述するプーリング層の出力を全結合層の入力として用いる場合も同様である。

5.7.5 畳み込み層の計算（順伝播）と実装

畳み込み層の計算は、式 58 の計算に 3 重ループ、式 58 の計算を出力画像⁵の各チャンネル、画素毎に行うのでさらに外側に 3 重ループ、さらに畳み込み層の計算をミニバッチ単位で行うともうさらに外側に 1 重ループとなり、素直に実装すると 7 重ループの計算になる。python で for 文を用いた繰り返し処理を多用すると処理が遅くなるため、畳み込み層の計算を以下の方法 [5] により行列演算に変換する。

まず（バイアス項 b を除く）フィルタを、図 4 のようにひとつの行列 W で表す。フィルタサイズ $R \times R$ 、チャンネル数 ch 、フィルタ枚数 K のフィルタ群の重みを、 K 行 $R \times R \times ch$ 列の行列 W で表す。各行が 1 枚のフィルタに対応し、フィルタを構成する $R \times R \times ch$ 個の要素をベクトルで表現する。

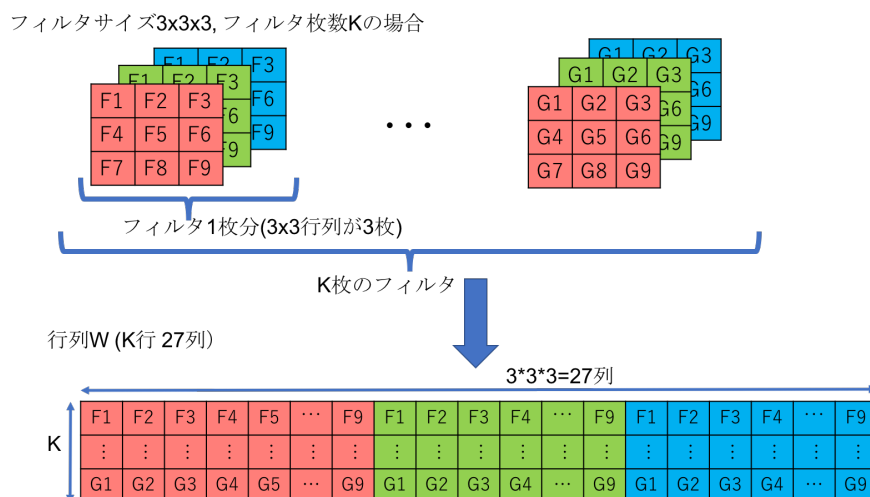


図 4: フィルタから行列 W への変換

次に、多チャンネルの入力画像に対しパディング処理を行う。パディング済みの多チャンネル画像を、図 5 のようにひとつの行列 X で表す。元の画像サイズを $d_x \times d_y$ としたとき、 X は $R \times R \times ch$ 行、 $d_x \times d_y$ 列の大きな行列となる。 X の各列が、出力画像の 1 画素での畳み込み演算に必要な入力画像の画素に対応する。例えば、図 5 の 1 列目は、出力画像の一番左上の画素の出力を計算するために必要な画素に対応し、行列 W の 1 行目と X の 1 列目との内積を計算することで、出力画像の一番左上の画素の出力が得られる。

⁵ 正確に書くと出力は 2 次元配列なのですが、画像と書いた方が分かりやすいので以降画像と表記しています。

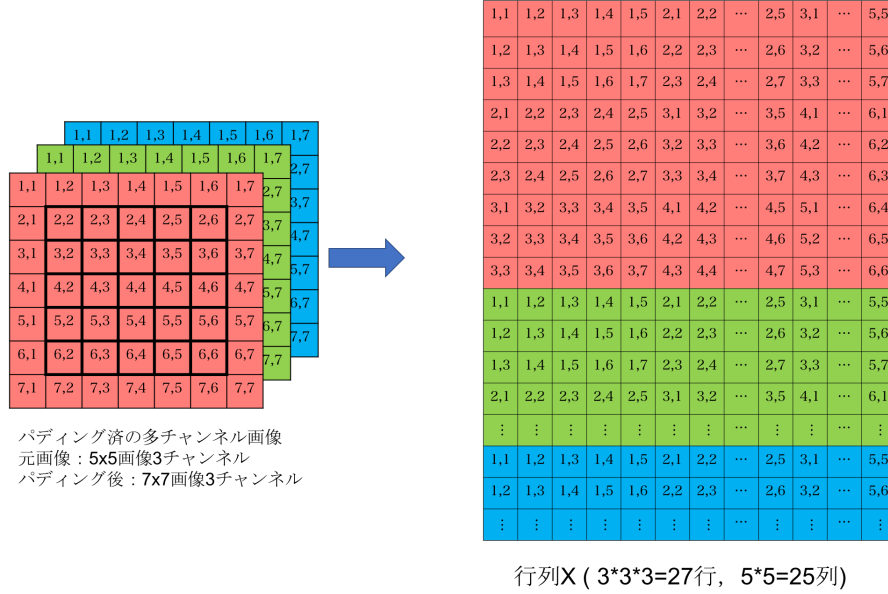


図 5: 画像から行列 X への変換

以上の処理によって得られた行列 W と X との行列積を計算することにより、 K 行 $d_x \times d_y$ 列の行列 Y が得られる。行列 Y の各行が1枚のフィルタによる畳み込み結果に対応する。ミニバッチ処理で B 個の入力を一度に処理する場合、各入力画像に対して得られた X を横に並べて、 $R \times R \times ch$ 行、 $d_x \times d_y \times B$ 列の巨大な行列を作成し、これを X として用いることで一度の行列積により畳み込み演算を実行することができる。

K 枚のフィルタの各バイアス $b_i (i = 1, \dots, K)$ については、 W と X との行列積を計算した後に、得られた行列の i 行目の全てに要素に b_i を加算していくことで計算できる。

$$Y = WX + B \quad (59)$$

ここで、 B は K 行 $d_x \times d_y \times B$ 列の行列で、 B の i 行目の要素が全て b_i となるような行列である。

なお、畳み込み層についても出力 Y に対して活性化関数を適用する。具体的には Y の各要素に対してシグモイド関数 σ や ReLU 関数を適用し、その結果を最終的な出力とする。

5.7.6 畳み込み層の計算（逆伝播）

畳み込み層の逆伝播は、上で述べた行列 W 、 X 、 B 、 Y を利用すれば、全結合層の逆伝播と同様にして計算することができる。

$$\frac{\partial E_n}{\partial X} = W^T \frac{\partial E_n}{\partial Y} \quad (60)$$

$$\frac{\partial E_n}{\partial W} = \frac{\partial E_n}{\partial Y} X^T \quad (61)$$

$$\frac{\partial E_n}{\partial b} = \text{rowSum} \left(\frac{\partial E_n}{\partial Y} \right) \quad \text{※行列} \frac{\partial E_n}{\partial Y} \text{の行ごとの和} \quad (62)$$

5.8 [発展課題] プーリング層

プーリング層は多チャンネルの畳み込み層の出力を入力として受け取り、多チャンネルの画像を出力する層である。

具体的には ch チャンネル $d_x \times d_y$ の多チャンネル画像（3次元配列）を入力として受け取り、 ch チャンネル $d_x/d \times d_y/d$ の多チャンネル画像を出力する。ここで d はプーリング層のサイズである。

ここでは、プーリング層のなかでも最も良く用いられている max プーリングについて説明する。入力を3次元配列 $X[c, i, j]$ ($c = 1, \dots, ch, i = 1, \dots, d_x, j = 1, \dots, d_y$) とし、出力を3次元配列 $Y[c, i, j]$ ($c = 1, \dots, ch, i = 1, \dots, \lfloor d_x/d \rfloor, j = 1, \dots, \lfloor d_y/d \rfloor$) とする。出力 Y は、

$$Y[c, i, j] = \max X[c, id + k, jd + l] \quad \text{where } 0 \leq k < d, 0 \leq l < d \quad (63)$$

によって得られる。

プーリング層において学習されるパラメータは存在しない。逆伝播は、

$$\frac{\partial E_n}{\partial X[c, id + k, jd + l]} = \begin{cases} \frac{\partial E_n}{\partial Y[c, i, j]} & \text{if } Y[c, i, j] = X[c, id + k, jd + l] \\ 0 & \text{otherwise} \end{cases} \quad (64)$$

によって得ることができる。

参考文献

- [1] “THE MNIST DATABASE of handwritten digits”, <http://yann.lecun.com/exdb/mnist/>
- [2] “CIFAR-10 dataset”, <https://www.cs.toronto.edu/~kriz/cifar.html>
- [3] “Python チュートリアル”, <https://docs.python.jp/2.7/tutorial/index.html>
- [4] “NumPy Quickstart tutorial”, <https://docs.scipy.org/doc/numpy-dev/user/quickstart.html>
- [5] “cuDNN: Efficient Primitives for Deep Learning”, arXiv:1410.0759, 2014.