Classify Text using LSTM in PyTorch

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

2. preprocess
   
   ```python
   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()
   ```
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 ()])

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]

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

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_1s[i] for i in train_idx]
test_x, test_y = features[test_idx, :], [encoded_1s[i] for i in test_idx]
valid_x, valid_y = features[valid_idx, :], [encoded_1s[i] for i in valid_idx]
8. Loading data

Useful methods from torchvision.data

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

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

```python
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

```
i → embed → LSTM → Sigmoid →
```

```
i' → embed → LSTM → Sigmoid → label
```

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
```python
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]  # get the last output only

    return out, hidden
```
Example model instantiation:
```python
net = SentimentRNN(vocab_size, output_size, embedding_dim, hidden_dim, n_layers)
```

10. Train (only new stuff)
```python
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())`, prevent exploding $\approx$ e.g., 5 (clip)
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”
def tokenize(rev):
    rev = rev.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(0))
    return out.item()  # 0, 1