

パターン認識

2019-05-28 授業分 レポート

37-196360 森田涼介

2019 年 6 月 12 日

宿題 1

05-21 授業分で配布された三種類のデータで、入力層・隠れ層 1 つ・出力層からなるニューラルネットワークを誤差逆伝播法により学習させる。

3 種類全てのデータに対し、学習率は 0.10 とし、隠れ層のノード数は 20 とした。また、更新はバッチ学習で行った。結果を以下の表 1, 図 1-3 に示す。前回の課題にあったパーセプトロンや MSE ではうまく対応できなかった非線形な境界を持つデータに対しても、ある程度うまく対応できていることがわかる。

表 1: 結果

Data Type	#Data	#hidden	Learning Rate	Epoch	Loss	#Correct	Accuracy
linear	100	10	0.90	500	0.1829	100	1.00
nonlinear	100	10	0.90	1,500	0.5186	83	0.830
slinear	500	10	0.90	500	0.1416	490	0.980

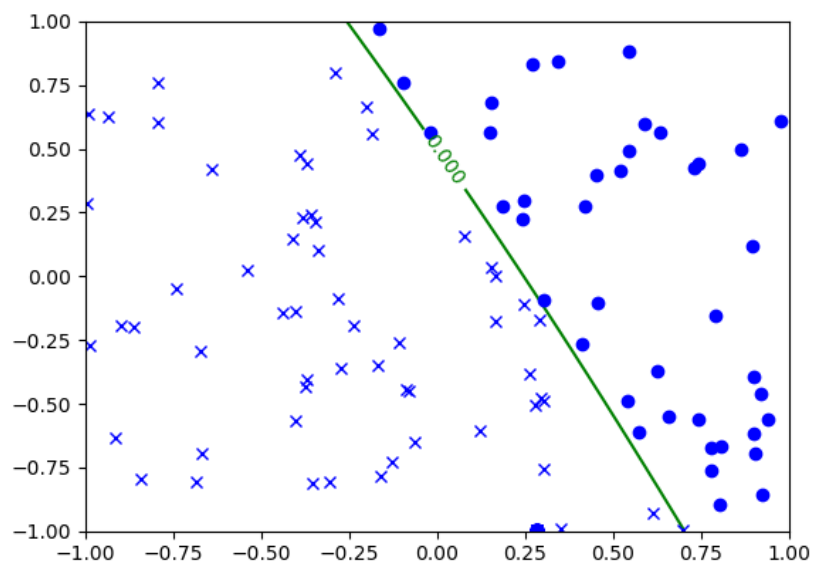


図 1: linear データに対する結果

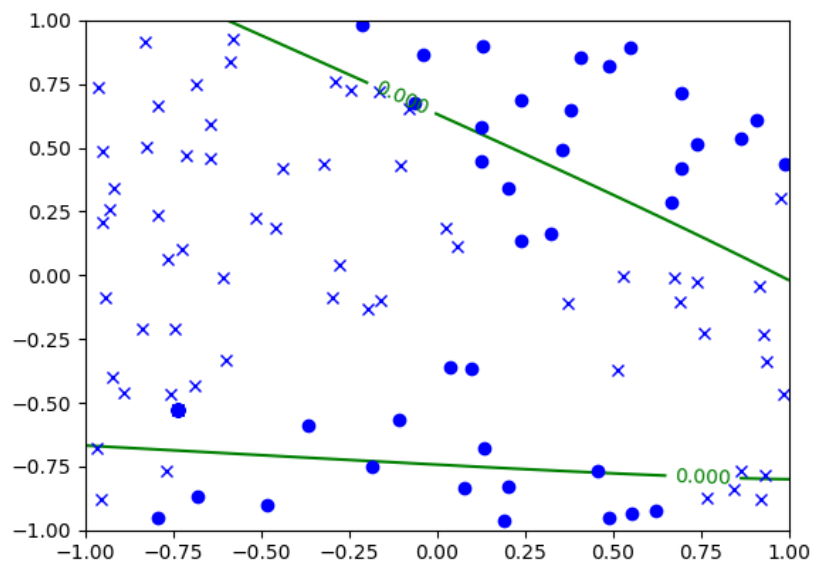


図 2: nonlinear データに対する結果

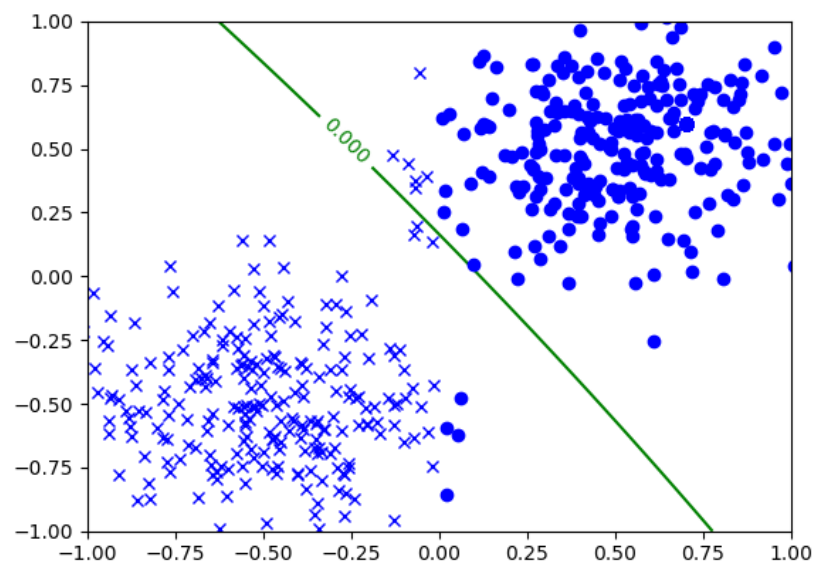


図 3: slinear データに対する結果

プログラムはページ 13 の Listing 2 に示した。また，プログラム中で呼び出している `neural_network` モジュールは，ページ 10 の Listing 1 に示した。以下にまず，`neural_network` モジュールの関数とクラスの説明を示す。なお，このモジュールは宿題 3 でも用いる。

- `identity_function`
恒等関数
- `deriv_identity_function`
恒等関数の微分
- `sigmoid`
sigmoid 関数
- `deriv_sigmoid`
sigmoid 関数の微分
- `FC`
ニューラルネットワークの 1 層を表すクラス。入出力のサイズと活性化関数及びその微分の関数を引数に取る。重み・バイアスの他に，それらの勾配，順伝搬時の活性化前の値，誤差なども属性として持つ。また，誤差逆伝搬と勾配を計算するメソッドを持つ。
- `Model`
層の構成を引数に取る，モデルを表すクラス。全体に対し誤差逆伝搬とパラメータの更新を行うメソッドを持つ。

また，以下に `assignment1.py` の各関数の説明を示す。

- `load_data`
.mat ファイルからデータを読み込む関数
- `compute_loss`
実値と予測値から損失を計算する関数
- `plot`
境界と点群をプロットする関数
- `train`
ニューラルネットワークのモデルの最適なパラメータを，勾配降下法と誤差逆伝搬法によって求める関数
- `main`
実行時の処理をまとめた関数

宿題 2

MNIST を識別する MSE 識別器を実装する。

学習は LMS 法によって行った。学習率を 0.007, ミニバッチサイズを 100, エポックを 500 とした。訓練データのうち 10,000 個を validation 用に分け, 学習状況の確認に利用した。

混同行列は表 2 のようになり, また, 各カテゴリごとの正解率等は表 3 のようになった。

表 2: MSE に対する混同行列

	0	1	2	3	4	5	6	7	8	9
0	957	0	0	2	0	6	7	2	6	0
1	0	1101	3	1	3	1	5	1	20	0
2	33	51	802	27	21	1	32	20	40	5
3	9	18	24	875	7	19	4	19	25	10
4	1	16	7	1	895	3	8	1	10	40
5	34	16	3	74	25	656	16	16	36	16
6	40	9	12	0	26	17	843	0	11	0
7	5	39	11	7	26	1	0	890	4	45
8	20	43	7	23	31	34	13	13	771	19
9	19	11	2	12	85	1	0	83	6	790

表 3: MSE に対する各カテゴリごとの結果

Category	#Data	#Correct	Accuracy
0	980	957	0.977
1	1,135	1101	0.970
2	1,032	802	0.777
3	1,010	875	0.866
4	9,82	895	0.911
5	8,92	656	0.735
6	9,58	843	0.880
7	1,028	890	0.866
8	9,74	771	0.792
9	1,009	790	0.783
All	10,000	8,580	0.858

プログラムは 16 ページの Listing 3 に示した。その説明を以下に簡単に記す。なお、同じプログラムで宿題 3 の実行もできる。

- `load_data`
MNIST データを読み込んで返す関数。04-23 の課題で用いた `mnread` モジュールを用いている。
- `add_augment_axis`
入力を拡張ベクトルにする関数
- `normalize`
入力を正規化する関数。今回は `[0, 256]` のグレースケールが対象だったため、簡単に、各要素に対して 128 を引いてから 256 で割る操作を行っている。
- `split`
入力を与えられたバッチサイズごとに分ける関数
- `train`
訓練を行う関数。`mode` 引数で `MSE` か `MLP` かを切り替え可能にしている。ミニバッチに対して順伝搬を行い損失を計算したのち、勾配を計算してパラメータを更新する。`MLP` を用いる場合は、勾配の計算に誤差逆伝搬法が用いられる。
- `test`
テストデータに対する結果を求める関数
- `print_result_in_TeX_tabular_format`
混同行列と各ラベルに対する精度を、`TeX` の表の形式で出力する関数
- `compute_loss`
損失を計算する関数
- `Linear`
`MSE` 用の線形モデルを表したクラス。勾配計算とパラメータの更新を行うメソッドを持つ。

宿題 3

MNIST を識別するニューラルネットワークを実装する。宿題 1 と同等の構造を利用する。

中間層のユニット数を 1,000, 学習率を 0.007, ミニバッチサイズを 100, エポックを 500 とした。訓練データのうち 10,000 個を validation 用に分け、学習状況の確認に利用した。

混同行列は表 4 のようになり、また、各カテゴリごとの正解率等は表 5 のようになった。

なお、プログラムは 16 ページの Listing 3 に示した。関数についての説明は宿題 2 に示した通りである。また、モデルについては宿題 1 で示した通りである。

表 4: NN に対する混同行列

	0	1	2	3	4	5	6	7	8	9
0	957	0	0	2	0	6	7	2	6	0
1	0	1101	3	1	3	1	5	1	20	0
2	33	51	802	27	21	1	32	20	40	5
3	9	18	24	875	7	19	4	19	25	10
4	1	16	7	1	895	3	8	1	10	40
5	34	16	3	74	25	656	16	16	36	16
6	40	9	12	0	26	17	843	0	11	0
7	5	39	11	7	26	1	0	890	4	45
8	20	43	7	23	31	34	13	13	771	19
9	19	11	2	12	85	1	0	83	6	790

表 5: NN に対する各カテゴリごとの結果

Category	#Data	#Correct	Accuracy
0	980	957	0.977
1	1,135	1101	0.970
2	1,032	802	0.777
3	1,010	875	0.866
4	9,82	895	0.911
5	8,92	656	0.735
6	9,58	843	0.880
7	1,028	890	0.866
8	9,74	771	0.792
9	1,009	790	0.783
All	10,000	8,580	0.858

宿題 4

以下の深層学習に関する技術用語について説明する。これらを講義で説明した多層ニューラルネットワークに組み込む方法についても検討する。

Dropout

Dropout は、訓練時にニューロンのうち一部を確率的に不活性化させて学習させることにより、汎化性能を上げる手法である。つまり、Dropout を適用する層において、重みのうち一部を確率的に選び、選ばれたものの重みを 0 として順伝搬を行い、また逆伝搬時にも重みの更新をしないようにする。ここで、Dropout する確率はハイパーパラメータとなる。推論時には、全てのノードを用いて計算を行った後、訓練時に Dropout していた割合を出力にかけることによって、用いているノードの数が訓練時より増加し大きくなった出力の値を修正している。このように学習することによって、複数のネットワークを独立に学習し、推論時にその出力を平均する、アンサンブル学習の近似になるといわれている。

これを多層ニューラルネットワークに組み込もうとするときは、単純に層の後にこれを入れればよい。なお、どのノードを不活性化したかを記憶しておく必要があることに注意である。

momentum

パラメータの更新は SGD 以外の方法によっても行うことができる。momentum はその手法の 1 つで、直訳すると運動量である。momentum によるパラメータの更新式は以下の式で表される (α はハイパーパラメータ)。

$$\mathbf{v} \leftarrow \alpha \mathbf{v} - \eta \frac{\partial E}{\partial \boldsymbol{\theta}} \quad (1)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v} \quad (2)$$

式からわかるように、momentum では更新に用いる値を計算された勾配の値そのものからその指数平均へと変更している。これにより、振動が抑制され、損失関数が極小値に向かいやすくなるという手法である。パラメータが張る空間上での損失関数の値の動きを考えると、これは慣性が働いているかのような動きとなる。

これを多層ニューラルネットワークに組み込もうとするときは、パラメータの更新式を SGD からこれに取り換えればよい。 \mathbf{v} の値を記憶しておかなければならないことに注意である。

data augmentation

data augmentation とは、データを増やす手法のことである。これは、新規にデータを集めるということではなく、現在手元にあるデータに対し何かしらの処理を加えることでデータを増やすということである。この手法は当然データの種類によって様々であるが、例えば画像について考えると、次のようなものが挙げられる

- 上下・左右の反転
- 回転
- ノイズの付加
- 色の反転
- Random Crop (画像の一部を切り抜く)

- CutOut（画像の一部にマスクをかける）

当然，これらの手法は状況に応じて使い分けられるべきである。例えば，手書き文字認識をしたいときには，左右の反転は用いられないだろう。

これは前処理にあたる部分なので，多層ニューラルネットワークに組み込もうとするときは，その入力に対する操作となる。例えば MNIST に対しては，ノイズの付加や多少の回転は適用してみてもよい手法であると考えられる。

1 プログラム

実行環境と用いた言語・ライブラリを以下の表 6 に示す。

表 6: プログラムの実行環境

OS	: Microsoft Windows 10 Pro (64bit)
CPU	: Intel(R) Core(TM) i5-4300U
RAM	: 4.00 GB
使用言語	: Python3.6
可視化	: matplotlib ライブラリ

Listings 1: neural_network.py

```
1  # -*- coding: utf-8 -*-
2
3  import numpy as np
4
5
6  def identity_function(x):
7      return x
8
9
10 def deriv_identity_function(x):
11     return np.ones_like(x)
12
13
14 def sigmoid(x):
15     return 1.0 / (1.0 + np.exp(-x))
16
17
18 def deriv_sigmoid(x):
19     return sigmoid(x) * (1 - sigmoid(x))
20
21
22 def softmax(x):
23     x -= x.max(axis=1, keepdims=True)
24     x_exp = np.exp(x)
25     x_exp /= np.sum(x_exp, axis=1, keepdims=True)
26     return x_exp
27
28
29 def deriv_softmax(x):
30     return softmax(x) * (1 - softmax(x))
31
32
33 class FC(object):
```

```

34     def __init__(self, input_size, output_size, activate_func,
35 activate_func_deriv,):
36         self.W = np.random.rand(input_size, output_size).astype(float)
37         self.b = np.zeros(output_size, dtype=float)
38
39         self.dW = None
40         self.db = None
41
42         self.x = None
43         self.u = None
44
45         self.delta = None
46
47         self.activate_func = activate_func
48         self.activate_func_deriv = activate_func_deriv
49
50     def __call__(self, x):
51         self.x = x
52         self.u = x.dot(self.W) + self.b
53         h = self.activate_func(self.u)
54         return h
55
56     def back_prop(self, delta, W):
57         self.delta = self.activate_func_deriv(self.u) * delta.dot(W.T)
58         return self.delta
59
60     def compute_grad(self):
61         batch_size = self.delta.shape[0]
62         self.dW = self.x.T.dot(self.delta) / batch_size
63         self.db = self.delta.mean()
64
65 class Model(object):
66     def __init__(self, layers):
67         self.layers = layers
68
69     def __call__(self, x, is_training=False):
70         if is_training:
71             for layer in self.layers:
72                 x = layer(x)
73         else:
74             for layer in self.layers:
75                 x = x.dot(layer.W) + layer.b
76                 x = layer.activate_func(x)
77         return x
78
79     def back_propagate(self, delta):

```

```

80         W = None
81         for i, layer in enumerate(self.layers[::-1]):
82             if i == 0:
83                 layer.delta = delta
84             else:
85                 delta = layer.back_prop(delta, W)
86             layer.compute_grad()
87             W = layer.W
88
89     def update(self, lr):
90         for layer in self.layers:
91             layer.W -= lr * layer.dW
92             layer.b -= lr * layer.db
93         return

```

Listings 2: assignment1.py

```

1  # -*- coding: utf-8 -*-
2
3
4  import numpy as np
5  import matplotlib.pyplot as plt
6  from scipy.io import loadmat
7
8  from neural_network import *
9
10
11 def load_data(path):
12     data = loadmat(path)
13     #print(data.keys())
14     x = data['x'].T
15     y = data['l'].T
16     n = data['n'][0, 0]
17     d = data['d'][0, 0]
18     return x, y, n, d
19
20
21 def compute_loss(y_pred, y_true):
22     loss = np.mean((y_pred - y_true)**2)
23     return loss
24
25
26 def plot(model, x, l, xx, yy, axy):
27     # settings
28     plt.clf()
29     plt.xlim([-1, 1])
30     plt.ylim([-1, 1])
31
32     # scatter
33     plt.plot(x[np.where(l==1), 0], x[np.where(l==1), 1], 'bo')
34     plt.plot(x[np.where(l==0), 0], x[np.where(l==0), 1], 'bx')
35
36     # contour
37     p = model(axy, is_training=False) # compute classification results
38     cs = plt.contour(
39         xx, yy, np.reshape(p, xx.shape),
40         levels=[-5, 0, 5],
41         colors='g',
42     )
43     plt.clabel(cs)
44
45     # show
46     plt.show()

```

```

47     plt.pause(0.000001)
48
49
50 def train(model, x, y, lr, epochs, fig_path=None):
51     n_sample = len(y)
52
53     xx, yy = np.meshgrid(np.linspace(-1,1), np.linspace(-1,1))
54     xf = xx.flatten()[:, np.newaxis]
55     yf = yy.flatten()[:, np.newaxis]
56     axy = np.concatenate((
57         xf,
58         yf
59     ),axis=1)
60     l = (y == 1)
61     plt.figure()
62     plt.ion()
63
64     for epoch in range(epochs):
65         # forward
66         y_pred = model(x, is_training=True)
67         loss = compute_loss(y, y_pred)
68
69         # backward
70         delta = y_pred - y
71         model.back_propagate(delta)
72         model.update(lr)
73
74         # result
75         pred = y_pred
76         pred[pred > 0] = 1
77         pred[pred < 0] = -1
78         n_correct = (pred == y).sum()
79         acc = n_correct / n_sample
80         print(f'Epoch: {epoch+1}\t#Loss: {loss:.4f}\t#Correct:
{n_correct}\tAcc: {acc:.3f}')
81
82         # plot
83         #plot(model, x, l, xx, yy, axy)
84
85
86     # plot
87     plot(model, x, l, xx, yy, axy)
88     if fig_path:
89         plt.savefig(fig_path)
90     plt.show()
91
92     return model

```

```

93
94
95 def main():
96     # settings
97     data_type = 'linear'
98     #data_type = 'nonlinear'
99     #data_type = 'slinear'
100    data_path = f'../data/{data_type}-data.mat'
101    fig_path = f'../figures/assignment1_1_{data_type}_result.png'
102    np.random.seed(1)
103
104
105    # hyperparameters
106    epochs = 500
107    lr = 0.9
108
109    # model
110    input_size = 2
111    output_size = 1
112    hidden_size = 10
113    model = Model([
114        FC(input_size, hidden_size, sigmoid, deriv_sigmoid),
115        FC(hidden_size, output_size, identity_function,
116            deriv_identity_function),
117    ])
118
119    # load data
120    x, y, n, d = load_data(data_path)
121    print(f'Data Type: {data_type} #Sample: {n} #Dim: {d}\n')
122
123
124    # train
125    model = train(model, x, y, lr, epochs, fig_path=fig_path,)
126
127
128 if __name__ == '__main__':
129     main()

```

Listings 3: assignment23.py

```

1  # -*- coding: utf-8 -*-
2
3
4  import numpy as np
5  import matplotlib.pyplot as plt
6
7  import mne
8  from neural_network import *
9
10
11 def load_data():
12     train_X = mne.read_meg_data(mne.read_raw_datafile('train.meg'))
13     train_X = np.reshape(train_X, [train_X.shape[0], -1]) #flatten
14     train_y = mne.read_label(mne.read_label_file('train_label.txt'))
15
16     test_X = mne.read_meg_data(mne.read_raw_datafile('test.meg'))
17     test_X = np.reshape(test_X, [test_X.shape[0], -1]) #flatten
18     test_y = mne.read_label(mne.read_label_file('test_label.txt'))
19     return train_X, train_y, test_X, test_y
20
21
22 def add_augment_axis(data_X):
23     n = data_X.shape[0]
24     data_X_augmented = np.concatenate((
25         np.ones((n, 1)),
26         data_X
27     ), axis=1)
28     return data_X_augmented
29
30
31 def normalize(data_X):
32     #data_X -= 128
33     #data_X /= 128
34     data_X /= 256
35     return data_X
36
37
38 def split(data_X, data_y, batch_size):
39     n_data = len(data_y)
40     n_abandoned = n_data % batch_size
41     if n_abandoned != 0:
42         print(f'Warning: {n_abandoned} samples are abandoned')
43     data_X_split = [data_X[i:i+batch_size] for i in range(0, n_data,
44         batch_size)]
45     data_y_split = [data_y[i:i+batch_size] for i in range(0, n_data,
46         batch_size)]

```



```

45     return data_X_split, data_y_split
46
47
48 def train(
49     model, labels,
50     train_X, train_y,
51     valid_X, valid_y,
52     lr, epochs, batch_size,
53     mode='MSE',
54 ):
55     if mode.lower() not in ['mse', 'lms', 'widrow-hoff', 'nn']:
56         raise ValueError(f'Unknown mode: {mode}')
57     n_label = len(labels)
58     train_X_split, train_y_split = split(train_X, train_y, batch_size)
59     n_minibatch = len(train_y_split)
60     print('train')
61     for epoch in range(epochs):
62         # train
63         for i in range(n_minibatch):
64             # forward
65             y_pred = model(train_X_split[i], is_training=True)
66             y_true = np.identity(n_label)[train_y_split[i]]
67             loss = compute_loss(y_pred, y_true)
68
69             # update
70             delta = y_pred - y_true
71             if mode.lower() in ['mse', 'lms', 'widrow-hoff']:
72                 model.compute_grad(delta)
73             else:
74                 model.back_propagate(delta)
75             model.update(lr)
76
77         # validate
78         y_pred = model(valid_X, is_training=False)
79         y_pred = np.argmax(y_pred, axis=1)
80         n_correct = (y_pred == valid_y).sum()
81         acc = n_correct / len(valid_y)
82
83         print(f'Epoch: {epoch+1}      #Loss: {loss:.3f}      #Correct:
84               {n_correct}      Accuracy: {acc:.3f}')
85     print()
86     return model
87
88 def test(model, test_X, test_y, labels):
89     # settings
90     n_label = len(labels)

```

```

91     confusion_matrix = np.zeros((n_label, n_label), dtype=int)
92     n_data_all = len(test_y)
93     result = {}
94
95     print('test')
96
97     # prediction
98     y_pred = model(test_X, is_training=False)
99     y_pred = np.argmax(y_pred, axis=1)
100
101     # calc scores
102     for label in labels:
103         print(f'Label: {label}\t', end='')
104
105         indices = np.where(test_y == label)[-1]
106         n_data = len(indices)
107         preds = y_pred[indices]
108
109         # make confusion matrix
110         for i in labels:
111             n = (preds == i).sum()
112             confusion_matrix[label, i] = n
113
114         # calc accuracy
115         n_correct = confusion_matrix[label, label]
116         acc = n_correct / n_data
117         print(f'#Data: {n_data}\t#Correct: {n_correct}\tAcc: {acc:.3f}')
118
119         result[label] = {
120             'data': n_data,
121             'correct': n_correct,
122             'accuracy': acc,
123         }
124     result['confusion_matrix'] = confusion_matrix
125
126     # overall score
127     n_crr_all = np.diag(confusion_matrix).sum()
128     acc_all = n_crr_all / n_data_all
129     result['all'] = {
130         'data': n_data_all,
131         'correct': n_crr_all,
132         'accuracy': acc_all,
133     }
134     print(f'All\t#Data: {n_data_all}\t#Correct: {n_crr_all}\tAcc: {acc_all:.3f}')
135     print()
136     print('Confusion Matrix:\n', confusion_matrix)

```

```

137     print()
138     return result
139
140
141 def print_result_in_TeX_tabular_format(result):
142     labels = list(range(10))
143
144     print('Scores')
145     for label in labels:
146         print('{} & {} & {} & {:.3f} \\\\'
147             .format(
148                 label,
149                 int(result[label]['data']),
150                 int(result[label]['correct']),
151                 result[label]['accuracy']
152             ))
153     print('All & {} & {} & {:.3f} \\\\'
154         .format(
155             int(result['all']['data']),
156             int(result['all']['correct']),
157             result['all']['accuracy']
158         ))
159     print()
160
161     print('Confusion Matrix')
162     for i in labels:
163         print('{} '.format(i), end='')
164         for j in labels:
165             print(' & {}'.format(int(result['confusion_matrix'][i, j])),
166                 end='')
167         print(' \\\\'
168             .format(
169                 int(result['all']['data']),
170                 int(result['all']['correct']),
171                 result['all']['accuracy']
172             ))
173     return
174
175 def compute_loss(y_pred, y_true):
176     #loss = np.mean((y_pred - y_true)**2)
177     log_y_pred = np.log(np.clip(y_pred, 1e-8, y_pred))
178     loss = np.mean(np.sum((- y_true * log_y_pred), axis=1))
179     return loss
180
181 class Linear(object):
182     def __init__(self, input_size, output_size):
183         self.W = np.random.rand(input_size, output_size).astype(float)
184         self.dW = None
185         self.x = None
186
187     def __call__(self, x, is_training=True):
188         self.x = x

```

```

183         y = x.dot(self.W)
184         return y
185
186     def compute_grad(self, delta):
187         batch_size = delta.shape[0]
188         self.dW = self.x.T.dot(delta) / batch_size
189
190     def update(self, lr):
191         self.W -= lr * self.dW
192
193
194 def main():
195     # settings
196     #mode = 'MSE'
197     mode = 'NN'
198     valid_ratio = 1/6
199     np.random.seed(0)
200
201
202     # hyperparameters
203     hidden_size = 32 # for only MLP
204     #lr = 0.0015 # MSE
205     lr = 0.5 # NN
206     batch_size = 100
207     epochs = 1000
208
209
210     # load data
211     train_X, train_y, test_X, test_y = load_data()
212     labels = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
213     n_train = len(train_y)
214     n_valid = int(valid_ratio * n_train)
215     n_train -= n_valid
216     n_test = len(test_y)
217     d = train_X.shape[1]
218     n_labels = len(labels)
219     print(f'Mode: {mode} #Dim: {d} #Train: {n_train} #Valid: {n_valid}
220     #Test: {n_test}\n')
221
222
223     # model
224     if mode.lower() in ['mse', 'lms', 'widrow-hoff']:
225         train_X = add_augment_axis(train_X)
226         test_X = add_augment_axis(test_X)
227         model = Linear(d+1, n_labels)
228     elif mode.lower() in ['nn']:
229         model = Model([

```

```

229         FC(d, hidden_size, sigmoid, deriv_sigmoid),
230         #FC(hidden_size, hidden_size, sigmoid, deriv_sigmoid),
231         FC(hidden_size, n_labels, softmax, deriv_softmax)
232     ])
233 else:
234     raise ValueError(f'Unknown mode: {mode}')
235
236
237 # preprocess and split data
238 test_X = normalize(test_X)
239 train_X = normalize(train_X)
240
241 valid_X = train_X[n_train:]
242 valid_y = train_y[n_train:]
243 train_X = train_X[:n_train]
244 train_y = train_y[:n_train]
245
246
247 # train
248 model = train(
249     model, labels,
250     train_X, train_y,
251     valid_X, valid_y,
252     lr=lr, epochs=epochs, batch_size=batch_size,
253     mode=mode,
254 )
255
256
257 # result
258 result = test(model, test_X, test_y, labels)
259 print_result_in_TeX_tabular_format(result)
260
261
262 if __name__ == '__main__':
263     main()

```