

70. 単語ベクトルの和による特徴量

問題50で構築した学習データ, 検証データ, 評価データを行列・ベクトルに変換したい.

```
In [:] 1 """ 3データを持ってくる
2 tokenizeする
3 単語の特徴量をとる
4 平均をとる
5 xiができる
6 Xを作る
7 Yを作る"""
8 import pandas as pd
9 import numpy as np
10 import re
11 from collections import Counter
12 from tqdm import tqdm
13 from gensim.models import KeyedVectors
14 tqdm.pandas()
15 googlenews = KeyedVectors.load_word2vec_format(
16     '../data/GoogleNews-vectors-negative300.bin', binary=True)
17
```

```
In [:] 1 d = 300 # 単語ベクトルの次元
2
3 # TODO np.zerosを使う
4
5 def preprocessor(doc):
6     doc = re.sub(r"[.]", "", doc) # 記号を削除
7     doc = doc.lower() # 小文字に統一
8     return doc
9
10
11 def tokenize(doc):
12     tokens = doc.split(' ')
13     return tokens
14
15
16 def emb(token):
17     if token in googlenews:
18         return googlenews[token]
19     else:
20         return [0.0]*d
21
22
23 def get_x(tokens): # t = 0 の場合ここでnanが出てしまっていた.
24     t = len(tokens)
25     x = np.array([0.0]*d)
26     if t == 0:
27         return x
28     for token in tokens:
29         x += np.array(emb(token))
30     return x/t
31
32
33 def reduce_vocab(tokens):
34     tokens = [token for token in tokens if token in googlenews]
35     return tokens
36
37
38 def is_empty(tokens):
39     return len(tokens) == 0
40
41
42 def bag_of_tokens(doc):
43     vector = [0]*len(vocab)
44     for token in doc:
45         if token in vocab:
46             vector[vocab.index(token)] += 1
47     return pd.Series(vector)
48
49
50
```

```
In [:]
1 columns = ('category', 'title')
2 train = pd.read_csv('../data/NewsAggregatorDataset/train.txt',
3                     names=columns, sep='\t')
4 valid = pd.read_csv('../data/NewsAggregatorDataset/valid.txt',
5                     names=columns, sep='\t')
6 test = pd.read_csv('../data/NewsAggregatorDataset/test.txt',
7                    names=columns, sep='\t')
8 print(len(train))
9 # preprocess
10 train['tokens'] = train.title.progress_apply(preprocessor)
11 test['tokens'] = test.title.progress_apply(preprocessor)
12 valid['tokens'] = valid.title.progress_apply(preprocessor)
13
14 # tokenize
15 train['tokens'] = train.tokens.apply(tokenize)
16 test['tokens'] = test.tokens.apply(tokenize)
17 valid['tokens'] = valid.tokens.apply(tokenize)
18
19 # reduce vocabulary
20 train['tokens'] = train.tokens.apply(reduce_vocab)
21 test['tokens'] = test.tokens.apply(reduce_vocab)
22 valid['tokens'] = valid.tokens.apply(reduce_vocab)
23
24 train['is_empty'] = train.tokens.apply(is_empty)
25 test['is_empty'] = test.tokens.apply(is_empty)
26 valid['is_empty'] = valid.tokens.apply(is_empty)
27
28 train = train[train.is_empty == False]
29 test = test[test.is_empty == False]
30 valid = valid[valid.is_empty == False]
31
```

```
In [:]
1 X_train = np.array(train.tokens.apply(get_x).values.tolist())
2 X_valid = np.array(valid.tokens.apply(get_x).values.tolist())
3 X_test = np.array(test.tokens.apply(get_x).values.tolist())
4
5 label2int = {'b': 0, 't': 1, 'e': 2, 'm': 3}
6 Y_train = np.array(train.category.map(label2int))
7 Y_valid = np.array(valid.category.map(label2int))
8 Y_test = np.array(test.category.map(label2int))
9
10 np.save('../data/nlp2020_70/X_train', X_train)
11 np.save('../data/nlp2020_70/X_valid', X_valid)
12 np.save('../data/nlp2020_70/X_test', X_test)
13 np.save('../data/nlp2020_70/Y_train', Y_train)
14 np.save('../data/nlp2020_70/Y_valid', Y_valid)
15 np.save('../data/nlp2020_70/Y_test', Y_test)
```

1 # 71. 単層ニューラルネットワークによる予測

```
In [:]
1 import numpy as np
2 import torch
3
4 X_train = np.load('../data/nlp2020_70/X_train.npy')
5 X_valid = np.load('../data/nlp2020_70/X_valid.npy')
6 X_test = np.load('../data/nlp2020_70/X_test.npy')
7 Y_train = np.load('../data/nlp2020_70/Y_train.npy')
8 Y_valid = np.load('../data/nlp2020_70/Y_valid.npy')
9 Y_test = np.load('../data/nlp2020_70/Y_test.npy')
```

```
In [ ]: 1
2 X_train = torch.tensor(X_train, dtype=torch.float32) # 10672x300行列
3 W = torch.randn(300, 4) # 300x4行列
4 softmax = torch.nn.Softmax(dim=1)
5
6 xW = torch.matmul(X_train[:, 1], W)
7 XW = torch.matmul(X_train[:, 4], W)
8 y = softmax(xW)[0]
9 Y = softmax(XW)
10 print(y)
11 print(Y)
12 """
13
14 tensor([[0.3985, 0.0194, 0.4485, 0.1337]])
15 tensor([[0.3985, 0.0194, 0.4485, 0.1337],
16         [0.7893, 0.0462, 0.0289, 0.1356],
17         [0.8456, 0.0623, 0.0191, 0.0730],
18         [0.4550, 0.1102, 0.0192, 0.4156]])
19 """
20
```

torch.nn.Moduleを使う場合

```
In [ ]: 1 from torch import nn
2
3 class Net(nn.Module):
4     def __init__(self):
5         super().__init__()
6         self.fc = nn.Linear(300, 4) # 重みを作成
7         nn.init.xavier_uniform_(self.fc.weight) # 一様分布の乱数で重みを初期化
8
9     def forward(self, x):
10         x = self.fc(x)
11         return x
```

```
In [ ]: 1 model = Net()
2 X_train = torch.tensor(X_train, dtype=torch.float32) # 10672x300行列
3 x = model(X_train[0])
4 x = torch.softmax(x, dim=-1) # なぜdim=-1でしかできない?
5 x
6 """
7 tensor([0.2419, 0.2966, 0.2355, 0.2261], grad_fn=<SoftmaxBackward>)
8 """
```

```
In [ ]: 1 x = model(X_train[:, 4])
2 x = torch.softmax(x, dim=-1)
3 x
4 """
5 tensor([[0.2419, 0.2966, 0.2355, 0.2261],
6         [0.2209, 0.2652, 0.2306, 0.2833],
7         [0.2366, 0.2597, 0.2268, 0.2769],
8         [0.2327, 0.2713, 0.2357, 0.2604]], grad_fn=<SoftmaxBackward>)
9 """
```

72. 損失と勾配の計算

学習データの事例 x_1 と事例集合 x_1, x_2, x_3, x_4 に対して、クロスエントロピー損失と、行列 W に対する勾配を計算せよ。なお、ある事例 x_i に対して損失は次式で計算される。

$$l_i = -\log[\text{事例}x_i\text{が}y_i\text{に分類される確率}]$$

ただし、事例集合に対するクロスエントロピー損失は、その集合に含まれる各事例の損失の平均とする。

In []:

```
1 import numpy as np
2 import torch.nn.functional as F
3
4 def cross_entropy_loss(p):
5     return -1 * np.log(p)
6
7 # xi
8 x_loss = cross_entropy_loss(y[Y_train[0]])
9 print('x loss: ', x_loss)
10
11 #Xi
12 X_loss = []
13 for y, i in zip(Y, Y_train[:4]):
14     X_loss.append(cross_entropy_loss(y[i]))
15
16 # TODO 勾配を求める
17 X_loss = np.mean(X_loss)
18 print('X loss: ', X_loss)
19
20 """
21 x loss: tensor(1.9115)
22 X loss:  1.4298041
23 """
```

torch.nn.Moduleを使う場合

In []:

```
1 # 勾配の求めかたがわからなかったのでtorch.nn.moduleを使って実装
2 model = Net()
3 criterion = torch.nn.CrossEntropyLoss() # クロスエントロピー損失関数を定義
4
5 X_train = torch.tensor(X_train, dtype=torch.float32) # 10672x300行列
6 Y_train = torch.tensor(Y_train).long() # long型にする
7
8 inputs = X_train[:4]
9 labels = Y_train[:4]
10
11 outputs = model(inputs)
12 loss = criterion(outputs, labels)
13 model.zero_grad() # 勾配をゼロにする
14 loss.backward()
15 print('損失', loss)
16 print('勾配', model.fc.weight.grad)
17 """
18 損失 tensor(1.3699, grad_fn=<NllLossBackward>)
19 勾配 tensor([[ -0.0051,  0.0070, -0.0172, ..., -0.0031, -0.0094, -0.0025],
20             [ 0.0071, -0.0023,  0.0036, ..., -0.0033,  0.0029,  0.0176],
21             [-0.0080, -0.0136,  0.0235, ...,  0.0014,  0.0058, -0.0064],
22             [ 0.0060,  0.0088, -0.0099, ...,  0.0050,  0.0006, -0.0087]])
23 """
```

73. 確率的勾配降下法による学習

確率的勾配降下法（SGD: Stochastic Gradient Descent）を用いて、行列Wを学習せよ。なお、学習は適当な基準で終了させればよい（例えば「100エポックで終了」など）。

In []:

```
1 import torch.optim as optim
2 import torch.nn as nn
3 import torch
4 from torch.utils.data import TensorDataset, DataLoader
5 import copy
6 import numpy as np
7
8 X_train = np.load('../data/nlp2020_70/X_train.npy')
9 X_valid = np.load('../data/nlp2020_70/X_valid.npy')
10 X_test = np.load('../data/nlp2020_70/X_test.npy')
11 Y_train = np.load('../data/nlp2020_70/Y_train.npy')
12 Y_valid = np.load('../data/nlp2020_70/Y_valid.npy')
13 Y_test = np.load('../data/nlp2020_70/Y_test.npy')
14
15 model = Net()
16 criterion = nn.CrossEntropyLoss() # クロスエントロピー損失関数
17 optimizer = optim.SGD(model.parameters(), lr=0.01) # 確率的勾配降下法
18
19 X_train = torch.tensor(X_train, dtype=torch.float32) # float32型にする
20 Y_train = torch.tensor(Y_train).long() # long型にする
21
22 ds = TensorDataset(X_train, Y_train)
23 loader = DataLoader(ds, batch_size=1, shuffle=True)
24
25 prev_model = copy.deepcopy(model)
26 prev_optimizer = copy.deepcopy(optimizer)
27
28 for epoch in range(10):
29     for inputs, labels in loader:
30
31         outputs = model(inputs)
32         loss = criterion(outputs, labels)
33         optimizer.zero_grad()
34         loss.backward()
35         optimizer.step()
36
37         if torch.isnan(loss):
38             print(loss)
39             model = prev_model
40             optimizer = optim.SGD(model.parameters(), lr=0.01)
41             optimizer.load_state_dict(prev_optimizer.state_dict())
42         else:
43             prev_model = copy.deepcopy(model)
44             prev_optimizer = copy.deepcopy(optimizer)
45
46     print('epoch: %d loss: %f' % (epoch, loss))
47
48
49 print('Finished Training')
50
51 """
52 epoch: 0 loss: 0.606497
53 epoch: 1 loss: 0.027537
54 epoch: 2 loss: 0.718384
55 epoch: 3 loss: 1.221328
56 epoch: 4 loss: 0.476774
57 epoch: 5 loss: 0.986584
58 epoch: 6 loss: 0.080321
59 epoch: 7 loss: 0.048740
60 epoch: 8 loss: 0.053016
61 epoch: 9 loss: 0.045396
62 Finished Training
63 """
```

74. 正解率の計測

問題73で求めた行列を用いて学習データおよび評価データの事例を分類したとき、その正解率をそれぞれ求めよ。

In []:

```
1 def accuracy(pred, label):
2     pred = np.argmax(pred.data.numpy(), axis=1) # 行ごとに最大値のインデックスを取得する.
3     label = label.data.numpy()
4     return (pred == label).mean()
5
6 X_test = torch.tensor(X_test, dtype=torch.float32) #float32型にする
7 Y_test = torch.tensor(Y_test).long() # long型にする
8
9 outputs = model(X_train)
10 print (accuracy(outputs, Y_train))
11 outputs = model(X_test)
12 print (accuracy(outputs, Y_test))
13
14 ""
15 0.8758316933745666
16 0.881559220389805
17 ""
```

75. 損失と正解率のプロット

問題73のコードを改変し、各エポックのパラメータ更新が完了するたびに、訓練データでの損失、正解率、検証データでの損失、正解率をグラフにプロットし、学習の進捗状況を確認できるようにせよ。

In []:

```
1 # % matplotlib inline
2 import torch.optim as optim
3 import torch.nn as nn
4 import torch
5 from torch.utils.data import TensorDataset, DataLoader
6 import copy
7 import numpy as np
8 import matplotlib.pyplot as plt
9 from torch.utils.tensorboard import SummaryWriter
10 writer = SummaryWriter()
11
12 X_train = np.load('../data/nlp2020_70/X_train.npy')
13 X_valid = np.load('../data/nlp2020_70/X_valid.npy')
14 X_test = np.load('../data/nlp2020_70/X_test.npy')
15 Y_train = np.load('../data/nlp2020_70/Y_train.npy')
16 Y_valid = np.load('../data/nlp2020_70/Y_valid.npy')
17 Y_test = np.load('../data/nlp2020_70/Y_test.npy')
18
19 model = Net()
20 criterion = nn.CrossEntropyLoss() # クロスエントロピー損失関数
21 optimizer = optim.SGD(model.parameters(), lr=0.001) # 確率的勾配降下法
22
23 X_train = torch.tensor(X_train, dtype=torch.float32) # float32型にする
24 Y_train = torch.tensor(Y_train).long() # long型にする
25 X_valid = torch.tensor(X_valid, dtype=torch.float32) # float32型にする
26 Y_valid = torch.tensor(Y_valid).long() # long型にする
27
28 ds = TensorDataset(X_train, Y_train)
29 loader = DataLoader(ds, batch_size=1, shuffle=True)
30
31 prev_model = copy.deepcopy(model)
32 prev_optimizer = copy.deepcopy(optimizer)
33
34 loss_train_list = []
35 loss_valid_list = []
36 acc_train_list = []
37 acc_valid_list = []
38
39
40 for epoch in range(10):
41     for index, data in enumerate(loader):
42         inputs, labels = data
43         outputs = model(inputs)
44         loss = criterion(outputs, labels)
45         optimizer.zero_grad()
46         loss.backward()
47         optimizer.step()
48
49     if torch.isnan(loss):
50         model = prev_model
51         optimizer = optim.SGD(model.parameters(), lr=0.001)
52         optimizer.load_state_dict(prev_optimizer.state_dict())
53     else:
54         prev_model = copy.deepcopy(model)
55         prev_optimizer = copy.deepcopy(optimizer)
56
57     # train
58     outputs = model(X_train)
59     loss = criterion(outputs, Y_train)
60     acc = accuracy(outputs, Y_train)
61     loss_train_list.append(loss.data.numpy())
62     acc_train_list.append(acc)
63     print('train\tePOCH: %d loss: %f accuracy: %f' % (epoch, loss, acc))
64
65     # valid
66     outputs = model(X_valid)
67     loss = criterion(outputs, Y_valid)
68     acc = accuracy(outputs, Y_valid)
69     loss_valid_list.append(loss.data.numpy())
70     acc_valid_list.append(acc)
71     print('valid\tePOCH: %d loss: %f accuracy: %f' % (epoch, loss, acc))
72
73     # plot
74     fig = plt.figure(figsize=(8, 8))
75     plt.xlabel('epoch')
76     plt.ylabel('loss')
77     plt.plot(range(epoch+1), loss_train_list, label='train')
78     plt.plot(range(epoch+1), loss_valid_list, label='valid')
```

```
79 plt.legend()
80 plt.savefig('../nlp75_loss.png')
81 plt.show()
82
83
84 fig = plt.figure(figsize=(8, 8))
85 plt.xlabel('epoch')
86 plt.ylabel('accuracy')
87 plt.plot(range(epoch+1), acc_train_list, label="train")
88 plt.plot(range(epoch+1), acc_valid_list, label="valid")
89 plt.legend()
90 plt.savefig('../nlp75_accuracy.png')
91 plt.show()
92
93
94 print('Finished Training')
95
96
97
```

76. チェックポイント

問題75のコードを改変し，各エポックのパラメータ更新が完了するたびに，チェックポイント（学習途中のパラメータ（重み行列など）の値や最適化アルゴリズムの内部状態）をファイルに書き出せ

In []:

```
1 # % matplotlib inline
2 import torch.optim as optim
3 import torch.nn as nn
4 import torch
5 from torch.utils.data import TensorDataset, DataLoader
6 import copy
7 import numpy as np
8 import matplotlib.pyplot as plt
9 from torch.utils.tensorboard import SummaryWriter
10 writer = SummaryWriter()
11
12 X_train = np.load('../data/nlp2020_70/X_train.npy')
13 X_valid = np.load('../data/nlp2020_70/X_valid.npy')
14 X_test = np.load('../data/nlp2020_70/X_test.npy')
15 Y_train = np.load('../data/nlp2020_70/Y_train.npy')
16 Y_valid = np.load('../data/nlp2020_70/Y_valid.npy')
17 Y_test = np.load('../data/nlp2020_70/Y_test.npy')
18
19 model = Net()
20 criterion = nn.CrossEntropyLoss() # クロスエントロピー損失関数
21 optimizer = optim.SGD(model.parameters(), lr=0.001) # 確率的勾配降下法
22
23 X_train = torch.tensor(X_train, dtype=torch.float32) # float32型にする
24 Y_train = torch.tensor(Y_train).long() # long型にする
25
26 ds = TensorDataset(X_train, Y_train)
27 loader = DataLoader(ds, batch_size=1, shuffle=True)
28
29 prev_model = copy.deepcopy(model)
30 prev_optimizer = copy.deepcopy(optimizer)
31
32 for epoch in range(10):
33     for index, data in enumerate(loader):
34         inputs, labels = data
35         outputs = model(inputs)
36         loss = criterion(outputs, labels)
37         optimizer.zero_grad()
38         loss.backward()
39         optimizer.step()
40
41     if torch.isnan(loss):
42         model = prev_model
43         optimizer = optim.SGD(model.parameters(), lr=0.001)
44         optimizer.load_state_dict(prev_optimizer.state_dict())
45     else:
46         prev_model = copy.deepcopy(model)
47         prev_optimizer = copy.deepcopy(optimizer)
48
49     torch.save(model.state_dict(), '../nlp76_%d.model' % (epoch))
50     torch.save(optimizer.state_dict(), '../nlp76_%d.palams' % (epoch))
51     print('epoch: %d loss: %f accuracy: %f' % (epoch, loss, acc))
52
53 print('Finished Training')
54
```

In []:

```
1 model = torch.load('./nlp76_0.model')
2 model
3 """
4 odict_items([
5 ('fc.weight',
6 tensor([[[-0.1008, -0.1395, -0.1014, ..., -0.0492, 0.1527, -0.2046],
7          [ 0.1128, -0.1131, 0.0435, ..., 0.0622, 0.0962, -0.0228],
8          [-0.1079, 0.1268, -0.0365, ..., -0.0685, -0.1348, 0.0695],
9          [-0.0075, 0.0643, 0.0924, ..., 0.1210, 0.0523, 0.0239]]])),
10 ('fc.bias', tensor([ 0.4536, -0.3080, 0.4848, -0.6180]])))
11 """
12
13 params = torch.load('./nlp76_0.palams')
14 params
15 """
16 {'state': {},
17  'param_groups': [{'lr': 0.001,
18                    'momentum': 0,
19                    'dampening': 0,
20                    'weight_decay': 0,
21                    'nesterov': False,
22                    'params': [10752651200, 10752656896]]}]
23 """
```

77. ミニバッチ化

問題76のコードを改変し、B事例ごとに損失・勾配を計算し、行列Wの値を更新せよ（ミニバッチ化）。Bの値を1,2,4,8,...と変化させながら、1エポックの学習に要する時間を比較せよ。

In []:

```
1 # % matplotlib inline
2 import torch.optim as optim
3 import torch.nn as nn
4 import torch
5 from torch.utils.data import TensorDataset, DataLoader
6 import copy
7 import numpy as np
8 import matplotlib.pyplot as plt
9 import time
10
11
12 X_train = np.load('../data/nlp2020_70/X_train.npy')
13 X_valid = np.load('../data/nlp2020_70/X_valid.npy')
14 X_test = np.load('../data/nlp2020_70/X_test.npy')
15 Y_train = np.load('../data/nlp2020_70/Y_train.npy')
16 Y_valid = np.load('../data/nlp2020_70/Y_valid.npy')
17 Y_test = np.load('../data/nlp2020_70/Y_test.npy')
18
19 model = Net()
20 criterion = nn.CrossEntropyLoss() # クロスエントロピー損失関数
21 optimizer = optim.SGD(model.parameters(), lr=0.001) # 確率的勾配降下法
22
23 X_train = torch.tensor(X_train, dtype=torch.float32) # float32型にする
24 Y_train = torch.tensor(Y_train).long() # long型にする
25
26
27 prev_model = copy.deepcopy(model)
28 prev_optimizer = copy.deepcopy(optimizer)
29
30 times = []
31
32 ds = TensorDataset(X_train, Y_train)
33
34 for batch in [2**x for x in range(10)]:
35     loader = DataLoader(ds, batch_size=batch, shuffle=True)
36     start = time.time()
37
38     for epoch in range(1):
39         for index, data in enumerate(loader):
40             inputs, labels = data
41             outputs = model(inputs)
42             loss = criterion(outputs, labels)
43             optimizer.zero_grad()
44             loss.backward()
45             optimizer.step()
46
47         if torch.isnan(loss):
48             model = prev_model
49             optimizer = optim.SGD(model.parameters(), lr=0.001)
50             optimizer.load_state_dict(prev_optimizer.state_dict())
51         else:
52             prev_model = copy.deepcopy(model)
53             prev_optimizer = copy.deepcopy(optimizer)
54
55     times.append(time.time() - start)
56     print('epoch: %d loss: %f accuracy: %f' % (epoch, loss, acc))
57
58 print('Finished Training')
59
60 times
61
62 """
63 [5.0293779373168945,
64  2.6831369400024414,
65  1.2833211421966553,
66  0.7357909679412842]
67 """
```

78. GPU上での学習Permalink

問題77のコードを改変し、GPU上で学習を実行せよ。

79. 多層ニューラルネットワーク

問題78のコードを改変し、バイアス項の導入や多層化など、ニューラルネットワークの形状を変更しながら、高性能なカテゴリ分類器を構築せよ。

In []:

```
1 class Net(nn.Module):
2     def __init__(self):
3         super().__init__()
4         self.fc1 = nn.Linear(300, 128) # 重みを作成
5         self.act = nn.ReLU()
6         self.fc2 = nn.Linear(128, 4)
7         nn.init.xavier_uniform_(self.fc1.weight) # 一様分布の乱数で重みを初期化
8         nn.init.xavier_uniform_(self.fc2.weight) # 一様分布の乱数で重みを初期化
9
10    def forward(self, x):
11        x = self.fc1(x)
12        x = self.act(x)
13        x = self.fc2(x)
14        return x
```

In []:

```
1 # % matplotlib inline
2 import torch.optim as optim
3 import torch.nn as nn
4 import torch
5 from torch.utils.data import TensorDataset, DataLoader
6 import copy
7 import numpy as np
8 import matplotlib.pyplot as plt
9 from torch.utils.tensorboard import SummaryWriter
10 writer = SummaryWriter()
11
12 X_train = np.load('../data/nlp2020_70/X_train.npy')
13 X_valid = np.load('../data/nlp2020_70/X_valid.npy')
14 X_test = np.load('../data/nlp2020_70/X_test.npy')
15 Y_train = np.load('../data/nlp2020_70/Y_train.npy')
16 Y_valid = np.load('../data/nlp2020_70/Y_valid.npy')
17 Y_test = np.load('../data/nlp2020_70/Y_test.npy')
18
19 model = Net()
20 criterion = nn.CrossEntropyLoss() # クロスエントロピー損失関数
21 optimizer = optim.SGD(model.parameters(), lr=0.001) # 確率的勾配降下法
22
23 X_train = torch.tensor(X_train, dtype=torch.float32) # float32型にする
24 Y_train = torch.tensor(Y_train).long() # long型にする
25 X_valid = torch.tensor(X_valid, dtype=torch.float32) # float32型にする
26 Y_valid = torch.tensor(Y_valid).long() # long型にする
27
28 ds = TensorDataset(X_train, Y_train)
29 loader = DataLoader(ds, batch_size=8, shuffle=True)
30
31 prev_model = copy.deepcopy(model)
32 prev_optimizer = copy.deepcopy(optimizer)
33
34 loss_train_list = []
35 loss_valid_list = []
36 acc_train_list = []
37 acc_valid_list = []
38 times = []
39
40 start = time.time()
41 for epoch in range(100):
42     for index, data in enumerate(loader):
43         inputs, labels = data
44         outputs = model(inputs)
45         loss = criterion(outputs, labels)
46         optimizer.zero_grad()
47         loss.backward()
48         optimizer.step()
49
50     if torch.isnan(loss):
51         model = prev_model
52         optimizer = optim.SGD(model.parameters(), lr=0.001)
53         optimizer.load_state_dict(prev_optimizer.state_dict())
54     else:
55         prev_model = copy.deepcopy(model)
56         prev_optimizer = copy.deepcopy(optimizer)
57
58     times.append(time.time() - start)
59     fig = plt.figure(figsize=(5, 5))
60     plt.xlabel('epoch')
61     plt.ylabel('loss')
62     plt.plot(range(epoch+1), loss_train_list, label="train")
63     plt.plot(range(epoch+1), loss_valid_list, label="valid")
64     plt.legend()
65     plt.show()
66     plt.savefig('../nlp79_loss.png')
67
68     fig = plt.figure(figsize=(5, 5))
69     plt.xlabel('epoch')
70     plt.ylabel('accuracy')
71     plt.plot(range(epoch+1), acc_train_list, label="train")
72     plt.plot(range(epoch+1), acc_valid_list, label="valid")
73     plt.legend()
74     plt.show()
75     plt.savefig('../nlp79_accuracy.png')
76
77
78
```

```
79 print('Finished Training')
80
```

In []:

```
1
2
3 fig = plt.figure(figsize=(8, 8))
4 plt.xlabel('epoch')
5 plt.ylabel('loss')
6 plt.plot(range(20), loss_train_list, label="train")
7 plt.plot(range(20), loss_valid_list, label="valid")
8 plt.legend()
9 plt.savefig('../nlp79_loss.png')
10
11 fig = plt.figure(figsize=(8, 8))
12 plt.xlabel('epoch')
13 plt.ylabel('accuracy')
14 plt.plot(range(20), acc_train_list, label="train")
15 plt.plot(range(20), acc_valid_list, label="valid")
16 plt.legend()
17 plt.savefig('../nlp79_accuracy.png')
18
```