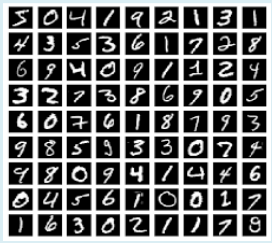
大阪府立高津高等学校　情報班

AIによる画像認識の基礎的理論

1:要旨

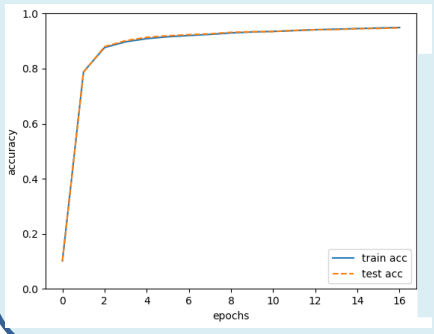
AIブームが到来して画像認識に関する研究が進んだ。そこで、どのような理論で画像を判定するのかを基礎的な部分を実装しながら理解する。使用する言語はPython。

検証に用いる画像データ



左図はMNISTの一部。訓練用とテスト用とで7万枚の手書き数字を収容する。

検証結果

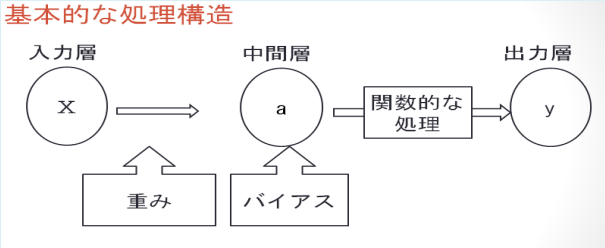


時間が経つにつれて訓練データと教師データが 1 に近づいている。

今後の展望

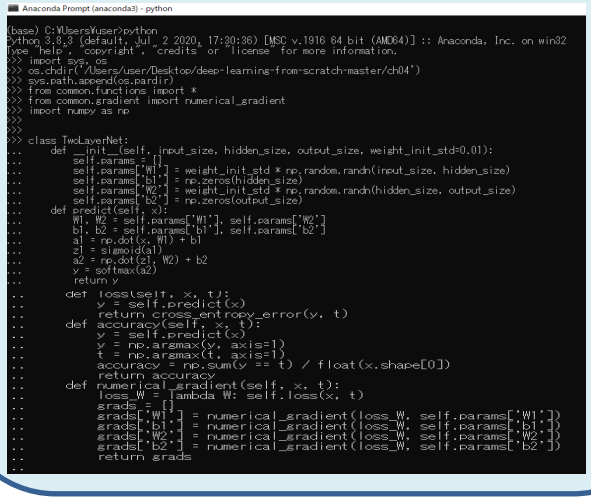
今回用いたニューラルネットワークという手法にこだわらず、HOGとKNNを使った手法にも挑戦したい。

3:検証の行程



入力した数字に重みを乗算してバイアス（足かせ）を加算する。バイアスは関数的な処理をする際の変数の値を調整する役割を果たす。
重みとバイアスの二つを少しずつ変えながら最適な値を見つける。

プログラミング本文



```
Anaconda Prompt (anaconda3) - python

(base) C:¥Users¥user>python
Python 3.8.3 (default, Jul  2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys, os
>>> os.chdir('/Users/user/Desktop/deep-learning-from-scratch-master/ch04')
>>> sys.path.append(os.pardir)
>>> from common.functions import *
>>> from common.gradient import numerical_gradient
>>> import numpy as np
>>>
>>>
>>> class TwoLayerNet:
...     def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
...         self.params = {}
...         self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
...         self.params['b1'] = np.zeros(hidden_size)
...         self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
...         self.params['b2'] = np.zeros(output_size)
...     def predict(self, x):
...         W1, W2 = self.params['W1'], self.params['W2']
...         b1, b2 = self.params['b1'], self.params['b2']
...         a1 = np.dot(x, W1) + b1
...         z1 = sigmoid(a1)
...         a2 = np.dot(z1, W2) + b2
...         y = softmax(a2)
...         return y
...
...     def loss(self, x, t):
...         y = self.predict(x)
...         return cross_entropy_error(y, t)
...     def accuracy(self, x, t):
...         y = self.predict(x)
...         y = np.argmax(y, axis=1)
...         t = np.argmax(t, axis=1)
...         accuracy = np.sum(y == t) / float(x.shape[0])
...         return accuracy
...     def numerical_gradient(self, x, t):
...         loss_W = lambda W: self.loss(x, t)
...         grads = {}
...         grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
...         grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
...         grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
...         grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
...         return grads
...
```

参考文献

ゼロから始めるDeep Learning―Pythonで学ぶディープラーニングの理論と実装　斎藤康毅 O'REILLY 発行