Chapter 4

5,June,2020 藤本

学習アルゴリズム

- 確率的勾配降下法 (stochastic gradient descent)

ステップ 1（ミニバッチ）

訓練データの中からランダムに一部のデータを選び出す。その選ばれたデータをミニバッチと言い、ここでは、そのミニバッチの損失関数の値を減らすことを目的とする。

ステップ 2（勾配の算出）

ミニバッチの損失関数を減らすために、各重みパラメータの勾配を求める。勾配は、損失関数の値を最も減らす方向を示す。

ステップ 3（パラメータの更新）

重みパラメータを勾配方向に微小量だけ更新する。

ステップ 4（繰り返す）

ステップ 1、ステップ 2、ステップ 3 を繰り返す。

実装！

- 手書きの数字を学習する2層NN

  ch04/two_layer_net.pyにあります

```python
# coding: utf-8
import sys, os
sys.path.append(os.pardir)  # 親ディレクトリのファイルをインポートするための設定
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):
        # 重みの初期化
        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
```

$$y = wx + b$$

変数	説明
params	ニューラルネットワークのパラメータを保持するディクショナリ変数（インスタンス変数）。 params['W1'] は 1 層目の重み、params['b1'] は 1 層目のバイアス。 params['W2'] は 2 層目の重み、params['b2'] は 2 層目のバイアス。
grads	勾配を保持するディクショナリ変数（numerical_gradient() メソッドの返り値）。 grads['W1'] は 1 層目の重みの勾配、grads['b1'] は 1 層目のバイアスの勾配。 grads['W2'] は 2 層目の重みの勾配、grads['b2'] は 2 層目のバイアスの勾配。

実装！

- 手書きの数字を学習する2層NN

```
18
19      def predict(self, x):
20          W1, W2 = self.params['W1'], self.params['W2']
21          b1, b2 = self.params['b1'], self.params['b2']
22
23          a1 = np.dot(x, W1) + b1
24          z1 = sigmoid(a1)
25          a2 = np.dot(z1, W2) + b2
26          y = softmax(a2)
27
28          return y
29
30      # x:入力データ, t:教師データ
31      def loss(self, x, t):
32          y = self.predict(x)
33
34          return cross_entropy_error(y, t)
35
36      def accuracy(self, x, t):
37          y = self.predict(x)
38          y = np.argmax(y, axis=1)
39          t = np.argmax(t, axis=1)
40
41          accuracy = np.sum(y == t) / float(x.shape[0])
42          return accuracy
43
```

推論処理

実装！

- 手書きの数字を学習する2層NN

```
43
44      # x:入力データ, t:教師データ
45      def numerical_gradient(self, x, t):
46          loss_W = lambda W: self.loss(x, t)
47
48          grads = {}
49          grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
50          grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
51          grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
52          grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
53
54          return grads
55
56      def gradient(self, x, t):
57          W1, W2 = self.params['W1'], self.params['W2']
58          b1, b2 = self.params['b1'], self.params['b2']
59          grads = {}
60
61          batch_num = x.shape[0]
62
63          # forward
64          a1 = np.dot(x, W1) + b1
65          z1 = sigmoid(a1)
66          a2 = np.dot(z1, W2) + b2
67          y = softmax(a2)
68
69          # backward
70          dy = (y - t) / batch_num
71          grads['W2'] = np.dot(z1.T, dy)
72          grads['b2'] = np.sum(dy, axis=0)
73
74          dz1 = np.dot(dy, W2.T)
75          da1 = sigmoid_grad(a1) * dz1
76          grads['W1'] = np.dot(x.T, da1)
77          grads['b1'] = np.sum(da1, axis=0)
78
79          return grads
```

変数	説明
params	ニューラルネットワークのパラメータを保持するディクショナリ変数（インスタンス変数）。 params['W1'] は 1 層目の重み、params['b1'] は 1 層目のバイアス。 params['W2'] は 2 層目の重み、params['b2'] は 2 層目のバイアス。
grads	勾配を保持するディクショナリ変数（numerical_gradient() メソッドの返り値）。 grads['W1'] は 1 層目の重みの勾配、grads['b1'] は 1 層目のバイアスの勾配。 grads['W2'] は 2 層目の重みの勾配、grads['b2'] は 2 層目のバイアスの勾配。

実装！

- 手書きの数字を学習する2層NN

```
10
11      def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
12          # 重みの初期化
13          self.params = {}
14          self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
15          self.params['b1'] = np.zeros(hidden_size)
16          self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
17          self.params['b2'] = np.zeros(output_size)
```

まず初期化
入力画像が28x28 = 784
出力は10個のクラス

wはgaussianで振っていて、bは0で初期化

実装！

- 手書きの数字を学習する2層NN

```
29
30      # x:入力データ, t:教師データ
31      def loss(self, x, t):
32          y = self.predict(x)
33
34          return cross_entropy_error(y, t)
35
36      def accuracy(self, x, t):
37          y = self.predict(x)
38          y = np.argmax(y, axis=1)
39          t = np.argmax(t, axis=1)
40
41          accuracy = np.sum(y == t) / float(x.shape[0])
42          return accuracy
43
44      # x:入力データ, t:教師データ
45      def numerical_gradient(self, x, t):
46          loss_W = lambda W: self.loss(x, t)
47
48          grads = {}
49          grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
50          grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
51          grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
52          grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
53
54          return grads
55
56      def gradient(self, x, t):
57          W1, W2 = self.params['W1'], self.params['W2']
58          b1, b2 = self.params['b1'], self.params['b2']
59          grads = {}
60
61          batch_num = x.shape[0]
62
```

実装！

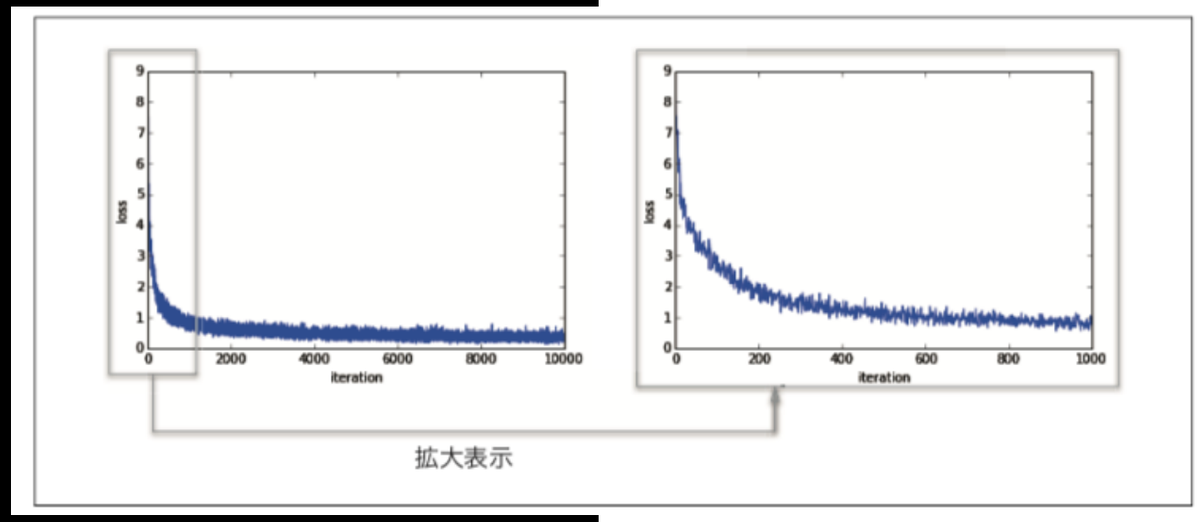
- ミニバッチ学習する。
  ch04/train_neuralnet.pyにあります。
- ミニバッチのサイズ100

```python
# coding: utf-8
import sys, os
sys.path.append(os.pardir)  # 親ディレクトリのファイルをインポートするための設定
import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from two_layer_net import TwoLayerNet

# データの読み込み
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

iters_num = 10000  # 繰り返しの回数を適宜設定する
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

iter_per_epoch = max(train_size / batch_size, 1)
```

実装！

- ミニバッチ学習する。
  ch04/train_neuralnet.pyにあります。

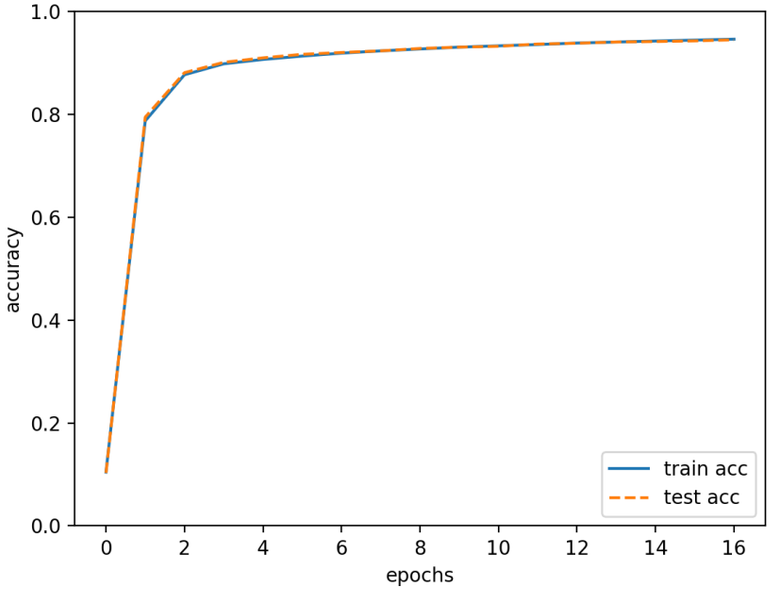
```
24
25 for i in range(iters_num):
26     batch_mask = np.random.choice(train_size, batch_size)
27     x_batch = x_train[batch_mask]
28     t_batch = t_train[batch_mask]
29
30     # 勾配の計算
31     #grad = network.numerical_gradient(x_batch, t_batch)
32     grad = network.gradient(x_batch, t_batch)
33
34     # パラメータの更新
35     for key in ('W1', 'b1', 'W2', 'b2'):
36         network.params[key] -= learning_rate * grad[key]
37
38     loss = network.loss(x_batch, t_batch)
39     train_loss_list.append(loss)
40
41     if i % iter_per_epoch == 0:
42         train_acc = network.accuracy(x_train, t_train)
43         test_acc = network.accuracy(x_test, t_test)
44         train_acc_list.append(train_acc)
45         test_acc_list.append(test_acc)
46         print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))
```



評価

- テストデータで定期的に評価する。
  訓練データとテストデータを対象に認識精度を記録

```
22
23 iter_per_epoch = max(train_size / batch_size, 1)
24
41     if i % iter_per_epoch == 0:
42         train_acc = network.accuracy(x_train, t_train)
43         test_acc = network.accuracy(x_test, t_test)
44         train_acc_list.append(train_acc)
45         test_acc_list.append(test_acc)
46         print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))
```



2本の線が重なっている
＝過学習されていない