```python
import numpy as np
import pandas as pd
import torch
import torch.nn.functional as F
import torchtext
import time
import random
import pandas as pd
import spacy
```

Import Required Libraries & Data Loading

In [2]:
```python
#importing the training data
df=pd.read_csv('IMDB Dataset.csv')
print(df.shape)
df.head(10)
```

```
(50000, 2)
```

Out[2]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative
9	If you like original gut wrenching laughter yo...	positive

Data Preparation

In [3]:
```python
"""
sentiment : 0 = negative, 1 = positive
use the following to get the sentiment of a sentence :
sentiment = 0 if sentiment is negative else 1


use np.where to get the sentiment of a sentence :
"""
df['sentiment'] = np.where(df['sentiment'] == 'positive', 1, 0)
```

In [4]:
```python
df.head()
```

Out[4]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	1
1	A wonderful little production. The...	1
2	I thought this was a wonderful way to spend ti...	1
3	Basically there's a family where a little boy ...	0
4	Petter Mattei's "Love in the Time of Money" is...	1

In [5]:
```python
df.columns = ['TEXT_COLUMN_NAME', 'LABEL_COLUMN_NAME']
```

In [6]:
```python
"""
Load the spacy model and load the English language model from https://spacy.io/usage/models
"""
nlp = spacy.load('en_core_web_sm')
```

In [7]:
```python
# general Settings

RANDOM_SEED = 123
torch.manual_seed(RANDOM_SEED)

VOCABULARY_SIZE = 20000
LEARNING_RATE = 0.002 ### ADD YOUR LEARNING RATE HERE ###
BATCH_SIZE = 50 ### ADD YOUR BATCH SIZE HERE ###
NUM_EPOCHS = 25 ### ADD YOUR NUMBER OF EPOCHS HERE ###
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

EMBEDDING_DIM = 100 ### ADD YOUR EMBEDDING DIMENSION HERE ###
HIDDEN_DIM = 100 ### ADD YOUR HIDDEN DIMENSION HERE ###
NUM_CLASSES = 2
```

Text & label Preparation

In [8]:
```python
# Define feature processing
"""
Define the fields for the data.
"""
TEXT = torchtext.legacy.data.Field(tokenize = 'spacy', tokenizer_language = 'en_core_web_sm')
```

In [9]:
```python
# Define Label processing
LABEL = torchtext.legacy.data.LabelField(dtype = torch.long)
```

In [10]:
```python
"""
Define the fields for the data.
"""

df.to_csv('moviedata.csv', index = None)
df = pd.read_csv('moviedata.csv')
df.head()
```

Out[10]:

	TEXT_COLUMN_NAME	LABEL_COLUMN_NAME
0	One of the other reviewers has mentioned that ...	1
1	A wonderful little production. The...	1
2	I thought this was a wonderful way to spend ti...	1
3	Basically there's a family where a little boy ...	0
4	Petter Mattei's "Love in the Time of Money" is...	1

In [11]:
```python
# process the dataset

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

dataset = torchtext.legacy.data.TabularDataset(
                    path = "moviedata.csv", ### ADD YOUR DATASET PATH HERE ###
                    format = "CSV", ### ADD YOUR DATASET FORMAT HERE ###
                    skip_header = True, ### ADD YOUR SKIP HEADER HERE ###
                    fields = fields ### ADD YOUR FIELDS HERE ###
)
```

Data Split

In [12]:
```python
# Split dataset into train and test set

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

print('Length of train data', len(train_data))
print('Length of test data', len(test_data))
```

```
Length of train data 40000
Length of test data 10000
```

In [13]:
```python
train_data, val_data = train_data.split(split_ratio = [0.85, 0.15], random_state =
random.seed(RANDOM_SEED))

print('Length of train data', len(train_data))
print('Length of valid data', len(val_data))
```

```
Length of train data 34000
Length of valid data 6000
```

Data Observation after Tokenization

In [14]:
```python
# Look at first traning example

print(vars(train_data.examples[2009]))
```

```
{'TEXT_COLUMN_NAME': ['There', 'was', 'a', 'lot', 'of', 'hype', 'of', 'this', 'movie', 'and', 'the', 'commercia
ls', 'made', 'it', 'seem', 'like', 'it', 'would', 'be', 'great', '.', 'Sadly', ',', 'like', 'Bring', 'It', 'O
n', '2', ',', 'Bring', 'It', 'On', '3', 'shamed', 'glory', 'of', 'the', 'original', 'Bring', 'It', 'On', '.',
'There', 'is', 'shameless', 'stereotyping', 'throughout', 'the', 'film', '.', 'The', 'lines', 'given', 'to', 't
he', 'actors', 'were', 'humiliating', 'for', 'all', 'the', 'races', 'involved', 'in', 'the', 'film', '.', 'Th
e', 'performance', 'of', 'Hayden', 'Panattiere', 'was', 'sub', '-', 'par', 'both', 'in', 'terms', 'of', 'actin
g', 'and', 'cheerleading', '.', 'There', 'were', 'several', 'scenes', 'in', 'which', 'I', 'literally', 'cringe
d', 'because', 'I', 'was', 'embarrassed', 'for', 'the', 'cast', 'because', 'the', 'scene', '(', 'lines', ',',
'plot', ',', 'etc', ')', 'were', 'just', 'so', 'stupid', '.', 'My', 'recommendation', 'to', 'the', 'makers', 'o
f', 'any', 'future', 'Bring', 'It', 'On', 'films', 'is', 'that', 'you', 'should', 'hire', 'good', 'cheerleader
s', 'and', 'teach', 'them', 'to', 'act', 'because', 'the', '"', 'acting', '"', 'of', 'the', 'cast', 'was', 'hor
rendous', 'and', 'their', 'lack', 'of', 'cheerleading', 'ability', 'made', 'them', 'completely', 'useless', 't
o', 'the', 'film', '.', 'Only', 'great', 'character', ':', 'Kirresha', '.'], 'LABEL_COLUMN_NAME': '0'}
```

In [15]:
```python
# Build Vocabulary

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

print(f'vocabulary size: {len(TEXT.vocab)}')
print(f'label Size: {len(LABEL.vocab)}')
```

```
vocabulary size: 20002
label Size: 2
```

2 extra value in vocabulary is because added (unknown) and (padding)

In [16]:
```python
# Print the most common words: Use the most_common method of the TEXT vocabulary
most_common_words = TEXT.vocab.freqs.most_common()
print(most_common_words[:20])
```

```
[('the', 390972), (',', 369444), ('.', 318509), ('a', 210502), ('and', 210008), ('of', 194659), ('to', 180163),
('is', 145895), ('in', 118266), ('I', 105681), ('it', 103588), ('that', 93995), ('"', 85535), ("'s", 83149),
('this', 81775), ('-', 71249), ('/><br', 68787), ('was', 67372), ('as', 57734), ('movie', 57572)]
```

In [17]:
```python
# Token corresponding to first 10 Indices

print(TEXT.vocab.itos[:20]) #itos = Integer to string
```

```
['<unk>', '<pad>', 'the', ',', '.', 'a', 'and', 'of', 'to', 'is', 'in', 'I', 'it', 'that', '"', "'s", 'this',
'-', '/><br', 'was']
```

Data Preparation for Batch wise Implimentation

In [18]:
```python
# Define Dataloader

train_loader, valid_loader, test_loader = torchtext.legacy.data.BucketIterator.splits(
        (train_data, val_data, test_data), ### ADD YOUR SPLIT DATA HERE (Make sure you add it in
a tuple) ###
        batch_size = BATCH_SIZE, ### ADD YOUR BATCH SIZE HERE ###
        sort_within_batch = True, ### ADD YOUR SORT WITHIN BATCH HERE ###
        sort_key = lambda x : len(x.TEXT_COLUMN_NAME),
        device = DEVICE
    )
```

In [19]:
```python
# 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.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.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.size()}')
    print(f'Target vector size: {batch.LABEL_COLUMN_NAME.size()}')
    break
```

```
Train
Text matrix size: torch.Size([250, 50])
Target vector size: torch.Size([50])

Valid:
Text matrix size: torch.Size([47, 50])
Target vector size: torch.Size([50])

Test:
Text matrix size: torch.Size([41, 50])
Target vector size: torch.Size([50])
```

Model Building

In [49]:
```python
import torch.nn as nn

class RNN(nn.Module):

    def __init__(self, input_dim, embedding_dim, hidden_dim, output_dim):
        super().__init__()
        ### ADD YOUR CODE HERE ###
        self.hidden_dim = hidden_dim

        self.embed = nn.Embedding(input_dim, embedding_dim)
        self.rnn = nn.LSTM(embedding_dim, hidden_dim, num_layers=1)
        self.fc = nn.Linear(hidden_dim, output_dim)
        ### END YOUR CODE ###

    def forward(self, text):
        ### ADD YOUR CODE HERE ###
        # text dim: [sentence length, batch size]
        # embedded dim: [sentence length, batch size, embedding dim]
        embedded = self.embed(text)
        output, (hidden, cell) = self.rnn(embedded)
        # hidden dim: [1, batch size, hidden dim]
        hidden.squeeze(0)
        # hidden dim: [batch size, hidden dim]
        output = self.fc(hidden)
        ### END YOUR CODE ###
        #reshape output to (batch size, output_dim)
        return output.permute(1, 2, 0).squeeze(2)
```

In [50]:
```python
torch.manual_seed(RANDOM_SEED)
model = RNN(input_dim=len(TEXT.vocab), ### ADD YOUR INPUT DIM HERE. This can be the length of
your vocabulary or the embedding dim ###
            embedding_dim= EMBEDDING_DIM, ### ADD YOUR EMBEDDING DIM HERE ###
            hidden_dim= HIDDEN_DIM, ### ADD YOUR HIDDEN DIM HERE ###
            output_dim= NUM_CLASSES ### ADD NUMBER OF CLASSES HERE ###
)

model = model.to(DEVICE)

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE) ### ADD YOUR OPTIMIZER HERE
###
```

Define Accuracy

In [51]:
```python
def compute_accuracy(model, data_loader, device):

    with torch.no_grad():

        correct_pred, num_examples = 0, 0

        for i, (features, targets) in enumerate(data_loader):

            features = features.to(device)
            targets = targets.float().to(device)

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

            num_examples += targets.size(0)
            correct_pred += (predicted_labels == targets).sum()
    return correct_pred.float()/num_examples * 100
```

Model Run

In [52]:
```python
start_time = time.time()

for epoch in range(NUM_EPOCHS):
    model.train()
    for batch_idx, batch_data in enumerate(train_loader):
        text = batch_data.TEXT_COLUMN_NAME.to(DEVICE)
        labels = batch_data.LABEL_COLUMN_NAME.to(DEVICE)

        ### FORWARD AND BACK PROP
        model.zero_grad()
        output = model(text)
        loss = loss_fn(output, labels)
        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: 001/025 | Batch 000/680 | Loss: 0.7216
Epoch: 001/025 | Batch 050/680 | Loss: 0.7055
Epoch: 001/025 | Batch 100/680 | Loss: 0.6662
Epoch: 001/025 | Batch 150/680 | Loss: 0.7529
Epoch: 001/025 | Batch 200/680 | Loss: 0.6569
Epoch: 001/025 | Batch 250/680 | Loss: 0.5226
Epoch: 001/025 | Batch 300/680 | Loss: 0.5207
Epoch: 001/025 | Batch 350/680 | Loss: 0.6768
Epoch: 001/025 | Batch 400/680 | Loss: 0.5450
Epoch: 001/025 | Batch 450/680 | Loss: 0.4612
Epoch: 001/025 | Batch 500/680 | Loss: 0.5053
Epoch: 001/025 | Batch 550/680 | Loss: 0.5656
Epoch: 001/025 | Batch 600/680 | Loss: 0.4411
Epoch: 001/025 | Batch 650/680 | Loss: 0.5087
training accuracy: 74.19%
valid accuracy: 71.65%
Time elapsed: 0.22 min
Epoch: 002/025 | Batch 000/680 | Loss: 0.4463
Epoch: 002/025 | Batch 050/680 | Loss: 0.4999
Epoch: 002/025 | Batch 100/680 | Loss: 0.4702
Epoch: 002/025 | Batch 150/680 | Loss: 0.3764
Epoch: 002/025 | Batch 200/680 | Loss: 0.3966
Epoch: 002/025 | Batch 250/680 | Loss: 0.5544
Epoch: 002/025 | Batch 300/680 | Loss: 0.3925
Epoch: 002/025 | Batch 350/680 | Loss: 0.2967
Epoch: 002/025 | Batch 400/680 | Loss: 0.2121
Epoch: 002/025 | Batch 450/680 | Loss: 0.3268
Epoch: 002/025 | Batch 500/680 | Loss: 0.2702
Epoch: 002/025 | Batch 550/680 | Loss: 0.4205
Epoch: 002/025 | Batch 600/680 | Loss: 0.3472
Epoch: 002/025 | Batch 650/680 | Loss: 0.2054
training accuracy: 92.59%
valid accuracy: 88.02%
Time elapsed: 0.43 min
Epoch: 003/025 | Batch 000/680 | Loss: 0.3225
Epoch: 003/025 | Batch 050/680 | Loss: 0.1894
Epoch: 003/025 | Batch 100/680 | Loss: 0.3585
Epoch: 003/025 | Batch 150/680 | Loss: 0.2802
Epoch: 003/025 | Batch 200/680 | Loss: 0.1672
Epoch: 003/025 | Batch 250/680 | Loss: 0.1932
Epoch: 003/025 | Batch 300/680 | Loss: 0.3166
Epoch: 003/025 | Batch 350/680 | Loss: 0.2152
Epoch: 003/025 | Batch 400/680 | Loss: 0.3204
Epoch: 003/025 | Batch 450/680 | Loss: 0.3158
Epoch: 003/025 | Batch 500/680 | Loss: 0.0997
Epoch: 003/025 | Batch 550/680 | Loss: 0.2566
Epoch: 003/025 | Batch 600/680 | Loss: 0.1703
Epoch: 003/025 | Batch 650/680 | Loss: 0.1751
training accuracy: 96.03%
valid accuracy: 89.10%
Time elapsed: 0.65 min
Epoch: 004/025 | Batch 000/680 | Loss: 0.3321
Epoch: 004/025 | Batch 050/680 | Loss: 0.1261
Epoch: 004/025 | Batch 100/680 | Loss: 0.2623
Epoch: 004/025 | Batch 150/680 | Loss: 0.1936
Epoch: 004/025 | Batch 200/680 | Loss: 0.1060
Epoch: 004/025 | Batch 250/680 | Loss: 0.1477
Epoch: 004/025 | Batch 300/680 | Loss: 0.0688
Epoch: 004/025 | Batch 350/680 | Loss: 0.1660
Epoch: 004/025 | Batch 400/680 | Loss: 0.0584
Epoch: 004/025 | Batch 450/680 | Loss: 0.0277
Epoch: 004/025 | Batch 500/680 | Loss: 0.2512
Epoch: 004/025 | Batch 550/680 | Loss: 0.0610
Epoch: 004/025 | Batch 600/680 | Loss: 0.0625
Epoch: 004/025 | Batch 650/680 | Loss: 0.1324
training accuracy: 97.79%
valid accuracy: 88.88%
Time elapsed: 0.86 min
Epoch: 005/025 | Batch 000/680 | Loss: 0.1894
Epoch: 005/025 | Batch 050/680 | Loss: 0.1874
Epoch: 005/025 | Batch 100/680 | Loss: 0.0340
Epoch: 005/025 | Batch 150/680 | Loss: 0.0935
Epoch: 005/025 | Batch 200/680 | Loss: 0.0886
Epoch: 005/025 | Batch 250/680 | Loss: 0.0887
Epoch: 005/025 | Batch 300/680 | Loss: 0.0507
Epoch: 005/025 | Batch 350/680 | Loss: 0.0489
Epoch: 005/025 | Batch 400/680 | Loss: 0.0782
Epoch: 005/025 | Batch 450/680 | Loss: 0.0212
Epoch: 005/025 | Batch 500/680 | Loss: 0.1121
Epoch: 005/025 | Batch 550/680 | Loss: 0.0481
Epoch: 005/025 | Batch 600/680 | Loss: 0.0304
Epoch: 005/025 | Batch 650/680 | Loss: 0.1240
training accuracy: 98.74%
valid accuracy: 88.80%
Time elapsed: 1.08 min
Epoch: 006/025 | Batch 000/680 | Loss: 0.0200
Epoch: 006/025 | Batch 050/680 | Loss: 0.0627
Epoch: 006/025 | Batch 100/680 | Loss: 0.0752
Epoch: 006/025 | Batch 150/680 | Loss: 0.1364
Epoch: 006/025 | Batch 200/680 | Loss: 0.0127
Epoch: 006/025 | Batch 250/680 | Loss: 0.1421
Epoch: 006/025 | Batch 300/680 | Loss: 0.1489
Epoch: 006/025 | Batch 350/680 | Loss: 0.0113
Epoch: 006/025 | Batch 400/680 | Loss: 0.0364
Epoch: 006/025 | Batch 450/680 | Loss: 0.0189
Epoch: 006/025 | Batch 500/680 | Loss: 0.2116
Epoch: 006/025 | Batch 550/680 | Loss: 0.0344
Epoch: 006/025 | Batch 600/680 | Loss: 0.0317
Epoch: 006/025 | Batch 650/680 | Loss: 0.0365
training accuracy: 99.25%
valid accuracy: 88.78%
Time elapsed: 1.30 min
Epoch: 007/025 | Batch 000/680 | Loss: 0.0351
Epoch: 007/025 | Batch 050/680 | Loss: 0.0390
Epoch: 007/025 | Batch 100/680 | Loss: 0.0133
Epoch: 007/025 | Batch 150/680 | Loss: 0.0031
Epoch: 007/025 | Batch 200/680 | Loss: 0.0035
Epoch: 007/025 | Batch 250/680 | Loss: 0.1014
Epoch: 007/025 | Batch 300/680 | Loss: 0.0200
Epoch: 007/025 | Batch 350/680 | Loss: 0.0050
Epoch: 007/025 | Batch 400/680 | Loss: 0.0038
Epoch: 007/025 | Batch 450/680 | Loss: 0.0054
Epoch: 007/025 | Batch 500/680 | Loss: 0.0510
Epoch: 007/025 | Batch 550/680 | Loss: 0.0057
Epoch: 007/025 | Batch 600/680 | Loss: 0.0173
Epoch: 007/025 | Batch 650/680 | Loss: 0.0402
training accuracy: 99.57%
valid accuracy: 88.65%
Time elapsed: 1.52 min
Epoch: 008/025 | Batch 000/680 | Loss: 0.0115
Epoch: 008/025 | Batch 050/680 | Loss: 0.0013
Epoch: 008/025 | Batch 100/680 | Loss: 0.0017
Epoch: 008/025 | Batch 150/680 | Loss: 0.0125
Epoch: 008/025 | Batch 200/680 | Loss: 0.0062
Epoch: 008/025 | Batch 250/680 | Loss: 0.0043
Epoch: 008/025 | Batch 300/680 | Loss: 0.0050
Epoch: 008/025 | Batch 350/680 | Loss: 0.0058
Epoch: 008/025 | Batch 400/680 | Loss: 0.0018
Epoch: 008/025 | Batch 450/680 | Loss: 0.0178
Epoch: 008/025 | Batch 500/680 | Loss: 0.0266
Epoch: 008/025 | Batch 550/680 | Loss: 0.0599
Epoch: 008/025 | Batch 600/680 | Loss: 0.0349
Epoch: 008/025 | Batch 650/680 | Loss: 0.0117
training accuracy: 99.59%
valid accuracy: 89.05%
Time elapsed: 1.73 min
Epoch: 009/025 | Batch 000/680 | Loss: 0.0100
Epoch: 009/025 | Batch 050/680 | Loss: 0.0046
Epoch: 009/025 | Batch 100/680 | Loss: 0.0402
Epoch: 009/025 | Batch 150/680 | Loss: 0.0937
Epoch: 009/025 | Batch 200/680 | Loss: 0.0027
Epoch: 009/025 | Batch 250/680 | Loss: 0.0044
Epoch: 009/025 | Batch 300/680 | Loss: 0.0060
Epoch: 009/025 | Batch 350/680 | Loss: 0.0020
Epoch: 009/025 | Batch 400/680 | Loss: 0.0022
Epoch: 009/025 | Batch 450/680 | Loss: 0.0035
Epoch: 009/025 | Batch 500/680 | Loss: 0.0097
Epoch: 009/025 | Batch 550/680 | Loss: 0.0031
Epoch: 009/025 | Batch 600/680 | Loss: 0.0116
Epoch: 009/025 | Batch 650/680 | Loss: 0.1065
training accuracy: 99.70%
valid accuracy: 88.73%
Time elapsed: 1.95 min
Epoch: 010/025 | Batch 000/680 | Loss: 0.0553
Epoch: 010/025 | Batch 050/680 | Loss: 0.0008
Epoch: 010/025 | Batch 100/680 | Loss: 0.0189
Epoch: 010/025 | Batch 150/680 | Loss: 0.0058
Epoch: 010/025 | Batch 200/680 | Loss: 0.0016
Epoch: 010/025 | Batch 250/680 | Loss: 0.0083
Epoch: 010/025 | Batch 300/680 | Loss: 0.0022
Epoch: 010/025 | Batch 350/680 | Loss: 0.0064
Epoch: 010/025 | Batch 400/680 | Loss: 0.0077
Epoch: 010/025 | Batch 450/680 | Loss: 0.0077
Epoch: 010/025 | Batch 500/680 | Loss: 0.0413
Epoch: 010/025 | Batch 550/680 | Loss: 0.0124
Epoch: 010/025 | Batch 600/680 | Loss: 0.0195
Epoch: 010/025 | Batch 650/680 | Loss: 0.0156
training accuracy: 99.62%
valid accuracy: 88.33%
Time elapsed: 2.17 min
Epoch: 011/025 | Batch 000/680 | Loss: 0.0031
Epoch: 011/025 | Batch 050/680 | Loss: 0.0087
Epoch: 011/025 | Batch 100/680 | Loss: 0.0056
Epoch: 011/025 | Batch 150/680 | Loss: 0.0082
Epoch: 011/025 | Batch 200/680 | Loss: 0.1452
Epoch: 011/025 | Batch 250/680 | Loss: 0.0017
Epoch: 011/025 | Batch 300/680 | Loss: 0.0037
Epoch: 011/025 | Batch 350/680 | Loss: 0.0031
Epoch: 011/025 | Batch 400/680 | Loss: 0.0035
Epoch: 011/025 | Batch 450/680 | Loss: 0.0503
Epoch: 011/025 | Batch 500/680 | Loss: 0.0048
Epoch: 011/025 | Batch 550/680 | Loss: 0.0119
Epoch: 011/025 | Batch 600/680 | Loss: 0.0062
Epoch: 011/025 | Batch 650/680 | Loss: 0.0083
training accuracy: 99.73%
valid accuracy: 88.62%
Time elapsed: 2.39 min
Epoch: 012/025 | Batch 000/680 | Loss: 0.0022
Epoch: 012/025 | Batch 050/680 | Loss: 0.0008
Epoch: 012/025 | Batch 100/680 | Loss: 0.0163
Epoch: 012/025 | Batch 150/680 | Loss: 0.0020
Epoch: 012/025 | Batch 200/680 | Loss: 0.0038
Epoch: 012/025 | Batch 250/680 | Loss: 0.0014
Epoch: 012/025 | Batch 300/680 | Loss: 0.0010
Epoch: 012/025 | Batch 350/680 | Loss: 0.0061
Epoch: 012/025 | Batch 400/680 | Loss: 0.0044
Epoch: 012/025 | Batch 450/680 | Loss: 0.0164
Epoch: 012/025 | Batch 500/680 | Loss: 0.0020
Epoch: 012/025 | Batch 550/680 | Loss: 0.0058
Epoch: 012/025 | Batch 600/680 | Loss: 0.0044
Epoch: 012/025 | Batch 650/680 | Loss: 0.0005
training accuracy: 99.84%
valid accuracy: 89.07%
Time elapsed: 2.61 min
Epoch: 013/025 | Batch 000/680 | Loss: 0.0023
Epoch: 013/025 | Batch 050/680 | Loss: 0.0027
Epoch: 013/025 | Batch 100/680 | Loss: 0.0003
Epoch: 013/025 | Batch 150/680 | Loss: 0.0005
Epoch: 013/025 | Batch 200/680 | Loss: 0.0003
Epoch: 013/025 | Batch 250/680 | Loss: 0.0036
Epoch: 013/025 | Batch 300/680 | Loss: 0.0010
Epoch: 013/025 | Batch 350/680 | Loss: 0.0003
Epoch: 013/025 | Batch 400/680 | Loss: 0.0030
Epoch: 013/025 | Batch 450/680 | Loss: 0.0019
Epoch: 013/025 | Batch 500/680 | Loss: 0.0014
Epoch: 013/025 | Batch 550/680 | Loss: 0.1283
Epoch: 013/025 | Batch 600/680 | Loss: 0.0175
Epoch: 013/025 | Batch 650/680 | Loss: 0.0022
training accuracy: 99.50%
valid accuracy: 88.12%
Time elapsed: 2.83 min
Epoch: 014/025 | Batch 000/680 | Loss: 0.0069
Epoch: 014/025 | Batch 050/680 | Loss: 0.0073
Epoch: 014/025 | Batch 100/680 | Loss: 0.0037
Epoch: 014/025 | Batch 150/680 | Loss: 0.0527
Epoch: 014/025 | Batch 200/680 | Loss: 0.0060
Epoch: 014/025 | Batch 250/680 | Loss: 0.0008
Epoch: 014/025 | Batch 300/680 | Loss: 0.0007
Epoch: 014/025 | Batch 350/680 | Loss: 0.0064
Epoch: 014/025 | Batch 400/680 | Loss: 0.0139
Epoch: 014/025 | Batch 450/680 | Loss: 0.0970
Epoch: 014/025 | Batch 500/680 | Loss: 0.0432
Epoch: 014/025 | Batch 550/680 | Loss: 0.1009
Epoch: 014/025 | Batch 600/680 | Loss: 0.0076
Epoch: 014/025 | Batch 650/680 | Loss: 0.0115
training accuracy: 99.83%
valid accuracy: 88.70%
Time elapsed: 3.05 min
Epoch: 015/025 | Batch 000/680 | Loss: 0.0016
Epoch: 015/025 | Batch 050/680 | Loss: 0.0185
Epoch: 015/025 | Batch 100/680 | Loss: 0.0022
Epoch: 015/025 | Batch 150/680 | Loss: 0.0007
Epoch: 015/025 | Batch 200/680 | Loss: 0.0022
Epoch: 015/025 | Batch 250/680 | Loss: 0.0423
Epoch: 015/025 | Batch 300/680 | Loss: 0.0065
Epoch: 015/025 | Batch 350/680 | Loss: 0.0019
Epoch: 015/025 | Batch 400/680 | Loss: 0.0169
Epoch: 015/025 | Batch 450/680 | Loss: 0.0007
Epoch: 015/025 | Batch 500/680 | Loss: 0.0014
Epoch: 015/025 | Batch 550/680 | Loss: 0.0150
Epoch: 015/025 | Batch 600/680 | Loss: 0.0007
Epoch: 015/025 | Batch 650/680 | Loss: 0.0017
training accuracy: 99.64%
valid accuracy: 88.55%
Time elapsed: 3.27 min
Epoch: 016/025 | Batch 000/680 | Loss: 0.0015
Epoch: 016/025 | Batch 050/680 | Loss: 0.0491
Epoch: 016/025 | Batch 100/680 | Loss: 0.0004
Epoch: 016/025 | Batch 150/680 | Loss: 0.0013
Epoch: 016/025 | Batch 200/680 | Loss: 0.0018
Epoch: 016/025 | Batch 250/680 | Loss: 0.0015
Epoch: 016/025 | Batch 300/680 | Loss: 0.0004
Epoch: 016/025 | Batch 350/680 | Loss: 0.0026
Epoch: 016/025 | Batch 400/680 | Loss: 0.0071
Epoch: 016/025 | Batch 450/680 | Loss: 0.0007
Epoch: 016/025 | Batch 500/680 | Loss: 0.0555
Epoch: 016/025 | Batch 550/680 | Loss: 0.0008
Epoch: 016/025 | Batch 600/680 | Loss: 0.0132
Epoch: 016/025 | Batch 650/680 | Loss: 0.0232
training accuracy: 99.90%
valid accuracy: 88.57%
Time elapsed: 3.48 min
Epoch: 017/025 | Batch 000/680 | Loss: 0.0007
Epoch: 017/025 | Batch 050/680 | Loss: 0.0008
Epoch: 017/025 | Batch 100/680 | Loss: 0.0008
Epoch: 017/025 | Batch 150/680 | Loss: 0.0028
Epoch: 017/025 | Batch 200/680 | Loss: 0.0014
Epoch: 017/025 | Batch 250/680 | Loss: 0.0003
Epoch: 017/025 | Batch 300/680 | Loss: 0.0026
Epoch: 017/025 | Batch 350/680 | Loss: 0.0027
Epoch: 017/025 | Batch 400/680 | Loss: 0.0010
Epoch: 017/025 | Batch 450/680 | Loss: 0.0006
Epoch: 017/025 | Batch 500/680 | Loss: 0.0100
Epoch: 017/025 | Batch 550/680 | Loss: 0.0520
Epoch: 017/025 | Batch 600/680 | Loss: 0.0003
Epoch: 017/025 | Batch 650/680 | Loss: 0.0004
training accuracy: 99.92%
valid accuracy: 88.25%
Time elapsed: 3.70 min
Epoch: 018/025 | Batch 000/680 | Loss: 0.0010
Epoch: 018/025 | Batch 050/680 | Loss: 0.0009
Epoch: 018/025 | Batch 100/680 | Loss: 0.0014
Epoch: 018/025 | Batch 150/680 | Loss: 0.0006
Epoch: 018/025 | Batch 200/680 | Loss: 0.0006
Epoch: 018/025 | Batch 250/680 | Loss: 0.0018
Epoch: 018/025 | Batch 300/680 | Loss: 0.0057
Epoch: 018/025 | Batch 350/680 | Loss: 0.0003
Epoch: 018/025 | Batch 400/680 | Loss: 0.0043
Epoch: 018/025 | Batch 450/680 | Loss: 0.0003
Epoch: 018/025 | Batch 500/680 | Loss: 0.0179
Epoch: 018/025 | Batch 550/680 | Loss: 0.0009
Epoch: 018/025 | Batch 600/680 | Loss: 0.0007
Epoch: 018/025 | Batch 650/680 | Loss: 0.0004
training accuracy: 99.87%
valid accuracy: 89.12%
Time elapsed: 3.91 min
Epoch: 019/025 | Batch 000/680 | Loss: 0.0005
Epoch: 019/025 | Batch 050/680 | Loss: 0.0004
Epoch: 019/025 | Batch 100/680 | Loss: 0.0237
Epoch: 019/025 | Batch 150/680 | Loss: 0.0047
Epoch: 019/025 | Batch 200/680 | Loss: 0.0003
Epoch: 019/025 | Batch 250/680 | Loss: 0.0005
Epoch: 019/025 | Batch 300/680 | Loss: 0.0008
Epoch: 019/025 | Batch 350/680 | Loss: 0.0006
Epoch: 019/025 | Batch 400/680 | Loss: 0.0168
Epoch: 019/025 | Batch 450/680 | Loss: 0.0073
Epoch: 019/025 | Batch 500/680 | Loss: 0.0080
Epoch: 019/025 | Batch 550/680 | Loss: 0.0022
Epoch: 019/025 | Batch 600/680 | Loss: 0.0808
Epoch: 019/025 | Batch 650/680 | Loss: 0.0004
training accuracy: 99.82%
valid accuracy: 88.57%
Time elapsed: 4.13 min
Epoch: 020/025 | Batch 000/680 | Loss: 0.0005
Epoch: 020/025 | Batch 050/680 | Loss: 0.0014
Epoch: 020/025 | Batch 100/680 | Loss: 0.0032
Epoch: 020/025 | Batch 150/680 | Loss: 0.0224
Epoch: 020/025 | Batch 200/680 | Loss: 0.0013
Epoch: 020/025 | Batch 250/680 | Loss: 0.0002
Epoch: 020/025 | Batch 300/680 | Loss: 0.0001
Epoch: 020/025 | Batch 350/680 | Loss: 0.0001
Epoch: 020/025 | Batch 400/680 | Loss: 0.0002
Epoch: 020/025 | Batch 450/680 | Loss: 0.1181
Epoch: 020/025 | Batch 500/680 | Loss: 0.0015
Epoch: 020/025 | Batch 550/680 | Loss: 0.0087
Epoch: 020/025 | Batch 600/680 | Loss: 0.0002
Epoch: 020/025 | Batch 650/680 | Loss: 0.0026
training accuracy: 99.86%
valid accuracy: 88.37%
Time elapsed: 4.35 min
Epoch: 021/025 | Batch 000/680 | Loss: 0.0005
Epoch: 021/025 | Batch 050/680 | Loss: 0.0004
Epoch: 021/025 | Batch 100/680 | Loss: 0.0007
Epoch: 021/025 | Batch 150/680 | Loss: 0.0004
Epoch: 021/025 | Batch 200/680 | Loss: 0.0003
Epoch: 021/025 | Batch 250/680 | Loss: 0.0001
Epoch: 021/025 | Batch 300/680 | Loss: 0.0004
Epoch: 021/025 | Batch 350/680 | Loss: 0.0001
Epoch: 021/025 | Batch 400/680 | Loss: 0.0042
Epoch: 021/025 | Batch 450/680 | Loss: 0.0174
Epoch: 021/025 | Batch 500/680 | Loss: 0.0119
Epoch: 021/025 | Batch 550/680 | Loss: 0.0001
Epoch: 021/025 | Batch 600/680 | Loss: 0.0000
Epoch: 021/025 | Batch 650/680 | Loss: 0.0004
training accuracy: 99.92%
valid accuracy: 88.58%
Time elapsed: 4.56 min
Epoch: 022/025 | Batch 000/680 | Loss: 0.0002
Epoch: 022/025 | Batch 050/680 | Loss: 0.0009
Epoch: 022/025 | Batch 100/680 | Loss: 0.0001
Epoch: 022/025 | Batch 150/680 | Loss: 0.0004
Epoch: 022/025 | Batch 200/680 | Loss: 0.0006
Epoch: 022/025 | Batch 250/680 | Loss: 0.0002
Epoch: 022/025 | Batch 300/680 | Loss: 0.0001
Epoch: 022/025 | Batch 350/680 | Loss: 0.0001
Epoch: 022/025 | Batch 400/680 | Loss: 0.0002
Epoch: 022/025 | Batch 450/680 | Loss: 0.2063
Epoch: 022/025 | Batch 500/680 | Loss: 0.1061
Epoch: 022/025 | Batch 550/680 | Loss: 0.0003
Epoch: 022/025 | Batch 600/680 | Loss: 0.0006
Epoch: 022/025 | Batch 650/680 | Loss: 0.0064
training accuracy: 99.96%
valid accuracy: 88.82%
Time elapsed: 4.78 min
Epoch: 023/025 | Batch 000/680 | Loss: 0.0004
Epoch: 023/025 | Batch 050/680 | Loss: 0.0012
Epoch: 023/025 | Batch 100/680 | Loss: 0.0001
Epoch: 023/025 | Batch 150/680 | Loss: 0.0078
Epoch: 023/025 | Batch 200/680 | Loss: 0.0002
Epoch: 023/025 | Batch 250/680 | Loss: 0.0015
Epoch: 023/025 | Batch 300/680 | Loss: 0.0028
Epoch: 023/025 | Batch 350/680 | Loss: 0.0004
Epoch: 023/025 | Batch 400/680 | Loss: 0.0011
Epoch: 023/025 | Batch 450/680 | Loss: 0.0016
Epoch: 023/025 | Batch 500/680 | Loss: 0.0036
Epoch: 023/025 | Batch 550/680 | Loss: 0.0017
Epoch: 023/025 | Batch 600/680 | Loss: 0.0005
Epoch: 023/025 | Batch 650/680 | Loss: 0.0036
training accuracy: 99.34%
valid accuracy: 87.50%
Time elapsed: 5.00 min
Epoch: 024/025 | Batch 000/680 | Loss: 0.0026
Epoch: 024/025 | Batch 050/680 | Loss: 0.0017
Epoch: 024/025 | Batch 100/680 | Loss: 0.0002
Epoch: 024/025 | Batch 150/680 | Loss: 0.0005
Epoch: 024/025 | Batch 200/680 | Loss: 0.0003
Epoch: 024/025 | Batch 250/680 | Loss: 0.0007
Epoch: 024/025 | Batch 300/680 | Loss: 0.0001
Epoch: 024/025 | Batch 350/680 | Loss: 0.0077
Epoch: 024/025 | Batch 400/680 | Loss: 0.0001
Epoch: 024/025 | Batch 450/680 | Loss: 0.0003
Epoch: 024/025 | Batch 500/680 | Loss: 0.0017
Epoch: 024/025 | Batch 550/680 | Loss: 0.0006
Epoch: 024/025 | Batch 600/680 | Loss: 0.0021
Epoch: 024/025 | Batch 650/680 | Loss: 0.0188
training accuracy: 99.96%
valid accuracy: 88.72%
Time elapsed: 5.21 min
Epoch: 025/025 | Batch 000/680 | Loss: 0.0095
Epoch: 025/025 | Batch 050/680 | Loss: 0.0001
Epoch: 025/025 | Batch 100/680 | Loss: 0.0001
Epoch: 025/025 | Batch 150/680 | Loss: 0.0003
Epoch: 025/025 | Batch 200/680 | Loss: 0.0028
Epoch: 025/025 | Batch 250/680 | Loss: 0.0007
Epoch: 025/025 | Batch 300/680 | Loss: 0.0002
Epoch: 025/025 | Batch 350/680 | Loss: 0.0003
Epoch: 025/025 | Batch 400/680 | Loss: 0.0001
Epoch: 025/025 | Batch 450/680 | Loss: 0.0002
Epoch: 025/025 | Batch 500/680 | Loss: 0.0001
Epoch: 025/025 | Batch 550/680 | Loss: 0.0002
Epoch: 025/025 | Batch 600/680 | Loss: 0.0005
Epoch: 025/025 | Batch 650/680 | Loss: 0.0001
training accuracy: 99.98%
valid accuracy: 88.50%
Time elapsed: 5.43 min
Total Training Time: 5.43 min
Test accuracy: 88.99%
```

Model Testing

In [53]:

```python
import spacy



nlp = spacy.blank("en")

def predict_sentiment(model, sentence):

    model.eval()
    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)
    prediction = torch.nn.functional.softmax(model(tensor), dim=1)
    return prediction[0][1].item()

print('Probability positive:')
predict_sentiment(model, "This is such an awesome movie, I really love it!")
```

```
Probability positive:
```

Out[53]:

```
0.999997615814209
```

In [54]:

```python
print('Probability positive:')
predict_sentiment(model, "I really hate this movie. It is really bad and sucks!")
```

```
Probability positive:
```

Out[54]:

```
3.919917162420461e-06
```

In [62]:

```python
print("Let's see...")
predict_sentiment(model, "My only problem is that I haven't seen it sooner!")
```

```
Let's see...
```

Out[62]:

```
0.034998420625925064
```

In [ ]: