

深層学習 day2 実装演習キャプチャ 2

1 2_5_overfitting

The screenshot shows a Google Colab notebook titled "2_5_overfitting.ipynb". The code cell at the top contains:

```
# Google Colab のマイドライブを基準でDNN_code/colab/day2フォルダを置くことを促進しています。必要に応じて、パスを変更してください。
! Google Colab での実行か確認する
# フォルダの作成
3 import os
4 if 'BM_COLAB' in os.environ else False
5
6 # Google Drive のマウント
7 ! mount --bind /content/drive/MyDrive/colab/notebooks/notebook/ /content/drive/MyDrive/colab/notebooks/notebook/
8
9 from google.colab import drive
10 drive.mount('/content/drive')
11 os.chdir('/content/drive/notebook')
```

The code cell below it is expanded, showing the definition of `sys.path`:

sys.pathの設定

```
1 import sys
2 sys.path.append(os.getcwd()) # 構ディレクトリのファイルをインポートするための設定
```

The status bar at the bottom indicates "20 秒 実行時間 20.38".



```

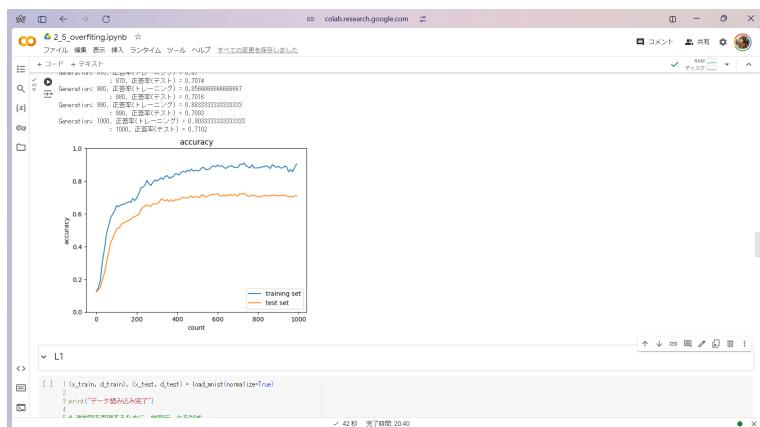
2.5_overfitting.ipynb
+ コード + テスト
+ ファイル 嵌入 素材 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
コメント RAM ディスク ▾
weight decay
L2
(x)
@x
1 from common import optimizer
2
3 (x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)
4 print("データ読み込み完了")
5
6 # 過学習を防ぐために、学習データを削減
7 x_train = x_train[100]
8 d_train = d_train[100]
9
10
11 network = MultiLayerNet(input_size=74, hidden_size_list=[100, 100, 100, 100, 100], output_size=10)
12
13 iter_size = 1000
14 train_size = x_train.shape[0]
15 batch_size = 100
16 learning_rate=0.01
17
18 train_loss_list = []
19 accuraccy_train = []
20 accuraccy_val = []
21
22 plot_interval = 100
23 network_name = network.hidden_layer[0]
24
25 # 重みの初期化
26 weight_decay = 0.1
27
28 for i in range(iter_size):
29     batch_idx = np.random.choice(train_size, batch_size)
30     x_batch = x_train[batch_idx]
31     d_batch = d_train[batch_idx]
32
33     grad = network.gradient(x_batch, d_batch)
34
35     weight_decay = 0
36
37     for id in range(1, len(network)):
38         grad[id] += weight_decay * network.params['W' + str(id)]
39
40         for id in range(1, len(network)):
41             grad['W' + str(id)] = network.layers[id].grad['W' + str(id)] + weight_decay * grads['W' + str(id)]
42             grad['b' + str(id)] = network.layers[id].grad['b' + str(id)] + weight_decay * grads['b' + str(id)]
43             network.params['W' + str(id)] += learning_rate * grad['W' + str(id)]
44             network.params['b' + str(id)] += learning_rate * grad['b' + str(id)]
45
46     loss = network.loss(x_batch, d_batch) + weight_decay * sum([np.sum(network_params['W' + str(id)]) ** 2 for id in range(1, len(network))])
47     train_loss_list.append(loss)
48
49     if (i+1) % plot_interval == 0:
50         accr_train = network.accuracy(x_train, d_train)
51         accr_val = network.accuracy(x_test, d_test)
52         accuraccy_train.append(accr_train)
53         accuraccy_val.append(accr_val)
54
55         print('Generation : ' + str(i+1) + ', 正答率(トレーニング) : ' + str(accr_train))
56         print('Generation : ' + str(i+1) + ', 正答率(テスト) : ' + str(accr_val))
57
58         lists = neural01_stemcurve.plot_interval()
59         plt.plot(lists['accuracy_train'], label='training set')
60         plt.plot(lists['accuracy_val'], label='test set')
61         plt.legend(loc='lower right')
62         plt.title('accuracy')
63         plt.xlabel('count')
64         plt.ylabel('accuracy')
65         plt.show()
66
67 # クラスの数
68
69

```

```

2.5_overfitting.ipynb
+ コード + テスト
+ ファイル 嵌入 素材 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
コメント RAM ディスク ▾
(x)
@x
1 for i in range(1000):
2     batch_idx = np.random.choice(train_size, batch_size)
3     x_batch = x_train[batch_idx]
4     d_batch = d_train[batch_idx]
5
6     grad = network.gradient(x_batch, d_batch)
7
8     weight_decay = 0
9
10    for id in range(1, len(network)):
11        grad[id] += weight_decay * network_params['W' + str(id)]
12
13        for id in range(1, len(network)):
14            grad['W' + str(id)] = network.layers[id].grad['W' + str(id)] + weight_decay * grads['W' + str(id)]
15            grad['b' + str(id)] = network.layers[id].grad['b' + str(id)] + weight_decay * grads['b' + str(id)]
16            network_params['W' + str(id)] += learning_rate * grad['W' + str(id)]
17            network_params['b' + str(id)] += learning_rate * grad['b' + str(id)]
18
19    loss = network.loss(x_batch, d_batch) + weight_decay * sum([np.sum(network_params['W' + str(id)]) ** 2 for id in range(1, len(network))])
20    train_loss_list.append(loss)
21
22    if (i+1) % plot_interval == 0:
23        accr_train = network.accuracy(x_train, d_train)
24        accr_val = network.accuracy(x_test, d_test)
25        accuraccy_train.append(accr_train)
26        accuraccy_val.append(accr_val)
27
28        print('Generation : ' + str(i+1) + ', 正答率(トレーニング) : ' + str(accr_train))
29        print('Generation : ' + str(i+1) + ', 正答率(テスト) : ' + str(accr_val))
30
31
32    lists = neural01_stemcurve.plot_interval()
33    plt.plot(lists['accuracy_train'], label='training set')
34    plt.plot(lists['accuracy_val'], label='test set')
35    plt.legend(loc='lower right')
36    plt.title('accuracy')
37    plt.xlabel('count')
38    plt.ylabel('accuracy')
39    plt.show()
40
41 # クラスの数
42
43

```





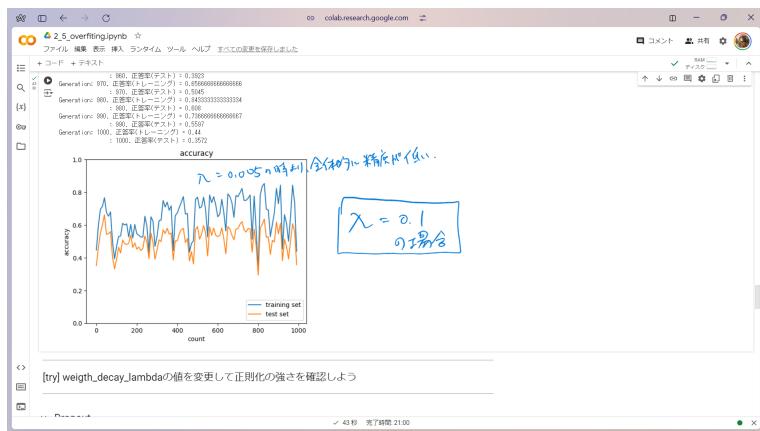
```
# 2.5_overfitting.pypt
# ファイル データ 表示 拡張子 ランタイム ツール ヘルプ
+ コード + テキスト
搜尋 L1
1 # (x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)
2
3 print("データ読み込み完了")
4
5 # 通常学習と同様のために、学習データを削減
6 x_train = x_train[0:100]
7 d_train = d_train[0:100]
8
9 network = MultiLayerNet(neuron_size=74, hidden_size_list=[100, 100, 100, 100, 100], output_size=10)
10
11 learning_rate = 0.001
12 batch_size = 10
13 train_size = len(x_train)
14 iters_num = int(train_size / batch_size)
15 learning_rate_t = learning_rate
16
17 train_loss_list = []
18 accuracy_train = []
19 accuracy_val = []
20
21 plot_interval = 10
22 hidden_layer_num = network.hidden_layer_num
23
24 # 正解の表示確認#####
25 min_pct_accuracy = 0.05
26
27
28 for i in range(iters_num):
29     batch_size = i % batch_size
30     batch_x = x_train[batch_size * 10:(batch_size + 1) * 10]
31     batch_d = d_train[batch_size * 10:(batch_size + 1) * 10]
32
33     x_batch = np.reshape(batch_x, (batch_size, 1, 28, 28))
34     y_batch = np.reshape(batch_d, (batch_size, 1))
35
36     network.forward(x_batch)
37     network.backward(y_batch)
38     weight_update = 0
39
40     for id in range(hidden_layer_num):
41         hidden_layer_num =
```



```

2.5_overfitting.ipynb
1 (x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)
2 print("データ読み込み完了")
3 # ディレクトリを確認
4 x_train = x_train[100]
5 d_train = d_train[100]
6 network = MultiLayerPerceptron(input_size=74, hidden_size=[100, 100, 100, 100, 100], output_size=10)
7
8 for i in range(1000):
9     train_size = 100
10    batch_size = 100
11    train_loss, loss = []
12    accuracetrain = []
13    accuracetest = []
14    for id in range(1000):
15        grad = network.gradient(x_train, d_train)
16        weight_decay = 0
17        for idx in range(hidden_layer_num):
18            grad[id][idx] = network.layers[idx].weight * weight_decay / 1000 + network.params[id] * str(id)
19            grad[id][idx] += network.layers[idx].backward(x_train, d_train, batch_size)
20    plot_interval(10)
21    hidden_layer_num = network.hidden_layer_num
22    for i in range(1000):
23        if i % 100 == 0:
24            f = open('accuracy.txt', 'w')
25            f.write(str(accuracy))
26            f.close()
27
28    for i in range(1000):
29        batch_idx = np.random.choice(train_size, batch_size)
30        d_batch = d_train[batch_idx]
31        x_batch = x_train[batch_idx]
32        grad = network.gradient(x_batch, d_batch)
33        weight_decay = 0
34        for idx in range(hidden_layer_num):
35            grad[idx][id] = network.layers[idx].weight * weight_decay / 1000 + network.params[id] * str(id)
36            grad[idx][id] += network.layers[idx].backward(x_batch, d_batch, batch_size)
37
38    for i in range(1000):
39        grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
40        grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)
41
42    for i in range(1000):
43        grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
44        grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)
45
46    for i in range(1000):
47        grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
48        grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)
49
50    for i in range(1000):
51        grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
52        grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)
53
54    for i in range(1000):
55        grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
56        grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)
57
58    for i in range(1000):
59        grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
60        grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)

```

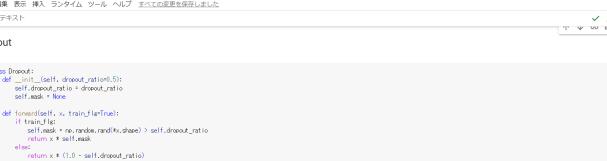


```

2.5_overfitting.ipynb
1 (x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)
2 print("データ読み込み完了")
3 # ディレクトリを確認
4 x_train = x_train[100]
5 d_train = d_train[100]
6 network = MultiLayerPerceptron(input_size=74, hidden_size=[100, 100, 100, 100, 100], output_size=10)
7
8 for i in range(1000):
9     train_size = 100
10    batch_size = 100
11    train_loss, loss = []
12    accuracetrain = []
13    accuracetest = []
14    learning_rate=0.1
15    train_loss_list = []
16    loss_list = []
17    accuracetrain_list = []
18    accuracetest_list = []
19    for id in range(hidden_layer_num):
20        grad = network.gradient(x_train, d_train)
21        weight_decay = 0
22        for i in range(1000):
23            batch_idx = np.random.choice(train_size, batch_size)
24            d_batch = d_train[batch_idx]
25            x_batch = x_train[batch_idx]
26            grad = network.gradient(x_batch, d_batch)
27            weight_decay = 0
28            for idx in range(hidden_layer_num):
29                grad[idx][id] = network.layers[idx].weight * weight_decay / 1000 + network.params[id] * str(id)
30                grad[idx][id] += network.layers[idx].backward(x_batch, d_batch, batch_size)
31            for i in range(1000):
32                grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
33                grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)
34            for i in range(1000):
35                grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
36                grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)
37            for i in range(1000):
38                grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
39                grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)
40            for i in range(1000):
41                grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
42                grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)
43            for i in range(1000):
44                grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
45                grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)
46            for i in range(1000):
47                grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
48                grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)
49            for i in range(1000):
50                grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
51                grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)
52            for i in range(1000):
53                grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
54                grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)
55            for i in range(1000):
56                grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
57                grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)
58            for i in range(1000):
59                grad[i][id] = network.layers[i].weight * weight_decay / 1000 + network.params[i] * str(i)
60                grad[i][id] += network.layers[i].backward(x_train, d_train, batch_size)

```





The screenshot shows a Google Colab notebook titled "2_5_overfitting.ipynb". The code implements a Dropout layer. It includes imports for numpy, tensorflow, and tensorflow.keras, and defines a Dropout class with methods for training and testing. The training method uses a random mask to drop units based on a specified ratio. The testing method applies the same mask to all units. The code also includes a check function to verify the implementation against a baseline.

```
class Dropout(layers.Layer):
    def __init__(self, dropout_ratio=0.5):
        super().__init__()
        self.dropout_ratio = dropout_ratio
        self.trainable = True

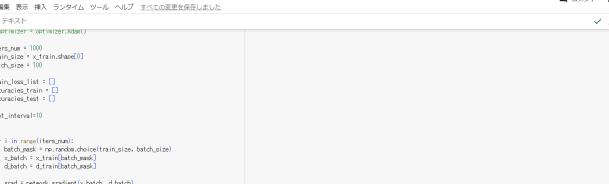
    def _create_dropout(self, x, train=False):
        if train:
            self.mask = np.random.rand(*x.shape) > self.dropout_ratio
            return x * self.mask
        else:
            return x * (1.0 - self.dropout_ratio)

    def call(self, inputs, training=False):
        return self._create_dropout(inputs, training)

# from common import optimizers
# (x_train, y_train), (x_test, y_test) = load_mnist(normalize=True)
# print("データ読み込み終了")

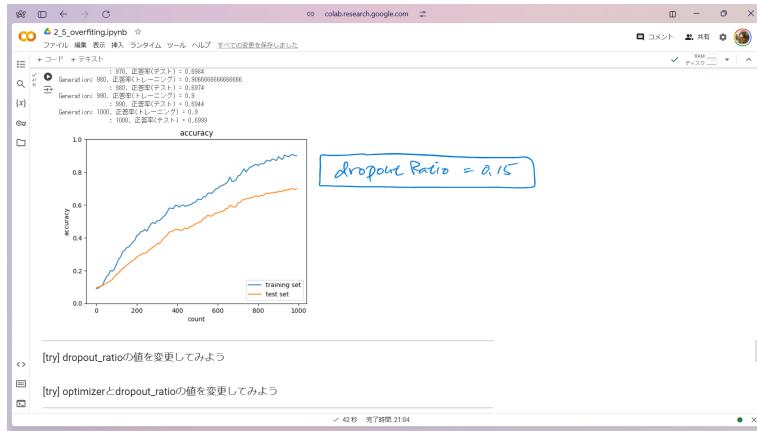
# モデル定義と訓練
# (x_train, y_train) = load_mnist()
# x_train = x_train[0:1000]
# y_train = y_train[0:1000]

# ⑩ ドロップアウト実装
# ⑪ use_dropout = True
# dropout_rate = 0.15
# ⑫ ⑬ ⑭ ⑮ ⑯ ⑰ ⑱ ⑲ ⑳
```



```
2.5_overfitting.ipynb の内容
```

```
# ファイル ディレクトリ ヘルプ ライタイム サーチ ヘルプ すべての変更を保存しました
+ コード + テキスト
20 import tensorflow as tf
21 from tensorflow import keras
22 HIDDEN_SIZE = 1000
23 TRAIN_SIZE = 10000
24 BATCH_SIZE = 100
25
26 train_loss_hist = []
27 accuracies_train = []
28 accuracies_val = []
29
30 plot_interval = 10
31
32
33 for i in range(TRAIN_SIZE):
34     batch_index = np.random.choice(train_size, batch_size)
35     batch_x = x_train[batch_index]
36     batch_y = y_train[batch_index]
37
38     grad = network.trainable_variables[0].grad
39     optimizer.update(network_params, grad)
40
41     loss = network.loss(batch_x, batch_y)
42     train_loss_hist.append(loss)
43
44     if (i+1) % plot_interval == 0:
45         accuracy_train = network.accuracy(x_train, y_train)
46         accuracy_val = network.accuracy(x_val, y_val)
47         accuracies_train.append(accuracy_train)
48         accuracies_val.append(accuracy_val)
49
50         print(f'iteration : {str(i+1)} | 正解率(トレーニング) : {str(accuracy_train)}')
51         print(f'iteration : {str(i+1)} | 正解率(テスト) : {str(accuracy_val)}')
52
53 hist = network.history(HIDDEN_SIZE, plot_interval)
54 plt.plot(hist['loss'], hist['accuracy'], label='training set')
55 plt.plot(hist['val_loss'], hist['val_accuracy'], label='test set')
56 plt.legend(loc='lower right')
57 plt.xlabel('epoch')
58 plt.ylabel('accuracy')
59 plt.title('accuracy')
60 plt.show()
```



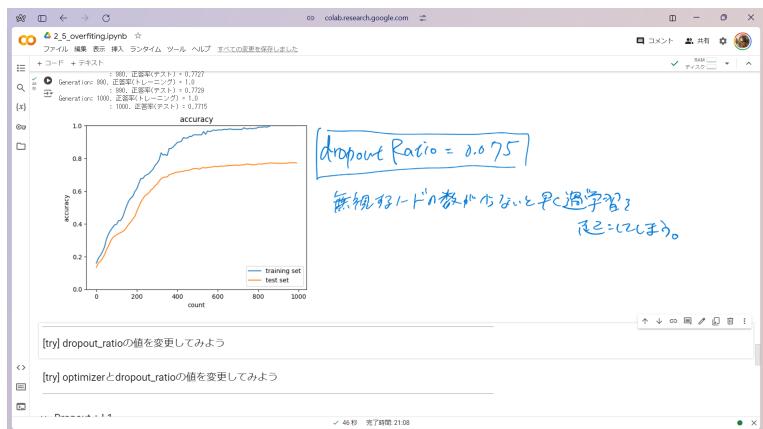
✓ 42 秒 完了時間: 21



```

2.5_overfitting.ipynb
+ コード + テキスト
+ ファイル 延長 表示 ランタイム ツール ヘルプ
+ コード + テキスト
Q [x] 1 from tensorflow import optimizers
2 (x_train, y_train, x_test, y_test) = load_iris(normalize=True)
3 print("データ読み込み完了")
4
5 # データを削除するのに、学習データを削除
6 # x_train = x_train[100]
7 # y_train = y_train[100]
8
9 # ログ出力
10 dropout_ratio = 0.25
11 use_dropout = True
12 dropout_ratio = 0.05
13
14 network = MultiLayerNet(x=x_train, y=y_train, batch_size=100, hidden_size=[100, 100, 100, 100], output_size=10)
15 optimizer = optimizers.SGD(learning_rate=0.01)
16 if use_dropout:
17     optimizer = optimizers.Adam(learning_rate=0.01, momentum=0.9)
18 else:
19     optimizer = optimizers.Adam(learning_rate=0.01)
20
21 optimizer.compile()
22 train_size = len(x_train)
23 batch_size = 100
24 batch_size = 100
25
26 train_loss_list = []
27 accuracies_train = []
28 accuracies_val = []
29
30 plot_interval = 10
31
32 for i in range(len(x_train)):
33     batch_index = np.random.choice(train_size, batch_size)
34     x_batch = x_train[batch_index]
35     y_batch = y_train[batch_index]
36
37     grad = network.gradient(x_batch, y_batch)
38     optimizer.update(network.params, grad)
39
40    実行 (3秒) - cell line 33 > accuracy() > predict() > forward()

```



```

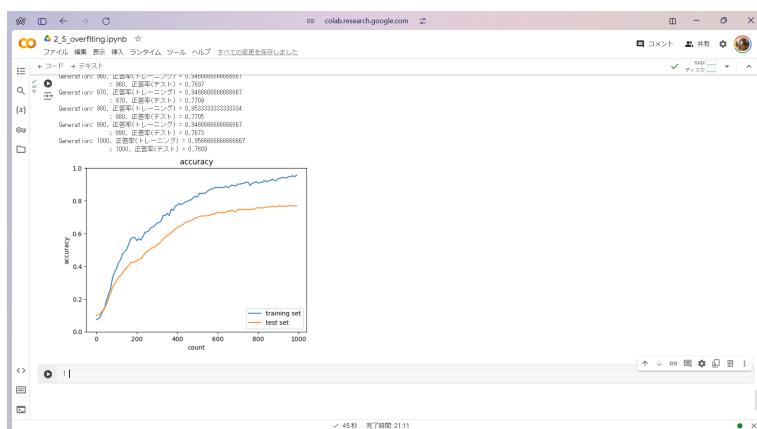
2.5_overfitting.ipynb
+ コード + テキスト
+ ファイル 延長 表示 ランタイム ツール ヘルプ
+ コード + テキスト
Q [x] 1 from tensorflow import optimizers
2 (x_train, y_train, x_test, y_test) = load_iris(normalize=True)
3 print("データ読み込み完了")
4
5 # データを削除するのに、学習データを削除
6 # x_train = x_train[100]
7 # y_train = y_train[100]
8
9 # ログ出力
10 dropout_ratio = 0.15
11 use_dropout = True
12 dropout_ratio = 0.15
13
14 network = MultiLayerNet(x=x_train, hidden_size=[100, 100, 100, 100, 100], output_size=10,
15                         weight_decay_lambda=0.001, batch_size=100, use_dropout=True, dropout_ratio=dropout_ratio)
16 optimizer = optimizers.SGD(learning_rate=0.01)
17 if use_dropout:
18     optimizer = optimizers.Adam(learning_rate=0.01, momentum=0.9)
19 else:
20     optimizer = optimizers.Adam(learning_rate=0.01)
21 optimizer.compile()
22
23 train_size = len(x_train)
24 batch_size = 100
25
26 train_loss_list = []
27 accuracies_train = []
28 accuracies_val = []
29
30 plot_interval = 10
31
32 for i in range(len(x_train)):
33     batch_index = np.random.choice(len(x_train), batch_size)
34     x_batch = x_train[batch_index]
35     y_batch = y_train[batch_index]
36
37     grad = network.gradient(x_batch, y_batch)
38     optimizer.update(network.params, grad)
39
40    実行 (3秒) - cell line 33 > accuracy() > predict() > forward()

```

The figure shows a line graph titled "accuracy" plotted against "count" (0 to 1000). The legend indicates two series: "training set" (blue line) and "test set" (orange line). The training set accuracy starts at ~0.15 and quickly rises to ~0.85, plateauing around 100 counts. The test set accuracy starts at ~0.15 and rises more slowly, reaching ~0.85 by 100 counts and plateauing. Handwritten notes in blue ink say "Optimizer to Adam = 12.0% ↓" above the graph and "Adam 自体が早く収束する最適化エンジニアリング。dropout の効果を幅消しにしている。" below it.

Optimizer & Adam- $\beta_1=0.9$, $\beta_2=0.999$

Adam 自体が早く収束する最適化マシンアーキ、
dropoutの強みを十分消しにいってしまう。



2 2_6_simple_convolution_network

Google Colab の実行結果

```
# Google Colab での実行が各欄へも
import sys
print(sys.version)
ENV_COLAB = True if "google.colab" in sys.modules else False
if ENV_COLAB:
    from google.colab import drive
    drive.mount('/content/drive')
    my_note_path = "/content/drive/My Drive/Colab Notebooks/仮2布ノートブック/DNN_code_colab_dv2/notebook"
    os.chdir(my_note_path)
```

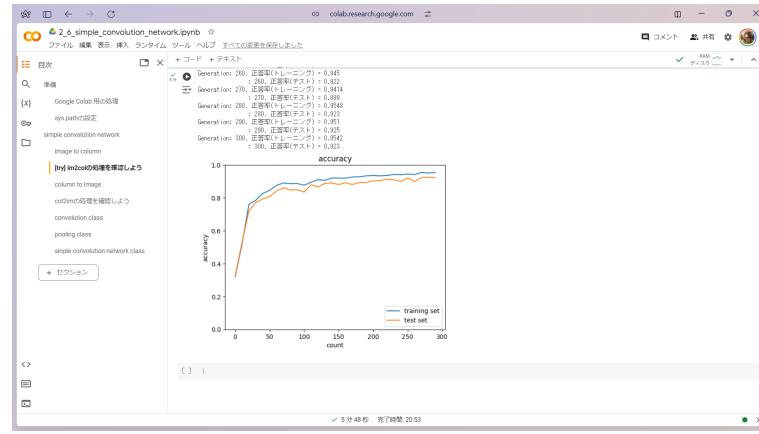
Mounted at /content/drive



The screenshot shows a Google Colab notebook titled "2.2_simple_convolution_network.ipynb". The code cell contains the following Python code:

```
def im2col(x, stride=1, pad=0):
    """x: input image, stride: stride of output, pad: padding size"""
    if len(x.shape) == 2:
        x = x[:, :, None]
    n, c, h, w = x.shape
    sh, sw = stride, stride
    nh = int((h + 2 * pad) / sh)
    nw = int((w + 2 * pad) / sw)
    col = np.zeros((c, nh, nw))
    for i in range(c):
        for j in range(nh):
            for k in range(nw):
                col[i, j, k] = x[i, j * sh: j * sh + sh, k * sw: k * sw + sw]
    return col, (nh, nw)
```

The code cell is highlighted with a yellow box and has a green checkmark icon indicating it has been run successfully. The output of the code is displayed below the cell, showing the resulting matrix and its dimensions.



試しに値を入れて True が返るか確認してみよう。

✓ [35] 1.4 試しに一つ値を入れてみて、予測が正しいかを見てみる。
④ 8000番目の組み合わせを使用
4 x_add_dtest = test[8000].reshape([1,28,28])
5 d_add_dtest = d_test[8000] 答え:8
6
7 ④ SimpleComnetのpredict関数で予測用

3 2_6_simple_convolution_network_after

simple convolution network

image to column → 縦横を2次元、2次元を(次元の形)に並べ替える。
→ 線上操作で計算式

↓

```
11 import pickle  
12 from collections import OrderedDict  
13 from common import *  
14 from conv import conv2d, max_pool2d  
15 from softmax import softmax  
16 from fully_connected import fully_connected  
17 from linear import linear  
18 from loss import cross_entropy  
19 from util import data, filter2d, stride2d, pad2d  
20  
21 # データ読み込み  
22 x_train, y_train = data('mnist/mnist_train.pkl')  
23 x_test, y_test = data('mnist/mnist_test.pkl')  
24  
25 # フィルター定義  
26 filter_w, filter_b = filter2d(3, 3, 1, 1, 1, 1)  
27 pad_w, pad_b = pad2d(1, 1, 1, 1, 1, 1)  
28 stride_w, stride_b = stride2d(2, 2, 1, 1, 1, 1)  
29  
30 # パラメータ初期化  
31 w1, b1, w2, b2, w3, b3 = init_params()  
32  
33 cat = convolution(x_train, w1, b1, filter_w, filter_b, stride_w, stride_b)  
34 cat = max_pool2d(cat, 2, 2, 1, 1)  
35 cat = convolution(cat, w2, b2, filter_w, filter_b, stride_w, stride_b)  
36 cat = max_pool2d(cat, 2, 2, 1, 1)  
37 cat = convolution(cat, w3, b3, filter_w, filter_b, stride_w, stride_b)
```

↑ ① channel
↑ ② height
↑ ③ width
↑ (4, 28)
↑ np.pad : padding3d (constant: 0.0, 0.0, 0.0)
↓ ④ 0番目と1番目の操作を繰り返す。

2.6_simple_convolution_network_after.ipynb

コメント ブックマーク ヘルプ この変更を保存しました

+ コード + テキスト

▼ [try] im2colの処理を確認しよう

(n) [箇所内にコメントを挿入している行をコメントアウトして下のコードを実行してみよう

- input_dataの各次元のサイズやフィルターサイズ、ストライド・パディングを変えてみよう

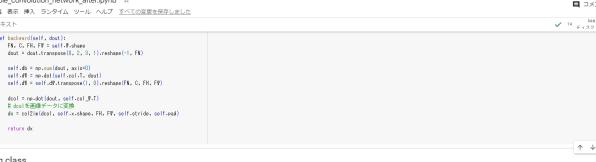
↑ [try] im2colの処理を確認しよう

input_data = np.random.randint(1, 4, (4, 4, 1)) # color, channel, height, width, valueを変更
print("input data : \n", input_data)
print("-----")
filter_size = 3
filter_x = 3
filter_y = 3
stride = 1
pad = 1
col = im2col(input_data, filter_x, filter_y, stride, pad)
print("col : \n", col)
print("-----")

[[[1, 2, 3, 4],
 [5, 6, 7, 8],
 [9, 10, 11, 12],
 [13, 14, 15, 16]]]
[[[16, 17, 18, 19],
 [20, 21, 22, 23],
 [24, 25, 26, 27],
 [28, 29, 30, 31]]]
[[[32, 33, 34, 35],
 [36, 37, 38, 39],
 [40, 41, 42, 43],
 [44, 45, 46, 47]]]
[[[48, 49, 50, 51],
 [52, 53, 54, 55],
 [56, 57, 58, 59],
 [60, 61, 62, 63]]]]

[[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63]]]

column to image



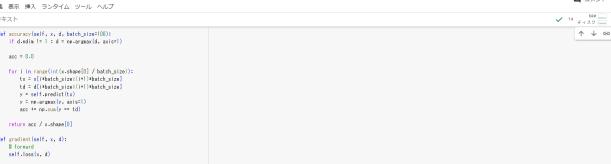
The screenshot shows a Google Colab notebook with the following code:

```
class Pooling:
    def __init__(self, pool_h, pool_w, stride=2):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.out_h = (self.in_h - pool_h) // stride + 1
        self.out_w = (self.in_w - pool_w) // stride + 1
        self.argmax = None

    def forward(self, x):
        self.in_h, self.in_w, self.x, self.stride, self.out_h, self.out_w = x
        self.argmax = np.zeros((self.out_h, self.out_w))
        self.out = np.zeros((self.out_h, self.out_w))

        for i in range(self.out_h):
            for j in range(self.out_w):
                h1 = i * self.stride
                w1 = j * self.stride
                h2 = h1 + self.pool_h
                w2 = w1 + self.pool_w
                pool_x = self.x[h1:h2, w1:w2]
                self.argmax[i][j] = np.argmax(pool_x)
                self.out[i][j] = np.max(pool_x)

    def backward(self, dout):
        dx = np.zeros_like(self.x)
        for i in range(self.out_h):
            for j in range(self.out_w):
                h1 = i * self.stride
                w1 = j * self.stride
                h2 = h1 + self.pool_h
                w2 = w1 + self.pool_w
                pool_x = self.x[h1:h2, w1:w2]
                argmax = self.argmax[i][j]
                dx[h1:h2, w1:w2][argmax] = dout[i][j]
        return dx
```



The screenshot shows a Google Colab notebook titled "1.6_simple_convolutional_network_after.ipynb". The code implements a simple convolutional neural network with two layers. It uses TensorFlow's Keras API and includes functions for initializing weights, calculating gradients, and performing backpropagation. The notebook also includes a check function to verify the correctness of the implemented layers.

```
class Layer:
    def __init__(self, n_out, n_in=None):
        self.n_out = n_out
        self.n_in = n_in
        self.v = np.random.randn(n_out, n_in) * 0.01
        self.b = np.zeros((n_out, 1))

    def forward(self, x):
        self.x = x
        return self.x @ self.v + self.b

    def backward(self, v):
        self.v -= self.x.T @ v * 0.01
        self.b -= v * 0.01
        return v @ self.v.T

    def gradient(self, v, b):
        self.v -= v @ self.v.T * 0.01
        self.b -= b * 0.01
        return self.v, self.b

    def layer_backward(self, v):
        self.v -= self.x.T @ v * 0.01
        self.b -= v * 0.01
        return v @ self.v.T

    def layer_gradient(self, v):
        self.v -= v @ self.v.T * 0.01
        self.b -= v * 0.01
        return self.v, self.b

    def update(self, lr):
        self.v -= lr * self.v
        self.b -= lr * self.b

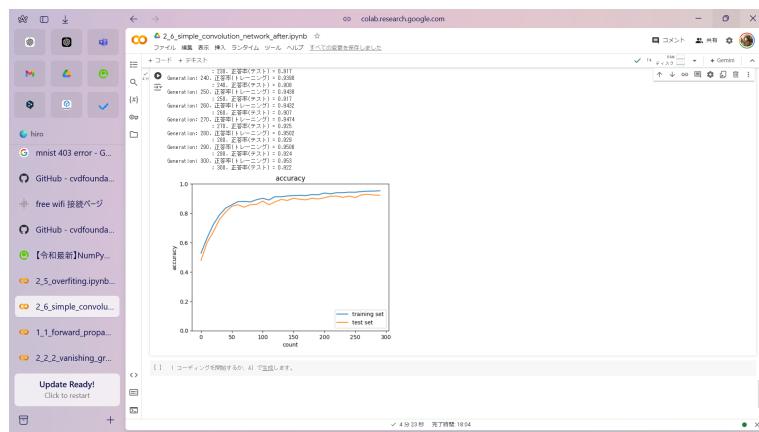
    def __str__(self):
        return f'Layer({self.n_out}, {self.n_in})'

class ConvLayer(Layer):
    def __init__(self, n_out, n_in, k=3, s=1, p=0):
        super().__init__(n_out, n_in)
        self.k = k
        self.s = s
        self.p = p
        self.v = np.random.randn(n_out, n_in // s ** 2, k, k) * 0.01
        self.b = np.zeros((n_out, 1))

    def forward(self, x):
        self.x = x
        n, c, h, w = x.shape
        sh = (h - self.k + 2 * self.p) // self.s + 1
        sw = (w - self.k + 2 * self.p) // self.s + 1
        self.z = np.zeros((n, self.n_out, sh, sw))
        for i in range(n):
            for j in range(self.n_out):
                for y in range(sh):
                    for x in range(sw):
                        self.z[i, j, y, x] = np.sum(x * self.v[j, :, :, :] * self.x[i, :, y * self.s:y * self.s + self.k, x * self.s:x * self.s + self.k])
        return self.z + self.b

    def backward(self, v):
        self.v -= self.x.T @ v * 0.01
        self.b -= v * 0.01
        return v @ self.v.T

    def gradient(self, v, b):
        self.v -= v @ self.v.T * 0.01
        self.b -= b * 0.01
        return self.v, self.b
```

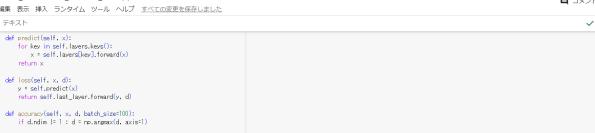


4 2_7_double_convolution_network

double_convolution_network_after.ipynb

```
1 import pickle
2 import numpy as np
3 from collections import OrderedDict
4 from common import layers
5 from common import optimizer
6 from common import load_weight
7 import math
8 import util
9
10 class DoubleConvNet:
11     def __init__(self, input_size=28*28, n_classes=10,
12                  conv_params=[{'filter_size': 5, 'out_channels': 16, 'stride': 1}, 
13                               {'filter_size': 5, 'out_channels': 16, 'stride': 1}, 
14                               {'filter_size': 5, 'out_channels': 32, 'stride': 1}, 
15                               {'filter_size': 5, 'out_channels': 32, 'stride': 1}, 
16                               {'filter_size': 5, 'out_channels': 64, 'stride': 1}, 
17                               {'filter_size': 5, 'out_channels': 64, 'stride': 1}], 
18                  pool_params=[{'size': 2, 'stride': 2}, 
19                               {'size': 2, 'stride': 2}, 
20                               {'size': 2, 'stride': 2}, 
21                               {'size': 2, 'stride': 2}, 
22                               {'size': 2, 'stride': 2}], 
23                  weight_init_fn=lambda x: util.xavier_initializer(x), 
24                  weight_std_fn=lambda x: util.random_normal_outlier_std(x), 
25                  bias_init_fn=lambda x: util.zeros_initializer(x), 
26                  bias_std_fn=lambda x: util.random_normal_outlier_std(x), 
27                  nonlin_fn=lambda x: util.relu(x), 
28                  nonlin_std_fn=lambda x: util.random_normal_outlier_std(x), 
29                  loss_fn=lambda y, p: util.softmax_crossentropy(y, p), 
30                  loss_std_fn=lambda y, p: util.softmax_crossentropy(y, p)], 
31                  layers=layers, 
32                  layers_order=layers_order, 
33                  layers_fn=layers_fn, 
34                  layers_order_fn=layers_order_fn, 
35                  layers_fn_order=layers_fn_order, 
36                  layers_fn_order_fn=layers_fn_order_fn)
```

double Convolution



The screenshot shows a Google Colab notebook titled "2_d_double_convolution_network_after.ipynb". The code implements a neural network layer with two parallel convolutions. It includes methods for forward pass, backward pass, and gradient calculation. The code uses PyTorch's nn.Module and nn.Conv2d classes.

```
40 def predict(self, x):
41     for key in self.layers.keys():
42         x = self.layers[key].forward(x)
43     return x
44
45 def forward(self, x, d):
46     v = self.predict(x)
47     v.backward(d, keep_forward=True)
48
49 def backward(self, d, backprop=True):
50     if not backprop:
51         d = d.clone()
52     if d.dim() == 1:
53         d = d.unsqueeze(0)
54     acc = 0.0
55
56     for i in range(len(self.layers) - 1):
57         b_size = d.size(0)
58         d = self.layers[i].backward(d)
59         v = self.predict(x)
60         x = v.detach_()
61         acc += torch.sum(v * d)
62
63     return acc / b_size()
64
65 def gradient(self, x, d):
66     if not self.training:
67         self.loss(x, d)
68
69     # backward
70     dout = d
71     layers = list(self.layers.values())
72     layers.reverse()
73
74     for layer in layers:
75         dout = layer.backward(dout)
76
77     # backward
78     grad_d = []
79     grad_v1 = []
80     grad_v2 = []
81     grad_v3 = []
82     grad_v4 = []
83
84     for i in range(len(layers)):
85         if i == 0:
86             grad_d.append(d)
87             grad_v1.append(layers[i].backward(d))
88             grad_v2.append(layers[i].backward(grad_v1[-1]))
89             grad_v3.append(layers[i].backward(grad_v2[-1]))
90             grad_v4.append(layers[i].backward(grad_v3[-1]))
91
92         else:
93             grad_d.append(layers[i].backward(grad_d[-1]))
94             grad_v1.append(layers[i].backward(grad_d[-1]))
95             grad_v2.append(layers[i].backward(grad_v1[-1]))
96             grad_v3.append(layers[i].backward(grad_v2[-1]))
97             grad_v4.append(layers[i].backward(grad_v3[-1]))
```

```
1 # データの読み込み
2 (x_train, d_train), (x_test, d_test) = load_mnist(flatten=False)
3
4 print("データ読み込み完了")
5
6 # モデル構築
7 network = DoubleConvNet(input_size=28*28,
8                         conv_params=[{'filter_size': 5, 'pad': 0}, {'filter_size': 5, 'pad': 0}], 
9                         pool_params=[{'pool_size': 2, 'stride': 2}, {'pool_size': 2, 'stride': 2}], 
10                        hidden_units=[100], output_size=10)
11
12 optimizer = optimizers.Adam()
13
14 # ハイパーパラメータの設定
15 learning_rate = 0.001
16 max_iter = 100
17 batch_size = 100
18
19 # ハイパーパラメータの設定
20 train_size = x_train.shape[0]
21 batch_size = 100
22
23 train_loss_list = []
24 accuracy_train = []
25 accuracy_val = []
26
27 plot_interval = 10
28
29
30
31 for i in range(max_iter):
32     batch_idx = np.random.choice(train_size, batch_size)
33     x_batch_d = x_train[batch_idx]
34     d_batch_d = d_train[batch_idx]
35
36     grad_d = network.backward(x_batch_d, d_batch_d)
37     network.backward_and_update_params(x_batch_d, grad_d)
38     loss = network.loss(x_batch_d, d_batch_d)
39     train_loss_list.append(loss)
40
```

```
# 2.2_double_convolution_network_after.ipynb ☆
# ファイル データ 表示 検索 ランタイム サーバ ヘルプ
# コメント & 共有
# フィルター: すべて
# 14 / 14 ページ 1 / 1 ブラウザ

for i in range(1000):
    batch_size = np.random.choice(trn_size, batch_size)
    dBatch = dTrain[dTrain['batch_idx'] == batch_size]
    dBatch = dBatch.sample(n=batch_size)

    xBatch = np.array(dBatch['x'].values)
    yBatch = np.array(dBatch['y'].values)

    trainLossHist.append(loss)

    if (i+1) % self.interval == 0:
        mon_sess = tf.train.MonitoredTrainingSession(
            master='localhost:2222',
            save_checkpoints_secs=1,
            checkpoint_dir='./checkpoints',
            log_step_count_steps=1000)
        accTrain, accTest = mon_sess.run([accuracy, accuracy],
                                         feed_dict={x: xBatch,
                                                    y: yBatch})
        accTrainHist.append(accTrain)
        accTestHist.append(accTest)
        print('Generation: ' + str(i+1) + ' 正解率(トレーニング) = ' + str(accTrain))
        print('Generation: ' + str(i+1) + ' 正解率(テスト) = ' + str(accTest))

    lists = np.arange(0, len(x), plot_interval)
    plt.subplots(2, 2, figsize=(10, 8))
    plt.subplot(2, 2, 1)
    plt.title('accuracy', label='training set')
    plt.plot(lists, accTrainHist)
    plt.subplot(2, 2, 2)
    plt.title('accuracy', label='test set')
    plt.plot(lists, accTestHist)
    plt.subplot(2, 2, 3)
    plt.title('loss')
    plt.plot(lists, trainLossHist)
    plt.subplot(2, 2, 4)
    plt.title('accuracy')
    plt.plot(lists, accTestHist)
    plt.show()

# データ読み込み終了
```

