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 [ ]:

```python
import pandas as pd
import re
from collections import Counter
from tqdm import tqdm
tqdm.pandas()


def tokenize(doc):
    tokens = doc.split(' ')
    return tokens


def normalize(doc):
    doc = re.sub(r"[',.]", '', doc)    # 記号を削除
    doc = re.sub(r" {2,}", ' ', doc)  # 2回以上続くスペースを削除
    doc = re.sub(r" *?$", '', doc)     # 行頭と行末のスペースを削除
    doc = re.sub(r"^ *?", '', doc)
    doc = doc.lower()                  # 小文字に統一
    return doc


def token2id(token):
    if token in token2id_dic:
        return token2id_dic[token]
    else:
        return 0


columns = ('category', 'title')

train = pd.read_csv('../../data/NewsAggregatorDataset/train.txt',
                    names=columns, sep='\t')


docs = [normalize(doc) for doc in train.title.values.tolist()]
tokens = [tokenize(doc) for doc in docs]
tokens = sum(tokens, [])  # flat list
counter = Counter(tokens)

token2id_dic = {}
vocab_size = len(counter)
for index, (token, freq) in enumerate(counter.most_common(), 1):
    if freq < 2:
        token2id_dic[token] = 0
    else:
        token2id_dic[token] = index

```

In [ ]:

```python
token2id('the')
'''
3
'''
```

81. RNNによる予測

ID番号で表現された単語列x=(x1,x2,…,xT)がある．ただし，Tは単語列の長さ，xt∈ℝV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)

In [ ]:
```python
from torch.nn.utils.rnn import pad_sequence
import torch.nn as nn
import torch

def preprocessor(doc):
    doc = normalize(doc)
    tokens = tokenize(doc)
    return tokens


def tokens2ids(tokens):
    tokens = [token2id(token) for token in tokens]
    return torch.tensor(tokens, dtype=torch.int64)
```

In [ ]:
```python
dw = 300
dh = 50
L = 4

class RNN(nn.Module):
    def __init__(self, data_size, hidden_size, output_size, vocab_size):
        super(RNN, self).__init__()
        self.emb = torch.nn.Embedding(vocab_size, data_size)
        self.rnn = torch.nn.RNN(dw, dh, nonlinearity='relu')
        self.liner = nn.Linear(hidden_size, output_size)


    def forward(self, data, last_hidden):           # data: (max_len)
        data = self.emb(data)                       # data: (max_length, dw)
        y, hidden = self.rnn(data, last_hidden)     # y: (max_len, dh), hidden: (max_len, dh)
        y = y[:,-1,:]
        y = self.liner(y)
        y = torch.softmax(y, dim=1)
        return y, hidden

```

In [ ]:
```python
train['tokens'] = train.title.apply(preprocessor)
X_train = train.tokens.apply(tokens2ids)
X_train[0]

# tensor([   8,    0, 2416, 1604, 2143,    5, 1605,    4,  745])
```

In [ ]:
```python
max_len = train.tokens.apply(len).max()
model = RNN(dw, dh, L, vocab_size)

inputs = pad_sequence(X_train, batch_first=True)
max_len = len(inputs[0])
h0 = torch.zeros(1, max_len, dh, dtype=torch.float32)

outputs, hidden = model(inputs, h0)
print(outputs.size())
print(hidden.size())

'''
torch.Size([10672, 4])
torch.Size([1, 121, 50])
'''
```

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）で終了させせよ．

In [ ]:

```
columns = ('category', 'title')

train = pd.read_csv('../../data/NewsAggregatorDataset/train.txt',
                    names=columns, sep='\t')
test = pd.read_csv('../../data/NewsAggregatorDataset/test.txt',
                   names=columns, sep='\t')

train['tokens'] = train.title.apply(preprocessor)
test['tokens'] = test.title.apply(preprocessor)

X_train = train.tokens.apply(tokens2ids)
X_test = test.tokens.apply(tokens2ids)

label2int = {'b': 0, 't': 1, 'e': 2, 'm': 3}
Y_train = train.category.map(label2int)
Y_test = test.category.map(label2int)
Y_train = torch.tensor(Y_train).long()
Y_test = torch.tensor(Y_test).long()

dataset_size = len(train)
```

In [ ]:

```python
from torch.utils.data import TensorDataset, DataLoader
import torch.optim as optim
import numpy as np

def accuracy(pred, label):
    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.
    label = label.data.numpy()
    return (pred == label).mean()

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = RNN(dw, dh, L, vocab_size)
criterion = nn.CrossEntropyLoss()  # クロスエントロピー損失関数
optimizer = optim.SGD(model.parameters(), lr=0.01)  # 確率的勾配降下法

X_train = pad_sequence(X_train, batch_first=True)
max_len = len(X_train[0])
ds = TensorDataset(X_train, Y_train)
loader = DataLoader(ds, batch_size=1, shuffle=True)

model   = model.to(device)

for epoch in range(10):
    hidden = torch.zeros(1, max_len, dh, dtype=torch.float32)
    n_correct = 0
    total_loss = 0
    for inputs, label in tqdm(loader):
        inputs = inputs.to(device)
        label = label.to(device)
        outputs, hidden = model(inputs, hidden)
        loss = criterion(outputs, label)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step() # パラメータを更新

        total_loss += loss.data
        outputs = np.argmax(outputs.data.numpy(), axis=1)
        label = label.data.numpy()
        hidden = hidden.detach()
        if outputs == label:
            n_correct += 1

    print('epoch: %d loss: %f accuracy: %f' % (epoch, loss, n_correct/dataset_size))


print('Finished Training')

'''
100%|██████████| 10672/10672 [02:33<00:00, 69.55it/s]
  0%|          | 9/10672 [00:00<02:10, 81.85it/s]epoch: 0 loss: 1.295777 accuracy: 0.416698
100%|██████████| 10672/10672 [02:31<00:00, 70.39it/s]
  0%|          | 7/10672 [00:00<02:38, 67.40it/s]epoch: 1 loss: 1.220215 accuracy: 0.407234
100%|██████████| 10672/10672 [02:17<00:00, 77.80it/s]
  0%|          | 8/10672 [00:00<02:21, 75.47it/s]epoch: 2 loss: 1.665452 accuracy: 0.411544
100%|██████████| 10672/10672 [02:12<00:00, 80.81it/s]
  0%|          | 8/10672 [00:00<02:21, 75.41it/s]epoch: 3 loss: 1.081051 accuracy: 0.410795
100%|██████████| 10672/10672 [02:38<00:00, 67.43it/s]
  0%|          | 8/10672 [00:00<02:18, 77.02it/s]epoch: 4 loss: 1.035693 accuracy: 0.412950
100%|██████████| 10672/10672 [02:38<00:00, 67.30it/s]
  0%|          | 3/10672 [00:00<06:55, 25.66it/s]epoch: 5 loss: 1.218706 accuracy: 0.416229
100%|██████████| 10672/10672 [02:19<00:00, 76.62it/s]
  0%|          | 7/10672 [00:00<02:37, 67.82it/s]epoch: 6 loss: 1.135605 accuracy: 0.408452
100%|██████████| 10672/10672 [02:02<00:00, 86.95it/s]
  0%|          | 9/10672 [00:00<02:08, 82.73it/s]epoch: 7 loss: 1.205219 accuracy: 0.413137
100%|██████████| 10672/10672 [02:12<00:00, 80.55it/s]
  0%|          | 8/10672 [00:00<02:21, 75.44it/s]epoch: 8 loss: 1.667904 accuracy: 0.414543
100%|██████████| 10672/10672 [02:12<00:00, 80.39it/s]epoch: 9 loss: 1.123815 accuracy: 0.413981
Finished Training
'''
```

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 [ ]:

```
from torch.utils.data import TensorDataset, DataLoader
import torch.optim as optim
import numpy as np

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = RNN(dw, dh, L, vocab_size)
criterion = nn.CrossEntropyLoss()  # クロスエントロピー損失関数
optimizer = optim.SGD(model.parameters(), lr=0.01)  # 確率的勾配降下法

X_train = pad_sequence(X_train, batch_first=True)
ds = TensorDataset(X_train, Y_train)
loader = DataLoader(ds, batch_size=1024, shuffle=True)

model   = model.to(device)

for epoch in range(10):
    hidden = torch.zeros(1, max_len, dh, dtype=torch.float32)
    n_correct = 0
    total_loss = 0
    for inputs, labels in tqdm(loader):
        inputs = inputs.to(device)
        labels = labels.to(device)
        outputs, hidden = model(inputs, hidden)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step() # パラメータを更新

        total_loss += loss.data
        outputs = np.argmax(outputs.data.numpy(), axis=1)
        labels = labels.data.numpy()
        hidden = hidden.detach()
        for output, label in zip(outputs, labels):
            if output == label:
                n_correct += 1

    print('epoch: %d loss: %f accuracy: %f' % (epoch, loss, n_correct/dataset_size))


print('Finished Training')
'''

100%|██████████| 11/11 [00:07<00:00,  1.53it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 0 loss: 1.363497 accuracy: 0.262650
100%|██████████| 11/11 [00:07<00:00,  1.57it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 1 loss: 1.324291 accuracy: 0.397395
100%|██████████| 11/11 [00:07<00:00,  1.44it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 2 loss: 1.300277 accuracy: 0.397395
100%|██████████| 11/11 [00:07<00:00,  1.41it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 3 loss: 1.280347 accuracy: 0.397395
100%|██████████| 11/11 [00:07<00:00,  1.56it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 4 loss: 1.279698 accuracy: 0.403298
100%|██████████| 11/11 [00:07<00:00,  1.55it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 5 loss: 1.278832 accuracy: 0.418572
100%|██████████| 11/11 [00:07<00:00,  1.50it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 6 loss: 1.287330 accuracy: 0.418572
100%|██████████| 11/11 [00:07<00:00,  1.53it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 7 loss: 1.281508 accuracy: 0.418572
100%|██████████| 11/11 [00:07<00:00,  1.55it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 8 loss: 1.274512 accuracy: 0.418572
100%|██████████| 11/11 [00:07<00:00,  1.38it/s]epoch: 9 loss: 1.271275 accuracy: 0.418572
Finished Training


'''
```

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 [ ]:
```
from gensim.models import KeyedVectors


# googlenews = KeyedVectors.load_word2vec_format(
#     '../../data/GoogleNews-vectors-negative300.bin', binary=True)

class RNN(nn.Module):
    def __init__(self, data_size, hidden_size, output_size, vocab_size):
        super(RNN, self).__init__()
        self.rnn = torch.nn.RNN(dw, dh, nonlinearity='relu')
        self.liner = nn.Linear(hidden_size, output_size)


    def forward(self, data, last_hidden):           # data: (max_len, dw)
        y, hidden = self.rnn(data, last_hidden)      # y: (max_len, dh), hidden: (max_len, dh)
        y = y[:,-1,:]
        y = self.liner(y)
        y = torch.softmax(y, dim=1)
        return y, hidden


def tokens2vec(tokens, max_len):
    vec = []
    for token in tokens:
        if token in googlenews:
            vec.append(googlenews[token])
        else:
            vec.append([0]*dw)

    # padding
    zeros = [0]*dw
    vec += [zeros for _ in range(max_len-len(vec))]
    return np.array(vec)

dataset_size = len(train)
max_len = train.tokens.apply(len).max()

X_train = train.tokens.progress_apply(tokens2vec, max_len=max_len).values.tolist()
X_train = torch.tensor(X_train, dtype=torch.float32)
max_len = len(X_train[0])

Y_train = torch.tensor(Y_train).long()
```

In [ ]:

```python
from torch.utils.data import TensorDataset, DataLoader
import torch.optim as optim
import numpy as np

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = RNN(dw, dh, L, vocab_size)
criterion = nn.CrossEntropyLoss()  # クロスエントロピー損失関数
optimizer = optim.SGD(model.parameters(), lr=0.01)  # 確率的勾配降下法

ds = TensorDataset(X_train, Y_train)
loader = DataLoader(ds, batch_size=1024, shuffle=True)

model = model.to(device)

for epoch in range(10):
    hidden = torch.zeros(1, max_len, dh, dtype=torch.float32)
    n_correct = 0
    total_loss = 0
    for inputs, labels in tqdm(loader):
        inputs = inputs.to(device)
        labels = labels.to(device)
        outputs, hidden = model(inputs, hidden)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step() # パラメータを更新

        total_loss += loss.data
        outputs = np.argmax(outputs.data.numpy(), axis=1)
        labels = labels.data.numpy()
        hidden = hidden.detach()
        for output, label in zip(outputs, labels):
            if output == label:
                n_correct += 1

    print('epoch: %d loss: %f accuracy: %f' % (epoch, loss, n_correct/dataset_size))


print('Finished Training')
'''

100%|██████████| 11/11 [00:04<00:00,  2.51it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 0 loss: 1.386395 accuracy: 0.114318
100%|██████████| 11/11 [00:03<00:00,  2.80it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 1 loss: 1.384714 accuracy: 0.418478
100%|██████████| 11/11 [00:03<00:00,  2.88it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 2 loss: 1.383234 accuracy: 0.418572
100%|██████████| 11/11 [00:03<00:00,  2.89it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 3 loss: 1.381624 accuracy: 0.418572
100%|██████████| 11/11 [00:03<00:00,  2.89it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 4 loss: 1.380694 accuracy: 0.418572
100%|██████████| 11/11 [00:04<00:00,  2.68it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 5 loss: 1.378607 accuracy: 0.418572
100%|██████████| 11/11 [00:04<00:00,  2.38it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 6 loss: 1.377145 accuracy: 0.418572
100%|██████████| 11/11 [00:04<00:00,  2.47it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 7 loss: 1.376932 accuracy: 0.418572
100%|██████████| 11/11 [00:04<00:00,  2.55it/s]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 8 loss: 1.373910 accuracy: 0.418572
100%|██████████| 11/11 [00:04<00:00,  2.60it/s]epoch: 9 loss: 1.374367 accuracy: 0.418572
Finished Training
'''
```

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 [ ]:

```python
class BidirectionalRNN(nn.Module):
    def __init__(self, data_size, hidden_size, output_size, vocab_size):
        super(BidirectionalRNN, self).__init__()
        self.emb = torch.nn.Embedding(vocab_size, data_size)
        self.rnn1 = torch.nn.RNN(data_size, hidden_size, nonlinearity='relu', bidirectional=True)
        self.rnn2 = torch.nn.RNN(2*hidden_size, hidden_size, nonlinearity='relu', bidirectional=True)
        self.liner = nn.Linear(2*hidden_size, output_size)


    def forward(self, data, last_hidden):           # data: (max_len)
        data = self.emb(data)                       # data: (max_length, dw)
        y, hidden = self.rnn1(data, last_hidden)     # y: (max_len, dh), hidden: (max_len, dh)
        y, hidden = self.rnn2(y, hidden)
        y = y[:,-1,:]
        y = self.liner(y)
        y = torch.softmax(y, dim=1)
        return y, hidden

```

In [ ]:

```python
from torch.utils.data import TensorDataset, DataLoader
import torch.optim as optim
import numpy as np

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = BidirectionalRNN(dw, dh, L, vocab_size)
criterion = nn.CrossEntropyLoss()  # クロスエントロピー損失関数
optimizer = optim.SGD(model.parameters(), lr=0.01)  # 確率的勾配降下法

X_train = pad_sequence(X_train, batch_first=True)
max_len = len(X_train[0])
ds = TensorDataset(X_train, Y_train)
loader = DataLoader(ds, batch_size=1024, shuffle=True)

model   = model.to(device)

for epoch in range(10):
    hidden = torch.zeros(2, max_len, dh, dtype=torch.float32)
    n_correct = 0
    total_loss = 0
    for inputs, labels in tqdm(loader):
        inputs = inputs.to(device)
        labels = labels.to(device)
        outputs, hidden = model(inputs, hidden)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step() # パラメータを更新

        total_loss += loss.data
        outputs = np.argmax(outputs.data.numpy(), axis=1)
        labels = labels.data.numpy()
        hidden = hidden.detach()
        for output, label in zip(outputs, labels):
            if output == label:
                n_correct += 1

    print('epoch: %d loss: %f accuracy: %f' % (epoch, loss, n_correct/dataset_size))


print('Finished Training')

```

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，$x_t \in \mathbb{R}^V$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 [ ]:

```python
from torch import nn
import torch

dw = 300
dh = 50
L = 4


class CNN(nn.Module):
    def __init__(self, data_size, hidden_size, output_size, vocab_size):
        super(CNN, self).__init__()
        self.emb = torch.nn.Embedding(vocab_size, data_size)
        self.conv = torch.nn.Conv1d(data_size, hidden_size, 3, padding=1) # in_channels, out_channels, kernel_sizes
        self.pool = torch.nn.MaxPool1d(120)
        self.liner_px = nn.Linear(data_size*3, hidden_size)
        self.liner_yc = nn.Linear(hidden_size, output_size)
        self.act = nn.ReLU()


    def forward(self, x):                    # x: (max_len)
        x = self.emb(x)                      # x: (max_length, dw)
        x = x.view(-1, x.shape[2], x.shape[1])  # x: (dw, max_length)
        x = self.conv(x)                     # 畳み込み x: (dh, max_len)
        p = self.act(x)
        c = self.pool(p)                     # c: (dh, 1)
        c = c.view(c.shape[0], c.shape[1])       # c: (1, dh)
        y = self.liner_yc(c)                 # c: (1, L)
        y = torch.softmax(y, dim=1)
        return y


X_train = train.tokens.apply(tokens2ids)
max_len = train.tokens.apply(len).max()
model = CNN(dw, dh, L, vocab_size)

inputs = pad_sequence(X_train, batch_first=True)

outputs = model(inputs[:1])
print('output.size', outputs.size())
print(outputs)

'''
output.size torch.Size([1, 4])
tensor([[0.1083, 0.2877, 0.4019, 0.2021]], grad_fn=<SoftmaxBackward>)
'''
```

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 [ ]:

```
columns = ('category', 'title')

train = pd.read_csv('../../data/NewsAggregatorDataset/train.txt',
                    names=columns, sep='\t')
test = pd.read_csv('../../data/NewsAggregatorDataset/test.txt',
                   names=columns, sep='\t')

train['tokens'] = train.title.apply(preprocessor)
test['tokens'] = test.title.apply(preprocessor)

X_train = train.tokens.apply(tokens2ids)
X_train = pad_sequence(X_train, batch_first=True)
X_test = test.tokens.apply(tokens2ids)
X_test = pad_sequence(X_test, batch_first=True)

label2int = {'b': 0, 't': 1, 'e': 2, 'm': 3}
Y_train = train.category.map(label2int)
Y_test = test.category.map(label2int)
Y_train = torch.tensor(Y_train).long()
Y_test = torch.tensor(Y_test).long()

max_len = train.tokens.apply(len).max()
dataset_size = len(train)

```

In [ ]:

```python
columns = ('category', 'title')

train = pd.read_csv('../../data/NewsAggregatorDataset/train.txt',
                    names=columns, sep='\t')
test = pd.read_csv('../../data/NewsAggregatorDataset/test.txt',
                   names=columns, sep='\t')

train['tokens'] = train.title.apply(preprocessor)
test['tokens'] = test.title.apply(preprocessor)

X_train = train.tokens.apply(tokens2ids)
X_train = pad_sequence(X_train, batch_first=True)
X_test = test.tokens.apply(tokens2ids)
X_test = pad_sequence(X_test, batch_first=True)

label2int = {'b': 0, 't': 1, 'e': 2, 'm': 3}
Y_train = train.category.map(label2int)
Y_test = test.category.map(label2int)
Y_train = torch.tensor(Y_train).long()
Y_test = torch.tensor(Y_test).long()

max_len = train.tokens.apply(len).max()
dataset_size = len(train)


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNN(dw, dh, L, vocab_size)
criterion = nn.CrossEntropyLoss()  # クロスエントロピー損失関数
optimizer = optim.SGD(model.parameters(), lr=0.01)  # 確率的勾配降下法

ds = TensorDataset(X_train, Y_train)
loader = DataLoader(ds, batch_size=1024, shuffle=True)

model   = model.to(device)

for epoch in range(10):
    n_correct = 0
    total_loss = 0
    for inputs, labels in tqdm(loader):
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = model(inputs)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step() # パラメータを更新

        total_loss += loss.data
        outputs = np.argmax(outputs.data.numpy(), axis=1)
        labels = labels.data.numpy()
        for output, label in zip(outputs, labels):
            if output == label:
                n_correct += 1

    print('epoch: %d loss: %f accuracy: %f' % (epoch, loss, n_correct/dataset_size))


print('Finished Training')

'''
100%|██████████| 11/11 [00:14<00:00,  1.36s/it]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 0 loss: 1.324250 accuracy: 0.366567
100%|██████████| 11/11 [00:15<00:00,  1.42s/it]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 1 loss: 1.291704 accuracy: 0.397395
100%|██████████| 11/11 [00:18<00:00,  1.69s/it]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 2 loss: 1.272223 accuracy: 0.456241
100%|██████████| 11/11 [00:16<00:00,  1.49s/it]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 3 loss: 1.263339 accuracy: 0.484726
100%|██████████| 11/11 [00:23<00:00,  2.10s/it]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 4 loss: 1.244924 accuracy: 0.500281
100%|██████████| 11/11 [00:16<00:00,  1.52s/it]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 5 loss: 1.251264 accuracy: 0.479854
100%|██████████| 11/11 [00:14<00:00,  1.35s/it]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 6 loss: 1.244529 accuracy: 0.523145
100%|██████████| 11/11 [00:15<00:00,  1.39s/it]
  0%|          | 0/11 [00:00<?, ?it/s]epoch: 7 loss: 1.243323 accuracy: 0.515555
100%|██████████| 11/11 [00:21<00:00,  1.98s/it]
```

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

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 [ ]:

```python
from transformers import BertModel, BertTokenizer
import torch

class Bert(nn.Module):
    def __init__(self):
        super().__init__()
        self.bert = 

    def forward(self, data):
        x = self.bert(data)
        return x

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

input_ids = torch.tensor(tokenizer.encode("Hello, my dog is cute", add_special_tokens=True)).unsqueeze(0)  # Bat
outputs = model(input_ids)

last_hidden_states = outputs[0]
```