Classify Text using LSTM in PyTorch

1. load data

```
with open('text_file', 'r') as f:
    reviews = f.read()
with open('labels', 'r') as f:
    labels = f.read()
```

2. preprocess

```
reviews = reviews.lower()
all_text = ''.join([c for c in reviews if c not in punctuation])
reviews_split = all_text.split('\n')
all_text = ' '.join(reviews_split)
words = all_text.split()
```

from string import punctuation

3. encode the tokens

```
c = Counter(words)
vocab = sorted(c, key=c.get, reverse=True)
vocab2int = {word : i for i, word in enumerate(vocab, 1)}
rev_ints = []
for rev in reviews_split:
    rev_ints.append([vocab2int[w] for w in rev.split()])
```

start from one

4. encode the labels

```
enc_labels = np.array([1 if label == 'positive' else 0
                       for label in labels.split('\n')])
```

5. Remove bad samples

```
non_zero_idx = [i for i, rev in enumerate(reviews_ints)
                if len(rev) != 0]
```

Clean data

```
reviews_ints = [reviews_ints[i] for i in non_zero_idx]
encoded_labels = np.array([encoded_labels[i] for i in non_zero_idx])
```

6. Pad / truncate sequences

```
def pad(rev_ints, max_len):
    features = np.zeros((len(rev_ints), max_len), dtype=int)
    for i, row in enumerate(rev_ints):
        features[i, -len(row):] = np.array(row)[:max_len]
    return features
```

zero is a pad token in the vocab

start from the end

make it work for long sequences

7. Split data

```
split_frac = 0.8
num_train = int(split_frac * len(features))
num_test = (len(features) - num_train) // 2
idxs = list(range(len(features)))
train_idx = idxs[:num_train]
test_idx = idxs[num_train, num_train + num_test]
valid_idx = idxs[num_train + num_test:]

train_x, train_y = features[train_idx,:], [encoded_ls[i] for i in train_idx]
test_x, test_y = features[test_idx,:], [encoded_ls[i] for i in test_idx]
valid_x, valid_y = features[valid_idx,:], [encoded_ls[i] for i in valid_idx]
```

idx for each

8. loading data

useful methods from torch.utils.data

```
data = TensorDataset(torch.from_numpy(x),
                     torch.from_numpy(y))

loader = DataLoader(data, batch_size=bsize,
                    shuffle=True)
```

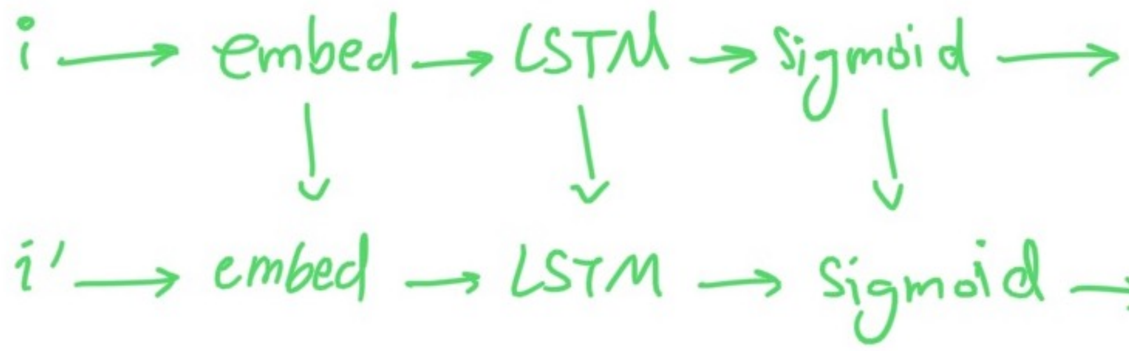
```
train_data = TensorDataset(torch.from_numpy(train_x),
                           torch.from_numpy(train_y))

train_loader = DataLoader(train_data, batch_size = bsize,
                          shuffle=True)
```

→ repeat for test_x and valid_x.

→ we don't want the model to learn anything from the order of the data.

9. Define the Model




```
class SentimentRNN(nn.Module):
    def __init__(self, vocab_size, output_size, embed_dim,
                 hidden_dim, n_layers, drop_prob=0.5):
        super(SentimentRNN, self).__init__()
```

```
self.output_size = output_size
self.n_layers =  n_layers
self.hidden_dim = hidden_dim
```

* `self.embedding = nn.Embedding(vocab_size, embedding_dim)`
* `self.lstm = nn.LSTM(embedding_dim, hidden_dim, n_layers, dropout=drop_prob, batch_first=True)`

```
self.dropout = nn.Dropout(drop_prob)
```

* `self.fc = nn.Linear(hidden_dim, output_size)`

```
self.sigmoid = nn.Sigmoid
```

```
def forward(self, x, hidden):
    bsize = x.size(0)

    embeds = self.embedding(x)
    lstm_out, hidden = self.lstm(embeds, hidden)

    lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)

    out = self.fc(self.dropout(lstm_out))
    out = self.sigmoid(out)
    out = out.view(bsize, -1)
    out = out[:, -1]

    return out, hidden
```

stack up lstm output

stack up output

get the last output only

Example model instantiation:

len(vocab2int) + 1

net = SentimentRNN(vocab_size, output_size, embedding_dim, hidden_dim, n_layers)

10. Train (only new stuff)

Criterion = nn.BCELoss() → designed to work with sigmoid output (binary cross entropy loss)

at the beginning of epoch loop: h = net.init_hidden(bsize)

before optimizer.step(): nn.utils.clip_grad_norm_(net.parameters(), clip)

clip ← e.g., 5 ← prevent exploding (RNN)

before zero_grad(): h = tuple([each.data for each in h])

↳ this avoid backprop to all the history

after zero_grad(): out, h = net(inputs, h)

we get the data like this: "for input, label in train_loader"

11. Predict

```
def tokenize(rev):
    rev = rew.lower()
    text = ''.join([c for c in rev if c not in punctuation])
    words = text.split()
    idx = []
    idx.append([vocab2int[word] for word in words])
    return idx

def predict(model, rev, max_len=200):
    model.eval()
    idx = tokenize(rev)
    features = pad(idx, max_len)
    feature_tensor = torch.from_numpy(features)
    bsize = feature_tensor.size(0)
    h = model.init_hidden(bsize) → hidden state
    if on_gpu: feature_tensor = feature_tensor.cuda()
    out, h = model(feature_tensor, h)
    pred = torch.round(out.squeeze())
    return out.item() → 0,1
```