

## 80. ID番号への変換

問題51で構築した学習データ中の単語にユニークなID番号を付与したい。学習データ中で最も頻出する単語に1, 2番目に頻出する単語に2, .....といった方法で、学習データ中で2回以上出現する単語にID番号を付与せよ。そして、与えられた単語列に対して、ID番号の列を返す関数を実装せよ。ただし、出現頻度が2回未満の単語のID番号はすべて0とせよ。

```
In [ ]: 1 import pandas as pd
2 import re
3 from collections import Counter
4 from tqdm import tqdm
5 tqdm.pandas()
6
7
8 def tokenize(doc):
9     tokens = doc.split(' ')
10    return tokens
11
12
13 def normalize(doc):
14     doc = re.sub(r"[.,]", "", doc) # 記号を削除
15     doc = re.sub(r" {2,}", ' ', doc) # 2回以上続くスペースを削除
16     doc = re.sub(r" *?$", "", doc) # 行頭と行末のスペースを削除
17     doc = re.sub(r"^\s*", "", doc)
18     doc = doc.lower() # 小文字に統一
19     return doc
20
21
22 def token2id(token):
23     if token in token2id_dic:
24         return token2id_dic[token]
25     else:
26         return 0
27
28
29 columns = ('category', 'title')
30
31 train = pd.read_csv("../data/NewsAggregatorDataset/train.txt",
32                     names=columns, sep='\t')
33
34
35 docs = [normalize(doc) for doc in train.title.values.tolist()]
36 tokens = [tokenize(doc) for doc in docs]
37 tokens = sum(tokens, []) # flat list
38 counter = Counter(tokens)
39
40 token2id_dic = {}
41 vocab_size = len(counter)
42 for index, (token, freq) in enumerate(counter.most_common(), 1):
43     if freq < 2:
44         token2id_dic[token] = 0
45     else:
46         token2id_dic[token] = index
47
```

```
In [ ]: 1 token2id('the')
2 ""
3 3
4 ""
```

## 81. RNNによる予測

ID番号で表現された単語列 $x=(x_1, x_2, \dots, x_T)$ がある。ただし、 $T$ は単語列の長さ、 $x_t \in R^V$ は単語のID番号のone-hot表記である（ $V$ は単語の総数である）。再帰型ニューラルネットワーク（RNN: Recurrent Neural Network）を用い、単語列 $x$ からカテゴリ $y$ を予測するモデルとして、次式を実装せよ... [参考](https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html) ([https://pytorch.org/tutorials/intermediate/char\\_rnn\\_classification\\_tutorial.html](https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html))

```
In [ ]: 1 from torch.nn.utils.rnn import pad_sequence
2 import torch.nn as nn
3 import torch
4
5 def preprocessor(doc):
6     doc = normalize(doc)
7     tokens = tokenize(doc)
8     return tokens
9
10
11 def tokens2ids(tokens):
12     tokens = [token2id(token) for token in tokens]
13     return torch.tensor(tokens, dtype=torch.int64)
```

```
In [ ]: 1 dw = 300
2 dh = 50
3 L = 4
4
5 class RNN(nn.Module):
6     def __init__(self, data_size, hidden_size, output_size, vocab_size):
7         super(RNN, self).__init__()
8         self.emb = torch.nn.Embedding(vocab_size, data_size)
9         self.rnn = torch.nn.RNN(dw, dh, nonlinearity='relu')
10        self.liner = nn.Linear(hidden_size, output_size)
11
12
13    def forward(self, data, last_hidden):          # data: (max_len)
14        data = self.emb(data)                      # data: (max_length, dw)
15        y, hidden = self.rnn(data, last_hidden)    # y: (max_len, dh), hidden: (max_len, dh)
16        y = y[:,-1,:]
17        y = self.liner(y)
18        y = torch.softmax(y, dim=1)
19        return y, hidden
20
```

```
In [ ]: 1 train['tokens'] = train.title.apply(preprocessor)
2 X_train = train.tokens.apply(tokens2ids)
3 X_train[0]
4
5 # tensor([ 8,  0, 2416, 1604, 2143,  5, 1605,  4, 745])
```

```
In [ ]: 1 max_len = train.tokens.apply(len).max()
2 model = RNN(dw, dh, L, vocab_size)
3
4 inputs = pad_sequence(X_train, batch_first=True)
5 max_len = len(inputs[0])
6 h0 = torch.zeros(1, max_len, dh, dtype=torch.float32)
7
8 outputs, hidden = model(inputs, h0)
9 print(outputs.size())
10 print(hidden.size())
11
12 """
13 torch.Size([10672, 4])
14 torch.Size([1, 121, 50])
15 """
```

## 82. 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，問題81で構築したモデルを学習せよ．訓練データ上の損失と正解率，評価データ上の損失と正解率を表示しながらモデルを学習し，適当な基準（例えば10エポックなど）で終了させよ．

In [ ]:

```
1 columns = ('category', 'title')
2
3 train = pd.read_csv('../data/NewsAggregatorDataset/train.txt',
4                     names=columns, sep='\t')
5 test = pd.read_csv('../data/NewsAggregatorDataset/test.txt',
6                    names=columns, sep='\t')
7
8 train['tokens'] = train.title.apply(preprocessor)
9 test['tokens'] = test.title.apply(preprocessor)
10
11 X_train = train.tokens.apply(tokens2ids)
12 X_test = test.tokens.apply(tokens2ids)
13
14 label2int = {'b': 0, 't': 1, 'e': 2, 'm': 3}
15 Y_train = train.category.map(label2int)
16 Y_test = test.category.map(label2int)
17 Y_train = torch.tensor(Y_train).long()
18 Y_test = torch.tensor(Y_test).long()
19
20 dataset_size = len(train)
```

In [ ]:

```
1 from torch.utils.data import TensorDataset, DataLoader
2 import torch.optim as optim
3 import numpy as np
4
5 def accuracy(pred, label):
6     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。
7     label = label.data.numpy()
8     return (pred == label).mean()
9
10 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
11 model = RNN(dw, dh, L, vocab_size)
12 criterion = nn.CrossEntropyLoss() # クロスエントロピー損失関数
13 optimizer = optim.SGD(model.parameters(), lr=0.01) # 確率的勾配降下法
14
15 X_train = pad_sequence(X_train, batch_first=True)
16 max_len = len(X_train[0])
17 ds = TensorDataset(X_train, Y_train)
18 loader = DataLoader(ds, batch_size=1, shuffle=True)
19
20 model = model.to(device)
21
22 for epoch in range(10):
23     hidden = torch.zeros(1, max_len, dh, dtype=torch.float32)
24     n_correct = 0
25     total_loss = 0
26     for inputs, label in tqdm(loader):
27         inputs = inputs.to(device)
28         label = label.to(device)
29         outputs, hidden = model(inputs, hidden)
30         loss = criterion(outputs, label)
31         optimizer.zero_grad()
32         loss.backward()
33         optimizer.step() # パラメータを更新
34
35         total_loss += loss.data
36         outputs = np.argmax(outputs.data.numpy(), axis=1)
37         label = label.data.numpy()
38         hidden = hidden.detach()
39         if outputs == label:
40             n_correct += 1
41
42     print('epoch: %d loss: %f accuracy: %f' % (epoch, loss, n_correct/dataset_size))
43
44
45 print('Finished Training')
46
47 """
48 100%|██████████| 10672/10672 [02:33<00:00, 69.55it/s]
49 0%|          | 9/10672 [00:00<02:10, 81.85it/s]epoch: 0 loss: 1.295777 accuracy: 0.416698
50 100%|██████████| 10672/10672 [02:31<00:00, 70.39it/s]
51 0%|          | 7/10672 [00:00<02:38, 67.40it/s]epoch: 1 loss: 1.220215 accuracy: 0.407234
52 100%|██████████| 10672/10672 [02:17<00:00, 77.80it/s]
53 0%|          | 8/10672 [00:00<02:21, 75.47it/s]epoch: 2 loss: 1.665452 accuracy: 0.411544
54 100%|██████████| 10672/10672 [02:12<00:00, 80.81it/s]
55 0%|          | 8/10672 [00:00<02:21, 75.41it/s]epoch: 3 loss: 1.081051 accuracy: 0.410795
56 100%|██████████| 10672/10672 [02:38<00:00, 67.43it/s]
57 0%|          | 8/10672 [00:00<02:18, 77.02it/s]epoch: 4 loss: 1.035693 accuracy: 0.412950
58 100%|██████████| 10672/10672 [02:38<00:00, 67.30it/s]
59 0%|          | 3/10672 [00:00<06:55, 25.66it/s]epoch: 5 loss: 1.218706 accuracy: 0.416229
60 100%|██████████| 10672/10672 [02:19<00:00, 76.62it/s]
61 0%|          | 7/10672 [00:00<02:37, 67.82it/s]epoch: 6 loss: 1.135605 accuracy: 0.408452
62 100%|██████████| 10672/10672 [02:02<00:00, 86.95it/s]
63 0%|          | 9/10672 [00:00<02:08, 82.73it/s]epoch: 7 loss: 1.205219 accuracy: 0.413137
64 100%|██████████| 10672/10672 [02:12<00:00, 80.55it/s]
65 0%|          | 8/10672 [00:00<02:21, 75.44it/s]epoch: 8 loss: 1.667904 accuracy: 0.414543
66 100%|██████████| 10672/10672 [02:12<00:00, 80.39it/s]epoch: 9 loss: 1.123815 accuracy: 0.413981
67 Finished Training
68 """
```

## 83. ミニバッチ化・GPU上での学習

問題82のコードを改変し、B事例ごとに損失・勾配を計算して学習を行えるようにせよ（Bの値は適当に選べ）。また、GPU上で学習を実行せよ。

```
In [ ]:
```

```
1 from torch.utils.data import TensorDataset, DataLoader
2 import torch.optim as optim
3 import numpy as np
4
5 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
6 model = RNN(dw, dh, L, vocab_size)
7 criterion = nn.CrossEntropyLoss() # クロスエントロピー損失関数
8 optimizer = optim.SGD(model.parameters(), lr=0.01) # 確率的勾配降下法
9
10 X_train = pad_sequence(X_train, batch_first=True)
11 ds = TensorDataset(X_train, Y_train)
12 loader = DataLoader(ds, batch_size=1024, shuffle=True)
13
14 model = model.to(device)
15
16 for epoch in range(10):
17     hidden = torch.zeros(1, max_len, dh, dtype=torch.float32)
18     n_correct = 0
19     total_loss = 0
20     for inputs, labels in tqdm(loader):
21         inputs = inputs.to(device)
22         labels = labels.to(device)
23         outputs, hidden = model(inputs, hidden)
24         loss = criterion(outputs, labels)
25         optimizer.zero_grad()
26         loss.backward()
27         optimizer.step() # パラメータを更新
28
29     total_loss += loss.data
30     outputs = np.argmax(outputs.data.numpy(), axis=1)
31     labels = labels.data.numpy()
32     hidden = hidden.detach()
33     for output, label in zip(outputs, labels):
34         if output == label:
35             n_correct += 1
36
37     print('epoch: %d loss: %f accuracy: %f' % (epoch, loss, n_correct/dataset_size))
38
39
40 print('Finished Training')
41 """
42
43 100%|██████████| 11/11 [00:07<00:00, 1.53it/s]
44 0%|          | 0/11 [00:00<?, ?it/s]epoch: 0 loss: 1.363497 accuracy: 0.262650
45 100%|██████████| 11/11 [00:07<00:00, 1.57it/s]
46 0%|          | 0/11 [00:00<?, ?it/s]epoch: 1 loss: 1.324291 accuracy: 0.397395
47 100%|██████████| 11/11 [00:07<00:00, 1.44it/s]
48 0%|          | 0/11 [00:00<?, ?it/s]epoch: 2 loss: 1.300277 accuracy: 0.397395
49 100%|██████████| 11/11 [00:07<00:00, 1.41it/s]
50 0%|          | 0/11 [00:00<?, ?it/s]epoch: 3 loss: 1.280347 accuracy: 0.397395
51 100%|██████████| 11/11 [00:07<00:00, 1.56it/s]
52 0%|          | 0/11 [00:00<?, ?it/s]epoch: 4 loss: 1.279698 accuracy: 0.403298
53 100%|██████████| 11/11 [00:07<00:00, 1.55it/s]
54 0%|          | 0/11 [00:00<?, ?it/s]epoch: 5 loss: 1.278832 accuracy: 0.418572
55 100%|██████████| 11/11 [00:07<00:00, 1.50it/s]
56 0%|          | 0/11 [00:00<?, ?it/s]epoch: 6 loss: 1.287330 accuracy: 0.418572
57 100%|██████████| 11/11 [00:07<00:00, 1.53it/s]
58 0%|          | 0/11 [00:00<?, ?it/s]epoch: 7 loss: 1.281508 accuracy: 0.418572
59 100%|██████████| 11/11 [00:07<00:00, 1.55it/s]
60 0%|          | 0/11 [00:00<?, ?it/s]epoch: 8 loss: 1.274512 accuracy: 0.418572
61 100%|██████████| 11/11 [00:07<00:00, 1.38it/s]epoch: 9 loss: 1.271275 accuracy: 0.418572
62 Finished Training
63
64
65 """
```

## 84. 単語ベクトルの導入

事前学習済みの単語ベクトル（例えば、Google Newsデータセット（約1,000億単語）での学習済み単語ベクトル）で単語埋め込みemb(x)を初期化し、学習せよ。

In [ ]:

```
1 from gensim.models import KeyedVectors
2
3
4 # googlenews = KeyedVectors.load_word2vec_format(
5 #     './../data/GoogleNews-vectors-negative300.bin', binary=True)
6
7 class RNN(nn.Module):
8     def __init__(self, data_size, hidden_size, output_size, vocab_size):
9         super(RNN, self).__init__()
10        self.rnn = torch.nn.RNN(dw, dh, nonlinearity='relu')
11        self.liner = nn.Linear(hidden_size, output_size)
12
13
14    def forward(self, data, last_hidden):      # data: (max_len, dw)
15        y, hidden = self.rnn(data, last_hidden) # y: (max_len, dh), hidden: (max_len, dh)
16        y = y[:,-1,:]
17        y = self.liner(y)
18        y = torch.softmax(y, dim=1)
19        return y, hidden
20
21
22 def tokens2vec(tokens, max_len):
23     vec = []
24     for token in tokens:
25         if token in googlenews:
26             vec.append(googlenews[token])
27         else:
28             vec.append([0]*dw)
29
30     # padding
31     zeros = [0]*dw
32     vec += [zeros for _ in range(max_len-len(vec))]
33     return np.array(vec)
34
35 dataset_size = len(train)
36 max_len = train.tokens.apply(len).max()
37
38 X_train = train.tokens.progress_apply(tokens2vec, max_len=max_len).values.tolist()
39 X_train = torch.tensor(X_train, dtype=torch.float32)
40 max_len = len(X_train[0])
41
42 Y_train = torch.tensor(Y_train).long()
```

In [ ]:

```
1 from torch.utils.data import TensorDataset, DataLoader
2 import torch.optim as optim
3 import numpy as np
4
5 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
6 model = RNN(dw, dh, L, vocab_size)
7 criterion = nn.CrossEntropyLoss() # クロスエントロピー損失関数
8 optimizer = optim.SGD(model.parameters(), lr=0.01) # 確率的勾配降下法
9
10 ds = TensorDataset(X_train, Y_train)
11 loader = DataLoader(ds, batch_size=1024, shuffle=True)
12
13 model = model.to(device)
14
15 for epoch in range(10):
16     hidden = torch.zeros(1, max_len, dh, dtype=torch.float32)
17     n_correct = 0
18     total_loss = 0
19     for inputs, labels in tqdm(loader):
20         inputs = inputs.to(device)
21         labels = labels.to(device)
22         outputs, hidden = model(inputs, hidden)
23         loss = criterion(outputs, labels)
24         optimizer.zero_grad()
25         loss.backward()
26         optimizer.step() # パラメータを更新
27
28     total_loss += loss.data
29     outputs = np.argmax(outputs.data.numpy(), axis=1)
30     labels = labels.data.numpy()
31     hidden = hidden.detach()
32     for output, label in zip(outputs, labels):
33         if output == label:
34             n_correct += 1
35
36     print('epoch: %d loss: %f accuracy: %f' % (epoch, loss, n_correct/dataset_size))
37
38
39 print('Finished Training')
40 """
41
42 100%|██████████| 11/11 [00:04<00:00, 2.51it/s]
43 0%|          | 0/11 [00:00<?, ?it/s]epoch: 0 loss: 1.386395 accuracy: 0.114318
44 100%|██████████| 11/11 [00:03<00:00, 2.80it/s]
45 0%|          | 0/11 [00:00<?, ?it/s]epoch: 1 loss: 1.384714 accuracy: 0.418478
46 100%|██████████| 11/11 [00:03<00:00, 2.88it/s]
47 0%|          | 0/11 [00:00<?, ?it/s]epoch: 2 loss: 1.383234 accuracy: 0.418572
48 100%|██████████| 11/11 [00:03<00:00, 2.89it/s]
49 0%|          | 0/11 [00:00<?, ?it/s]epoch: 3 loss: 1.381624 accuracy: 0.418572
50 100%|██████████| 11/11 [00:03<00:00, 2.89it/s]
51 0%|          | 0/11 [00:00<?, ?it/s]epoch: 4 loss: 1.380694 accuracy: 0.418572
52 100%|██████████| 11/11 [00:04<00:00, 2.68it/s]
53 0%|          | 0/11 [00:00<?, ?it/s]epoch: 5 loss: 1.378607 accuracy: 0.418572
54 100%|██████████| 11/11 [00:04<00:00, 2.38it/s]
55 0%|          | 0/11 [00:00<?, ?it/s]epoch: 6 loss: 1.377145 accuracy: 0.418572
56 100%|██████████| 11/11 [00:04<00:00, 2.47it/s]
57 0%|          | 0/11 [00:00<?, ?it/s]epoch: 7 loss: 1.376932 accuracy: 0.418572
58 100%|██████████| 11/11 [00:04<00:00, 2.55it/s]
59 0%|          | 0/11 [00:00<?, ?it/s]epoch: 8 loss: 1.373910 accuracy: 0.418572
60 100%|██████████| 11/11 [00:04<00:00, 2.60it/s]epoch: 9 loss: 1.374367 accuracy: 0.418572
61 Finished Training
62 """
```

## 85. 双方向RNN・多層化

順方向と逆方向のRNNの両方を用いて入力テキストをエンコードし、モデルを学習せよ。

```

In [ ]: 1 class BidirectionalRNN(nn.Module):
2         def __init__(self, data_size, hidden_size, output_size, vocab_size):
3             super(BidirectionalRNN, self).__init__()
4             self.emb = torch.nn.Embedding(vocab_size, data_size)
5             self.rnn1 = torch.nn.RNN(data_size, hidden_size, nonlinearity='relu', bidirectional=True)
6             self.rnn2 = torch.nn.RNN(2*hidden_size, hidden_size, nonlinearity='relu', bidirectional=True)
7             self.liner = nn.Linear(2*hidden_size, output_size)
8
9
10        def forward(self, data, last_hidden):          # data: (max_len)
11            data = self.emb(data)                      # data: (max_length, dw)
12            y, hidden = self.rnn1(data, last_hidden)    # y: (max_len, dh), hidden: (max_len, dh)
13            y, hidden = self.rnn2(y, hidden)
14            y = y[:,-1,:]
15            y = self.liner(y)
16            y = torch.softmax(y, dim=1)
17            return y, hidden
18

```

```

In [ ]: 1 from torch.utils.data import TensorDataset, DataLoader
2 import torch.optim as optim
3 import numpy as np
4
5 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
6 model = BidirectionalRNN(dw, dh, L, vocab_size)
7 criterion = nn.CrossEntropyLoss() # クロスエントロピー損失関数
8 optimizer = optim.SGD(model.parameters(), lr=0.01) # 確率的勾配降下法
9
10 X_train = pad_sequence(X_train, batch_first=True)
11 max_len = len(X_train[0])
12 ds = TensorDataset(X_train, Y_train)
13 loader = DataLoader(ds, batch_size=1024, shuffle=True)
14
15 model = model.to(device)
16
17 for epoch in range(10):
18     hidden = torch.zeros(2, max_len, dh, dtype=torch.float32)
19     n_correct = 0
20     total_loss = 0
21     for inputs, labels in tqdm(loader):
22         inputs = inputs.to(device)
23         labels = labels.to(device)
24         outputs, hidden = model(inputs, hidden)
25         loss = criterion(outputs, labels)
26         optimizer.zero_grad()
27         loss.backward()
28         optimizer.step() # パラメータを更新
29
30     total_loss += loss.data
31     outputs = np.argmax(outputs.data.numpy(), axis=1)
32     labels = labels.data.numpy()
33     hidden = hidden.detach()
34     for output, label in zip(outputs, labels):
35         if output == label:
36             n_correct += 1
37
38     print('epoch: %d loss: %f accuracy: %f' % (epoch, loss, n_correct/dataset_size))
39
40
41 print('Finished Training')
42

```

## 86. 畳み込みニューラルネットワーク (CNN)

ID番号で表現された単語列 $x=(x_1, x_2, \dots, x_T)$ がある。ただし、 $T$ は単語列の長さ、 $x_t \in R^V$ は単語のID番号のone-hot表記である（ $V$ は単語の総数である）。畳み込みニューラルネットワーク（CNN: Convolutional Neural Network）を用い、単語列 $x$ からカテゴリ $y$ を予測するモデルを実装せよ



In [ ]:

```
1 from torch import nn
2 import torch
3
4 dw = 300
5 dh = 50
6 L = 4
7
8
9 class CNN(nn.Module):
10     def __init__(self, data_size, hidden_size, output_size, vocab_size):
11         super(CNN, self).__init__()
12         self.emb = torch.nn.Embedding(vocab_size, data_size)
13         self.conv = torch.nn.Conv1d(data_size, hidden_size, 3, padding=1) # in_channels, out_channels, kernel_sizes
14         self.pool = torch.nn.MaxPool1d(120)
15         self.liner_px = nn.Linear(data_size*3, hidden_size)
16         self.liner_yc = nn.Linear(hidden_size, output_size)
17         self.act = nn.ReLU()
18
19
20     def forward(self, x):
21         # x: (max_len)
22         x = self.emb(x) # x: (max_length, dw)
23         x = x.view(-1, x.shape[2], x.shape[1]) # x: (dw, max_length)
24         x = self.conv(x) # 畳み込み x: (dh, max_len)
25         p = self.act(x)
26         c = self.pool(p) # c: (dh, 1)
27         c = c.view(c.shape[0], c.shape[1]) # c: (1, dh)
28         y = self.liner_yc(c) # c: (1, L)
29         y = torch.softmax(y, dim=1)
30         return y
31
32 X_train = train.tokens.apply(tokens2ids)
33 max_len = train.tokens.apply(len).max()
34 model = CNN(dw, dh, L, vocab_size)
35
36 inputs = pad_sequence(X_train, batch_first=True)
37
38 outputs = model(inputs[:1])
39 print('output.size', outputs.size())
40 print(outputs)
41
42 """
43 output.size torch.Size([1, 4])
44 tensor([[0.1083, 0.2877, 0.4019, 0.2021]], grad_fn=<SoftmaxBackward>)
45 """
```

## 87. 確率的勾配降下法によるCNNの学習

確率的勾配降下法（SGD: Stochastic Gradient Descent）を用いて、問題86で構築したモデルを学習せよ。訓練データ上の損失と正解率、評価データ上の損失と正解率を表示しながらモデルを学習し、適当な基準（例えば10エポックなど）で終了させよ。

In [ ]:

```
1 columns = ('category', 'title')
2
3 train = pd.read_csv('../data/NewsAggregatorDataset/train.txt',
4                     names=columns, sep='\t')
5 test = pd.read_csv('../data/NewsAggregatorDataset/test.txt',
6                    names=columns, sep='\t')
7
8 train['tokens'] = train.title.apply(preprocessor)
9 test['tokens'] = test.title.apply(preprocessor)
10
11 X_train = train.tokens.apply(tokens2ids)
12 X_train = pad_sequence(X_train, batch_first=True)
13 X_test = test.tokens.apply(tokens2ids)
14 X_test = pad_sequence(X_test, batch_first=True)
15
16 label2int = {'b': 0, 't': 1, 'e': 2, 'm': 3}
17 Y_train = train.category.map(label2int)
18 Y_test = test.category.map(label2int)
19 Y_train = torch.tensor(Y_train).long()
20 Y_test = torch.tensor(Y_test).long()
21
22 max_len = train.tokens.apply(len).max()
23 dataset_size = len(train)
24
```

In [ ]:

```
1 columns = ('category', 'title')
2
3 train = pd.read_csv('../data/NewsAggregatorDataset/train.txt',
4                     names=columns, sep='\t')
5 test = pd.read_csv('../data/NewsAggregatorDataset/test.txt',
6                    names=columns, sep='\t')
7
8 train['tokens'] = train.title.apply(preprocessor)
9 test['tokens'] = test.title.apply(preprocessor)
10
11 X_train = train.tokens.apply(tokens2ids)
12 X_train = pad_sequence(X_train, batch_first=True)
13 X_test = test.tokens.apply(tokens2ids)
14 X_test = pad_sequence(X_test, batch_first=True)
15
16 label2int = {'b': 0, 't': 1, 'e': 2, 'm': 3}
17 Y_train = train.category.map(label2int)
18 Y_test = test.category.map(label2int)
19 Y_train = torch.tensor(Y_train).long()
20 Y_test = torch.tensor(Y_test).long()
21
22 max_len = train.tokens.apply(len).max()
23 dataset_size = len(train)
24
25
26 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
27 model = CNN(dw, dh, L, vocab_size)
28 criterion = nn.CrossEntropyLoss() # クロスエントロピー損失関数
29 optimizer = optim.SGD(model.parameters(), lr=0.01) # 確率的勾配降下法
30
31 ds = TensorDataset(X_train, Y_train)
32 loader = DataLoader(ds, batch_size=1024, shuffle=True)
33
34 model = model.to(device)
35
36 for epoch in range(10):
37     n_correct = 0
38     total_loss = 0
39     for inputs, labels in tqdm(loader):
40         inputs = inputs.to(device)
41         labels = labels.to(device)
42
43         outputs = model(inputs)
44         loss = criterion(outputs, labels)
45         optimizer.zero_grad()
46         loss.backward()
47         optimizer.step() # パラメータを更新
48
49         total_loss += loss.data
50         outputs = np.argmax(outputs.data.numpy(), axis=1)
51         labels = labels.data.numpy()
52         for output, label in zip(outputs, labels):
53             if output == label:
54                 n_correct += 1
55
56     print('epoch: %d loss: %f accuracy: %f' % (epoch, loss, n_correct/dataset_size))
57
58
59 print('Finished Training')
60
61 """
62 100%|██████████| 11/11 [00:14<00:00, 1.36s/it]
63   0%|          | 0/11 [00:00<?, ?it/s]epoch: 0 loss: 1.324250 accuracy: 0.366567
64 100%|██████████| 11/11 [00:15<00:00, 1.42s/it]
65   0%|          | 0/11 [00:00<?, ?it/s]epoch: 1 loss: 1.291704 accuracy: 0.397395
66 100%|██████████| 11/11 [00:18<00:00, 1.69s/it]
67   0%|          | 0/11 [00:00<?, ?it/s]epoch: 2 loss: 1.272223 accuracy: 0.456241
68 100%|██████████| 11/11 [00:16<00:00, 1.49s/it]
69   0%|          | 0/11 [00:00<?, ?it/s]epoch: 3 loss: 1.263339 accuracy: 0.484726
70 100%|██████████| 11/11 [00:23<00:00, 2.10s/it]
71   0%|          | 0/11 [00:00<?, ?it/s]epoch: 4 loss: 1.244924 accuracy: 0.500281
72 100%|██████████| 11/11 [00:16<00:00, 1.52s/it]
73   0%|          | 0/11 [00:00<?, ?it/s]epoch: 5 loss: 1.251264 accuracy: 0.479854
74 100%|██████████| 11/11 [00:14<00:00, 1.35s/it]
75   0%|          | 0/11 [00:00<?, ?it/s]epoch: 6 loss: 1.244529 accuracy: 0.523145
76 100%|██████████| 11/11 [00:15<00:00, 1.39s/it]
77   0%|          | 0/11 [00:00<?, ?it/s]epoch: 7 loss: 1.243323 accuracy: 0.515555
78 100%|██████████| 11/11 [00:21<00:00, 1.98s/it]
```

```
79 0%|          | 0/11 [00:00<?, ?it/s]epoch: 8 loss: 1.242815 accuracy: 0.535139
80 100%|██████████| 11/11 [00:14<00:00, 1.36s/it]epoch: 9 loss: 1.226233 accuracy: 0.531297
81 Finished Training
82 ""
```

## 88. パラメータチューニング

問題85や問題87のコードを改変し、ニューラルネットワークの形状やハイパーパラメータを調整しながら、高性能なカテゴリ分類器を構築せよ。

## 89. 事前学習済み言語モデルからの転移学習

事前学習済み言語モデル（例えばBERTなど）を出発点として、ニュース記事見出しをカテゴリに分類するモデルを構築せよ。

```
In [102]: 1 from transformers import BertTokenizer, BertForSequenceClassification
          2 import torch
          3
          4 class Bert(nn.Module):
          5     def __init__(self):
          6         super().__init__()
          7         # self.bert =
          8
          9     def forward(self, data):
         10         x = self.bert(data)
         11         return x
         12
         13
         14 tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
         15 model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=4)
         16
         17 # inputs = torch.tensor(tokenizer.encode("Hello, my dog is cute", add_special_tokens=True)).unsqueeze(0) # Batch size 1
         18 inputs = X_train[0].unsqueeze(0)
         19 # labels = torch.tensor([1]).unsqueeze(0) # Batch size 1
         20 labels = Y_train[0]
         21 outputs = model(inputs, labels=labels)
         22
         23 loss = outputs[0]
```

```
In [103]: 1 loss
```

```
Out[103]: tensor(0.6970, grad_fn=<NllLossBackward>)
```