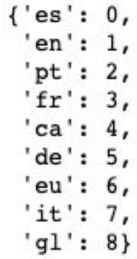
EE 511 Homework Assignment 5 Solutions

1(a) and 1(b)

First I created a language map to denote language by index:



```
{'es': 0,
 'en': 1,
 'pt': 2,
 'fr': 3,
 'ca': 4,
 'de': 5,
 'eu': 6,
 'it': 7,
 'gl': 8}
```

I created a vocabulary using training data, I selected the characters from the text that occurs at least 10 times. Below are the size and 'out of vocabulary' percentage:

size of vocablury: 509
out of vocablury tokens in training set:0.05%
out of vocablury tokens in validation set:0.06%

Then I computed the perplexity of the distribution using validation data and below formula:

$$-\frac{1}{n}\sum_{i=1}^{n}\log_2 p(x_i)$$

Below are the cross_entropy and perplexity:

cross_entropy: 5.0599
perplexity:33.357

2:

I used CNN model, I started with full length tweets but later found out that using 140 characters limit works reasonably faster. Then I replaced all the characters in the tweets with corresponding indexes of the characters from the vocabulary and made them vectorized sequences. I used 'out of vocabulary' special characters to represent missing characters from the vocabulary.

I converted all the vectorized sequences into tensor sequences using torch.LongTensor and length 140 with added 0 padding at the end of each sequence. Also converted labels into a tensors.

Then, I created my CNN model using the Embedding layer, Convolution and Max pooling layers and output layer. I have used ReLU as my activation function and used BatchNorm2d for normalization. The model is pasted below:

```
Net(
  (embedding): Embedding(509, 14)
  (layer1): Sequential(
    (0): Conv2d(1, 4, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (1): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(4, 8, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (1): ReLU()
    (2): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (drop_out): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=72, out_features=9, bias=True)
)
```

The next thing is optimizer, I used Adam optimizer and tried different learning rates and found 0.07 working pretty well on training and validation data. I used CrossEntropyLoss as my loss function.

During training I have tried multiple options, let me try to summarize here:

1. I tried different tweet lengths and found 140 to be working reasonable faster without compromising accuracy
2. I tried different combinations of input output channels and kernel_size in Convolution layers
3. I also found stride=2 and padding=2 are working better with kernel_size=5
4. I tried different batch sizes and different epoch values and found batch_size=10000 and epoch=25 reasonably faster without compromising on accuracy
5. And I tried different learning rates and found 0.07 to be working well on training and validation data.

I finally could achieve 77.82% accuracy on my test data with a cross entropy loss of 0.6876.

importing important libraries including numpy, pandas and pyTorch

In [1]:

```
import numpy as np
import pandas as pd
import math

# for evaluating the model
from sklearn.metrics import accuracy_score
from tqdm import tqdm

# PyTorch libraries and modules
import torch
import torch.nn as nn
from torch.autograd import Variable
import torch.nn.functional as F
from torch.nn import Module, CrossEntropyLoss
from torch.optim import Adam, SGD
```

Load the data, convert language labels into int form

In [2]:

```
df_train= pd.read_csv('/Users/neha/Downloads/Neha Project/Assignment5/train
df_val= pd.read_csv('/Users/neha/Downloads/Neha Project/Assignment5/val.csv
df_test= pd.read_csv('/Users/neha/Downloads/Neha Project/Assignment5/test.c
```

In [3]:

```
language_map = {}
i = 0
for lan in df_train.language.unique():
    language_map[lan] = i
    i = i+1
df_train.language.replace(language_map, inplace=True)
df_val.language.replace(language_map, inplace=True)
df_test.language.replace(language_map, inplace=True)
language_map
```

Out[3]:

```
{'es': 0,
 'en': 1,
 'pt': 2,
 'fr': 3,
 'ca': 4,
 'de': 5,
 'eu': 6,
 'it': 7,
 'gl': 8}
```

Creating a vocabulury and computing the perplexity for this distribution using the validation data.

In [15]:

```
def text_to_vocab(text_arr):
    v = {}
    all_freq = {}
    for text in text_arr:
        for i in text:
            if i in all_freq:
                all_freq[i] += 1
            else:
                all_freq[i] = 1
    v[spl_char] = 0
    for i in all_freq:
        if(all_freq[i] >= 10):
            v[i] = all_freq[i]
        else:
            v[spl_char] = v[spl_char] + all_freq[i]
    return v

spl_char = 'क'
vocab = {}

train_tweets = []
for i in range(df_train.shape[0]):
    text = df_train.document[i]
    train_tweets.append(text)

vocab = text_to_vocab(train_tweets)

relative_freq = {}
s = sum(vocab.values())
for x in vocab:
    relative_freq[x] = vocab[x]/s

val_tweets = []
for i in range(df_val.shape[0]):
    text = df_val.document[i]
    val_tweets.append(text)


logpx = []
oovv = 0
totalv = 0
for t in val_tweets:
    for x in t:
        if x in relative_freq:
            totalv = totalv+1
            logpx.append(math.log2(relative_freq[x]))
        else:
            oovv = oovv + 1
            logpx.append(math.log2(relative_freq[spl_char]))

cross_entropy = (-1)*sum(logpx)/len(logpx)
perplexity = 2**cross_entropy
print('size of vocablury:', len(vocab))
print('out of vocablury tokens in training set:{:.2f}%'.format(relative_fre
print('out of vocablury tokens in validation set:{:.2f}%'.format(oovv/total
print('cross_entropy:{:.4f}\nperplexity:{:.4f}'.format(cross_entropy, perpl
size of vocablury: 509
out of vocablury tokens in training set:0.05%
out of vocablury tokens in validation set:0.06%
cross_entropy:5.0599
perplexity:33.3577
```

Converting text data into tensor sequenes

In [5]:

```python
tweet_size = 140
vocab_list = list(vocab.keys())
def index_of(tok):
    if tok in vocab_list:
        return vocab_list.index(tok)
    else:
        return 0

train_tweets = df_train.document.str.slice(0, tweet_size)
val_tweets = df_val.document.str.slice(0, tweet_size)
test_tweets = df_test.document.str.slice(0, tweet_size)

train_vectorized_seqs = [[index_of(tok) for tok in seq]for seq in train_twe
val_vectorized_seqs = [[index_of(tok) for tok in seq]for seq in val_tweets]
test_vectorized_seqs = [[index_of(tok) for tok in seq]for seq in test_tweet

train_seq_lengths = torch.LongTensor(list(map(len, train_vectorized_seqs)))
val_seq_lengths = torch.LongTensor(list(map(len, val_vectorized_seqs)))
test_seq_lengths = torch.LongTensor(list(map(len, test_vectorized_seqs)))

x_train = Variable(torch.zeros((len(train_vectorized_seqs), tweet_size))).l
for idx, (seq, seqlen) in enumerate(zip(train_vectorized_seqs, train_seq_le
    x_train[idx, :seqlen] = torch.LongTensor(seq)

x_val = Variable(torch.zeros((len(val_vectorized_seqs), tweet_size))).long(
for idx, (seq, seqlen) in enumerate(zip(val_vectorized_seqs, val_seq_length
    x_val[idx, :seqlen] = torch.LongTensor(seq)

# converting all the vectorized seq into tensor seq with length 140
x_test = Variable(torch.zeros((len(test_vectorized_seqs), tweet_size))).lon
for idx, (seq, seqlen) in enumerate(zip(test_vectorized_seqs, test_seq_leng
    x_test[idx, :seqlen] = torch.LongTensor(seq)

train_y = torch.tensor(df_train.language.values)
val_y = torch.tensor(df_val.language.values)
test_y = torch.tensor(df_test.language.values)
y_train, y_val, y_test = Variable(train_y), Variable(val_y), Variable(test_
```

Define CNN model

In [10]:

```
class Net(Module):
    def __init__(self):
        super(Net, self).__init__()
        self.embedding = nn.Embedding(len(vocab), 14)
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 4, kernel_size=5, stride=2, padding=2),
            nn.BatchNorm2d(4),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(4, 8, kernel_size=5, stride=2, padding=2),
            nn.ReLU(),
            nn.BatchNorm2d(8),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.drop_out = nn.Dropout()
        self.fc1 = nn.Linear(72, 9)

    def forward(self, x):
        out = self.embedding(x)
        out = out.reshape(out.shape[0],1,out.shape[1],out.shape[2])
        out = self.layer1(out)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.drop_out(out)
        out = self.fc1(out)
        return out

# defining the model
model = Net()
# defining the optimizer
optimizer = Adam(model.parameters(), lr=0.07)
# defining the loss function
criterion = CrossEntropyLoss()

print(model)


def train(x, y):
    model.train()
    tr_loss = 0

    # clearing the Gradients of the model parameters
    optimizer.zero_grad()

    # prediction for training set
    output = model(x)

    # computing the training and validation loss
    loss = criterion(output, y)

    # computing the updated weights of all the model parameters
    loss.backward(retain_graph=True)
    optimizer.step()
    loss.detach_()
    tr_loss = loss.item()
    # Track the accuracy
    total = y.shape[0]
    _, predicted = torch.max(output.data, 1)
    correct = (predicted == y).sum().item()

    print('Loss: {:.4f}, Accuracy: {:.2f}%'.format(tr_loss, (correct / tota

def test(x, y):
    tr_loss = 0

    # prediction for training set
    output = model(x)

    # computing the training and validation loss
    loss = criterion(output, y)
    tr_loss = loss.item()

    # Track the accuracy
    total = y.shape[0]
    _, predicted = torch.max(output.data, 1)
    correct = (predicted == y).sum().item()

    print('Loss: {:.4f}, Accuracy: {:.2f}%\n'.format(tr_loss, (correct / to
```

```
Net(
  (embedding): Embedding(509, 14)
  (layer1): Sequential(
    (0): Conv2d(1, 4, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (1): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_runni
ng_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(4, 8, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (1): ReLU()
    (2): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_runni
ng_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
  )
  (drop_out): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=72, out_features=9, bias=True)
)
```

Train the model

In [11]:

```
def batch(iterable, n=10000):
    l = len(iterable)
    for ndx in range(0, l, n):
        yield iterable[ndx:min(ndx + n, l)]

i = 0
epoch = 25
# training the model
for e in range(0,epoch):
    for x_t, y_t in zip(batch(x_train), batch(y_train)):
        print('Epoch: ', e, ' Batch: ', i)
        train(x_t, y_t)
        i=i+1
```

```
Loss: 0.6809, Accuracy: 77.77%
Epoch:  19  Batch:  158
Loss: 0.7090, Accuracy: 77.08%
Epoch:  19  Batch:  159
Loss: 0.6748, Accuracy: 77.89%
Epoch:  20  Batch:  160
Loss: 0.7077, Accuracy: 76.86%
Epoch:  20  Batch:  161
Loss: 0.6919, Accuracy: 77.83%
Epoch:  20  Batch:  162
Loss: 0.6902, Accuracy: 77.47%
Epoch:  20  Batch:  163
Loss: 0.6773, Accuracy: 77.65%
Epoch:  20  Batch:  164
Loss: 0.6913, Accuracy: 77.76%
Epoch:  20  Batch:  165
Loss: 0.6531, Accuracy: 79.02%
Epoch:  20  Batch:  166
Loss: 0.6916, Accuracy: 78.21%
Epoch:  20  Batch:  167
```

Validate the model

In [12]:

```
test(x_val, y_val)
```

```
Loss: 0.6670, Accuracy: 78.04%
```

Test the model

In [21]:

```
test(x_test, y_test)
```

```
Loss: 0.6876, Accuracy: 77.82%
```

In [ ]: