機械学習のための
データ前処理の基本と実践法
8

畳み込みニューラルネットワークの実装

徳島大学
デザイン型AI教育研究センター
鳥井 浩平

今回の内容

1. MNISTデータセット
2. モデルの実装と学習
3. 学習結果のグラフ表示

前回の復習

画像認識、畳み込みニューラルネットワーク

画像認識とは

- 画像に何が写っているのかを認識すること
- 深層学習モデルの導入により精度が飛躍的に向上した

入力

画像認識システム

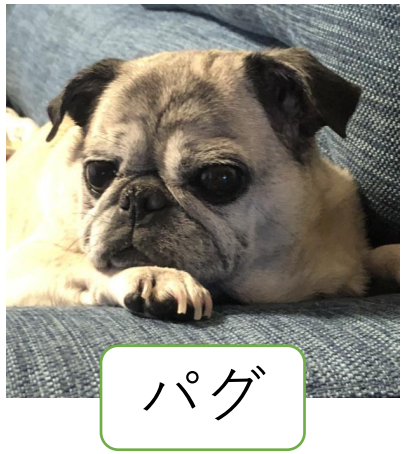
出力

パグ

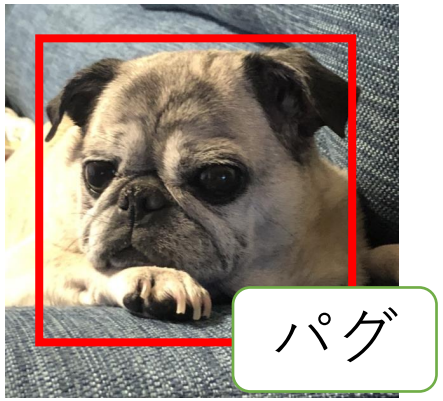
画像認識の種類

- 画像認識は主に以下の3つに分けられる
  画像分類：画像全体に対して種類を予測する
  物体検出：物体の位置と物体の種類を予測する
  セグメンテーション：画素単位で種類を予測する
- 画像分類＜物体検出＜セグメンテーションの順に難易度が上がる

画像分類



物体検出



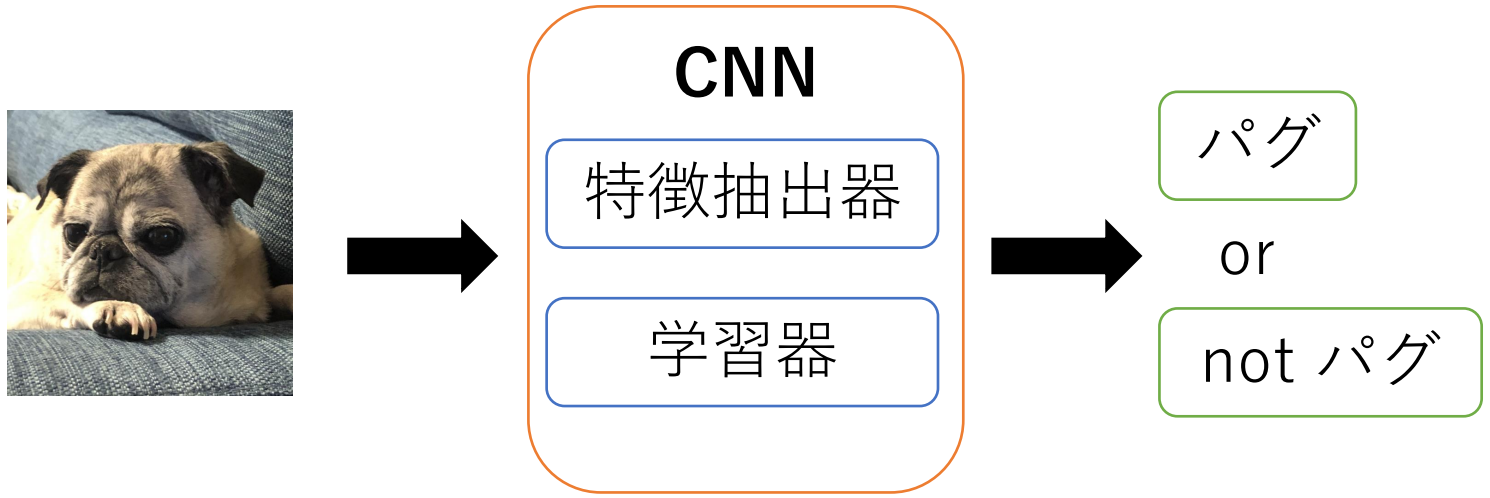
セグメンテーション

畳み込みニューラルネットワーク

- **Convolutional Neural Network (CNN)**
- **ニューラルネットワーク（NN）**に**畳み込み処理**を導入したもの
- 画像に特化したネットワーク構造をもつ
- 1998年にYann LeCunが提案したLeNetが元祖CNN
  Yann LeCun：2019年にチューリング賞を受賞
- 現代の高精度な画像認識AIの多くはCNNを用いている

CNNの構造

- 主に**畳み込み層**、**プーリング層**、**全結合層**で構成される
- 畳み込み層とプーリング層は特徴量抽出の役割をもつ
- 全結合層は学習器の役割をもつ
- CNNは一人二役の**エンドツーエンド**な手法



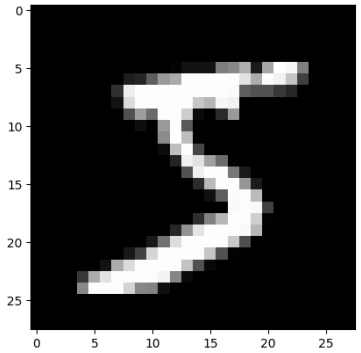
7.1 MNISTデータセット

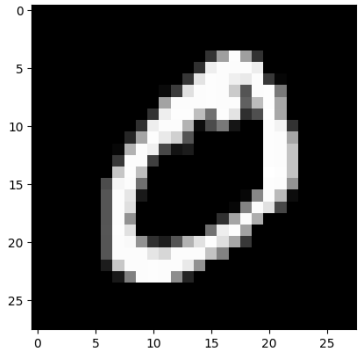
MNISTデータセットとは

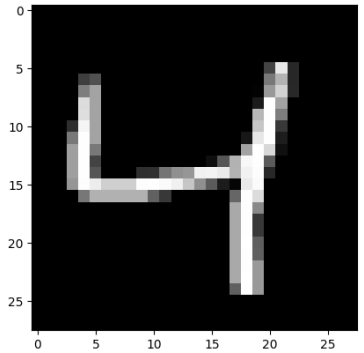
MNISTデータセット

- 手書き文字認識用のデータセット
- 数字の0~9を手書きした画像とラベルのセット
- 画像は28×28のグレースケール画像
- 学習データ60,000枚、テストデータ10,000枚

画像







ラベル

5

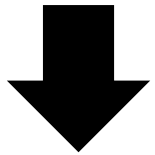
0

4

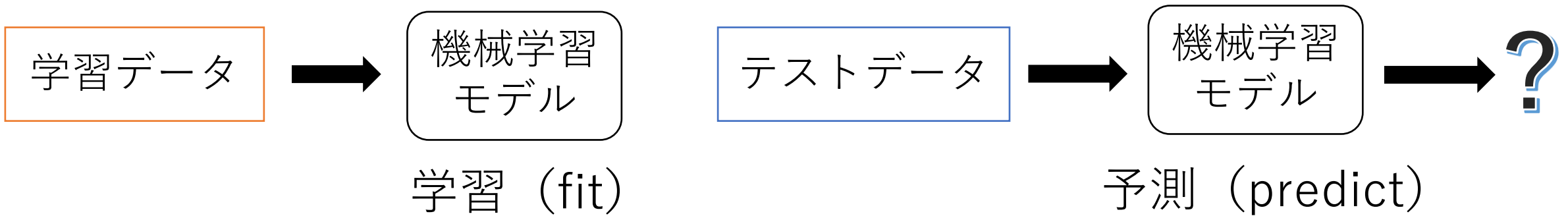
学習データとテストデータ

精度が高い≒誤差が小さい

- 機械学習モデルは学習したデータ（**学習データ**）に対して精度が高い
  学習したのだから当たり前

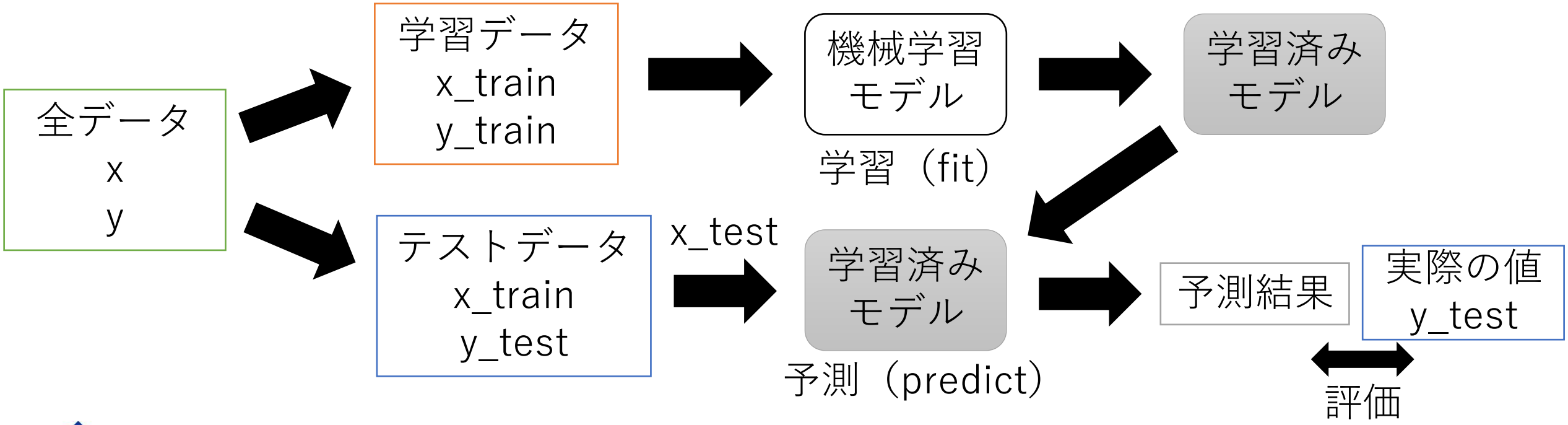


- 学習していないデータ（**テストデータ**）による評価が重要である
  テストデータに対する精度も高ければそのモデルは良いモデルといえる



機械学習の実験プロセス

1. 全データを学習データとテストデータに分ける
2. 学習データを用いて機械学習モデルの学習（fit）を行う
3. テストデータを用いて機械学習モデルの評価（predict）を行う



MNISTデータセットの読み込み

```
# MINISTデータセットの読み込み
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 4次元テンソル形式に変換
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
```

4次元テンソル

- CNNの入力は**テンソル**という形式で与えられる
  テンソル≒多次元配列
- 画像の場合は基本的に（データ数、高さ、幅、チャネル数）で表現
  カラー画像ならチャネル数は3、グレースケール画像ならチャネル数は1

MNISTデータセットの場合は...

学習データ　（60000, 28, 28, 1）
テストデータ（10000, 28, 28, 1）

データの変換

```
# データの正規化および実数値化
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# ラベルデータをOne-hotベクトルに変換
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

正規化・実数値化

正規化

- データを0~1の値域に変換する処理
- データに対してデータの最大値で除算する
- 画像の場合は最大画素値である**255**で除算する
  ただし8bit画像の場合に限る（16bitの場合は65535で除算する）
- 正規化を行うことでデータのスケールを揃えることができる

実数値化

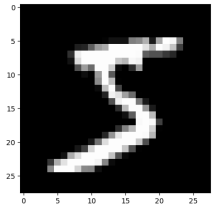
- 画像の場合、元データは整数（unsigned int）である
- CNNの計算用に単精度浮動小数点数（float32）に変換する

One-hotベクトル

- ある1つの要素を1、それ以外を0としたベクトル
- One-hotベクトルに変換することを**One-hotエンコーディング**という
- One-hotエンコーディングはラベルをベクトルとして扱うために必要

MNISTデータセットの場合は…

ラベルは0~9の10種類→10次元のベクトル



5 ➡ **(0, 0, 0, 0, 0, 1, 0, 0, 0, 0)**

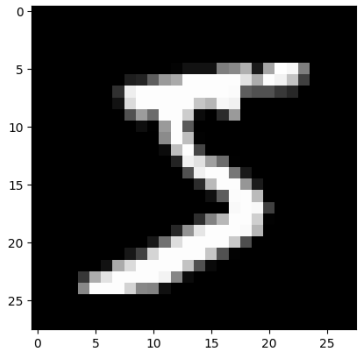
0 1 2 3 4 5 6 7 8 9

7.2 モデルの実装と学習

実際にCNNを実装してみよう

モデルのイメージ

- 画像を入力すると画像に写る数字が何であるかを出力する
- 出力は10次元のベクトル



CNN

0 1 2 3 4 5 6 7 8 9

(0, 0, 0, 0.1, 0, 0.9, 0, 0, 0, 0)

ハイパーパラメータの設定

```
# バッチサイズ、エポック数の設定
batch_size = 128
epochs = 20
```

【補足】

ハイパーパラメータ	人が決定する定数
バッチサイズ	ミニバッチ学習における1グループのデータ数
エポック数	学習回数

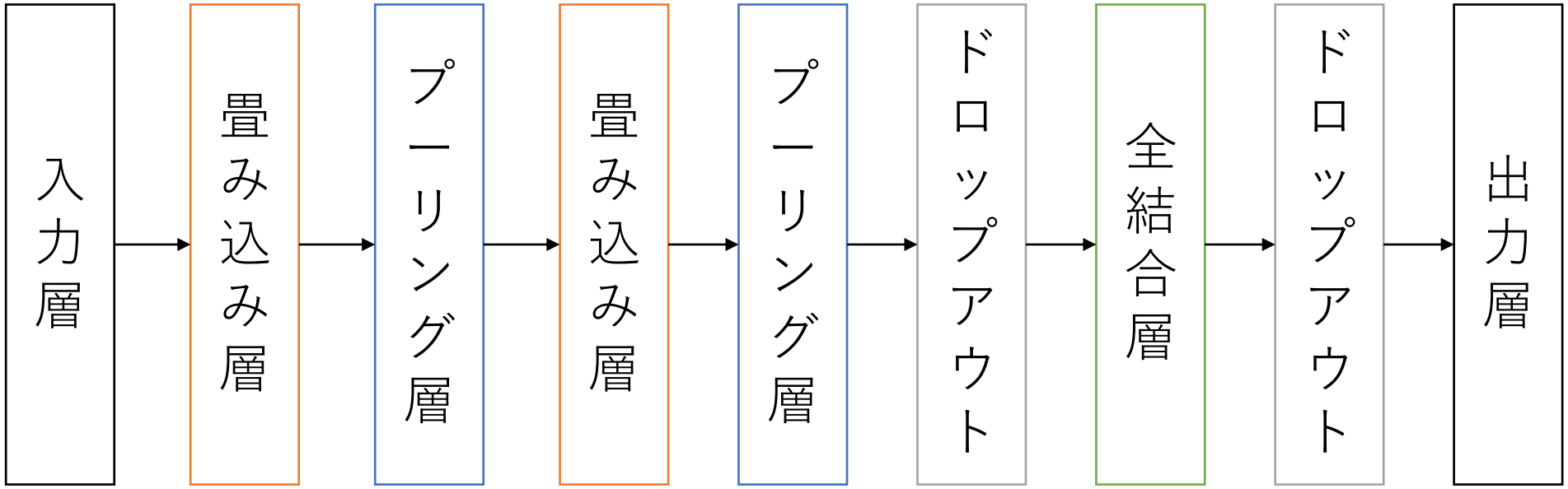
CNNモデルの定義

```
# CNNモデルの定義
model = Sequential()
model.add(Conv2D(64, (3, 3), padding='same', input_shape=(28, 28, 1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(10, activation='softmax'))

# CNNモデルの可視化
model.summary()
```

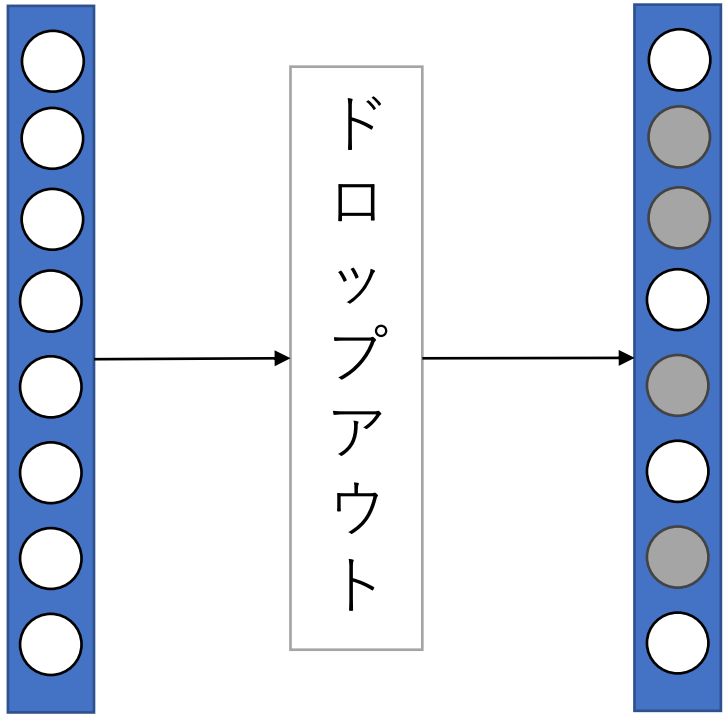
CNNモデルの概要図

今回は以下のような構成のCNNモデルを構築する



ドロップアウト

- ランダムにユニットの値を0に変換する
- 過学習の防止に効果がある



Sequential

model = Sequential()

- ニューラルネットワークモデルの初期化に用いるクラス
- **add**メソッドで層を定義するクラスを追加してネットワークを構築

Conv2D

2次元畳み込み層を定義するクラス

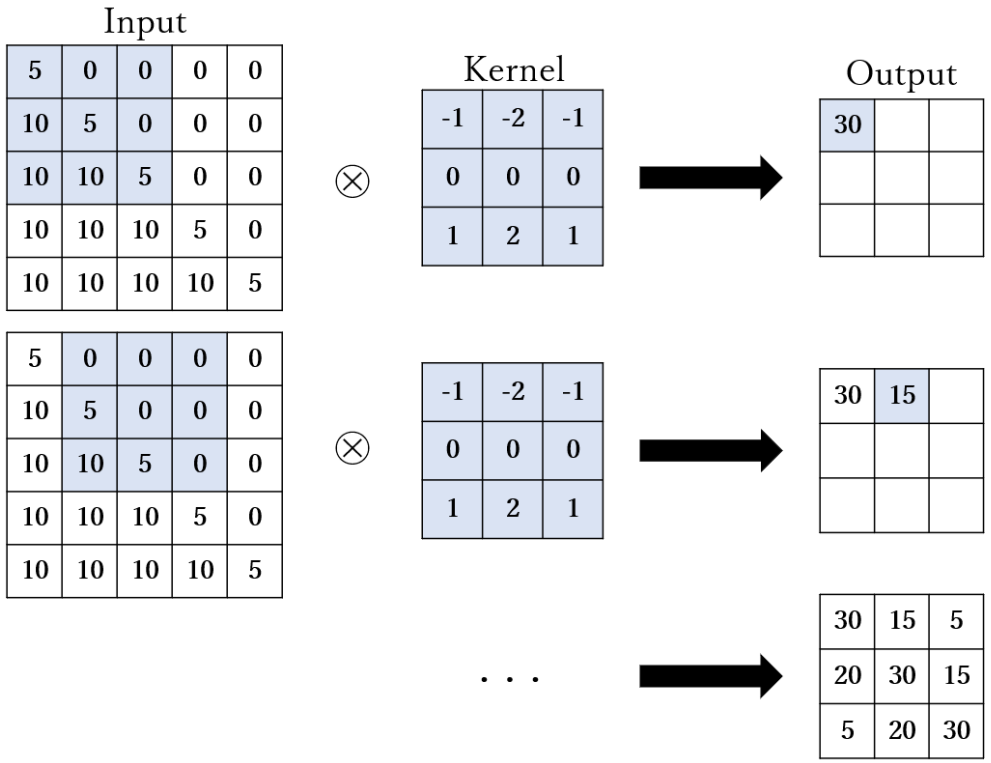
```
model.add(Conv2D(64, (3, 3), padding='same', input_shape=(28, 28, 1), activation='relu'))
model.add(Conv2D(128, (3, 3), activation='relu'))
```

【引数】

第1引数	フィルタ数
第2引数	フィルタサイズ
padding	"same"にするとゼロパディングを行う
input_shape	入力サイズ（前層が存在しない場合に必要）
activation	活性化関数の種類

パディング

- 入出力サイズが同じになるようにデータを補完する処理
- 0埋めを行う**ゼロパディング**が主流



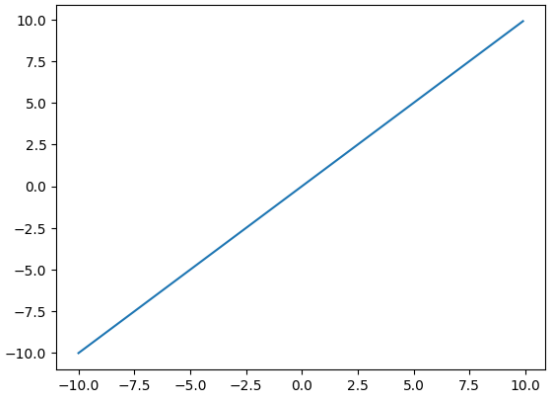
30	15	5
20	30	15
5	20	30

0	0	0	0	0
0	30	15	5	0
0	20	30	15	0
0	5	20	30	0
0	0	0	0	0

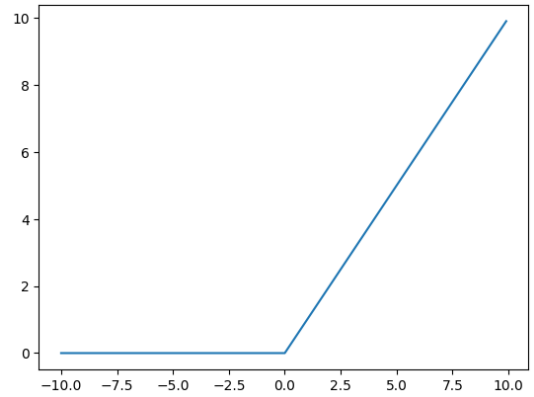
活性化関数

<u>よく使われる活性化関数</u>

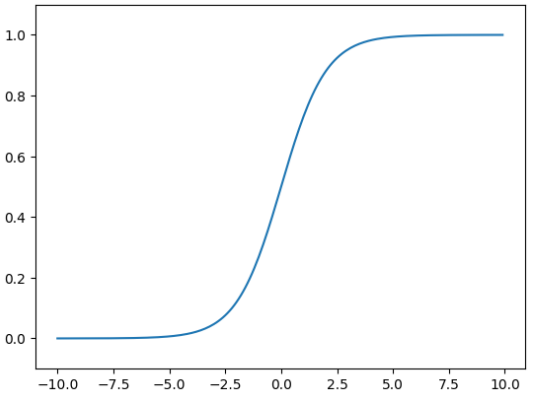
- 恒等関数（Linear）
  $f(x) = x$　つまりそのまま出力
  主に回帰の出力層で使用
- ReLU（Rectified Linear Unit）
  主に中間層で利用
- Sigmoid
  主に出力層で利用
- Softmax
  主に多クラス分類の出力層で利用



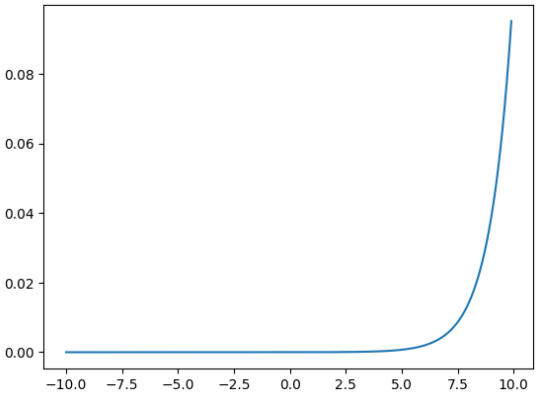
恒等関数



ReLU



Sigmoid



Softmax

MaxPooling2D

最大値プーリング層を定義するクラス

```
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

【引数】

pool_size　　最大値プーリングに用いるフィルタのサイズ

Dropout

ドロップアウトを定義するクラス

```
model.add(Dropout(0.5))
model.add(Dropout(0.25))
```

【引数】

第1引数　　ドロップアウトするユニットの割合

Flatten

1次元ベクトルへの変換を定義するクラス

```
model.add(Flatten())
```

Dense

全結合層を定義するクラス

```
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

出力層のユニット数
＝ラベルの種類の数

【引数】

第1引数	ユニット数
activation	活性化関数の種類

summary

model.summary()

- 構築したネットワークの情報を表示するメソッド
- パラメータの数やテンソルの形などを一括して見ることができる

summaryの見方

- Layerは層の種類
- Output Shapeはテンソルの形
  チャネル数＝特徴マップの数
  ＝フィルターの数
- Paramはパラメータの数
- Total Paramsはパラメータの合計
- Trainable Paramsは学習によって値が定まるパラメータの合計

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 28, 28, 64)        640

 max_pooling2d_2 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 12, 12, 128)       73856

 max_pooling2d_3 (MaxPoolin  (None, 6, 6, 128)         0
 g2D)

 dropout_2 (Dropout)         (None, 6, 6, 128)         0

 flatten_1 (Flatten)         (None, 4608)              0

 dense_2 (Dense)             (None, 128)               589952

 dropout_3 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 665738 (2.54 MB)
Trainable params: 665738 (2.54 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

CNNモデルのコンパイル

```
# CNNモデルのコンパイル
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

【引数】

loss	誤差関数の種類
optimizer	最適化手法の種類
metrics	追加する評価指標の種類

誤差関数の設計（二値分類の場合）

- 二値分類では**二値交差エントロピー（Binary Cross Entropy）**を用いる

$$E(\mathbf{w}) = -[t \log y(\mathbf{x}) + (1-t) \log\{1 - y(\mathbf{x})\}]$$

- 計算結果を$y(\mathbf{x})$、教師データを$t = \{0, 1\}$とする
- $t = 1$のときは$-\log y(\mathbf{x})$、$t = 0$のときは$-\log\{1 - y(\mathbf{x})\}$が誤差となる

  例）計算結果が0.78、正解がパグである（$t = 1$）のとき

  $E = -\log 0.78 = -(-0.1079) = 0.1079$

多クラス交差エントロピー

categorical_crossentropy

- 多クラス分類などに用いる誤差関数
- t_cと$y_c(\mathbf{x})$の分布が近いほど0に近い値となる

$$E(\mathbf{w}) = -\sum_c t_c \log y_c(\mathbf{x})$$

パラメータの更新

- 以下の式によるパラメータの更新方法を**勾配降下法** という

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$$

- αは**学習率**といい、パラメータの更新度合いを決める値
  学習率は人があらかじめ決めておく
- 誤差を偏微分した値（勾配）を用いてパラメータを更新する
- 勾配降下法にはさまざまな進化系がある
  モメンタム、RMSProp、**Adam**など
- 最近では「とりあえずAdam」というイメージ

Accuracy

- 予測と答えがどれくらい合っているかを表す評価指標のひとつ
- 正解率、精度とも呼ばれる

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

TP　True Positive　真陽性
TN　True Negative　真陰性
FP　False Positive　偽陽性
FN　False Negative　偽陰性

	答えは真	答えは偽
予測は真	TP	FP
予測は偽	FN	TN

予測値の扱い

- 出力層の活性化関数がSoftmaxなので、出力は確率として出力される
- しきい値処理や四捨五入などにより出力を真(1)or偽(0)に変換する
- TensorFlow(Keras)の場合は四捨五入がデフォルト

予測　　四捨五入

(0, 0, 0, 0.15, 0, 0.8, 0, 0.05, 0, 0) ➡ (0, 0, 0, 0, 0, 1, 0, 0, 0, 0)

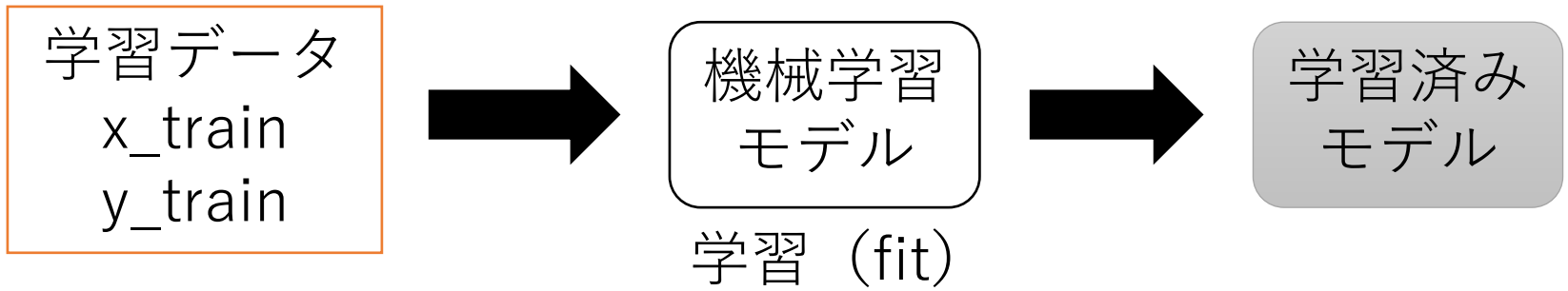
(0, 0, 0, 0, 1, 0, 0, 0, 0, 0)　正解

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

TN TN TN TN FN FP TN TN TN TN

CNNモデルの学習

```
# CNNモデルの学習
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
```



validation_data

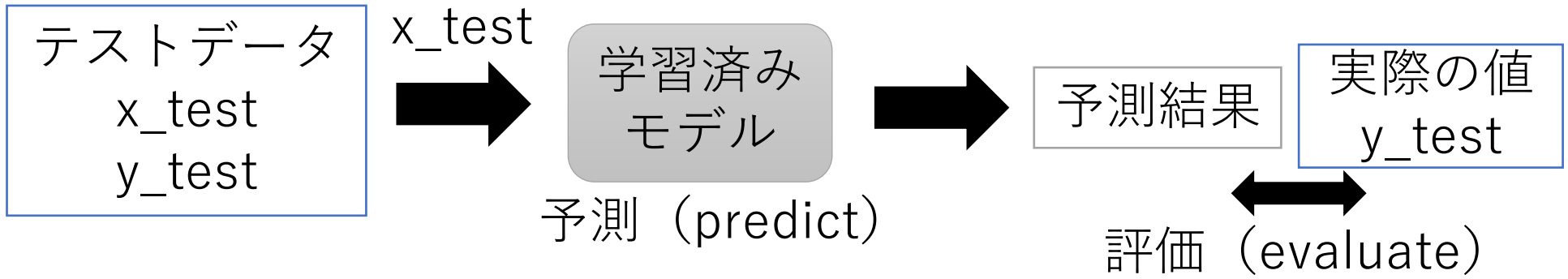
- validation_dataは**検証データ**と呼ばれる
- 検証データは学習に用いない
- 学習の成否の確認や学習率の調整に用いる
- 今回は検証データにテストデータを用いている
- 学習データ、テストデータとは別に検証データを用意することもある

7.3 学習結果のグラフ表示

さまざまなデータを可視化しよう

CNNモデルの評価

```
# CNNモデルの評価
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```



誤差のグラフ化

```
# 予測誤差のグラフ化
plt.plot(range(epochs),
         history.history['loss'],
         marker='o', color = 'red', label='loss')
plt.plot(range(epochs),
         history.history['val_loss'],
         marker='v', linestyle='--', color='green', label='val_loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(loc='best')
plt.show()
```

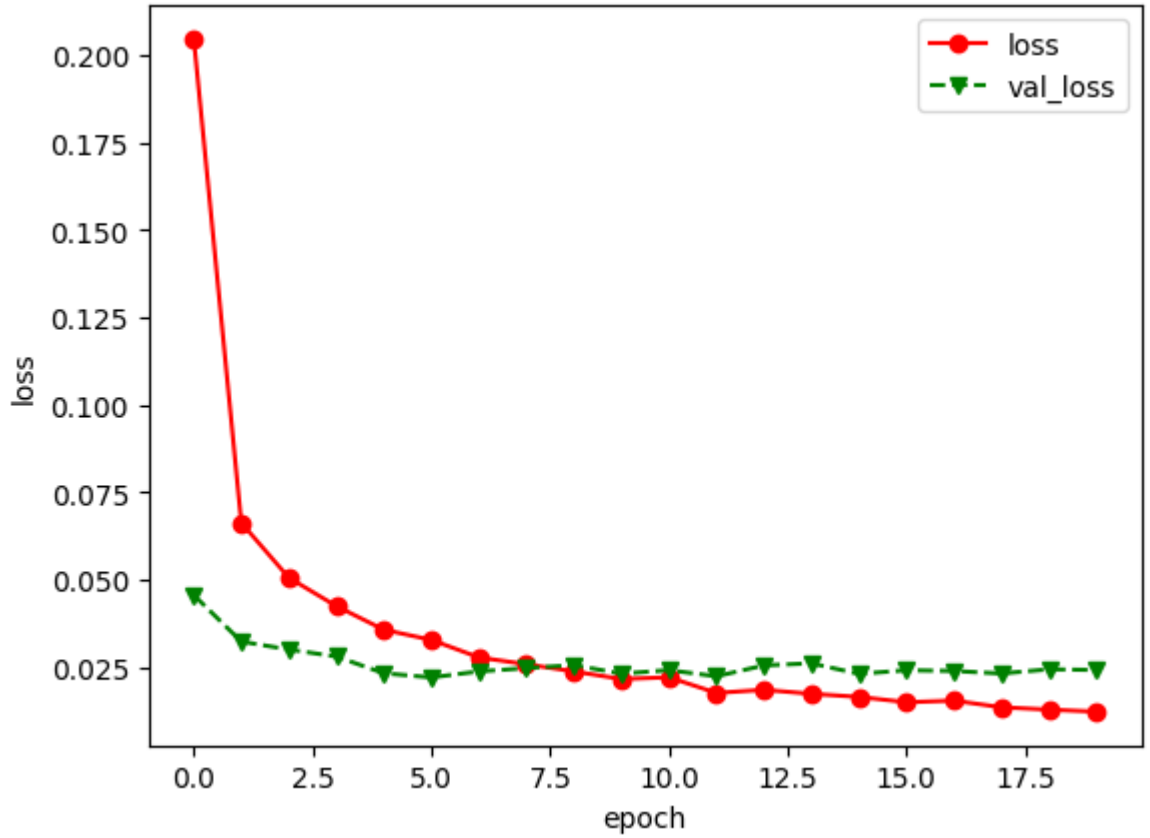
plt.plot

折れ線を描画する関数

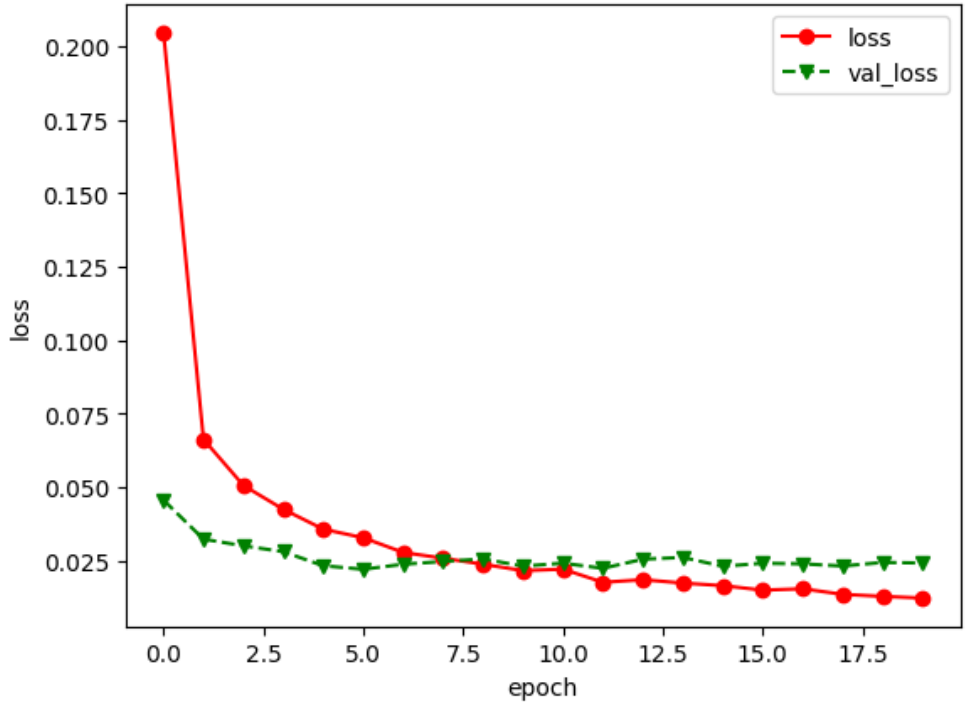
```
plt.plot(range(epochs),
         history.history['loss'],
         marker='o', color = 'red', label='loss')
plt.plot(range(epochs),
         history.history['val_loss'],
         marker='v', linestyle='--', color='green', label='val_loss')
```

【引数】

第1引数	データx
第2引数	データy
marker	点の形状
linestyle	折れ線のスタイル
color	折れ線の色
label	データラベル



グラフの設定と表示



<u>オプション</u>

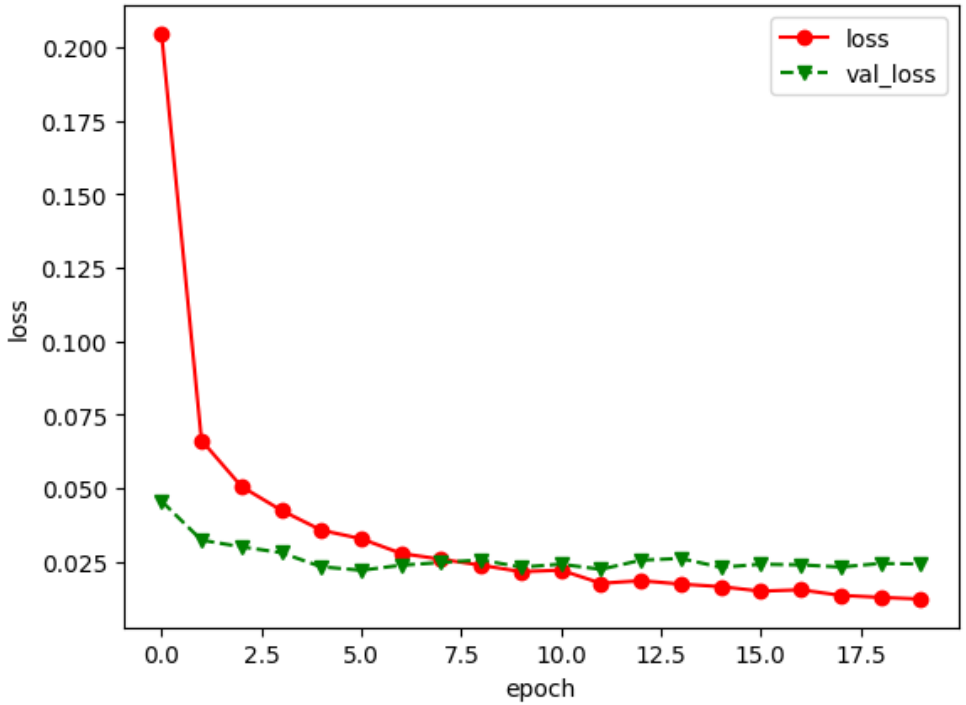
plt.xlabel	x軸ラベル
plt.ylabel	y軸ラベル
plt.legend	凡例の詳細設定、loc="best"で適切な位置に凡例表示

<u>グラフの表示</u>

plt.show	グラフを表示する

誤差グラフの見方

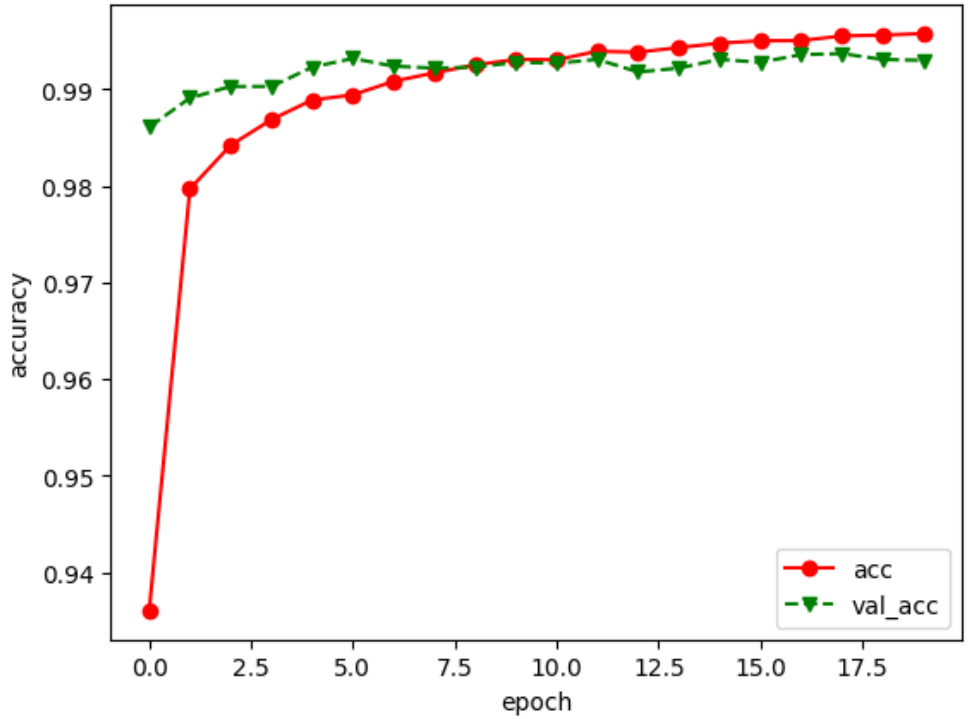
- 赤が学習データに対する誤差
- 緑が検証（テスト）データに対する誤差
- 両方とも右肩下がりがベスト



【よくある事例】

- 緑だけ右肩上がり→**過学習**を疑う
- 誤差が安定しない→学習データの質を疑う

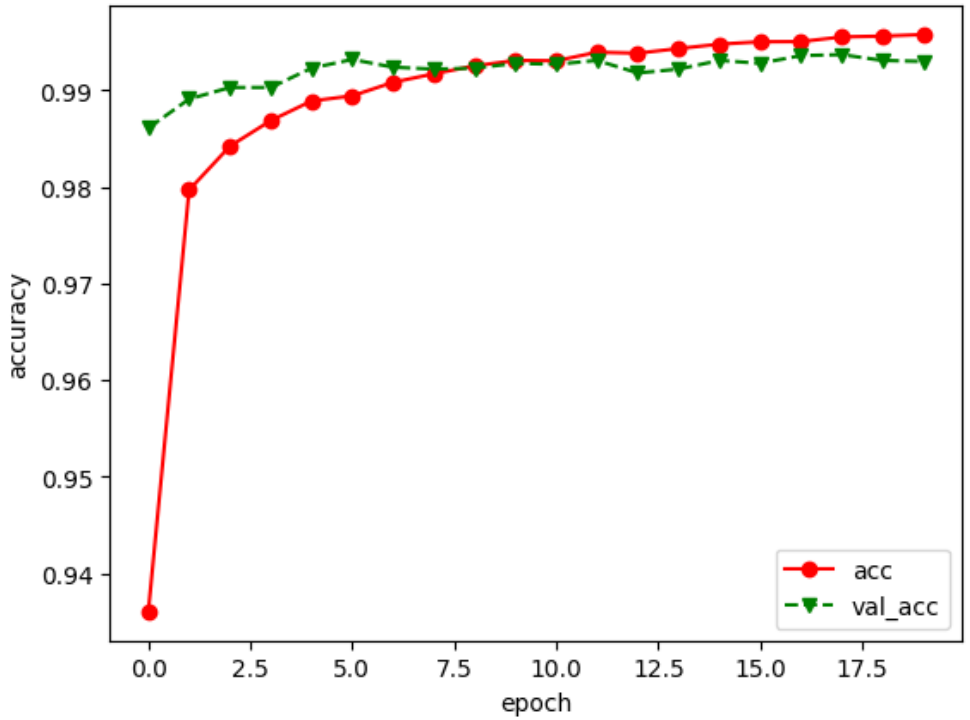
Accuracyのグラフ化




```
# Accuracyのグラフ化
plt.plot(range(epochs),
         history.history['accuracy'],
         marker='o', color = 'red', label='acc')
plt.plot(range(epochs),
         history.history['val_accuracy'],
         marker='v', linestyle='--', color = 'green', label='val_acc')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.legend(loc='best')
plt.show()
```

Accuracyグラフの見方

- Accuracyは1（100%）に近ければ近いほど良い
- 両方とも右肩上がりがベスト



まとめ

- MNISTデータセットとデータの前処理について学んだ
- CNNの実装を行った
- 学習結果の可視化を行った

【次回】
- データ拡張を行う
- ファインチューニングを行う

演習

- MNISTデータセットを準備する
- CNNモデルを構築・学習する
- 学習結果を可視化する

サンプルプログラム

https://github.com/wt501/sample_programs