知能プログラミング演習 I

第 7 回: 畳み込みニューラルネットワーク II

梅津 佑太
2 **号館** 404A: umezu.yuta@nitech.ac.jp

課題のダウンロード

前回作ったディレクトリに移動して今日の課題のダウンロードと解凍

step1: `cd ./DLL`

step2: `wget http://www-als.ics.nitech.ac.jp/~umezu/Lec7.zip`

step3: `unzip Lec7.zip`

- ✓ **まだ** DLL **のフォルダを作ってない人は**, step1 **の前に**
  `mkdir -p DLL`
  でフォルダを作成する

講義ノート更新しました.

今日の講義内容

1. **畳み込みニューラルネットワークの逆伝播と** keras **による実装**

前回の復習

畳み込みニューラルネットワークの構成要素:

- **畳み込み層**: **入力** $\mathbb{R}^{d\times d\times K} \ni Z \mapsto Z' \in \mathbb{R}^{d'\times d'\times M}$:

$$u_{ijm} = \langle W_m, Z_{ij} \rangle + b_m \mapsto z'_{ijm} = f(u_{ijm})$$

- **プーリング層**: **入力** $\mathbb{R}^{d\times d\times K} \ni Z \mapsto Z' \in \mathbb{R}^{d'\times d'\times K}$:

$$u_{ijk} = \langle W_{ijk}, Z_k \rangle \mapsto z'_{ijk} = u_{ijk}$$

- **全結合層**: **入力** $\mathbb{R}^{d\times d\times K} \ni Z \mapsto Z' \in \mathbb{R}^{d'}$:

$$u_i = \langle W_i, Z \rangle + b_i \mapsto z'_i = f(u_i)$$

ただし, $\langle \cdot, \cdot \rangle$ **はテンソルの内積**

$$\langle A, B \rangle = \sum_{ijk} a_{ijk} b_{ijk}, \quad A, B \in \mathbb{R}^{d\times d\times K}$$

$$\langle A, B \rangle = \sum_{ij} a_{ij} b_{ij} = \mathrm{tr}(A^\top B), \quad A, B \in \mathbb{R}^{d\times d}$$

逆伝播

- c.f., **これまでの逆伝播**: **適当なサイズの行列** W_{r+1} ($r+1$ **層目のパラメータ**) **と** δ_{r+1} **を用いて**,

$$\Rightarrow \delta_{r,j} = \tilde{\boldsymbol{w}}_{r+1,j}^{\top} \delta_{r+1} \nabla f(u_{r,j}) = \langle \tilde{\boldsymbol{w}}_{r+1,j}, \boldsymbol{\delta}_{r+1} \rangle \nabla f(u_{r,j})$$

とかけた[1]. **つまり**, δ_r **は** "$r+1$ **層目のパラメータ** $\boldsymbol{w}_{r+1,j}$ **と誤差** δ_{r+1} **に** r **層目の勾配** $\nabla f(u_{r,j})$ **の積を適当な順番に並べたもの**"

- **畳み込みニューラルネットワークの場合も適当なサイズのパラメータ** $W_{r+1,ijk}$ **や** δ_{r+1}, $\nabla f(u_{r,ijk})$ **を用いて**

$$\delta_{r,ijk} = \langle W_{r+1,ijk}, \boldsymbol{\delta}_{r+1} \rangle \nabla f(u_{r,ijk})$$

✓ $\nabla f(u_{r,ijk})$ **は活性化関数やプーリングに用いた関数の勾配**

[1] $\tilde{w}_{r+1,j}$ **は** W_{r+1} **の第** j **列ベクトル**

逆伝播

- **実際に** numpy **だけで逆伝播を実装するのはかなり大変**
  - ✓ **ナイーブに実装すると** 6 **重** for **文とかが出てきて面倒**
  - ✓ **興味がある人は講義ノートなどを見て大変さを実感してみると良い**

- **ということで**, keras[2] **を使って畳み込みニューラルネットワークを実装する**
  - ✓ `import keras` **で** keras **をインポートできる**

[2] tensorflow **というライブラリのラッパー**. **非常に直感的に実装できる**. keras **の使い方や**, kerasu **の関数は** `https://keras.io/ja/` **で確認できる**.

keras のモジュール I

keras は色々なモジュールを含んでいて, 以下のように読み込む.

- `from keras.datasets import mnist`
  - ✓ MNIST データ. そのほか, CIFAR100[3] やボストンの住宅価格データ [4] など, いくつかのデータセットが用意されている.

- `from keras.utils.np_utils import to_categorical`
  - ✓ 正解ラベル y を one-of-K 表記に変換する関数

- `from keras.models import Sequential, Model`
  - ✓ ネットワークを定義するための関数
    - Sequential はシンプルなネットワークを簡単に記述できる
    - Model は分岐があるネットワークや, 生成モデルなどの複雑なモデルを記述するための関数. Functional API という.

[3] 100 クラスからなる 32×32 のカラー画像
[4] 回帰用のベンチマークデータ

keras のモジュール II

- `from keras.layers import Activation`
  - ✓ ReLU や sigmoid, ハイパボリックタンジェントなどの活性化関数.

- keras.layers には, `Flatten`, `Dense`, `Conv2D`, `MaxPooling2D`, `Input` なども含まれている
  - ✓ Flatten は配列を 1 次元に変換, Dense は全結合, Conv2D は畳み込み, MaxPooling2D はマックスプーリングの出力を返す.[5] また, Input は functional API で実装する際に利用する.

[5] マックスプーリングのほか, AveragePooling2D なども利用できる. 2D は, 2 次元データ (行列) に対する処理であることを指している.

Conv2D の引数

```
Conv2D(
filters,      # 出力するフィルタ（チャネル）の枚数
kernel_size,      # 畳み込みのカーネル（フィルタ）の大きさ
strides,      # ストライド数[6]
padding,      # パディング数[7]
activation,       # 活性化関数
input_shape       # 入力の次元（はじめの層のみ）
)
```

- input_shape は 3 次元配列でなければならない
- activation は別のレイヤーとしても定義可能:
  e.g., Activation('relu')

---

[6] (i, j) なら，それぞれの軸でのストライド数を指定できる．デフォルトは (1,1)

[7] "valid" (パディングしない) or "same" (入力と同じサイズの配列を出力) を指定する．デフォルトは"valid"

Dense と MaxPooling2D の引数

```
Dense(
units,     # 全結合層の出力のユニット数
input_shape     # 入力の次元（はじめの層のみ）
)
```

```
MaxPooling2D(
pool_size,     # プーリングのフィルタサイズ [8]
padding    # パディング数 [9]
)
```

[8] (i, j) なら，それぞれの軸でのを指定できる．デフォルトは (2,2)
[9] "valid" (パディングしない) or "same" (入力と同じサイズの配列を出力) を指定する．デフォルトは"valid"

Sequential による実行例

ネットワークの設計は add **を用いて行う**. **例えば**,
入力層 (input_shape $= (28, 28, 1)$))[10]$\rightarrow$ **畳み込み層** $\rightarrow$
Max **プーリング** $\rightarrow$ **全結合層** $\rightarrow$ **出力層** $(m = 10)$
のネットワークを設計する場合,

```
model = Sequential()
model.add((Conv2D(9, (3, 3), padding="same",
           activation="relu", input_shape=(28, 28, 1))))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(64, activation="relu"))
model.add(Dense(m, activation="softmax"))
```

[10]MNIST **など**, **元の** shape **が** (28, 28) **などの** 2 **次元配列の場合**, newaxis **や** reshape **を使って** 3 **次元配列に変換しておく**.

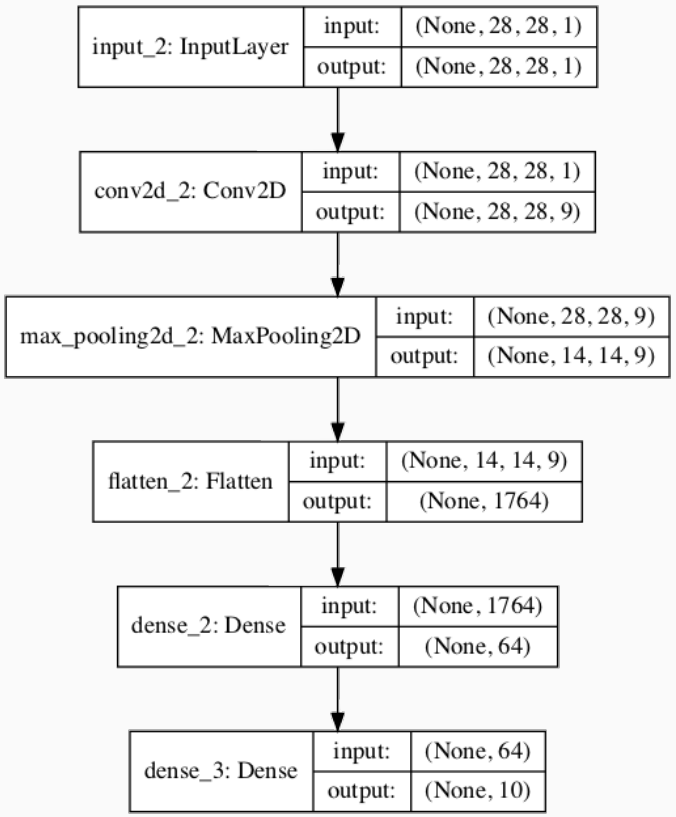
Model による実行例

同じネットワークを Functional API で実装すると, 次の通り.

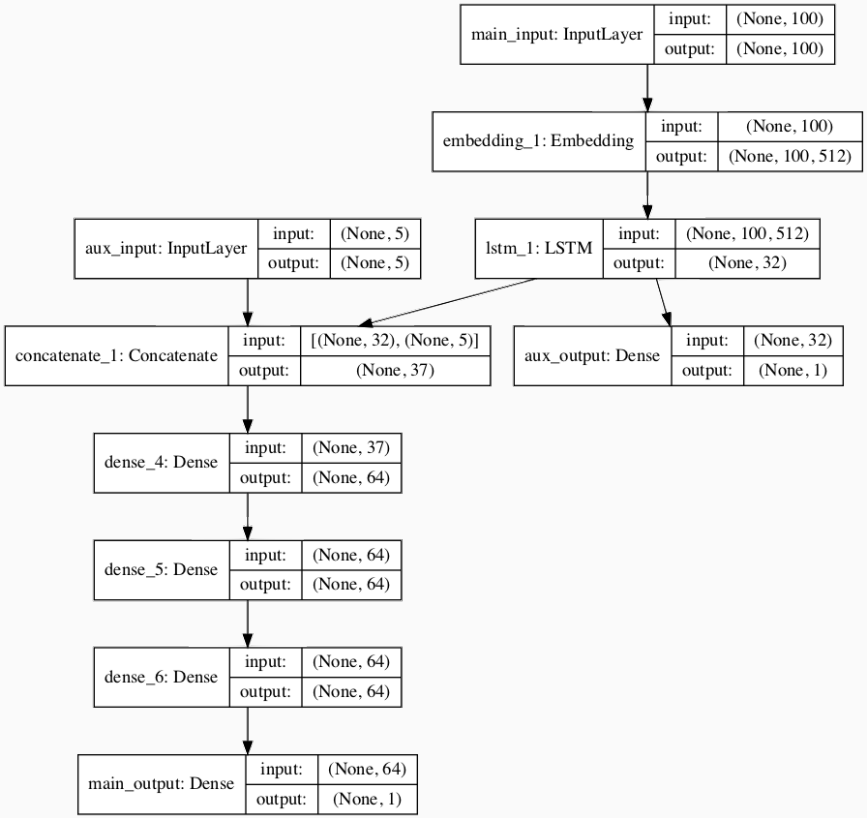
```
inputs = Input(shape=(28, 28, 1))
x = Conv2D(9, (3, 3), padding="same")(inputs)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Flatten()(x)
x = Dense(64, activation="relu")(x)
outputs = Dense(m, activation="softmax")(x)
model = Model(inputs=inputs, outputs=outputs)
```

Sequential や Model で作ったネットワークは, summary で確認出来る.
e.g., `model.summary()`

設計したモデル



(a) シンプルなネットワーク



(b) 複雑なネットワーク

keras.utils にある plot_model を使って，ネットワークを可視化できる [11]

[11] pydot をインストールしておく必要がある．なければ pip でインストールすれば良い．

モデルのコンパイル

ネットワークを定義したら，誤差関数やオプティマイザなどを指定する.

```
model.compile(
loss,     # 誤差関数:クロスエントロピーや二乗誤差など[12]
optimizer,     # パラメータの更新規則:sdg や adam など
metrics,     # 評価指標:分類精度 (accuracy) など[13]
)
```

[12] **クロスエントロピーを指定する場合**, categorical_crossentropy **を指定する**.
[13] **指定しない場合，誤差関数の推移のみ評価される**.

モデルのあてはめ

fit を用いてパラメータ推定 (重みは自動的に初期化される)

```
model.fit(
x_train, y_train,     # 訓練データ
batch_size,     # バッチサイズ[14]
epochs,     # エポック数
validation_data=(x_test, y_test)     # テストデータ
)
```

validation_data を指定しておくと, エポックごとに compile の metrics で指定した評価指標を計算してくれる. 誤差の推移をプロットするために, model.fit を呼ぶので, あらかじめ変数として保存しておくと良い.

e.g., `history = model.fit(hogehoge)`

`plt.plot(history)`

[14] パラメータの更新をサンプルごとではなく, 少数の塊 (バッチ) ごとに更新することができる.

モデルのあてはめ

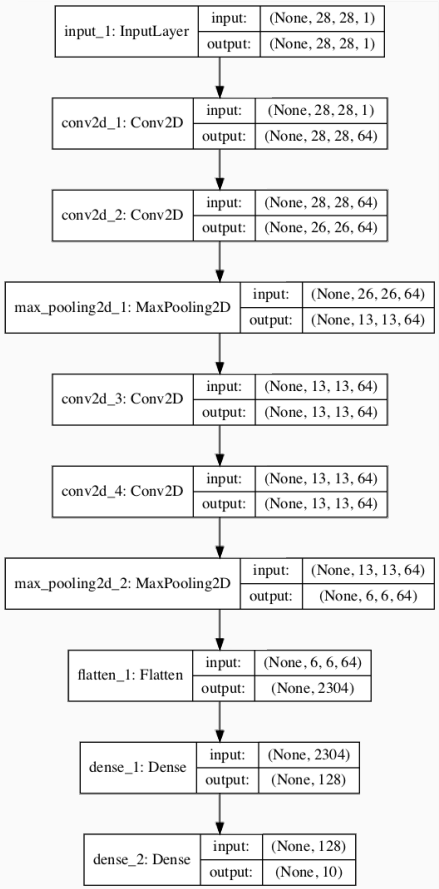
```
model.predict(x_test)
```

で, テストデータに対するモデルの出力 (ソフトマックス関数の値) が出力される. したがって, ネットワークの予測ラベルは

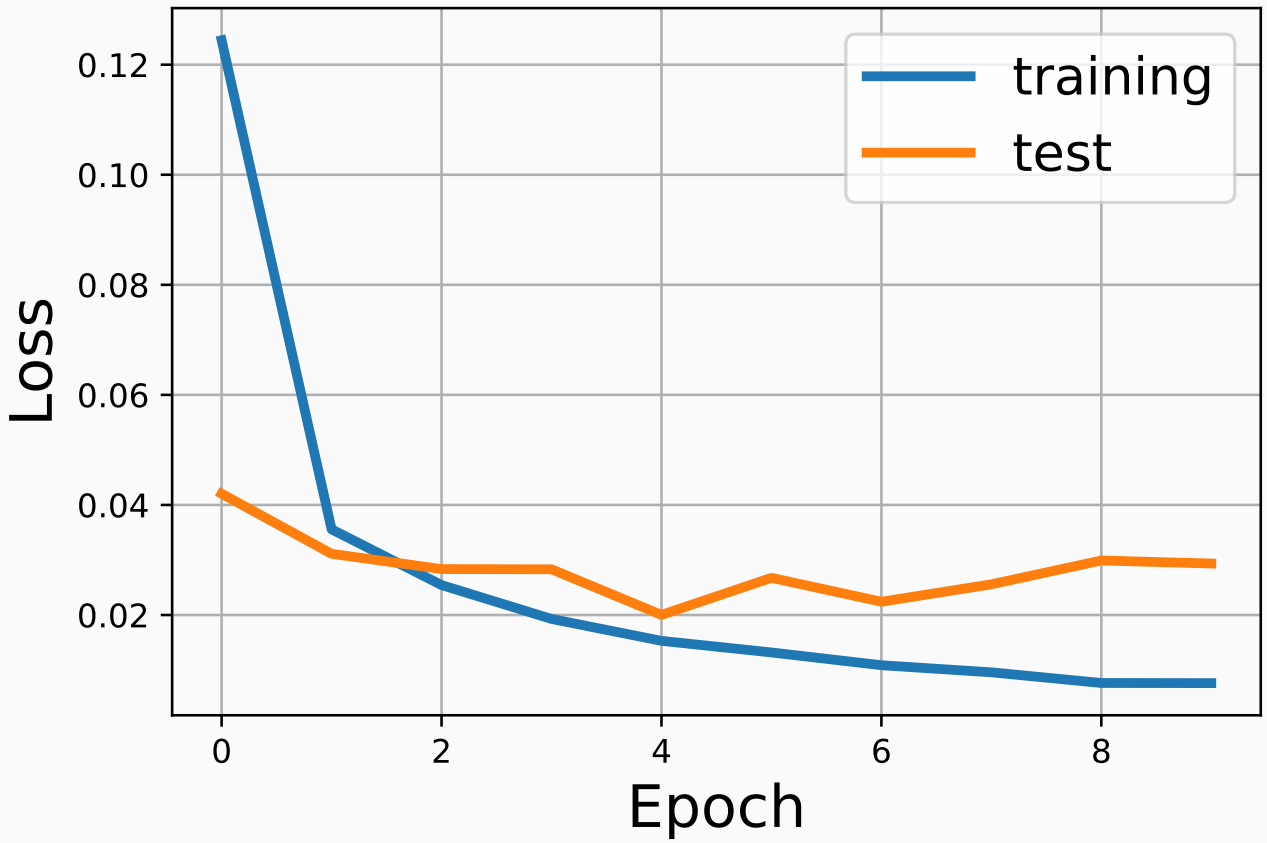
```
np.argmax(
model.predict(x_test)
)
```

で評価できる.

実行結果 I

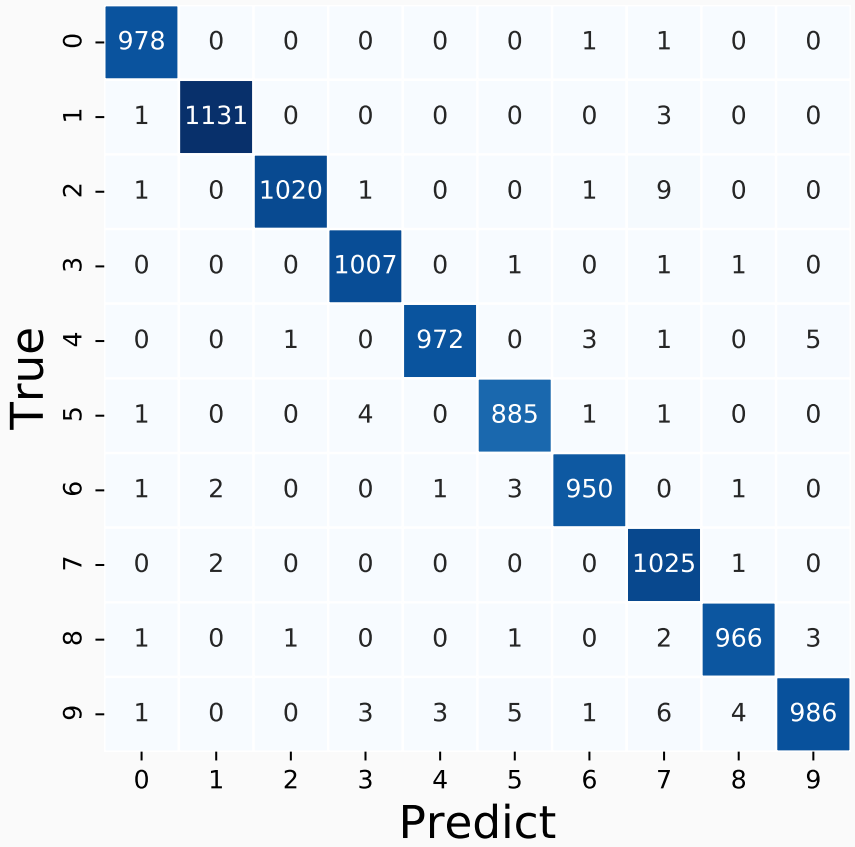


(c) ネットワーク



(d) 誤差関数の推移

実行結果 II



(e) confusion matrix

(f) 誤分類例