Assignment 3 - Part 2

In this assignment, you are to architect a neural network to perform text classification.

Grading scheme:

- Model: 10
- Training: 10
- Test accuracy: 20

```
>= 70%: 5/10
>= 75%: 10/20
>= 80%: 15/20
>= 85%: 20/20
```

Total: 40

In [5]:
```
"🔒"
import torch
from torch import (nn, optim)
from torch.utils.data import (Dataset, DataLoader, random_split)
from torchsummaryX import summary
import pandas as pd
import numpy as np
import warnings
import test_lib
from importlib import reload
reload(test_lib)
warnings.filterwarnings('ignore')
```

Load dataset and vocabulary from file

You are given two dataset files for training and testing.

In [6]:
```
"🔒"
train_dataset = torch.load('./train_dataset.npz')
test_dataset = torch.load('./test_dataset.npz')

print("Training dataset: %d" % len(train_dataset))
print("Test dataset: %d" % len(test_dataset))
```

```
Training dataset: 20000
Test dataset: 5000
```

Load the vocabulary

You are given the vocabulary file. This is used **only** for decoding the integers in the dataset. It is not used for training, nor testing.

In [7]:
```
"🔒"
vocab = torch.load('./vocab.pt')
print("There are %d tokens in vocabulary." % len(vocab))
```

```
There are 2000 tokens in vocabulary.
```

Check data

In [8]:
```
"🔒"
# @check
# @title: data integrity

x, y = train_dataset[100]
print("Review:", " ".join(vocab.lookup_tokens(x.numpy().tolist())))
print("Label:", y.item())
```

```
Review: one of the very best three <unk> <unk> ever . a <unk> house full of evil g
uys and the <unk> <unk> the <unk> detective <unk> s best men . <unk> is in top for
m in the famous in the dark scene . <unk> <unk> provides excellent support in his
mr . <unk> role as the <unk> of a murder plot . before it s over <unk> s <unk> lit
tle <unk> is <unk> to great effect . this minute gem moves about as fast as any <u
nk> s short and <unk> twice the <unk> . highly recommended . <pad> <pad> <pad>
Label: 1
```

Model

Construct a model that can process and learn from the samples from dataset.

Hints: Learn advanced architectural layers:

- LSTM: https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html
- Conv1D: https://pytorch.org/docs/stable/generated/torch.nn.Conv1d.html
- MaxPool1d: https://pytorch.org/docs/stable/generated/torch.nn.MaxPool1d.html
- Dropout: https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html

In [9]:
```
"✍"
# @workUnit

train_dataset = torch.load('./train_dataset_extended.npz')
test_dataset = torch.load('./test_dataset_extended.npz')


print("Training dataset: %d" % len(train_dataset))
print("Test dataset: %d" % len(test_dataset))

x, y = train_dataset[100]
print("Review:", " ".join(vocab.lookup_tokens(x.numpy().tolist())))
print("Label:", y.item())
```

```
print("Sequence Length: %d"% len(x))

class MyModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.embedding = nn.Embedding(2000, 32)
        self.conv1d = nn.Conv1d(32, 64, kernel_size=3, padding=1)
        self.ReLU = nn.ReLU()
        self.dropout = nn.Dropout(0.8)
        self.maxpool = nn.MaxPool1d(kernel_size=2)
        self.lstm = nn.LSTM(64, 256, num_layers=2, batch_first=True)
        self.mlp = nn.Sequential(
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU()
        )
        self.fc = nn.Linear(32, 2)
    def forward(self, x):
        out = self.embedding(x)
        out = out.transpose(1, 2)
        out = self.conv1d(out)
        out = self.ReLU(out)
        out = self.dropout(out)
        out = self.maxpool(out)
        out = out.transpose(1, 2)
        y, (s, c) = self.lstm(out)
        out = self.mlp(c[0])
        return self.fc(out)
```

```
Training dataset: 20000
Test dataset: 5000
Review: <unk> the life of being <unk> by her father . . and the <unk> <unk> who di
es in an <unk> <unk> powers <unk> <unk> who is simply <unk> <unk> her way through
the <unk> inside a bank business in big city <unk> . when a <unk> lover murders wh
o was supposed to be his next father in law and <unk> s new lover the sky s the <u
nk> for <unk> as she has written down her various relationships in a <unk> and <un
k> makes it known the <unk> will <unk> it if certain pay doesn t come into her han
ds . <unk> <unk> <unk> to the bank <unk> <unk> george <unk> <unk> <unk> to paris i
nstead of <unk> over lots of <unk> but soon finds himself <unk> in love after vari
ous <unk> with her in the city of love . this makes <unk> s mouth water as now she
ll have <unk> the <unk> of success <unk> a man of <unk> and <unk> bring <unk> her
way . though <unk> <unk> which will bring her to make a <unk> that <unk> her succe
ssful way of <unk> those <unk> . . <unk> now her husband is being <unk> with <unk>
certain
Label: 1
Sequence Length: 200
```

In [10]:

```
"🔒"
# @check
# @title: verify model output

model = MyModel()
dataloader = DataLoader(train_dataset, batch_size=128, shuffle=True)
```

```
xs, targets = next(iter(dataloader))
model(xs).shape
```

Out[10]: torch.Size([128, 2])

Training

Implement a function `train` that will train the model.

Inputs are:

- model: an instance of the `MyModel`
- train_dataset: a dataset to be trained on.
- epochs: the number of epochs
- max_batches: optional integer that will limit the number of batches per epoch.

Returns a Pandas DataFrame will columns: `train_loss` and `train_acc` which are the training loss and accuracy per epoch.

Hint:

- Start with a simple model, and make sure that you can get a decent performance.
- Start with a small number of `max_batches` to make sure you get a decent training accuracy.
- Output debugging message with timing information, so you can estimate the training duration.
- For good test accuracy, you need `max_matches ~ 500` and 20 epochs or more.

In [550…

```
"✍"
# @workUnit
import time
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print("Using device: %s" % device)

# print the number of model parameters
print("Number of parameters: %d" % sum(p.numel() for p in model.parameters()))
def train(model: MyModel, train_dataset: Dataset, epochs: int, max_batches=None) ->

    model = model.to(device)
    dataloader = DataLoader(train_dataset, batch_size=64, shuffle=True)
    loss_fn = nn.CrossEntropyLoss()
    loss_fn = loss_fn.to(device)
    optimzer = optim.Adam(model.parameters(), lr=0.001, weight_decay=0.0001)
    # scheduler = optim.lr_scheduler.StepLR(optimzer, step_size=8, gamma=0.01)
    history = {
        'train_loss': [],
        'train_acc': []
    }
    start0 = time.time()
    total, succ = 0.0, 0.0
```

```
    for epoch in range(epochs):
        start = time.time()
        for i, (xs, targets) in enumerate(dataloader):
            xs, targets = xs.to(device), targets.to(device)
            optimzer.zero_grad()
            y = model(xs)
            loss = loss_fn(y, targets)
            loss.backward()
            optimzer.step()
            history['train_loss'].append(loss.item())
            with torch.no_grad():
                pred = y.argmax(axis=-1)
                succ += (pred == targets).sum().item()
                total += len(targets)
                history['train_acc'].append(succ / total)
            if i % 100 == 0:
                print("Batch %d: train_loss=%.4f, train_acc=%.4f" % (i, loss.item()
            if max_batches is not None and i >= max_batches:
                break
        duration = time.time() - start
        print("[Epoch [%d/%d]: train_loss=%.4f, train_acc=%.4f [%.2f Seconds]]" % (
    duration0 = time.time() - start0
    print("Training finished in %.2f Seconds" % duration0)
    return pd.DataFrame(history)
```

```
Using device: cuda
Number of parameters: 969570
```

In [558…

```
%%time
"✍"
# @workUnit

#
# train the model
#
model = MyModel()
hist = train(model, train_dataset, epochs=30, max_batches=500)
```

```
Batch 0: train_loss=0.6907, train_acc=0.5312
Batch 100: train_loss=0.6951, train_acc=0.4997
Batch 200: train_loss=0.6901, train_acc=0.4981
Batch 300: train_loss=0.6930, train_acc=0.4967
[Epoch [0/30]: train_loss=0.6935, train_acc=0.4974 [5.99 Seconds]]
Batch 0: train_loss=0.6942, train_acc=0.4973
Batch 100: train_loss=0.6922, train_acc=0.4999
Batch 200: train_loss=0.6971, train_acc=0.5029
Batch 300: train_loss=0.6916, train_acc=0.5030
[Epoch [1/30]: train_loss=0.6933, train_acc=0.5030 [5.95 Seconds]]
Batch 0: train_loss=0.6936, train_acc=0.5030
Batch 100: train_loss=0.6921, train_acc=0.5040
Batch 200: train_loss=0.6990, train_acc=0.5052
Batch 300: train_loss=0.7298, train_acc=0.5073
[Epoch [2/30]: train_loss=0.6929, train_acc=0.5076 [5.92 Seconds]]
Batch 0: train_loss=0.7446, train_acc=0.5076
Batch 100: train_loss=0.6922, train_acc=0.5077
Batch 200: train_loss=0.6922, train_acc=0.5081
Batch 300: train_loss=0.6934, train_acc=0.5076
[Epoch [3/30]: train_loss=0.6929, train_acc=0.5077 [5.91 Seconds]]
Batch 0: train_loss=0.6938, train_acc=0.5076
Batch 100: train_loss=0.6931, train_acc=0.5083
Batch 200: train_loss=0.6955, train_acc=0.5077
Batch 300: train_loss=0.6872, train_acc=0.5091
[Epoch [4/30]: train_loss=0.6926, train_acc=0.5096 [5.90 Seconds]]
Batch 0: train_loss=0.6710, train_acc=0.5096
Batch 100: train_loss=0.6907, train_acc=0.5107
Batch 200: train_loss=0.6833, train_acc=0.5117
Batch 300: train_loss=0.6534, train_acc=0.5140
[Epoch [5/30]: train_loss=0.6916, train_acc=0.5141 [5.90 Seconds]]
Batch 0: train_loss=0.6961, train_acc=0.5141
Batch 100: train_loss=0.6954, train_acc=0.5142
Batch 200: train_loss=0.6958, train_acc=0.5153
Batch 300: train_loss=0.5838, train_acc=0.5198
[Epoch [6/30]: train_loss=0.6896, train_acc=0.5206 [5.91 Seconds]]
Batch 0: train_loss=0.6401, train_acc=0.5206
Batch 100: train_loss=0.5301, train_acc=0.5303
Batch 200: train_loss=0.3780, train_acc=0.5408
Batch 300: train_loss=0.6054, train_acc=0.5517
[Epoch [7/30]: train_loss=0.6624, train_acc=0.5529 [5.93 Seconds]]
Batch 0: train_loss=0.3545, train_acc=0.5530
Batch 100: train_loss=0.3709, train_acc=0.5640
Batch 200: train_loss=0.3850, train_acc=0.5743
Batch 300: train_loss=0.2940, train_acc=0.5841
[Epoch [8/30]: train_loss=0.6281, train_acc=0.5853 [5.91 Seconds]]
Batch 0: train_loss=0.4377, train_acc=0.5853
Batch 100: train_loss=0.2168, train_acc=0.5950
Batch 200: train_loss=0.2445, train_acc=0.6039
Batch 300: train_loss=0.2927, train_acc=0.6121
[Epoch [9/30]: train_loss=0.5978, train_acc=0.6129 [5.93 Seconds]]
Batch 0: train_loss=0.2825, train_acc=0.6130
Batch 100: train_loss=0.3385, train_acc=0.6209
Batch 200: train_loss=0.4184, train_acc=0.6283
Batch 300: train_loss=0.2931, train_acc=0.6354
[Epoch [10/30]: train_loss=0.5716, train_acc=0.6362 [5.95 Seconds]]
Batch 0: train_loss=0.3219, train_acc=0.6363
```

```
Batch 100: train_loss=0.3338, train_acc=0.6430
Batch 200: train_loss=0.2165, train_acc=0.6495
Batch 300: train_loss=0.2909, train_acc=0.6554
[Epoch [11/30]: train_loss=0.5485, train_acc=0.6561 [5.97 Seconds]]
Batch 0: train_loss=0.3834, train_acc=0.6561
Batch 100: train_loss=0.2995, train_acc=0.6620
Batch 200: train_loss=0.2343, train_acc=0.6676
Batch 300: train_loss=0.3511, train_acc=0.6728
[Epoch [12/30]: train_loss=0.5286, train_acc=0.6733 [5.91 Seconds]]
Batch 0: train_loss=0.2607, train_acc=0.6734
Batch 100: train_loss=0.1568, train_acc=0.6786
Batch 200: train_loss=0.3070, train_acc=0.6834
Batch 300: train_loss=0.2976, train_acc=0.6879
[Epoch [13/30]: train_loss=0.5109, train_acc=0.6884 [5.60 Seconds]]
Batch 0: train_loss=0.2817, train_acc=0.6884
Batch 100: train_loss=0.2157, train_acc=0.6930
Batch 200: train_loss=0.3398, train_acc=0.6973
Batch 300: train_loss=0.2287, train_acc=0.7012
[Epoch [14/30]: train_loss=0.4950, train_acc=0.7016 [5.58 Seconds]]
Batch 0: train_loss=0.2796, train_acc=0.7017
Batch 100: train_loss=0.3040, train_acc=0.7057
Batch 200: train_loss=0.2235, train_acc=0.7094
Batch 300: train_loss=0.2901, train_acc=0.7131
[Epoch [15/30]: train_loss=0.4806, train_acc=0.7135 [5.57 Seconds]]
Batch 0: train_loss=0.2007, train_acc=0.7135
Batch 100: train_loss=0.3552, train_acc=0.7171
Batch 200: train_loss=0.3246, train_acc=0.7205
Batch 300: train_loss=0.2143, train_acc=0.7238
[Epoch [16/30]: train_loss=0.4673, train_acc=0.7241 [5.61 Seconds]]
Batch 0: train_loss=0.2583, train_acc=0.7242
Batch 100: train_loss=0.2206, train_acc=0.7274
Batch 200: train_loss=0.3068, train_acc=0.7304
Batch 300: train_loss=0.3494, train_acc=0.7333
[Epoch [17/30]: train_loss=0.4554, train_acc=0.7336 [5.62 Seconds]]
Batch 0: train_loss=0.1923, train_acc=0.7337
Batch 100: train_loss=0.2502, train_acc=0.7366
Batch 200: train_loss=0.1851, train_acc=0.7394
Batch 300: train_loss=0.2539, train_acc=0.7419
[Epoch [18/30]: train_loss=0.4446, train_acc=0.7422 [5.57 Seconds]]
Batch 0: train_loss=0.3849, train_acc=0.7423
Batch 100: train_loss=0.2821, train_acc=0.7450
Batch 200: train_loss=0.1830, train_acc=0.7476
Batch 300: train_loss=0.4226, train_acc=0.7500
[Epoch [19/30]: train_loss=0.4345, train_acc=0.7502 [5.60 Seconds]]
Batch 0: train_loss=0.2240, train_acc=0.7502
Batch 100: train_loss=0.3129, train_acc=0.7527
Batch 200: train_loss=0.2393, train_acc=0.7550
Batch 300: train_loss=0.2230, train_acc=0.7573
[Epoch [20/30]: train_loss=0.4252, train_acc=0.7575 [5.63 Seconds]]
Batch 0: train_loss=0.1433, train_acc=0.7576
Batch 100: train_loss=0.1201, train_acc=0.7599
Batch 200: train_loss=0.2035, train_acc=0.7620
Batch 300: train_loss=0.3145, train_acc=0.7640
[Epoch [21/30]: train_loss=0.4166, train_acc=0.7642 [5.57 Seconds]]
Batch 0: train_loss=0.2716, train_acc=0.7642
Batch 100: train_loss=0.3275, train_acc=0.7663
```

```
Batch 200: train_loss=0.2435, train_acc=0.7683
Batch 300: train_loss=0.3889, train_acc=0.7701
[Epoch [22/30]: train_loss=0.4087, train_acc=0.7703 [5.59 Seconds]]
Batch 0: train_loss=0.1541, train_acc=0.7703
Batch 100: train_loss=0.1987, train_acc=0.7723
Batch 200: train_loss=0.4584, train_acc=0.7741
Batch 300: train_loss=0.2758, train_acc=0.7759
[Epoch [23/30]: train_loss=0.4012, train_acc=0.7761 [5.62 Seconds]]
Batch 0: train_loss=0.1849, train_acc=0.7761
Batch 100: train_loss=0.2985, train_acc=0.7779
Batch 200: train_loss=0.3264, train_acc=0.7796
Batch 300: train_loss=0.2188, train_acc=0.7813
[Epoch [24/30]: train_loss=0.3942, train_acc=0.7814 [5.61 Seconds]]
Batch 0: train_loss=0.2429, train_acc=0.7815
Batch 100: train_loss=0.3031, train_acc=0.7831
Batch 200: train_loss=0.1832, train_acc=0.7848
Batch 300: train_loss=0.1530, train_acc=0.7863
[Epoch [25/30]: train_loss=0.3875, train_acc=0.7865 [5.62 Seconds]]
Batch 0: train_loss=0.2776, train_acc=0.7865
Batch 100: train_loss=0.1833, train_acc=0.7882
Batch 200: train_loss=0.1758, train_acc=0.7897
Batch 300: train_loss=0.1506, train_acc=0.7912
[Epoch [26/30]: train_loss=0.3811, train_acc=0.7913 [5.60 Seconds]]
Batch 0: train_loss=0.1968, train_acc=0.7913
Batch 100: train_loss=0.2167, train_acc=0.7929
Batch 200: train_loss=0.2961, train_acc=0.7943
Batch 300: train_loss=0.2187, train_acc=0.7956
[Epoch [27/30]: train_loss=0.3751, train_acc=0.7958 [5.57 Seconds]]
Batch 0: train_loss=0.0935, train_acc=0.7958
Batch 100: train_loss=0.2203, train_acc=0.7972
Batch 200: train_loss=0.1042, train_acc=0.7985
Batch 300: train_loss=0.2695, train_acc=0.7997
[Epoch [28/30]: train_loss=0.3695, train_acc=0.7998 [5.61 Seconds]]
Batch 0: train_loss=0.1003, train_acc=0.7999
Batch 100: train_loss=0.1299, train_acc=0.8012
Batch 200: train_loss=0.1690, train_acc=0.8025
Batch 300: train_loss=0.2135, train_acc=0.8037
[Epoch [29/30]: train_loss=0.3641, train_acc=0.8038 [5.57 Seconds]]
Training finished in 172.23 Seconds
CPU times: total: 2min 52s
Wall time: 2min 52s
```

In [559…

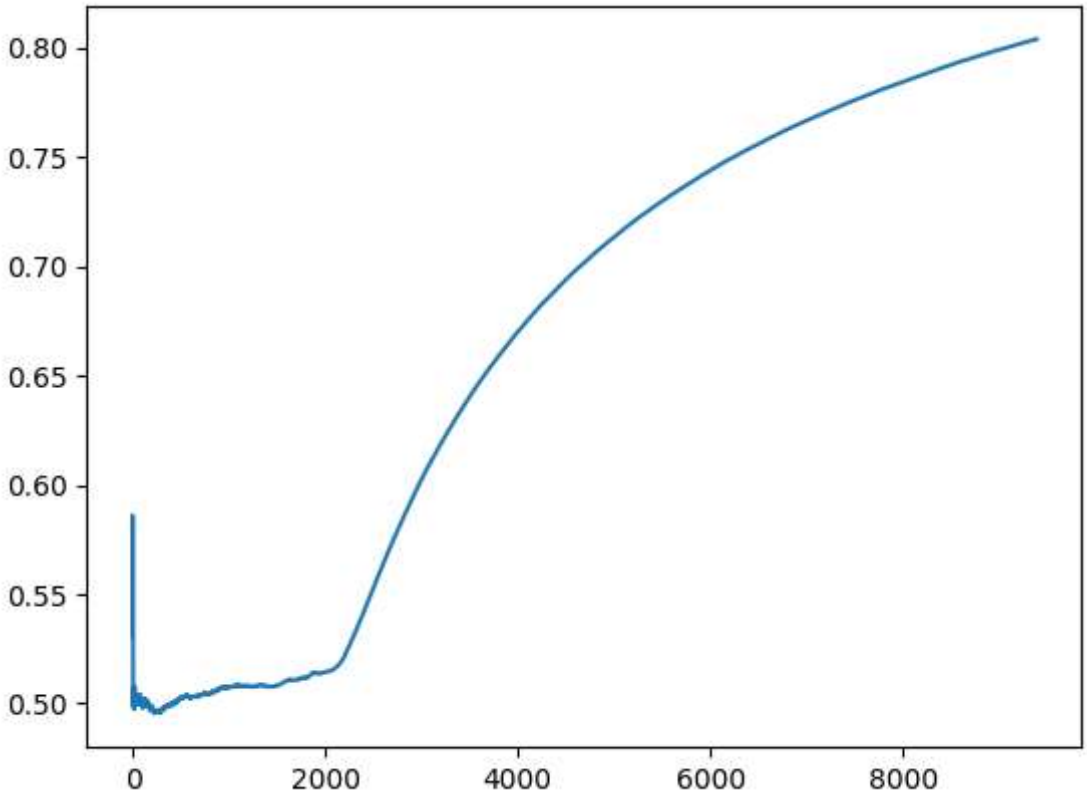
```
"🔒"
# @check
# @title: verify dataframe columns

hist.columns
```

Out[559]:

```
Index(['train_loss', 'train_acc'], dtype='object')
```

In [560…

```
"🔒"
#
# Plot the training accuracy
#
hist.train_acc.plot.line();
```



In [561…

```
"🔒"
#
# Save the entire model to disk
#
torch.save(model, 'mymodel.pt')
```

Testing

The following code evaluates your model using `test_dataset`.

In [574…

```
"🔒"
#
# Test your current model in memory
#
test_lib.test_saved_model(model)
```

```
Saved model has test accuracy = 85.38
```

In [575…

```
"🔒"
#
# Test your saved model on disk
#
test_lib.test_saved_model()
```

```
Loading from mymodel.pt
Saved model has test accuracy = 85.96
```