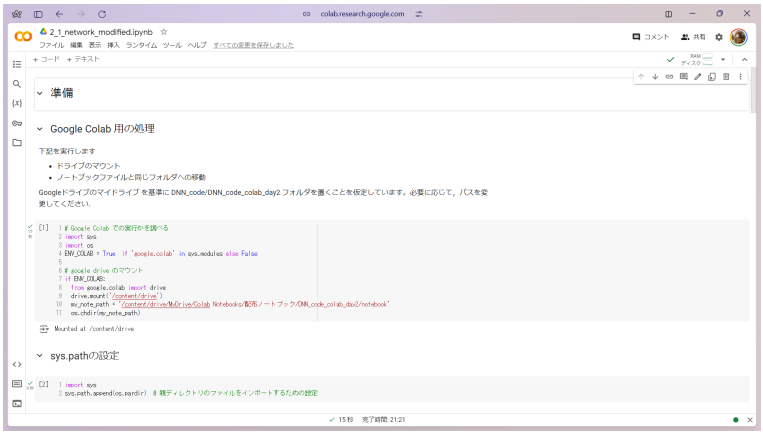
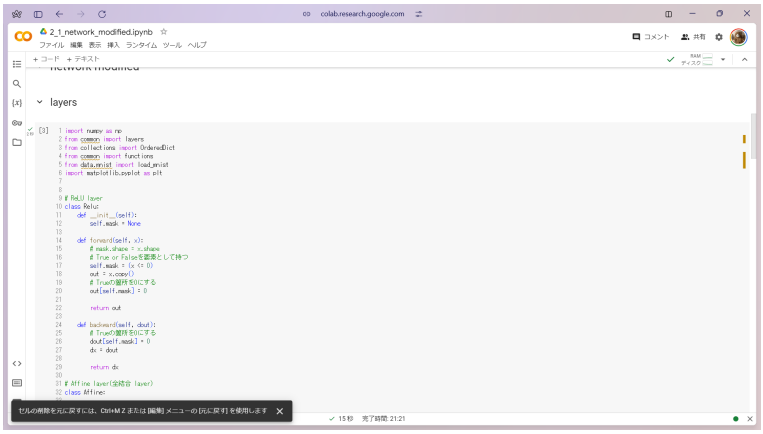
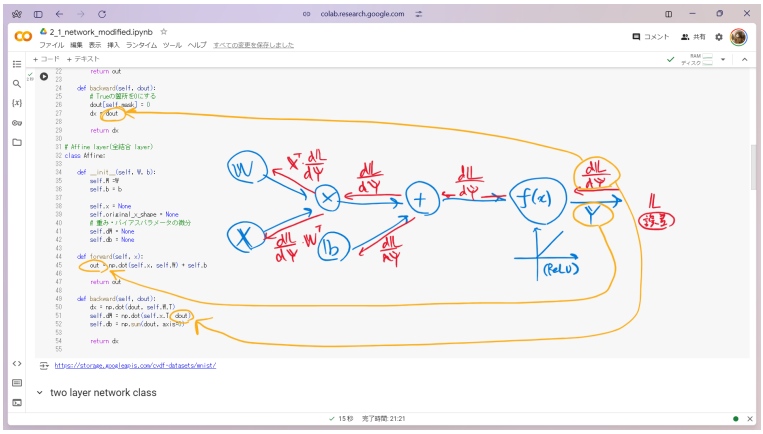
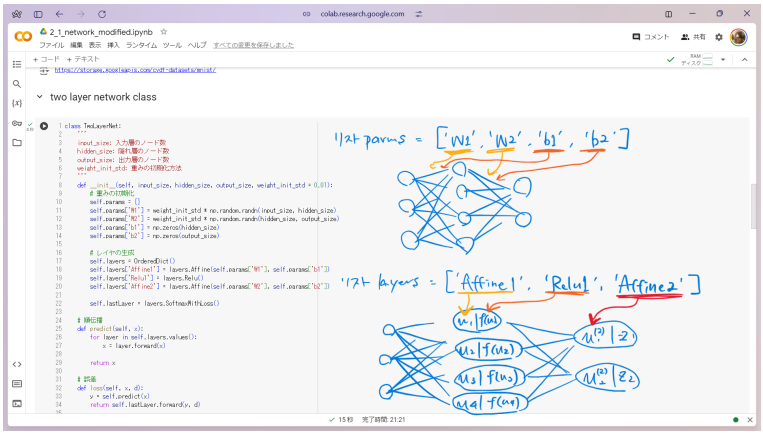
深層学習 day2 実装演習キャプチャ 1

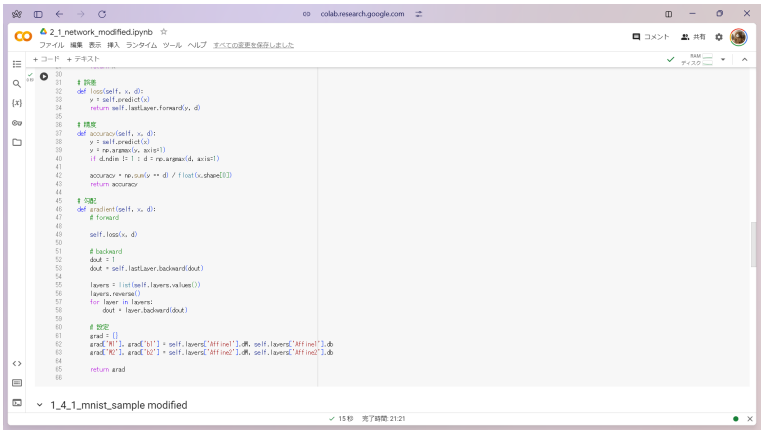
1　2_1_network_modified

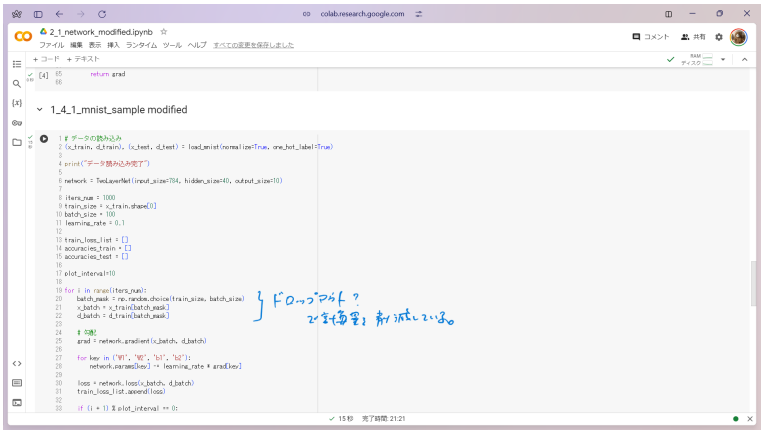


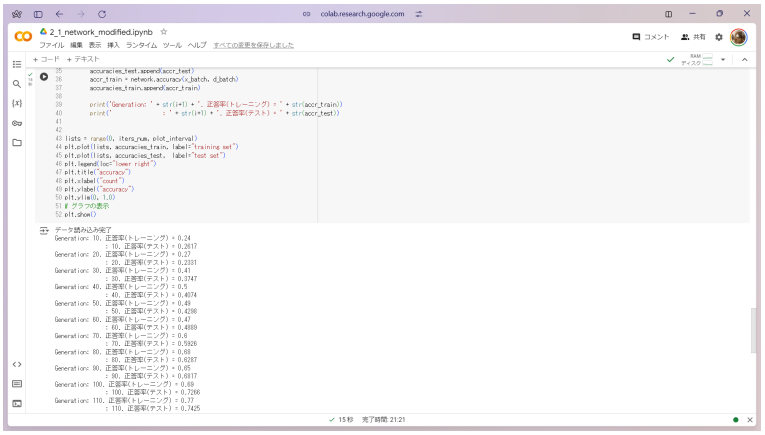


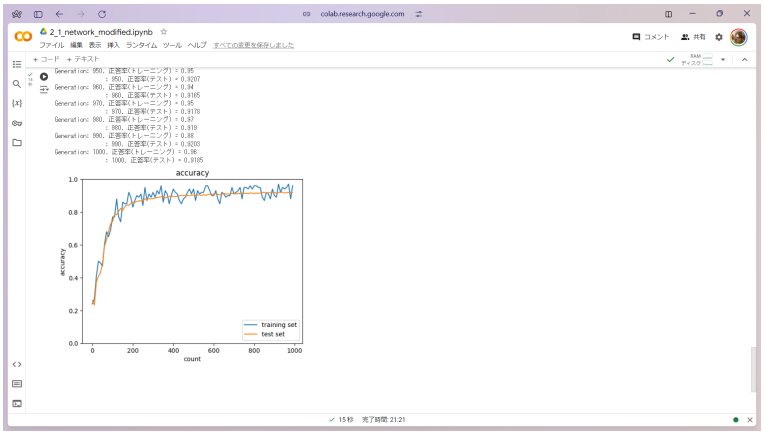



colab.research.google.com
2_1_network_modified.ipynb
two layer network class
1class TwoLayerNet:
2 """
3 input_size: 入力層のノード数
4 hidden_size: 隠れ層のノード数
5 output_size: 出力層のノード数
6 weight_init_std: 重みの初期化方法
7 """
8 def __init__(self, input_size, hidden_size, output_size, weight_init_std = 0.01):
9 # 重みの初期化
10 self.params = {}
11 self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
12 self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
13 self.params['b1'] = np.zeros(hidden_size)
14 self.params['b2'] = np.zeros(output_size)
15
16 # レイヤの生成
17 self.layers = OrderedDict()
18 self.layers['Affine1'] = layers.Affine(self.params['W1'], self.params['b1'])
19 self.layers['Relu1'] = layers.Relu()
20 self.layers['Affine2'] = layers.Affine(self.params['W2'], self.params['b2'])
21
22 self.lastLayer = layers.SoftmaxWithLoss()
23
24 # 推論
25 def predict(self, x):
26 for layer in self.layers.values():
27 x = layer.forward(x)
28
29 return x
30
31 # 誤差
32 def loss(self, x, d):
33 y = self.predict(x)
34 return self.lastLayer.forward(y, d)
self.params = ['W1', 'W2', 'b1', 'b2']
self.layers = ['Affine1', 'Relu1', 'Affine2']
15秒 完了時間 21:21

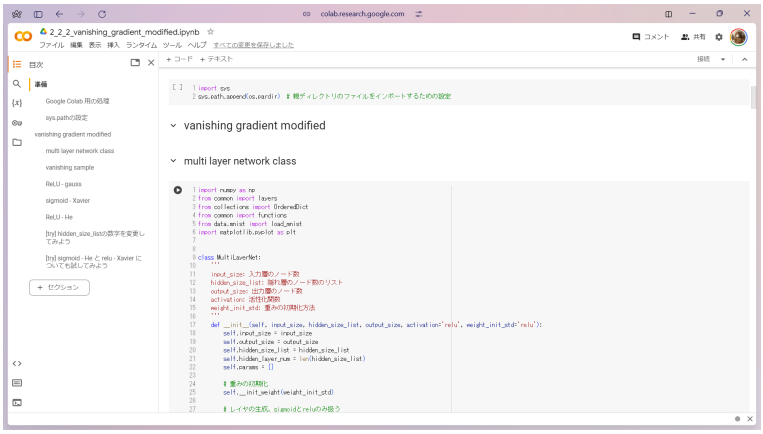

colab.research.google.com
2_1_network_modified.ipynb
31 # 誤差
32 def loss(self, x, d):
33 y = self.predict(x)
34 return self.lastLayer.forward(y, d)
35
36 # 精度
37 def accuracy(self, x, d):
38 y = self.predict(x)
39 y = np.argmax(y, axis=1)
40 if d.ndim != 1 : d = np.argmax(d, axis=1)
41
42 accuracy = np.sum(y == d) / float(x.shape[0])
43 return accuracy
44
45 # 勾配
46 def gradient(self, x, d):
47 # forward
48
49 self.loss(x, d)
50
51 # backward
52 dout = 1
53 dout = self.lastLayer.backward(dout)
54
55 layers = list(self.layers.values())
56 layers.reverse()
57 for layer in layers:
58 dout = layer.backward(dout)
59
60 # 設定
61 grad = {}
62 grad['W1'], grad['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
63 grad['W2'], grad['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
64
65 return grad
66
1_4_1_mnist_sample modified
15秒 完了時間 21:21

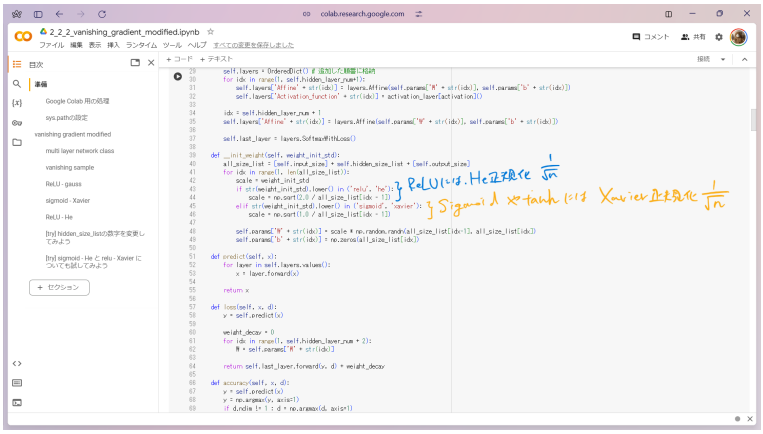

colab.research.google.com
2_1_network_modified.ipynb
[4] 65 return grad
66
1_4_1_mnist_sample modified
1 # データの読み込み
2 (x_train, d_train), (x_test, d_test) = load_mnist(normalize=True, one_hot_label=True)
3
4 print("データ読み込み完了")
5
6 network = TwoLayerNet(input_size=784, hidden_size=40, output_size=10)
7
8 iters_num = 1000
9 train_size = x_train.shape[0]
10 batch_size = 100
11 learning_rate = 0.1
12
13 train_loss_list = []
14 accuracies_train = []
15 accuracies_test = []
16
17 plot_interval=10
18
19 for i in range(iters_num):
20 batch_mask = np.random.choice(train_size, batch_size)
21 x_batch = x_train[batch_mask]
22 d_batch = d_train[batch_mask]
23
24 # 勾配
25 grad = network.gradient(x_batch, d_batch)
26
27 for key in ('W1', 'W2', 'b1', 'b2'):
28 network.params[key] -= learning_rate * grad[key]
29
30 loss = network.loss(x_batch, d_batch)
31 train_loss_list.append(loss)
32
33 if (i + 1) % plot_interval == 0:
15秒 完了時間 21:21

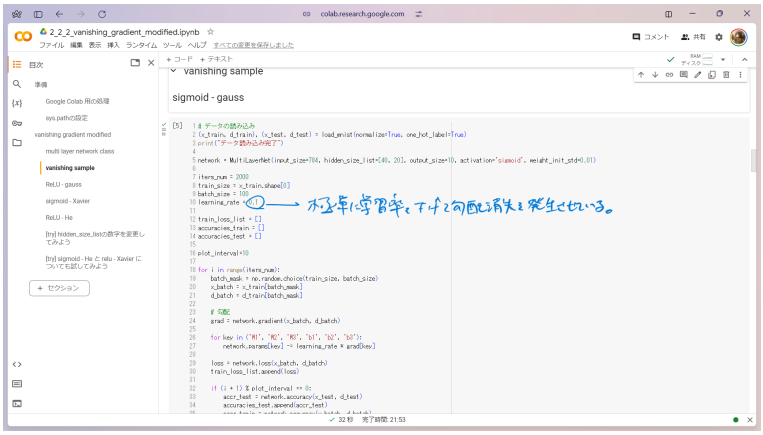

colab.research.google.com
2_1_network_modified.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
コメント
共有
+ コード + テキスト
RAM
ディスク
accuracies_test.append(acc_test)
acc_train = network.accuracy(x_batch, d_batch)
accuracies_train.append(acc_train)
print("Generation: " + str(i+1) + ". 正答率(トレーニング) = " + str(acc_train))
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)
# グラフの表示
plt.show()
データ読み込み完了
Generation: 10. 正答率(トレーニング) = 0.24
                : 10. 正答率(テスト) = 0.2617
Generation: 20. 正答率(トレーニング) = 0.27
                : 20. 正答率(テスト) = 0.2331
Generation: 30. 正答率(トレーニング) = 0.41
                : 30. 正答率(テスト) = 0.3747
Generation: 40. 正答率(トレーニング) = 0.5
                : 40. 正答率(テスト) = 0.4674
Generation: 50. 正答率(トレーニング) = 0.49
                : 50. 正答率(テスト) = 0.4298
Generation: 60. 正答率(トレーニング) = 0.47
                : 60. 正答率(テスト) = 0.4889
Generation: 70. 正答率(トレーニング) = 0.6
                : 70. 正答率(テスト) = 0.5926
Generation: 80. 正答率(トレーニング) = 0.68
                : 80. 正答率(テスト) = 0.6287
Generation: 90. 正答率(トレーニング) = 0.85
                : 90. 正答率(テスト) = 0.8017
Generation: 100. 正答率(トレーニング) = 0.89
                : 100. 正答率(テスト) = 0.7298
Generation: 110. 正答率(トレーニング) = 0.77
                : 110. 正答率(テスト) = 0.7425
✓ 15秒 完了時間: 21:21

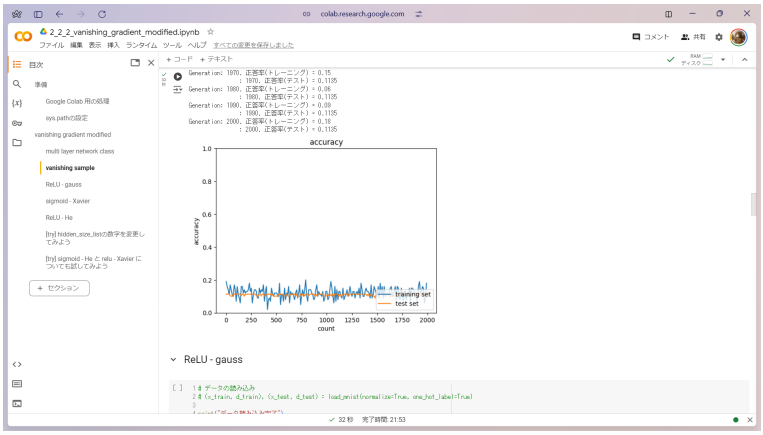

colab.research.google.com
2_1_network_modified.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
コメント
共有
+ コード + テキスト
Generation: 950. 正答率(トレーニング) = 0.95
                : 950. 正答率(テスト) = 0.9207
Generation: 960. 正答率(トレーニング) = 0.94
                : 960. 正答率(テスト) = 0.9105
Generation: 970. 正答率(トレーニング) = 0.95
                : 970. 正答率(テスト) = 0.9178
Generation: 980. 正答率(トレーニング) = 0.97
                : 980. 正答率(テスト) = 0.919
Generation: 990. 正答率(トレーニング) = 0.88
                : 990. 正答率(テスト) = 0.9203
Generation: 1000. 正答率(トレーニング) = 0.96
                : 1000. 正答率(テスト) = 0.9105
accuracy
accuracy
count
training set
test set
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
1000
✓ 15秒 完了時間: 21:21

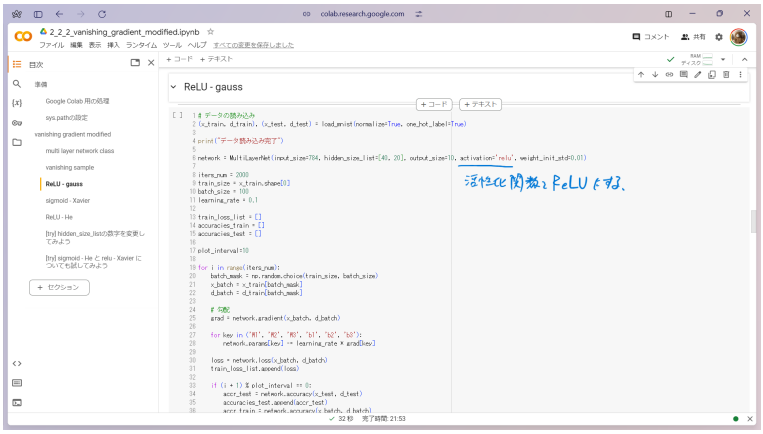
2　2_2_2_vanishing_gradient_modified

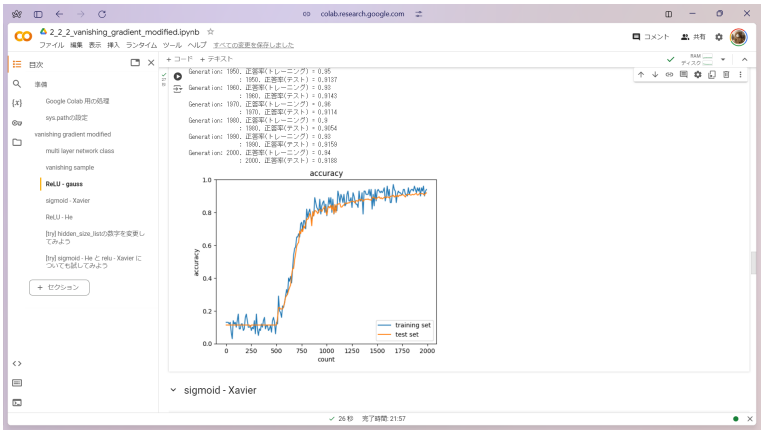


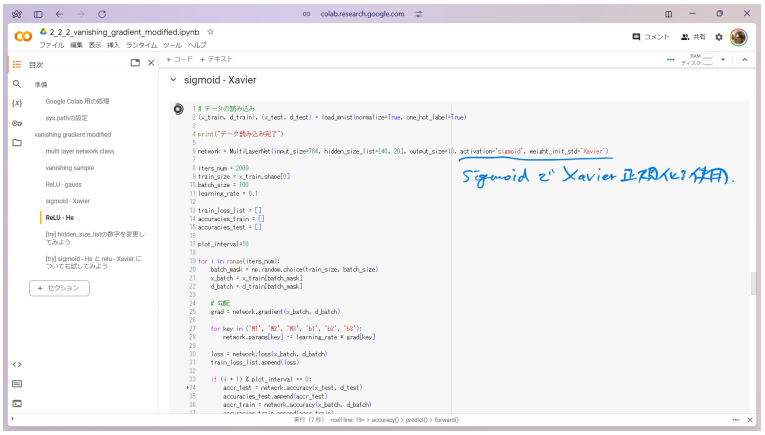


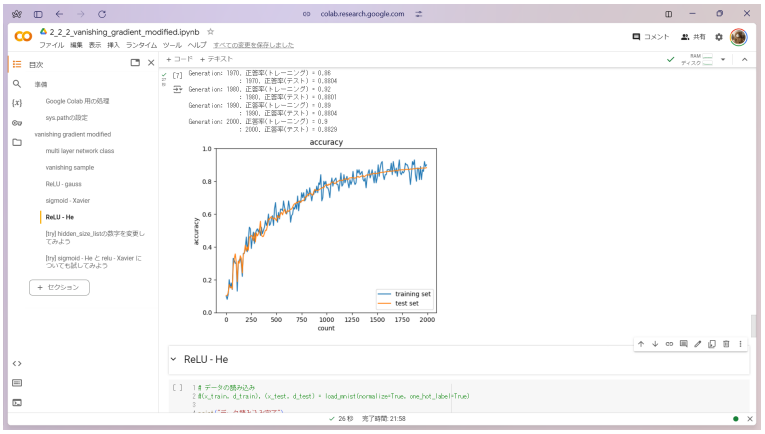


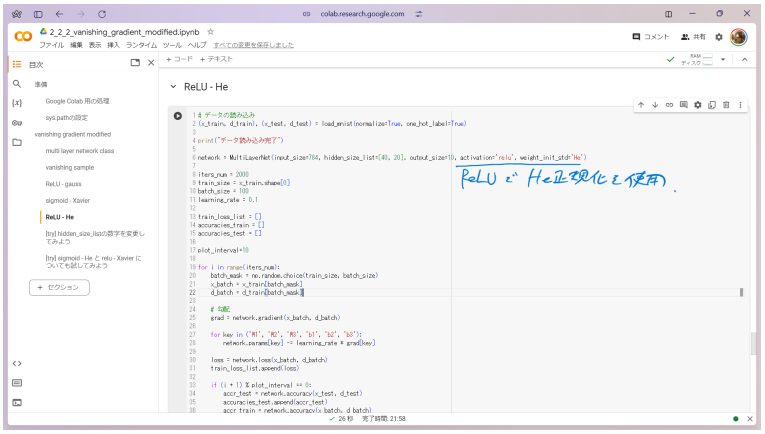

2_2_2_vanishing_gradient_modified.ipynb
accuracy
training set
test set
count
ReLU - gauss

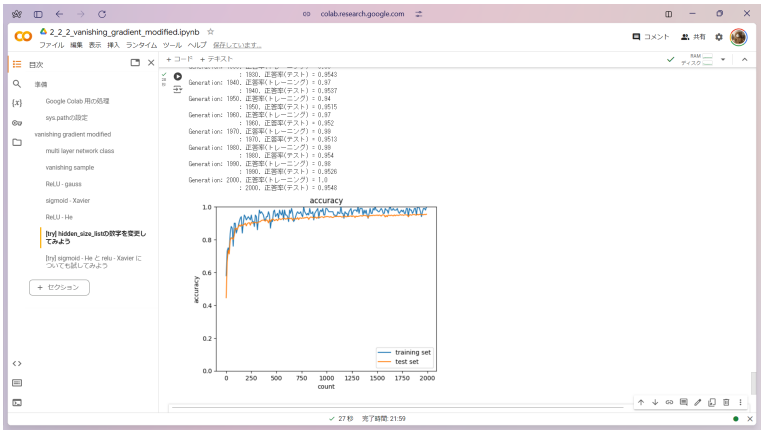

2_2_2_vanishing_gradient_modified.ipynb
ReLU - gauss
活性化関数をReLUとする。


2_2_2_vanishing_gradient_modified.ipynb
accuracy
training set
test set
count
sigmoid - Xavier

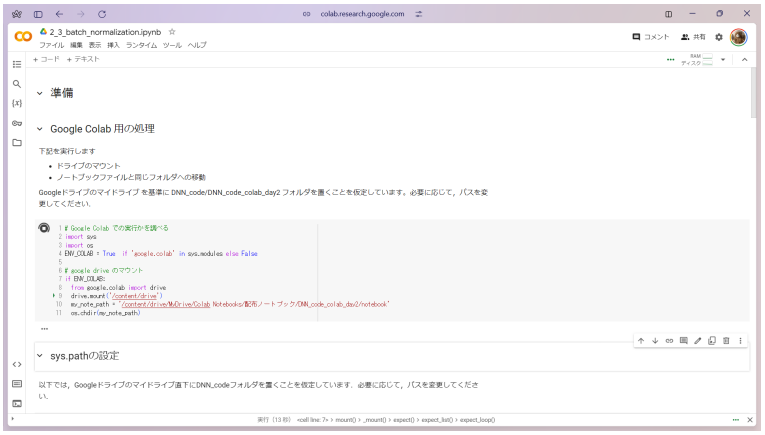

2_2_2_vanishing_gradient_modified.ipynb
sigmoid - Xavier
Sigmoid と Xavier 正規化を使用。

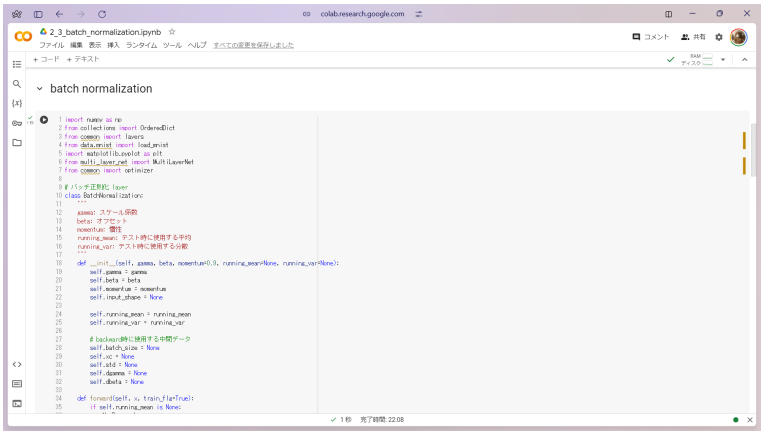

2_2_2_vanishing_gradient_modified.ipynb
accuracy
training set
test set
count
ReLU - He

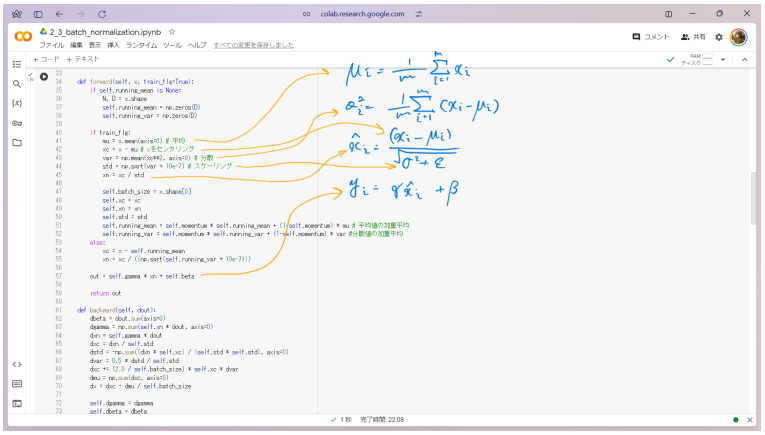

2_2_2_vanishing_gradient_modified.ipynb
ReLU - He
ReLU と He正規化を使用。

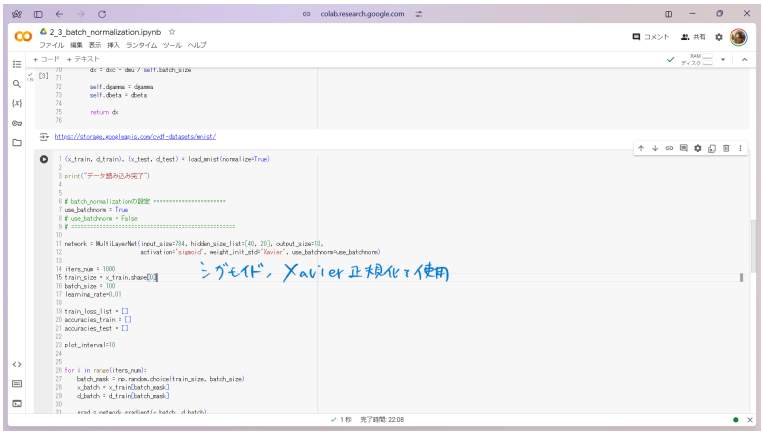

colab.research.google.com
2_2_2_vanishing_gradient_modified.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ 保存しています...
コメント
共有
目次
準備
Google Colab 用の処理
sys.path の設定
vanishing gradient modified
multi layer network class
vanishing sample
ReLU - gauss
sigmoid - Xavier
ReLU - He
[try] hidden_size_list の数字を変更してみよう
[try] sigmoid - He と relu - Xavier についても試してみよう
＋ セクション
＋ コード ＋ テキスト
: 1930, 正答率(テスト) = 0.9543
Generation: 1940, 正答率(トレーニング) = 0.97
: 1940, 正答率(テスト) = 0.9537
Generation: 1950, 正答率(トレーニング) = 0.94
: 1950, 正答率(テスト) = 0.9515
Generation: 1960, 正答率(トレーニング) = 0.97
: 1960, 正答率(テスト) = 0.952
Generation: 1970, 正答率(トレーニング) = 0.99
: 1970, 正答率(テスト) = 0.9513
Generation: 1980, 正答率(トレーニング) = 0.98
: 1980, 正答率(テスト) = 0.954
Generation: 1990, 正答率(トレーニング) = 0.96
: 1990, 正答率(テスト) = 0.9526
Generation: 2000, 正答率(トレーニング) = 1.0
: 2000, 正答率(テスト) = 0.9548
accuracy
accuracy
count
training set
test set
27 秒 完了時間: 21:59

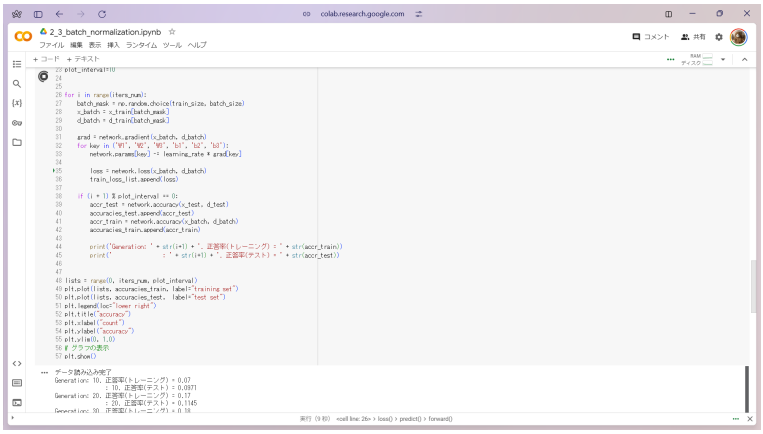
3　2_3_batch_normalization

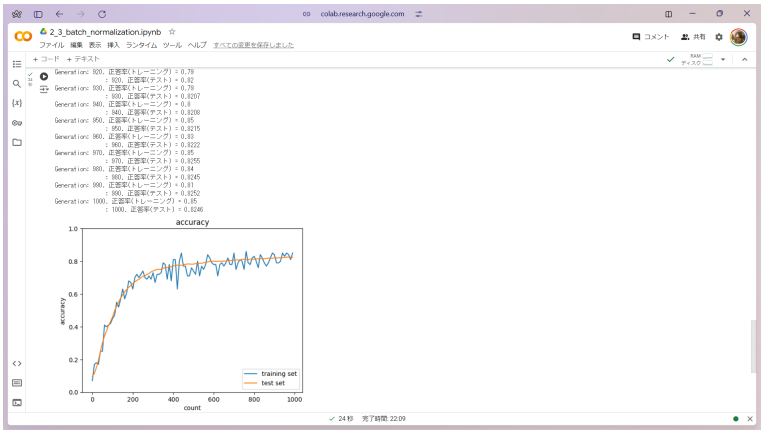


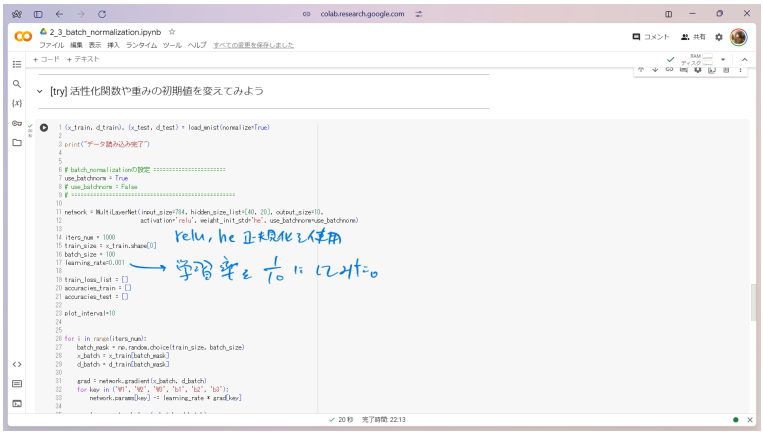


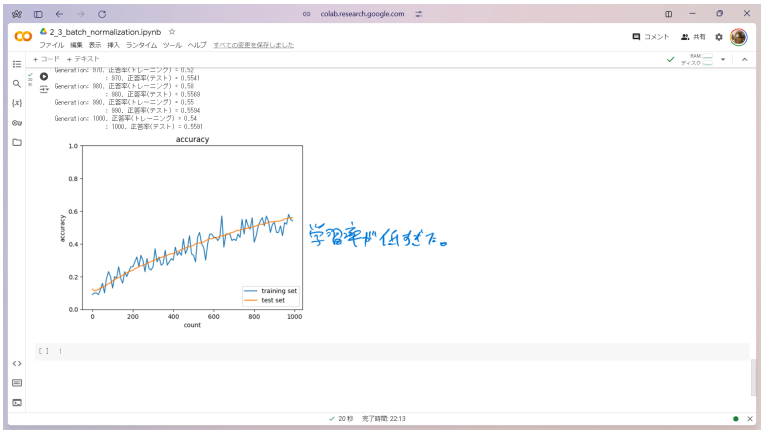


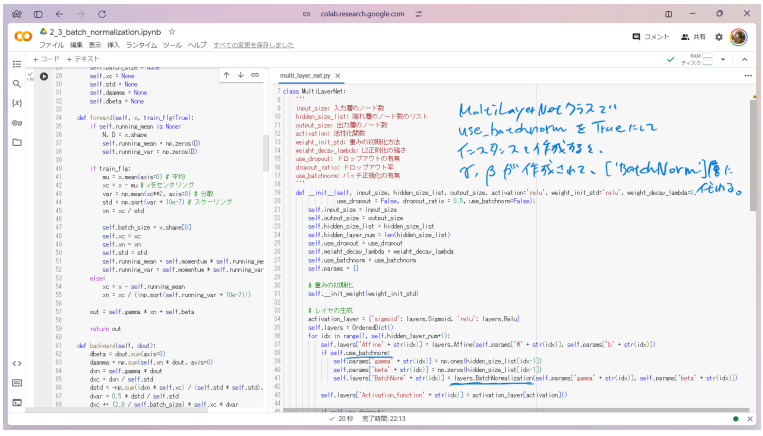

2_3_batch_normalization.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ
シグモイド、Xavier 正規化で使用


2_3_batch_normalization.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ
データ読み込み完了
Generation: 10. 正答率(トレーニング) : 0.07
: 10. 正答率(テスト) : 0.0971
Generation: 20. 正答率(トレーニング) : 0.77
: 20. 正答率(テスト) : 0.1145

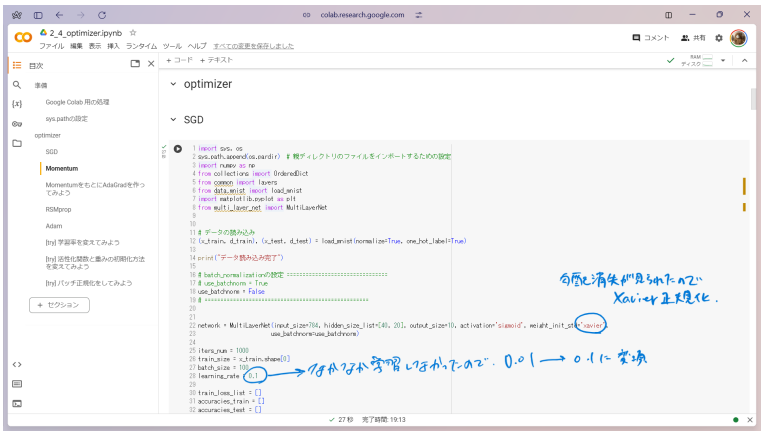

2_3_batch_normalization.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
Generation: 920. 正答率(トレーニング) : 0.79
: 920. 正答率(テスト) : 0.82
Generation: 930. 正答率(トレーニング) : 0.79
: 930. 正答率(テスト) : 0.8207
Generation: 940. 正答率(トレーニング) : 0.8
: 940. 正答率(テスト) : 0.8208
Generation: 950. 正答率(トレーニング) : 0.85
: 950. 正答率(テスト) : 0.8215
Generation: 960. 正答率(トレーニング) : 0.83
: 960. 正答率(テスト) : 0.8222
Generation: 970. 正答率(トレーニング) : 0.85
: 970. 正答率(テスト) : 0.8255
Generation: 980. 正答率(トレーニング) : 0.84
: 980. 正答率(テスト) : 0.8245
Generation: 990. 正答率(トレーニング) : 0.81
: 990. 正答率(テスト) : 0.8252
Generation: 1000. 正答率(トレーニング) : 0.85
: 1000. 正答率(テスト) : 0.8249
accuracy
training set
test set
count

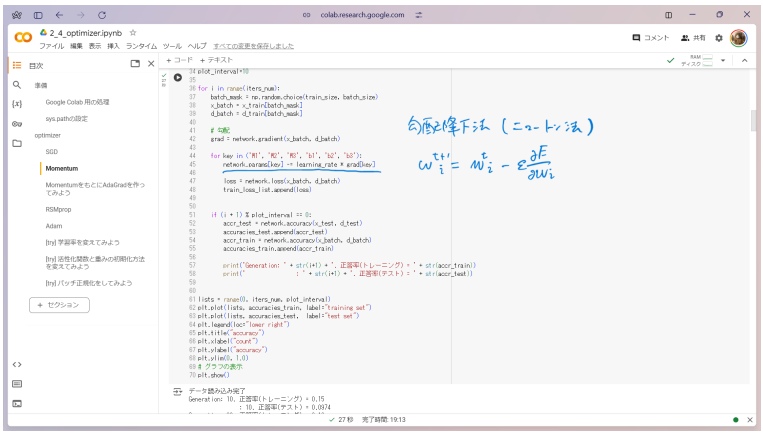

2_3_batch_normalization.ipynb
[try] 活性化関数や重みの初期値を変えてみよう
relu, he 正規化を使用
学習率を $\frac{1}{10}$ にしてみた。


2_3_batch_normalization.ipynb
accuracy
training set
test set
学習率が低すぎた。

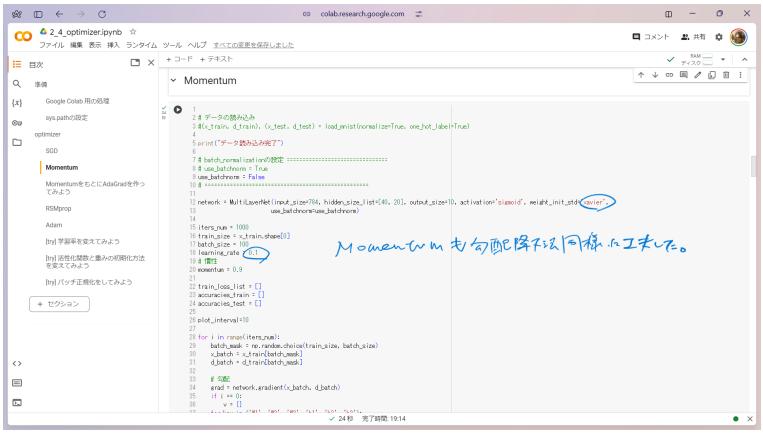

2_3_batch_normalization.ipynb
multi_layer_net.py
MultiLayerNetクラスで use_batchnorm を True にして インスタンスを作成すると、 γ, β が作成され、['BatchNorm']層に 格納。

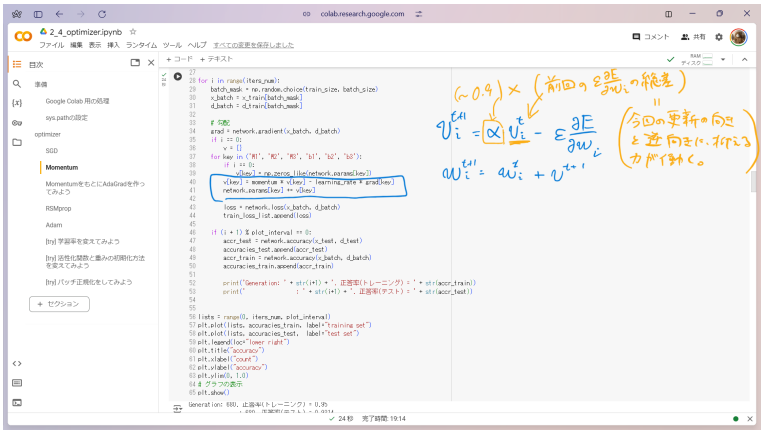
4　2_4_optimizer

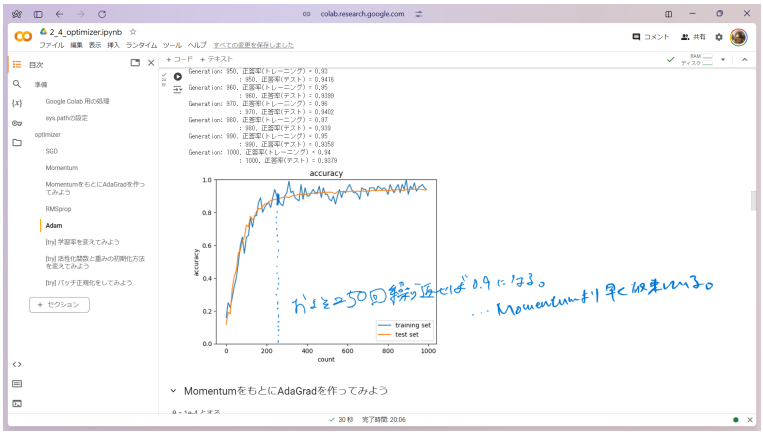


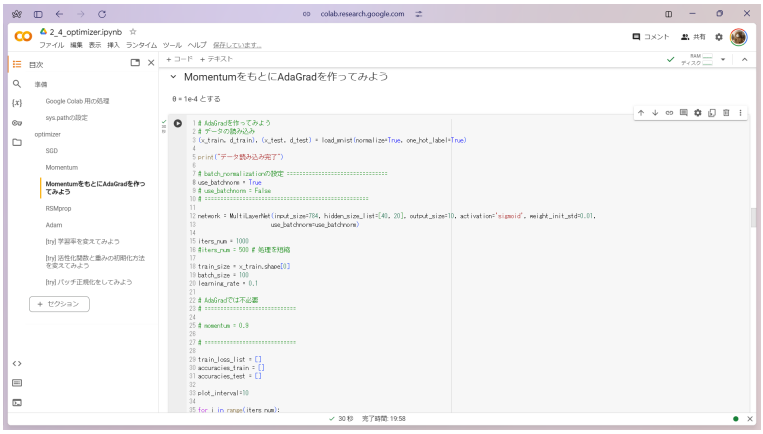


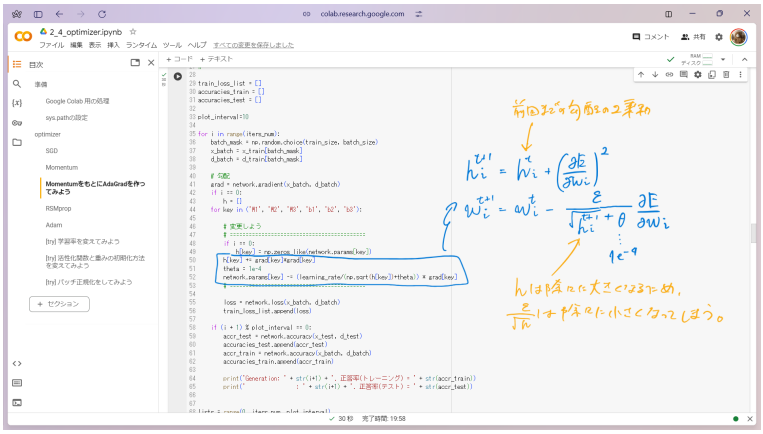

2_4_optimizer.ipynb
Momentum
accuracy
training set
test set
count
最初は、精度0.1~0.2で停滞していたが、
正規化や学習率を調整することで
0.7くらいまで上昇

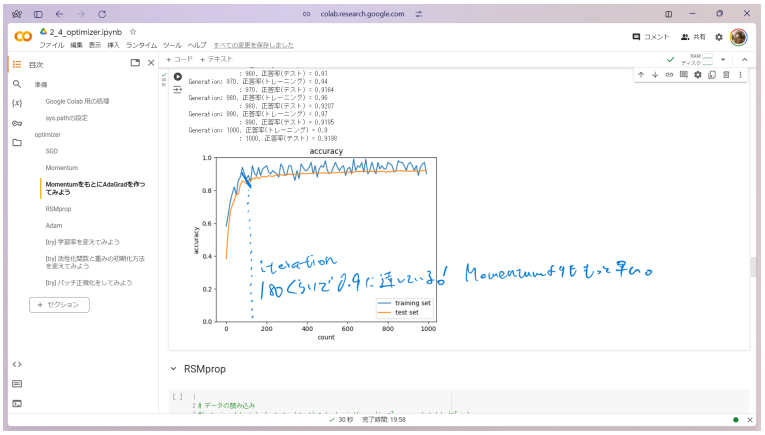

2_4_optimizer.ipynb
Momentum
Momentumも勾配降下法同様に工夫した。

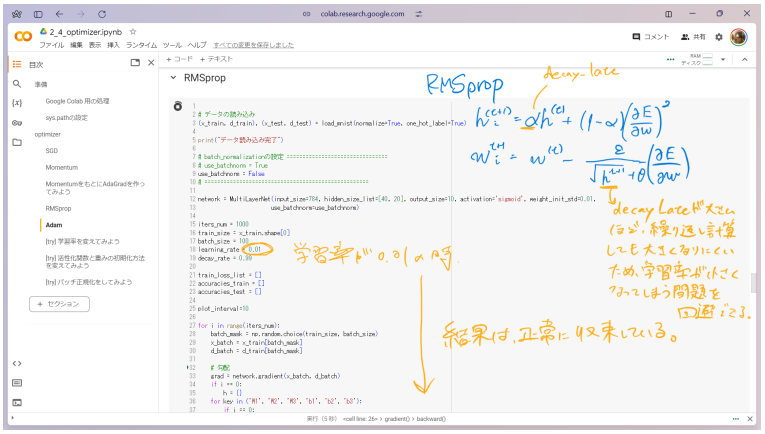

2_4_optimizer.ipynb
(~0.9) × (前回の重みの値)
(今回の更新の向きと逆向きに、力がかかる)

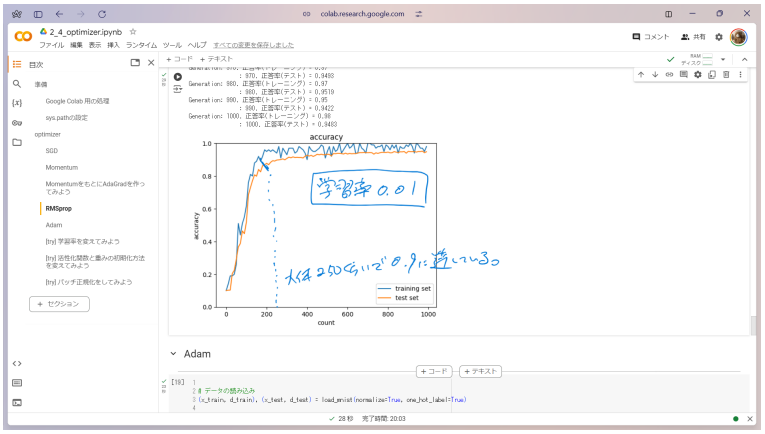

2_4_optimizer.ipynb
accuracy
training set
test set
count
Momentumをもとに AdaGradを作ってみよう

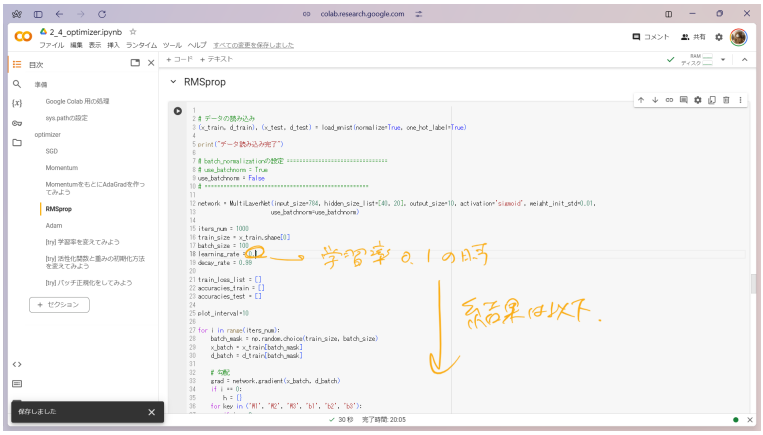

2_4_optimizer.ipynb
Momentumをもとに AdaGradを作ってみよう

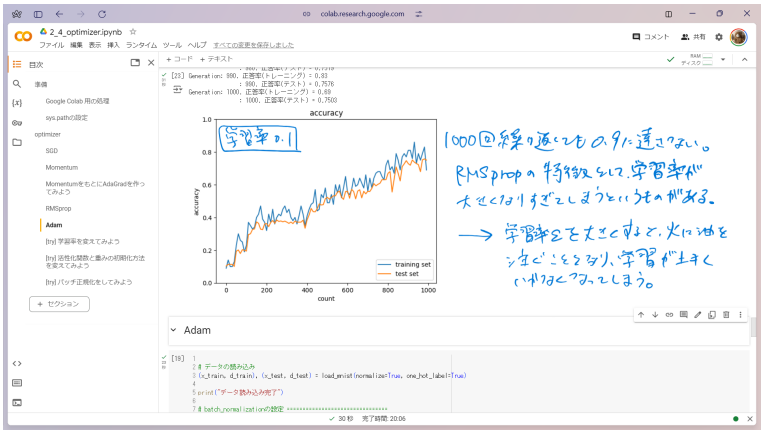

2_4_optimizer.ipynb

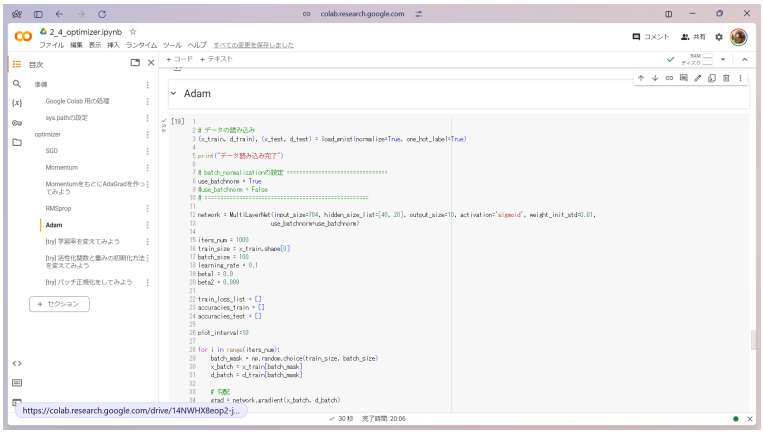

2_4_optimizer.ipynb
accuracy
training set
test set
count
RSMprop
Momentumをもとに作ってみよう
RMSprop
Adam

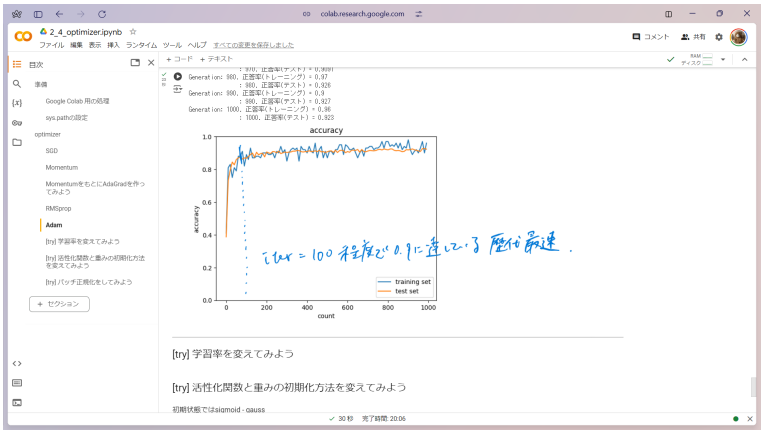

2_4_optimizer.ipynb
RMSprop
learning_rate = 0.01
decay_rate = 0.99
accuracy
training set
test set
count
Adam


2_4_optimizer.ipynb
accuracy
training set
test set
count
RMSprop
Adam


RMSprop
学習率0.1の時
結果は以下.


accuracy
学習率0.1
training set
test set
count
1000回繰り返しても0.9に達さない。
RMSpropの特徴として学習率が
大きくなりすぎてしまうというものがある。
→ 学習率を大きくすると、火に油を
注ぐことになり、学習が上手く
いかなくなってしまう。
Adam


Adam


colab.research.google.com
2_4_optimizer.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
目次
準備
Google Colab 用の処理
sys.pathの設定
optimizer
SGD
Momentum
Momentumをもとに AdaGrad を作ってみよう
RMSprop
Adam
[try] 学習率を変えてみよう
[try] 活性化関数と重みの初期化方法を変えてみよう
[try] バッチ正規化をしてみよう
＋ セクション
＋ コード ＋ テキスト
Generation: 980, 正答率(トレーニング) = 0.97
: 980, 正答率(テスト) = 0.926
Generation: 990, 正答率(トレーニング) = 0.9
: 990, 正答率(テスト) = 0.927
Generation: 1000, 正答率(トレーニング) = 0.96
: 1000, 正答率(テスト) = 0.923
accuracy
accuracy
count
training set
test set
lr = 100 程度で 0.9 に達している 歴が最速.
[try] 学習率を変えてみよう
[try] 活性化関数と重みの初期化方法を変えてみよう
初期状態ではsigmoid - gauss
30 秒 完了時間: 20:06