既存ライブラリの活用

前田利之（まえだ　としゆき）
aipr@e-chan.jp

阪南大学　経営情報学部

(2024.12.20)

既存のライブラリ

- デイープラーニングを行なうためのライブラリは様々なものがある
- この授業では [　　] のために１からコードをおこしていたが、（将来）業務等でディープラーニングを行なう場合は (GAFAM 等のトップエンジニアでない限り）既存のライブラリを活用するほうが、いろいろな意味で [　　] と思われる

既存のライブラリ

- デイープラーニングを行なうためのライブラリは様々なものがある
- この授業では **中身の学習** のために１からコードをおこしていたが、（将来）業務等でディープラーニングを行なう場合は (GAFAM 等のトップエンジニアでない限り）既存のライブラリを活用するほうが、いろいろな意味で [　　] と思われる

既存のライブラリ

- デイープラーニングを行なうためのライブラリは様々なものがある
- この授業では **中身の学習** のために１からコードをおこしていたが、（将来）業務等でディープラーニングを行なう場合は (GAFAM 等のトップエンジニアでない限り）既存のライブラリを活用するほうが、いろいろな意味で **効率が良い** と思われる

既存のライブラリ

-
- ,
-
- ,
-
- 、、等がある

既存のライブラリ

- TensorFlow(テンソルフロー)
- ,
-
- ,
-
- 、、等がある

既存のライブラリ

- TensorFlow(テンソルフロー)
- Keras(ケラス),
-
- ,
-
- 、、等がある

既存のライブラリ

- TensorFlow(テンソルフロー)
- Keras(ケラス),
- Chainer(チェイナー)
- ,
-
- 、、等がある

既存のライブラリ

- TensorFlow(テンソルフロー)
- Keras(ケラス),
- Chainer(チェイナー)
- Pytorch(パイトーチ),
-
- 、、等がある

既存のライブラリ

- TensorFlow(テンソルフロー)
- Keras(ケラス),
- Chainer(チェイナー)
- Pytorch(パイトーチ),
- Caffe(カフェ)
- 、、等がある

ライブラリ Tensorflow

- [ ] が開発しオープンソースで公開している、機械学習に用いるためのライブラリ
- 機械学習や数値解析、ニューラルネットワーク（ディープラーニング）に対応しており、[ ] と [ ] の各種サービスなどでも広く活用されている
- 元々、社内での使用のために [ ] チームによって開発された

ライブラリ Tensorflow

- Google が開発しオープンソースで公開している、機械学習に用いるためのライブラリ
- 機械学習や数値解析、ニューラルネットワーク（ディープラーニング）に対応しており、 と の各種サービスなどでも広く活用されている
- 元々、社内での使用のために チームによって開発された

ライブラリ Tensorflow

- Google が開発しオープンソースで公開している、機械学習に用いるためのライブラリ
- 機械学習や数値解析、ニューラルネットワーク（ディープラーニング）に対応しており、Google と [　　] の各種サービスなどでも広く活用されている
- 元々、社内での使用のために [　　] チームによって開発された

ライブラリ Tensorflow

- Google が開発しオープンソースで公開している、機械学習に用いるためのライブラリ
- 機械学習や数値解析、ニューラルネットワーク（ディープラーニング）に対応しており、Google と DeepMind の各種サービスなどでも広く活用されている
- 元々、社内での使用のためにチームによって開発された

ライブラリ Tensorflow

- Google が開発しオープンソースで公開している、機械学習に用いるためのライブラリ
- 機械学習や数値解析、ニューラルネットワーク（ディープラーニング）に対応しており、Google と DeepMind の各種サービスなどでも広く活用されている
- 元々、社内での使用のために Google Brain チームによって開発された

ライブラリ keras

- Python で書かれたニューラルネットワークのライブラリ
- [ ]や [ ]といった他のディープラーニングのライブラリの上で動くような作りになっていて、簡単に深層学習のネットワークを作ることができるよう設計されている
- プログラマにとって分かりやすい設計のため [ ]でも簡単に迅速なプロトタイピングが可能となっている

ライブラリ keras

- Python で書かれたニューラルネットワークのライブラリ
- TensorFlow や [　　] といった他のディープラーニングのライブラリの上で動くような作りになっていて、簡単に深層学習のネットワークを作ることができるよう設計されている
- プログラマにとって分かりやすい設計のため [　　] でも簡単に迅速なプロトタイピングが可能となっている

ライブラリ keras

- Python で書かれたニューラルネットワークのライブラリ
- TensorFlow や Theano といった他のディープラーニングのライブラリの上で動くような作りになっていて、簡単に深層学習のネットワークを作ることができるよう設計されている
- プログラマにとって分かりやすい設計のため [　　] でも簡単に迅速なプロトタイピングが可能となっている

ライブラリ keras

- Python で書かれたニューラルネットワークのライブラリ
- TensorFlow や Theano といった他のディープラーニングのライブラリの上で動くような作りになっていて、簡単に深層学習のネットワークを作ることができるよう設計されている
- プログラマにとって分かりやすい設計のため初心者でも簡単に迅速なプロトタイピングが可能となっている

ライブラリ keras の特徴

- 容易に素早く [　　] の作成が可能
- [　　] の両方，およびこれらの２つの組み合わせをサポート
- [　　] 上でシームレスな動作
- Google Colab. では Tensorflow とあわせて最初から導入済（例は次頁以降で）

ライブラリ keras の特徴

- 容易に素早く プロトタイプ の作成が可能
- [　　] の両方，およびこれらの２つの組み合わせをサポート
- [　　] 上でシームレスな動作
- Google Colab. では Tensorflow とあわせて最初から導入済（例は次頁以降で）

ライブラリ keras の特徴

- 容易に素早く プロトタイプ の作成が可能
- CNN と RNN の両方，およびこれらの２つの組み合わせをサポート
- 上でシームレスな動作
- Google Colab. では Tensorflow とあわせて最初から導入済（例は次頁以降で）

ライブラリ keras の特徴

- 容易に素早く **プロトタイプ** の作成が可能
- **CNN と RNN** の両方，およびこれらの２つの組み合わせをサポート
- **CPU と GPU** 上でシームレスな動作
- Google Colab. では Tensorflow とあわせて最初から導入済（例は次頁以降で）

keras の利用例（実装１）

1. まず、keras-mnist.ipynb (zip で圧縮済) のリンク先を Colab Notebooks の中 (mymodules の中ではない!) に保存し、展開しておく
2. Chrome または Edge を立ち上げ Google Drive からファイルをひらく
3. 「ライタイム」メニューの「ランタイムのタイプを変更」で GPU を選ぶ（元々なっている場合もあるが、確認する）
4. 実行して、結果を確認する

keras の利用例（実装１）

☆サンプルのコードは以下の通り

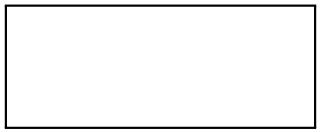
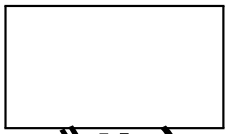
```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
from keras.utils.np_utils import to_categorical
# 画像を 1 次元化
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
# 画素を 0~1 の範囲に変換（正規化）
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
# 正解ラベルを one-hot-encoding
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

（…次頁に続く）

keras の利用例（実装１）（続き）

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=784))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train,
          batch_size=100, epochs=20, verbose=1)
score = model.evaluate(x_test, y_test)
print(score[0])
print(score[1])
```

補足: Dense

- 個々のパーセプトロンが次のレイヤーのすべてに繋がる  ニューラルネットワークレイヤー
- 単純なモデルで、このレイヤーはいくらでも  に追加する事が出来る（シーケンシャルモデル）
- ⇒ Dense はもっとも良く使われるレイヤー

補足: Dense

- 個々のパーセプトロンが次のレイヤーのすべてに繋がる 全結合 ニューラルネットワークレイヤー
- 単純なモデルで、このレイヤーはいくらでも [　　] に追加する事が出来る（シーケンシャルモデル）
- ⇒ Dense はもっとも良く使われるレイヤー

補足: Dense

- 個々のパーセプトロンが次のレイヤーのすべてに繋がる **全結合** ニューラルネットワークレイヤー
- 単純なモデルで、このレイヤーはいくらでも **直列** に追加する事が出来る（シーケンシャルモデル）
- ⇒ Dense はもっとも良く使われるレイヤー

keras の利用例（実装２）

1. まず、keras-ex.ipynb (zip で圧縮済) のリンク先を Colab Notebooks の中（ mymodules の中ではない！）に保存し、展開しておく
2. Chrome または Edge を立ち上げ Google Drive からファイルをひらく
3. 「ライタイム」メニューの「ランタイムのタイプを変更」で GPU を選ぶ（元々なっている場合もあるが、確認する）
4. ２分割されているので、先に上部の学習部を実行してから、下の結果表示部（グラフ描画等）を実行する

keras の利用例（実装２）

☆サンプルのコードは以下の通り

```
# パッケージのインポート
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.layers import Activation, Dense, \
     Dropout, Conv2D, Flatten, MaxPool2D
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

（…次頁に続く）

keras の利用例（実装２）（続き）

```
# データセットの準備
(train_images, train_labels), (test_images, test_labels) \
        = cifar10.load_data()
# データセットの画像の前処理
train_images = train_images.astype('float32')/255.0
test_images = test_images.astype('float32')/255.0
# データセットのラベルの前処理
train_labels = to_categorical(train_labels, 10)
test_labels = to_categorical(test_labels, 10)
#モデルの作成
model = Sequential()
```

（…次頁に続く）

keras の利用例（実装２）（続き）

```
# ここから、いくつかの層を add で追加していく
# Conv → Conv → Pool → Dropout
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', \
          input_shape=(32, 32, 3)))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
# Conv → Conv → Pool → Dropout
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

（…次頁に続く）

補足: Dropout

- 中間層などのニューロンを一定確率でランダムに選択し [ ] すること
- [ ] : 学習時に選択したニューロンは [ ] ものとして更新すること
- ディープラーニングによる [ ] を防ぐなどの目的で導入
- 学習した結果から予測を行う際には [ ] のニューロンを（出力が大きくならないよう補正した上で）使用

補足: Dropout

- 中間層などのニューロンを一定確率でランダムに選択し 非活性化 すること
- [　　] : 学習時に選択したニューロンは [　　] ものとして更新すること
- ディープラーニングによる [　　] を防ぐなどの目的で導入
- 学習した結果から予測を行う際には [　　] のニューロンを（出力が大きくならないよう補正した上で）使用

補足: Dropout

- 中間層などのニューロンを一定確率でランダムに選択し 非活性化 すること
- 非活性化 : 学習時に選択したニューロンは [　　] ものとして更新すること
- ディープラーニングによる [　　] を防ぐなどの目的で導入
- 学習した結果から予測を行う際には [　　] のニューロンを（出力が大きくならないよう補正した上で）使用

補足: Dropout

- 中間層などのニューロンを一定確率でランダムに選択し **非活性化** すること
- **非活性化** : 学習時に選択したニューロンは **存在しない** ものとして更新すること
- ディープラーニングによる [　　] を防ぐなどの目的で導入
- 学習した結果から予測を行う際には [　　] のニューロンを（出力が大きくならないよう補正した上で）使用

補足: Dropout

- 中間層などのニューロンを一定確率でランダムに選択し **非活性化** すること
- **非活性化** : 学習時に選択したニューロンは **存在しない** ものとして更新すること
- ディープラーニングによる **過学習** を防ぐなどの目的で導入
- 学習した結果から予測を行う際には [　　] のニューロンを（出力が大きくならないよう補正した上で）使用

補足: Dropout

- 中間層などのニューロンを一定確率でランダムに選択し **非活性化** すること
- **非活性化** : 学習時に選択したニューロンは **存在しない** ものとして更新すること
- ディープラーニングによる **過学習** を防ぐなどの目的で導入
- 学習した結果から予測を行う際には **全て** のニューロンを（出力が大きくならないよう補正した上で）使用

keras の利用例（実装２）（続き）

```
# Flatten → Dense → Dropout → Dense
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
# コンパイル
model.compile(loss='categorical_crossentropy', \
              optimizer=Adam(lr=0.001), metrics=['acc'])
# 学習
history = model.fit(train_images, train_labels,
    batch_size=128, epochs=50, validation_split=0.1)
# モデルの保存
model.save('convolution.h5')
```

（…次頁に続く）

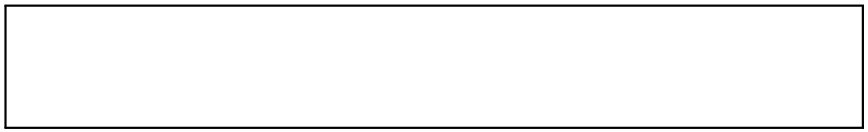
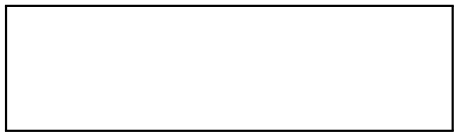
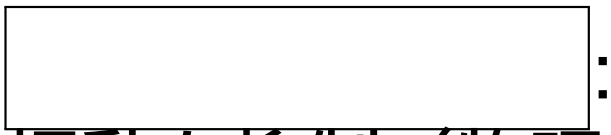
補足: Flatten

- ２次元以上の入力データを [　　] のデータに変換する
- 画像データはピクセル数要素の（１次元）データとなり次の層のニューロンへ送られる

補足: Flatten

- ２次元以上の入力データを **１次元** のデータに変換する
- 画像データはピクセル数要素の（１次元）データとなり次の層のニューロンへ送られる

補足: Adam

-  の１つ
- Adam =  + 
-  :  に移動平均を適用して、振動を抑制（物理学で言う「慣性」の導入に相当）
-  :ランダムに取り出したデータを用いて、パラメータ（重み）を更新していく勾配降下法。データ１つだけをサンプルして使うことで、最急降下法にランダム性を入れる

補足: Adam

- 最適化アルゴリズムの１つ
- Adam = [ ] + [ ]
- [ ]: [ ] に移動平均を適用して、振動を抑制（物理学で言う「慣性」の導入に相当）
- [ ]:ランダムに取り出したデータを用いて、パラメータ（重み）を更新していく勾配降下法。データ１つだけをサンプルして使うことで、最急降下法にランダム性を入れる

補足: Adam

- 最適化アルゴリズムの１つ
- Adam = モーメンタム +
- : に移動平均を適用して、振動を抑制（物理学で言う「慣性」の導入に相当）
- :ランダムに取り出したデータを用いて、パラメータ（重み）を更新していく勾配降下法。データ１つだけをサンプルして使うことで、最急降下法にランダム性を入れる

補足: Adam

- 最適化アルゴリズムの１つ
- Adam = モーメンタム + RMSProp
- : に移動平均を適用して、振動を抑制（物理学で言う「慣性」の導入に相当）
- :ランダムに取り出したデータを用いて、パラメータ（重み）を更新していく勾配降下法。データ１つだけをサンプルして使うことで、最急降下法にランダム性を入れる

補足: Adam

- 最適化アルゴリズムの１つ
- Adam = モーメンタム + RMSProp
- モーメンタム: 　に移動平均を適用して、振動を抑制（物理学で言う「慣性」の導入に相当）
- 　:ランダムに取り出したデータを用いて、パラメータ（重み）を更新していく勾配降下法。データ１つだけをサンプルして使うことで、最急降下法にランダム性を入れる

補足: Adam

- 最適化アルゴリズムの１つ
- Adam = モーメンタム + RMSProp
- モーメンタム: SGD に移動平均を適用して、振動を抑制（物理学で言う「慣性」の導入に相当）
- :ランダムに取り出したデータを用いて、パラメータ（重み）を更新していく勾配降下法。データ１つだけをサンプルして使うことで、最急降下法にランダム性を入れる

補足: Adam

- 最適化アルゴリズムの１つ
- Adam = モーメンタム + RMSProp
- モーメンタム: SGD に移動平均を適用して、振動を抑制（物理学で言う「慣性」の導入に相当）
- SGD:ランダムに取り出したデータを用いて、パラメータ（重み）を更新していく勾配降下法。データ１つだけをサンプルして使うことで、最急降下法にランダム性を入れる

補足: Adam

- [ ]: [ ] に移動平均を導入する。勾配の大きさに応じて学習率を調整するようにして、振動を抑制
- [ ]: 学習が進むにつれて、学習係数を小さくしていくアルゴリズム
  - パラメータごとに見かけの学習率を設定できる
  - 学習が進むにつれて、見かけの学習率は単調減衰していく

補足: Adam

- RMSProp: [　　] に移動平均を導入する。勾配の大きさに応じて学習率を調整するようにして、振動を抑制
- [　　]: 学習が進むにつれて、学習係数を小さくしていくアルゴリズム
  - パラメータごとに見かけの学習率を設定できる
  - 学習が進むにつれて、見かけの学習率は単調減衰していく

補足: Adam

- RMSProp: AdaGrad に移動平均を導入する。勾配の大きさに応じて学習率を調整するようにして、振動を抑制
- : 学習が進むにつれて、学習係数を小さくしていくアルゴリズム
  - パラメータごとに見かけの学習率を設定できる
  - 学習が進むにつれて、見かけの学習率は単調減衰していく

補足: Adam

- RMSProp: AdaGrad に移動平均を導入する。勾配の大きさに応じて学習率を調整するようにして、振動を抑制
- AdaGrad: 学習が進むにつれて、学習係数を小さくしていくアルゴリズム
  - パラメータごとに見かけの学習率を設定できる
  - 学習が進むにつれて、見かけの学習率は単調減衰していく

```
# グラフの表示
plt.plot(history.history['acc'], label='acc')
plt.plot(history.history['val_acc'], label='val_acc')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(loc='best')
plt.show()
# 評価
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('loss: {:.3f}\nacc: {:.3f}'.format(test_loss, test_acc ))
```

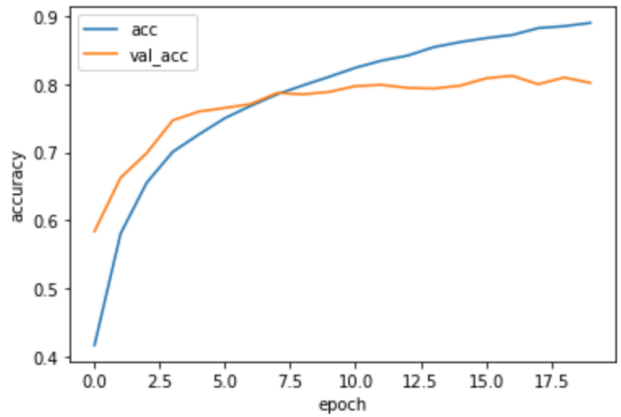
（…次頁に続く）

keras の利用例（実装２）（続き）

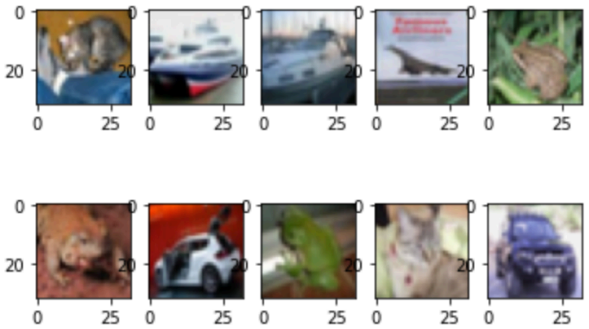
```
# 推論する画像の表示
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(test_images[i])
plt.show()
# 推論したラベルの表示
test_predictions = model.predict(test_images[0:10])
test_predictions = np.argmax(test_predictions, axis=1)
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer',
        'dog', 'frog', 'horse', 'ship', 'truck']
print([labels[n] for n in test_predictions])
```

keras の利用例（実行結果）

☆ cifar-10 (MNIST 同様、機械学習用データとして良く使われるカラー画像集）について約８割の認識率となる




```
313/313 [==============================] - 1s 3ms/step - loss: 0.6997 - acc: 0.7896
loss: 0.700
acc: 0.790
```



```
['dog', 'ship', 'ship', 'ship', 'frog', 'frog', 'automobile', 'bird', 'cat', 'automobile']
```

おしまい

質問・コメント等あれば、何でもお気軽に
aipr@e-chan.jp に
メールください！