hwk2

October 15, 2022

1 CS 447 Homework 2 – Text Classification with Neural Networks

In this homework, you will build machine learning models to detect the sentiment of movie reviews using the IMDb movie reviews dataset. Specifically, you will implement classifiers based on Convolutional Neural Networks (CNN's) and Recurrent Neural Networks (RNN's).

In addition to the Pytorch tutorial we have provided on Coursera, we highly recommend that you take a look at the PyTorch tutorials before starting this assignment:

https://pytorch.org/tutorials/beginner/pytorch_with_examples.html

https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

https://github.com/yunjey/pytorch-tutorial

While you work, we suggest that you keep your hardware accelerator set to "CPU" (the default for Colab). However, when you have finished debugging and are ready to train your models, you should select "GPU" as your runtime type. This will speed up the training of your models. You can find this by going to Runtime > Change Runtime Type and select "GPU" from the dropdown menu.

As usual, you should not import any other libraries.

[1]:
```
### DO NOT EDIT ###

import torch

DEVICE = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

if __name__=='__main__':
    print('Using device:', DEVICE)
```

```
Using device: cuda:0
```

2 Step 1: Download the Data

First we will download the dataset using torchtext, which is a package that supports NLP for PyTorch.

Unfortunately, you have to install the torchdata package on the Colab machine in order to access the data. To do this, run the cell below (you may need to click the "Restart Runtime" button when

it finishes). You will have to do this every time you return to work on the homework.

[2]:
```
!pip install torchdata
```

```
Requirement already satisfied: torchdata in
/home/kulbir/anaconda3/envs/nlp_cuda116_python3_9/lib/python3.9/site-packages
(0.4.1)
Requirement already satisfied: requests in
/home/kulbir/anaconda3/envs/nlp_cuda116_python3_9/lib/python3.9/site-packages
(from torchdata) (2.28.1)
Requirement already satisfied: torch==1.12.1 in
/home/kulbir/anaconda3/envs/nlp_cuda116_python3_9/lib/python3.9/site-packages
(from torchdata) (1.12.1)
Requirement already satisfied: portalocker>=2.0.0 in
/home/kulbir/anaconda3/envs/nlp_cuda116_python3_9/lib/python3.9/site-packages
(from torchdata) (2.5.1)
Requirement already satisfied: urllib3>=1.25 in
/home/kulbir/anaconda3/envs/nlp_cuda116_python3_9/lib/python3.9/site-packages
(from torchdata) (1.26.11)
Requirement already satisfied: typing_extensions in
/home/kulbir/anaconda3/envs/nlp_cuda116_python3_9/lib/python3.9/site-packages
(from torch==1.12.1->torchdata) (4.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/home/kulbir/anaconda3/envs/nlp_cuda116_python3_9/lib/python3.9/site-packages
(from requests->torchdata) (2022.9.24)
Requirement already satisfied: idna<4,>=2.5 in
/home/kulbir/anaconda3/envs/nlp_cuda116_python3_9/lib/python3.9/site-packages
(from requests->torchdata) (3.3)
Requirement already satisfied: charset-normalizer<3,>=2 in
/home/kulbir/anaconda3/envs/nlp_cuda116_python3_9/lib/python3.9/site-packages
(from requests->torchdata) (2.0.4)
```

The following cell will get you `train_data` and `test_data`. It also does some basic tokenization.

- To access the list of textual tokens for the ith example, use `train_data[i][1]`
- To access the label for the ith example, use `train_data[i][0]`

[3]:
```
### DO NOT EDIT ###

import torchtext
import random

def preprocess(review):
    '''
    Simple preprocessing function.
    '''
    res = []
    for x in review.split(' '):
        remove_beg=True if x[0] in {'(', '"', "'"} else False
```

```
        remove_end=True if x[-1] in {'.', ',', ';', ':', '?', '!', '"', "'",␣
 ↪')'} else False
        if remove_beg and remove_end: res += [x[0], x[1:-1], x[-1]]
        elif remove_beg: res += [x[0], x[1:]]
        elif remove_end: res += [x[:-1], x[-1]]
        else: res += [x]
    return res

if __name__=='__main__':
    train_data = torchtext.datasets.IMDB(root='.data', split='train')
    train_data = list(train_data)
    train_data = [(x[0], preprocess(x[1])) for x in train_data]
    train_data, test_data = train_data[0:10000] + train_data[12500:
 ↪12500+10000], train_data[10000:12500] + train_data[12500+10000:],

    print('Num. Train Examples:', len(train_data))
    print('Num. Test Examples:', len(test_data))


    # for i in range(len(train_data)):
    for i in range(0,10,1):
        print(f"the label for the *i*th example: {train_data[i][0]} ")
        print(f"list of textual tokens for the *i*th example:␣
 ↪{train_data[i][1]} ")


    # *   To access the list of textual tokens for the *i*th example, use␣
 ↪`train_data[i][1]`
    # *   To access the label for the *i*th example, use `train_data[i][0]`


    print("\nSAMPLE DATA:")
    for x in random.sample(train_data, 5):
        print('Sample text:', x[1])
        print('Sample label:', x[0], '\n')
```

```
Num. Train Examples: 20000
Num. Test Examples: 5000
the label for the *i*th example: pos
list of textual tokens for the *i*th example: ['Zentropa', 'has', 'much', 'in',
'common', 'with', 'The', 'Third', 'Man', ',', 'another', 'noir-like', 'film',
'set', 'among', 'the', 'rubble', 'of', 'postwar', 'Europe', '.', 'Like', 'TTM',
',', 'there', 'is', 'much', 'inventive', 'camera', 'work', '.', 'There', 'is',
'an', 'innocent', 'American', 'who', 'gets', 'emotionally', 'involved', 'with',
'a', 'woman', 'he', "doesn't", 'really', 'understand', ',', 'and', 'whose',
'naivety', 'is', 'all', 'the', 'more', 'striking', 'in', 'contrast', 'with',
'the', 'natives.<br', '/><br', '/>But', "I'd", 'have', 'to', 'say', 'that',
```

'The', 'Third', 'Man', 'has', 'a', 'more', 'well-crafted', 'storyline', '.',
'Zentropa', 'is', 'a', 'bit', 'disjointed', 'in', 'this', 'respect', '.',
'Perhaps', 'this', 'is', 'intentional', ':', 'it', 'is', 'presented', 'as', 'a',
'dream/nightmare', ',', 'and', 'making', 'it', 'too', 'coherent', 'would',
'spoil', 'the', 'effect', '.', '<br', '/><br', '/>This', 'movie', 'is',
'unrelentingly', 'grim--"noir', '"', 'in', 'more', 'than', 'one', 'sense', ';',
'one', 'never', 'sees', 'the', 'sun', 'shine', '.', 'Grim', ',', 'but',
'intriguing', ',', 'and', 'frightening', '.']
the label for the *i*th example: pos
list of textual tokens for the *i*th example: ['Zentropa', 'is', 'the', 'most',
'original', 'movie', "I've", 'seen', 'in', 'years', '.', 'If', 'you', 'like',
'unique', 'thrillers', 'that', 'are', 'influenced', 'by', 'film', 'noir', ',',
'then', 'this', 'is', 'just', 'the', 'right', 'cure', 'for', 'all', 'of',
'those', 'Hollywood', 'summer', 'blockbusters', 'clogging', 'the', 'theaters',
'these', 'days', '.', 'Von', "Trier's", 'follow-ups', 'like', 'Breaking', 'the',
'Waves', 'have', 'gotten', 'more', 'acclaim', ',', 'but', 'this', 'is',
'really', 'his', 'best', 'work', '.', 'It', 'is', 'flashy', 'without', 'being',
'distracting', 'and', 'offers', 'the', 'perfect', 'combination', 'of',
'suspense', 'and', 'dark', 'humor', '.', "It's", 'too', 'bad', 'he', 'decided',
'handheld', 'cameras', 'were', 'the', 'wave', 'of', 'the', 'future', '.',
"It's", 'hard', 'to', 'say', 'who', 'talked', 'him', 'away', 'from', 'the',
'style', 'he', 'exhibits', 'here', ',', 'but', "it's", "everyone's", 'loss',
'that', 'he', 'went', 'into', 'his', 'heavily', 'theoretical', 'dogma',
'direction', 'instead', '.']
the label for the *i*th example: pos
list of textual tokens for the *i*th example: ['Lars', 'Von', 'Trier', 'is',
'never', 'backward', 'in', 'trying', 'out', 'new', 'techniques', '.', 'Some',
'of', 'them', 'are', 'very', 'original', 'while', 'others', 'are', 'best',
'forgotten.<br', '/><br', '/>He', 'depicts', 'postwar', 'Germany', 'as', 'a',
'nightmarish', 'train', 'journey', '.', 'With', 'so', 'many', 'cities', 'lying',
'in', 'ruins', ',', 'Leo', 'Kessler', 'a', 'young', 'American', 'of', 'German',
'descent', 'feels', 'obliged', 'to', 'help', 'in', 'their', 'restoration', '.',
'It', 'is', 'not', 'a', 'simple', 'task', 'as', 'he', 'quickly', 'finds',
'out.<br', '/><br', '/>His', 'uncle', 'finds', 'him', 'a', 'job', 'as', 'a',
'night', 'conductor', 'on', 'the', 'Zentropa', 'Railway', 'Line', '.', 'His',
'job', 'is', 'to', 'attend', 'to', 'the', 'needs', 'of', 'the', 'passengers',
'.', 'When', 'the', 'shoes', 'are', 'polished', 'a', 'chalk', 'mark', 'is',
'made', 'on', 'the', 'soles', '.', 'A', 'terrible', 'argument', 'ensues',
'when', 'a', "passenger's", 'shoes', 'are', 'not', 'chalked', 'despite', 'the',
'fact', 'they', 'have', 'been', 'polished', '.', 'There', 'are', 'many',
'allusions', 'to', 'the', 'German', 'fanaticism', 'of', 'adherence', 'to',
'such', 'stupid', 'details.<br', '/><br', '/>The', 'railway', 'journey', 'is',
'like', 'an', 'allegory', 'representing', "man's", 'procession', 'through',
'life', 'with', 'all', 'its', 'trials', 'and', 'tribulations', '.', 'In', 'one',
'sequence', 'Leo', 'dashes', 'through', 'the', 'back', 'carriages', 'to',
'discover', 'them', 'filled', 'with', 'half-starved', 'bodies', 'appearing',
'to', 'have', 'just', 'escaped', 'from', 'Auschwitz', '', '.', 'These',
'images', ',', 'horrible', 'as', 'they', 'are', ',', 'are', 'fleeting', 'as',

```
'in', 'a', 'dream', ',', 'each', 'with', 'its', 'own', 'terrible', 'impact',
'yet', 'unconnected.<br', '/><br', '/>At', 'a', 'station', 'called', 'Urmitz',
'Leo', 'jumps', 'from', 'the', 'train', 'with', 'a', 'parceled', 'bomb', '.',
'In', 'view', 'of', 'many', 'by-standers', 'he', 'connects', 'the', 'bomb',
'to', 'the', 'underside', 'of', 'a', 'carriage', '.', 'He', 'returns', 'to',
'his', 'cabin', 'and', 'makes', 'a', 'connection', 'to', 'a', 'time', 'clock',
'.', 'Later', 'he', 'jumps', 'from', 'the', 'train', '(', 'at', 'high', 'speed',
')', 'and', 'lies', 'in', 'the', 'cool', 'grass', 'on', 'a', 'river', 'bank',
'.', 'Looking', 'at', 'the', 'stars', 'above', 'he', 'decides', 'that', 'his',
'job', 'is', 'to', 'build', 'and', 'not', 'destroy', '.', 'Subsequently', 'as',
'he', 'sees', 'the', 'train', 'approaching', 'a', 'giant', 'bridge', 'he',
'runs', 'at', 'breakneck', 'speed', 'to', 'board', 'the', 'train', 'and',
'stop', 'the', 'clock', '.', 'If', 'you', 'care', 'to', 'analyse', 'the',
'situation', 'it', 'is', 'a', 'completely', 'impossible', 'task', '.', 'Quite',
'ridiculous', 'in', 'fact', '.', 'It', 'could', 'only', 'happen', 'in', 'a',
'dream.<br', '/><br', "/>It's", 'strange', 'how', 'one', 'remembers', 'little',
'details', 'such', 'as', 'a', 'row', 'of', 'cups', 'hanging', 'on', 'hooks',
'and', 'rattling', 'away', 'with', 'the', 'swaying', 'of', 'the', 'train.<br',
'/><br', '/>Despite', 'the', 'fact', 'that', 'this', 'film', 'is', 'widely',
'acclaimed', ',', 'I', 'prefer', 'Lars', 'Von', "Trier's", 'later', 'films',
'(', 'Breaking', 'the', 'Waves', 'and', 'The', 'Idiots)', '.', 'The', 'bomb',
'scene', 'described', 'above', 'really', 'put', 'me', 'off', '.', 'Perhaps',
"I'm", 'a', 'realist', '.']
the label for the *i*th example: pos
list of textual tokens for the *i*th example: ['*Contains', 'spoilers', 'due',
'to', 'me', 'having', 'to', 'describe', 'some', 'film', 'techniques', ',', 'so',
'read', 'at', 'your', 'own', 'risk!*<br', '/><br', '/>I', 'loved', 'this',
'film', '.', 'The', 'use', 'of', 'tinting', 'in', 'some', 'of', 'the', 'scenes',
'makes', 'it', 'seem', 'like', 'an', 'old', 'photograph', 'come', 'to', 'life',
'.', 'I', 'also', 'enjoyed', 'the', 'projection', 'of', 'people', 'on', 'a',
'back', 'screen', '.', 'For', 'instance', ',', 'in', 'one', 'scene', ',',
'Leopold', 'calls', 'his', 'wife', 'and', 'she', 'is', 'projected', 'behind',
'him', 'rather', 'than', 'in', 'a', 'typical', 'split', 'screen', '.', 'Her',
'face', 'is', 'huge', 'in', 'the', 'back', 'and', "Leo's", 'is', 'in', 'the',
'foreground.<br', '/><br', '/>One', 'of', 'the', 'best', 'uses', 'of', 'this',
'is', 'when', 'the', 'young', 'boys', 'kill', 'the', 'Ravensteins', 'on', 'the',
'train', ',', 'a', 'scene', 'shot', 'in', 'an', 'almost', 'political', 'poster',
'style', ',', 'with', 'facial', 'close', 'ups', '.', 'It', 'reminded', 'me',
'of', 'Battleship', 'Potemkin', ',', 'that', 'intense', 'constant', 'style',
'coupled', 'with', 'the', 'spray', 'of', 'red', 'to', 'convey', 'tons', 'of',
'horror', 'without', 'much', 'gore', '.', 'Same', 'with', 'the', 'scene',
'when', 'Katharina', 'finds', 'her', 'father', 'dead', 'in', 'the',
'bathtub…you', 'can', 'only', 'see', 'the', 'red', 'water', 'on', 'the',
'side', '.', 'It', 'is', 'one', 'of', 'the', 'things', 'I', 'love', 'about',
'Von', 'Trier', ',', 'his', 'understatement', 'of', 'horror', ',', 'which',
'ends', 'up', 'making', 'it', 'all', 'the', 'more', 'creepy.<br', '/><br',
'/>The', 'use', 'of', 'text', 'in', 'the', 'film', 'was', 'unique', ',', 'like',
'when', "Leo's", 'character', 'is', 'pushed', 'by', 'the', 'word', ',', '"',
```

```
'Werewolf.', '"', 'I', 'have', 'never', 'seen', 'anything', 'like', 'that',
'in', 'a', 'film.<br', '/><br', '/>The', 'use', 'of', 'black', 'comedy', 'in',
'this', 'film', 'was', 'well', 'done', '.', 'Ernst-Hugo', 'Järegård', 'is',
'great', 'as', "Leo's", 'uncle', '.', 'It', 'brings', 'up', 'the', 'snickers',
'I', 'got', 'from', 'his', 'role', 'in', 'the', 'Kingdom', '(', 'Riget.', ')',
'This', 'humor', 'makes', 'the', 'plotline', 'of', 'absurd', 'anal',
'retentiveness', 'of', 'train', 'conductors', 'against', 'the', 'terrible',
'backdrop', 'of', 'WW2', 'and', 'all', 'the', 'chaos', ',', 'easier', 'to',
'take', '.', 'It', 'reminds', 'me', 'of', 'Riget', 'in', 'the', 'way', 'the',
'hospital', 'administrator', 'is', 'trying', 'to', 'maintain', 'a', 'normalcy',
'at', 'the', 'end', 'of', 'part', 'one', 'when', 'everything', 'is', 'going',
'crazy', '.', 'It', 'shows', 'that', 'some', 'people', 'are', 'truly',
'oblivious', 'to', 'the', 'awful', 'things', 'happening', 'around', 'them', '.',
'Yet', 'some', 'people', ',', 'like', 'Leo', ',', 'are', 'tuned', 'in', ',',
'but', 'do', 'nothing', 'positive', 'about', 'it.<br', '/><br', '/>The',
'voice', 'over', ',', 'done', 'expertly', 'well', 'by', 'Max', 'von', 'Sydow',
',', 'is', 'amusing', 'too', '.', 'It', 'draws', 'you', 'into', 'the', 'story',
'and', 'makes', 'you', 'jump', 'into', "Leo's", 'head', ',', 'which', 'at',
'times', 'is', 'a', 'scary', 'place', 'to', 'be.<br', '/><br', '/>The', 'movie',
'brings', 'up', 'the', 'point', 'that', 'one', 'is', 'a', 'coward', 'if',
'they', "don't", 'choose', 'a', 'side', '.', 'I', 'see', 'the', 'same', 'idea',
'used', 'in', 'Dancer', 'in', 'the', 'Dark', ',', 'where', "Bjork's",
'character', "doesn't", 'speak', 'up', 'for', 'herself', 'and', 'ends', 'up',
'being', 'her', 'own', 'destruction', '.', 'Actually', ',', 'at', 'one', 'time',
',', 'Von', 'Trier', 'seemed', 'anti-woman', 'to', 'me', ',', 'by', 'making',
'Breaking', 'the', 'Waves', 'and', 'Dancer', ',', 'but', 'now', 'I', 'know',
'his', 'male', 'characters', "don't", 'fare', 'well', 'either', '!', 'I',
'found', 'myself', 'at', 'the', 'same', 'place', 'during', 'the', 'end', 'of',
'Dancer', ',', 'when', 'you', 'seriously', 'want', 'the', 'main', 'character',
'to', 'rethink', 'their', 'actions', ',', 'but', 'of', 'course', ',', 'they',
'never', 'do', '!']
the label for the *i*th example: pos
list of textual tokens for the *i*th example: ['That', 'was', 'the', 'first',
'thing', 'that', 'sprang', 'to', 'mind', 'as', 'I', 'watched', 'the', 'closing',
'credits', 'to', 'Europa', 'make', 'there', 'was', 'across', 'the', 'screen',
',', 'never', 'in', 'my', 'entire', 'life', 'have', 'I', 'seen', 'a', 'film',
'of', 'such', 'technical', 'genius', ',', 'the', 'visuals', 'of', 'Europa',
'are', 'so', 'impressive', 'that', 'any', 'film', 'I', 'watch', 'in', "it's",
'wake', 'will', 'only', 'pale', 'in', 'comparison', ',', 'forget', 'your',
'Michael', 'Bay', ',', 'Ridley', 'Scott', 'slick', 'Hollywood',
'cinematography', ',', 'Europa', 'has', 'more', 'ethereal', 'beauty', 'than',
'anything', 'those', 'two', 'could', 'conjure', 'up', 'in', 'a', 'million',
'years', '.', 'Now', "I'd", 'be', 'the', 'first', 'to', 'hail', 'Lars', 'von',
'Trier', 'a', 'genius', 'just', 'off', 'the', 'back', 'of', 'his', 'films',
'Breaking', 'the', 'Waves', 'and', 'Dancer', 'in', 'the', 'Dark', ',', 'but',
'this', 'is', 'stupid', ',', 'the', 'fact', 'that', 'Europa', 'has', 'gone',
'un-noticed', 'by', 'film', 'experts', 'for', 'so', 'long', 'is', 'a', 'crime',
'against', 'cinema', ',', 'whilst', 'overrated', 'rubbish', 'like', 'Crouching',
```

'Tiger', ',', 'Hidden', 'Dragon', 'and', 'Life', 'is', 'Beautiful', 'clean',
'up', 'at', 'the', 'academy', 'awards', '(', 'but', 'what', 'do', 'the', 'know',
')', 'Europa', 'has', 'been', 'hidden', 'away', ',', 'absent', 'form', 'video',
'stores', 'and', '(', 'until', 'recently', ')', 'any', 'British', 'TV',
'channels', '.', '<br', '/><br', '/>The', 'visuals', 'in', 'Europa', 'are',
'not', 'MTV', 'gloss', ';', "it's", 'not', 'a', 'case', 'of', 'style', 'over',
'substance', ',', 'its', 'more', 'a', 'case', 'of', 'substance', 'dictating',
'style', '.', 'Much', 'like', 'his', 'first', 'film', 'The', 'Element', 'of',
'Crime', ',', 'von', 'Trier', 'uses', 'the', 'perspective', 'of', 'the', 'main',
'character', 'to', 'draw', 'us', 'into', 'his', 'world', ',', 'and', 'much',
'like', 'Element', ',', 'the', 'film', 'begins', 'with', 'the', 'main',
'character', '(', 'or', 'in', 'the', 'case', 'of', 'Europa', ',', 'we', 'the',
'audience', ')', 'being', 'hypnotized', '.', 'As', 'we', 'move', 'down', 'the',
'tracks', ',', 'the', 'voice', 'of', 'the', 'Narrator', '(', 'Max', 'von',
'Sydow', ')', 'counts', 'us', 'down', 'into', 'a', 'deep', 'sleep', ',',
'until', 'we', 'awake', 'in', 'Europa', '.', 'This', 'allows', 'von', 'Trier',
'and', 'his', 'three', 'cinematographers', 'to', 'pay', 'with', 'the',
'conventions', 'of', 'time', 'and', 'imagery', ',', 'there', 'are', 'many',
'scenes', 'in', 'Europa', 'when', 'a', 'character', 'in', 'the', 'background',
',', 'who', 'is', 'in', 'black', 'and', 'white', ',', 'will', 'interact',
'with', 'a', 'person', 'in', 'the', 'foreground', 'who', 'will', 'be', 'colour',
',', 'von', 'Trier', 'is', 'trying', 'to', 'show', 'us', 'how', 'much',
'precedence', 'the', 'coloured', 'item', 'or', 'person', 'has', 'over', 'the',
'plot', ',', 'for', 'instance', ',', "it's", 'no', 'surprise', 'that', 'the',
'first', 'shot', 'of', 'Leopold', 'Kessler', '(', 'Jean-marc', 'Barr', ')',
'is', 'in', 'colour', ',', 'since', 'he', 'is', 'the', 'only', 'character',
"who's", 'actions', 'have', 'superiority', 'over', 'the', 'film', '.', '<br',
'/><br', '/>The', 'performances', 'are', 'good', ',', 'they', 'may', 'not',
'be', 'on', 'par', 'with', 'performances', 'in', 'later', 'von', 'Trier',
'films', ',', 'but', "that's", 'just', 'because', 'the', 'images', 'are',
'sometimes', 'so', 'distracting', 'that', 'you', "don't", 'really', 'pick',
'up', 'on', 'them', 'the', 'first', 'time', 'round', '.', 'But', 'I', 'would',
'like', 'to', 'point', 'out', 'the', 'fantastic', 'performance', 'of', 'Jean-
Marc', 'Barr', 'in', 'the', 'lead', 'role', ',', 'whose', 'blind', 'idealism',
'is', 'slowly', 'warn', 'down', 'by', 'the', 'two', 'opposing', 'sides', ',',
'until', 'he', 'erupts', 'in', 'the', 'films', 'final', 'act', '.', 'Again',
',', 'muck', 'like', 'The', 'Element', 'of', 'Crime', ',', 'the', 'film',
'ends', 'with', 'our', 'hero', 'unable', 'to', 'wake', 'up', 'from', 'his',
'nightmare', 'state', ',', 'left', 'in', 'this', 'terrible', 'place', ',',
'with', 'only', 'the', 'continuing', 'narration', 'of', 'von', 'Sydow', 'to',
'seal', 'his', 'fate', '.', 'Europa', 'is', 'a', 'tremendous', 'film', ',',
'and', 'I', 'cant', 'help', 'thinking', 'what', 'a', 'shame', 'that', 'von',
'Trier', 'has', 'abandoned', 'this', 'way', 'of', 'filming', ',', 'since', 'he',
'was', 'clearly', 'one', 'of', 'the', 'most', 'talented', 'visual', 'directors',
'working', 'at', 'that', 'time', ',', 'Europa', ',', 'much', 'like', 'the',
'rest', 'of', 'his', 'cinematic', 'cannon', 'is', 'filled', 'with', 'a',
'wealth', 'of', 'iconic', 'scenes', '.', 'His', 'dedication', 'to',
'composition', 'and', 'mise-en-scene', 'is', 'unrivalled', ',', 'not', 'to',

```
'mention', 'his', 'use', 'of', 'sound', 'and', 'production', 'design', '.',
'But', 'since', 'his', 'no-frills', 'melodramas', 'turned', 'out', 'to', 'be',
'Breaking', 'the', 'Waves', 'and', 'Dancer', 'in', 'the', 'Dark', 'then', 'who',
'can', 'argue', ',', 'but', 'it', 'does', 'seems', 'like', 'a', 'waste', 'of',
'an', 'imaginative', 'talent', '.', '10/10']
the label for the *i*th example: pos
list of textual tokens for the *i*th example: ['I', 'had', 'started', 'to',
'lose', 'my', 'faith', 'in', 'films', 'of', 'recent', 'being', 'inundated',
'with', 'the', 'typical', 'Genre', 'Hollywood', 'film', '.', 'Story', 'lines',
'fail', ',', 'and', 'camera', 'work', 'is', 'merely', 'copied', 'from', 'the',
'last', 'film', 'of', 'similiar', 'taste', '.', 'But', ',', 'then', 'I', 'saw',
'Zentropa', '(', 'Europa', ')', 'and', 'my', 'faith', 'was', 'renewed', '.',
'Not', 'only', 'is', 'the', 'metaphorical', 'storyline', 'enthralling', 'but',
'the', 'use', 'of', 'color', 'and', 'black', 'and', 'white', 'is', 'visually',
'stimulating', '.', 'The', 'narrator', '(', 'Max', 'Von', 'Sydow', ')', 'takes',
'you', 'through', 'a', 'spellbounding', 'journey', 'every', 'step', 'of', 'the',
'way', 'and', 'engrosses', 'you', 'into', 'Europa', '1945', '.', 'We', 'have',
'all', 'seen', 'death', 'put', 'on', 'screen', 'in', 'a', 'hundred', 'thousand',
'ways', 'but', 'the', 'beauty', 'of', 'this', 'film', 'is', 'how', 'it',
'takes', 'you', 'through', 'every', 'slow-moving', 'moment', 'that', 'leads',
'you', 'to', 'death', '.', 'Unlike', 'many', 'films', 'it', "doesn't", 'cut',
'after', 'one', 'second', 'of', 'showing', '(', 'for', 'example', ')', 'a',
'knife', 'but', 'forces', 'you', 'to', 'watch', 'the', 'devastating', 'yet',
'sensuous', 'beauty', 'of', 'a', "man's", 'final', 'moments', '.', 'I', 'think',
'we', 'can', 'all', 'take', 'something', 'different', 'away', 'from', 'what',
'this', 'movie', 'is', 'trying', 'to', 'say', 'but', 'it', 'is', 'definitely',
'worth', 'taking', 'the', 'time', 'to', 'find', 'out', 'what', 'it', 'all',
'really', 'means', '.', 'I', 'would', 'love', 'to', 'talk', 'more', 'in',
'depth', 'about', 'the', 'film', 'for', 'any', 'one', 'who', 'wishes', 'to',
'send', 'me', 'an', 'email', '.', 'Enjoy', 'it', '!']
the label for the *i*th example: pos
list of textual tokens for the *i*th example: ['Critics', 'need', 'to',
'review', 'what', 'they', 'class', 'as', 'a', 'quality', 'movie', '.', 'I',
'think', 'the', 'critics', 'have', 'seen', 'too', 'many', 'actions', 'films',
'and', 'have', 'succumbed', 'to', 'the', 'Matrix', 'style', 'of', 'films', '.',
'Europa', 'is', 'a', 'breath', 'of', 'fresh', 'air', ',', 'a', 'film', 'with',
'so', 'many', 'layers', 'that', 'one', 'viewing', 'is', 'not', 'enough', 'to',
'understand', 'or', 'appreciate', 'this', 'outstanding', 'film', '.', 'Lars',
'von', 'Trier', 'shows', 'that', 'old', 'styles', 'of', 'filming', 'can',
'produce', 'marvellous', 'cinema', 'and', 'build', 'drama', 'and', 'tension',
'.', 'The', 'back', 'projection', 'effect', 'he', 'uses', 'during', 'the',
'film', 'arouses', 'and', 'enhances', 'the', 'characters', ',', 'and', 'the',
'focus', 'of', 'the', 'conversation', 'they', 'are', 'having', '.', 'Other',
'effects', 'he', 'uses', 'such', 'as', 'the', 'colour', 'and', 'black', 'and',
'white', 'in', 'one', 'scene', 'much', 'like', 'Hitchcock', 'and', 'the',
'girl', 'with', 'the', 'red', 'coat', 'grabs', 'attention', 'and', 'enhances',
'the', 'drama', 'and', 'meaning', 'of', 'the', 'scene', '.', 'The',
'commentary', 'is', 'superb', 'and', 'has', 'a', 'hypnotic', 'effect', ',',
```

'again', 'maintaining', 'the', 'focus', 'on', 'the', 'central', 'characters',
'in', 'the', 'scene', 'and', 'there', 'actions.<br', '/><br', '/>I', 'could',
'talk', 'about', 'the', 'effects', 'more', 'but', 'I', 'think', 'you', 'all',
'would', 'agree', 'they', 'push', 'this', 'film', 'into', 'a', 'category', 'of',
'its', 'own', ',', 'and', 'really', 'heighten', 'the', 'drama', 'of', 'the',
'film', '.', 'A', 'film', 'to', 'buy', 'if', 'you', "don't", 'own', 'already',
'and', 'one', 'to', 'see', 'if', 'you', 'have', 'not.<br', '/><br', '/>10/10',
"Don't", 'miss', 'this', 'artistic', 'noir', 'film', 'from', 'one', 'of', 'the',
'great', 'film', 'directors', '.']
the label for the *i*th example: pos
list of textual tokens for the *i*th example: ['It', 'is', 'not', 'every',
"film's", 'job', 'to', 'stimulate', 'you', 'superficially', '.', 'I', 'will',
'take', 'an', 'ambitious', 'failure', 'over', 'a', 'mass-market', 'hit', 'any',
'day', '.', 'While', 'this', 'really', "can't", 'be', 'described', 'as', 'a',
'failure', ',', 'the', 'sum', 'of', 'its', 'parts', 'remains', 'ambiguous', '.',
'That', 'indecipherable', 'quality', 'tantalizes', 'me', 'into', 'watching',
'it', 'again', 'and', 'again', '.', 'This', 'is', 'a', 'challenging', ',',
'provocative', 'movie', 'that', 'does', 'not', 'wrap', 'things', 'up', 'neatly',
'.', 'The', 'problem', 'with', 'the', 'movie', 'is', 'in', 'its', 'structure',
'.', 'Its', 'inpenetrable', 'plot', 'seems', 'to', 'be', 'winding', 'up', ',',
'just', 'as', 'a', 'second', 'ending', 'is', 'tacked', 'on', '.', 'Though',
'everything', 'is', 'technically', 'dazzling', ',', 'the', 'movie', 'is',
'exactly', 'too', 'long', 'by', 'that', 'unit', '.', 'The', 'long-delayed',
'climax', 'of', "Leo's", 'awakening', 'comes', 'about', '20', 'minutes',
'late.<br', '/><br', '/>Great', 'cinematography', 'often', 'comes', 'at', 'the',
'expense', 'of', 'a', 'decent', 'script', ',', 'but', 'here', 'the',
'innovative', 'camera', 'technique', 'offers', 'a', 'wealth', 'of', 'visual',
'ideas', '.', 'The', 'compositing', 'artifice', 'is', 'provocative', 'and',
'engaging', ';', 'A', 'character', 'is', 'rear-projected', 'but', 'his', 'own',
'hand', 'in', 'the', 'foreground', "isn't", '.', 'The', 'world', 'depicted',
'is', 'deliberate', ',', 'treacherous', 'and', 'absurd', '.', 'Keep', 'your',
'eyes', 'peeled', 'for', 'a', 'memorable', ',', 'technically', 'astonishing',
'assassination', 'that', 'will', 'make', 'your', 'jaw', 'drop.<br', '/><br',
'/>The', 'compositions', 'are', 'stunning', '.', 'Whomever', 'chose', 'to',
'release', 'the', '(', 'out', 'of', 'print', ')', 'videotape', 'in', 'the',
'pan', '&', 'scan', 'format', 'must', 'have', 'never', 'seen', 'it', '.',
'Where', 'is', 'the', 'DVD?<br', '/><br', '/>It', 'is', 'unfathomable', 'how',
'anyone', 'could', 'give', 'this', 'much', 'originality', 'a', 'bad', 'review',
'.', 'You', 'should', 'see', 'it', 'at', 'least', 'once', '.', 'You', 'get',
'the', 'sense', 'that', 'von', 'Trier', 'bit', 'off', 'more', 'than', 'he',
'could', 'chew', ',', 'but', 'this', 'movie', 'ends', 'up', 'being', 'richer',
'for', 'it', '.', 'I', 'suspect', 'he', 'is', 'familiar', 'with', "Hitchcock's",
'Foreign', 'Correspondent', 'in', 'which', 'devious', 'Europeans', 'also',
'manipulate', 'an', 'American', 'dupe', 'and', 'several', 'Welles', 'movies',
'that', 'take', 'delirious', 'joy', 'in', 'technique', 'as', 'much', 'as', 'he',
'does', '.', 'All', 'von', 'Trier', 'movies', 'explore', 'the', 'plight', 'of',
'the', 'naif', 'amidst', 'unforgiving', 'societies', '.', 'After', 'Zentropa',
',', 'von', 'Trier', 'moved', 'away', 'from', 'this', 'type', 'of', 'audacious',

```
'technical', 'experiment', 'towards', 'dreary', ',', 'over-rated', ',', 'un-
nuanced', 'sap', 'like', 'Breaking', 'the', 'Waves', 'and', 'Dancer', 'in',
'the', 'Dark', '.']
the label for the *i*th example: pos
list of textual tokens for the *i*th example: ['The', 'best', 'way', 'for',
'me', 'to', 'describe', 'Europa', ',', 'which', 'is', 'high', 'on', 'the',
'list', 'of', 'my', 'favourite', 'films', ',', 'is', 'the', 'exclamation',
'that', 'came', 'from', 'a', 'companion', 'after', 'the', 'film', 'ended', ':',
'"', 'I', "didn't", 'know', 'films', 'could', 'be', 'made', 'like', 'that"',
'.', 'Entirely', 'original', 'in', "it's", 'visual', 'style', ',', 'it', 'is',
'one', 'of', 'the', 'best', 'examples', 'of', 'what', 'cinema', 'can', 'be',
'.', "It's", 'as', 'far', 'away', 'from', 'the', '"', 'master', 'and',
'coverage', '"', 'style', 'of', 'shooting', 'as', 'one', 'can', 'get', ';',
'perfectly', 'integrating', 'many', 'layers', 'of', 'image', ',', 'sound', ',',
'effects', ',', 'props', ',', 'dialogue', ',', 'voice', 'over', ',',
'performance', ',', 'editing', ',', 'lighting', ',', 'etc..', '.', 'all',
'equal', ',', 'none', 'predominant', '.', 'Despite', "Hollywood's", '"',
'dialogue', '"', 'myopia', ',', 'cinema', 'is', 'not', 'about', 'dialogue', ',',
'nor', 'is', 'it', 'about', 'beautiful', 'lighting', ',', 'action', 'or',
'music', '.', 'It', 'works', 'best', 'when', 'all', 'the', 'elements', 'are',
'on', 'an', 'equal', 'footing', ',', 'where', 'ONLY', 'the', 'BLENDING', 'of',
'those', 'elements', ',', 'in', 'the', 'order', 'or', 'combination', 'in',
'which', 'they', 'are', 'presented', ',', 'will', 'communicate', 'the', 'idea',
'.', 'Reduce', 'or', 'eliminate', 'the', 'contribution', 'of', 'one', 'element',
',', 'and', 'the', 'film', 'has', 'no', 'meaning', '.', '"', 'Europa', '"',
'is', 'what', 'cinema', 'should', 'strive', 'to', 'be', '.']
the label for the *i*th example: pos
list of textual tokens for the *i*th example: ['Released', 'as', 'Zentropa',
'in', 'North', 'America', 'to', 'avoid', 'confusion', 'with', 'Agniezska',
"Holland's", 'own', 'Holocaust', 'film', 'Europa', 'Europa', ',', 'this',
'third', 'theatrical', 'feature', 'by', 'a', 'filmmaker', 'who', 'never',
'ceases', 'to', 'surprise', ',', 'inspire', 'or', 'downright', 'shock', 'is',
'a', 'bizarre', ',', 'nostalgic', ',', 'elaborate', 'film', 'about', 'a',
'naive', 'American', 'in', 'Germany', 'shortly', 'following', 'the', 'end',
'of', 'WWII', '.', 'The', 'American', ',', 'named', 'Leo', ',', "doesn't",
'fully', 'get', 'what', "he's", 'doing', 'there', '.', 'He', 'has', 'come',
'to', 'take', 'part', 'in', 'fixing', 'up', 'the', 'country', 'since', ',',
'in', 'his', 'mind', ',', "it's", 'about', 'time', 'Germany', 'was', 'shown',
'some', 'charity', '.', 'No', 'matter', 'how', 'that', 'sounds', ',', 'he',
'is', 'not', 'a', 'Nazi', 'sympathizer', 'or', 'so', 'much', 'as', 'especially',
'pro-German', ',', 'merely', 'mixed', 'up', '.', 'His', 'uncle', ',', 'who',
'works', 'on', 'the', 'railroad', ',', 'gets', 'Leo', 'a', 'job', 'as', 'a',
'helmsman', 'on', 'a', 'sleeping', 'car', ',', 'and', 'he', 'is',
'increasingly', 'enmeshed', 'in', 'a', 'vortex', 'of', '1945', "Germany's",
'horrors', 'and', 'enigmas.<br', '/><br', '/>This', 'progression', 'starts',
'when', 'Leo', ',', 'played', 'rather', 'memorably', 'by', 'the', 'calm', 'yet',
'restless', 'actor', 'Jean-Marc', 'Barr', ',', 'meets', 'a', 'sultry',
'heiress', 'on', 'the', 'train', 'played', 'by', 'Barbara', 'Sukowa', ',', 'an',
```

'actress', 'with', 'gentility', 'on', 'the', 'surface', 'but', 'internal',
'vigor', '.', 'She', 'seduces', 'him', 'and', 'then', 'takes', 'him', 'home',
'to', 'meet', 'her', 'family', ',', 'which', 'owns', 'the', 'company', 'which',
'manufactures', 'the', 'trains', '.', 'These', 'were', 'the', 'precise',
'trains', 'that', 'took', 'Jews', 'to', 'their', 'deaths', 'during', 'the',
'war', ',', 'but', 'now', 'they', 'run', 'a', 'drab', 'day-to-day', 'timetable',
',', 'and', 'the', "woman's", 'Uncle', 'Kessler', 'postures', 'as', 'another',
'one', 'of', 'those', 'good', 'Germans', 'who', 'were', 'just', 'doing',
'their', 'jobs', '.', 'There', 'is', 'also', 'Udo', 'Kier', ',', 'the',
'tremendous', 'actor', 'who', 'blew', 'me', 'away', 'in', 'Von', "Trier's",
'shocking', 'second', 'film', 'Epidemic', ',', 'though', 'here', 'he', 'is',
'mere', 'scenery.<br', '/><br', '/>Another', 'guest', 'at', 'the', 'house',
'is', 'Eddie', 'Constantine', ',', 'an', 'actor', 'with', 'a', 'quiet',
'strength', ',', 'playing', 'a', 'somber', 'American', 'intelligence', 'man',
'.', 'He', 'can', 'confirm', 'that', 'Uncle', 'Kessler', 'was', 'a', 'war',
'criminal', ',', 'though', 'it', 'is', 'all', 'completely', 'baffling', 'to',
'Leo', '.', 'Americans', 'have', 'been', 'characterized', 'as', 'gullible',
'rubes', 'out', 'of', 'their', 'element', 'for', 'decades', ',', 'but',
'little', 'have', 'they', 'been', 'more', 'blithely', 'unconcerned', 'than',
'Leo', ',', 'who', 'goes', 'back', 'to', 'his', 'job', 'on', 'what',
'gradually', 'looks', 'like', 'his', 'own', 'customized', 'death', 'train.<br',
'/><br', '/>The', 'story', 'is', 'told', 'in', 'a', 'purposely',
'uncoordinated', 'manner', 'by', 'the', "film's", 'Danish', 'director', ',',
'Lars', 'Von', 'Trier', ',', 'whose', 'anchor', 'is', 'in', 'the', "film's",
'breathtaking', 'editing', 'and', 'cinematography', '.', 'He', 'shoots', 'in',
'black', 'and', 'white', 'and', 'color', ',', 'he', 'uses', 'double-exposures',
',', 'optical', 'effects', 'and', 'trick', 'photography', ',', 'having',
'actors', 'interact', 'with', 'rear-projected', 'footage', ',', 'he', 'places',
'his', 'characters', 'inside', 'a', 'richly', 'shaded', 'visceral', 'world',
'so', 'that', 'they', 'sometimes', 'feel', 'like', 'insects', ',', 'caught',
'between', 'glass', 'for', 'our', 'more', 'precise', 'survey.<br', '/><br',
'/>This', 'Grand', 'Jury', 'Prize-winning', 'surrealist', 'work', 'is',
'allegorical', ',', 'but', 'maybe', 'in', 'a', 'distinct', 'tone', 'for',
'every', 'viewer', '.', 'I', 'interpret', 'it', 'as', 'a', 'film', 'about',
'the', 'last', 'legs', 'of', 'Nazism', ',', 'symbolized', 'by', 'the', 'train',
',', 'and', 'the', 'ethical', 'accountability', 'of', 'Americans', 'and',
'others', 'who', 'appeared', 'too', 'late', 'to', 'salvage', 'the', 'martyrs',
'of', 'these', 'trains', 'and', 'the', 'camps', 'where', 'they', 'distributed',
'their', 'condemned', 'shiploads', '.', 'During', 'the', 'time', 'frame', 'of',
'the', 'movie', ',', 'and', 'the', 'Nazi', 'state', ',', 'and', 'such',
'significance', 'to', 'the', 'train', ',', 'are', 'dead', ',', 'but', 'like',
'decapitated', 'chickens', 'they', 'persist', 'in', 'jolting', 'through',
'their', 'reflexes.<br', '/><br', '/>The', 'characters', ',', 'music', ',',
'dialogue', ',', 'and', 'plot', 'are', 'deliberately', 'hammy', 'and', 'almost',
'satirically', 'procured', 'from', 'film', 'noir', 'conventions', '.', 'The',
'most', 'entrancing', 'points', 'in', 'the', 'movie', 'are', 'the', 'entirely',
'cinematographic', 'ones', '.', 'Two', 'trains', 'halting', 'back', 'and',
'forth', ',', 'Barr', 'on', 'one', 'and', 'Sukowa', 'on', 'another', '.', 'An',

```
'underwater', 'shot', 'of', 'proliferating', 'blood', '.', 'An', 'uncommonly',
'expressive', 'sequence', 'on', 'what', 'it', 'must', 'be', 'like', 'to',
'drown', '.', 'And', 'most', 'metaphysically', 'affecting', 'of', 'all', ',',
'an', 'anesthetic', 'shot', 'of', 'train', 'tracks', ',', 'as', 'Max', 'von',
"Sydow's", 'voice', 'allures', 'us', 'to', 'hark', 'back', 'to', 'Europe',
'with', 'him', ',', 'and', 'abandon', 'our', 'personal', 'restraint', '.']

SAMPLE DATA:
Sample text: ['This', 'is', 'one', 'irresistible', 'great', 'cheerful-', 'and',
'technically', 'greatly', 'made', 'movie!<br', '/><br', '/>The', 'movie',
'features', 'some', 'of', 'the', 'greatest', 'looking', 'sets', "you'll",
'ever', 'see', 'in', 'a', "'", "30's", 'movie', ',', 'even', 'though', "it's",
'all', 'too', 'obvious', 'that', 'they', 'are', 'sets', ',', 'rather', 'than',
'real', 'place', 'locations', '.', 'Often', 'if', 'a', 'character', 'would',
'fall', 'or', 'shake', 'a', 'doorpost', 'too', 'aggressive', ',', 'the',
'entire', 'set', 'would', 'obviously', 'move.<br', '/><br', '/>The', 'best',
'moments', 'of', 'the', 'movie', 'were', 'the', 'silent', ',', 'more', 'old',
'fashioned', ',', 'slapstick', 'kind', 'of', 'moments', '.', 'It', 'shows',
'that', 'René', "Clair's", 'true', 'heart', 'was', 'at', 'silent', 'movie-
making', '.', 'The', 'overall', 'humor', 'is', 'really', 'great', 'in', 'this',
'movie', '.', 'Also', 'of', 'course', 'the', 'musical', 'moments', 'were',
'more', 'than', 'great', '.', 'This', 'is', 'a', 'really', 'enjoyable', 'light',
'and', 'simple', 'pleasant', 'early', 'French', 'musical', '.', 'Though', 'the',
'best', 'moments', 'are', 'the', 'silent', 'moments', ',', 'that', 'does',
'not', 'mean', 'that', 'the', 'movie', 'is', 'not', 'filled', 'with', 'some',
'great', 'humorous', 'dialog', ',', 'that', 'gets', 'very', 'well', 'delivered',
'by', 'the', 'main', 'actors', ',', 'who', 'all', 'seemed', 'like', 'stage',
'actors', 'to', 'me', ',', 'which', 'in', 'this', 'case', 'worked', 'extremely',
'well', 'for', 'the', 'movie', 'its', 'overall', 'style', 'and', 'pleasant',
'no-worries', 'atmosphere', '.', 'No', 'wonder', 'this', 'worked', 'out', 'so',
'well', ',', 'since', 'this', 'movie', 'is', 'actually', 'based', 'on', 'stage',
'play', 'by', 'Georges', 'Berr.<br', '/><br', "/>It's", 'a', 'technical',
'really', 'great', 'movie', ',', 'with', 'also', 'some', 'great', 'innovation',
'camera-work', 'in', 'it', 'and', 'some', 'really', 'great', 'editing', ',',
'that', 'create', 'some', 'fast', 'going', 'and', 'pleasant', 'to', 'watch',
'enjoyable', 'sequences', '.', 'There', 'is', 'never', 'a', 'dull', 'moment',
'in', 'this', 'movie!<br', '/><br', '/>René', 'Clair', 'was', 'such', 'a',
'clever', 'director', ',', 'who', 'knew', 'how', 'to', 'build', 'up', 'and',
'plan', 'comical', 'moments', 'within', 'in', 'movies', '.', "It's", 'a',
'very', 'creative', 'made', 'movie', ',', 'that', 'despite', 'its',
'simplicity', 'still', 'at', 'all', 'times', 'feel', 'as', 'a', 'totally',
'original', 'and', 'cleverly', 'constructed', 'movie', ',', 'that', 'never',
'seizes', 'to', 'entertain.<br', '/><br', '/>The', 'last', 'half', 'hour', 'is',
'especially', 'unforgettably', 'fun', ',', 'without', 'spoiling', 'too', 'much',
',', 'and', 'is', 'really', 'among', 'the', 'greatest', ',', 'as', 'well', 'as',
'most', 'creative', 'moments', 'in', 'early', 'comedy', 'film-making.<br',
'/><br', '/>The', 'movie', 'is', 'filled', 'with', 'some', 'really',
'enjoyable', 'characters', ',', 'who', 'are', 'of', 'course', 'all', 'very',
```

```
'stereotypical', 'and', 'silly', 'and', 'were', 'obviously', 'cast', 'because',
'of', 'their', 'looks', '.', 'It', 'all', 'adds', 'to', 'the', 'pleasant',
'light', 'comical', 'atmosphere', 'and', 'cuteness', 'of', 'the', 'movie.<br',
'/><br', '/>One', 'of', 'the', 'most', 'pleasant', 'movies', "you'll", 'ever',
'see!<br', '/><br', '/>8/10']
Sample label: pos

Sample text: ['Having', 'seen', 'and', 'loved', 'Greg', "Lombardo's", 'most',
'recent', 'film', '"', 'Knots', '"', '(', 'he', 'co-wrote', 'and', 'directed',
'that', 'feature', 'as', 'well)', ',', 'I', 'decided', 'to', 'check', 'out',
'his', 'earlier', 'work', ',', 'and', 'this', 'movie', 'was', 'well', 'worth',
'the', 'effort', 'and', 'rental', '.', 'Macbeth', 'in', 'Manhattan', 'is', 'a',
'tongue', 'in', 'cheek', ',', 'excellent', 'take', 'on', 'the', 'Shakespeare',
'favorite', ',', 'updated', 'and', 'moved', 'to', 'NYC', '.', 'I', 'was',
'impressed', 'by', 'the', 'underlying', 'wit', 'and', 'intelligence', 'of',
'the', 'script', 'and', 'was', 'wowed', 'by', 'the', 'way', 'the', 'storyline',
'of', 'the', 'production', 'in', 'the', 'movie', 'mirrors', 'the', 'storyline',
'of', 'the', 'play', 'itself', '-', 'and', 'very', 'cleverly', 'at', 'that',
'.', 'The', 'trials', 'and', 'tribulations', 'of', 'life', 'in', 'Manhattan',
'parallel', 'many', 'a', 'Shakespeare', 'play', ',', 'and', 'Central', 'Park',
'was', 'rarely', 'put', 'to', 'better', 'use', 'than', 'as', 'the', 'woods',
'around', "Macbeth's", 'castle', '.', 'Mr', '.', 'Lombardo', 'obviously', 'has',
'a', 'fond', 'place', 'in', 'his', 'heart', 'for', 'New', 'York', 'and', 'New',
'York', 'stories', '(', 'Knots', 'is', 'a', 'funny', 'and', 'warm', 'sex',
'comedy', 'about', 'six', 'thirty-something', 'New', 'Yorkers', 'set',
'primarily', 'in', 'a', 'charming', 'Brooklyn', 'neighborhood', ',', 'with',
'Manhattan', 'offices', 'and', 'a', 'downtown', 'loft', 'thrown', 'in', 'for',
'good', 'measure', ')', 'and', 'has', 'spent', 'considerable', 'time', 'around',
'the', 'plays', 'of', 'Shakespeare', '.', 'The', 'movie', 'is', 'well-paced',
'and', 'the', 'story', 'reflects', 'a', 'deep', 'understanding', 'of', 'the',
'essential', 'drama', 'at', 'the', 'core', 'of', 'Macbeth', '.', 'It',
'reminded', 'me', 'of', 'Al', "Pacino's", '"', 'Looking', 'for', 'Richard', '"',
'-', 'another', 'wonderful', 'Shakespeare', '"', 'play', 'within', 'a',
'movie.', '"', 'I', 'highly', 'recommend', 'checking', 'out', 'Macbeth', 'in',
'Manhattan', '.']
Sample label: pos

Sample text: ['EUROPA', '(', 'ZENTROPA', ')', 'is', 'a', 'masterpiece', 'that',
'gives', 'the', 'viewer', 'the', 'excitement', 'that', 'must', 'have', 'come',
'with', 'the', 'birth', 'of', 'the', 'narrative', 'film', 'nearly', 'a',
'century', 'ago', '.', 'This', 'film', 'is', 'truly', 'unique', ',', 'and', 'a',
'work', 'of', 'genius', '.', 'The', 'camerawork', 'and', 'the', 'editing',
'are', 'brilliant', ',', 'and', 'combined', 'with', 'the', 'narrative',
'tropes', 'of', 'alienation', 'used', 'in', 'the', 'film', ',', 'creates', 'an',
'eerie', 'and', 'unforgettable', 'cinematic', 'experience.<br', '/><br',
'/>The', 'participation', 'of', 'Barbara', 'Suwkowa', 'and', 'Eddie',
'Constantine', 'in', 'the', 'cast', 'are', 'two', 'guilty', 'pleasures', 'that',
'should', 'be', 'seen', 'and', 'enjoyed', '.', 'Max', 'Von', 'Sydow',
```

```
'provides', 'his', 'great', 'voice', 'as', 'the', 'narrator.<br', '/><br',
'/>A', 'one', 'of', 'a', 'kind', 'movie', '!', 'Four', 'stars', '(', 'highest',
'rating)', '.']
Sample label: pos

Sample text: ["It's", 'a', 'road', 'movie', ',', 'with', 'a', 'killer', 'on-
board', '.', 'Brian', 'Kessler', '(', 'David', 'Duchovny)', ',', 'a',
'sophisticated', ',', 'urbane', 'writer', ',', 'wants', 'to', 'conduct',
'field', 'research', 'on', 'American', 'serial', 'killers', '.', 'But', ',',
'neither', 'he', ',', 'nor', 'his', 'girlfriend', ',', 'Carrie', '(',
'Michelle', 'Forbes)', ',', 'has', 'the', 'money', 'for', 'a', 'cross-country',
'tour', 'of', 'murder', 'sites', ',', 'so', 'they', 'advertise', 'for',
'someone', 'to', 'share', 'travel', 'expenses', '.', 'Who', 'they', 'end', 'up',
'with', 'is', 'a', 'young', 'couple', ',', 'Early', 'Grayce', '(', 'Brad',
'Pitt', ')', 'and', 'his', 'girlfriend', ',', 'Adele', '(', 'Juliette',
'Lewis)', ',', 'two', 'better', 'examples', 'of', '"', 'poor', 'white', 'trash',
'"', 'you', 'will', 'never', 'find', 'in', 'all', 'of', 'cinema.<br', '/><br',
'/>Indeed', ',', 'Early', 'and', 'Adele', 'are', 'what', 'make', 'this', 'film',
'so', 'entertaining', ',', 'as', 'they', 'babble', ',', 'cackle', ',',
'confide', ',', 'muse', ',', 'speculate', ',', 'drool', ',', 'and', 'otherwise',
'behave', 'in', 'ways', 'I', "haven't", 'seen', 'since', 'reruns', 'of', '"',
'The', 'Beverly', 'Hillbillies"', '.', "Early's", 'idea', 'of', 'California',
':', '"', 'People', 'think', 'faster', 'out', 'there', ',', 'on', 'account',
'of', 'all', 'that', 'warm', 'weather', ';', 'cold', 'weather', 'makes',
'people', 'stupid"', '.', "That's", 'enough', 'to', 'convince', 'Adele', ':',
'"', 'I', 'guess', 'that', 'explains', 'why', 'there', 'are', 'so', 'many',
'stupid', 'people', 'around', 'here"', '.', 'To', 'which', 'Early', 'responds',
'proudly', ':', '"', 'It', 'sure', 'does"', '.', 'Early', 'continues', 'to',
'instruct', 'Adele', 'about', 'California', ':', '"', 'You', 'never', 'have',
'to', 'buy', 'no', 'fruit', ',', 'on', 'account', "it's", 'all', 'on', 'the',
'trees', '..', '.', 'and', 'they', "ain't", 'got', 'no', 'speed', 'limits', ',',
'and', 'I', 'hear', 'your', 'first', "month's", 'rent', 'is', 'free', ',',
'state', 'law".<br', '/><br', '/>But', 'poor', 'Early', 'has', 'some', ',',
'well', ',', 'mental', 'problems', ',', 'which', 'become', 'ever', 'more',
'obvious', 'to', 'Brian', 'and', 'Carrie', 'as', 'the', 'four', 'travelers',
'proceed', 'west', 'across', 'the', 'U.S', '.', 'As', 'they', 'enter', 'the',
'desert', 'Southwest', ',', 'with', 'its', 'beautifully', 'stark', 'landscape',
',', '"', 'Kalifornia', '"', 'starts', 'to', 'look', 'more', 'and', 'more',
'like', '"', 'The', 'Hitcher', '"', '(', '1986)', ',', 'and', 'Early', 'starts',
'to', 'act', 'more', 'and', 'more', 'like', 'John', 'Ryder', ',', "everyone's",
'maniacal', 'hitchhiker', ',', 'whose', 'terror', 'seemed', 'so',
'unstoppable.<br', '/><br', '/>In', '"', 'Kalifornia"', ',', 'the', 'acting',
'is', 'uneven', '.', "Duchovny's", 'performance', 'is', 'flat', '.', 'Brad',
'Pitt', 'is', 'surprisingly', 'effective', ',', 'despite', 'his', 'overacting',
'at', 'times', '.', 'Michelle', 'Forbes', 'is', 'great', 'as', 'the', 'avant-
garde', ',', 'photographic', 'artist', '.', 'But', 'my', 'choice', 'for',
'best', 'performance', 'goes', 'to', 'Juliette', 'Lewis', '.', 'With', 'her',
'nasal', 'voice', 'and', 'heavy-duty', 'Southern', 'accent', ',', 'she', 'is',
```

```
'stunning', ',', 'as', 'the', 'naive', ',', 'highly', 'animated', ',', 'child-
like', 'Adele.<br', '/><br', '/>Toward', 'the', 'end', ',', 'the', 'film',
'takes', 'on', 'a', 'Twilight', 'Zone', 'feel', 'to', 'it', ',', 'as', 'our',
'travelers', 'enter', 'a', 'Nevada', 'nuclear', 'test', 'site', 'with', 'a',
'dilapidated', 'old', 'house', 'full', 'of', 'test', 'mannequins', '.', 'The',
'plot', 'dissolves', 'rather', 'messily', 'into', 'unnecessary', 'and',
'preposterous', 'violence', ',', 'an', 'ending', 'that', 'was', 'somewhat',
'disappointing.<br', '/><br', '/>Overall', ',', 'however', ',', '"',
'Kalifornia', '"', 'is', 'an', 'entertaining', 'film', ',', 'thanks', 'to', 'a',
'clever', 'concept', ',', 'great', 'scenery', ',', 'especially', 'in', 'the',
'second', 'half', ',', 'good', 'cinematography', ',', 'great', 'dialogue', ',',
'and', 'that', 'wonderful', 'performance', 'by', 'Juliette', 'Lewis', '.']
Sample label: pos

Sample text: ['I', 'have', 'to', 'admit', ',', 'this', 'movie', 'moved', 'me',
'to', 'the', 'extent', 'that', 'I', 'burst', 'in', 'tears', '.', 'However', ',',
'I', 'always', 'think', 'about', 'things', 'twice', ',', 'and', 'instead', 'of',
'writing', 'a', 'eulogy', 'that', 'would', 'define', 'the', 'film', 'as',
'flawless', 'and', 'impeccable', ',', 'I', 'prefer', 'taking', 'the', 'risk',
'of', 'a', 'closer', 'look.<br', '/><br', '/>First', "what's", 'first', ':',
'The', 'movie', 'has', 'an', 'undeniable', 'impact', 'on', 'the', 'viewer',
'simply', 'because', 'it', 'starts', 'out', 'and', 'continues', 'as', 'a',
'slow-paced', 'movie', 'that', "doesn't", 'try', 'to', 'blow', 'you', 'away',
'with', 'the', 'actual', 'scenes', 'from', '9/11', '.', 'Thumbs', 'up', 'for',
'this', 'stroke', 'of', 'genius', ',', 'because', ',', 'unlike', "Stone's",
'WORLD', 'TRADE', 'CENTER', 'this', 'film', 'fortunately', "doesn't", 'focus',
'on', 'the', 'attack', 'itself', 'but', 'on', 'the', 'fallout', 'which', ',',
'similar', 'to', 'the', 'fallout', 'of', 'a', 'nuclear', 'explosion', ',', 'is',
'hardly', 'visible', 'but', 'nonetheless', 'dangerous', 'and', 'devastating',
'.', 'The', 'psychological', 'impact', ',', 'the', 'sheer', 'devastation',
'that', '9/11', 'caused', 'and', 'the', 'havoc', 'it', 'wreaked', 'on', 'the',
'American', 'people', 'is', 'almost', 'palpable', 'in', 'this', 'movie', '.',
'I', 'think', 'Binder', 'managed', 'an', 'astute', 'observation', 'of', 'the',
'American', 'post', '9/11', 'society', 'and', 'Sandler', 'in', 'my', 'opinion',
'sky', 'rocketed', 'from', 'an', 'average', 'comedy', 'actor', 'to', 'a',
'real', 'talent', 'who', 'delivers', 'a', 'performance', 'worthy', 'of', 'an',
'Oscar.<br', '/><br', '/>However', ':', 'In', 'the', 'film', 'BLOOD', 'DIAMOND',
',', 'the', 'Di', 'Caprio', 'character', 'says', 'and', 'I', 'quote', ':', '"',
'Ah', ',', 'these', 'Americans', '.', 'Always', 'want', 'to', 'take', 'about',
'their', 'feelings"', '.', 'Now', ',', 'I', "don't", 'want', 'to', 'belittle',
'their', 'sufferíngs', ',', 'but', 'I', 'sure', 'would', 'like', 'to', 'make',
'a', 'comparison', '.', 'Ever', 'since', '9/11', 'the', 'entire', 'world', 'is',
'confronted', 'with', 'mementos', ',', 'memorials', 'and', 'commemorations',
'of', '9/11', '.', 'The', 'Hollywood', 'industry', 'and', 'writers', 'such',
'as', 'Safran', 'Foer', 'more', 'than', 'allude', 'to', '9/11', 'in', 'their',
'works', '.', 'Now', ',', 'this', 'huge', 'amount', 'of', 'cultural',
'products', ',', 'dealing', 'with', '9/11', ',', 'turn', 'the', 'death', 'of',
'3000', 'people', 'into', 'the', 'biggest', 'tragedy', 'of', 'this', 'young',
```

```
'century', '.', 'The', 'number', 'of', 'books', 'written', 'on', 'the',
'subject', 'and', 'the', 'number', 'of', 'films', 'directed', 'on', 'this',
'subject', ',', 'and', 'I', 'say', 'this', 'with', 'all', 'due', 'respect', ',',
'blow', 'the', 'importance', 'of', 'this', 'atrocious', 'crime', 'somewhat',
'out', 'of', 'proportion.<br', '/><br', '/>Fact', 'is', ':', 'People', 'die',
'every', 'day', 'due', 'to', 'unjust', 'actions', 'and', 'horrible', 'crimes',
'committed', 'by', 'bad', 'or', 'simply', 'lost', 'people', '.', 'We', 'have',
'a', 'war', 'in', 'Iraq', ',', 'in', 'Afghanistan', ',', 'in', 'Birma', 'and',
'lots', 'of', 'other', 'countries', '.', 'On', 'a', 'daily', 'basis', ',', 'we',
'forget', 'about', 'the', 'poverty', 'the', 'African', 'people', 'suffer',
'from', 'and', 'we', 'tend', 'do', 'empathize', 'with', 'them', 'to', 'a',
'lesser', 'degree', 'than', 'with', 'the', 'American', 'victims', 'of', '9/11',
'simply', 'because', 'they', 'are', 'black', 'and', 'because', 'their', 'lives',
"don't", 'have', 'much', 'in', 'common', 'with', 'our', 'Western', 'lives', '.',
'Africa', 'neither', 'has', 'the', 'money', 'nor', 'the', 'potential', 'to',
'commemorate', 'their', 'national', 'tragedies', 'in', 'a', 'way', 'America',
'can', '.', 'So', ',', 'what', 'I', 'am', 'saying', 'is', 'this', ':', 'The',
'reason', 'why', 'we', 'feel', 'more', 'for', 'the', '3000', 'victims', 'of',
'9/11', 'and', 'their', 'families', 'is', 'because', 'we', 'are', 'constantly',
'reminded', 'of', '9/11', '.', 'Not', 'a', 'day', 'goes', 'by', 'without', 'a',
'newspaper', 'article', ',', 'a', 'film', 'or', 'a', 'book', 'that',
'discusses', '9/11.<br', '/><br', '/>In', 'conclusion', ':', 'I',
'commiserated', 'with', 'Charlie', 'Fineman', ',', 'but', 'I', "wasn't", 'sure',
'whether', 'I', 'had', 'the', 'right', 'to', 'feel', 'for', 'him', 'more',
'than', 'for', 'a', 'Hutu', 'who', 'lost', 'his', 'entire', 'family', 'in',
'the', 'Rwandan', 'civil', 'war.<br', '/><br', '/>You', 'catch', 'my', 'thrift',
'?']
Sample label: pos
```

3 Step 2: Create Dataloader [20 points]

3.1 TODO: Define the Dataset Class [20 Points]

In the following cell, we will define the dataset class. The dataset contains the tokenized data for your model. You need to implement the following functions:

- `build_dictionary(self)`: [10 points] Creates the dictionaries `idx2word` and `word2idx`. You will represent each word in the dataset with a unique index, and keep track of this in these dictionaries. Use the hyperparameter `threshold` to control which words appear in the dictionary: a training word's frequency should be `>= threshold` to be included in the dictionary.
- `convert_text(self)`: Converts each review in the dataset to a list of indices, given by your `word2idx` dictionary. You should store this in the `textual_ids` variable, and the function does not return anything. If a word is not present in the `word2idx` dictionary, you should use the `<UNK>` token for that word. Be sure to append the `<END>` token to the end of each review.
- `get_text(self, idx)`: Return the review at `idx` in the dataset as an array of indices corresponding to the words in the review. If the length of the review is less than `max_len`, you

should pad the review with the `<PAD>` character up to the length of `max_len`. If the length is greater than `max_len`, then it should only return the first `max_len` words. The return type should be `torch.LongTensor`.

- `get_label(self, idx)`: Return the value `1` if the label for `idx` in the dataset is `positive`, and should return `0` if it is `negative`. The return type should be `torch.LongTensor`.
- `__len__(self)`: Return the total number of reviews in the dataset as an `int`.
- `__getitem__(self, idx)`: [10 points] Return the (padded) text, and the label. The return type for both these items should be `torch.LongTensor`. You should use the `get_label(self, idx)` and `get_text(self, idx)` functions here.

Note: You should convert all words to lower case in your functions.

Hint: Make sure that you use instance variables such as `self.threshold` throughout your code, rather than the global variable `THRESHOLD` (defined later on). The variable `THRESHOLD` will not be known to the autograder, and the use of it within the class will cause an autograder error.

[4]:
```
PAD = '<PAD>'
END = '<END>'
UNK = '<UNK>'

from torch.utils import data
from collections import defaultdict

class TextDataset(data.Dataset):
    def __init__(self, examples, split, threshold, max_len, idx2word=None,␣
 ↪word2idx=None):
        ### DO NOT EDIT ###

        self.examples = examples
        assert split in {'train', 'val', 'test'}
        self.split = split
        self.threshold = threshold
        self.max_len = max_len

        #custom
        self.map_token_to_unigram_frequency = {}

        # Dictionaries
        self.idx2word = idx2word
        self.word2idx = word2idx
        if split == 'train':
            self.build_dictionary()
        self.vocab_size = len(self.word2idx)

        # Convert text to indices
        self.textual_ids = []
        self.convert_text()
```

```
  def build_dictionary(self):
      '''
      Build the dictionaries idx2word and word2idx. This is only called when split='train', as these
      dictionaries are passed in to the __init__(...) function otherwise. Be sure to use self.threshold
      to control which words are assigned indices in the dictionaries.
      Returns nothing.
      '''
      assert self.split == 'train'

      # Don't change this
      self.idx2word = {0:PAD, 1:END, 2: UNK}
      self.word2idx = {PAD:0, END:1, UNK: 2}

      ##### TODO #####
      # Count the frequencies of all words in the training data (self.examples)
      # Assign idx (starting from 3) to all words having word_freq >= self.threshold
      # Make sure you call word.lower() on each word to convert it to lowercase
      # print(" BUILDING DICT")
      # print(f"training dataset is: {self.examples[0]}")

      # ('pos', ['Your', 'life', 'is', 'good', 'when', 'you', 'have', 'money', ',', 'success', 'and', 'health'])

      for data in self.examples:           #iterate through preprocessed sentences
          # print(f"data: {data}")
          testset_label, preprocessed_sentence = data[0], data[1]
          # print(f"testset_label: {testset_label}, preprocessed_sentence: {preprocessed_sentence}")
          for token in preprocessed_sentence:             #iterate through each token
              # print(f"lowercase token is: {token}")
              token = token.lower()

              if token not in self.map_token_to_unigram_frequency.keys():
                  self.map_token_to_unigram_frequency[token] = 0          # add token as key if not present as a key in map_token_to_unigram_frequency
              self.map_token_to_unigram_frequency[token] += 1             # increment the frequency of unigram
```

```
        # print(f"self.map_token_to_unigram_frequency: {self.
↪map_token_to_unigram_frequency}")
        index = 3
        for token, freq in self.map_token_to_unigram_frequency.items():
            if freq >= self.threshold:
                self.idx2word[index] = token
                self.word2idx[token] = index
                index += 1
        #
        # print(f"self.idx2word: {self.idx2word}")
        # print(f"self.word2idx: {self.word2idx}")

        pass

    def convert_text(self):
        '''
        Convert each review in the dataset (self.examples) to a list of␣
↪indices, given by self.word2idx.
        Store this in self.textual_ids; returns nothing.
        '''

        ##### TODO #####
        # Remember to replace a word with the <UNK> token if it does not exist␣
↪in the word2idx dictionary.
        # Remember to append the <END> token to the end of each review.
        for data in self.examples:           #iterate through preprocessed␣
↪sentences
            # print(f"data: {data}")
            testset_label, preprocessed_sentence = data[0], data[1]
            converted_preprocessed_sentence = []
            # print(f"testset_label: {testset_label}, preprocessed_sentence:
↪{preprocessed_sentence}")
            for token_index in range(len(preprocessed_sentence)):            ␣
↪#iterate through each token
                token = preprocessed_sentence[token_index]
                # print(f"IN CONVERT TEXT: token: {token}")
                token = token.lower()
                # print(f"lowercase token is: {token}")
                if token not in self.word2idx.keys():
                    # preprocessed_sentence[token_index] = self.word2idx[UNK]
                    converted_preprocessed_sentence.append(self.word2idx[UNK])
                if token in self.word2idx.keys():
                    # preprocessed_sentence[token_index] = self.word2idx[token]
                    converted_preprocessed_sentence.append(self.word2idx[token])
```

```
            converted_preprocessed_sentence.append(self.word2idx[END])

            # print(f"testset_label: {testset_label}, converted␣
↪preprocessed_sentence:{converted_preprocessed_sentence}")
            # self.textual_ids.
↪append([testset_label,converted_preprocessed_sentence])
            self.textual_ids.append(converted_preprocessed_sentence)
            # print(f"self.textual_ids is: {self.textual_ids}")

        pass

    def get_text(self, idx):
        '''
        Return the review at idx as a long tensor (torch.LongTensor) of␣
↪integers corresponding to the words in the review.
        You may need to pad as necessary (see above).

        <b>` get_text(self, idx) `:</b> Return the review at `idx` in the␣
↪dataset as an array of indices corresponding to the words in the review. If␣
↪the length of the review is less than `max_len`, you should pad the review␣
↪with the `<PAD>` character up to the length of `max_len`. If the length is␣
↪greater than `max_len`, then it should only return the first `max_len` words.
↪ The return type should be `torch.LongTensor`.
        '''

        ##### TODO #####
        review = self.examples[idx]
        indice_review = self.textual_ids[idx]
        # print(f"indice review is: {indice_review}")

        if len(indice_review) >= self.max_len:
            indice_review_long_tensor = torch.LongTensor(indice_review[0:self.
↪max_len])

        elif len(indice_review) < self.max_len:
            padding = [self.word2idx[PAD]] * (self.max_len - len(indice_review))
            padded_version = indice_review + padding
            # print(f"padding list is: {padding} and padded version:␣
↪{padded_version}")
            indice_review_long_tensor = torch.LongTensor(padded_version)
            # indice_review_long_tensor = torch.Tensor(padded_version)


        # print('Content of indice_review_long_tensor:',␣
↪indice_review_long_tensor)
```

```
        # print('Shape of indice_review_long_tensor:',␣
↪indice_review_long_tensor.shape, '\n')
        # print('Type of indice_review_long_tensor:', indice_review_long_tensor.
↪dtype, '\n')



        return (indice_review_long_tensor)

    def get_label(self, idx):
        '''
        This function should return the value 1 if the label for idx in the␣
↪dataset is 'positive',
        and 0 if it is 'negative'. The return type should be torch.LongTensor.
        '''
        ##### TODO #####
        review_label = self.examples[idx][0]
        # print(f"review label: {review_label}")

        if review_label == 'pos':
            return torch.squeeze(torch.LongTensor([1]))
            # print('Shape of torch.squeeze(torch.Tensor(1)):', torch.
↪squeeze(torch.Tensor(1)).shape, '\n')
            # return torch.squeeze(torch.Tensor([1]))

        elif review_label == 'neg':
            return torch.squeeze(torch.LongTensor([0]))
            # print('Shape of torch.squeeze(torch.Tensor(0)):', torch.
↪squeeze(torch.Tensor(0)).shape, '\n')
            # return torch.squeeze(torch.Tensor([0]))



    def __len__(self):
        '''
        Return the number of reviews (int value) in the dataset
        '''
        ##### TODO #####
        return int(len(self.examples))

    def __getitem__(self, idx):
        '''
        Return the review, and label of the review specified by idx.
```

```
        *   <b>` __getitem__(self, idx)`:</b> <b>[10 points]</b> Return the␣
↪(padded) text, and the label. The return type for both these items should be␣
↪`torch.LongTensor`. You should use the ` get_label(self, idx) ` and `␣
↪get_text(self, idx) ` functions here.
        '''
        ##### TODO #####
        # return self.examples[idx][1], self.examples[idx][0]
        return self.get_text(idx), self.get_label(idx)
```

##Sanity Check: Dataset Class

The code below runs a sanity check for your `Dataset` class. The tests are similar to the hidden ones in Gradescope. However, note that passing the sanity check does not guarantee that you will pass the autograder; it is intended to help you debug.

[5]:
```
### DO NOT EDIT ###

def sanityCheckDataSet():
    #        Read in the sample corpus
    reviews = [('pos', 'Your life is good when you have money, success and␣
↪health'),
               ('neg', 'Life is bad when you got not a lot')]
    data = [(x[0], preprocess(x[1])) for x in reviews]
    print("Sample dataset:")
    for x in data: print(x)

    thresholds = [1,2,3]
    print('\n--- TEST: idx2word and word2idx dictionaries ---') # max_len does␣
↪not matter for this test
    correct = [[',', '<END>', '<PAD>', '<UNK>', 'a', 'and', 'bad', 'good',␣
↪'got', 'have', 'health', 'is', 'life', 'lot', 'money', 'not', 'success',␣
↪'when', 'you', 'your'], ['<END>', '<PAD>', '<UNK>', 'is', 'life', 'when',␣
↪'you'], ['<END>', '<PAD>', '<UNK>']]
    for i in range(len(thresholds)):
        dataset = TextDataset(data, 'train', threshold=thresholds[i], max_len=3)

        has_passed, message = True, ''
        if has_passed and (dataset.vocab_size != len(dataset.word2idx) or␣
↪dataset.vocab_size != len(dataset.idx2word)):
            has_passed, message = False, 'dataset.vocab_size (' + str(dataset.
↪vocab_size) + ') must be the same length as dataset.word2idx (' +␣
↪str(len(dataset.word2idx)) + ') and dataset.idx2word ('+str(len(dataset.
↪idx2word)) +').'
        if has_passed and (dataset.vocab_size != len(correct[i])):
            has_passed, message = False, 'Your vocab size is incorrect.␣
↪Expected: ' + str(len(correct[i])) + '\tGot: ' + str(dataset.vocab_size)
```

```
      if has_passed and sorted(list(dataset.idx2word.keys())) !=␣
↪list(range(0, dataset.vocab_size)):
          has_passed, message = False, 'dataset.idx2word must have keys␣
↪ranging from 0 to dataset.vocab_size-1. Keys in your dataset.idx2word: ' +␣
↪str(sorted(list(dataset.idx2word.keys())))
      if has_passed and sorted(list(dataset.word2idx.keys())) != correct[i]:
          has_passed, message = False, 'Your dataset.word2idx has incorrect␣
↪keys. Expected: ' + str(correct[i]) + '\tGot: ' + str(sorted(list(dataset.
↪word2idx.keys())))
      if has_passed: # Check that word2idx and idx2word are consistent
          widx = sorted(list(dataset.word2idx.items()))
          idxw = sorted(list([(v,k) for k,v in dataset.idx2word.items()]))
          if not (len(widx) == len(idxw) and all([widx[q] == idxw[q] for q in␣
↪range(len(widx))])):
              has_passed, message = False, 'Your dataset.word2idx and dataset.
↪idx2word are not consistent. dataset.idx2word: ' + str(dataset.idx2word) +␣
↪'\tdataset.word2idx: ' + str(dataset.word2idx)

      status = 'PASSED' if has_passed else 'FAILED'
      print('\tthreshold:', thresholds[i], '\tmax_len:', 3, '\t'+status,␣
↪'\t'+message)

  print('\n--- TEST: len(dataset) ---')
  has_passed = len(dataset) == 2
  if has_passed: print('\tPASSED')
  else: print('\tlen(dataset) is incorrect. Expected: 2\tGot: ' +␣
↪str(len(dataset)))

  print('\n--- TEST: __getitem__(self, idx) ---')
  max_lens = [3,8,15]
  idxes = [0,1]
  combos = [{'threshold': t, 'max_len': m, 'idx': idx} for t in thresholds␣
↪for m in max_lens for idx in idxes]
```

```
  correct = [(torch.tensor([3, 4, 5]), torch.tensor(1)), (torch.tensor([ 4, 5, 15]), torch.tensor(0)), (torch.tensor([ 3,  4,  5,  6,  7,  8,  9, 10]), torch.tensor(1)), (torch.tensor([ 4,  5, 15,  7,  8, 16, 17, 18]), torch.tensor(0)), (torch.tensor([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,  1,  0,  0]), torch.tensor(1)), (torch.tensor([ 4,  5, 15,  7,  8, 16, 17, 18, 19,  1,  0,  0,  0,  0,  0]), torch.tensor(0)), (torch.tensor([2, 3, 4]), torch.tensor(1)), (torch.tensor([3, 4, 2]), torch.tensor(0)), (torch.tensor([2, 3, 4, 2, 5, 6, 2, 2]), torch.tensor(1)), (torch.tensor([3, 4, 2, 5, 6, 2, 2, 2]), torch.tensor(0)), (torch.tensor([2, 3, 4, 2, 5, 6, 2, 2, 2, 2, 2, 1, 0, 0]), torch.tensor(1)), (torch.tensor([3, 4, 2, 5, 6, 2, 2, 2, 2, 1, 0, 0, 0, 0, 0]), torch.tensor(0)), (torch.tensor([2, 2, 2]), torch.tensor(1)), (torch.tensor([2, 2, 2]), torch.tensor(0)), (torch.tensor([2, 2, 2, 2, 2, 2, 2]), torch.tensor(1)), (torch.tensor([2, 2, 2, 2, 2, 2, 2]), torch.tensor(0)), (torch.tensor([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0, 0]), torch.tensor(1)), (torch.tensor([2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0, 0, 0, 0, 0]), torch.tensor(0))]
  for i in range(len(combos)):
      combo = combos[i]
      dataset = TextDataset(data, 'train', threshold=combo['threshold'], max_len=combo['max_len'])
      returned = dataset.__getitem__(combo['idx'])

      has_passed, message = True, ''
      if has_passed and len(returned) != 2:
          has_passed, message = False, 'dataset.__getitem__(idx) must return 2 things. Got ' + str(len(returned)) +' things instead.'
      if has_passed and (type(returned[0]) != torch.Tensor or type(returned[1]) != torch.Tensor):
          has_passed, message = False, 'Both returns must be of type torch.Tensor. Got: (' + str(type(returned[0])) + ', ' + str(type(returned[1])) + ')'
      if has_passed and (returned[0].shape != correct[i][0].shape):
          has_passed, message = False, 'Shape of first return is incorrect. Expected: ' + str(correct[i][0].shape) + '.\tGot: ' + str(returned[0].shape)
      if has_passed and (returned[1].shape != correct[i][1].shape):
          has_passed, message = False, 'Shape of second return is incorrect. Expected: ' + str(correct[i][1].shape) + '.\tGot: ' + str(returned[1].shape) + '\n\t\tHint: torch.Size([]) means that the tensor should be dimensionless (just a number). Try squeezing your result.'
      if has_passed and (returned[1] != correct[i][1]):
          has_passed, message = False, 'Label (second return) is incorrect. Expected: ' + str(correct[i][1]) + '.\tGot: ' + str(returned[1])
      if has_passed:
          correct_padding_idxes, your_padding_idxes = torch.where(correct[i][0] == 0)[0], torch.where(returned[0] == dataset.word2idx[PAD])[0]
```

```
            if not (correct_padding_idxes.shape == your_padding_idxes.shape and␣
 ↪torch.all(correct_padding_idxes == your_padding_idxes)):
                has_passed, message = False, 'Padding is not correct. Expected␣
 ↪padding indxes: ' + str(correct_padding_idxes) + '.\tYour padding indexes: '␣
 ↪+ str(your_padding_idxes)

        status = 'PASSED' if has_passed else 'FAILED'
        print('\tthreshold:', combo['threshold'], '\tmax_len:',␣
 ↪combo['max_len'] , '\tidx:', combo['idx'], '\t'+status, '\t'+message)

if __name__ == '__main__':
    sanityCheckDataSet()
```

```
Sample dataset:
('pos', ['Your', 'life', 'is', 'good', 'when', 'you', 'have', 'money', ',',
'success', 'and', 'health'])
('neg', ['Life', 'is', 'bad', 'when', 'you', 'got', 'not', 'a', 'lot'])

--- TEST: idx2word and word2idx dictionaries ---
        threshold: 1    max_len: 3      PASSED
        threshold: 2    max_len: 3      PASSED
        threshold: 3    max_len: 3      PASSED

--- TEST: len(dataset) ---
        PASSED

--- TEST: __getitem__(self, idx) ---
        threshold: 1    max_len: 3      idx: 0  PASSED
        threshold: 1    max_len: 3      idx: 1  PASSED
        threshold: 1    max_len: 8      idx: 0  PASSED
        threshold: 1    max_len: 8      idx: 1  PASSED
        threshold: 1    max_len: 15     idx: 0  PASSED
        threshold: 1    max_len: 15     idx: 1  PASSED
        threshold: 2    max_len: 3      idx: 0  PASSED
        threshold: 2    max_len: 3      idx: 1  PASSED
        threshold: 2    max_len: 8      idx: 0  PASSED
        threshold: 2    max_len: 8      idx: 1  PASSED
        threshold: 2    max_len: 15     idx: 0  PASSED
        threshold: 2    max_len: 15     idx: 1  PASSED
        threshold: 3    max_len: 3      idx: 0  PASSED
        threshold: 3    max_len: 3      idx: 1  PASSED
        threshold: 3    max_len: 8      idx: 0  PASSED
        threshold: 3    max_len: 8      idx: 1  PASSED
        threshold: 3    max_len: 15     idx: 0  PASSED
        threshold: 3    max_len: 15     idx: 1  PASSED
```

The following cell builds the dataset on the IMDb movie reviews and prints an example:

[6]:
```
### DO NOT EDIT ###

if __name__=='__main__':
    train_dataset = TextDataset(train_data, 'train', threshold=10, max_len=150)
    print('Vocab size:', train_dataset.vocab_size, '\n')

    randidx = random.randint(0, len(train_dataset)-1)
    text, label = train_dataset[randidx]
    print('Example text:')
    print(train_data[randidx][1])
    print(text)
    print('\nExample label:')
    print(train_data[randidx][0])
    print(label)
```

```
Vocab size: 19002

Example text:
["I'm", 'still', 'trying', 'to', 'decide', 'if', 'this', 'is', 'indeed', ',',
'the', 'worst', 'film', 'I', 'have', 'ever', 'seen', '-', 'A', 'very',
'disturbing', 'problem', 'with', 'this', 'film', 'is', 'that', 'real',
'scientists', 'are', 'interviewed', ',', 'but', 'their', 'footage', 'is',
'edited', 'to', 'make', 'it', 'look', 'as', 'though', 'they', 'support', 'the',
'ideas', 'of', 'the', 'many', 'BSers', 'who', 'populate', 'this', 'film', '.',
'The', 'BS', 'to', 'signal', 'ratio', 'of', 'the', 'interviews', 'is', 'about',
'ten', 'thousand', 'to', 'one', '-', 'at', 'the', 'end', ',', 'the',
'interviewees', 'seem', 'to', 'be', 'saying', ',', '"', 'We', 'want', 'you',
'to', '_think_', '!!"', ',', 'but', 'they', 'themselves', 'are', 'too', 'lazy',
'to', 'do', 'simple', 'research', 'about', 'things', 'they', 'assert', 'as',
'fact.<br', '/><br', '/>If', 'you', 'feel', 'that', 'you', 'are', 'open-minded',
',', 'and', 'wish', 'to', 'expand', 'your', 'consciousness', ',', 'please',
'be', 'open-minded', 'enough', 'to', 'read', 'some', 'actual', 'books', 'about',
'quantum', 'theory', ':', '"', "Einstein's", 'Universe"', ',', 'Nigel',
'Calder', '(', 'a', 'slim', 'volume', ',', 'not', 'a', 'challenge)', ',', '"',
'The', 'Cosmic', 'Code"', ',', 'by', 'Heinz', 'Pagels', '.', 'If', 'you',
"can't", 'bring', 'yourself', 'to', 'read', 'a', 'book', ',', 'please', "don't",
'complain', 'to', 'reviewers', 'about', 'being', '"', 'open-minded".<br',
'/><br', '/>To', 'recap', ',', 'this', 'film', 'is', 'just', 'unbelievably',
'bad.<br', '/><br', '/>You', 'know', "what's", 'a', 'really', 'good', 'film',
'which', 'questions', 'the', 'nature', 'of', 'reality', '?', '"', 'Thirteenth',
'Floor"', ',', 'directed', 'by', 'Roland', 'Emmerich', ',', 'with', 'Craig',
'Bierko', ',', 'Gretchen', 'Mol', ',', 'Vincent', "D'onofrio", '.', 'Smart',
',', 'sexy', ',', 'thought-provoking', '.']
tensor([  317,  1040,   159,    52,  4093,    94,    59,    24,  1564,    12,
             9,  3388,    14,   309,    51,  1286,    92,  1400,    35,   165,
          2491,   758,     8,    59,    14,    24,    54,  1316, 14058,    98,
         16810,    12,    86,   187,   981,    24, 12666,    52,   482,    64,
```

```
        1684,    66,   763,   218,  2357,     9,   781,    18,     9,   175,
           2,    31, 16854,    59,    14,    21,     9,     2,    52, 10994,
        6242,    18,     9,  6557,    24,   380,  1728,   666,    52,    78,
        1400,   264,     9,   416,    12,     9,     2,   328,    52,   506,
        3143,    12,    76,   545,   470,    95,    52,     2,     2,    12,
          86,   218,  1430,    98,    68,  8305,    52,   428,   190,  7489,
         380,   378,   218,  8045,    66, 15405,    48,  1927,    95,   986,
          54,    95,    98, 16091,    12,    41,  1667,    52,  9067,   323,
        9225,    12,  1600,   506, 16091,   707,    52,   322,   163,  4814,
        3796,   380, 17656,  6283,    63,    76,     2,     2,    12, 16830,
           2,   263,    35,  7487, 16900,    12,   189,    35,     2,    12])
```

```
Example label:
neg
tensor(0)
```

4 Step 3: Train a Convolutional Neural Network (CNN) [40 points]

4.1 TODO: Define the CNN Model [20 points]

Here you will define your convolutional neural network for text classification. We provide you with the CNN class, you need to fill in parts of the `__init__(...)` and `forward(...)` functions. Each of these functions is worth 10 points.

We have provided you with instructions and hints in the comments. In particular, pay attention to the desired shapes; you may find it helpful to print the shape of the tensors as you code. It may also help to keep PyTorch documentation open for the modules & functions you are using, since they describe input and output dimensions.

[7]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F

class CNN(nn.Module):
    def __init__(self, vocab_size, embed_size, out_channels, filter_heights,␣
 ↪stride, dropout, num_classes, pad_idx):
        super(CNN, self).__init__()

        ##### TODO #####
        # Create an embedding layer (https://pytorch.org/docs/stable/generated/
 ↪torch.nn.Embedding.html)
        #   to represent the words in your vocabulary. Make sure to use␣
 ↪vocab_size, embed_size, and pad_idx here.
        # print(f" vocab size is: {vocab_size}, embed size is: {embed_size},␣
 ↪filter heights: {filter_heights}")
        # number of embeddings = vocab size??
```

```
        # embedding size = embed size? = max len??
        self.embedding = nn.Embedding(vocab_size, embed_size,␣
↪padding_idx=pad_idx)
        # self.embedding = nn.Embedding(vocab_size, embed_size)
        #self.embedding is learnable

        # print(f"weights of embedding: {self.embedding.weight}")


        # Define multiple Convolution layers (nn.Conv2d) with filter (kernel)␣
↪size [filter_height, embed_size] based on your
        #   different filter_heights.
        # Input channels will be 1 and output channels will be out_channels␣
↪(these many different filters will be trained
        #   for each convolution layer)


        # If you want, you can store a list of modules inside nn.ModuleList.
        self.conv_module_list = nn.ModuleList([nn.Conv2d(in_channels=1,␣
↪out_channels=out_channels, kernel_size=(filter_height, embed_size),␣
↪stride=stride) for filter_height in filter_heights])
        # print(f"conv module list is: {self.conv_module_list}")

        self.conv_model_outputs = {}

        # conv module list is: ModuleList(
        #                                   (0): Conv2d(1, 32, kernel_size=(3,␣
↪32), stride=(1, 1))
        #                                   (1): Conv2d(1, 32, kernel_size=(4,␣
↪32), stride=(1, 1))
        #                                   (2): Conv2d(1, 32, kernel_size=(5,␣
↪32), stride=(1, 1))
        #                                 )

        # self.maxpool = nn.MaxPool1d(2)

        # Note: even though your conv layers are nn.Conv2d, we are doing a 1d␣
↪convolution since we are only moving the filter
        #   in one direction

        # Create a dropout layer (nn.Dropout) using dropout
        self.dropout_layer = nn.Dropout(dropout)

        # Define a linear layer (nn.Linear) that consists of num_classes units
        #   and takes as input the $concatenated output$ for all cnn layers␣
↪(out_channels * num_of_cnn_layers units) ???
```

```
        # print(f"out_channels * len(filter_heights), :{out_channels *␣
↪len(filter_heights)}")
        self.dense_layer = nn.Linear(out_channels * len(filter_heights),␣
↪num_classes)
        # another dense layer
        # self.d2 = nn.Linear(128, 10)


    def forward(self, texts):
        """
        texts: LongTensor [batch_size, max_len]

        Returns output: Tensor [batch_size, num_classes]

        # Pass these texts to each of your conv layers and compute their output␣
↪as follows:
        #   Your cnn output will have shape [batch_size, out_channels, *, 1]␣
↪where * depends on filter_height and stride
        #   Convert to shape [batch_size, out_channels, *] (see torch's␣
↪squeeze() function)
        #   Apply non-linearity on it (F.relu() is a commonly used one. Feel␣
↪free to try others)
        #   Take the max value across last dimension to have shape [batch_size,␣
↪out_channels]
        # Concatenate (torch.cat) outputs from all your cnns [batch_size,␣
↪(out_channels*num_of_cnn_layers)]
        #
        """
        ##### TODO #####

        # print('Content of texts:', texts)
        # print('Shape of texts:', texts.shape, '\n')
        # print('Type of texts:', texts.dtype, '\n')

        texts = texts.type(torch.int64)
        final_embedding = self.embedding(texts)

        # print('Content of embedding:', final_embedding)
        # print('Shape of embedding:', final_embedding.shape, '\n')
        # print('Type of embedding:', final_embedding.dtype, '\n')
        # Resulting: shape: [batch_size, max_len, embed_size]
        # Shape of embedding: torch.Size([1, 150, 16])


        # Input to conv should have 1 channel. Take a look at torch's␣
↪unsqueeze() function
```

```
        #   Resulting shape: [batch_size, 1, MAX_LEN, embed_size]
        y = torch.unsqueeze(final_embedding, 1)
        # print('Shape of unsqueeze:', y.shape, '\n')  #CORRECT

        # Pass these texts to each of your conv layers and compute their output␣
↪as follows:
        #   Your cnn output will have shape [batch_size, out_channels, *, 1]␣
↪where * depends on filter_height and stride
        #   Convert to shape [batch_size, out_channels, *] (see torch's␣
↪squeeze() function)
        #   Apply non-linearity on it (F.relu() is a commonly used
        #   one. Feel free to try others)

        list_to_be_concatenated = []

        for i, conv_model in enumerate(self.conv_module_list):

            # Shape of x: torch.Size([1, 32, 148, 1])   #148 = * = depends on␣
↪filter_height and stride
            # Shape of x1_squeezed: torch.Size([1, 32, 148])
            # Shape of x1_squeezed_relu: torch.Size([1, 32, 148])

            # print(f"i: {i}, conv_model: {conv_model}")
            x = conv_model(y)
            # print('Shape of x:', x.shape, '\n')  #

            x1_squeezed = torch.squeeze(x, dim=3)
            # print('Shape of x1_squeezed:', x1_squeezed.shape, '\n')

            x1_squeezed_relu = F.relu(x1_squeezed) #??? #TODO: apply non␣
↪linearity here??
            # print('Shape of x1_squeezed_relu:', x1_squeezed_relu.shape, '\n')

            #   Take the max value across last dimension to have shape␣
↪[batch_size, out_channels] ??? #TODO: how and why?
            # torch.nn.functional.max_pool1d(input, kernel_size, stride=None,␣
↪padding=0, dilation=1, ceil_mode=False, return_indices=False)
            maxpool_values, maxpool_indices = F.max_pool1d(x1_squeezed_relu,␣
↪x1_squeezed_relu.shape[2], return_indices=True)
            # print(f"maxpool_values is: {maxpool_values}")
            # print(f"maxpool_values.shape: {maxpool_values.shape}")

            # print(f"maxpool_indices is: {maxpool_indices}, maxpool_indices.
↪shape: {maxpool_indices.shape}")

            maxpool_values_squeezed = torch.squeeze(maxpool_values, dim=2)
```

```
            # print(f" maxpool_values_squeezed is: {maxpool_values_squeezed},␣
↪maxpool_values_squeezed.shape: {maxpool_values_squeezed.shape}")
            list_to_be_concatenated.append(maxpool_values_squeezed)

        ##########
        # i: 0, conv_model: Conv2d(1, 32, kernel_size=(3, 16), stride=(1, 1))
        # i: 1, conv_model: Conv2d(1, 32, kernel_size=(4, 16), stride=(1, 1))
        # i: 2, conv_model: Conv2d(1, 32, kernel_size=(5, 16), stride=(1, 1))
        # shape of tensor: torch.Size([1, 32, 148])
        # shape of tensor: torch.Size([1, 32, 147])
        # shape of tensor: torch.Size([1, 32, 146])
        ##########
        #  Concatenate (torch.cat) outputs from all your cnns [batch_size,␣
↪(out_channels*num_of_cnn_layers)]
        list_to_be_concatenated_tensor = torch.cat([i for i in␣
↪list_to_be_concatenated], dim=1)

        # print(f"list_to_be_concatenated_tensor is:␣
↪{list_to_be_concatenated_tensor}, shape: {list_to_be_concatenated_tensor.
↪shape}")
        # print(f"list_to_be_concatenated_tensor:␣
↪{list_to_be_concatenated_tensor.shape}")

        #  #print shapes
        # for tensor in list_to_be_concatenated:
        #     print(f"shape of tensor: {tensor.shape}")

        #[batch_size, (out_channels*num_of_cnn_layers)]
        # shape of tensor: torch.Size([20, 32])
        # shape of tensor: torch.Size([20, 32])
        # shape of tensor: torch.Size([20, 32])
        # list_to_be_concatenated_tensor: torch.Size([20, 96])


        # Let's understand what you just did:
        #   Since each cnn is of different filter_height, it will look at␣
↪different number of words at a time
        #     So, a filter_height of 3 means your cnn looks at 3 words␣
↪(3-grams) at a time and tries to extract some information from it
        #   Each cnn will learn out_channels number of features from the words␣
↪it sees at a time
        #   Then you applied a non-linearity and took the max value for all␣
↪channels
        #     You are essentially trying to find important n-grams from the␣
↪entire text
```

```
        # Everything happens on a batch simultaneously hence you have that␣
↪additional batch_size as the first dimension


        # flattened_tensor = torch.flatten(list_to_be_concatenated_tensor)
        # print(f"flattened_tensor is: {flattened_tensor}, shape:␣
↪{flattened_tensor.shape}")

        # Apply dropout
        dropout = self.dropout_layer(list_to_be_concatenated_tensor)
        # print(f"dropout is: {dropout}, shape: {dropout.shape}")

        # Pass your output through the linear layer and return its output
        #   Resulting shape: [batch_size, num_classes]
        # x = self.dense_layer(self.relu(x))

        linear = (self.dense_layer(dropout))
        # print(f"linear is: {linear}, shape: {linear.shape}, linear dtype:␣
↪{linear.dtype}")


        # linear = linear.type(torch.int32)

        ##### NOTE: Do not apply a sigmoid or softmax to the final output -␣
↪done in training method!


        return linear
        # return None
```

##Sanity Check: CNN Model

The code below runs a sanity check for your `CNN` class. The tests are similar to the hidden ones in Gradescope. However, note that passing the sanity check does not guarantee that you will pass the autograder; it is intended to help you debug.

[8]:
```
### DO NOT EDIT ###

count_parameters = lambda model: sum(p.numel() for p in model.parameters() if p.
↪requires_grad)

def sanityCheckModel(all_test_params, NN, expected_outputs, init_or_forward,␣
↪data_loader):
    # print('--- TEST: ' + ('Number of Model Parameters (tests __init__(...))'␣
↪if init_or_forward=='init' else 'Output shape of forward(...)') + ' ---')

    if init_or_forward == "forward":
```

```
        # Reading the first batch of data for testing
        for texts_, labels_ in data_loader:
            texts_batch, labels_batch = texts_, labels_
            break

    for tp_idx, (test_params, expected_output) in␣
↪enumerate(zip(all_test_params, expected_outputs)):
        if init_or_forward == "forward":
            batch_size = test_params['batch_size']
            texts = texts_batch[:batch_size]

        # Construct the student model
        tps = {k:v for k, v in test_params.items() if k != 'batch_size'}
        stu_nn = NN(**tps)

        if init_or_forward == "forward":
            with torch.no_grad():
                stu_out = stu_nn(texts)
            ref_out_shape = expected_output

            has_passed = torch.is_tensor(stu_out)
            if not has_passed: msg = 'Output must be a torch.Tensor; received '␣
↪+ str(type(stu_out))
            else:
                has_passed = stu_out.shape == ref_out_shape
                msg = 'Your Output Shape: ' + str(stu_out.shape)


            status = 'PASSED' if has_passed else 'FAILED'
            message = '\t' + status + "\t Init Input: " + str({k:v for k,v in␣
↪tps.items()}) + '\tForward Input Shape: ' + str(texts.shape) + '\tExpected␣
↪Output Shape: ' + str(ref_out_shape) + '\t' + msg
            print(message)
        else:
            stu_num_params = count_parameters(stu_nn)
            ref_num_params = expected_output
            comparison_result = (stu_num_params == ref_num_params)

            status = 'PASSED' if comparison_result else 'FAILED'
            message = '\t' + status + "\tInput: " + str({k:v for k,v in␣
↪test_params.items()}) + ('\tExpected Num. Params: ' + str(ref_num_params) +␣
↪'\tYour Num. Params: '+ str(stu_num_params))
            print(message)

        del stu_nn
```

```
if __name__ == '__main__':
    # Test init
```

```
   inputs = [{'vocab_size': 1000, 'embed_size': 16, 'out_channels': 32,␣
↪'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'out_channels': 32,␣
↪'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes': 3,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'out_channels': 32,␣
↪'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'out_channels': 32,␣
↪'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0, 'num_classes': 3,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'out_channels': 32,␣
↪'filter_heights': [5, 10], 'stride': 1, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'out_channels': 32,␣
↪'filter_heights': [5, 10], 'stride': 1, 'dropout': 0, 'num_classes': 3,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'out_channels': 32,␣
↪'filter_heights': [5, 10], 'stride': 3, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'out_channels': 32,␣
↪'filter_heights': [5, 10], 'stride': 3, 'dropout': 0, 'num_classes': 3,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'out_channels': 128,␣
↪'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'out_channels': 128,␣
↪'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes': 3,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'out_channels': 128,␣
↪'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'out_channels': 128,␣
↪'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0, 'num_classes': 3,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'out_channels': 128,␣
↪'filter_heights': [5, 10], 'stride': 1, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'out_channels': 128,␣
↪'filter_heights': [5, 10], 'stride': 1, 'dropout': 0, 'num_classes': 3,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'out_channels': 128,␣
↪'filter_heights': [5, 10], 'stride': 3, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'out_channels': 128,␣
↪'filter_heights': [5, 10], 'stride': 3, 'dropout': 0, 'num_classes': 3,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 32, 'out_channels': 32,␣
↪'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 32, 'out_channels': 32,␣
↪'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes': 3,␣
```

```
  expected_outputs = [22434, 22531, 22434, 22531, 23874, 23939, 23874, 23939,␣
↪41730, 42115, 41730, 42115, 47490, 47747, 47490, 47747, 44578, 44675, 44578,␣
↪44675, 47554, 47619, 47554, 47619, 82306, 82691, 82306, 82691, 94210, 94467,␣
↪94210, 94467]

  sanityCheckModel(inputs, CNN, expected_outputs, "init", None)
  print()

  # Test forward
```

```
   inputs = [{'vocab_size': 29730, 'embed_size': 16, 'out_channels': 32,␣
↪'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0, 'batch_size': 1}, {'vocab_size': 29730, 'embed_size': 16,␣
↪'out_channels': 32, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0,␣
↪'num_classes': 2, 'pad_idx': 0, 'batch_size': 20}, {'vocab_size': 29730,␣
↪'embed_size': 16, 'out_channels': 32, 'filter_heights': [3, 4, 5], 'stride':␣
↪3, 'dropout': 0, 'num_classes': 2, 'pad_idx': 0, 'batch_size': 1},␣
↪{'vocab_size': 29730, 'embed_size': 16, 'out_channels': 32, 'filter_heights':
↪ [3, 4, 5], 'stride': 3, 'dropout': 0, 'num_classes': 2, 'pad_idx': 0,␣
↪'batch_size': 20}, {'vocab_size': 29730, 'embed_size': 16, 'out_channels':␣
↪32, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0, 'batch_size': 1}, {'vocab_size': 29730, 'embed_size': 16,␣
↪'out_channels': 32, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0,␣
↪'num_classes': 2, 'pad_idx': 0, 'batch_size': 20}, {'vocab_size': 29730,␣
↪'embed_size': 16, 'out_channels': 32, 'filter_heights': [5, 10], 'stride':␣
↪3, 'dropout': 0, 'num_classes': 2, 'pad_idx': 0, 'batch_size': 1},␣
↪{'vocab_size': 29730, 'embed_size': 16, 'out_channels': 32, 'filter_heights':
↪ [5, 10], 'stride': 3, 'dropout': 0, 'num_classes': 2, 'pad_idx': 0,␣
↪'batch_size': 20}, {'vocab_size': 29730, 'embed_size': 16, 'out_channels':␣
↪128, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes':␣
↪2, 'pad_idx': 0, 'batch_size': 1}, {'vocab_size': 29730, 'embed_size': 16,␣
↪'out_channels': 128, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0,␣
↪'num_classes': 2, 'pad_idx': 0, 'batch_size': 20}, {'vocab_size': 29730,␣
↪'embed_size': 16, 'out_channels': 128, 'filter_heights': [3, 4, 5], 'stride':
↪ 3, 'dropout': 0, 'num_classes': 2, 'pad_idx': 0, 'batch_size': 1},␣
↪{'vocab_size': 29730, 'embed_size': 16, 'out_channels': 128,␣
↪'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0, 'num_classes': 2,␣
```

```
    expected_outputs = [torch.Size([1, 2]), torch.Size([20, 2]), torch.Size([1,␣
↪2]), torch.Size([20, 2]), torch.Size([1, 2]), torch.Size([20, 2]), torch.
↪Size([1, 2]), torch.Size([20, 2]), torch.Size([1, 2]), torch.Size([20, 2]),␣
↪torch.Size([1, 2]), torch.Size([20, 2]), torch.Size([1, 2]), torch.Size([20,␣
↪2]), torch.Size([1, 2]), torch.Size([20, 2]), torch.Size([1, 2]), torch.
↪Size([20, 2]), torch.Size([1, 2]), torch.Size([20, 2]), torch.Size([1, 2]),␣
↪torch.Size([20, 2]), torch.Size([1, 2]), torch.Size([20, 2]), torch.Size([1,␣
↪2]), torch.Size([20, 2]), torch.Size([1, 2]), torch.Size([20, 2]), torch.
↪Size([1, 2]), torch.Size([20, 2]), torch.Size([1, 2]), torch.Size([20, 2])]
    sanity_dataset = TextDataset(train_data, 'train', 5, 150)
    sanity_loader = torch.utils.data.DataLoader(sanity_dataset, batch_size=50,␣
↪shuffle=True, num_workers=2, drop_last=True)

    sanityCheckModel(inputs, CNN, expected_outputs, "forward", sanity_loader)
```

```
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
32, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}       Expected Num. Params: 22434     Your Num. Params: 22434
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
32, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0}       Expected Num. Params: 22531     Your Num. Params: 22531
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
32, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}       Expected Num. Params: 22434     Your Num. Params: 22434
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
32, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0}       Expected Num. Params: 22531     Your Num. Params: 22531
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
32, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0} Expected Num. Params: 23874     Your Num. Params: 23874
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
32, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0} Expected Num. Params: 23939     Your Num. Params: 23939
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
32, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0} Expected Num. Params: 23874     Your Num. Params: 23874
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
32, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0} Expected Num. Params: 23939     Your Num. Params: 23939
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
128, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}      Expected Num. Params: 41730     Your Num. Params: 41730
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
128, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0}      Expected Num. Params: 42115     Your Num. Params: 42115
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
128, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0, 'num_classes': 2,
```

```
'pad_idx': 0}      Expected Num. Params: 41730     Your Num. Params: 41730
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
128, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0}      Expected Num. Params: 42115     Your Num. Params: 42115
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
128, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}        Expected Num. Params: 47490     Your Num. Params: 47490
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
128, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0}        Expected Num. Params: 47747     Your Num. Params: 47747
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
128, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}        Expected Num. Params: 47490     Your Num. Params: 47490
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'out_channels':
128, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0}        Expected Num. Params: 47747     Your Num. Params: 47747
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
32, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}       Expected Num. Params: 44578     Your Num. Params: 44578
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
32, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0}       Expected Num. Params: 44675     Your Num. Params: 44675
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
32, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}       Expected Num. Params: 44578     Your Num. Params: 44578
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
32, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0}       Expected Num. Params: 44675     Your Num. Params: 44675
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
32, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0} Expected Num. Params: 47554     Your Num. Params: 47554
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
32, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0} Expected Num. Params: 47619     Your Num. Params: 47619
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
32, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0} Expected Num. Params: 47554     Your Num. Params: 47554
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
32, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0} Expected Num. Params: 47619     Your Num. Params: 47619
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
128, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}      Expected Num. Params: 82306     Your Num. Params: 82306
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
128, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0}      Expected Num. Params: 82691     Your Num. Params: 82691
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
128, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0, 'num_classes': 2,
```

```
'pad_idx': 0}      Expected Num. Params: 82306     Your Num. Params: 82306
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
128, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0}      Expected Num. Params: 82691     Your Num. Params: 82691
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
128, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}        Expected Num. Params: 94210     Your Num. Params: 94210
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
128, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0}        Expected Num. Params: 94467     Your Num. Params: 94467
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
128, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}        Expected Num. Params: 94210     Your Num. Params: 94210
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 32, 'out_channels':
128, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0, 'num_classes': 3,
'pad_idx': 0}        Expected Num. Params: 94467     Your Num. Params: 94467

        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'out_channels': 32, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}        Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'out_channels': 32, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}        Forward Input Shape: torch.Size([20,
150])      Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
torch.Size([20, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'out_channels': 32, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}        Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'out_channels': 32, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}        Forward Input Shape: torch.Size([20,
150])      Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
torch.Size([20, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'out_channels': 32, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'out_channels': 32, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([20, 150])
Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
torch.Size([20, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
```

```
'out_channels': 32, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'out_channels': 32, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([20, 150])
Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
torch.Size([20, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'out_channels': 128, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}       Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'out_channels': 128, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}       Forward Input Shape: torch.Size([20, 150])
Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
torch.Size([20, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'out_channels': 128, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}       Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'out_channels': 128, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}       Forward Input Shape: torch.Size([20, 150])
Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
torch.Size([20, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'out_channels': 128, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0} Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'out_channels': 128, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0} Forward Input Shape: torch.Size([20, 150])
Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
torch.Size([20, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'out_channels': 128, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0} Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'out_channels': 128, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0} Forward Input Shape: torch.Size([20, 150])
Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
```

```
torch.Size([20, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 32, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}        Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 32, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}        Forward Input Shape: torch.Size([20,
150])      Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
torch.Size([20, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 32, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}        Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 32, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}        Forward Input Shape: torch.Size([20,
150])      Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
torch.Size([20, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 32, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 32, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([20, 150])
Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
torch.Size([20, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 32, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 32, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([20, 150])
Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
torch.Size([20, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 128, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}       Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 128, 'filter_heights': [3, 4, 5], 'stride': 1, 'dropout': 0,
```

```
'num_classes': 2, 'pad_idx': 0}       Forward Input Shape: torch.Size([20, 150])
Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
torch.Size([20, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 128, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}       Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 128, 'filter_heights': [3, 4, 5], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}       Forward Input Shape: torch.Size([20, 150])
Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
torch.Size([20, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 128, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0} Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 128, 'filter_heights': [5, 10], 'stride': 1, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0} Forward Input Shape: torch.Size([20, 150])
Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
torch.Size([20, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 128, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0} Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 32,
'out_channels': 128, 'filter_heights': [5, 10], 'stride': 3, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0} Forward Input Shape: torch.Size([20, 150])
Expected Output Shape: torch.Size([20, 2])      Your Output Shape:
torch.Size([20, 2])
```

[9]:
```
# class MyModule(nn.Module):
#     def __init__(self):
#         super(MyModule, self).__init__()
#         self.linears = nn.ModuleList([nn.Linear(10, 10) for i in range(10)])
#
#     def forward(self, x):
#         # ModuleList can act as an iterable, or be indexed using ints
#         for i, l in enumerate(self.linears):
#             print(f"i: {i}, l: {l}")
#
#             first_part = self.linears[i // 2](x)
#             second_part =  l(x)
#             x = first_part + second_part
```

```
#
#         return x
```

[10]:
```
# test_object = MyModule()
# test_object.forward(torch.ones(10,10))
```

4.2 Train CNN Model

First, we initialize the train and test dataloaders. A dataloader is responsible for providing batches of data to your model. Notice how we first instantiate datasets for the train and test data, and that we use the training vocabulary for both.

You do not need to edit this cell.

[11]:
```
if __name__=='__main__':
    THRESHOLD = 5 # Don't change this
    MAX_LEN = 200 # Don't change this
    BATCH_SIZE = 128 # Feel free to try other batch sizes

    train_dataset = TextDataset(train_data, 'train', THRESHOLD, MAX_LEN)
    train_loader = torch.utils.data.DataLoader(train_dataset,␣
 ↪batch_size=BATCH_SIZE, shuffle=True, num_workers=2, drop_last=True)

    test_dataset = TextDataset(test_data, 'test', THRESHOLD, MAX_LEN,␣
 ↪train_dataset.idx2word, train_dataset.word2idx)
    test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=1,␣
 ↪shuffle=False, num_workers=1, drop_last=False)
```

Now we provide you with a function that takes your model and trains it on the data.

You do not need to edit this cell. However, you may want to write code to save your model periodically, as Colab connections are not permanent. See the tutorial here if you wish to do this: https://pytorch.org/tutorials/beginner/saving_loading_models.html.

[12]:
```
### DO NOT EDIT ###

from tqdm.notebook import tqdm

def train_model(model, num_epochs, data_loader, optimizer, criterion):
    print('Training Model...')
    model.train()
    for epoch in tqdm(range(num_epochs)):
        epoch_loss = 0
        epoch_acc = 0
        for texts, labels in data_loader:
            texts = texts.to(DEVICE) # shape: [batch_size, MAX_LEN]
            labels = labels.to(DEVICE) # shape: [batch_size]
```

```
            # print(f"labels is: {labels}, labels shape: {labels.shape}, labels␣
↪dtype: {labels.dtype}")

            optimizer.zero_grad()

            output = model(texts)
            # print(f"output is: {output}, output shape: {output.shape}, output␣
↪dtype: {output.dtype}")

            acc = accuracy(output, labels)

            loss = criterion(output, labels)
            loss.backward()
            optimizer.step()

            epoch_loss += loss.item()
            epoch_acc += acc.item()
        print('[TRAIN]\t Epoch: {:2d}\t Loss: {:.4f}\t Train Accuracy: {:.2f}%'.
↪format(epoch+1, epoch_loss/len(data_loader), 100*epoch_acc/len(data_loader)))
    print('Model Trained!\n')
```

Here are some other helper functions we will need.

[13]:
```
### DO NOT EDIT ###

def accuracy(output, labels):
    """
    Returns accuracy per batch
    output: Tensor [batch_size, n_classes]
    labels: LongTensor [batch_size]
    """
    preds = output.argmax(dim=1) # find predicted class
    correct = (preds == labels).sum().float() # convert into float for division
    acc = correct / len(labels)
    return acc
```

Now you can instantiate your model. We provide you with some recommended hyperparameters; you should be able to get the desired accuracy with these, but feel free to play around with them.

[14]:
```
if __name__=='__main__':
    cnn_model = CNN(vocab_size = train_dataset.vocab_size, # Don't change this
                embed_size = 128,
                out_channels = 64,
                filter_heights = [2, 3, 4],
                stride = 1,
                dropout = 0.5,
                num_classes = 2, # Don't change this
```

```
                pad_idx = train_dataset.word2idx[PAD]) # Don't change this

    # Put your model on the device (cuda or cpu)
    cnn_model = cnn_model.to(DEVICE)

    print('The model has {:,d} trainable parameters'.
↪format(count_parameters(cnn_model)))
```

```
The model has 3,879,746 trainable parameters
```

Next, we create the **criterion**, which is our loss function: it is a measure of how well the model matches the empirical distribution of the data. We use cross-entropy loss (https://en.wikipedia.org/wiki/Cross_entropy).

We also define the **optimizer**, which performs gradient descent. We use the Adam optimizer (https://arxiv.org/pdf/1412.6980.pdf), which has been shown to work well on these types of models.

[15]:
```
import torch.optim as optim

if __name__=='__main__':
    LEARNING_RATE = 5e-4 # Feel free to try other learning rates

    # Define the loss function
    criterion = nn.CrossEntropyLoss().to(DEVICE)

    # Define the optimizer
    optimizer = optim.Adam(cnn_model.parameters(), lr=LEARNING_RATE)
```

Finally, we can train the model. If the model is implemented correctly and you're using the GPU, this cell should take around 4 minutes (or less). Feel free to change the number of epochs.

[16]:
```
if __name__=='__main__':
    N_EPOCHS = 15 # Feel free to change this == 20 == best

    # train model for N_EPOCHS epochs
    train_model(cnn_model, N_EPOCHS, train_loader, optimizer, criterion)
```

```
Training Model…

  0%|          | 0/15 [00:00<?, ?it/s]

[TRAIN]  Epoch:  1       Loss: 0.7578    Train Accuracy: 55.26%
[TRAIN]  Epoch:  2       Loss: 0.6062    Train Accuracy: 66.51%
[TRAIN]  Epoch:  3       Loss: 0.5491    Train Accuracy: 71.81%
[TRAIN]  Epoch:  4       Loss: 0.5133    Train Accuracy: 74.20%
[TRAIN]  Epoch:  5       Loss: 0.4814    Train Accuracy: 76.60%
[TRAIN]  Epoch:  6       Loss: 0.4547    Train Accuracy: 78.44%
[TRAIN]  Epoch:  7       Loss: 0.4303    Train Accuracy: 80.05%
[TRAIN]  Epoch:  8       Loss: 0.4119    Train Accuracy: 81.02%
[TRAIN]  Epoch:  9       Loss: 0.3920    Train Accuracy: 81.96%
```

```
[TRAIN]  Epoch: 10       Loss: 0.3737    Train Accuracy: 83.19%
[TRAIN]  Epoch: 11       Loss: 0.3541    Train Accuracy: 84.19%
[TRAIN]  Epoch: 12       Loss: 0.3310    Train Accuracy: 85.72%
[TRAIN]  Epoch: 13       Loss: 0.3104    Train Accuracy: 86.59%
[TRAIN]  Epoch: 14       Loss: 0.2951    Train Accuracy: 87.53%
[TRAIN]  Epoch: 15       Loss: 0.2684    Train Accuracy: 88.75%
Model Trained!
```

4.3 Evaluate CNN Model [20 points]

Now that we have trained a model for text classification, it is time to evaluate it. We have provided you with a function to do this; you do not need to modify anything.

To pass the autograder for the CNN, you will need to achieve **82% accuracy** on the hidden test set on Gradescope. Note that the Gradescope test set is very similar, and the accuracies between the two datasets should be comparable.

[17]:
```python
### DO NOT EDIT ###

import random

def evaluate(model, data_loader, criterion, use_tqdm=False):
    print('Evaluating performance on the test dataset...')
    model.eval()
    epoch_loss = 0
    epoch_acc = 0
    all_predictions = []
    print("\nSOME PREDICTIONS FROM THE MODEL:")
    iterator = tqdm(data_loader) if use_tqdm else data_loader
    total = 0
    for texts, labels in iterator:
        bs = texts.shape[0]
        total += bs
        texts = texts.to(DEVICE)
        labels = labels.to(DEVICE)

        output = model(texts)
        acc = accuracy(output, labels) * len(labels)
        pred = output.argmax(dim=1)
        all_predictions.append(pred)

        loss = criterion(output, labels) * len(labels)

        epoch_loss += loss.item()
        epoch_acc += acc.item()

        if random.random() < 0.0015 and bs == 1:
```

```
            print("Input: "+' '.join([data_loader.dataset.idx2word[idx] for idx␣
↪in texts[0].tolist() if idx not in {data_loader.dataset.word2idx[PAD],␣
↪data_loader.dataset.word2idx[END]}]))
            print("Prediction:", pred.item(), '\tCorrect Output:', labels.
↪item(), '\n')

    full_acc = 100*epoch_acc/total
    full_loss = epoch_loss/total
    print('[TEST]\t Loss: {:.4f}\t Accuracy: {:.2f}%'.format(full_loss,␣
↪full_acc))
    predictions = torch.cat(all_predictions)
    return predictions, full_acc, full_loss
```

[18]:
```
if __name__=='__main__':
    evaluate(cnn_model, test_loader, criterion, use_tqdm=True) # Compute test␣
↪data accuracy
    # pass
```

```
Evaluating performance on the test dataset…

SOME PREDICTIONS FROM THE MODEL:

  0%|          | 0/5000 [00:00<?, ?it/s]

Input: this is one of the best episode from the second season of <UNK> , i think
mick <UNK> has a problem with <UNK> . he <UNK> all , they are often the victims
( <UNK> solution , pro-life , valerie on the stairs , i don't remember the
argento's episode in season 1 , etc. , obviously <UNK> . i think he enjoys to
watch women been burn , torture , mutilated and i don't know . never least "
right to die " is one of the best , with good turns and graphic scenes and
suspense ( specially with the photos from the cell scene , <UNK> . the acting is
like the entire series , regular i could be worst like " pro-life " or " we
scream for ice <UNK> . also i think the plot it could be made for a movie and
not just for an episode . the ideology of the series is horrible , killing and
<UNK> women , <UNK> animals and on and on.. . the first season it was better
than the second one with episodes like " <UNK> burns " ( the best of all) , "
homecoming "
Prediction: 0   Correct Output: 1

Input: are you a <UNK> . ask john to nadia , and she , sure of responding well ,
responds him : yes . in this way begin the communication between a man and a
woman who don't know each other , and at the same time , the questions and
doubts in " birthday girl" . a film that i heard a lot of times , but i don't
dare to see.. . until two hours of write this.<br /><br <UNK> girl " is a
passionate movie that makes me fall in count , at the same time , that nicole
kidman is one of the best actress ( besides she is pretty and intelligent ) that
i have ever seen . " birthday girl " is the story of a lonely and routine man
who looks for a wife at internet . the woman that he finds comes from russia .
```

she seems to be that delicate woman , normal , not more . one day , in her birthday comes suddenly , his cousin and his friend . the man , begin to discover certain things . since here , he don't going to be the lonely and
Prediction: 1   Correct Output: 1

Input: first of all , don't go into revolver expecting another snatch or lock stock , this is a different sort of gangster film.

i saw the <UNK> the other night and this movie definitely split the audience . it's the kind of movie where half the audience will leave thinking what was that ? that was awful , and the other half will leave thinking what was that ? that was cool . personally i like films that i don't understand , <UNK> drive , and usual suspects , so i enjoyed revolver .

it definitely wasn't perfect though . i saw the big twist coming a mile away , at least part of it , and though sometimes some loose ends left unexplained is good , revolver leaves a lot of questions unexplained for no reason it seemed . also some scenes , like the animation , and the scene where <UNK> goes on a killing <UNK> one of my <UNK> , although , awesome scenes to watch , seemed to just be there because they were awesome to watch , not because they fit in with the movie.

however there were many good things
Prediction: 0   Correct Output: 1

Input: i saw this movie once as a kid on the <UNK> show and fell in love with it.

it took 30+ years , but i recently did find it on dvd - it wasn't cheap , either - in a catalog that specialized in war movies . we watched it last night for the first time . the audio was good , however it was grainy and had the trailers between reels . even so , it was better than i remembered it . i was also impressed at how true it was to the play.

the catalog is around here someplace . if you're sincere in finding it , fire me a <UNK> and i'll see if i can get you the info . <UNK>
Prediction: 1   Correct Output: 1

Input: this german documentary , in english , is about a scottish environmental sculptor named andy <UNK> . he makes art from objects he finds in nature . for example , early in the film we see him taking sections of <UNK> and " <UNK> " them together with a little <UNK> into a <UNK> shape that seems to repeatedly go through a <UNK> <UNK> />
of course , the <UNK> melt , but that <UNK> is a part of most of <UNK> work . he goes to a site and gets a feeling for it , deciding <UNK> what to make that day . he talks of having a " dialog " with the rocks and other materials that he works with , attempting to work <UNK> rather than against them . it might be stones , or flowers , or leaves , or sticks . the sculpture might last for minutes or years , or might not even last long enough to be completed and photographed . the work seems to be more of a process than a <UNK> />
the film , and the work , is beautiful , inspiring , and thought provoking . it moves pretty
Prediction: 1   Correct Output: 1

Input: all the world said that the film <UNK> would be a good movie with great pleasure , but this is not the case . vijay krishna <UNK> made a serious mistake

to take as an actress <UNK> kapoor . she was unbearable throughout the film . her <UNK> look does not really goes well . even the film the story of the film is not making sense at all . everyone said that the <UNK> <UNK> of india is vijay but its not at all <UNK> . the talent anil kapoor was involved in this stupid movie . anil is an actor of large caliber and this film is not . akshay kumar has also been a victim of this film as all is <UNK> . the style and the <UNK> is not really good in this film i was disappointed
Prediction: 0   Correct Output: 0

Input: flight of fury takes the mantle of being the very worst steven seagal flick i've ever <UNK> to now.

it's a dreadful bore with no action scenes of any interest , seagal isn't really trying in this - he's fat and his voice is dubbed once more.

the co-stars fare no better , being a rather sorry load of 3rd <UNK> />
the direction by <UNK> is very poor and it comes as no surprise that he's also responsible for another couple of seagal stinkers (shadow man & attack force) the screenplay co-written by seagal himself is laughably <UNK> />
according to imdb <UNK> was spent on this boring load of old tosh - if these figures are correct i sense a big tax fiddle as nowhere near that amount was <UNK> /><br <UNK> of fury is actually a shot for shot remake of the michael dudikoff flick black thunder - which has to be better than this tripe.

this has no redeeming qualities <UNK> it a miss ! 1/2 * out of *****
Prediction: 0   Correct Output: 0

Input: the minute you give an ' art film ' 1/10 , you have people <UNK> for your ignorant , <UNK> , artistically retarded blood . i won't try and justify how i am not an aesthetically challenged retard by listing out all the ' art house cinema ' i have liked or mentioning how i gave some unknown ' cult classic ' a 10/10 . all i ask is that someone explain to me the point , purpose and message of this film.

here is how i would summarize the film : opening montage of three unrelated urban legends depicting almost absurd levels of <UNK> . this followed by (in a nutshell , to save you 3 hours of pain) the following - a children's game show host dying of lung cancer tries to patch things up with his <UNK> daughter , who he may or may not have raped when she was a child , and who is being courted by a bumbling police officer with relationship issues , while the <UNK> star contestant decides that he doesn't want to be a failed child prodigy , a fate which has <UNK> another one of the game
Prediction: 0   Correct Output: 0

Input: this dumb comedy really does a good job of wasting comedic talent . in particular , dan aykroyd and howard hesseman are misused badly here . i might have chuckled once or twice during this film , but in general , it's a boring movie , with a little bit of stupidity thrown in for good measure . the premise , although routine , still wasn't bad , but once the plot was set , the film went nowhere . don't waste your time with this misfire .
Prediction: 0   Correct Output: 0

Input: the treasure island dvd should be required viewing in any film production

```
course ! it's a textbook example of how not to make a movie . watching the movie
and then listening to the <UNK> commentary demonstrates graphically the vast
<UNK> between what he knows about the characters and what he communicates to his
audience about them . call me old-fashioned , but i think of movies as a means
of communication , and communication isn't complete if the audience doesn't know
what the hell the director is talking about . the director's <UNK> purpose is to
make a movie void of " hollywood <UNK> . among those conventions , alas , is
consistency of character and clarity of concept . the director himself realizes
that audiences often don't understand points where he has purposely avoided a "
hollywood <UNK> . however , he never seems to grasp the idea that clichés exist
for a reason . they are shorthand for conveying complex ideas quickly and
clearly . it's fine to avoid them , but they need to be replaced with some other
way of communicating the same idea , not simply eliminated . the film is built
on an
Prediction: 0   Correct Output: 0

[TEST]   Loss: 0.3804    Accuracy: 83.66%
```

5 Step 4: Train a Recurrent Neural Network (RNN) [40 points]

You will now build a text clasification model that is based on **recurrences**.

5.1 TODO: Define the RNN Model [20 points]

First, you will define the RNN. As with the CNN, we provide you with the skeleton of the class, and you need to fill in parts of the `__init__(...)` and `forward(...)` methods. Each of these functions is worth 10 points.

[19]:
```
class RNN(nn.Module):
    def __init__(self, vocab_size, embed_size, hidden_size, num_layers,␣
 ↪bidirectional, dropout, num_classes, pad_idx):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        ##### TODO #####

        # Create an embedding layer (https://pytorch.org/docs/stable/generated/
 ↪torch.nn.Embedding.html)
        #   to represent the words in your vocabulary. Make sure to use␣
 ↪vocab_size, embed_size, and pad_idx here.
        self.embedding = nn.Embedding(vocab_size, embed_size,␣
 ↪padding_idx=pad_idx)
        self.embed_size = embed_size
        # Create a recurrent network (use nn.GRU, not nn.LSTM) with batch_first␣
 ↪= True
```

```
        # Make sure you use hidden_size, num_layers, dropout, and bidirectional␣
↪here.
        self.GRU = nn.GRU(input_size=embed_size, hidden_size=hidden_size,␣
↪num_layers=num_layers, batch_first=True, dropout=dropout,␣
↪bidirectional=bidirectional).to(DEVICE)


        if bidirectional==True:
            self.D =2
        elif bidirectional==False:
            self.D=1

        # self.initial_state_h0 = 0


        # Create a dropout layer (nn.Dropout) using dropout
        self.dropout_layer = nn.Dropout(dropout).to(DEVICE)

        # Define a linear layer (nn.Linear) that consists of num_classes units
        #   and takes as input the output of the last timestep. In the␣
↪bidirectional case, you should concatenate
        #   the output of the last timestep of the forward direction with the␣
↪output of the last timestep of the backward direction).

        self.dense_layer = nn.Linear(hidden_size*self.D, num_classes).to(DEVICE)



    def forward(self, texts):
        """
        texts: LongTensor [batch_size, MAX_LEN]

        Returns output: Tensor [batch_size, num_classes]
        """
        ##### TODO #####
        # device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        # Pass texts through your embedding layer to convert from word ids to␣
↪word embeddings
        #   Resulting: shape: [batch_size, max_len, embed_size]
        texts = texts.type(torch.int64)
        final_embedding = (self.embedding(texts))
        final_embedding_gpu = final_embedding.to(DEVICE)
        # print(f"  final_embedding_gpu: {final_embedding_gpu.shape},␣
↪final_embedding_gpu dtype: {final_embedding_gpu.dtype}, final_embedding_gpu␣
↪device: {final_embedding_gpu.get_device()}")
```

```
        # print('Content of embedding:', final_embedding)
        # print('Shape of embedding:', final_embedding.shape, '\n')
        # print('Type of embedding:', final_embedding.dtype, '\n')
        batch_size, max_len = texts.shape
        # print(f"batch_size: {batch_size}, max_len: {max_len}")
        # print(f"(batch_size, max_len, self.hidden_size): ({batch_size},
↪{max_len}, {self.hidden_size})")

        initial_state_h0 = torch.nn.parameter.Parameter(torch.randn(self.D*self.
↪num_layers, batch_size, self.hidden_size)).to(DEVICE)
        # print(f"  initial_state_h0: {initial_state_h0.shape},
↪initial_state_h0 dtype: {initial_state_h0.dtype}, initial_state_h0 device:
↪{initial_state_h0.get_device()}")

        # gru_input = torch.randn(batch_size, max_len, self.hidden_size).
↪to(device)
        # print(f"  gru_input: {gru_input.shape}, gru_input dtype: {gru_input.
↪dtype}, gru_input device: {gru_input.get_device()}")
        # # h_out = 32



        # Pass the result through your recurrent network
        #   See PyTorch documentation for resulting shape for nn.GRU
        output, hn = self.GRU(final_embedding_gpu, initial_state_h0)
        # print(f"  output: {output.shape}, output dtype: {output.dtype}")
        # print(f" hn: {hn.shape}, hn dtype: {hn.dtype}")


        # Concatenate the outputs of the last timestep for each direction (see
↪torch.cat(...))
        #   This depends on whether or not your model is bidirectional.
        #   Resulting shape: [batch_size, num_dirs*hidden_size]
        # concatenated_output = torch.cat([h for h in output], dim=0)
        concatenated_output = output[:, -1, :]
        # print(f"concatenated_output: {concatenated_output.shape},
↪concatenated_output dtype: {concatenated_output.dtype}")

        # batch_size: 1, max_len: 150
        # (batch_size, max_len, self.hidden_size): (1, 150, 32)
        #   gru_input: torch.Size([1, 150, 16]), gru_input dtype: torch.float32
        #   initial_state_h0: torch.Size([4, 1, 32]), initial_state_h0 dtype:
↪torch.float32
        #   output: torch.Size([1, 150, 64]), output dtype: torch.float32
        #  hn: torch.Size([4, 1, 32]), hn dtype: torch.float32
```

```
        #  concatenated_output: torch.Size([4, 32]), concatenated_output dtype:␣
↪torch.float32
        #  dropout: torch.Size([4, 32]), dropout dtype: torch.float32

        # Apply dropout
        dropout = self.dropout_layer(concatenated_output)
        # print(f" dropout: {dropout.shape}, dropout dtype: {dropout.dtype}")

        # Pass your output through the linear layer and return its output
        #   Resulting shape: [batch_size, num_classes]
        linear_output = self.dense_layer(dropout)
        # print(f" linear_output: {linear_output.shape}, linear_output dtype:␣
↪{linear_output.dtype}")


        ##### NOTE: Do not apply a sigmoid or softmax to the final output -␣
↪done in training method!


        return linear_output
```

##Sanity Check: RNN Model

The code below runs a sanity check for your `RNN` class. The tests are similar to the hidden ones in Gradescope. However, note that passing the sanity check does not guarantee that you will pass the autograder; it is intended to help you debug.

[20]:
```
### DO NOT EDIT ###

if __name__ == '__main__':
    # Test init
```

```
   inputs = [{'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,␣
↪'num_layers': 2, 'bidirectional': True, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,␣
↪'num_layers': 2, 'bidirectional': True, 'dropout': 0, 'num_classes': 4,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,␣
↪'num_layers': 2, 'bidirectional': False, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,␣
↪'num_layers': 2, 'bidirectional': False, 'dropout': 0, 'num_classes': 4,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,␣
↪'num_layers': 4, 'bidirectional': True, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,␣
↪'num_layers': 4, 'bidirectional': True, 'dropout': 0, 'num_classes': 4,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,␣
↪'num_layers': 4, 'bidirectional': False, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,␣
↪'num_layers': 4, 'bidirectional': False, 'dropout': 0, 'num_classes': 4,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 256,␣
↪'num_layers': 2, 'bidirectional': True, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 256,␣
↪'num_layers': 2, 'bidirectional': True, 'dropout': 0, 'num_classes': 4,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 256,␣
↪'num_layers': 2, 'bidirectional': False, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 256,␣
↪'num_layers': 2, 'bidirectional': False, 'dropout': 0, 'num_classes': 4,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 256,␣
↪'num_layers': 4, 'bidirectional': True, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 256,␣
↪'num_layers': 4, 'bidirectional': True, 'dropout': 0, 'num_classes': 4,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 256,␣
↪'num_layers': 4, 'bidirectional': False, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 256,␣
↪'num_layers': 4, 'bidirectional': False, 'dropout': 0, 'num_classes': 4,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 64, 'hidden_size': 32,␣
↪'num_layers': 2, 'bidirectional': True, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0}, {'vocab_size': 1000, 'embed_size': 64, 'hidden_size': 32,␣
↪'num_layers': 2, 'bidirectional': True, 'dropout': 0, 'num_classes': 4,␣
```

```
  expected_outputs = [44546, 44676, 27202, 27268, 82178, 82308, 39874, 39940,␣
↪1620610, 1621636, 621698, 622212, 3986050, 3987076, 1411202, 1411716,␣
↪101762, 101892, 79810, 79876, 139394, 139524, 92482, 92548, 1742338,␣
↪1743364, 706562, 707076, 4107778, 4108804, 1496066, 1496580]

  sanityCheckModel(inputs, RNN, expected_outputs, "init", None)
  print()

  # Test forward
```

```
   inputs = [{'vocab_size': 29730, 'embed_size': 16, 'hidden_size': 32,␣
↪'num_layers': 2, 'bidirectional': True, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0, 'batch_size': 1}, {'vocab_size': 29730, 'embed_size': 16,␣
↪'hidden_size': 32, 'num_layers': 2, 'bidirectional': True, 'dropout': 0,␣
↪'num_classes': 2, 'pad_idx': 0, 'batch_size': 2}, {'vocab_size': 29730,␣
↪'embed_size': 16, 'hidden_size': 32, 'num_layers': 2, 'bidirectional': True,␣
↪'dropout': 0, 'num_classes': 4, 'pad_idx': 0, 'batch_size': 1},␣
↪{'vocab_size': 29730, 'embed_size': 16, 'hidden_size': 32, 'num_layers': 2,␣
↪'bidirectional': True, 'dropout': 0, 'num_classes': 4, 'pad_idx': 0,␣
↪'batch_size': 2}, {'vocab_size': 29730, 'embed_size': 16, 'hidden_size': 32,␣
↪'num_layers': 2, 'bidirectional': False, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0, 'batch_size': 1}, {'vocab_size': 29730, 'embed_size': 16,␣
↪'hidden_size': 32, 'num_layers': 2, 'bidirectional': False, 'dropout': 0,␣
↪'num_classes': 2, 'pad_idx': 0, 'batch_size': 2}, {'vocab_size': 29730,␣
↪'embed_size': 16, 'hidden_size': 32, 'num_layers': 2, 'bidirectional':␣
↪False, 'dropout': 0, 'num_classes': 4, 'pad_idx': 0, 'batch_size': 1},␣
↪{'vocab_size': 29730, 'embed_size': 16, 'hidden_size': 32, 'num_layers': 2,␣
↪'bidirectional': False, 'dropout': 0, 'num_classes': 4, 'pad_idx': 0,␣
↪'batch_size': 2}, {'vocab_size': 29730, 'embed_size': 16, 'hidden_size': 32,␣
↪'num_layers': 4, 'bidirectional': True, 'dropout': 0, 'num_classes': 2,␣
↪'pad_idx': 0, 'batch_size': 1}, {'vocab_size': 29730, 'embed_size': 16,␣
↪'hidden_size': 32, 'num_layers': 4, 'bidirectional': True, 'dropout': 0,␣
↪'num_classes': 2, 'pad_idx': 0, 'batch_size': 2}, {'vocab_size': 29730,␣
↪'embed_size': 16, 'hidden_size': 32, 'num_layers': 4, 'bidirectional': True,␣
↪'dropout': 0, 'num_classes': 4, 'pad_idx': 0, 'batch_size': 1},␣
↪{'vocab_size': 29730, 'embed_size': 16, 'hidden_size': 32, 'num_layers': 4,␣
↪'bidirectional': True, 'dropout': 0, 'num_classes': 4, 'pad_idx': 0,␣
↪'batch_size': 2}, {'vocab_size': 29730, 'embed_size': 16, 'hidden_size': 32,␣
```

```
    expected_outputs = [torch.Size([1, 2]), torch.Size([2, 2]), torch.Size([1,␣
 ↪4]), torch.Size([2, 4]), torch.Size([1, 2]), torch.Size([2, 2]), torch.
 ↪Size([1, 4]), torch.Size([2, 4]), torch.Size([1, 2]), torch.Size([2, 2]),␣
 ↪torch.Size([1, 4]), torch.Size([2, 4]), torch.Size([1, 2]), torch.Size([2,␣
 ↪2]), torch.Size([1, 4]), torch.Size([2, 4]), torch.Size([1, 2]), torch.
 ↪Size([2, 2]), torch.Size([1, 4]), torch.Size([2, 4]), torch.Size([1, 2]),␣
 ↪torch.Size([2, 2]), torch.Size([1, 4]), torch.Size([2, 4]), torch.Size([1,␣
 ↪2]), torch.Size([2, 2]), torch.Size([1, 4]), torch.Size([2, 4]), torch.
 ↪Size([1, 2]), torch.Size([2, 2]), torch.Size([1, 4]), torch.Size([2, 4])]
    sanity_dataset = TextDataset(train_data, 'train', 5, 150)
    sanity_loader = torch.utils.data.DataLoader(sanity_dataset, batch_size=50,␣
 ↪shuffle=True, num_workers=2, drop_last=True)

    sanityCheckModel(inputs, RNN, expected_outputs, "forward", sanity_loader)
```

```
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,
'num_layers': 2, 'bidirectional': True, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}  Expected Num. Params: 44546     Your Num. Params: 44546
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,
'num_layers': 2, 'bidirectional': True, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0}  Expected Num. Params: 44676     Your Num. Params: 44676
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,
'num_layers': 2, 'bidirectional': False, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0} Expected Num. Params: 27202     Your Num. Params: 27202
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,
'num_layers': 2, 'bidirectional': False, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0} Expected Num. Params: 27268     Your Num. Params: 27268
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,
'num_layers': 4, 'bidirectional': True, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}  Expected Num. Params: 82178     Your Num. Params: 82178
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,
'num_layers': 4, 'bidirectional': True, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0}  Expected Num. Params: 82308     Your Num. Params: 82308
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,
'num_layers': 4, 'bidirectional': False, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0} Expected Num. Params: 39874     Your Num. Params: 39874
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size': 32,
'num_layers': 4, 'bidirectional': False, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0} Expected Num. Params: 39940     Your Num. Params: 39940
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size':
256, 'num_layers': 2, 'bidirectional': True, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0} Expected Num. Params: 1620610   Your Num. Params: 1620610
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size':
256, 'num_layers': 2, 'bidirectional': True, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0} Expected Num. Params: 1621636   Your Num. Params: 1621636
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size':
256, 'num_layers': 2, 'bidirectional': False, 'dropout': 0, 'num_classes': 2,
```

```
'pad_idx': 0}        Expected Num. Params: 621698    Your Num. Params: 621698
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size':
256, 'num_layers': 2, 'bidirectional': False, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0}        Expected Num. Params: 622212    Your Num. Params: 622212
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size':
256, 'num_layers': 4, 'bidirectional': True, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0} Expected Num. Params: 3986050   Your Num. Params: 3986050
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size':
256, 'num_layers': 4, 'bidirectional': True, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0} Expected Num. Params: 3987076   Your Num. Params: 3987076
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size':
256, 'num_layers': 4, 'bidirectional': False, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}        Expected Num. Params: 1411202   Your Num. Params: 1411202
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 16, 'hidden_size':
256, 'num_layers': 4, 'bidirectional': False, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0}        Expected Num. Params: 1411716   Your Num. Params: 1411716
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size': 32,
'num_layers': 2, 'bidirectional': True, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}  Expected Num. Params: 101762    Your Num. Params: 101762
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size': 32,
'num_layers': 2, 'bidirectional': True, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0}  Expected Num. Params: 101892    Your Num. Params: 101892
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size': 32,
'num_layers': 2, 'bidirectional': False, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0} Expected Num. Params: 79810     Your Num. Params: 79810
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size': 32,
'num_layers': 2, 'bidirectional': False, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0} Expected Num. Params: 79876     Your Num. Params: 79876
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size': 32,
'num_layers': 4, 'bidirectional': True, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}  Expected Num. Params: 139394    Your Num. Params: 139394
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size': 32,
'num_layers': 4, 'bidirectional': True, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0}  Expected Num. Params: 139524    Your Num. Params: 139524
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size': 32,
'num_layers': 4, 'bidirectional': False, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0} Expected Num. Params: 92482     Your Num. Params: 92482
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size': 32,
'num_layers': 4, 'bidirectional': False, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0} Expected Num. Params: 92548     Your Num. Params: 92548
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size':
256, 'num_layers': 2, 'bidirectional': True, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0} Expected Num. Params: 1742338   Your Num. Params: 1742338
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size':
256, 'num_layers': 2, 'bidirectional': True, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0} Expected Num. Params: 1743364   Your Num. Params: 1743364
       PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size':
256, 'num_layers': 2, 'bidirectional': False, 'dropout': 0, 'num_classes': 2,
```

```
'pad_idx': 0}        Expected Num. Params: 706562    Your Num. Params: 706562
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size':
256, 'num_layers': 2, 'bidirectional': False, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0}        Expected Num. Params: 707076    Your Num. Params: 707076
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size':
256, 'num_layers': 4, 'bidirectional': True, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0} Expected Num. Params: 4107778   Your Num. Params: 4107778
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size':
256, 'num_layers': 4, 'bidirectional': True, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0} Expected Num. Params: 4108804   Your Num. Params: 4108804
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size':
256, 'num_layers': 4, 'bidirectional': False, 'dropout': 0, 'num_classes': 2,
'pad_idx': 0}        Expected Num. Params: 1496066   Your Num. Params: 1496066
        PASSED  Input: {'vocab_size': 1000, 'embed_size': 64, 'hidden_size':
256, 'num_layers': 4, 'bidirectional': False, 'dropout': 0, 'num_classes': 4,
'pad_idx': 0}        Expected Num. Params: 1496580   Your Num. Params: 1496580

        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 32, 'num_layers': 2, 'bidirectional': True, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}   Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 32, 'num_layers': 2, 'bidirectional': True, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}   Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 2])       Your Output Shape:
torch.Size([2, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 32, 'num_layers': 2, 'bidirectional': True, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}   Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 4])       Your Output Shape:
torch.Size([1, 4])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 32, 'num_layers': 2, 'bidirectional': True, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}   Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 4])       Your Output Shape:
torch.Size([2, 4])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 32, 'num_layers': 2, 'bidirectional': False, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 32, 'num_layers': 2, 'bidirectional': False, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 2])       Your Output Shape:
torch.Size([2, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
```

```
'hidden_size': 32, 'num_layers': 2, 'bidirectional': False, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}  Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 4])       Your Output Shape:
torch.Size([1, 4])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 32, 'num_layers': 2, 'bidirectional': False, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}  Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 4])       Your Output Shape:
torch.Size([2, 4])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 32, 'num_layers': 4, 'bidirectional': True, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}   Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 32, 'num_layers': 4, 'bidirectional': True, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}   Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 2])       Your Output Shape:
torch.Size([2, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 32, 'num_layers': 4, 'bidirectional': True, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}   Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 4])       Your Output Shape:
torch.Size([1, 4])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 32, 'num_layers': 4, 'bidirectional': True, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}   Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 4])       Your Output Shape:
torch.Size([2, 4])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 32, 'num_layers': 4, 'bidirectional': False, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 32, 'num_layers': 4, 'bidirectional': False, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 2])       Your Output Shape:
torch.Size([2, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 32, 'num_layers': 4, 'bidirectional': False, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}  Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 4])       Your Output Shape:
torch.Size([1, 4])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 32, 'num_layers': 4, 'bidirectional': False, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}  Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 4])       Your Output Shape:
```

```
torch.Size([2, 4])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 2, 'bidirectional': True, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}   Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 2, 'bidirectional': True, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}   Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 2])       Your Output Shape:
torch.Size([2, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 2, 'bidirectional': True, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}   Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 4])       Your Output Shape:
torch.Size([1, 4])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 2, 'bidirectional': True, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}   Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 4])       Your Output Shape:
torch.Size([2, 4])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 2, 'bidirectional': False, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 2, 'bidirectional': False, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 2])       Your Output Shape:
torch.Size([2, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 2, 'bidirectional': False, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}  Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 4])       Your Output Shape:
torch.Size([1, 4])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 2, 'bidirectional': False, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}  Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 4])       Your Output Shape:
torch.Size([2, 4])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 4, 'bidirectional': True, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}   Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 4, 'bidirectional': True, 'dropout': 0,
```

```
'num_classes': 2, 'pad_idx': 0}   Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 2])       Your Output Shape:
torch.Size([2, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 4, 'bidirectional': True, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}   Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 4])       Your Output Shape:
torch.Size([1, 4])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 4, 'bidirectional': True, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}   Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 4])       Your Output Shape:
torch.Size([2, 4])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 4, 'bidirectional': False, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 2])       Your Output Shape:
torch.Size([1, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 4, 'bidirectional': False, 'dropout': 0,
'num_classes': 2, 'pad_idx': 0}  Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 2])       Your Output Shape:
torch.Size([2, 2])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 4, 'bidirectional': False, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}  Forward Input Shape: torch.Size([1, 150])
Expected Output Shape: torch.Size([1, 4])       Your Output Shape:
torch.Size([1, 4])
        PASSED   Init Input: {'vocab_size': 29730, 'embed_size': 16,
'hidden_size': 64, 'num_layers': 4, 'bidirectional': False, 'dropout': 0,
'num_classes': 4, 'pad_idx': 0}  Forward Input Shape: torch.Size([2, 150])
Expected Output Shape: torch.Size([2, 4])       Your Output Shape:
torch.Size([2, 4])
```

5.2 Train RNN Model

First, we initialize the train and test dataloaders.

[21]:
```
if __name__=='__main__':
    THRESHOLD = 5 # Don't change this
    MAX_LEN = 200 # Don't change this
    BATCH_SIZE = 64 # Feel free to try other batch sizes

    train_dataset = TextDataset(train_data, 'train', THRESHOLD, MAX_LEN)
    train_loader = torch.utils.data.DataLoader(train_dataset,␣
 ↪batch_size=BATCH_SIZE, shuffle=True, num_workers=2, drop_last=True)
```

```
    test_dataset = TextDataset(test_data, 'test', THRESHOLD, MAX_LEN,␣
↪train_dataset.idx2word, train_dataset.word2idx)
    test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=1,␣
↪shuffle=False, num_workers=1, drop_last=False)
```

Now you can instantiate your model. We provide you with some recommended hyperparameters; you should be able to get the desired accuracy with these, but feel free to play around with them.

[22]:
```
if __name__=='__main__':
    rnn_model = RNN(vocab_size = train_dataset.vocab_size, # Don't change this
                embed_size = 128,
                hidden_size = 128,
                num_layers = 2,
                bidirectional = True,
                dropout = 0.5,
                num_classes = 2, # Don't change this
                pad_idx = train_dataset.word2idx[PAD]) # Don't change this

    # Put your model on device
    rnn_model = rnn_model.to(DEVICE)

    print('The model has {:,d} trainable parameters'.
↪format(count_parameters(rnn_model)))
```

```
The model has 4,300,546 trainable parameters
```

Here, we create the criterion and optimizer; as with the CNN, we use cross-entropy loss and Adam optimization.

[23]:
```
if __name__=='__main__':
    LEARNING_RATE = 6e-4 # Feel free to try other learning rates

    # Define your loss function
    criterion = nn.CrossEntropyLoss().to(DEVICE)

    # Define your optimizer
    optimizer = optim.Adam(rnn_model.parameters(), lr=LEARNING_RATE)
```

Finally, we can train the model. We use the same `train_model(...)` function that we defined for the CNN. If the model is implemented correctly and you're using the GPU, this cell should take around 2 minutes (or less). Feel free to change the number of epochs.

[24]:
```
if __name__=='__main__':
    N_EPOCHS = 8 # Feel free to change this

    # train model for N_EPOCHS epochs
    train_model(rnn_model, N_EPOCHS, train_loader, optimizer, criterion)
```

```
Training Model…
```

```
  0%|          | 0/8 [00:00<?, ?it/s]

[TRAIN]  Epoch:  1       Loss: 0.7068    Train Accuracy: 50.82%
[TRAIN]  Epoch:  2       Loss: 0.6818    Train Accuracy: 56.39%
[TRAIN]  Epoch:  3       Loss: 0.6525    Train Accuracy: 62.28%
[TRAIN]  Epoch:  4       Loss: 0.5990    Train Accuracy: 66.71%
[TRAIN]  Epoch:  5       Loss: 0.3924    Train Accuracy: 82.88%
[TRAIN]  Epoch:  6       Loss: 0.2920    Train Accuracy: 87.97%
[TRAIN]  Epoch:  7       Loss: 0.2179    Train Accuracy: 91.70%
[TRAIN]  Epoch:  8       Loss: 0.1664    Train Accuracy: 93.95%
Model Trained!
```

5.3 Evaluate RNN Model [20 points]

Now we can evaluate the RNN.

To pass the autograder for the RNN, you will need to achieve **82% accuracy** on the hidden test set on Gradescope. Note that the Gradescope test set is very similar, and the accuracies between the two datasets should be comparable.

[25]:
```
if __name__=='__main__':
    evaluate(rnn_model, test_loader, criterion, use_tqdm=True) # Compute test␣
↪data accuracy
```

```
Evaluating performance on the test dataset…

SOME PREDICTIONS FROM THE MODEL:

  0%|          | 0/5000 [00:00<?, ?it/s]

Input: an unusual film from ringo lam and one that's strangely under-appreciated
. the mix of fantasy kung-fu with a more realistic depiction of swords and
spears being driven thru bodies is startling especially during the first ten
minutes . a horseback rider get chopped in two and his waist and legs keep
riding the horse . several horses get chopped up . it's very <UNK> /><br />the
story is very simple , fong and his shaolin brothers are captured by a crazed
maniac general and imprisoned in the red <UNK> temple which seems to be more of
a torture chamber then a temple . the general has a similarity to kurtz in
apocalypse now as he spouts warped philosophy and makes frightening paintings
with human blood . <br /><br />the production is very impressive and the setting
is bleak . blood is everywhere . the action is very well done and mostly
coherent unlike many hk action scenes from the time . sometimes the movie veers
into absurdity or the effects are cheesy but it's never bad enough to ruin the
film . <br /><br <UNK> this one , it's one of the best hk kung fu films from the
Prediction: 1   Correct Output: 1

Input: fairly funny jim carrey vehicle that has him as a news reporter who
temporarily gets the power of god and wrecks havoc . carrey is back in familiar
ground here and looks to be having a good time , and jennifer aniston as his put
```

upon girlfriend is also charming and affecting . the story is <UNK> to the extreme but the cast (including morgan freeman as " <UNK>) is great and makes the film worth catching . grade : b
Prediction: 1   Correct Output: 1

Input: unique movie about confused woman (lindsay <UNK>) who gets involved with sharp con men . joe mantegna gives an <UNK> performance as the <UNK> of the group . absolutely enchanting first hour , as mantegna shows <UNK> " the ropes " of his con games . story line unravels a bit later on , but still stands as a unique portrayal of an innocent caught up in a dark world . definitely worth a shot .
Prediction: 1   Correct Output: 1

Input: i found the film quite expressive , the way the main character was lost but at the same much more clear about certain things in life than people who mocked him (his <UNK> for example) .

he was tortured and you loved to watch him being tortured ! it had this perverted side which was frightening but we were all happy to see him come out of the misery again .

it was like a game character or <UNK> through a <UNK> or to enemy and we love to watch him under sniper attack or fire but then at the end we are happy to see him survive .. .

 .
Prediction: 1   Correct Output: 1

Input: stranger than fiction angered me so much , i signed up on imdb just to write this review . stranger than fiction is a surprisingly complex , touching and thought-provoking movie until the very end . once you suspend multiple lapses of logic (why didn't will ferrell hear emma thompson's voice 10 years ago when she fist started writing her book ? " the phone rang . the phone rang again. " how could she not know it's him calling ? etc.) , the movie challenges one's thoughts about mortality , fate , and <UNK> />
the brief history of literary themes provided by dustin hoffman should especially entertain former english majors . and maggie gyllenhaal is always a pleasure , even though will ferrell might just as easily be an ax murderer as a bumbling soul . her quick trust of him is a mighty big leap of <UNK> />
ah , but the ending . until the very end , i would have given 9 out of 10 stars to this movie . the movie as a metaphor for life's journey , as a tribute to the notion of ' writing <UNK> ' as a reminder that
Prediction: 1   Correct Output: 0

Input: being a filmmaker myself , and possessing a somewhat dark and subversive sense of humour , i thought i was in for a treat when i took home " my <UNK> " (not that the dvd cover gives anything away , instead opting for the ambiguous quote from <UNK> director , chris morris , " a short film including <UNK> . i should have known better really , and avoided this insipid (and often offensive) piece of <UNK> />
the scene in the church is repulsive to watch (especially since we are all too familiar with <UNK> warped attitude towards <UNK> from his notorious tv series , brass eye) and serves no purpose other than to shock . how this film is labelled a comedy i will never understand.<br

```
/><br />the <UNK> commentary sounds like a novel idea in principle ( having been
a runner myself , it's often an interesting and uninhibited perspective on the
filmmaking <UNK> , however this is sadly not the case here . instead , we are
treated to some public schoolboy ranting about dogs on film , before concluding
that there are no really great movies starring dogs .
Prediction: 0   Correct Output: 0

[TEST]   Loss: 0.5039    Accuracy: 83.22%
```

6 What to Submit

To submit the assignment, download this notebook as a .py file. You can do this by going to File > Download > Download .py. Then (optionally) rename it to `hwk2.py`.

You will also need to save the `cnn_model` and `rnn_model`. You can run the cell below to do this. After you save the files to your Google Drive, you need to manually download the files to your computer, and then submit them to the autograder.

You will submit the following files to the autograder: 1. `hwk2.py`, the download of this notebook as a `.py` file (**not** a `.ipynb` file) 1. `cnn.pt`, the saved version of your `cnn_model` 1. `rnn.pt`, the saved version of your `rnn_model`

[26]:
```python
### DO NOT EDIT ###

if __name__=='__main__':
    # from google.colab import drive
    # drive.mount('/content/drive')
    print()

    try:
        cnn_model is None
        cnn_exists = True
    except:
        cnn_exists = False

    try:
        rnn_model is None
        rnn_exists = True
    except:
        rnn_exists = False

    if cnn_exists:
        print("Saving CNN model....")
        # torch.save(cnn_model, "drive/My Drive/cnn.pt")
        torch.save(cnn_model, "saved_models/cnn.pt")
    if rnn_exists:
        print("Saving RNN model....")
        # torch.save(rnn_model, "drive/My Drive/rnn.pt")
```

```
        torch.save(rnn_model, "saved_models/rnn.pt")
    print("Done!")
```

```
Saving CNN model…
Saving RNN model…
Done!
```

[26]:

[26]:

[26]: