STAT 453: Deep Learning (Spring 2021)

Instructor: Sebastian Raschka (sraschka@wisc.edu)

Course website: http://pages.stat.wisc.edu/~sraschka/teaching/stat453-ss2021/

GitHub repository: https://github.com/rasbt/stat453-deep-learning-ss21

Same as 1_lstm.ipynb but with packed sequences

Explanation of packing: https://stackoverflow.com/questions/51030782/why-do-we-pack-the-sequences-in-pytorch

```
!pip install torchtext==0.9
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Collecting torchtext==0.9
  Downloading torchtext-0.9.0-cp37-cp37m-manylinux1_x86_64.whl (7.1 MB)
     |████████████████████████████████| 7.1 MB 6.2 MB/s
Collecting torch==1.8.0
  Downloading torch-1.8.0-cp37-cp37m-manylinux1_x86_64.whl (735.5 MB)
     |████████████████████████████████| 735.5 MB 13 kB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dis
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/l
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dis
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/di
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-pac
Installing collected packages: torch, torchtext
  Attempting uninstall: torch
    Found existing installation: torch 1.12.1+cu113
    Uninstalling torch-1.12.1+cu113:
      Successfully uninstalled torch-1.12.1+cu113
  Attempting uninstall: torchtext
    Found existing installation: torchtext 0.13.1
    Uninstalling torchtext-0.13.1:
      Successfully uninstalled torchtext-0.13.1
ERROR: pip's dependency resolver does not currently take into account all the pa
torchvision 0.13.1+cu113 requires torch==1.12.1, but you have torch 1.8.0 which
torchaudio 0.12.1+cu113 requires torch==1.12.1, but you have torch 1.8.0 which i
Successfully installed torch-1.8.0 torchtext-0.9.0
```

```
# %load_ext watermark
# %watermark -a 'Sebastian Raschka' -v -p torch,torchtext
```

```
import torch
import torch.nn.functional as F
import torchtext
import time
import random
import pandas as pd

torch.backends.cudnn.deterministic = True
```

General Settings

```
RANDOM_SEED = 123
torch.manual_seed(RANDOM_SEED)

VOCABULARY_SIZE = 20000
LEARNING_RATE = 0.005
BATCH_SIZE = 128
NUM_EPOCHS = 3
DEVICE = torch.device('cuda:2' if torch.cuda.is_available() else 'cpu')

EMBEDDING_DIM = 128
HIDDEN_DIM = 256
NUM_CLASSES = 4
```

Download Dataset

Check that the dataset looks okay:

```
df = pd.read_csv('uci-news-aggregator.csv')
df = df[["TITLE", "CATEGORY"]]
df.head()
```

	TITLE	CATEGORY
0	Fed official says weak data caused by weather,...	b
1	Fed's Charles Plosser sees high bar for change...	b
2	US open: Stocks fall after Fed official hints ...	b
3	Fed risks falling 'behind the curve', Charles ...	b
4	Fed's Plosser: Nasty Weather Has Curbed Job Gr...	b

```
df.columns = ['TEXT_COLUMN_NAME', 'LABEL_COLUMN_NAME']
df.to_csv('news_data.csv', index=None)

df = pd.read_csv('news_data.csv')
df.head()
```

	TEXT_COLUMN_NAME	LABEL_COLUMN_NAME
0	Fed official says weak data caused by weather,...	b
1	Fed's Charles Plosser sees high bar for change...	b
2	US open: Stocks fall after Fed official hints ...	b
3	Fed risks falling 'behind the curve', Charles ...	b
4	Fed's Plosser: Nasty Weather Has Curbed Job Gr...	b

```
del df
```

Prepare Dataset with Torchtext

```
# !conda install spacy
```

Download English vocabulary via:

- `python -m spacy download en_core_web_sm`

Define the Label and Text field formatters:

```
### Defining the feature processing

TEXT = torchtext.legacy.data.Field(
    tokenize='spacy', # default splits on whitespace
    tokenizer_language='en_core_web_sm',
    include_lengths=True # NEW
)

### Defining the label processing

LABEL = torchtext.legacy.data.LabelField(dtype=torch.long)
```

Process the dataset:

```
fields = [('TEXT_COLUMN_NAME', TEXT), ('LABEL_COLUMN_NAME', LABEL)]

dataset = torchtext.legacy.data.TabularDataset(
    path='news_data.csv', format='csv',
    skip_header=True, fields=fields)
```

Split Dataset into Train/Validation/Test

Split the dataset into training, validation, and test partitions:

```
train_data, test_data = dataset.split(
    split_ratio=[0.8, 0.2],
    random_state=random.seed(RANDOM_SEED))

print(f'Num Train: {len(train_data)}')
print(f'Num Test: {len(test_data)}')
```

```
Num Train: 337935
Num Test: 84484
```

```
train_data, valid_data = train_data.split(
    split_ratio=[0.85, 0.15],
    random_state=random.seed(RANDOM_SEED))

print(f'Num Train: {len(train_data)}')
print(f'Num Validation: {len(valid_data)}')
```

```
Num Train: 287245
Num Validation: 50690
```

```
print(vars(train_data.examples[0]))
```

```
{'TEXT_COLUMN_NAME': ['Oil', 'falls', 'below', '$', '108', 'on', 'excess', 'supp
```

Build Vocabulary

Build the vocabulary based on the top "VOCABULARY_SIZE" words:

```
TEXT.build_vocab(train_data, max_size=VOCABULARY_SIZE)
LABEL.build_vocab(train_data)
```

```
print(f'Vocabulary size: {len(TEXT.vocab)}')
print(f'Number of classes: {len(LABEL.vocab)}')
```

```
Vocabulary size: 20002
Number of classes: 4
```

- 25,002 not 25,000 because of the `<unk>` and `<pad>` tokens
- PyTorch RNNs can deal with arbitrary lengths due to dynamic graphs, but padding is necessary for padding sequences to the same length in a given minibatch so we can store those in an array

Look at most common words:

```
print(TEXT.vocab.freqs.most_common(20))
```

```
[("'", 89722), (',', 58391), ('to', 56935), (':', 56104), ('-', 45972), ("'s", 4
```

Tokens corresponding to the first 10 indices (0, 1, ..., 9):

```
print(TEXT.vocab.itos[:10]) # itos = integer-to-string
```

```
['<unk>', '<pad>', "'", ',', 'to', ':', '-', "'s", 'in', '...']
```

Converting a string to an integer:

```
print(TEXT.vocab.stoi['the']) # stoi = string-to-integer
```

```
13
```

Class labels:

```
print(LABEL.vocab.stoi)
```

```
defaultdict(None, {'e': 0, 'b': 1, 't': 2, 'm': 3})
```

Class label count:

```
LABEL.vocab.freqs
```

```
Counter({'b': 78810, 'e': 103739, 'm': 31120, 't': 73576})
```

Define Data Loaders

```
train_loader, valid_loader, test_loader = \
    torchtext.legacy.data.BucketIterator.splits(
        (train_data, valid_data, test_data),
        batch_size=BATCH_SIZE,
        sort_within_batch=True, # NEW. necessary for packed_padded_sequence
             sort_key=lambda x: len(x.TEXT_COLUMN_NAME),
        device=DEVICE
)
```

Testing the iterators (note that the number of rows depends on the longest document in the respective batch):

```
print('Train')
for batch in train_loader:
    print(f'Text matrix size: {batch.TEXT_COLUMN_NAME[0].size()}')
    print(f'Target vector size: {batch.LABEL_COLUMN_NAME.size()}')
    break

print('\nValid:')
for batch in valid_loader:
    print(f'Text matrix size: {batch.TEXT_COLUMN_NAME[0].size()}')
    print(f'Target vector size: {batch.LABEL_COLUMN_NAME.size()}')
    break

print('\nTest:')
for batch in test_loader:
    print(f'Text matrix size: {batch.TEXT_COLUMN_NAME[0].size()}')
    print(f'Target vector size: {batch.LABEL_COLUMN_NAME.size()}')
    break
```

```
Train
Text matrix size: torch.Size([6, 128])
Target vector size: torch.Size([128])

Valid:
Text matrix size: torch.Size([2, 128])
Target vector size: torch.Size([128])

Test:
Text matrix size: torch.Size([2, 128])
Target vector size: torch.Size([128])
```

Model with Fully Connected Layer

```
class RNN(torch.nn.Module):

    def __init__(self, input_dim, embedding_dim, hidden_dim, output_dim):
        super().__init__()

        self.embedding = torch.nn.Embedding(input_dim, embedding_dim)
        #self.rnn = torch.nn.RNN(embedding_dim,
        #                        hidden_dim,
        #                        nonlinearity='relu')
        self.rnn = torch.nn.LSTM(embedding_dim,
                                 hidden_dim)

        self.fc = torch.nn.Linear(hidden_dim, output_dim)


    def forward(self, text, text_length):
        # text dim: [sentence length, batch size]

        embedded = self.embedding(text)
        # ebedded dim: [sentence length, batch size, embedding dim]

        ## NEW
        packed = torch.nn.utils.rnn.pack_padded_sequence(embedded, text_length.to('c

        packed_output, (hidden, cell) = self.rnn(packed)
        # output dim: [sentence length, batch size, hidden dim]
        # hidden dim: [1, batch size, hidden dim]

        hidden.squeeze_(0)
        # hidden dim: [batch size, hidden dim]

        output = self.fc(hidden)
        return output


torch.manual_seed(RANDOM_SEED)
model = RNN(input_dim=len(TEXT.vocab),
            embedding_dim=EMBEDDING_DIM,
            hidden_dim=HIDDEN_DIM,
            output_dim=NUM_CLASSES # could use 1 for binary classification
)

model = model.to(DEVICE)
optimizer = torch.optim.Adam(model.parameters(), lr=0.005)
```

Training

```
def compute_accuracy(model, data_loader, device):

    with torch.no_grad():

        correct_pred, num_examples = 0, 0

        for batch_idx, batch_data in enumerate(data_loader):

            # NEW
            features, text_length = batch_data.TEXT_COLUMN_NAME
            targets = batch_data.LABEL_COLUMN_NAME.to(DEVICE)

            logits = model(features, text_length)
            _, predicted_labels = torch.max(logits, 1)

            num_examples += targets.size(0)

            correct_pred += (predicted_labels == targets).sum()
    return correct_pred.float()/num_examples * 100


start_time = time.time()

for epoch in range(NUM_EPOCHS):
    model.train()
    for batch_idx, batch_data in enumerate(train_loader):

        # NEW
        features, text_length = batch_data.TEXT_COLUMN_NAME
        labels = batch_data.LABEL_COLUMN_NAME.to(DEVICE)

        ### FORWARD AND BACK PROP
        logits = model(features, text_length)
        loss = F.cross_entropy(logits, labels)
        optimizer.zero_grad()

        loss.backward()

        ### UPDATE MODEL PARAMETERS
        optimizer.step()

        ### LOGGING
        if not batch_idx % 50:
            print (f'Epoch: {epoch+1:03d}/{NUM_EPOCHS:03d} | '
                   f'Batch {batch_idx:03d}/{len(train_loader):03d} | '
                   f'Loss: {loss:.4f}')

    with torch.set_grad_enabled(False):
        print(f'training accuracy: '
              f'{compute_accuracy(model, train_loader, DEVICE):.2f}%'
              f'\nvalid accuracy: '
              f'{compute_accuracy(model, valid_loader, DEVICE):.2f}%')
```

```
    print(f'Time elapsed: {(time.time() - start_time)/60:.2f} min')

print(f'Total Training Time: {(time.time() - start_time)/60:.2f} min')
print(f'Test accuracy: {compute_accuracy(model, test_loader, DEVICE):.2f}%')
```

```
Epoch: 003/003 | Batch 2100/2245 | Loss: 0.1487
Epoch: 003/003 | Batch 2150/2245 | Loss: 0.1022
Epoch: 003/003 | Batch 2200/2245 | Loss: 0.0966
training accuracy: 97.47%
valid accuracy: 93.92%
Time elapsed: 31.36 min
Total Training Time: 31.36 min
Test accuracy: 93.89%
```

```
print(LABEL.vocab.stoi)
```

```
defaultdict(None, {'e': 0, 'b': 1, 't': 2, 'm': 3})
```

```
import spacy


nlp = spacy.blank("en")

def predict(model, sentence):

    model.eval()

    with torch.no_grad():
        tokenized = [tok.text for tok in nlp.tokenizer(sentence)]
        indexed = [TEXT.vocab.stoi[t] for t in tokenized]
        length = [len(indexed)]
        tensor = torch.LongTensor(indexed).to(DEVICE)
        tensor = tensor.unsqueeze(1)
        length_tensor = torch.LongTensor(length)
        predict_probas = torch.nn.functional.softmax(model(tensor, length_tensor), d
        predicted_label_index = torch.argmax(predict_probas)
        predicted_label_proba = torch.max(predict_probas)
        return predicted_label_index.item(), predicted_label_proba.item()


class_mapping = LABEL.vocab.stoi
inverse_class_mapping = {v: k for k, v in class_mapping.items()}
```

```
predicted_label_index, predicted_label_proba = \
    predict(model, "Oil prices have been increasing")
predicted_label = inverse_class_mapping[predicted_label_index]

print(f'Predicted label index: {predicted_label_index}'
      f' | Predicted label: {predicted_label}'
      f' | Probability: {predicted_label_proba} ')
```

```
Predicted label index: 1 | Predicted label: b | Probability: 0.9522053003311157
```

```
predicted_label_index, predicted_label_proba = \
    predict(model, "There new breakthrough in neuroscience will help doctors with he
```

```
predicted_label = inverse_class_mapping[predicted_label_index]

print(f'Predicted label index: {predicted_label_index}'
      f' | Predicted label: {predicted_label}'
      f' | Probability: {predicted_label_proba} ')
```

```
Predicted label index: 3 | Predicted label: m | Probability: 0.9992621541023254
```

Model without Fully connected layer

```
class RNN2(torch.nn.Module):

    def __init__(self, input_dim, embedding_dim, hidden_dim, output_dim):
        super().__init__()

        self.embedding = torch.nn.Embedding(input_dim, embedding_dim)
        #self.rnn = torch.nn.RNN(embedding_dim,
        #                        hidden_dim,
        #                        nonlinearity='relu')
        self.rnn = torch.nn.LSTM(embedding_dim,
                                output_dim)

        # self.fc = torch.nn.Linear(hidden_dim, output_dim)


    def forward(self, text, text_length):
        # text dim: [sentence length, batch size]

        embedded = self.embedding(text)
        # ebedded dim: [sentence length, batch size, embedding dim]

        ## NEW
        packed = torch.nn.utils.rnn.pack_padded_sequence(embedded, text_length.to('c

        packed_output, (hidden, cell) = self.rnn(packed)
        # output dim: [sentence length, batch size, hidden dim]
        # hidden dim: [1, batch size, hidden dim]

        hidden.squeeze_(0)
        # hidden dim: [batch size, hidden dim]

        output = hidden
        return output


torch.manual_seed(RANDOM_SEED)
model2 = RNN2(input_dim=len(TEXT.vocab),
            embedding_dim=EMBEDDING_DIM,
            hidden_dim=HIDDEN_DIM,
```

```
            output_dim=NUM_CLASSES # could use 1 for binary classification
)

model2 = model2.to(DEVICE)
optimizer2 = torch.optim.Adam(model2.parameters(), lr=0.005)


start_time = time.time()

for epoch in range(NUM_EPOCHS):
    model2.train()
    for batch_idx, batch_data in enumerate(train_loader):

        # NEW
        features, text_length = batch_data.TEXT_COLUMN_NAME
        labels = batch_data.LABEL_COLUMN_NAME.to(DEVICE)

        ### FORWARD AND BACK PROP
        logits = model2(features, text_length)
        loss2 = F.cross_entropy(logits, labels)
        optimizer2.zero_grad()

        loss2.backward()

        ### UPDATE MODEL PARAMETERS
        optimizer2.step()

        ### LOGGING
        if not batch_idx % 50:
            print (f'Epoch: {epoch+1:03d}/{NUM_EPOCHS:03d} | '
                   f'Batch {batch_idx:03d}/{len(train_loader):03d} | '
                   f'Loss: {loss:.4f}')

    with torch.set_grad_enabled(False):
        print(f'training accuracy: '
              f'{compute_accuracy(model2, train_loader, DEVICE):.2f}%'
              f'\nvalid accuracy: '
              f'{compute_accuracy(model2, valid_loader, DEVICE):.2f}%')

    print(f'Time elapsed: {(time.time() - start_time)/60:.2f} min')

print(f'Total Training Time: {(time.time() - start_time)/60:.2f} min')
print(f'Test accuracy: {compute_accuracy(model2, test_loader, DEVICE):.2f}%')
```

```
Epoch: 001/003 | Batch 000/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 050/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 100/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 150/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 200/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 250/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 300/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 350/2245 | Loss: 0.0026
```

```
Epoch: 001/003 | Batch 400/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 450/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 500/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 550/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 600/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 650/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 700/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 750/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 800/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 850/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 900/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 950/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1000/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1050/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1100/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1150/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1200/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1250/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1300/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1350/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1400/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1450/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1500/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1550/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1600/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1650/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1700/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1750/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1800/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1850/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1900/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 1950/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 2000/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 2050/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 2100/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 2150/2245 | Loss: 0.0026
Epoch: 001/003 | Batch 2200/2245 | Loss: 0.0026
training accuracy: 90.04%
valid accuracy: 88.75%
Time elapsed: 1.82 min
Epoch: 002/003 | Batch 000/2245 | Loss: 0.0026
Epoch: 002/003 | Batch 050/2245 | Loss: 0.0026
Epoch: 002/003 | Batch 100/2245 | Loss: 0.0026
Epoch: 002/003 | Batch 150/2245 | Loss: 0.0026
Epoch: 002/003 | Batch 200/2245 | Loss: 0.0026
Epoch: 002/003 | Batch 250/2245 | Loss: 0.0026
Epoch: 002/003 | Batch 300/2245 | Loss: 0.0026
Epoch: 002/003 | Batch 350/2245 | Loss: 0.0026
Epoch: 002/003 | Batch 400/2245 | Loss: 0.0026
```

2c) We can see that the model without fully connected layer trains very fast but the accuracy is very low.

This could be attributed to the fact that since we dont have a fully connected layer there are far lesser attributes so the training was fast however since there were fewer parameters, the accuracy was also low comparitively.

Start coding or generate with AI.