

深層学習 day2 実装演習キャプチャ 1

1 2_1_network_modified

This screenshot shows the initial setup phase in Google Colab. The code cell contains imports for os, collections, and numpy, along with environment variable definitions for GOOGLE_COLAB and GOOGLE_DRIVE. It also includes code to handle Google Drive mounting and setting up a notebook path. A note at the bottom indicates that the code is intended for Google Colab users.

```
[1]: 1 # Google Colab での実行かを調べる
2 import os
3 import tensorflow as tf
4 GOOGLE_COLAB = True if 'google.colab' in sys.modules else False
5
6 # Google Drive のマウント
7 if GOOGLE_COLAB:
8     from google.colab import drive
9     drive.mount('/content/drive')
10    my_notebook_path = "/content/drive/MyDrive/Colab Notebooks/配布ノック/DNN_code_colab/day2/notebook"
11    os.chdir(my_notebook_path)
12
13 MOUNTED_AT /content/drive
```

下記を実行します
• ドライブのマウント
• ノートブックファイルと同じフォルダへの移動
Googleドライブのマウントを基準にDNN_code/DNN_code_colab/day2 フォルダを置くことを仮定しています。必要に応じて、パスを変更してください。

```
[1]: 1 import tensorflow as tf
2
3 from collections import OrderedDict
4 from typing import Any, Dict, List, Tuple
5 from dataclasses import replace
6 import tensorflow as tf
7 import tensorflow.keras as tfk
```

This screenshot shows the definition of the 'Mfline' class within a 'layers' module. The class inherits from 'tf.keras.layers.Layer'. It has an __init__ method that initializes weights W and bias b. The forward pass (forward) method performs matrix multiplication between input x and weight W, adds bias b, and applies a ReLU activation function. The backward pass (backward) method calculates gradients for x and b.

```
[1]: 1 import tensorflow as tf
2 from tensorflow import keras
3 from collections import OrderedDict
4 from typing import Any, Dict, List, Tuple
5 from dataclasses import replace
6 import tensorflow as tf
7 import tensorflow.keras as tfk
8
9
10 class Mfline(tfk.layers.Layer):
11     def __init__(self, W, b):
12         self.W = None
13         self.b = None
14
15     def forward(self, x):
16         if not isinstance(x, tf.Tensor):
17             raise ValueError("x must be a tensor")
18         if not isinstance(W, tf.Tensor):
19             raise ValueError("W must be a tensor")
20         if not isinstance(b, tf.Tensor):
21             raise ValueError("b must be a tensor")
22
23         return tf.matmul(x, self.W) + self.b
24
25     def backward(self, dout):
26         if not isinstance(dout, tf.Tensor):
27             raise ValueError("dout must be a tensor")
28         if not self.b is None:
29             dx = tf.matmul(dout, self.W.T)
30             db = tf.reduce_sum(dout, axis=0)
31         else:
32             dx = tf.matmul(dout, self.W.T)
33             db = None
34
35         dw = tf.matmul(x, dout.T)
36
37         return dx, db, dw
```

This screenshot shows the definition of a 'two_layer_network' class. It contains two 'Mfline' layers, 'l1' and 'l2', and a final 'tf.keras.layers.Dense' layer 'l3'. The forward pass (forward) method takes input x, passes it through l1, then l2, and finally l3. The backward pass (backward) method calculates gradients for x, l1, l2, and l3 based on the gradients from l3. A handwritten diagram on the right illustrates the forward pass flow from input x through the layers l1, l2, and l3 to output y, with intermediate values like W, b, and f(x) labeled.

```
[1]: 1 class two_layer_network(tfk.Model):
2     def __init__(self, l1, l2, l3):
3         super().__init__()
4         self.l1 = l1
5         self.l2 = l2
6         self.l3 = l3
7
8     def forward(self, x):
9         out = self.l1.forward(x)
10        out = self.l2.forward(out)
11        out = self.l3.forward(out)
12
13        return out
14
15    def backward(self, dout):
16        dw1, db1, dx1 = self.l1.backward(dout)
17        dw2, db2, dx2 = self.l2.backward(dx1)
18        dw3, db3, dx3 = self.l3.backward(dx2)
19
20        return dx3, dw1, db1, dw2, db2, dw3, db3
```

```

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        self.layers = OrderedDict()
        self.layers["W1"] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.layers["b1"] = np.zeros(hidden_size)
        self.layers["W2"] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.layers["b2"] = np.zeros(output_size)

        # レイヤー構造
        self.layers["Affine1"] = layers.Affine(self.layers["W1"], self.layers["b1"])
        self.layers["ReLU1"] = layers.ReLU()
        self.layers["Affine2"] = layers.Affine(self.layers["W2"], self.layers["b2"])

        self.lastLayer = layers.Softmax(NLoss)

    def forward(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)
        return x

    def predict(self, x):
        y = self.forward(x)
        return y.argmax(axis=1)

    def loss(self, x, d):
        y = self.predict(x)
        return self.lastLayer.forward(y, d)

    def accuracy(self, x, d):
        y = self.predict(x)
        if d.ndim == 1:
            d = np.argmax(d, axis=1)
        accuracy = np.sum(y == d) / float(x.shape[0])
        return accuracy

    def backward(self, x, d):
        if d.ndim == 1:
            d = np.eye(d.size) * d
        self.loss(x, d).backward()

        if backward:
            dout = 1
            dout *= self.lastLayer.backward(dout)
            for layer in reversed(list(self.layers.values())):
                dout = layer.backward(dout)
            dout *= self.loss(x, d).backward(dout)

        grad = {}
        grad["W1"] = self.layers["W1"] + self.layers["Affine1"].dW * self.layers["Affine1"].db
        grad["b1"] = self.layers["b1"] + self.layers["Affine1"].db * self.layers["Affine1"].dW
        grad["W2"] = self.layers["W2"] + self.layers["Affine2"].dW * self.layers["Affine2"].db
        grad["b2"] = self.layers["b2"] + self.layers["Affine2"].db * self.layers["Affine2"].dW
        return grad

```

```

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        self.layers = OrderedDict()
        self.layers["W1"] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.layers["b1"] = np.zeros(hidden_size)
        self.layers["W2"] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.layers["b2"] = np.zeros(output_size)

        # レイヤー構造
        self.layers["Affine1"] = layers.Affine(self.layers["W1"], self.layers["b1"])
        self.layers["ReLU1"] = layers.ReLU()
        self.layers["Affine2"] = layers.Affine(self.layers["W2"], self.layers["b2"])

        self.lastLayer = layers.Softmax(NLoss)

    def forward(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)
        return x

    def predict(self, x):
        y = self.forward(x)
        return y.argmax(axis=1)

    def loss(self, x, d):
        y = self.predict(x)
        return self.lastLayer.forward(y, d)

    def accuracy(self, x, d):
        y = self.predict(x)
        if d.ndim == 1:
            d = np.argmax(d, axis=1)
        accuracy = np.sum(y == d) / float(x.shape[0])
        return accuracy

    def backward(self, x, d):
        if d.ndim == 1:
            d = np.eye(d.size) * d
        self.loss(x, d).backward()

        if backward:
            dout = 1
            dout *= self.lastLayer.backward(dout)
            for layer in reversed(list(self.layers.values())):
                dout = layer.backward(dout)
            dout *= self.loss(x, d).backward(dout)

        grad = {}
        grad["W1"] = self.layers["W1"] + self.layers["Affine1"].dW * self.layers["Affine1"].db
        grad["b1"] = self.layers["b1"] + self.layers["Affine1"].db * self.layers["Affine1"].dW
        grad["W2"] = self.layers["W2"] + self.layers["Affine2"].dW * self.layers["Affine2"].db
        grad["b2"] = self.layers["b2"] + self.layers["Affine2"].db * self.layers["Affine2"].dW
        return grad

```

```

# データの読み込み
x_train, d_train, (x_test, d_test) = load_mnist(normalize=True, onehot_label=True)
print("データ読み込み完了")

# ネットワーク
network = TwoLayerNet(input_size=784, hidden_size=40, output_size=10)

# 訓練用
batch_size = 1000
train_size = x_train.shape[0]
train_size *= batch_size
train_size *= 0.1
train_size = int(train_size)
train_size_list = []
accuracies_train = []
accuracies_test = []
plot_interval = 10
plot_size = 1000000

# フォーマット?
for i in range(plot_size):
    batch_idx = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_idx]
    d_batch = d_train[batch_idx]

    # グラム
    grad = network.gradient(x_batch, d_batch)

    for key in ("W1", "W2", "b1", "b2"):
        network.layers[key].update_params(learning_rate * grad[key])

    loss = network.loss(x_batch, d_batch)
    train_size_list.append(loss)
    accuracies_train.append(network.accuracy(x_train, d_train))
    accuracies_test.append(network.accuracy(x_test, d_test))

    if i % plot_interval == 0:
        print(i)

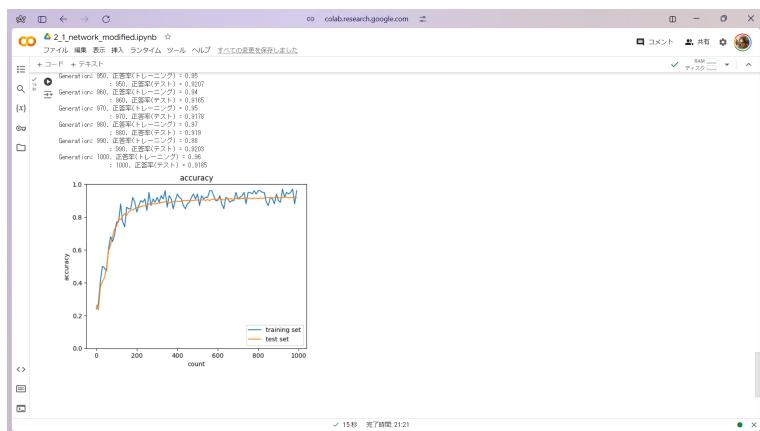
```

```

2.1_network_modified.ipynb が開かれています
+ コード + テキスト
[1]: 16  sourcetrain,sourceval,sourcetest]
17  sourcetrain = network.sourcetrain,batch_size,batch_size)
18  sourceval = network.sourceval,batch_size,batch_size)
19  sourcetest = network.sourcetest,batch_size,batch_size)
20
21  print("Generation: " + str(i) + ", 正確率(トレーニング)：" + str(sourcetrain))
22  print("Generation: " + str(i) + ", 正確率(バリデーション)：" + str(sourceval))
23  print("Generation: " + str(i) + ", 正確率(テスト)：" + str(sourcetest))
24
25
26  if i == 0:
27      f = open('accuracy.txt', 'w')
28  else:
29      f = open('accuracy.txt', 'a')
30
31  f.write(str(i) + "\t" + str(sourcetrain) + "\n")
32
33  f.close()
34
35  f = open('accuracy.txt', 'r')
36  accuracy = f.readlines()
37  f.close()
38
39  plt.xlabel('count')
40  plt.ylabel('accuracy')
41  plt.title('accuracy')
42  plt.plot(accuracy)
43  plt.show()
44
45  f.close()
46
```

データ読み込み完了。

Generation: 10, 正確率(トレーニング): 0.24
Generation: 10, 正確率(バリデーション): 0.07
Generation: 20, 正確率(トレーニング): 0.37
Generation: 20, 正確率(バリデーション): 0.2331
Generation: 30, 正確率(トレーニング): 0.41
Generation: 30, 正確率(バリデーション): 0.3747
Generation: 40, 正確率(トレーニング): 0.43
Generation: 40, 正確率(バリデーション): 0.4074
Generation: 50, 正確率(トレーニング): 0.438
Generation: 50, 正確率(バリデーション): 0.4208
Generation: 60, 正確率(トレーニング): 0.47
Generation: 60, 正確率(バリデーション): 0.4389
Generation: 70, 正確率(トレーニング): 0.6
Generation: 70, 正確率(バリデーション): 0.5268
Generation: 80, 正確率(トレーニング): 0.69
Generation: 80, 正確率(バリデーション): 0.6297
Generation: 90, 正確率(トレーニング): 0.75
Generation: 90, 正確率(バリデーション): 0.6917
Generation: 100, 正確率(トレーニング): 0.78
Generation: 100, 正確率(バリデーション): 0.7298
Generation: 110, 正確率(トレーニング): 0.785
Generation: 110, 正確率(バリデーション): 0.7425



2 2_2_2_vanishing_gradient_modified

```

2.2.2_vanishing_gradient_modified.ipynb
[1] 1 import sys
2     sys.path.append('~/code/asard') # 絶対パスをインポートするための設定
3
4 class MultiLayerNet:
5     def __init__(self, input_size, hidden_size_list, output_size, activation='relu', weight_init_std='relu'):
6         self.input_size = input_size
7         self.hidden_size_list = hidden_size_list
8         self.output_size = output_size
9         self.activation = activation
10        self.weight_init_std = weight_init_std
11
12        self.layers = []
13        self.loss_fn = None
14
15        self._init_layers() # レイヤー初期化
16
17    def _init_layers(self):
18        id = 0
19        self.layers.append(LayerAffine(self.input_size, self.hidden_size_list[0], self.activation))
20        self.hidden_size_list[0] = self.hidden_size_list[1]
21        self.hidden_size_list[1] = self.hidden_size_list[2]
22
23        for i in range(2, len(self.hidden_size_list)):
24            self.layers.append(LayerRelu())
25
26    def forward(self, x):
27        for layer in self.layers:
28            x = layer.forward(x)
29
30        return self.layers[-1].forward(x)
31
32    def backward(self, dout):
33        for layer in reversed(self.layers):
34            dout = layer.backward(dout)
35
36        return dout
37
38    def accuracy(self, x, t):
39        y = self.forward(x)
40        return np.sum(y.argmax(axis=1) == t) / float(x.shape[0])
41
42    def loss(self, x, t):
43        y = self.forward(x)
44        return self.loss_fn(y, t)
45
46    def predict(self, x):
47        y = self.forward(x)
48        return y.argmax(axis=1)
49
50    def __str__(self):
51        return f'MultiLayerNet({self.input_size}, {self.hidden_size_list}, {self.output_size})'
52
53    def __repr__(self):
54        return self.__str__()
55
56
57
58
59
5

```

ReLU: $y = \max(0, x)$

Sigmoid: $y = \frac{1}{1 + e^{-x}}$

Xavier Initialization: $\sqrt{\frac{1}{n}}$

```

2.2.2_vanishing_gradient_modified.ipynb
[1] 1 import sys
2     sys.path.append('~/code/asard') # 絶対パスをインポートするための設定
3
4 class MultiLayerNet:
5     def __init__(self, input_size, hidden_size_list, output_size, activation='relu', weight_init_std='relu'):
6         self.input_size = input_size
7         self.hidden_size_list = hidden_size_list
8         self.output_size = output_size
9         self.activation = activation
10        self.weight_init_std = weight_init_std
11
12        self.layers = []
13        self.loss_fn = None
14
15        self._init_layers() # レイヤー初期化
16
17    def _init_layers(self):
18        id = 0
19        self.layers.append(LayerAffine(self.input_size, self.hidden_size_list[0], self.activation))
20        self.hidden_size_list[0] = self.hidden_size_list[1]
21        self.hidden_size_list[1] = self.hidden_size_list[2]
22
23        for i in range(2, len(self.hidden_size_list)):
24            self.layers.append(LayerRelu())
25
26    def forward(self, x):
27        for layer in self.layers:
28            x = layer.forward(x)
29
30        return self.layers[-1].forward(x)
31
32    def backward(self, dout):
33        for layer in reversed(self.layers):
34            dout = layer.backward(dout)
35
36        return dout
37
38    def accuracy(self, x, t):
39        y = self.forward(x)
40        return np.sum(y.argmax(axis=1) == t) / float(x.shape[0])
41
42    def loss(self, x, t):
43        y = self.forward(x)
44        return self.loss_fn(y, t)
45
46    def predict(self, x):
47        y = self.forward(x)
48        return y.argmax(axis=1)
49
50    def __str__(self):
51        return f'MultiLayerNet({self.input_size}, {self.hidden_size_list}, {self.output_size})'
52
53    def __repr__(self):
54        return self.__str__()
55
56
57
58
59
5

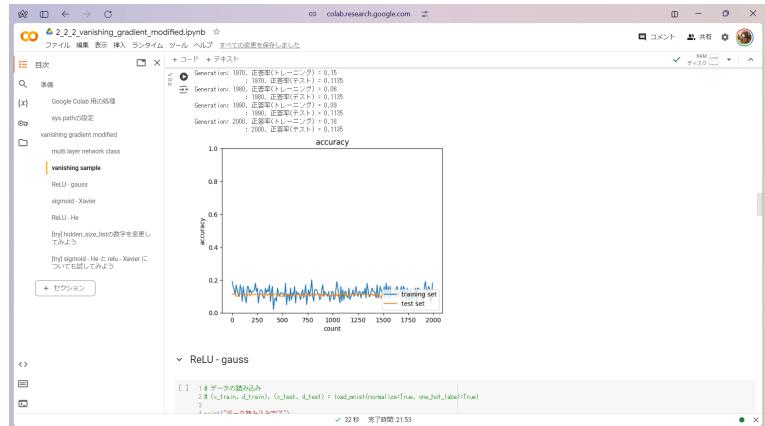
```

→ 條件分岐によって誤差が消えている。

```

2.2.2_vanishing_gradient_modified.ipynb
[1] 1 # データ読み込み
2     (x_train, _, x_test, _, y_train, _, y_test) = load_mnist(normalize=True, onehot_label=True)
3
4     print("学習データ数: " + str(len(x_train)))
5
6     # ネットワーク構成
7     net = MultiLayerNet(input_size=784, hidden_size=[50, 50, 50], output_size=10, activation='sigmoid', weight_init_std=0.01)
8
9     # ハイパーパラメータ
10    epochs = 2000
11    train_size = x_train.shape[0]
12    batch_size = 100
13    learning_rate = 0.01
14
15    # 訓練ループ
16    for epoch in range(epochs):
17        batch_idx = np.random.choice(train_size, batch_size)
18        x_batch = x_train[batch_idx]
19        d_batch = y_train[batch_idx]
20
21        # 勾配計算
22        grad = network.gradient(x_batch, d_batch)
23
24        for key in ('W1', 'W2', 'b1', 'b2', 'b3'):
25            network.backprop(key, learning_rate * grad[key])
26
27        loss = network.loss(x_batch, d_batch)
28        train_loss.append(loss)
29
30        if epoch % 100 == 0:
31            print("train loss: " + str(train_loss[-1]))
32
33        if (epoch + 1) % batch_size == 0:
34            acc_train = network.accuracy(x_train, y_train)
35            acc_test = network.accuracy(x_test, y_test)
36
37            print("train acc: " + str(acc_train))
38            print("test acc: " + str(acc_test))
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
5

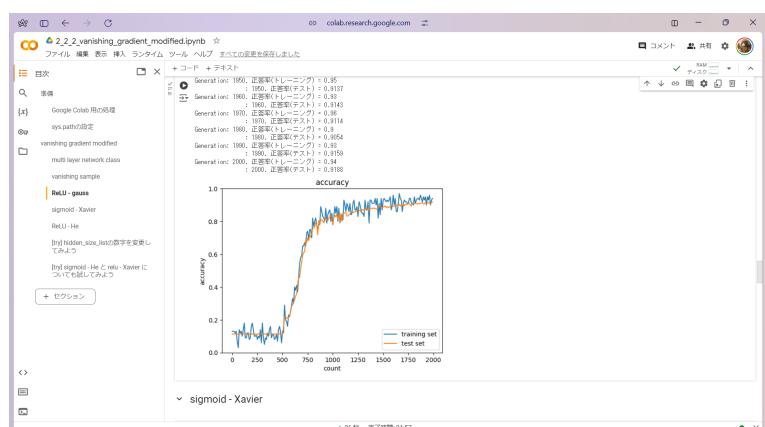
```

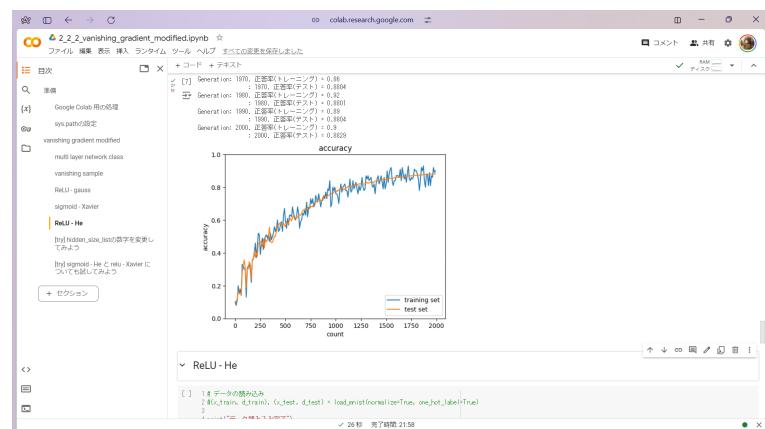


```
# データの読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True, onehot_label=True)

# ネットワーク
network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20], output_size=10, activation='relu', weight_init_std=0.01)
# 学習率
learning_rate = 0.1
# バッチサイズ
batch_size = 10
# トレーニング用データ
train_loss_list = []
# 誤差
accuracy_train = []
# テスト用データ
accuracy_test = []
# ポート
port_interval = 10
# ポート回数
for i in range(port_interval):
    batch_idx = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_idx]
    d_batch = d_train[batch_idx]
    # フロード
    network.gradient(x_batch, d_batch)
    for key in ('W1', 'B1', 'W2', 'B2', 'W3', 'B3'):
        network.gradient[key] += learning_rate * grad[key]
    loss = network.loss(x_batch, d_batch)
    train_loss_list.append(loss)
    if (i + 1) % port_interval == 0:
        accuracy_train.append(network.accuracy(x_train, d_train))
        accuracy_test.append(network.accuracy(x_test, d_test))
        print('train %d' % (i + 1))

# 活性化関数を ReLU にする。
```





```
# データの読み込み
(x_train, y_train), (x_test, y_test) = load_mnist(normalize=True, onehot_label=True)
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)

# ネットワーク
net = MultiLayerNet(input_size=784, hidden_size_list=[40, 20], output_size=10, activation='relu', weight_init_std=0.01)

# 学習パラメータ
iters_num = 2000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1
learning_rate_0 = 0.1

# 学習用データ
train_loss_list = []
accuracy_list = []
accuracy_train = 1.0
accuracy_val = 1.0

# 学習用バッチ数
batch_size = 100

# 1. In most iterations:
batch_size_k = np.random.choice(train_size, batch_size)
x_batch_k = x_train[batch_size_k]
y_batch_k = y_train[batch_size_k]

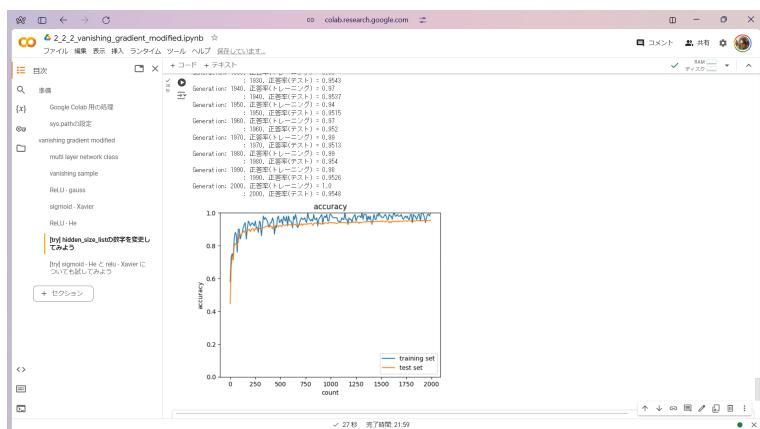
# 2. Compute gradients
grad_k = network.gradient(x_batch_k, y_batch_k)

# 3. Update
for key in grad_k:
    network.params[key] -= learning_rate * grad_k[key]

# 4. Compute loss
loss = network.loss(x_batch_k, y_batch_k)
train_loss_list.append(loss)

# 5. Compute accuracy
if (i + 1) % 10 == 0:
    train_accuracy = np.sum(np.argmax(network.predict(x_train), axis=1) == y_train) / train_size
    accuracy_list.append(train_accuracy)
    accuracy_train = train_accuracy
    accuracy_val = np.sum(np.argmax(network.predict(x_test), axis=1) == y_test) / test_size
    print("step %d, loss %.2f, train acc %.3f, val acc %.3f" % (i + 1, loss, accuracy_train, accuracy_val))
```

ReLU の He 値変化と使用



3 2_3_batch_normalization

```


```

1 # フォルダの作成
2 import os
3 if not os.path.exists('DNN_code'):
4 os.makedirs('DNN_code')
5 if not os.path.exists('DNN_code/colab/day2'):
6 os.makedirs('DNN_code/colab/day2')
7 if not os.path.exists('DNN_code/colab/day2/notebook'):
8 os.makedirs('DNN_code/colab/day2/notebook')
9 os.chdir('DNN_code/colab/day2/notebook')
10 os.getcwd()
11 os.listdir()

```


```

```


```

1 import numpy as np
2 from collections import OrderedDict
3 from copy import deepcopy
4 from math import sqrt
5 import tensorflow as tf
6 from tensorflow import keras
7 from keras import layers
8
9 # バッチ正規化 layer
10 class BatchNormalization:
11 def __init__(self, axis=1, epsilon=1e-05, momentum=0.9, running_mean=None, running_var=None):
12 self.axis = axis
13 self.epsilon = epsilon
14 self.momentum = momentum
15 self.running_mean = None
16 self.running_var = None
17 self.trainable = True
18 self.gamma = None
19 self.beta = None
20 self.running_mean = None
21 self.running_var = None
22 self.input_shape = None
23 self.trainable = True
24 self.running_mean = None
25 self.running_var = None
26 self.backup_mean = None
27 self.backup_var = None
28 self.old_beta = None
29 self.old_gamma = None
30 self.old_mean = None
31 self.old_var = None
32 self.data = None
33
34 def forward(self, x, train_flag=True):
35 if self.running_mean is None:
36 self.running_mean = np.zeros(x.shape)
37 self.running_var = np.zeros(x.shape)
38 self.old_mean = None
39 self.old_var = None
40
41 if train_flag:
42 mu = np.mean(x, axis=self.axis)
43 var = np.var(x, axis=self.axis)
44 std = np.sqrt(var + self.epsilon) # 分散
45 std = np.where(std == 0, 1e-07) # スケーリング
46 x_hat = (x - mu) / std
47 self.old_mean = mu
48 self.old_var = var
49 self.old_std = std
50 self.old_beta = self.beta
51 self.old_gamma = self.gamma
52 self.old_mean = self.old_mean * self.momentum + x_hat * (1 - self.momentum)
53 self.old_var = self.old_var * self.momentum + x_hat ** 2 * (1 - self.momentum) + (1 - self.momentum) * self.old_var
54 self.old_std = np.sqrt(self.old_var + self.epsilon)
55 self.old_beta = self.old_beta + self.old_mean * (0 - 1)
56 self.old_gamma = self.old_gamma / (self.old_std + (0 - 1))
57 out = self.old_gamma * x_hat + self.old_beta
58
59 else:
60 out = self.old_gamma * x_hat + self.old_beta
61
62 return out
63
64 def backward(self, dout):
65 dgamma = np.sum(dout * self.old_gamma, axis=0)
66 dbeta = np.sum(dout * self.old_beta, axis=0)
67 dx_hat = dout * self.old_gamma
68 dvar = -0.5 * self.old_gamma * np.sum(dx_hat * dx_hat, axis=0)
69 dmean = -np.sum(dx_hat, axis=0)
70 dstd = -dvar * dmean / (self.old_std * self.old_std * self.old_std)
71 dx = dx_hat * self.old_std + dmean / self.old_std
72 dx = dx * 2.0 / self.old_batch_size
73 dbeta = np.sum(dx * self.old_beta, axis=0)
74 dgamma = np.sum(dx * self.old_gamma, axis=0)
75 self.old_gamma = dgamma
76 self.old_beta = dbeta

```


```

Handwritten annotations in the code:

- $\mu = \frac{1}{m} \sum_{i=1}^m x_i$
- $\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu)^2$
- $\hat{x}_i = \frac{(x_i - \mu)}{\sqrt{\sigma^2 + \epsilon}}$
- $y_i = \gamma \hat{x}_i + \beta$

```


```

12 if self.running_mean is None:
13 self.running_mean = np.zeros(x.shape)
14 self.running_var = np.zeros(x.shape)
15 self.old_mean = None
16 self.old_var = None
17 self.old_std = None
18 self.old_beta = None
19 self.old_gamma = None
20 self.old_mean = None
21 self.old_var = None
22 self.old_std = None
23 self.old_beta = None
24 self.old_gamma = None
25 self.old_mean = None
26 self.old_var = None
27 self.old_std = None
28 self.old_beta = None
29 self.old_gamma = None
30 self.old_mean = None
31 self.old_var = None
32 self.old_std = None
33 self.old_beta = None
34 self.old_gamma = None
35 self.old_mean = None
36 self.old_var = None
37 self.old_std = None
38 self.old_beta = None
39 self.old_gamma = None
40
41 if train_flag:
42 mu = np.mean(x, axis=self.axis)
43 var = np.var(x, axis=self.axis)
44 std = np.sqrt(var + self.epsilon) # 分散
45 std = np.where(std == 0, 1e-07) # スケーリング
46 x_hat = (x - mu) / std
47 self.old_mean = mu
48 self.old_var = var
49 self.old_std = std
50 self.old_beta = self.beta
51 self.old_gamma = self.gamma
52 self.old_mean = self.old_mean * self.momentum + x_hat * (1 - self.momentum)
53 self.old_var = self.old_var * self.momentum + x_hat ** 2 * (1 - self.momentum) + (1 - self.momentum) * self.old_var
54 self.old_std = np.sqrt(self.old_var + self.epsilon)
55 self.old_beta = self.old_beta + self.old_mean * (0 - 1)
56 self.old_gamma = self.old_gamma / (self.old_std + (0 - 1))
57 out = self.old_gamma * x_hat + self.old_beta
58
59 else:
60 out = self.old_gamma * x_hat + self.old_beta
61
62 return out
63
64 def backward(self, dout):
65 dgamma = np.sum(dout * self.old_gamma, axis=0)
66 dbeta = np.sum(dout * self.old_beta, axis=0)
67 dx_hat = dout * self.old_gamma
68 dvar = -0.5 * self.old_gamma * np.sum(dx_hat * dx_hat, axis=0)
69 dmean = -np.sum(dx_hat, axis=0)
70 dstd = -dvar * dmean / (self.old_std * self.old_std * self.old_std)
71 dx = dx_hat * self.old_std + dmean / self.old_std
72 dx = dx * 2.0 / self.old_batch_size
73 dbeta = np.sum(dx * self.old_beta, axis=0)
74 dgamma = np.sum(dx * self.old_gamma, axis=0)
75 self.old_gamma = dgamma
76 self.old_beta = dbeta

```


```



```
2_3_batch_normalization.ipynb ☆
+ コード + テキスト ファイル 開く 検索 ライブセル ツール ヘルプ
[3] In [3]: %pylab inline
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# データの読み込み
mnist = keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train, y_test = y_train.astype("float32"), y_test.astype("float32")
y_train, y_test = keras.utils.to_categorical(y_train, 10), keras.utils.to_categorical(y_test, 10)

# モデルの構築
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])

# モデルの訓練
model.compile(optimizer="adam",
              loss="categorical_crossentropy",
              metrics=["accuracy"])
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.1)

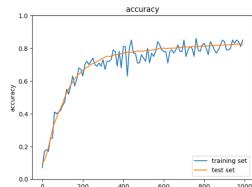
# モデルの評価
score = model.evaluate(x_test, y_test, batch_size=32)
print(score)
```

シグモイド、Xavier正規化で使用

面積 (9.3%) : recall100 > loss0 > predict0 > forward0

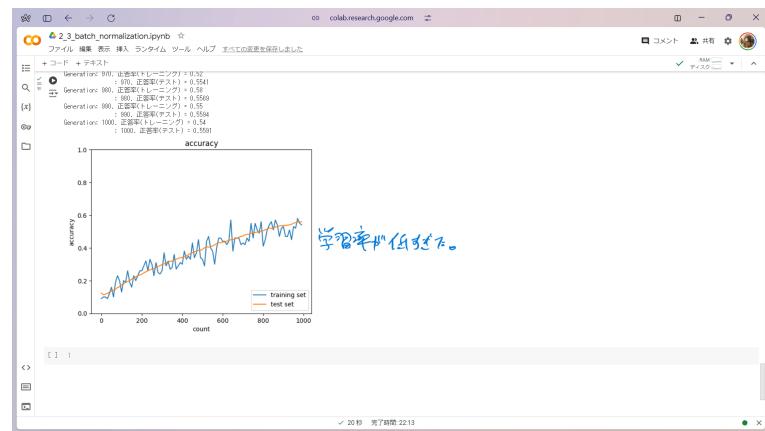
The figure shows a line graph titled "accuracy" plotted against "count" (ranging from 0 to 1000). Two data series are shown: "training set" (blue line) and "test set" (orange line). The training set accuracy starts at ~0.15 and quickly rises to ~0.85, then fluctuates between 0.8 and 0.9. The test set accuracy starts at ~0.15 and reaches ~0.85, tracking closely with the training set after count 200.

count	training set accuracy	test set accuracy
0	0.15	0.15
100	0.55	0.55
200	0.80	0.80
300	0.85	0.85
400	0.88	0.88
500	0.85	0.85
600	0.88	0.88
700	0.85	0.85
800	0.88	0.88
900	0.85	0.85
1000	0.88	0.88



2.3_batch_normalization.ipynb

```
1 #x_train, d_train, (x_test, d_test) = load_mnist(normalize=True)
2
3 print("データ読み込み終了")
4
5 if batch_normalization:
6     print("batch normalizationの設定")
7     use_batches = True
8     use_jd_update = False
9     use_jd_update = True
10
11 network = MultiLayerNet(input_size=784, hidden_size=[10, 20], output_size=10,
12                         activation='relu', weight_init_std=1e-2, use_batchnorm=batchnorm)
13
14 iter_num = 1000
15 train_loss = x_train.shape[0]
16 batch_size = 10
17 learning_rate = 0.001
18
19 train_loss = []
20 accuracy = []
21
22 plot_iter = 10
23
24
25 for i in range(iter_num):
26     batch_idx = np.random.choice(train_size, batch_size)
27     x_batch = x_train[batch_idx]
28     d_batch = d_train[batch_idx]
29     j_batch = j_train[batch_idx]
30
31     grad = network.gradient(x_batch, d_batch)
32     for key in ('W1', 'W2', 'b1', 'b2', 'gamma', 'beta'):
33         network.params[key] -= learning_rate * grad[key]
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
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
517
518
519
519
520
521
522
523
524
525
526
527
527
528
529
529
530
531
532
533
534
535
535
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
163
```



4 2_4_optimizer



The screenshot shows a Google Colab notebook titled "2_4_optimizer.ipynb". The code cell at the top defines a function `train` that performs gradient descent optimization. Below it, a cell imports TensorFlow and defines a variable `mnist`. A third cell contains a large block of code for training a neural network using Adam optimizer. The output pane shows the progress of the training process, including accuracy and loss metrics over time.

```
def train(optimizer):
    for epoch in range(100):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys})
        if epoch % 10 == 0:
            print('Epoch', epoch, 'Accuracy:', accuracy.eval({x: batch_xs, y: batch_ys}), 'Loss:', loss.eval({x: batch_xs, y: batch_ys}))
```

```
import tensorflow as tf
mnist = input_data.read_data_sets('./mnist', one_hot=True)
```

```
def train():
    x = tf.placeholder(tf.float32, [None, 784])
    y = tf.placeholder(tf.float32, [None, 10])

    W = tf.Variable(tf.zeros([784, 10]))
    b = tf.Variable(tf.zeros([10]))
    y_ = tf.matmul(x, W) + b

    cross_entropy = tf.reduce_mean(
        tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=y_))

    optimizer = tf.train.AdamOptimizer().minimize(cross_entropy)

    correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    sess = tf.Session()
    sess.run(tf.global_variables_initializer())

    for i in range(1000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys})

        if i % 100 == 0:
            print(sess.run(accuracy, feed_dict={x: batch_xs, y: batch_ys}))
```

解釈例、精度 0.1~0.2 で停滯(2.7~4)。
正規化地学習率を調整する: 0.7<いいすばり昇

Google Colab の環境

sys.pathの設定

optimizer

SGD

Momentum

MomentumをもとにAdagradを作つ
てみよう

RMSprop

Adam

[ipy] 学習率を変えてみよう

[ipy] 損失関数と他の学習手法を
変更してみよう

[ipy] パラメータを正規化してみよう

+ セクション

```
class Momentum:
    def __init__(self, lr=0.01, momentum=0.9):
        self.m = None
        self.lr = lr
        self.momentum = momentum

    def update(self, network, grads):
        if self.m is None:
            self.m = [np.zeros_like(param) for param in network.params]

        for i, (param, grad) in enumerate(zip(network.params, grads)):
            self.m[i] = self.momentum * self.m[i] + (1 - self.momentum) * grad
            param -= self.lr * self.m[i]
```

Momentumも勾配降尺度法同様に工いて。

Momentum 老句配降不法同様の工夫です。

2.4. optimizer.ipynb

Google Colab 用の環境
sys.pathの設定
optimizer
SGD
Momentum

MomentumをもとにAdagradを作つてみよう

IM2Mop

Adam

[ipy] 学習率を変えてみよう

[ipy] 時間的学習率の初期化方法を変えてみよう

[ipy] バイアス正則化をしてみよう

+ セクション

```
27 for i in range(len(x_train)):
28     batch_idx = np.random.choice(len(x_train), batch_size)
29     x_batch = x_train[batch_idx]
30     y_batch = y_train[batch_idx]
31
32     # SGD
33     f_x = f(x_batch, y_batch)
34     grad = network.gradient(x_batch, y_batch)
35     if i == 0:
36         print(f_x)
37     for k in range(100):
38         v_x = np.zeros(len(network.layers[0]))
39         v_y = np.zeros(len(network.layers[1]))
40         # v_x, v_y: momentumのkeyは「learning_rate + grad**2」
41         for j in range(len(network.layers[0])):
42             v_x[j] = network.layers[0].keys[j] - learning_rate * grad[0][j]
43             v_y[j] = network.layers[1].keys[j] - learning_rate * grad[1][j]
44             train_loss += f(x_batch, y_batch)
45
46         if i % 1000 == 0:
47             acc_test = network.accuracy(x_test, d_test)
48             acc_train = network.accuracy(x_train, d_train)
49             accuracies_train.append(acc_train)
50             accuracies_test.append(acc_test)
51
52             print('Generation : ' + str(i) + ', 正解率(トレーニング) : ' + str(acc_train))
53             print('Generation : ' + str(i) + ', 正解率(テスト) : ' + str(acc_test))
54
55
56         # 亂数生成
57         noise_0 = np.random.randn(len(x_train), len(x_train), len(x_train)))
58         noise_1 = np.random.randn(len(x_train), len(x_train), len(x_train)))
59         noise_2 = np.random.randn(len(x_train), len(x_train), len(x_train)))
60
61         # 乱数を足す
62         x_train += noise_0
63         x_test += noise_1
64         d_train += noise_2
65         d_test += noise_2
66
67         # データを並び替える
68         np.random.shuffle(x_train)
69         np.random.shuffle(d_train)
70         np.random.shuffle(x_test)
71         np.random.shuffle(d_test)
72
73         # データを並び替える
74         np.random.shuffle(x_train)
75         np.random.shuffle(d_train)
76         np.random.shuffle(x_test)
77         np.random.shuffle(d_test)
78
79         # データを並び替える
80         np.random.shuffle(x_train)
81         np.random.shuffle(d_train)
82         np.random.shuffle(x_test)
83         np.random.shuffle(d_test)
84
85         # データを並び替える
86         np.random.shuffle(x_train)
87         np.random.shuffle(d_train)
88         np.random.shuffle(x_test)
89         np.random.shuffle(d_test)
90
91         # データを並び替える
92         np.random.shuffle(x_train)
93         np.random.shuffle(d_train)
94         np.random.shuffle(x_test)
95         np.random.shuffle(d_test)
96
97         # データを並び替える
98         np.random.shuffle(x_train)
99         np.random.shuffle(d_train)
100        np.random.shuffle(x_test)
101        np.random.shuffle(d_test)
```

$$U_i^{t+1} = \boxed{\alpha U_i^t} + \left(\text{前回の } \frac{\partial E}{\partial w_i} \text{ の総差} \right)$$

```
Google Colab の起動確認
sys.pathの設定
optimizer
SGD
RMSprop
Momentum

MomentumをもとにAdaGradを作つてみよう
[ipy] SGDをもとに基礎的な実験方法を見てみよう
[ipy] パラメータ正則化をしてみよう
+ セッション
```

ツール ヘルプ 全ての変数を保存しました
+ コード + テキスト

日次

準備

Google Colab の起動確認

sys.pathの設定

optimizer

SGD

RMSprop

Momentum

MomentumをもとにAdaGradを作つてみよう

[ipy] SGDをもとに基礎的な実験方法を見てみよう

[ipy] パラメータ正則化をしてみよう

+ セッション

ツール ヘルプ 全ての変数を保存しました
+ コード + テキスト

Generation: 100, 正確率(トレーニング) : 0.30
: 100, 正確率(テスト) : 0.3416
Generation: 200, 正確率(トレーニング) : 0.35
: 200, 正確率(テスト) : 0.3889
Generation: 300, 正確率(トレーニング) : 0.39
: 300, 正確率(テスト) : 0.4202
Generation: 400, 正確率(トレーニング) : 0.47
: 400, 正確率(テスト) : 0.5158
Generation: 500, 正確率(トレーニング) : 0.8579
: 500, 正確率(テスト) : 0.8579

accuracy

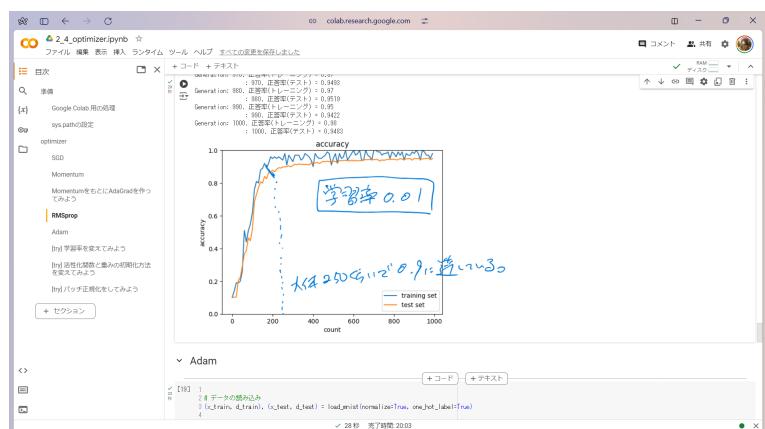
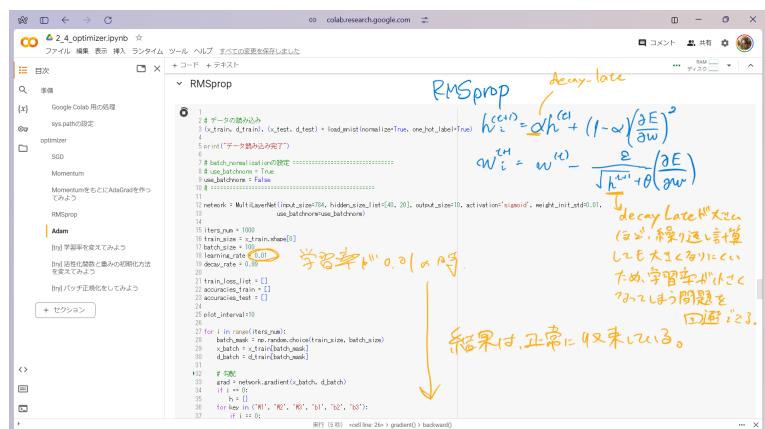
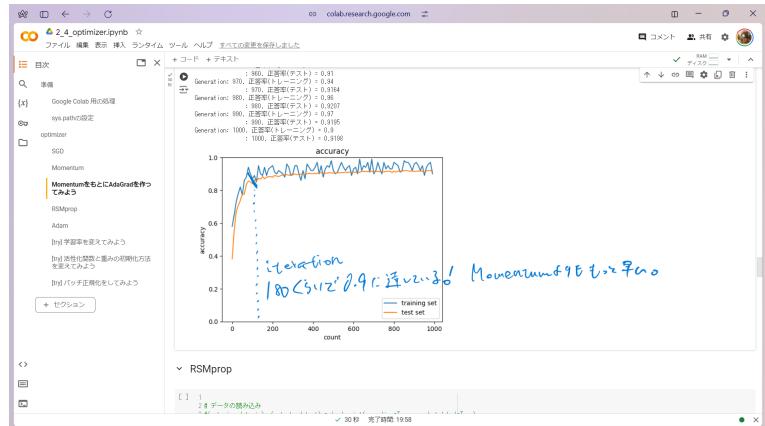
training set test set

count

0.0 0.2 0.4 0.6 0.8 1.0

0 200 400 600 800 1000

がまごは250回繰り返せば0.9に達する。
Momentumは早い結果になります。



Google Colab の環境

sys.pathの設定

optimizer

SGD

Momentum

MomentumをもとにAdaGradを作つてみる

RMSprop

Adam

ipython 実行環境をみてみよう

ipython 実行環境の重複の削除方法

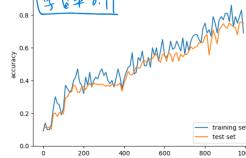
ipython バッチ正規化をしてみよう

+ セクション

```
1 # データの読み込み
2 (x_train, y_train), (x_test, y_test) = load_mnist(normalize=True, one_hot_label=True)
3
4 print("データ読み込み終了!")
5
6 batch_size = 100
7 learning_rate = 0.001
8 use_dropout = False
9
10 network = MultiLayerNetwork(input_size=784, hidden_size_list=[40, 20], output_size=10, activation='sigmoid', weight_init_std=0.01,
11 use_batch_norm=True, use_dropout=True)
12
13
14 for epoch in range(30):
15     for i in range(0, len(x_train), batch_size):
16         x_batch = x_train[i:i+batch_size]
17         y_batch = y_train[i:i+batch_size]
18         d_batch = network.gradient(x_batch, y_batch)
19
20         if use_dropout:
21             d_batch *= np.random.choice(2, size=d_batch.shape) - 1
22
23         network.backward(d_batch)
24
25         if i % 10 == 0:
26             print("train loss: ", network.loss(x_batch, y_batch))
27
28         if i % 1000 == 0:
29             accuracy = np.sum(np.argmax(network.predict(x_batch), 1) == y_batch) / float(batch_size)
30             print("accuracy: ", accuracy)
31
32         if i % 10000 == 0:
33             print("train loss: ", network.loss(x_train, y_train))
34
35         if i % 100000 == 0:
36             for param in network.params:
37                 print(param.name, param.value)
```

↑ 学習率 0.1 の時

↓ 結果はNG.



1000回繰り返しても0.9%達かない。
RMSpropの特徴として、学習率が
大きくなりすぎてしまうといふのがある。
→ 学習率で大きくなると、火に油
を注ぐごとく、学習率を上昇
してしまう。



The screenshot shows the code for the Adam optimizer in a Jupyter notebook cell. The code uses TensorFlow's `tf.train.AdamOptimizer` and includes a detailed docstring explaining the parameters and their meanings.

```
# 2_4_optimizer.pyrb
...
ツール ヘルプ すべての変更を保存しました
+ コード + テキスト
...
# [x] Google Colab 用の環境
...
@sys.pathの設定
optimizer
...
SSD
...
Momentum
...
MomentumをもとにAdamGradientを作った
...
RMSprop
...
Adam
...
[ln] 学習率を変えてみよう
...
[ln] 価値関数と重みの初期化方法
...
[ln] 変更をどうするか
...
[ln] バイアス正則化してみよう
...
+ セッション
...
<ipython>
...
https://colab.research.google.com/drive/14NWH90eap2...

```

