# 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：下西慶 (TA: 住江祐哉，有本拓矢)

2021 年度版 (Ver1.0)

## 目 次

**1 はじめに** **2**

**2 演習にあたって** **2**

**3 Python** **4**
3.1 Windows 環境での Python 利用 . . . . . 4
3.1.1 VSCode で python を使えるようにする . . . . . 4
3.2 Python チュートリアル . . . . . 7
3.3 NumPy チュートリアル . . . . . 7

**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** **7**
4.1 MNIST 手書き数字データ . . . . . 8
4.1.1 python パッケージのインストール . . . . . 8
4.1.2 Python でのデータの読み込み . . . . . 8
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 . . . . . 10
4.2.1 3 層ニューラルネットワーク . . . . . 10
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） . . . . . 12
4.3 ニューラルネットワークの学習 . . . . . 12
4.3.1 損失関数 . . . . . 13
4.3.2 ミニバッチ . . . . . 13
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 . . . . . 13
4.3.4 誤差逆伝播法による損失関数の勾配の計算 . . . . . 14
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 . . 16
4.3.6 [課題 4] 手書き数字画像識別器の構築 . . . . . 16

**5 発展課題 A** **17**
5.1 [発展課題 A1] 活性化関数 . . . . . 17
5.2 [発展課題 A2] Dropout . . . . . 17
5.3 [発展課題 A3] Batch Normalization . . . . . 18

5.4 [発展課題 A4] 最適化手法の改良 . . . . . 19
5.5 [発展課題 A5] カラー画像への拡張 . . . . . 20
5.5.1 CIFAR-10 一般画像物体認識用画像データ . . . . . 20
5.6 [発展課題 A6] 畳み込み層 . . . . . 21
5.6.1 入力層 . . . . . 21
5.6.2 画像の畳み込み . . . . . 22
5.6.3 多チャンネル画像の畳み込み . . . . . 22
5.6.4 畳み込み層の出力の変換 . . . . . 22
5.6.5 畳み込み層の計算（順伝播）と実装 . . . . . 23
5.6.6 畳み込み層の計算（逆伝播） . . . . . 23
5.7 [発展課題 A7] プーリング層 . . . . . 25

**6 [発展課題 B] 第 2 部 Keras の利用 25**
6.1 はじめに . . . . . 25
6.2 GPGPU サーバの利用 . . . . . 25
6.3 Keras 入門 . . . . . 26
6.4 [発展課題 B1] MNIST 手書き数字認識 . . . . . 30
6.5 [発展課題 B2] 顔画像認識 . . . . . 31
6.6 [発展課題 B3] 画像生成（他の発展課題に飽きた人向け） . . . . . 32

## 改訂履歴

**ver 1.0** 初稿

## 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標とする．さらに余力がある場合は，画像生成など画像処理に関する他の課題にも取り組む.

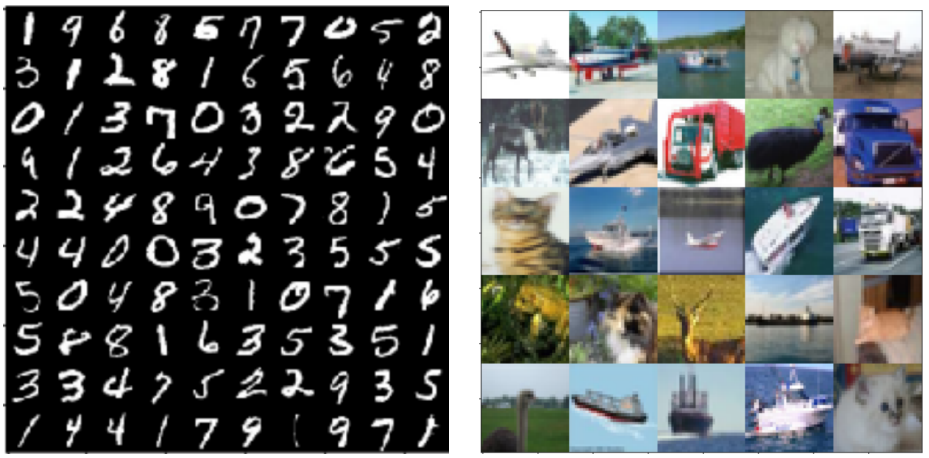
## 2 演習にあたって

本演習は，必修課題を含むニューラルネットを一から構築する課題と，深層学習フレームワークを用いた課題の 2 部構成である.

本演習の第一部では，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．

第二部では，GPGPU サーバ上で深層学習フレームワークを用いた実装を行う．なお，第二部については第一部の必修課題をクリアしたものを対象とし，すべて発展課題とする．



(a) MNIST　　(b) CIFAR-10

図 1: MNIST と CIFAR-10 データセット

ニューラルネットワークについては，caffe や tensorflow，chainer，keras 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，本演習の前半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（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．

第一部の演習は必修課題 4 つと，発展課題からなる．第二部は発展課題のみである．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 Windows 環境での Python 利用

本演習では Python 環境として anaconda を利用する．anaconda の web サイト (`https://www.anaconda.com/download/`) より，Individual Edition (64 − Bit Graphical Installer, Anaconda3 − 2021.11-Windows-x86_64.exe) をダウンロードして，実行する．途中でライセンスに関する質問や環境変数に関する質問が出るが，(とくにこだわりがなければ) すべてデフォルトのままで良い．

### 3.1.1 VSCode で python を使えるようにする

https://code.visualstudio.com/download より Windows 用のインストーラ (User Installer, VSCodeUserSetup-x64-1.62.3.exe) をダウンロードして，実行する．途中でライセンスに関する質問などが出るが，(とくにこだわりがなければ) すべてデフォルトのままで良い．

次に，VSCode を起動して，「Extensions」(左側のメニューの上から 5 つ目あたりにあるボタン) から”Python”で検索して，Python の拡張機能をインストールする．インストールが終了したら，一旦 VSCode を終了する．

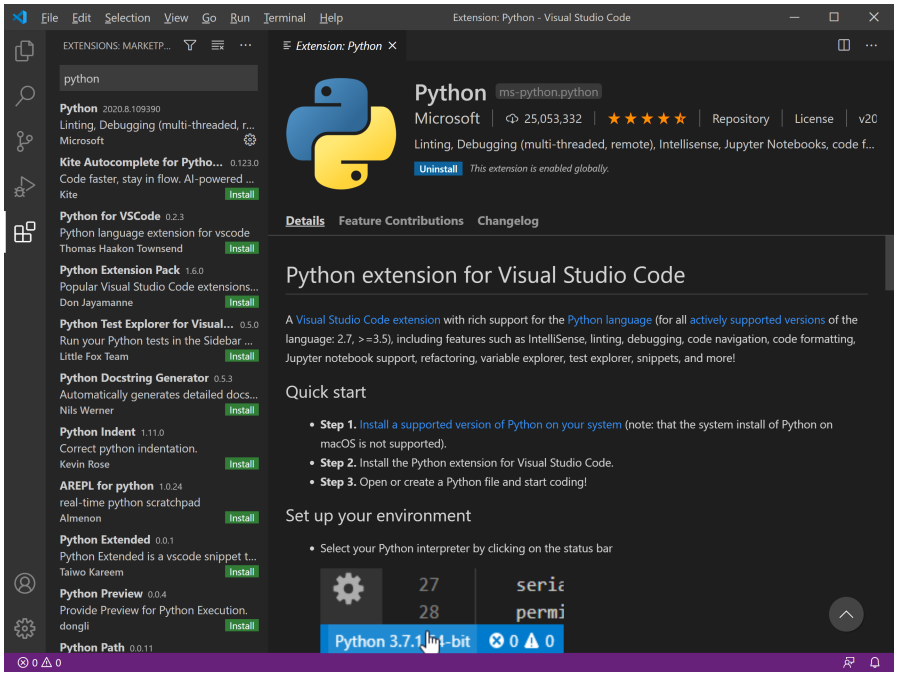



図 2: VSCode に Python の拡張機能をインストール

Windows のスタートボタンから Anaconda Navigator を起動し，そこから VSCode を”Launch”する．次回以降も Anaconda Navigator 経由で VSCode を起動すること（そうしないとエラーが出る）．

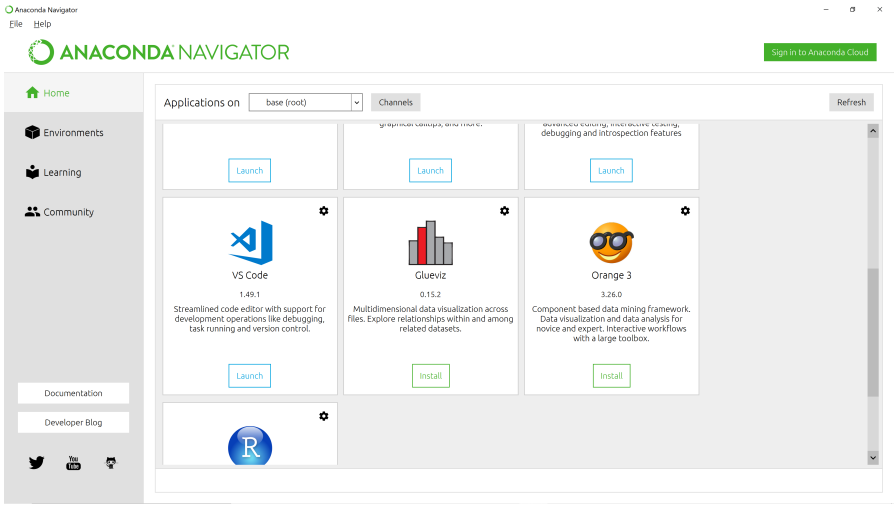



図 3: Anaconda Navigator から VSCode を起動

VSCode の “File” → “Open Folder” で作業用のフォルダを指定して，”New File”ボタンで python のスクリプトファイル（ここでは，test.py とします）を作成する．作成したら，Hello world 代わりに以下のコードを入力する．

test.py

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-10,10)
y = 1/(1+np.exp(-x))

plt.plot(x,y)
plt.show()
```

VSCode の ”Run” → ”Start Debugging”から”Python File”を選んでデバッグ実行する．シグモイド関数のグラフが表示されれば正常である．

下図のような ImportError が出たら，一旦 VSCode を終了して Anaconda Navigator から VSCode を起動してください．

**Jupyter Lab（お好みで）** Jupyter Lab は Web インタフェースのインタラクティブ 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．Anaconda Navigator から Jupyter Lab を起動することができる．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．

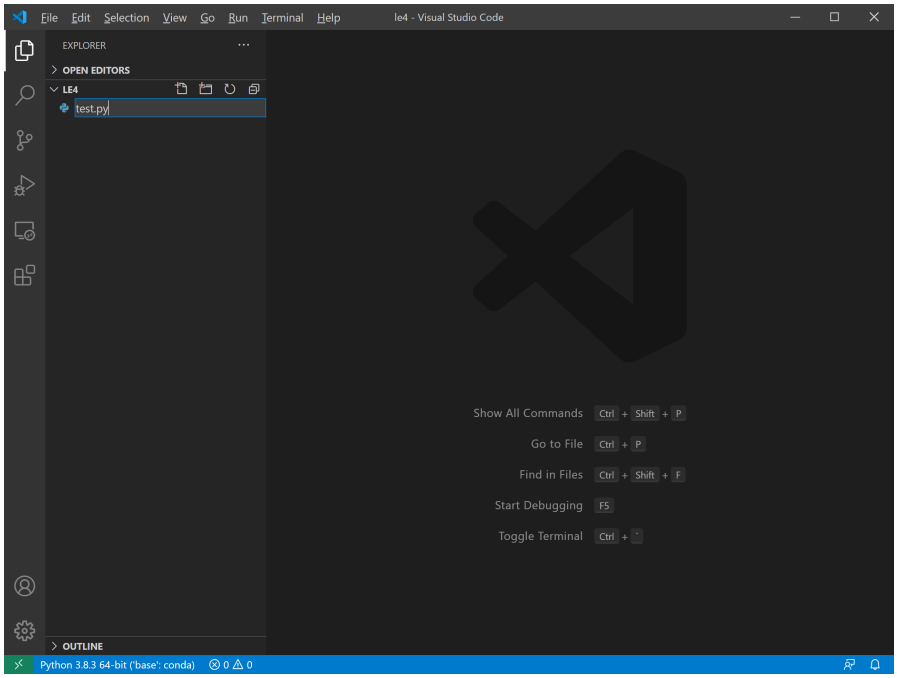



図 4: test.py

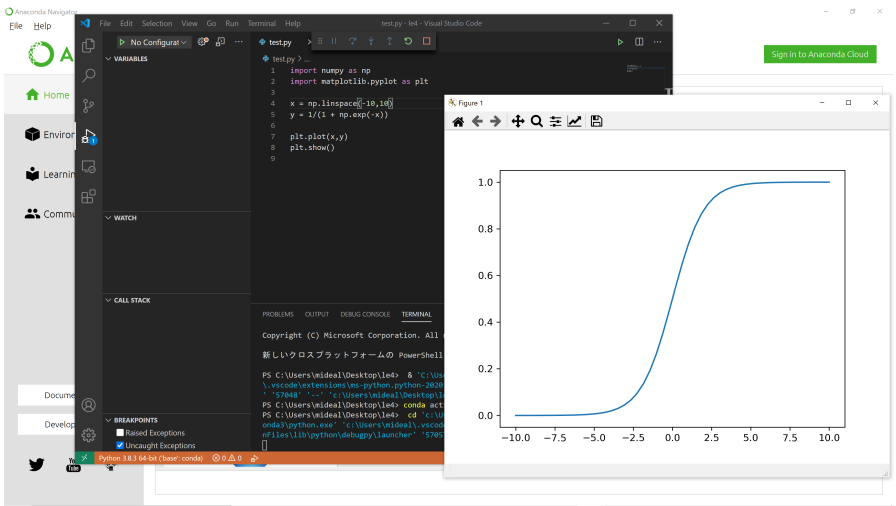



図 5: サンプルコードの実行結果

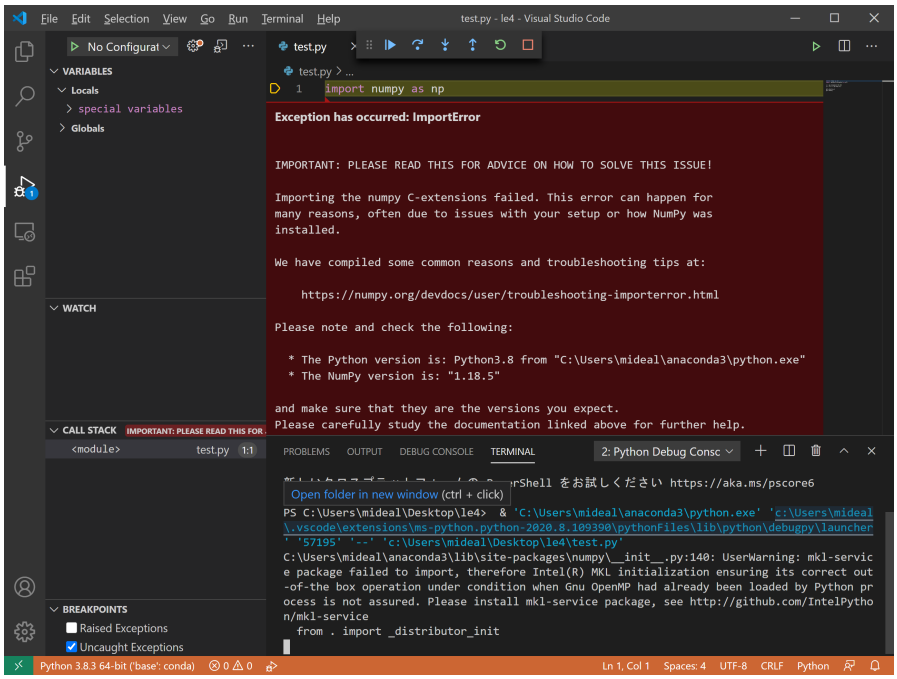



図 6: この画面が出たら VSCode を終了して Anaconda Navigator から起動しなおす

### 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，全学共通科目「プログラミング演習」のテキスト `https://repository.kulib.kyoto-u.ac.jp/dspace/handle/2433/265459` などを参考にしつつ自習して頂きたい．

### 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/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 $28 \times 28$ 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．

### 4.1.1 python パッケージのインストール

conda で mnist パッケージをインストールする．

1. Anaconda Navigator の Environments の Channels タブより，”Add...”で，conda-forge を追加して，”Update channels”を実行
2. 同じ画面で検索条件を”All”にして”mnist”で検索し，チェックボックスにチェックを入れて Apply．

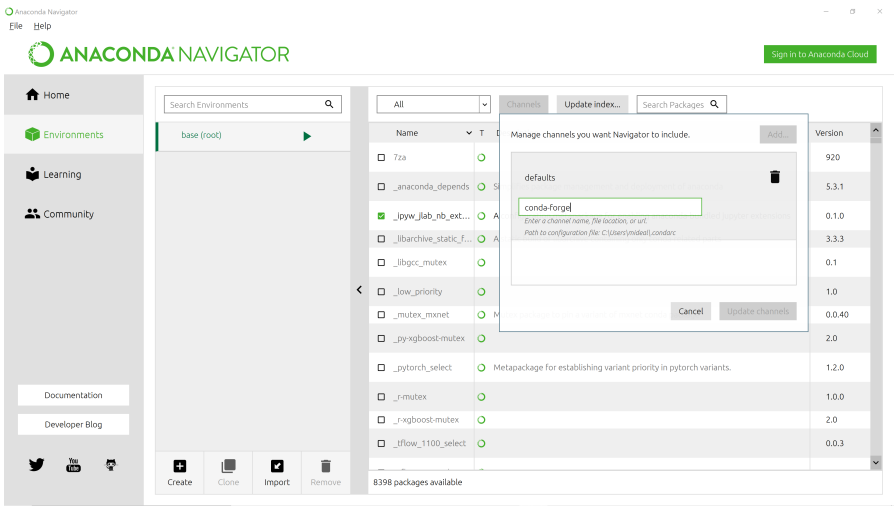



図 7: conda-forge をチャンネルに追加

### 4.1.2 Python でのデータの読み込み

sample1.py

```
import numpy as np
import mnist
X = mnist.download_and_parse_mnist_file("train-images-idx3-ubyte.gz")
Y = mnist.download_and_parse_mnist_file("train-labels-idx1-ubyte.gz")
```

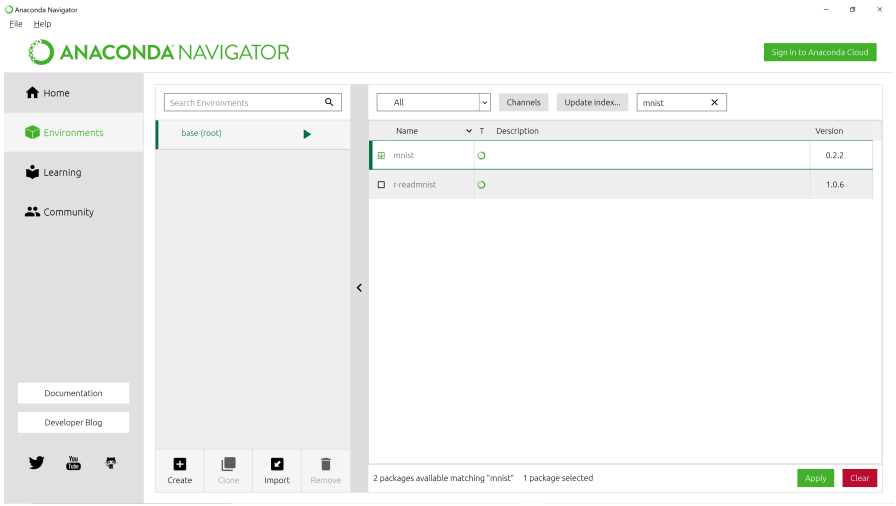


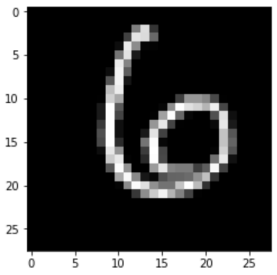

図 8: mnist パッケージのインストール

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，train-images-idx3-ubyte.gz と train-labels-idx1-ubyte.gz の代わりに t10k-images-idx3-ubyte.gz と t10k-labels-idx1-ubyte.gz を指定することにより，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 $\boldsymbol{x}$ で表す．MNIST 画像は $28 \times 28$ の 2 次元配列であるがこれを左上から $x$ 軸，$y$ 軸の順に一列に並びかえることにより $d = 784$ 次元のベクトルとする．画像 $\boldsymbol{x}$ を入力として，$\boldsymbol{x}$ に対応する数字 $y \in \{0, \ldots, 9\}$ を出力する関数 $y = f(\boldsymbol{x})$ を構築する．関数 $f$ の構築にあたり，$\boldsymbol{x}$ と $y$ とのペアの集合 $\{(\boldsymbol{x}_i, y_i)\}$ $(i = 1, \ldots, N)$ 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 9 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 $\boldsymbol{x}$ の各要素 $x_i$ $(i = 1, \ldots, d)$ 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 $j$ $(j = 1, \ldots, M)$ 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．

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

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

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

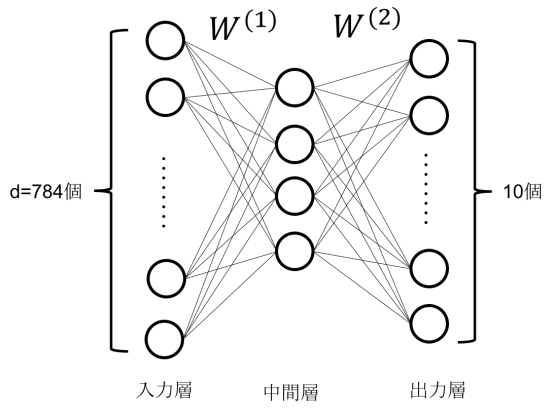



図 9: 3 層ニューラルネットワーク

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

$$y_j^{(1)} = a(\boldsymbol{w_j^{(1)}} \cdot \boldsymbol{x} + b_j^{(1)}) \tag{3}$$

さらに，線形和の重み $\boldsymbol{w_j^{(1)}}$ 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 $\boldsymbol{b^{(1)}} = (b_1^{(1)}, \ldots, b_d^{(1)})^T$ を用いて中間層への $M$ 個の入力を $W^{(1)}\boldsymbol{x} + \boldsymbol{b^{(1)}}$ 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．

$$a_k = \boldsymbol{w_k^{(2)}} \cdot \boldsymbol{y^{(1)}} + b_k^{(2)} \tag{4}$$

ここで，$\boldsymbol{y^{(1)}} = (y_1^{(1)}, \ldots, y_M^{(1)})^T$ である．また，$C$ 個の入力 $\boldsymbol{a} = (a_1, \ldots, a_C)^T$ を，線形和の重み $\boldsymbol{w_k^{(2)}}$ 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 $\boldsymbol{b^{(2)}} = (b_1^{(2)}, \ldots, b_d^{(2)})^T$ を用いて，$\boldsymbol{a} = W^{(2)}\boldsymbol{y^{(1)}} + \boldsymbol{b^{(2)}}$ 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 $a_i$ $(i = 1, \ldots, C)$ とし，ソフトマックス関数を用いて出力層の出力 $y_i^{(2)}$ $(i = 1, \ldots, C)$ を次式で得る[1]．

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

$$\alpha = \max a_i \tag{6}$$

出力層の出力 $y_i^{(2)}$ $(i = 1, \ldots, C)$ は入力 $\boldsymbol{x}$ 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．

[1] ソフトマックス関数の定義では $\alpha$ 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 $\alpha$ 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（$28 \times 28$），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 $W^{(1)}, W^{(2)}, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}$ 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[2]．

**（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 $W^{(1)}, W^{(2)}, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}$ 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

[2]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 $\boldsymbol{x}$ 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 $\boldsymbol{y}^{(2)}$ 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．

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

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．

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

$B$ 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 の学習画像 60000 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[3]．

[3] numpy.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 $W^{(1)}, W^{(2)}, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}$ 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

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

のように更新することができる．($\eta$ 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，

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x} \tag{10}$$

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⇒全結合層 $W^{(1)}, \boldsymbol{b^{(1)}}$ ⇒シグモイド関数⇒全結合層 $W^{(2)}, \boldsymbol{b^{(2)}}$ ⇒ Softmax 関数⇒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 $\boldsymbol{a} = (a_1, \ldots, a_C)^T$ を受け取り，$C$ 個の出力 $y_k^{(2)}$ $(k = 1, \ldots, C)$ 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）．

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

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，

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

となる．

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

$$\frac{\partial E_n}{\partial \boldsymbol{x}} = W^T \frac{\partial E_n}{\partial \boldsymbol{y}} \tag{13}$$

$$\frac{\partial E_n}{\partial W} = \frac{\partial E_n}{\partial \boldsymbol{y}} \boldsymbol{x}^T \tag{14}$$

$$\frac{\partial E_n}{\partial \boldsymbol{b}} = \frac{\partial E_n}{\partial \boldsymbol{y}} \tag{15}$$

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 $\boldsymbol{x_1}, \ldots, \boldsymbol{x_B}$ とそれに対する出力 $\boldsymbol{y_1}, \ldots, \boldsymbol{y_B}$，および出力に対する損失関数の勾配 $\partial E_N/\partial \boldsymbol{y_i}$ $(i = 1, \ldots, B)$ が

与えられた場合，$W$ や $\boldsymbol{b}$ に対する損失関数の勾配は，$B$ 個のベクトル $\boldsymbol{x_1}, \ldots, \boldsymbol{x_B}$ を各列に持つ行列を $X$，$\partial E_N/\partial \boldsymbol{y_i}$ を各列として持つ行列を $\partial E_N/\partial Y$ として，

$$\frac{\partial E_n}{\partial X} = W^T \frac{\partial E_n}{\partial Y} \tag{16}$$

$$\frac{\partial E_n}{\partial W} = \frac{\partial E_n}{\partial Y} X^T \tag{17}$$

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

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，

$$a(t)' = (1 - a(t))a(t) \tag{19}$$

で与えられる．

シグモイド関数への入力を $x$，出力を $y$ とし，$\frac{\partial E_n}{\partial y}$ が与えられたとき，$\frac{\partial E_n}{\partial x}$ は，

$$\frac{\partial E_n}{\partial x} = \frac{\partial y}{\partial x}\frac{\partial E_n}{\partial y} = \frac{\partial E_n}{\partial y}(1-y)y \tag{20}$$

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 16，17，18 を用いて $\frac{\partial E_n}{\partial X}$，$\frac{\partial E_n}{\partial W}$，$\frac{\partial E_n}{\partial \boldsymbol{b}}$ を計算．式 16，17，18 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，$\frac{\partial E_n}{\partial W}$，$\frac{\partial E_n}{\partial \boldsymbol{b}}$ が，それぞれ $\frac{\partial E_n}{\partial W^{(2)}}$，$\frac{\partial E_n}{\partial \boldsymbol{b^{(2)}}}$ となる．
6. 式 20 を $\frac{\partial E_n}{\partial X}$ の各要素に適用.
7. 式 16，17，18 を用いて $\frac{\partial E_n}{\partial X}$，$\frac{\partial E_n}{\partial W}$，$\frac{\partial E_n}{\partial \boldsymbol{b}}$ を計算．式 16，17，18 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，$\frac{\partial E_n}{\partial W}$，$\frac{\partial E_n}{\partial \boldsymbol{b}}$ が，それぞれ $\frac{\partial E_n}{\partial W^{(1)}}$，$\frac{\partial E_n}{\partial \boldsymbol{b^{(1)}}}$ となる．

8. 学習率 $\eta$ を用いて，パラメータを更新.

$$W^{(1)} \leftarrow W^{(1)} - \eta \frac{\partial E_n}{\partial W^{(1)}} \tag{21}$$

$$W^{(2)} \leftarrow W^{(2)} - \eta \frac{\partial E_n}{\partial W^{(2)}} \tag{22}$$

$$\boldsymbol{b^{(1)}} \leftarrow \boldsymbol{b^{(1)}} - \eta \frac{\partial E_n}{\partial \boldsymbol{b^{(1)}}} \tag{23}$$

$$\boldsymbol{b^{(2)}} \leftarrow \boldsymbol{b^{(2)}} - \eta \frac{\partial E_n}{\partial \boldsymbol{b^{(2)}}} \tag{24}$$

#### 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 $W^{(1)}, W^{(2)}, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}$ 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 $\eta$ 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 $W^{(1)}, W^{(2)}, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}$ 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 A

[発展課題 A] 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．必修課題すべてを完了しているもののみが発展課題に取り組むことができる．なお，発展課題 A をスキップして発展課題 B に取り組んでも良い．ただし，その場合でも発展課題 A の処理内容を理解しておくことをお勧めする．

## 5.1　[発展課題 A1] 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．

$$a(t) = \begin{cases} t & (t > 0) \\ 0 & (t \leq 0) \end{cases} \tag{25}$$

ReLU の微分は，

$$a(t)' = \begin{cases} 1 & (t > 0) \\ 0 & (t \leq 0) \end{cases} \tag{26}$$

で与えられる．

## 5.2　[発展課題 A2] 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，

$$a(t) = \begin{cases} t & (\text{ノードが無視されない場合}) \\ 0 & (\text{ノードが無視された場合}) \end{cases} \tag{27}$$

となり，Dropout の微分は，

$$a(t)' = \begin{cases} 1 & (\text{ノードが無視されない場合}) \\ 0 & (\text{ノードが無視された場合}\quad) \end{cases} \tag{28}$$

で与えられる．

### 5.3 [発展課題 A3] 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 $x_i$ $(i = 1, \ldots, B)$ とした場合，以下の式により $x_i$ を $y_i$ に変換する．

$$\mu_B \leftarrow \frac{1}{B}\sum_{i=1}^{B} x_i \quad \text{※ミニバッチの平均} \tag{29}$$

$$\sigma_B^2 \leftarrow \frac{1}{B}\sum_{i=1e}^{B} (x_i - \mu_B)^2 \quad \text{※ミニバッチの分散} \tag{30}$$

$$\hat{x_i} \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \text{※ } x_i\text{の正規化} \tag{31}$$

$$y_i \leftarrow \gamma\hat{x_i} + \beta \tag{32}$$

ここで，$\gamma$ と $\beta$ の初期値はそれぞれ 1，0 であり，学習が進むについ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．

$$\frac{\partial E_n}{\partial \hat{x_i}} = \frac{\partial E_n}{\partial y_i} \cdot \gamma \tag{33}$$

$$\frac{\partial E_n}{\partial \sigma_B^2} = \sum_{i=1}^{B} \frac{\partial E_n}{\partial \hat{x_i}} \cdot (x_i - \mu_B) \cdot \frac{-1}{2}(\sigma_B^2 + \epsilon)^{-3/2} \tag{34}$$

$$\frac{\partial E_n}{\partial \mu_B} = \left( \sum_{i=1}^{B} \frac{\partial E_n}{\partial \hat{x_i}} \cdot \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \right) + \frac{\partial E_n}{\partial \sigma_B^2} \cdot \frac{\sum_{i=1}^{B} -2(x_i - \mu_B)}{B} \tag{35}$$

$$\frac{\partial E_n}{\partial x_i} = \frac{\partial E_n}{\partial \hat{x_i}} \cdot \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial E_n}{\partial \sigma_B^2} \cdot \frac{2(x_i - \mu_B)}{B} + \frac{\partial E_n}{\partial \mu_B} \cdot \frac{1}{B} \tag{36}$$

$$\frac{\partial E_n}{\partial \gamma} = \sum_{i=1}^{B} \frac{\partial E_n}{\partial y_i} \cdot \hat{x_i} \tag{37}$$

$$\frac{\partial E_n}{\partial \beta} = \sum_{i=1}^{B} \frac{\partial E_n}{\partial y_i} \tag{38}$$

$\mu_B$，$\sigma_B^2$ 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 $\mu_B$ と $\sigma_B^2$ 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，

$$y_i \leftarrow \frac{\gamma}{\sqrt{Var[x_i] + \epsilon}} \cdot x_i + \left( \beta - \frac{\gamma E[x_i]}{\sqrt{Var[x_i] + \epsilon}} \right) \tag{39}$$

を入力 $x_i$ に対する出力として用いる．

## 5.4 [発展課題 A4] 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 $\eta$ を用いて

$$W \leftarrow W - \eta \frac{\partial E_n}{\partial W} \tag{40}$$

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 $\alpha$ 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 $\eta$ と $\alpha$ である．($\eta = 0.01$，$\alpha = 0.9$ くらいがおすすめ)

$$\Delta W \leftarrow \alpha \Delta W - \eta \frac{\partial E_n}{\partial W} \tag{41}$$

$$W \leftarrow W + \Delta W \tag{42}$$

なお，$\Delta W$ 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．

$$h \leftarrow h + \frac{\partial E_n}{\partial W} \circ \frac{\partial E_n}{\partial W} \tag{43}$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial E_n}{\partial W} \tag{44}$$

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のあるパラメータは学習率の初期値 $\eta$ 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．

$$h \leftarrow \rho h + (1 - \rho) \frac{\partial E_n}{\partial W} \circ \frac{\partial E_n}{\partial W} \tag{45}$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{h} + \epsilon} \frac{\partial E_n}{\partial W} \tag{46}$$

$h$ の初期値は 0 である．事前に設定する必要のあるパラメータは学習率の初期値 $\eta$，$\rho$，$\epsilon$ である．($\eta = 0.001$，$\rho = 0.9$，$\epsilon = 10^{-8}$ 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．

$$h \leftarrow \rho h + (1-\rho)\frac{\partial E_n}{\partial W} \circ \frac{\partial E_n}{\partial W} \tag{47}$$

$$\Delta W \leftarrow -\frac{\sqrt{s+\epsilon}}{\sqrt{h+\epsilon}}\frac{\partial E_n}{\partial W} \tag{48}$$

$$s \leftarrow \rho s + (1-\rho)\Delta W \circ \Delta W \tag{49}$$

$$W \leftarrow W + \Delta W \tag{50}$$

$h$ と $s$ の初期値は 0，事前に設定する必要のあるパラメータは $\rho$ と $\epsilon$ 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．

$$t \leftarrow t + 1 \tag{51}$$

$$m \leftarrow \beta_1 m + (1-\beta_1)\frac{\partial E_n}{\partial W} \tag{52}$$

$$v \leftarrow \beta_2 v + (1-\beta_2)\frac{\partial E_n}{\partial W} \circ \frac{\partial E_n}{\partial W} \tag{53}$$

$$\hat{m} \leftarrow m/(1-\beta_1^t) \tag{54}$$

$$\hat{v} \leftarrow v/(1-\beta_2^t) \tag{55}$$

$$W \leftarrow W - \alpha\hat{m}/(\sqrt{\hat{v}}+\epsilon) \tag{56}$$

$t, m, v$ 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 $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$)

## 5.5　[発展課題 A5] 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．

### 5.5.1　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 $32 \times 32$ 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 zxf 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 pickle

def unpickle(file):
    with open(file, 'rb') as fo:
        dict = pickle.load(fo,encoding='bytes')
    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
```

## 5.6 [発展課題 A6] 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 $\boldsymbol{x}$ としてきたが，本来入力は画像（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．

### 5.6.1 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．

### 5.6.2 画像の畳み込み

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

$$I'(i,j)=\sum_{s=1}^{R}\sum_{t=1}^{R}I(i+s-\lceil R/2\rceil,j+t-\lceil R/2\rceil)f(s,t) \tag{57}$$

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，

$$I'(i,j)=\sum_{s=1}^{R}\sum_{t=1}^{R}I(i+s-\lceil R/2\rceil,j+t-\lceil R/2\rceil)+b \tag{58}$$

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．

### 5.6.3 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 $I(k,i,j)\ (k=1,\ldots,ch)$ で表現できる．これに対するフィルタも 3 次元配列 $f(k,i,j)\ (k=1,\ldots,ch)$ となり，畳み込み処理は，

$$I'(i,j)=\sum_{c=1}^{c=ch}\sum_{s=1}^{R}\sum_{t=1}^{R}I(i+s-\lceil R/2\rceil,j+t-\lceil R/2\rceil)+b \tag{59}$$

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 $3\times3$ 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．

### 5.6.4 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．

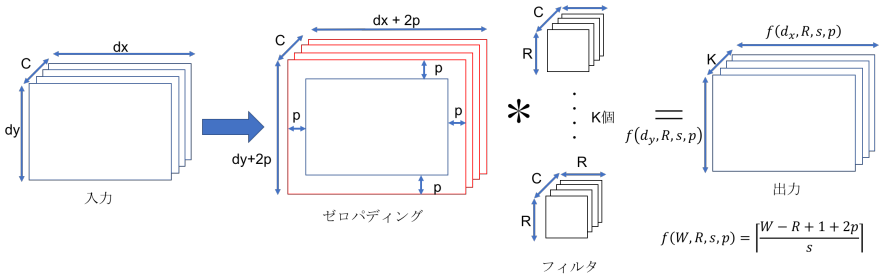



図 10: 畳み込み層（バイアス項 $\boldsymbol{b}$ 無しの例）

### 5.6.5 畳み込み層の計算（順伝播）と実装

畳み込み層の計算は，式 59 の計算に 3 重ループ，式 59 の計算を出力画像[4]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 11 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．

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 $b_i(i=1,\ldots,K)$ 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．

$$Y = WX + B \tag{60}$$

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 $a$ や ReLU 関数を適用し，その結果を最終的な出力とする．

### 5.6.6 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．

[4]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．

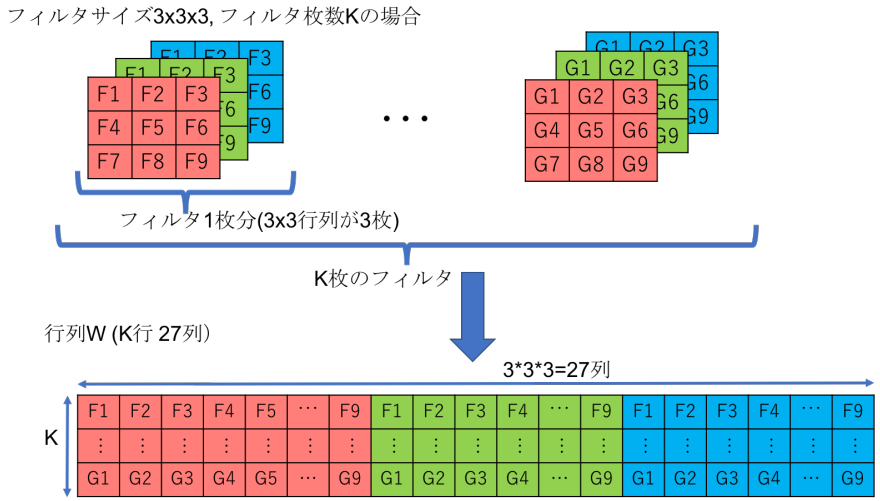



図 11: フィルタから行列 $W$ への変換

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 12 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 12 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．

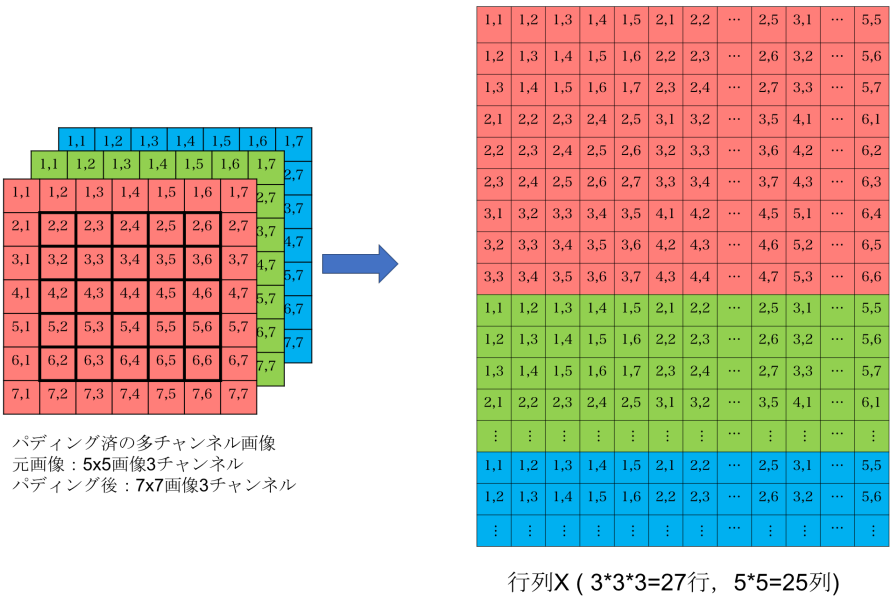



図 12: 画像から行列 $X$ への変換

$$\frac{\partial E_n}{\partial X} = W^T \frac{\partial E_n}{\partial Y} \tag{61}$$
$$\frac{\partial E_n}{\partial W} = \frac{\partial E_n}{\partial Y} X^T \tag{62}$$
$$\frac{\partial E_n}{\partial \boldsymbol{b}} = rowSum\left(\frac{\partial E_n}{\partial Y}\right) \quad \text{※行列}\frac{\partial E_n}{\partial Y}\text{の行ごとの和} \tag{63}$$

### 5.7 [発展課題 A7] 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 $X[c,i,j]$ $(c = 1,\ldots,ch,\ i = 1,\ldots,d_x,\ j = 1,\ldots,d_y)$ とし，出力を 3 次元配列 $Y[c,i,j]$ $(c = 1,\ldots,ch,\ i = 1,\ldots,\lfloor d_x/d \rfloor,\ j = 1,\ldots,\lfloor d_y/d \rfloor)$ とする．出力 $Y$ は，

$$Y[c,i,j] = \max X[c, id+k, jd+l] \ \ where\ 0 \le k < d,\ 0 \le l < d \tag{64}$$

によって得られる．

プーリング層におけて学習されるパラメータは存在しない．逆伝播は，

$$\frac{\partial E_n}{\partial X[c, id+k, jd+l]} = \begin{cases} \frac{\partial E_n}{\partial Y[c,i,j]} & if\ Y[c,i,j] = X[c, id+k, jd+l] \\ 0 & otherwise \end{cases} \tag{65}$$

によって得ることができる．

## 6 [発展課題 B] 第 2 部 Keras の利用

### 6.1 はじめに

以降では，GPGPU サーバを利用した画像認識器の作成について取りかかる．必修課題すべてが完了しているもののみが着手できる．なお，発展課題 A の内容についても（実装は不要であるが）処理の中身については理解していることを前提とする．

### 6.2 GPGPU サーバの利用

履修者のうち第 2 部に進む者については，GPGPU サーバログイン用のログイン ID と初期パスワードを配布する．希望者は担当教員（飯山）まで連絡すること．GPGPU サーバへのログイン方法，利用方法については ID 配布時に同時に配布する資料を参照のこと．

### 6.3 Keras 入門

Keras とは python で書かれたニューラルネットワーク用の API で，バックエンドとして TensorFlow や Theano を利用する．本演習第 2 部では Keras を用いて画像認識プログラムを作成する．TensorFlow や pyTorch など他のフレームワークを利用しても良いが，TA のサポートは得られないので各自の責任で用いること．(訂正：今年は pyTorch であれば TA のサポートを得られます)

Keras の詳細なリファレンスおよび使用例については，Keras 本家のサイト（`https://keras.io`）からアクセスできるが，ここでは，MNIST の手書き数字認識を例にして使用例を示す.

Keras で MNIST

```
import numpy as np
# Keras 関係の import
import keras
from keras import datasets, models, layers

# GPGPU リソースを全消費してしてしまう問題の回避
import tensorflow as tf
from keras.backend import set_session, tensorflow_backend
## 動的に必要なメモリだけを確保（allow_growth=True）
## デバイス「0」だけを利用（visible_device_list="0"） ※"0"の部分は，"0"
～"3"の中から空いているものを選んでください
config = tf.ConfigProto(gpu_options=tf.GPUOptions(allow_growth=True,
  visible_device_list="0"))
set_session(tf.Session(config=config))
```

**注意** Keras の仕様 (?) で，コードを実行するとサーバに搭載されている全ての GPGPU の全メモリを確保してしまう．今回の演習は，GPGPU サーバ（4 枚の GPGPU）を履修者で共有するため，上記のコードを参考に，必要最低限の GPGPU リソースを使うように設定してください．なお，現時点での GPGPU リソースの使用状況は，サーバー上で nvidia-smi コマンドを利用すると確認できる（詳細は別途マニュアル参照）.

Keras で MNIST (つづき (1))

```
# MNIST データの準備
img_rows, img_cols = 28, 28  # 画像サイズは 28x28
num_classes = 10 # クラス数

(X, Y), (Xtest, Ytest) = keras.datasets.mnist.load_data()  # 訓練用とテスト（兼 validation）用のデータを取得
X = X.reshape(X.shape[0],img_rows,img_cols,1) # X を（画像 ID, 28, 28, 1）の 4 次元配列に変換
Xtest = Xtest.reshape(Xtest.shape[0],img_rows,img_cols,1)

X = X.astype('float32') / 255.0  # 各画素の値を 0〜1 に正規化
Xtest = Xtest.astype('float32') /255.0

input_shape = (img_rows, img_cols, 1)

Y = keras.utils.to_categorical(Y, num_classes) # one-hot-vector へ変換
Ytest1 = keras.utils.to_categorical(Ytest, num_classes)
```

以下は，畳み込みニューラルネットのモデルの例である．

Keras で MNIST (つづき (2))

```
# モデルの定義
model = models.Sequential()
# 3x3 の畳み込み層．出力は 32 チャンネル．活性化関数に ReLU．入力データのサイズは input_shape で指定．入出力のサイズ（row と col）が同じになるように設定．
model.add(layers.Conv2D(32, kernel_size=(3,3), activation='relu',
          input_shape=input_shape, padding='same'))
# 3x3 の畳み込み層．出力は 64 チャンネル．活性化関数に ReLU．入力データのサイズは自動的に決まるので設定不要．
model.add(layers.Conv2D(64,(3,3),activation='relu',padding='same'))
# 2x2 の最大値プーリング
model.add(layers.MaxPooling2D(pool_size=(2,2)))
# 入力を 1 次元配列に並び替える
model.add(layers.Flatten())
# 全結合層．出力ノード数は 128．活性化関数に ReLU．
model.add(layers.Dense(128,activation='relu'))
# 全結合層．出力ノード数は num_classes（クラス数）．活性化関数に softmax．
model.add(layers.Dense(num_classes, activation='softmax'))

# 作成したモデルの概要を表示
print (model.summary())
```

このように，Sequential モデルのインスタンスを作成して，入力層から順に add メソッドで層を追加していく．（この他にも，Functional API を使う方法があるが割愛）

モデルの定義後，損失関数や最適化手法を設定し，その後，fit メソッドを用いて学習する．

Keras で MNIST（つづき (3)）

```
# モデルのコンパイル．損失関数や最適化手法を指定する．
# ここでは，損失関数にクロスエントロピー誤差，最適化手法に Adadelta を指定している．
model.compile(
  loss=keras.losses.categorical_crossentropy,
  optimizer=keras.optimizers.Adadelta(), metrics=['acc'])

# 学習．とりあえずここでは 10 エポックだけ学習．　1 バッチあたり 32 枚の画像を利用
epochs = 10
batch_size = 32
result = model.fit(X,Y, batch_size=batch_size,
                epochs=epochs, validation_data=(Xtest,Ytest1))
history = result.history
```

学習途中の損失関数の値の履歴等は fit メソッドの戻り値として取得することができる．

Keras で MNIST（つづき (4)）

```
# 学習済みのモデルをファイルに保存したい場合
model.save("my_model.h5")

# ファイルに保存したモデルを読み込みたい場合
#model = models.load_model("my_model.h5")

# 学習履歴をファイルに保存したい場合
import pickle
with open("my_model_history.dump", "wb") as f:
    pickle.dump(history, f)

# 学習履歴をファイルから読み込みたい場合
#with open("my_model_history.dump", "rb") as f:
#    history = pickle.load(f)
```

次に，10000 枚のテスト用画像 Xtest に対してクラス識別を行い，正答率を評価する．

Keras で MNIST（つづき (5)）

```
# Xtest に対してクラス識別.
pred = model.predict_classes(Xtest)

from sklearn.metrics import confusion_matrix, accuracy_score
# 混同行列 各行が正解のクラス，各列が推定されたクラスに対応
print (confusion_matrix(Ytest, pred, labels=np.arange(10)))
# 正答率の算出
print (accuracy_score(Ytest, pred))
```

Keras で MNIST（つづき (6)）

```
# 損失関数と精度のグラフを表示．縦軸が損失関数の値（or 精度)，横軸がエポック
数
import matplotlib.pyplot as plt
fig = plt.figure()
plt.plot(history['loss'], label='loss') # 教師データの損失
plt.plot(history['val_loss'], label='val_loss') # テストデータの損失
plt.legend()
plt.savefig("loss_history.png")

fig = plt.figure()
plt.plot(history['acc'], label='acc') # 教師データでの精度
plt.plot(history['val_acc'], label='val_acc') # テストデータでの精度
plt.legend()
plt.savefig("loss_acc.png")
```

### 6.4 [発展課題 B1] MNIST 手書き数字認識

前節のサンプルコードを参考に，MNIST 手書き数字認識器を作成し，認識精度を評価せよ．レポートには，構築したモデルの概要，エポック数を横軸，{ 教師データの損失関数の値，テストデータの損失関数の値，教師データでの精度，テストデータでの精度 } を縦軸にとったグラフを添付すること．課題 B1 については，以下の 2 つについて報告すること．

**B1-(1)（わざと）過学習**　モデルによっては過学習，すなわち，学習データでの精度は高いものの，テストデータでの精度が非常に低くなる現象が生じる．構築するモデルを工夫して（わざと）過学習が生じさせ，そのときの振る舞いをレポートにまとめること．

**B1-(2) 認識器の構築**　できるだけ精度が高くなるような認識器を構築せよ．その認識器の構築に至るまでの試行錯誤の内容についてもレポートで報告すること．

## 6.5 [発展課題 B2] 顔画像認識

人間によって,「笑顔」もしくは「笑顔でない」のいずれかがラベル付けされた顔画像を用意している．これらを教師データとして，入力された顔画像に対して,「笑顔」or「笑顔でない」の 2 クラス識別を行う認識器を構築せよ．

GPGPU サーバ上の，/home/iiyama/face/train/smile に笑顔画像，/home/iiyama/face/train/nonsmile に非笑顔画像があり，これを教師データとして用いる．

また，/home/iiyama/face/test/smile に笑顔画像，/home/iiyama/face/test/nonsmile に非笑顔画像があり，これを検証用データとして用いる．

これらの画像を（MNIST のときのように）一度にメモリ上に読み込むことはできないため，以下のように逐次ファイルから読み込むようにする．

顔画像データ読み込み

```
# 「Keras で MNIST（つづき (1)）」と差し替えること
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1.0/255)
test_datagen = ImageDataGenerator(rescale=1.0/255)

input_shape = (218, 178, 3)
num_classes = 2
batch_size = 32

train_generator =
 train_datagen.flow_from_directory('/home/iiyama/face/train/',
                            target_size=(218,178),
                            batch_size=batch_size,
                            class_mode='categorical')

test_generator =
 test_datagen.flow_from_directory('/home/iiyama/face/test/',
                            target_size=(218,178),
                            batch_size=batch_size,
                            class_mode='categorical')
```

学習時．fit メソッドではなく，fit_generator メソッドを使う

```
# 「Keras で MNIST（つづき (3)）」と差し替えること
model.compile(
  loss=keras.losses.categorical_crossentropy,
  optimizer=keras.optimizers.Adadelta(), metrics=['acc'])

# 学習.
epochs = 10
spe = 100 # 1 エポックあたりのバッチ数
result = model.fit_generator(train_generator, steps_per_epoch = spe,
   epochs=epochs, validation_data=test_generator, validation_steps=30)
history = result.history
```

精度評価

```
# 「Keras で MNIST（つづき (5)）」と差し替えること
loss, acc = model.evaluate_generator(test_generator, steps = 30)
print ("loss:", loss)
print ("accuracy:", acc)
```

手書き数字認識と比べて，クラス数は 2 クラスに減ったものの，画像サイズは大きくなり，また，タスクも難しくなっている．モデル構造，学習率などをチューニングする必要があるので注意すること．

レポートには，発展課題 B-1(2) と同じく，認識器の構築に至るまでの試行錯誤の内容についてもレポートで報告すること．

## 6.6 [発展課題 B3] 画像生成（他の発展課題に飽きた人向け）

画像データを訓練データとして，訓練データには無い（本物らしい）画像を生成するネットワークを構築せよ．画像データとしては，MNIST もしくは cifer10 のデータを用いること．（発展課題 B2 の顔画像データを用いてもよいが難易度は高い）

画像の生成モデルを構築する方法はいくつかあるが，以下では，敵対的生成ネットワーク（Generative Adversarial Network, GAN）および GAN を用いた画像生成手法 DCGAN[6] について説明する．

GAN では，generator（生成器）と discriminator（識別器）と呼ばれる 2 つのネットワークを用意し，それぞれを交互に学習させる．generator の目的は訓練データとよく似た画像を生成することであり，一方では discriminator の目的は generator が生成した（偽物の）画像と，訓練データの（本物の）画像とを識別することである．目的が相反する 2 つのネットワークを互いに競わせることにより高性能な generator を構築する．

**定式化**　generator は $d_n$ 次元のガウス分布 $p_z(\boldsymbol{z})$ に従うノイズ $\boldsymbol{z}$ を入力とし，画像 $\boldsymbol{x}$ を出力する．generator を $G(\boldsymbol{z};\theta_g)$ と表記し，これをパラメータ $\theta_g$ を持つニューラルネットで

構築する．discriminator は画像 $\boldsymbol{x}$ を入力として受け取り，$\boldsymbol{x}$ が本物かどうかの確率を出力する．discriminator を $D(\boldsymbol{x};\theta_d)$ と表記し，これもパラメータ $\theta_d$ を持つニューラルネットで構築する．

generator は「discriminator を騙す」画像を生成するようパラメータ $\theta_g$ を学習する．つまり，generator の出力 $G(\boldsymbol{z})$ を discriminator に入力したときの出力 $D(G(\boldsymbol{z}))$ ができるだけ 1 に近付くようにしたい．generator の学習では，以下の式を最小とする $\theta_g$ を学習する．

$$\mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))] \tag{66}$$

一方，discriminator は式 66 をできるだけ大きくしたい．また，本物の画像 $x$，言い換えると本物の画像を生成する分布 $p_{data}(\boldsymbol{x})$ に従う $x$，が入力されたときにその出力 $D(\boldsymbol{x})$ をできるだけ大きくしたい．

すなわち，GAN の学習では以下の式を用いて $\theta_g$ と $\theta_d$ を学習する．[5]

$$\underset{\theta_d}{\operatorname{argmax}}\,\underset{\theta_g}{\operatorname{argmin}}\left(\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})]+\mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))]\right) \tag{67}$$

**アルゴリズム**

**Data:** 画像集合 $X$, ミニバッチサイズ $B$
# モデルの準備;
(i) discriminator のモデルを構築する;
(ii) generator のモデルを構築する;
(iii) (i) のモデルと (ii) のモデルを結合したモデルを構築する．ただし，(ii) のパラメータは更新しないようにしておくこと．;
**while 気が済むまで do**
  # discriminator の学習;
  (1) $d_n$ 次元ガウス分布 $p_z(\boldsymbol{z})$ から $B/2$ 個の $d_n$ 次元ベクトルを生成;
  (2) $X$ から $B/2$ 枚の画像を抽出;
  (3) からモデル (ii) を使って画像を生成;
  (4) (2) の正解を 1，(3) の正解を 0 として合計 $B$ 枚の画像を用いてモデル (i) を学習;
  # generator の学習;
  (1) $d_n$ 次元ガウス分布 $p_z(\boldsymbol{z})$ から $B$ 個の $d_n$ 次元ベクトルを生成;
  (2) (1) の正解をすべて 1 として，モデル (iii) を学習;
**end**

**Algorithm 1:** (DC)GAN の学習

**実装にあたって**　以下はあくまで実装の一例である．GAN 実装はチューニングが非常に難しいことが知られているため,（かなりの）試行錯誤が必要である．

参考までに DCGAN の論文で指摘されている情報を以下に載せる．

[5] generator の学習において，実際は，$\log(1-D(G(\boldsymbol{z})))$ を最小化するのではなく，$-\log D(G(\boldsymbol{z}))$ を最小化するようにした方が収束が早い．

- 画像の各ピクセルは $[-1, 1]$ に正規化
- ミニバッチサイズは 128.
- $\boldsymbol{z}$ の次元 $d_n$ は 100 程度
- プーリングではなく，stride（※ Keras のドキュメントを参照）付きの畳み込みを用いる
- Batchnormalization を導入する
- generator の活性化関数は ReLU を用いる．ただし最終層は tanh.
- discriminator の活性化関数は LeakyReLU（ReLU の改良版）を用いる．

Keras で実装する場合，functional API が必要となる．詳細は，Keras のドキュメントを参照のこと．他にも，レイヤーを freeze するテクニック（`https://keras.io/ja/getting-started/faq/#freeze`）を駆使する必要がある．

## 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/3.8/tutorial/index.html`

[4] “NumPy Quickstart tutorial”,`https://docs.scipy.org/doc/numpy-1.15.1/user/quickstart.html`

[5] “cuDNN: Efficient Primitives for Deep Learning”, arXiv:1410.0759, 2014.

[6] “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, Alec Radford, Luke Metz, Soumith Chintala, `https://arxiv.org/pdf/1511.06434`