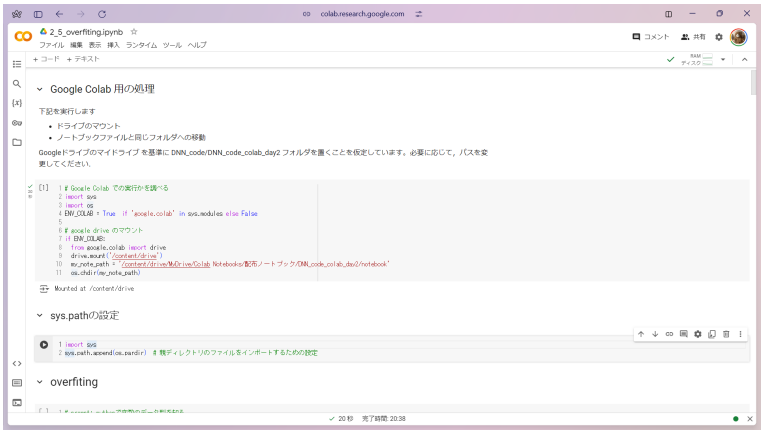
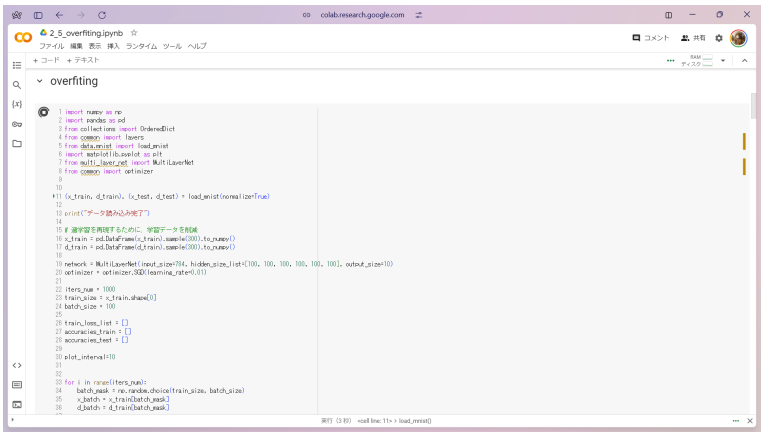
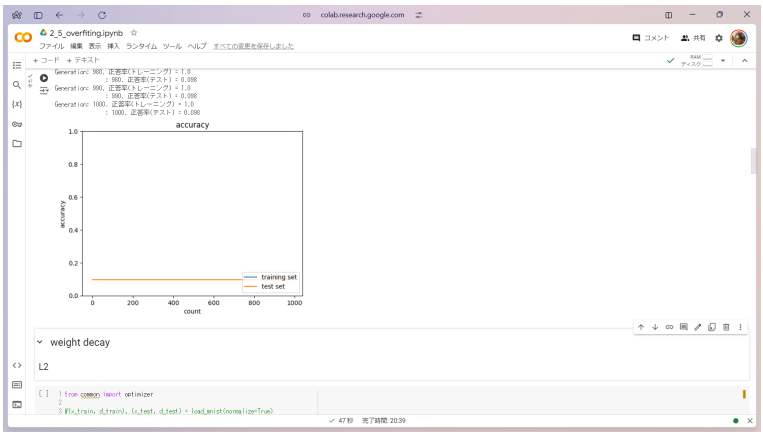
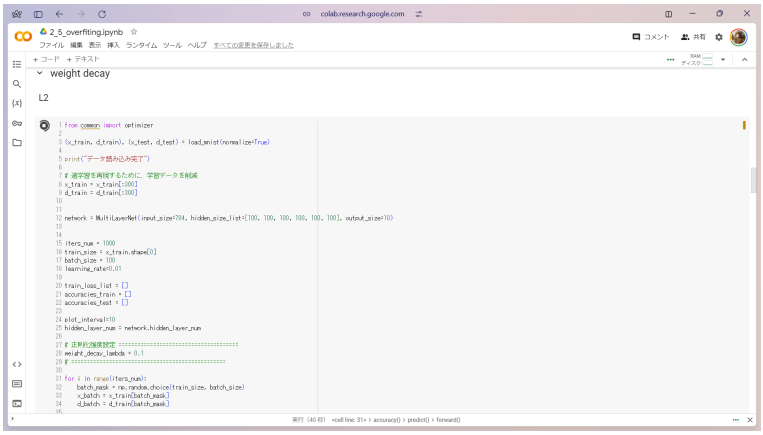
深層学習 day2 実装演習キャプチャ 2

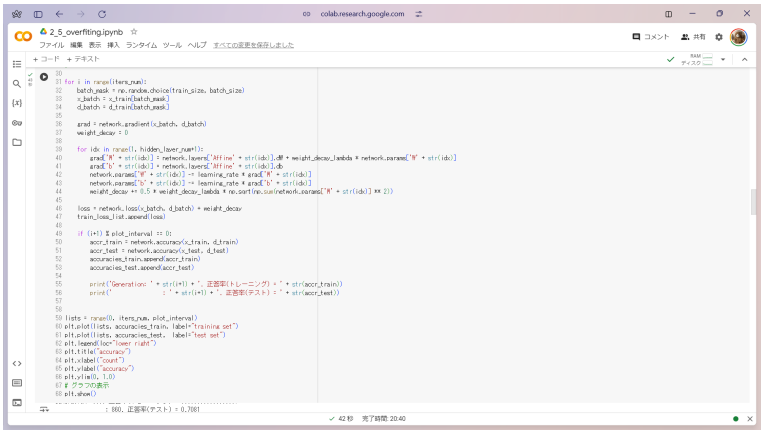
1　2_5_overfitting

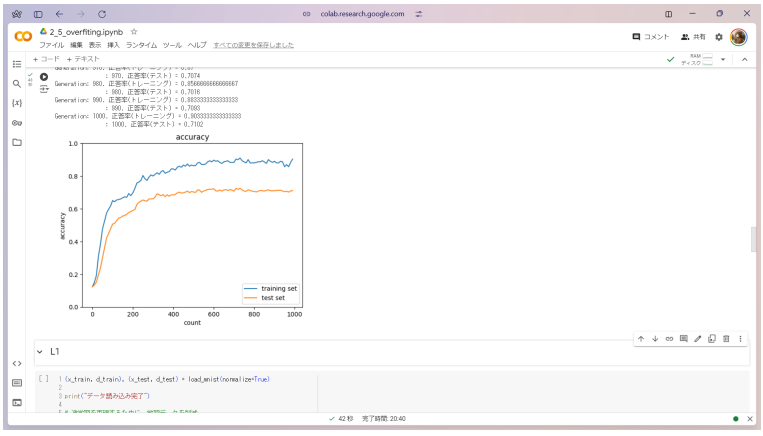


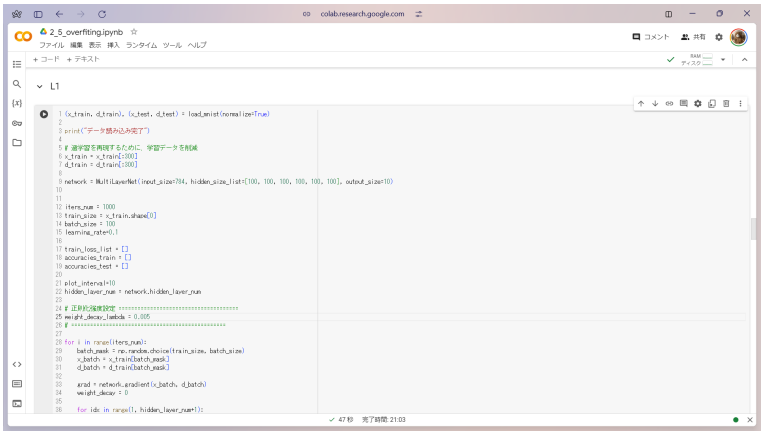


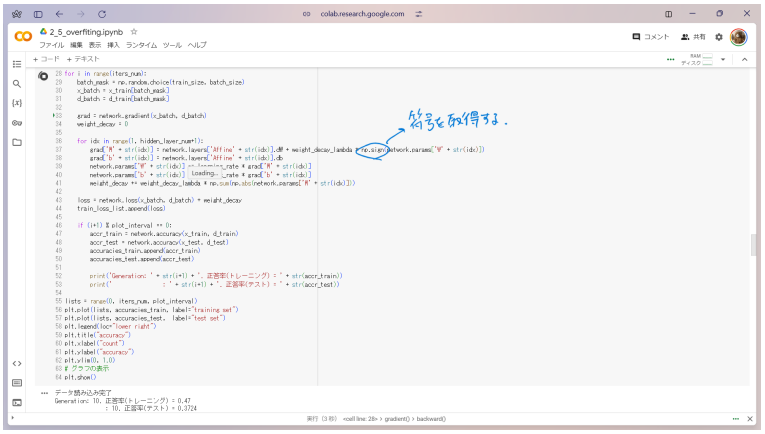


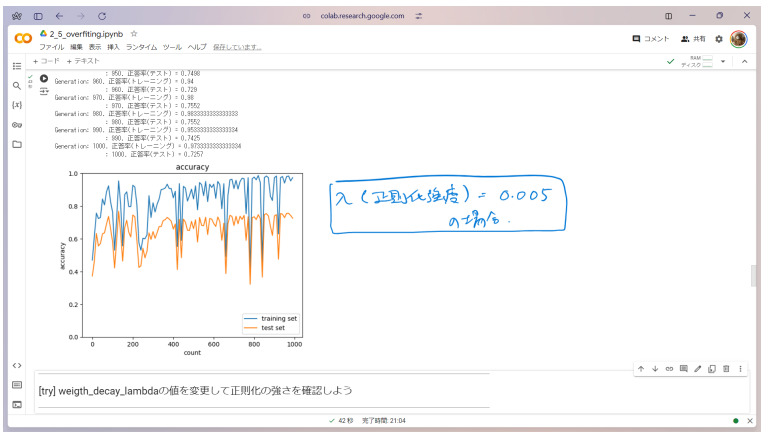

colab.research.google.com
2_5_overfitting.ipynb
weight decay
L2
from common import optimizer
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)
print("データ読み込み完了")
# 過学習を再現するために、学習データを削減
x_train = x_train[:300]
d_train = d_train[:300]
network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100], output_size=10)
iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate=0.01
train_loss_list = []
accuracies_train = []
accuracies_test = []
plot_interval=10
hidden_layer_num = network.hidden_layer_num
# 正則化強度設定 ======================================
weight_decay_lambda = 0.1
# ===
for i in range(iters_num):
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
d_batch = d_train[batch_mask]

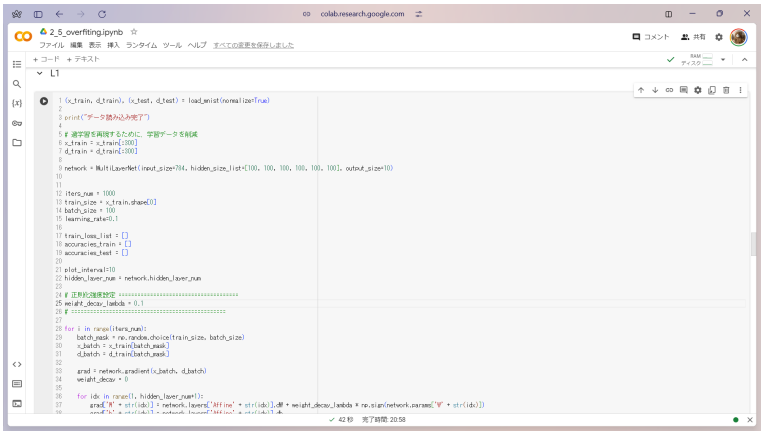

colab.research.google.com
2_5_overfitting.ipynb
for i in range(iters_num):
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
d_batch = d_train[batch_mask]
grad = network.gradient(x_batch, d_batch)
weight_decay = 0
for idx in range(1, hidden_layer_num+1):
grad['W' + str(idx)] = network.layers['Affine' + str(idx)].dW + weight_decay_lambda * network.params['W' + str(idx)]
grad['b' + str(idx)] = network.layers['Affine' + str(idx)].db
network.params['W' + str(idx)] -= learning_rate * grad['W' + str(idx)]
network.params['b' + str(idx)] -= learning_rate * grad['b' + str(idx)]
weight_decay += 0.5 * weight_decay_lambda * np.sqrt(np.sum(network.params['W' + str(idx)] ** 2))
loss = network.loss(x_batch, d_batch) + weight_decay
train_loss_list.append(loss)
if (i+1) % plot_interval == 0:
acc_train = network.accuracy(x_train, d_train)
acc_test = network.accuracy(x_test, d_test)
accuracies_train.append(acc_train)
accuracies_test.append(acc_test)
print("Generation: " + str(i+1) + ". 正答率(トレーニング) = " + str(acc_train))
print("   : " + str(i+1) + ". 正答率(テスト) = " + str(acc_test))
lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test, label="test set")
plt.legend(loc="lower right")
plt.title("accuracy")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
# グラフの描画
plt.show()
: 860, 正答率(テスト) = 0.7081
42 秒 完了時間: 20:40

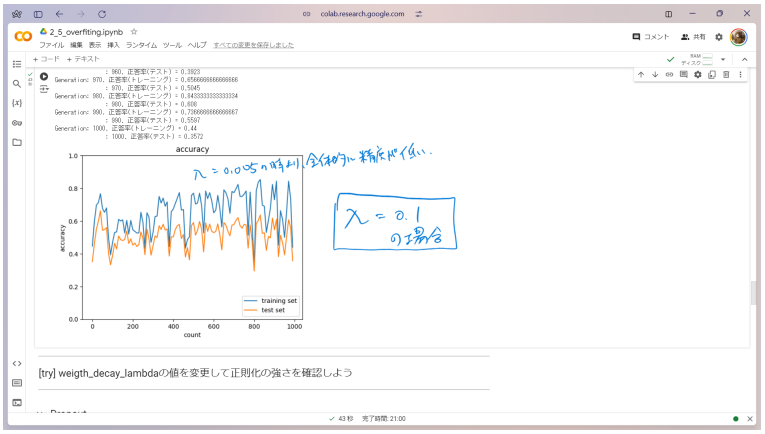

colab.research.google.com
2_5_overfitting.ipynb
: 870, 正答率(テスト) = 0.7074
Generation: 880. 正答率(トレーニング) = 0.8566666666666667
: 880, 正答率(テスト) = 0.7016
Generation: 890. 正答率(トレーニング) = 0.8633333333333333
: 890, 正答率(テスト) = 0.7093
Generation: 900. 正答率(トレーニング) = 0.8633333333333333
: 900, 正答率(テスト) = 0.7102
accuracy
accuracy
count
training set
test set
L1
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)
print("データ読み込み完了")
42 秒 完了時間: 20:40

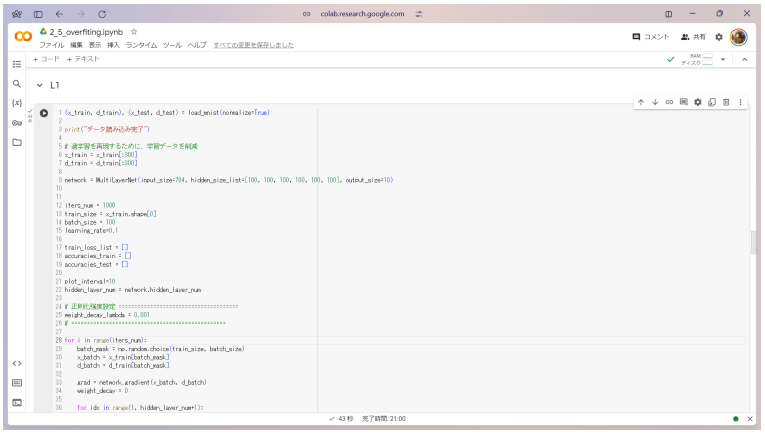

2_5_overfitting.ipynb
L1
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)
print("データ読み込み完了")
# 過学習を再現するために、学習データを削減
x_train = x_train[:300]
d_train = d_train[:300]
network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100], output_size=10)
iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate=0.1
train_loss_list = []
accuracies_train = []
accuracies_test = []
plot_interval=10
hidden_layer_num = network.hidden_layer_num
# 正則化強度設定 ======================================
weight_decay_lambda = 0.005
# ===
for i in range(iters_num):
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
d_batch = d_train[batch_mask]
grad = network.gradient(x_batch, d_batch)
weight_decay = 0
for idx in range(1, hidden_layer_num+1):
47 秒 完了時間 21:03

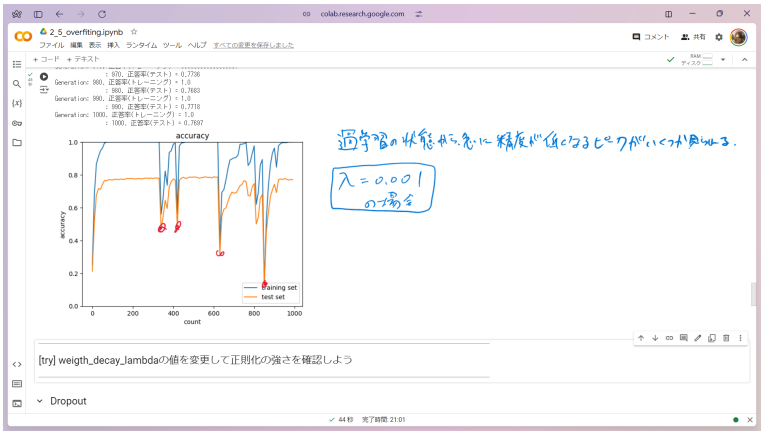

2_5_overfitting.ipynb
for i in range(iters_num):
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
d_batch = d_train[batch_mask]
grad = network.gradient(x_batch, d_batch)
weight_decay = 0
for idx in range(1, hidden_layer_num+1):
grad['W' + str(idx)] = network.layers['Affine' + str(idx)].dW + weight_decay_lambda * np.sign(network.params['W' + str(idx)])
grad['b' + str(idx)] = network.layers['Affine' + str(idx)].db
network.params['W' + str(idx)] -= learning_rate * grad['W' + str(idx)]
network.params['b' + str(idx)] -= learning_rate * grad['b' + str(idx)]
weight_decay += weight_decay_lambda * np.sum(np.abs(network.params['W' + str(idx)]))
loss = network.loss(x_batch, d_batch) + weight_decay
train_loss_list.append(loss)
if (i+1) % plot_interval == 0:
accr_train = network.accuracy(x_train, d_train)
accr_test = network.accuracy(x_test, d_test)
accuracies_train.append(accr_train)
accuracies_test.append(accr_test)
print("Generation: " + str(i+1) + ". 正答率(トレーニング) : " + str(accr_train))
print("   : " + str(i+1) + ". 正答率(テスト) : " + str(accr_test))
lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test, label="test set")
plt.legend(loc="lower right")
plt.title("accuracy")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
# グラフの表示
plt.show()
符号を取得する.
データ読み込み完了
Generation: 10. 正答率(トレーニング) : 0.47
: 10. 正答率(テスト) : 0.3724
実行 (3 秒) <cell line: 28> > gradient() > backward()

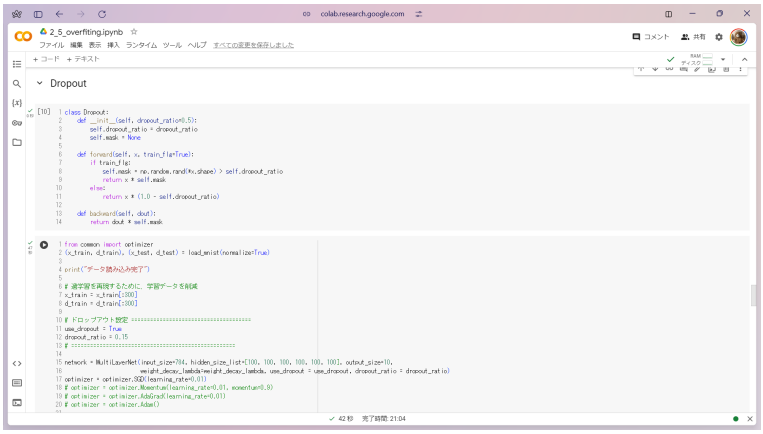

2_5_overfitting.ipynb
: 950, 正答率(テスト) : 0.7498
Generation: 960. 正答率(トレーニング) : 0.94
: 960, 正答率(テスト) : 0.729
Generation: 970. 正答率(トレーニング) : 0.98
: 970, 正答率(テスト) : 0.7552
Generation: 980. 正答率(トレーニング) : 0.9833333333333333
: 980, 正答率(テスト) : 0.7552
Generation: 990. 正答率(トレーニング) : 0.9533333333333334
: 990, 正答率(テスト) : 0.7425
Generation: 1000. 正答率(トレーニング) : 0.9733333333333334
: 1000, 正答率(テスト) : 0.7257
accuracy
training set
test set
count
入(正則化強度) = 0.005
の場合.
[try] weigth_decay_lambdaの値を変更して正則化の強さを確認しよう
42 秒 完了時間 21:04

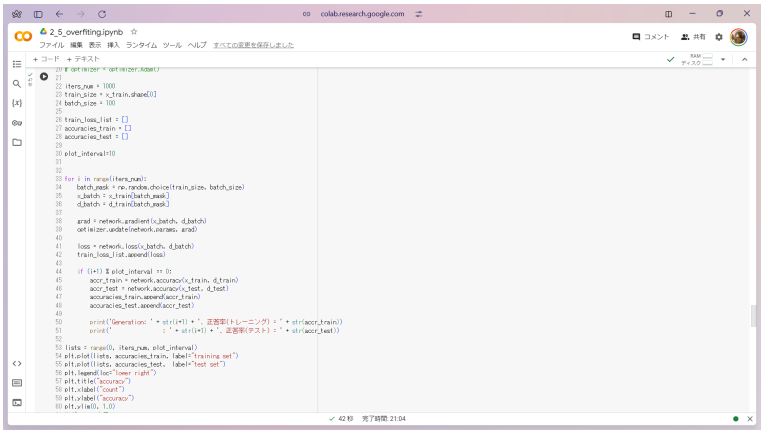

colab.research.google.com
2_5_overfitting.ipynb
L1
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)
print("データ読み込み完了")
# 過学習を再現するために、学習データを削減
x_train = x_train[:300]
d_train = d_train[:300]
network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100], output_size=10)
iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate=0.1
train_loss_list = []
accuracies_train = []
accuracies_test = []
plot_interval=10
hidden_layer_num = network.hidden_layer_num
# 正則化強度設定 ======================================
weight_decay_lambda = 0.1
# ===
for i in range(iters_num):
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
d_batch = d_train[batch_mask]
grad = network.gradient(x_batch, d_batch)
weight_decay = 0
for idx in range(1, hidden_layer_num+1):
grad['W' + str(idx)] = network.layers['Affine' + str(idx)].dW + weight_decay_lambda * np.sign(network.params['W' + str(idx)])
42 秒 完了時間 20:58

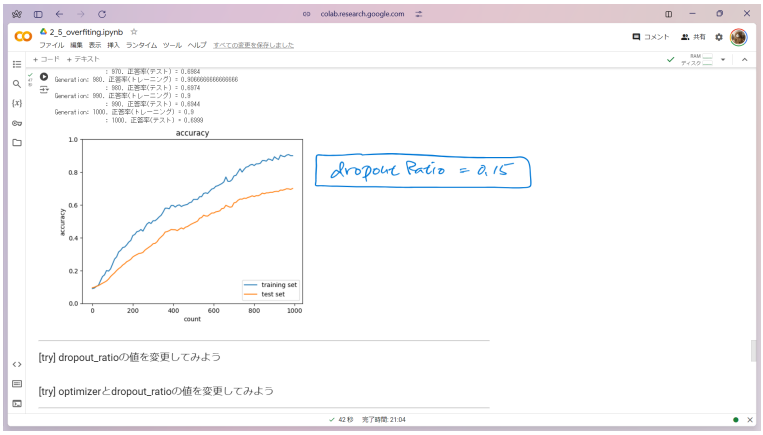

colab.research.google.com
2_5_overfitting.ipynb
: 960, 正答率(テスト) = 0.3923
Generation: 970, 正答率(トレーニング) = 0.6566666666666666
: 970, 正答率(テスト) = 0.5945
Generation: 980, 正答率(トレーニング) = 0.6433333333333334
: 980, 正答率(テスト) = 0.608
Generation: 990, 正答率(トレーニング) = 0.7366666666666667
: 990, 正答率(テスト) = 0.5502
Generation: 1000, 正答率(トレーニング) = 0.44
: 1000, 正答率(テスト) = 0.3572
accuracy
λ = 0.005の時は全体的に精度が低い.
λ = 0.1 の場合
training set
test set
count
[try] weigth_decay_lambdaの値を変更して正則化の強さを確認しよう
43 秒 完了時間 21:00

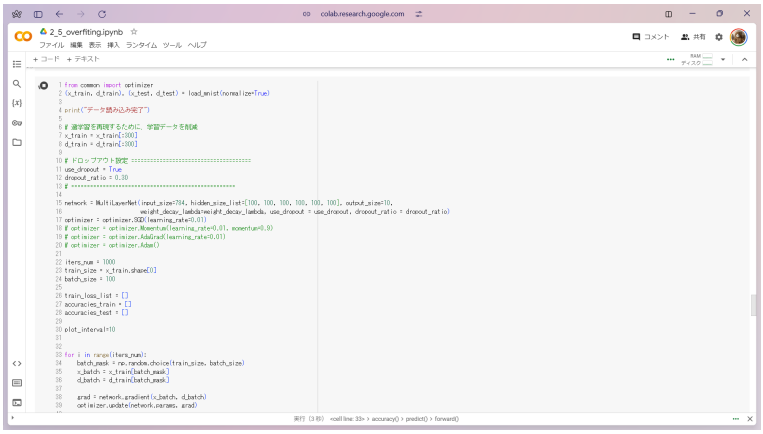

colab.research.google.com
2_5_overfitting.ipynb
L1
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)
print("データ読み込み完了")
# 過学習を再現するために、学習データを削減
x_train = x_train[:300]
d_train = d_train[:300]
network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100], output_size=10)
iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate=0.1
train_loss_list = []
accuracies_train = []
accuracies_test = []
plot_interval=10
hidden_layer_num = network.hidden_layer_num
# 正則化強度設定 ======================================
weight_decay_lambda = 0.001
# ===
for i in range(iters_num):
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
d_batch = d_train[batch_mask]
grad = network.gradient(x_batch, d_batch)
weight_decay = 0
for idx in range(1, hidden_layer_num+1):
43 秒 完了時間 21:00

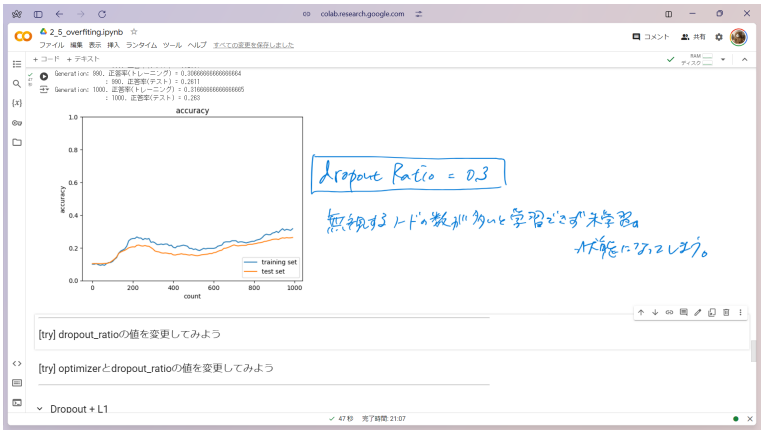

2_5_overfitting.ipynb
accuracy
count
training set
test set
過学習の状態から急に精度が低くなるヒゲがいくつか見られる。
λ=0.001
の場合
[try] weigth_decay_lambdaの値を変更して正則化の強さを確認しよう
Dropout

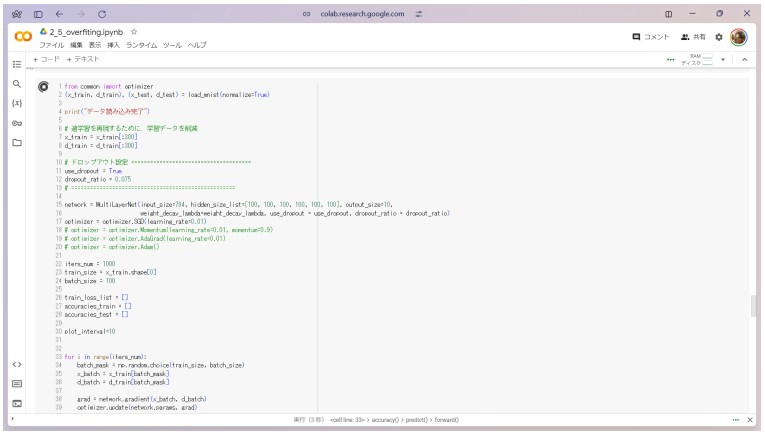

2_5_overfitting.ipynb
Dropout
class Dropout:
def __init__(self, dropout_ratio=0.5):
self.dropout_ratio = dropout_ratio
self.mask = None
def forward(self, x, train_flg=True):
if train_flg:
self.mask = np.random.rand(*x.shape) > self.dropout_ratio
return x * self.mask
else:
return x * (1.0 - self.dropout_ratio)
def backward(self, dout):
return dout * self.mask
from common import optimizer
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)
print("データ読み込み完了")
x_train = x_train[:300]
d_train = d_train[:300]
network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100], output_size=10,
optimizer = optimizer.SGD(learning_rate=0.01)
optimizer = optimizer.Momentum(learning_rate=0.01, momentum=0.9)
optimizer = optimizer.AdaGrad(learning_rate=0.01)
optimizer = optimizer.Adam()

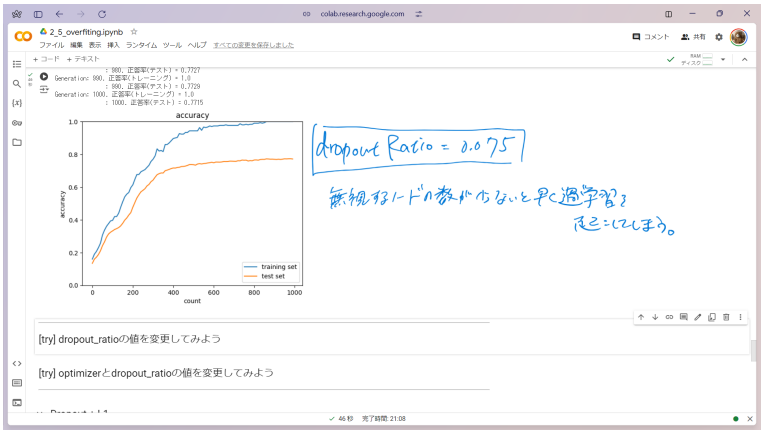

2_5_overfitting.ipynb
iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
train_loss_list = []
accuracies_train = []
accuracies_test = []
plot_interval=10
for i in range(iters_num):
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
d_batch = d_train[batch_mask]
grad = network.gradient(x_batch, d_batch)
optimizer.update(network.params, grad)
loss = network.loss(x_batch, d_batch)
train_loss_list.append(loss)
if (i+1) % plot_interval == 0:
accr_train = network.accuracy(x_train, d_train)
accr_test = network.accuracy(x_test, d_test)
accuracies_train.append(accr_train)
accuracies_test.append(accr_test)
print("Generation: " + str(i+1) + ". 正答率(トレーニング) = " + str(accr_train))
print("   : " + str(i+1) + ". 正答率(テスト) = " + str(accr_test))
lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test, label="test set")
plt.legend(loc="lower right")
plt.title("accuracy")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)

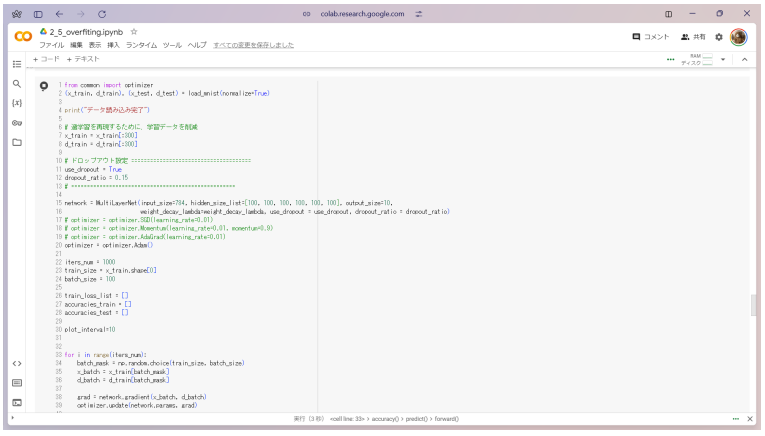

colab.research.google.com
2_5_overfitting.ipynb
accuracy
dropout Ratio = 0.15
training set
test set
count
[try] dropout_ratioの値を変更してみよう
[try] optimizerとdropout_ratioの値を変更してみよう

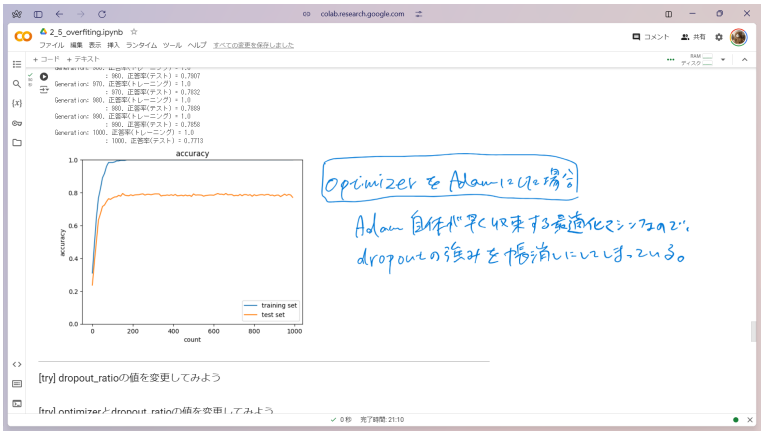

colab.research.google.com
2_5_overfitting.ipynb
from common import optimizer
print("データ読み込み完了")
network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100], output_size=10,
optimizer = optimizer.SGD(learning_rate=0.01)
iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
for i in range(iters_num):

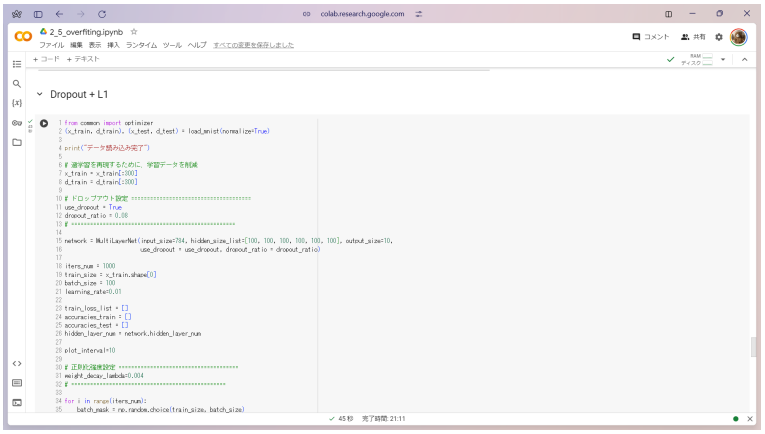

colab.research.google.com
2_5_overfitting.ipynb
accuracy
dropout Ratio = 0.3
training set
test set
count
[try] dropout_ratioの値を変更してみよう
[try] optimizerとdropout_ratioの値を変更してみよう
Dropout + L1

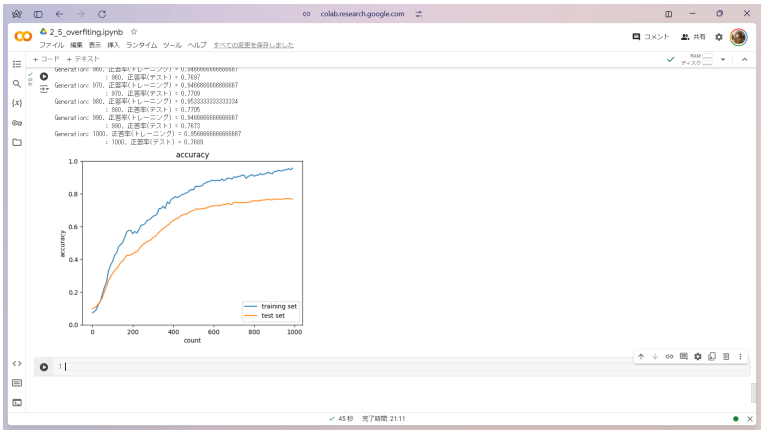

colab.research.google.com
2_5_overfitting.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ
コメント
共有
+ コード + テキスト
from common import optimizer
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)
print("データ読み込み完了")
# 過学習を再現するために、学習データを削減
x_train = x_train[:300]
d_train = d_train[:300]
# ドロップアウト設定 ======
use_dropout = True
dropout_ratio = 0.075
# ====================
network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100], output_size=10,
weight_decay_lambda=weight_decay_lambda, use_dropout = use_dropout, dropout_ratio = dropout_ratio)
optimizer = optimizer.SGD(learning_rate=0.01)
# optimizer = optimizer.Momentum(learning_rate=0.01, momentum=0.9)
# optimizer = optimizer.AdaGrad(learning_rate=0.01)
# optimizer = optimizer.Adam()
iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
train_loss_list = []
accuracies_train = []
accuracies_test = []
plot_interval=10
for i in range(iters_num):
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
d_batch = d_train[batch_mask]
grad = network.gradient(x_batch, d_batch)
optimizer.update(network.params, grad)
実行 (3 秒) <cell line: 33> > accuracy() > predict() > forward()


colab.research.google.com
2_5_overfitting.ipynb
ファイル 編集 表示 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
共有
+ コード + テキスト
Generation: 900, 正答率(テスト) = 0.7727
accuracy: 900, 正答率(トレーニング) = 1.0
Generation: 900, 正答率(テスト) = 0.7729
Generation: 1000, 正答率(トレーニング) = 1.0
accuracy: 1000, 正答率(テスト) = 0.7715
accuracy
count
training set
test set
dropout Ratio = 0.075
無視するニードの数が少ないと早く過学習する
[try] dropout_ratioの値を変更してみよう
[try] optimizerとdropout_ratioの値を変更してみよう
46 秒 完了時間: 21:08

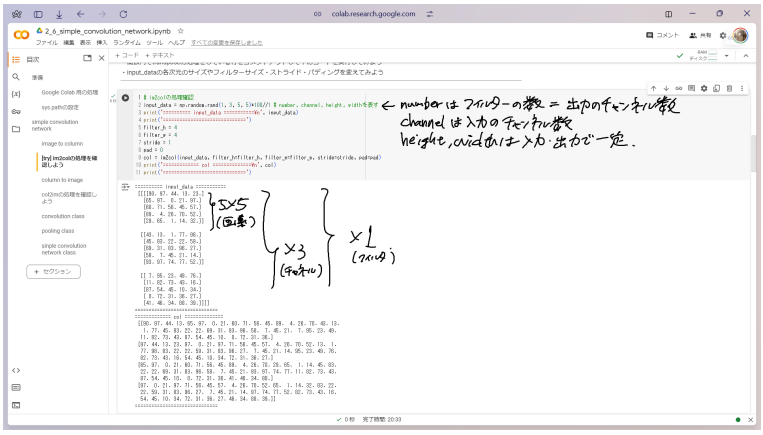

colab.research.google.com
2_5_overfitting.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
コメント
共有
+ コード + テキスト
from common import optimizer
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)
print("データ読み込み完了")
# 過学習を再現するために、学習データを削減
x_train = x_train[:300]
d_train = d_train[:300]
# ドロップアウト設定 ======
use_dropout = True
dropout_ratio = 0.15
# ====================
network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100], output_size=10,
weight_decay_lambda=weight_decay_lambda, use_dropout = use_dropout, dropout_ratio = dropout_ratio)
optimizer = optimizer.SGD(learning_rate=0.01)
# optimizer = optimizer.Momentum(learning_rate=0.01, momentum=0.9)
# optimizer = optimizer.AdaGrad(learning_rate=0.01)
optimizer = optimizer.Adam()
iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
train_loss_list = []
accuracies_train = []
accuracies_test = []
plot_interval=10
for i in range(iters_num):
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
d_batch = d_train[batch_mask]
grad = network.gradient(x_batch, d_batch)
optimizer.update(network.params, grad)
実行 (3 秒) <cell line: 33> > accuracy() > predict() > forward()

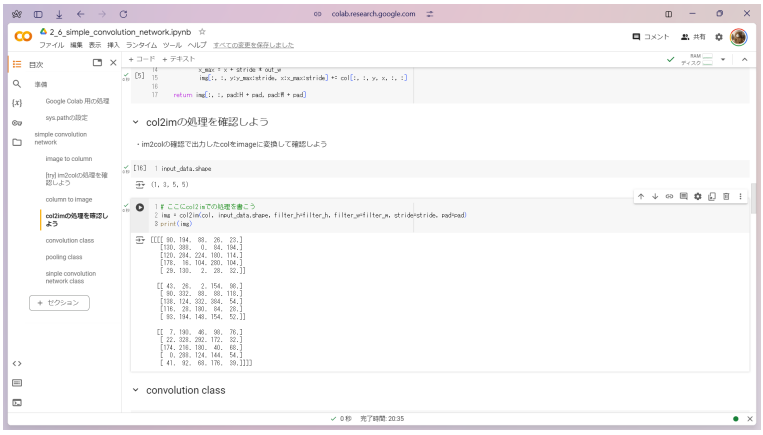

2_5_overfitting.ipynb
accuracy
training set
test set
count
Optimizer を Adam にした場合
Adam 自体が早く収束する最適化システムの為、
dropout の効果を帳消しにしてしまっている。
[try] dropout_ratioの値を変更してみよう
[try] optimizerとdropout_ratioの値を変更してみよう

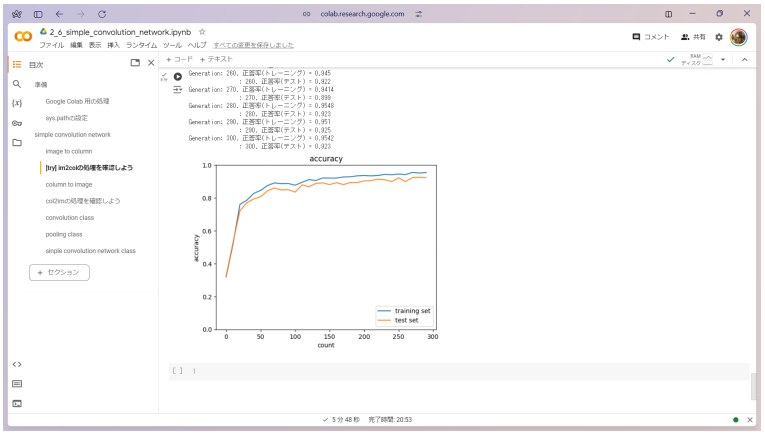

2_5_overfitting.ipynb
Dropout + L1
from common import optimizer
print("データ読み込み完了")
# 過学習を再現するために、学習データを削減
# ドロップアウト設定 ======
use_dropout = True
dropout_ratio = 0.08
# ====
iters_num = 1000
batch_size = 100
learning_rate=0.01
plot_interval=10
# 正則化強度設定 ======
weight_decay_lambda=0.004
# ====
for i in range(iters_num):

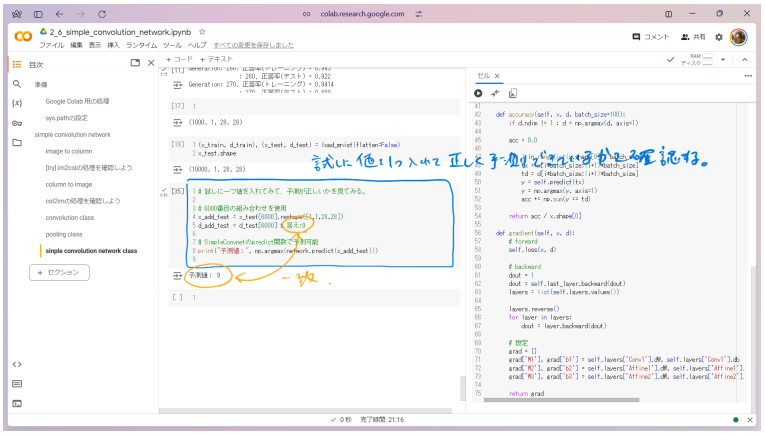

2_5_overfitting.ipynb
accuracy
training set
test set
count

2　2_6_simple_convolution_network

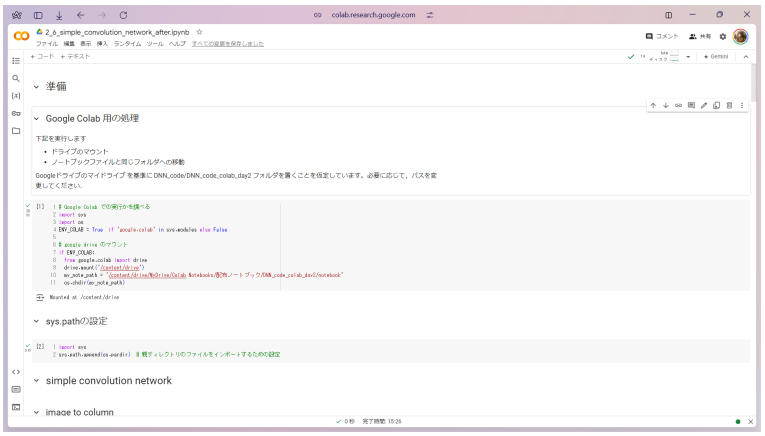


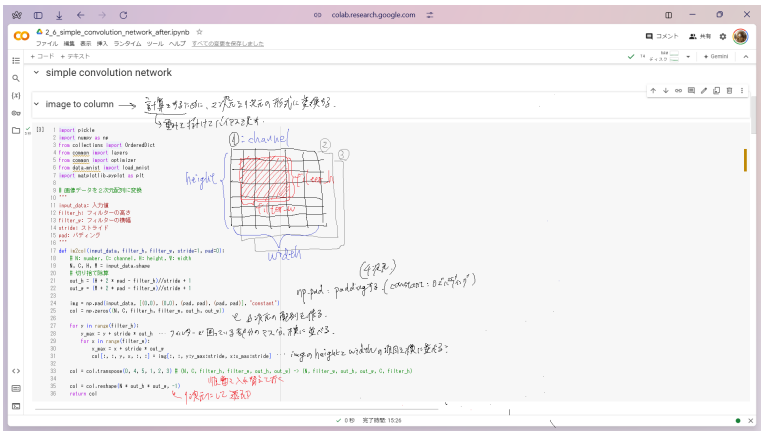


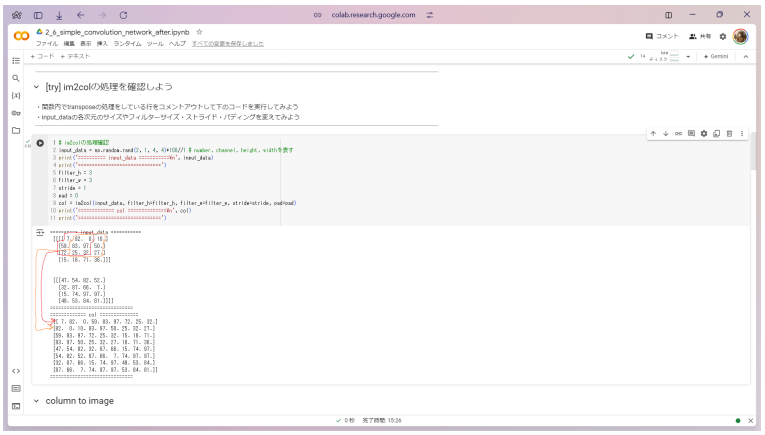

2_6_simple_convolution_network.ipynb
accuracy
training set
test set
count

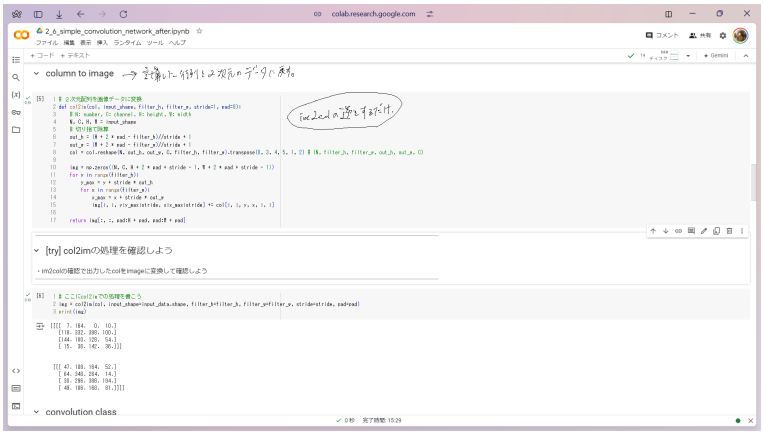

2_6_simple_convolution_network.ipynb
試しに他にも入れて正しく予測できるか確認する。
一致。

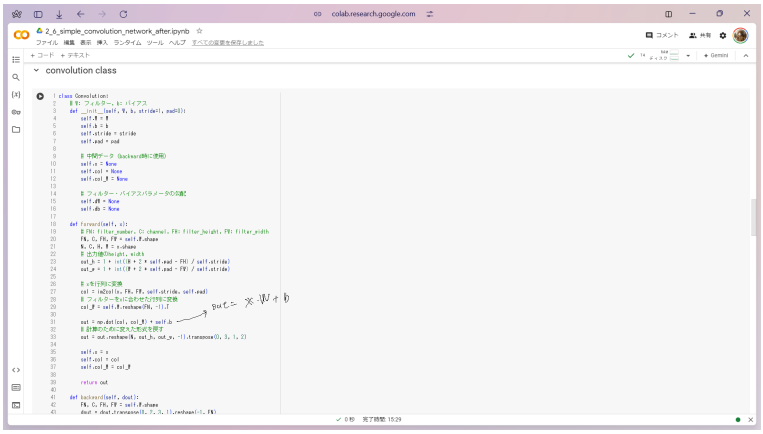
3　2_6_simple_convolution_network_after

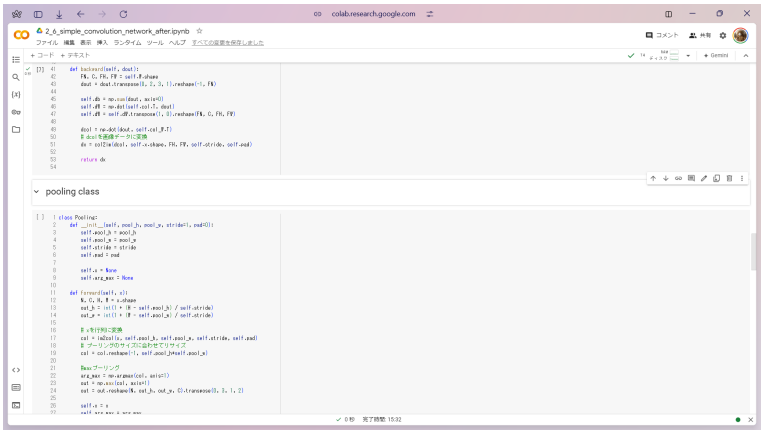


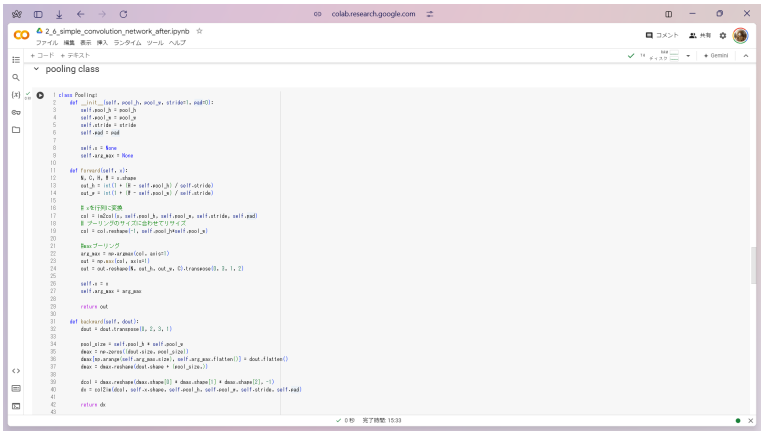


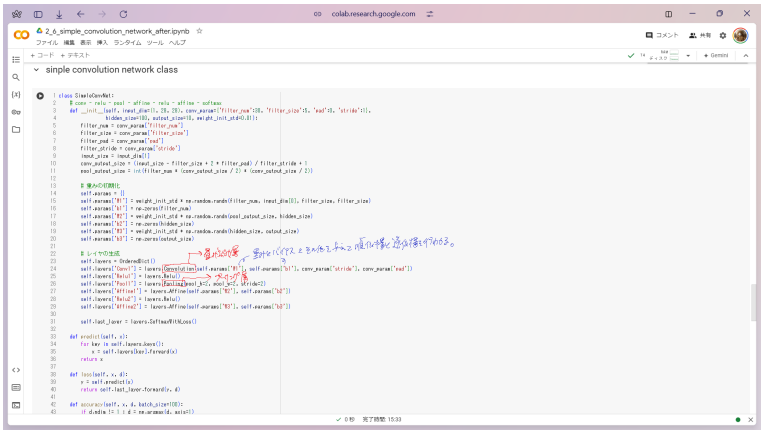


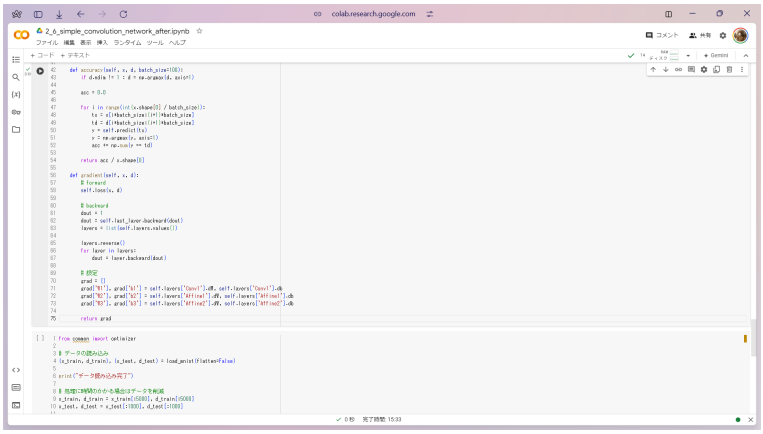

colab.research.google.com
2_6_simple_convolution_network_after.ipynb
column to image
[try] col2imの処理を確認しよう
convolution class

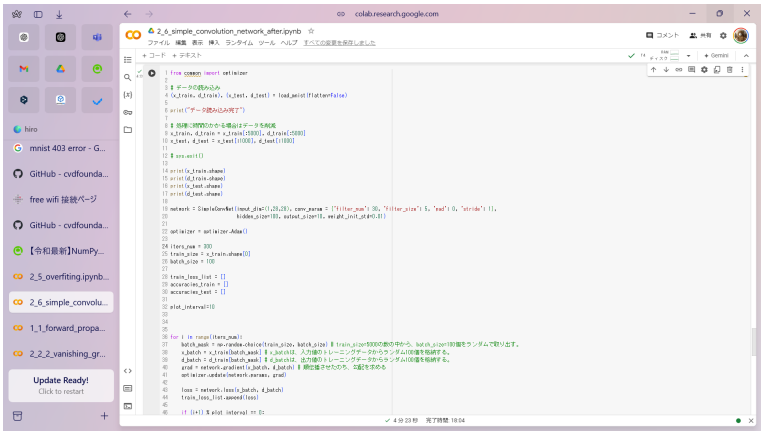

colab.research.google.com
2_6_simple_convolution_network_after.ipynb
convolution class

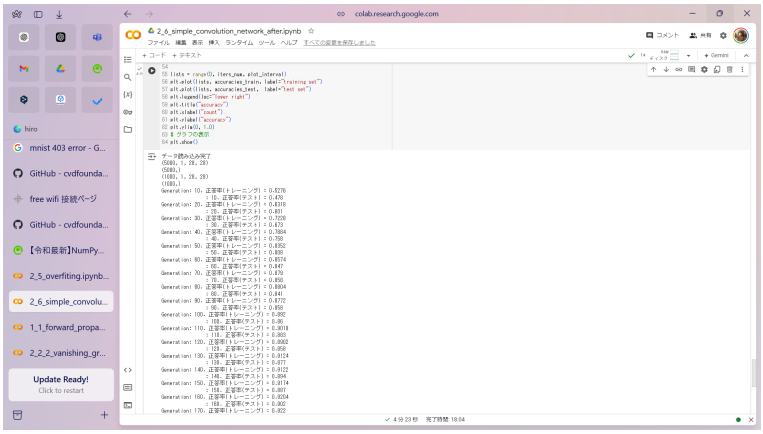

colab.research.google.com
2_6_simple_convolution_network_after.ipynb
pooling class

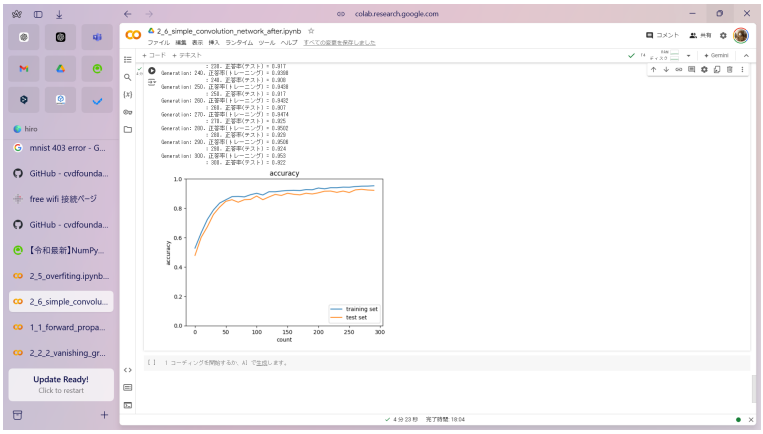

colab.research.google.com
2_6_simple_convolution_network_after.ipynb
pooling class


colab.research.google.com
2_6_simple_convolution_network_after.ipynb
sinple convolution network class

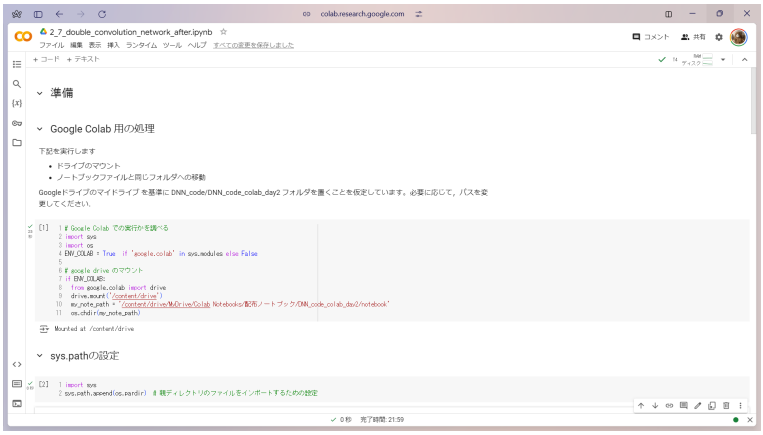

colab.research.google.com
2_6_simple_convolution_network_after.ipynb

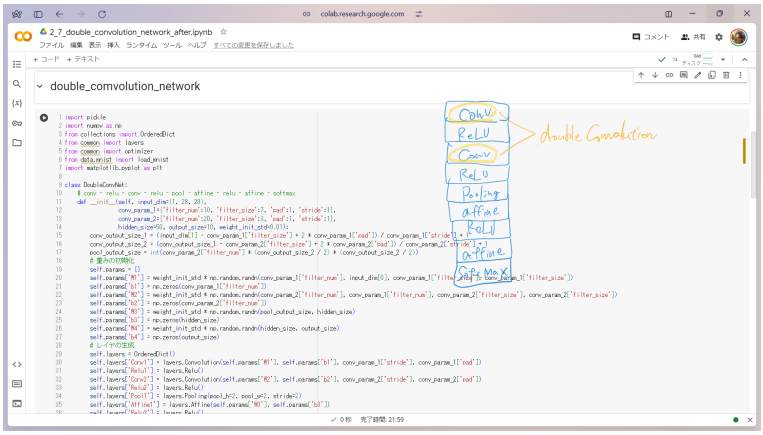


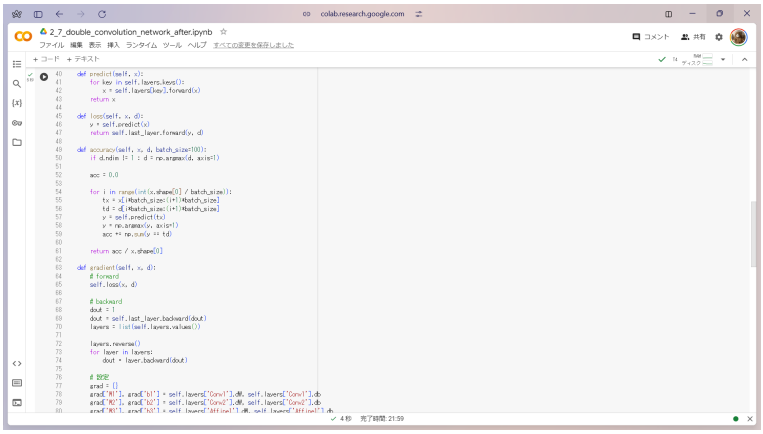


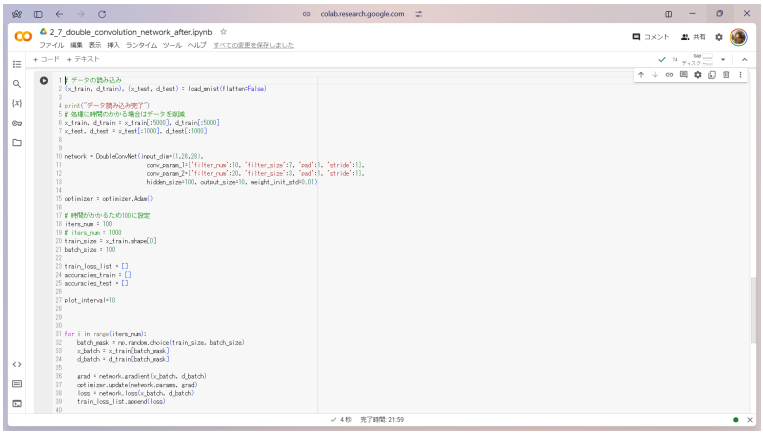


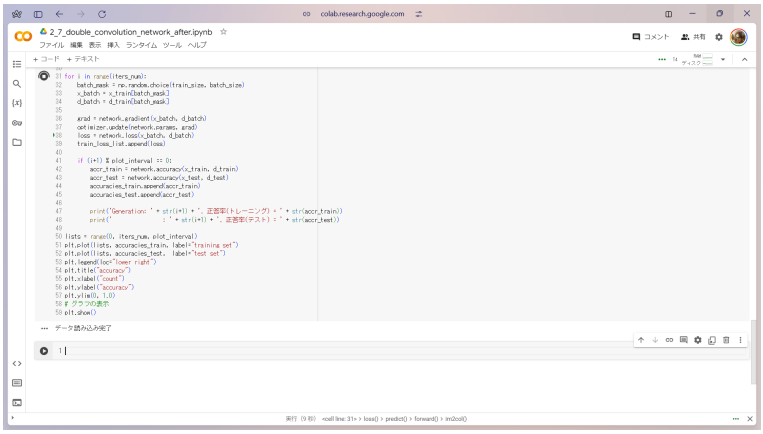
4 2_7_double_convolution_network

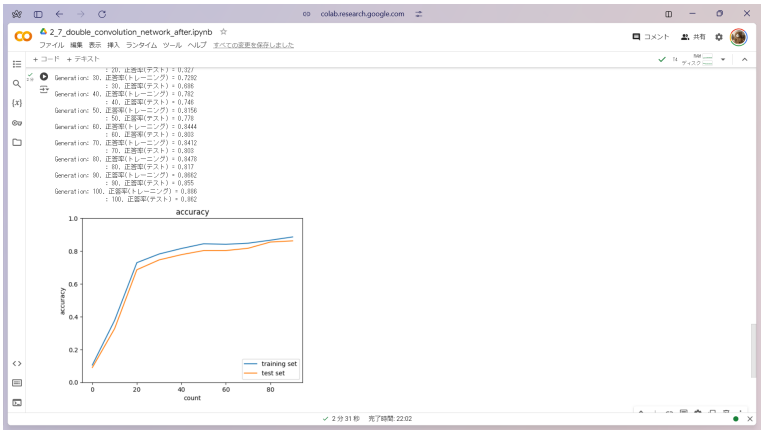







colab.research.google.com
2_7_double_convolution_network_after.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
コメント 共有
+ コード + テキスト
# データの読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(flatten=False)
print("データ読み込み完了")
# 処理に時間のかかる場合はデータを削減
x_train, d_train = x_train[:9000], d_train[:9000]
x_test, d_test = x_test[:1000], d_test[:1000]
network = DoubleConvNet(input_dim=(1,28,28),
conv_param_1={'filter_num':10, 'filter_size':7, 'pad':3, 'stride':1},
conv_param_2={'filter_num':20, 'filter_size':5, 'pad':3, 'stride':1},
hidden_size=100, output_size=10, weight_init_std=0.01)
optimizer = optimizer.Adam()
# 時間がかかるため100に設定
iters_num = 100
# iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
train_loss_list = []
accuracies_train = []
accuracies_test = []
plot_interval=10
for i in range(iters_num):
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
d_batch = d_train[batch_mask]
grad = network.gradient(x_batch, d_batch)
optimizer.update(network.params, grad)
loss = network.loss(x_batch, d_batch)
train_loss_list.append(loss)
4 秒 完了時間: 21:59


colab.research.google.com
2_7_double_convolution_network_after.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ
コメント 共有
+ コード + テキスト
for i in range(iters_num):
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
d_batch = d_train[batch_mask]
grad = network.gradient(x_batch, d_batch)
optimizer.update(network.params, grad)
loss = network.loss(x_batch, d_batch)
train_loss_list.append(loss)
if (i+1) % plot_interval == 0:
accr_train = network.accuracy(x_train, d_train)
accr_test = network.accuracy(x_test, d_test)
accuracies_train.append(accr_train)
accuracies_test.append(accr_test)
print("Generation: " + str(i+1) + ". 正答率(トレーニング) = " + str(accr_train))
print("   : " + str(i+1) + ". 正答率(テスト) = " + str(accr_test))
lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test, label="test set")
plt.legend(loc="lower right")
plt.title("accuracy")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
# グラフの表示
plt.show()
データ読み込み完了
実行 (9 秒) <cell line: 31> > loss() > predict() > forward() > im2col()


colab.research.google.com
2_7_double_convolution_network_after.ipynb
ファイル 編集 表示 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 共有
+ コード + テキスト
: 20, 正答率(テスト) = 0.327
Generation: 30. 正答率(トレーニング) = 0.7292
: 30, 正答率(テスト) = 0.686
Generation: 40. 正答率(トレーニング) = 0.782
: 40, 正答率(テスト) = 0.746
Generation: 50. 正答率(トレーニング) = 0.8156
: 50, 正答率(テスト) = 0.778
Generation: 60. 正答率(トレーニング) = 0.8444
: 60, 正答率(テスト) = 0.808
Generation: 70. 正答率(トレーニング) = 0.8412
: 70, 正答率(テスト) = 0.803
Generation: 80. 正答率(トレーニング) = 0.8478
: 80, 正答率(テスト) = 0.817
Generation: 90. 正答率(トレーニング) = 0.8662
: 90, 正答率(テスト) = 0.855
Generation: 100. 正答率(トレーニング) = 0.886
: 100, 正答率(テスト) = 0.862
accuracy
accuracy
count
training set
test set
2 分 31 秒 完了時間: 22:02