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 [22]:
```
"🔒"
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')
```

In [23]:
```
from torch.optim import Adam
import time
from torch.utils.data import random_split
from torchmetrics import Accuracy
```

Load dataset and vocabulary from file

You are given two dataset files for training and testing.

In [24]:
```
"🔒"
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 [25]:
```
"🔒"
vocab = torch.load('./vocab.pt')
print("There are %d tokens in vocabulary." % len(vocab))
```

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

Check data

In [26]:
```
"🔒"
# @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 guys and the <unk> <unk> the <unk> detective <unk> s best men . <unk> is in top form in the famous in the dark scene . <unk> <unk> provides excellen
t support in his mr . <unk> role as the <unk> of a murder plot . before it s over <unk> s <unk> little <unk> is <unk> to great effect . this minute gem moves about as fast as any <unk> s short and <unk> twice the <unk> . highly r
ecommended . <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 [81]:
```
"✍"
# @workUnit

class MyModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.embedding = nn.Embedding(2000,100)
        self.lstm = nn.LSTM(input_size = 100,
                            hidden_size = 100,
                            num_layers = 4,
                            batch_first = True,
                            dropout=0.2)
        self.mlp = nn.Sequential(nn.Linear(100,100),
                                 nn.ReLU(),
                                 nn.Linear(100,2))

    def forward(self, tokens):
        x = self.embedding(tokens)
        y, (s, c) = self.lstm(x)
        y = self.mlp(c[0])
        return y
```

In [82]:
```
"🔒"
# @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[82]:
```
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 [83]:
```
"✍"
# @workUnit
```

Out[83]:
```
"✍"
```

In [84]:
```
def train(model: MyModel,train_dataset: Dataset,epochs: int, max_batches=None) -> pd.DataFrame:
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    dataloader = DataLoader(train_dataset, batch_size=max_batches, shuffle=True)
    history = {
            'train_loss': [],
            'train_acc': [],
        }
    optimizer = torch.optim.Adam(model.parameters())
    loss = torch.nn.CrossEntropyLoss()
    for epoch in range(epochs):
        l_list = []
        acc_list = []
        accuracy = Accuracy(task='multiclass', num_classes=2)
        for (xs, targets) in dataloader:
            xs.to(device)
            targets.to(device)
            optimizer.zero_grad()
            pred = model(xs)
            l = loss(pred, targets)
            l.backward()
            optimizer.step()
            l_list.append(l.item())
            acc_list.append(accuracy(pred, targets).item())
        train_loss, train_acc = np.mean(l_list), np.mean(acc_list)

        history['train_loss'].append(train_loss)
        history['train_acc'].append(train_acc)
        print("train_loss=%.2f train_acc=%.2f" % (train_loss, train_acc))
    return pd.DataFrame(history)
```

In [85]:
```
"✍"
# @workUnit

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

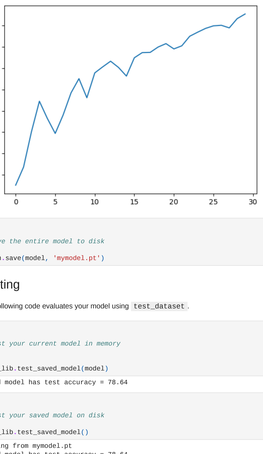
```
train_loss=0.69 train_acc=0.53
train_loss=0.68 train_acc=0.57
train_loss=0.63 train_acc=0.65
train_loss=0.56 train_acc=0.72
train_loss=0.62 train_acc=0.68
train_loss=0.64 train_acc=0.65
train_loss=0.60 train_acc=0.69
train_loss=0.54 train_acc=0.74
train_loss=0.49 train_acc=0.78
train_loss=0.54 train_acc=0.73
train_loss=0.47 train_acc=0.79
train_loss=0.44 train_acc=0.80
train_loss=0.41 train_acc=0.82
train_loss=0.46 train_acc=0.80
train_loss=0.47 train_acc=0.78
train_loss=0.40 train_acc=0.82
train_loss=0.38 train_acc=0.84
train_loss=0.38 train_acc=0.84
train_loss=0.35 train_acc=0.85
train_loss=0.34 train_acc=0.86
train_loss=0.35 train_acc=0.85
train_loss=0.35 train_acc=0.85
train_loss=0.30 train_acc=0.88
train_loss=0.28 train_acc=0.88
train_loss=0.27 train_acc=0.89
train_loss=0.25 train_acc=0.90
train_loss=0.24 train_acc=0.90
train_loss=0.26 train_acc=0.89
train_loss=0.21 train_acc=0.92
train_loss=0.19 train_acc=0.93
```

In [86]:
```
"🔒"
# @check
# @title: verify dataframe columns

hist.columns
```

Out[86]:
```
Index(['train_loss', 'train_acc'], dtype='object')
```

In [87]:
```
"🔒"
#
# Plot the training accuracy
#
hist.train_acc.plot.line();
```



In [88]:
```
"🔒"
#
# Save the entire model to disk
#
torch.save(model, 'mymodel.pt')
```

Testing

The following code evaluates your model using `test_dataset`.

In [89]:
```
"🔒"
#
# Test your current model in memory
#

test_lib.test_saved_model(model)
```

```
Saved model has test accuracy = 78.64
```

In [90]:
```
"🔒"
#
# Test your saved model on disk
#
test_lib.test_saved_model()
```

```
Loading from mymodel.pt
Saved model has test accuracy = 78.64
```

In [ ]:

In [ ]: