Part 0. Google Colab Setup

Hopefully you're looking at this notebook in Colab!

1. First, make a copy of this notebook to your local drive, so you can edit it.
2. Go ahead and upload the OnionOrNot.csv file from the assignment zip (https://www.cc.gatech.edu/classes/AY2022/cs4650_fall/programming/h2_torch.zip) in the files panel on the left.
3. Right click in the files panel, and select 'Create New Folder' - call this folder src
4. Upload all the files in the src/ folder from the assignment zip (https://www.cc.gatech.edu/classes/AY2022/cs4650_fall/programming/h2_torch.zip) to the src/ folder on colab.

NOTE: REMEMBER TO REGULARLY REDOWNLOAD ALL THE FILES IN SRC FROM COLAB.

IF YOU EDIT THE FILES IN COLAB, AND YOU DO NOT REDOWNLOAD THEM, YOU WILL LOSE YOUR WORK!

If you want GPU's, you can always change your instance type to GPU directly in Colab.

Part 1. Loading and Preprocessing Data [10 points]

The following cell loads the OnionOrNot dataset, and tokenizes each data item

In [ ]:
```
# DO NOT MODIFY #
import torch
import random
import numpy as np
RANDOM_SEED = 42
torch.manual_seed(RANDOM_SEED)
random.seed(RANDOM_SEED)
np.random.seed(RANDOM_SEED)
# this is how we select a GPU if it's avalible on your computer.
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [ ]:
```
import pandas as pd
from src.preprocess import clean_text
import nltk
from tqdm import tqdm

nltk.download('punkt')
df = pd.read_csv("OnionOrNot.csv")
df["tokenized"] = df["text"].apply(lambda x: nltk.word_tokenize(clean_text(x.lower())))
```

```
[nltk_data] Downloading package punkt to /home/andre/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Here's what the dataset looks like. You can index into specific rows with pandas, and try to guess some of these yourself :)

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

Out[ ]:

	text	label	tokenized
0	Entire Facebook Staff Laughs As Man Tightens P...	1	[entire, facebook, staff, laughs, as, man, tig...
1	Muslim Woman Denied Soda Can for Fear She Coul...	0	[muslim, woman, denied, soda, can, for, fear, ...
2	Bold Move: Hulu Has Announced That They're Gon...	1	[bold, move, :, hulu, has, announced, that, th...
3	Despondent Jeff Bezos Realizes He'll Have To W...	1	[despondent, jeff, bezos, realizes, he, ', ll,...
4	For men looking for great single women, online...	1	[for, men, looking, for, great, single, women,...

In [ ]:
```
df.iloc[42]
```

Out[ ]:
```
text         Customers continued to wait at drive-thru even...
label                                                        0
tokenized    [customers, continued, to, wait, at, drive-thr...
Name: 42, dtype: object
```

Now that we've loaded this dataset, we need to split the data into train, validation, and test sets. We also need to create a vocab map for words in our Onion dataset, which will map tokens to numbers. This will be useful later, since torch models can only use tensors of sequences of numbers as inputs. **Go to src/dataset.py, and fill out split_train_val_test, generate_vocab_map**

In [ ]:
```
## TODO: complete these methods in src/dataset.py
from src.dataset import split_train_val_test, generate_vocab_map
df = df.sample(frac=1)

train_df, val_df, test_df = split_train_val_test(df, props=[.8, .1, .1])
train_vocab, reverse_vocab = generate_vocab_map(train_df)
```

In [ ]:
```
# this line of code will help test your implementation
(len(train_df) / len(df)), (len(val_df) / len(df)), (len(test_df) / len(df))
```

Out[ ]:
```
(0.8, 0.1, 0.1)
```

PyTorch has custom Datset Classes that have very useful extentions. **Go to src/dataset.py, and fill out the HeadlineDataset class.** Refer to PyTorch documentation on Dataset Classes for help.

In [ ]:
```
from src.dataset import HeadlineDataset
from torch.utils.data import RandomSampler
#print(train_df)

train_dataset = HeadlineDataset(train_vocab, train_df)
val_dataset = HeadlineDataset(train_vocab, val_df)
test_dataset = HeadlineDataset(train_vocab, test_df)

# Now that we're wrapping our dataframes in PyTorch datsets, we c
an make use of PyTorch Random Samplers.
train_sampler = RandomSampler(train_dataset)
val_sampler = RandomSampler(val_dataset)
test_sampler = RandomSampler(test_dataset)
```

We can now use PyTorch DataLoaders to batch our data for us. **Go to src/dataset.py, and fill out collate_fn.** Refer to PyTorch documentation on Dataloaders for help.

In [ ]:
```
from torch.utils.data import DataLoader
from src.dataset import collate_fn
BATCH_SIZE = 16
train_iterator = DataLoader(train_dataset, batch_size=BATCH_SIZE,
sampler=train_sampler, collate_fn=collate_fn)
val_iterator = DataLoader(val_dataset, batch_size=BATCH_SIZE, sam
pler=val_sampler, collate_fn=collate_fn)
test_iterator = DataLoader(test_dataset, batch_size=BATCH_SIZE, s
ampler=test_sampler, collate_fn=collate_fn)
```

In [ ]:
```
# # Use this to test your collate_fn implementation.

# # You can look at the shapes of x and y or put print
# # statements in collate_fn while running this snippet

for x, y in test_iterator:
    print(x,y)
    break
test_iterator = DataLoader(test_dataset, batch_size=BATCH_SIZE, s
ampler=test_sampler, collate_fn=collate_fn)
```

```
tensor([[ 174, 2903, 9462,    1,  321,  563,   36, 2201,    1, 18
44, 2854,   12,
          223,    1,   33, 7472,    0,    0,    0,    0,    0],
        [  11, 1085,  828, 6476,  411,    1,   36,   37,   51,  1
58, 3298,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0],
        [   1, 2698, 8440,   33,  321,   33,  274,   52,    1,
41,  617, 4959,
           12, 1016,    0,    0,    0,    0,    0,    0,    0],
        [  25,  882,  289,  552,  301,   61,  181,  593,  289, 27
74, 2775,   36,
         4604, 5445,   36,  113,    1,    1, 1094,   52,    3],
        [3472, 4372, 1331,    1, 2430,   52,   57, 8872,    1,
0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0],
        [4223,  120,   15,  431,    1,  217, 5502,   24,    1,  6
08,   52,    1,
          925,    0,    0,    0,    0,    0,    0,    0,    0],
        [3019,   61, 3210, 2774,    1, 6016, 6635, 4447,    0,
0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0],
        [3123, 4183, 3475,  217, 5632, 8439,   52, 8594, 5089,
21,  605,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0],
        [3283,  411,   61, 2461,  158,  476, 3157,  939, 3310, 21
27, 6876,  411,
           33,  343,  327,   21, 7367,  301,    0,    0,    0],
        [  68,  147, 2389,   33, 7045,  951, 8886,    0,    0,
0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0],
        [1641, 5892,   61, 5897,  976, 8704,  133,  263,    1,
1,   56,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0],
        [   1, 2710,    1,  165,    1,    0,    0,    0,    0,
0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0],
        [  11,   21, 1021, 6181,  619,  148,  149, 3427,  128, 38
79,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0],
        [5753, 4651,  321,    1,    1,   21, 1651,    0,    0,
0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0],
        [7449, 6726,  770, 5105, 3296,   36,   81,  783,    1,
1, 6870,  783,
          326, 4127,   30,    0,    0,    0,    0,    0,    0],
        [1280,   36, 6891, 3256,    1, 1752, 1010,   24, 2486, 46
71,   30,   31,
         2121,  976,    1, 5917,    1,    0,    0,    0,    0]])
tensor([1., 0., 1., 1., 0., 0., 1., 0., 0., 0., 0., 1., 1., 1.,
1., 1.])
```

Part 2: Modeling [10 pts]

Let's move to modeling, now that we have dataset iterators that batch our data for us. **Go to src/model.py, and follow the instructions in the file to create a basic neural network. Then, create your model using the class, and define hyperparameters.**

In [ ]:
```
from src.models import ClassificationModel
model = None
### YOUR CODE GOES HERE (1 line of code) ###
model = ClassificationModel(len(train_vocab),embedding_dim=32,hid
den_dim = 32,num_layers = 1,bidirectional = True)

# model.to(device)
# #
### YOUR CODE ENDS HERE ###
```

In the following cell, **instantiate the model with some hyperparameters, and select an appropriate loss function and optimizer.**

Hint: we already use sigmoid in our model. What loss functions are availible for binary classification? Feel free to look at PyTorch docs for help!

In [ ]:
```
from torch.optim import AdamW

criterion, optimizer = None, None
### YOUR CODE GOES HERE ###
criterion, optimizer = torch.nn.BCEWithLogitsLoss(), torch.optim.
AdamW(model.parameters(), lr=0.01)
# scheduler = torch.optim.lr_scheduler.StepLR(optimizer, 1, gamma
=0.9)

### YOUR CODE ENDS HERE ###
```

Part 3: Training and Evaluation [10 Points]

The final part of this HW involves training the model, and evaluating it at each epoch. **Fill out the train and test loops below.**

In [ ]:
```
# returns the total loss calculated from criterion
def train_loop(model, criterion, iterator):
    model.train()
    total_loss = 0

    for x, y in tqdm(iterator):
        optimizer.zero_grad()
        # x = x.to(device)
        # y = y.to(device)
        ### YOUR CODE STARTS HERE (~6 lines of code) ###
        prediction = model(x)
        prediction = torch.squeeze(prediction,0)
        y = y.unsqueeze(1)
        y = y.round()


        loss = criterion(prediction,y)
        total_loss += loss.item()
        loss.backward()
        optimizer.step()
    # scheduler.step()
        ### YOUR CODE ENDS HERE ###
    return total_loss

# returns:
# - true: a Python boolean array of all the ground truth values
#         taken from the dataset iterator
# - pred: a Python boolean array of all model predictions.
def val_loop(model, criterion, iterator):
    true, pred = [], []
    ### YOUR CODE STARTS HERE (~8 lines of code) ###
    for x, y in tqdm(iterator):
        # x = x.to(device)
        # y = y.to(device)
        # print("x",x)
        # print("y",y)

        preds = model(x)
        preds = torch.flatten(preds)
        for i_batch in range(len(y)):
            true.append(y[i_batch])
            pred.append(torch.round(preds[i_batch]))




    ### YOUR CODE ENDS HERE ###
    return true, pred
```

We also need evaluation metrics that tell us how well our model is doing on the validation set at each epoch. **Complete the functions in src/eval.py.**

In [ ]:
```
# To test your eval implementation, let's see how well the untrai
ned model does on our dev dataset.
# It should do pretty poorly.
from src.eval_utils import binary_macro_f1, accuracy
true, pred = val_loop(model, criterion, val_iterator)
print(binary_macro_f1(true, pred))
print(accuracy(true, pred))
```

```
100%|██████████| 150/150 [00:02<00:00, 69.12it/s]

0.4216917201986171
0.42791666666666667
```

Part 4: Actually training the model [1 point]

Watch your model train :D You should be able to achieve a validation F-1 score of at least .8 if everything went correctly. **Feel free to adjust the number of epochs to prevent overfitting or underfitting.**

In [ ]:

```
TOTAL_EPOCHS = 7
for epoch in range(TOTAL_EPOCHS):
    train_loss = train_loop(model, criterion, train_iterator)
    true, pred = val_loop(model, criterion, val_iterator)
    print(f"EPOCH: {epoch}")
    print(f"TRAIN LOSS: {train_loss}")
    print(f"VAL F-1: {binary_macro_f1(true, pred)}")
    print(f"VAL ACC: {accuracy(true, pred)}")
```

```
100%|██████████| 1200/1200 [01:35<00:00, 12.61it/s]
100%|██████████| 150/150 [00:02<00:00, 73.79it/s]

EPOCH: 0
TRAIN LOSS: 775.4171956777573
VAL F-1: 0.7974621640060507
VAL ACC: 0.8170833333333334

100%|██████████| 1200/1200 [01:34<00:00, 12.68it/s]
100%|██████████| 150/150 [00:01<00:00, 79.89it/s]

EPOCH: 1
TRAIN LOSS: 723.8004207611084
VAL F-1: 0.8273450802293716
VAL ACC: 0.8391666666666666

100%|██████████| 1200/1200 [01:34<00:00, 12.65it/s]
100%|██████████| 150/150 [00:01<00:00, 85.69it/s]

EPOCH: 2
TRAIN LOSS: 706.5095884203911
VAL F-1: 0.8122257420283887
VAL ACC: 0.8304166666666667

100%|██████████| 1200/1200 [01:34<00:00, 12.72it/s]
100%|██████████| 150/150 [00:01<00:00, 81.95it/s]

EPOCH: 3
TRAIN LOSS: 698.9961122572422
VAL F-1: 0.8362141298875652
VAL ACC: 0.8475

100%|██████████| 1200/1200 [01:34<00:00, 12.67it/s]
100%|██████████| 150/150 [00:01<00:00, 82.50it/s]

EPOCH: 4
TRAIN LOSS: 693.1515847146511
VAL F-1: 0.8448174961697538
VAL ACC: 0.8525

100%|██████████| 1200/1200 [01:39<00:00, 12.03it/s]
100%|██████████| 150/150 [00:01<00:00, 78.85it/s]

EPOCH: 5
TRAIN LOSS: 689.8829775452614
VAL F-1: 0.8387900771050092
VAL ACC: 0.8495833333333334

100%|██████████| 1200/1200 [01:42<00:00, 11.65it/s]
100%|██████████| 150/150 [00:01<00:00, 77.04it/s]

EPOCH: 6
TRAIN LOSS: 686.4265978038311
VAL F-1: 0.8449919692293513
VAL ACC: 0.8545833333333334
```

We can also look at the models performance on the held-out test set, using the same val_loop we wrote earlier.

In [ ]:
```
true, pred = val_loop(model, criterion, test_iterator)
print(f"TEST F-1: {binary_macro_f1(true, pred)}")
print(f"TEST ACC: {accuracy(true, pred)}")
```

```
100%|██████████| 150/150 [00:01<00:00, 77.23it/s]

TEST F-1: 0.8448861905182727
TEST ACC: 0.8570833333333333
```

Part 5: Analysis [5 points]

Answer the following questions:

1. What happens to the vocab size as you change the cutoff in the cell below? Can you explain this in the context of Zipf's Law (https://en.wikipedia.org/wiki/Zipf%27s_law)?

In [ ]:
```
tmp_vocab, _ = generate_vocab_map(train_df, cutoff = 0)
print(len(tmp_vocab))
"""
The cutoff discriminates which words are going to be part of our vocabulary looking at the number of appearances in our training data.
Setting it to a value of 1 means that all the words that appear more than 0 times enter in our vocabulary.
The progression we experiment is:
cutoff len
------ ---
1      13298
2       9540
3       7612
4       6340
5       5476
6       4825
7       4296
8       3870
9       3590
We see that the number of words in our vocabulary decreases in what it seems a logarithmic progression.
Zipf's Law states that the frequency of any word is inversely proportional to the rank of the word in the freq table.
That means the most frequent words are always at the top of the table.
With the cutoff, what we are doing is removing the words with the less frequency. The most frequent words will have very different frequencies from one
another but the words at the bottom of the table will share their frequencies and they will be many. By increasing the cutoff, we remove each time less and
less words as there will be less words that share their frequencies.
"""
```

```
25387
```

Out[ ]:
```
"\nThe cutoff discriminates which words are going to be part of our vocabulary looking at the number of appearances in our training data.\nSetting it to a value of 1 means that all the words that appear more than 0 times enter in our vocabulary.\nThe progression we experiment is:\ncutoff len\n------ ---\n1      13298 \n2       9540\n3       7612\n4       6340\n5       5476\n6       4825\n7       4296\n8       3870\n9       3590\nWe see that the number of words in our vocabulary decreases in what it seems a logarithmic progression.\nZipf's Law states that the frequency of any word is inversely proportional to the rank of the word in the freq table.\nThat means the most frequent words are always at the top of the table.\nWith the cutoff, what we are doing is removing the words with the less frequency. The most frequent words will have very different frequencies from one\nanother but the words at the bottom of the table will share their frequencies and they will be many. By increasing the cutoff, we remove each time less and\nless words as there will be less words that share their frequencies.\n"
```

2. Can you describe what cases the model is getting wrong in the witheld test-set?

To do this, you'll need to create a new val_train_loop (`val_train_loop_incorrect`) so it returns incorrect sequences **and** you'll need to decode these sequences back into words. Thankfully, you've already created a map that can convert encoded sequences back to regular English: you will find the `reverse_vocab` variable useful.

```
# i.e. using a reversed map of {"hi": 2, "hello": 3, "UNK": 0}
# we can turn [1, 2, 0] into this => ["hi", "hello", "UNK"]
```

In [ ]:

```
# Implement this however you like! It should look very similar to
val_loop.
# Pass the test_iterator through this function to look at errors
in the test set.
def val_train_loop_incorrect(model, iterator):
    for x, y in tqdm(iterator):
        # x.to(device)
        # y.to(device)

        errors = []

        preds = model(x)
        preds = torch.flatten(preds)

        for i_batch in range(len(y)):
            if y[i_batch] != preds[i_batch]:
                sentence = []
                for word in range(len(x[i_batch])):
                    sentence.append(reverse_vocab[x[i_batch][wor
d].item()])
                errors.append(sentence)
        return errors



errors = val_train_loop_incorrect(model,test_iterator)
for sentence in errors:
    print(sentence)
```

```
  0%|          | 0/150 [00:00<?, ?it/s]

['us', 'airways', 'tweets', 'UNK', 'UNK', 'UNK', 'at', 'angry', '
customer', '', '', '', '', '', '', '', '', '', '', '', '', '',
'', '']
['UNK', 'replaces', 'UNK', 'coin', 'UNK', 'with', 'beer', 'taps',
',', 'UNK', 'winners', 'with', 'brew', '', '', '', '', '', '',
'', '', '', '', '']
['beloved', 'honorary', 'cat', 'mayor', 'in', 'small', 'alaska',
'town', 'dies', 'at', '20', '', '', '', '', '', '', '', '', '',
'', '', '', '']
['eagles', 'fans', 'finally', 'sober', 'enough', 'to', 'return',
'to', 'work', '', '', '', '', '', '', '', '', '', '', '', '', '',
'', '']
['news', ':', 'finally', ':', 'the', 'indians', 'are', 'replacing
', 'their', 'racist', 'mascot', 'chief', 'UNK', 'with', 'a', 'whi
te', 'woman', 'wearing', 'a', 'native', 'american', 'halloween',
'costume', '']
['these', 'brave', 'teens', 'went', 'UNK', 'for', '3', 'whole', '
days', 'and', 'miraculously', 'survived', '', '', '', '', '', '',
'', '', '', '', '', '']
['lou', 'UNK', 'blames', 'stanford', '’', 's', 'loss', 'to', 'uta
h', 'on', 'UNK', '', '', '', '', '', '', '', '', '', '', '', '',
'']
['mayor', 'rob', 'ford', 'calls', 'in', 'sick', 'on', 'bob', 'UNK
', 'day', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
['amsterdam', 'to', 'fly', 'rainbow', 'flag', 'for', 'russian', '
president', 'putin', '’', 's', 'visit', 'to', 'the', 'capital',
'', '', '', '', '', '', '', '', '']
['four', 'UNK', 'took', 'cocaine', 'thinking', 'it', 'was', 'UNK
', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '',
'']
['for', 'the', 'first', 'time', 'in', 'saudi', 'arabia', ',', 'wo
men', 'UNK', 'to', 'issue', 'UNK', '', '', '', '', '', '', '',
'', '', '', '']
['boy', 'suspended', 'over', 'UNK', 'gun', 'seeks', 'to', 'clear
', 'school', 'record', '', '', '', '', '', '', '', '', '', '',
'', '', '', '']
['‘', 'nothing', 'would', 'surprise', 'me', 'at', 'this', 'point
', ',', '’', 'says', 'man', 'who', 'will', 'be', 'shocked', 'by',
'8', 'separate', 'news', 'items', 'today', '', '']
['why', 'north', 'korea', "'s", 'capital', 'is', 'the', 'UNK', 's
cience', 'fiction', 'film', 'set', "'", '', '', '', '', '', '',
'', '', '', '', '']
```