Installing Modules

```
!pip install spacy==3
!python -m spacy download en_core_web_sm
!pip install pytorch_lightning torchmetrics tableprint
```

```
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: click<7.2.0,>=7.1.1 in /usr/local/lib/python
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.7/
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3
Requirement already satisfied: smart-open<4.0.0,>=2.2.0 in /usr/local/lib/py
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
Requirement already satisfied: pytorch_lightning in /usr/local/lib/python3.
Requirement already satisfied: torchmetrics in /usr/local/lib/python3.7/dist
Requirement already satisfied: tableprint in /usr/local/lib/python3.7/dist-
Requirement already satisfied: PyYAML<=5.4.1,>=5.1 in /usr/local/lib/python
Requirement already satisfied: torch>=1.4 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: tensorboard!=2.5.0,>=2.2.0 in /usr/local/lib/
Requirement already satisfied: numpy>=1.17.2 in /usr/local/lib/python3.7/di
Requirement already satisfied: pyDeprecate==0.3.0 in /usr/local/lib/python3
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: fsspec[http]!=2021.06.0,>=2021.05.0 in /usr/
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: future>=0.17.1 in /usr/local/lib/python3.7/d
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/
Requirement already satisfied: wheel>=0.26; python_version >= "3" in /usr/l
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/pyth
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.7/dist
Requirement already satisfied: protobuf>=3.6.0 in /usr/local/lib/python3.7/
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/loca
Requirement already satisfied: grpcio>=1.24.3 in /usr/local/lib/python3.7/d
Requirement already satisfied: aiohttp; extra == "http" in /usr/local/lib/py
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.7/
Requirement already satisfied: importlib-metadata; python_version < "3.8" i
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3.6" in /u
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/pyth
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/pyt
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/py
```

```
Requirement already satisfied: async-timeout<4.0,>=3.0 in /usr/local/lib/py
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pyasn1>=0.1.3 in /usr/local/lib/python3.7/di
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/
```

Imports

```
# Import Library
import random
import torch, torchtext
from torchtext.legacy import data
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim


import pandas as pd
import sys, os, pickle
import numpy as np
import matplotlib.pyplot  as plt
import seaborn as sns

import spacy
nlp = spacy.load('en_core_web_sm')

import pytorch_lightning as pl
import torchmetrics

from pytorch_lightning.loggers import CSVLogger
from pytorch_lightning.callbacks import ModelCheckpoint
from sklearn.metrics import confusion_matrix
import tableprint as tp


# Manual Seed
SEED = 43
torch.manual_seed(SEED)
```

```
<torch._C.Generator at 0x7f377210c890>
```

Loading Data

Files have been saved to google drive for faster access!

```
!gdown --id 1HmYahgrwNcZREWtUTr6H11ygJufuFcTc
!gdown --id 14hb3DlvmMeEvWhNYXAZjhE3MFS1T8Nte
```

```
!gdown --id 1xwvuoXp35tjE-rV7oA0kq6T42qli344P
```

```
Downloading...
From: https://drive.google.com/uc?id=1HmYahgrwNcZREWtUTr6H11ygJufuFcTc
To: /content/datasetSentences.txt
100% 1.29M/1.29M [00:00<00:00, 4.88MB/s]
Downloading...
From: https://drive.google.com/uc?id=14hb3DlvmMeEvWhNYXAZjhE3MFS1T8Nte
To: /content/sentiment_labels.txt
3.26MB [00:00, 12.4MB/s]
Downloading...
From: https://drive.google.com/uc?id=1xwvuoXp35tjE-rV7oA0kq6T42qli344P
To: /content/dictionary.txt
12.0MB [00:00, 33.0MB/s]
```

The sentiments are read for the phrases (with their ids as the mapping index)

```
sentiment_labels = pd.read_csv("sentiment_labels.txt", sep="|", header=0)
sentiment_labels.columns = ["id", "sentiment"]
sentiment_labels.head()
```

	id	sentiment
0	0	0.50000
1	1	0.50000
2	2	0.44444
3	3	0.50000
4	4	0.42708

The sentiments are mapped onto a discrete set of 5-values (and will be referred to as the label)

```
sentiment_labels["label"] = pd.cut(sentiment_labels.sentiment, [0, 0.2, 0.4, 0.6, (
                                   include_lowest=True,
                                   labels=[0, 1, 2, 3, 4])
sentiment_labels.drop('sentiment', inplace=True, axis=1)
sentiment_labels.head()
```

	id	label
0	0	2
1	1	2
2	2	2
3	3	2
4	4	2

The sentences are read here!

```
sentences = pd.read_csv("datasetSentences.txt", index_col="sentence_index",sep="\t
sentences.head()
```

	sentence
sentence_index	
1	The Rock is destined to be the 21st Century 's...
2	The gorgeously elaborate continuation of `` Th...
3	Effective but too-tepid biopic
4	If you sometimes like to go to the movies to h...
5	Emerges as something rare , an issue movie tha...

The `dictionary.txt` file maps the phrases to the ids

```
dictionary = pd.read_csv("dictionary.txt", sep="|", header=0)
dictionary.columns = ["phrase", "id"]
dictionary.head()
```

	phrase	id
0	! '	22935
1	! ''	18235
2	! Alas	179257
3	! Brilliant	22936
4	! Brilliant !	40532

Here, the mapping is done from phrase ids to phrases themselves, followed by mapping of sentences to the labels.

```
sentence_phrase_merge = pd.merge(sentences, dictionary, left_on='sentence', right_
dataset = pd.merge(sentence_phrase_merge, sentiment_labels, on='id')
dataset.head()
```

	sentence	phrase	id	label
0	The Rock is destined to be the 21st Century 's...	The Rock is destined to be the 21st Century 's...	226166	3
1	The gorgeously elaborate continuation of `` Th...	The gorgeously elaborate continuation of `` Th...	226300	4
2	Effective but too-tepid biopic	Effective but too-tepid biopic	13995	2
3	If you sometimes like to go to the movies to h...	If you sometimes like to go to the movies to h...	14123	3

The dataset is cleaned

```
dataset['sentence_cleaned'] = dataset['sentence'].str.replace(r"\s('s|'d|'re|'ll|'
dataset.head()
```

	sentence	phrase	id	label	sentence_cleaned
0	The Rock is destined to be the 21st Century 's...	The Rock is destined to be the 21st Century 's...	226166	3	The Rock is destined to be the 21st Century's ...
1	The gorgeously elaborate continuation of `` Th...	The gorgeously elaborate continuation of `` Th...	226300	4	The gorgeously elaborate continuation of `` Th...
	Effective but too-tepid	Effective but too-tepid			Effective but too-

Only the cleaned sentences and the labels are retained

```
dataset.drop(['phrase', 'id', 'sentence'], inplace=True,axis=1)
dataset.columns = ["label", "sentence"]
dataset.head()
```

	label	sentence
0	3	The Rock is destined to be the 21st Century's ...
1	4	The gorgeously elaborate continuation of `` Th...
2	2	Effective but too-tepid biopic
3	3	If you sometimes like to go to the movies to h...
4	4	Emerges as something rare , an issue movie tha...

Dataset Preview

Let's just preview the dataset.

```
dataset.head()
```

	label	sentence
0	3	The Rock is destined to be the 21st Century's ...
1	4	The gorgeously elaborate continuation of `` Th...
2	2	Effective but too-tepid biopic
3	3	If you sometimes like to go to the movies to h...
4	4	Emerges as something rare , an issue movie tha...

```
dataset.shape
```

```
(11286, 2)
```

```
dataset.label.value_counts()
```

```
1    2971
3    2966
2    2144
4    1773
0    1432
Name: label, dtype: int64
```

Defining Fields

Now we shall be defining Label as a LabelField, which is a subclass of Field that sets sequen tial to False (as it's our numerical category class). Sentence is a standard Field object, where we have decided to use the spaCy tokenizer and convert all the text to lower- case.

```
Sentence = data.Field(sequential = True, tokenize = 'spacy', batch_first =True, in
Label = data.LabelField(tokenize ='spacy', is_target=True, batch_first =True, sequ
```

```
/usr/local/lib/python3.7/dist-packages/torchtext/data/utils.py:123: UserWarni
  warnings.warn(f'Spacy model "{language}" could not be loaded, trying "{OLD_
```

Having defined those fields, we now need to produce a list that maps them onto the list of rows that are in the CSV:

```
fields = [('sentence', Sentence),('label',Label)]
```

Armed with our declared fields, lets convert from pandas to list to torchtext. We could also use TabularDataset to apply that definition to the CSV directly but showing an alternative approach too.

```
example = [data.Example.fromlist([dataset.sentence[i],dataset.label[i]], fields) f
```

```
stanfordDataset = data.Dataset(example, fields)
```

Finally, we can split into training, testing, and validation sets by using the split() method:

```
(train, test) = stanfordDataset.split(split_ratio=[0.70, 0.30], random_state=rando
```

Double-click (or enter) to edit

```
(len(train), len(test))
```

```
(7900, 3386)
```

An example from the dataset:

```
vars(train.examples[10])
```

```
{'label': 1,
 'sentence': ['Disney',
  'again',
  'ransacks',
  'its',
  'archives',
  'for',
  'a',
  'quick',
  '-',
  'buck',
  'sequel',
  '.']}
```

```
" ".join((vars(train.examples[10]))['sentence'])
```

```
'Disney again ransacks its archives for a quick - buck sequel .'
```

Building Vocabulary

We will build vocabulary only using the `train` dataset and not the `test` dataset

```
Sentence.build_vocab(train)
Label.build_vocab(train)
```

By default, torchtext will add two more special tokens, for unknown words and , a padding token that will be used to pad all our text to roughly the same size to help with efficient batching on the GPU.

```
print('Size of input vocab : ', len(Sentence.vocab))
print('Size of label vocab : ', len(Label.vocab))
print('Top 10 words appreared repeatedly :', list(Sentence.vocab.freqs.most_common
print('Labels : ', Label.vocab.stoi)
```

```
Size of input vocab :  16378
Size of label vocab :  5
Top 10 words appreared repeatedly : [('.', 7452), (',', 6567), ('the', 5603),
Labels :  defaultdict(None, {1: 0, 3: 1, 2: 2, 4: 3, 0: 4})
```

Initializing GPU as the device

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

The Train/Test iterators are initialized here

```
train_iterator, test_iterator = data.BucketIterator.splits((train, test), batch_si
                                                            sort_key = lambda x: l
                                                            sort_within_batch=True
```

Save the vocabulary for later use

```
with open('tokenizer.pkl', 'wb') as tokens:
    pickle.dump(Sentence.vocab.stoi, tokens)
```

Defining Our Model

Boilerplate code for PyTorchLightning

```
class TL(pl.LightningModule):
    def __init__(self):
        super(TL, self).__init__()

        self.train_accm = torchmetrics.Accuracy()
        self.valid_accm = torchmetrics.Accuracy()
        self.train_acc =  torch.tensor(0.)
        self.avg_train_loss = torch.tensor(0.)
        self.table_context = None


    def training_step(self, batch, batch_idx):
        sent, sent_lengths = batch.sentence
        output = self(sent, sent_lengths)
        loss_train = self.loss(output, batch.label).squeeze()
        predictions = torch.argmax(output, dim=1)
        acc_train = self.train_accm(predictions, batch.label)
        return loss_train

    def validation_step(self, batch, batch_idx):
        sent, sent_lengths = batch.sentence
        output = self(sent, sent_lengths)
        loss_valid = self.loss(output, batch.label).squeeze()
        predictions = torch.argmax(output, dim=1)
        acc_valid = self.valid_accm(predictions, batch.label)
        return {"loss": loss_valid, "p": predictions, "y": batch.label}
```

```
    def training_epoch_end(self, outputs):
        self.train_acc = self.train_accm.compute() * 100
        self.avg_train_loss = torch.stack([x['loss'] for x in outputs]).mean()
        self.train_accm.reset()

    def validation_epoch_end(self, outputs):
        if trainer.running_sanity_check:
            return
        valid_acc = self.valid_accm.compute() * 100
        avg_valid_loss = torch.stack([x['loss'] for x in outputs]).mean()
        metrics = {'epoch': self.current_epoch+1, 'Train Acc': self.train_acc, 'Tra
        if self.table_context is None:
            self.table_context = tp.TableContext(headers=['epoch', 'Train Acc', 'T
            self.table_context.__enter__()
        self.table_context([self.current_epoch+1, self.train_acc.item(), self.avg_
        self.logger.log_metrics(metrics)
        self.valid_accm.reset()
        if self.current_epoch == self.trainer.max_epochs - 1:
            self.validation_end(outputs)

    def validation_end(self, outputs):
        pb = [x['p'] for x in outputs]
        yb = [x['y'] for x in outputs]
        p = torch.cat(pb, 0).view(-1).cpu()
        y = torch.cat(yb, 0).view(-1).cpu()
        self.table_context.__exit__()
        # confusion matrix here!
        cm = confusion_matrix(y.tolist(), p.tolist())
        df_cm = pd.DataFrame(cm, columns=np.unique(y), index = np.unique(y))
        df_cm.index.name = 'Actual'
        df_cm.columns.name = 'Predicted'
        plt.figure(figsize = (10,7))
        sns.set(font_scale=1.4)#for label size
        fig_ = sns.heatmap(df_cm, annot=True, cmap="Blues",annot_kws={"size": 16})
```

The Actual Model

```
class classifier(TL):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim, n_layers
        super(classifier, self).__init__()

        self.loss = nn.CrossEntropyLoss()
        self.lr = 1e-3

        # Embedding layer
        self.embedding = nn.Embedding(vocab_size, embedding_dim)

        # LSTM layer
        self.encoder = nn.LSTM(embedding_dim,
                               hidden_dim,
                               num_layers=n_layers,
                               dropout=dropout,
```

```
                                   batch_first=True)

        # Dense layer
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, text, text_lengths):

        embedded = self.embedding(text)
        packed_embedded = nn.utils.rnn.pack_padded_sequence(embedded, text_lengths
        packed_output, (hidden, cell) = self.encoder(packed_embedded)
        dense_outputs = self.fc(hidden)
        return dense_outputs[-1]

    def configure_optimizers(self):
        optim = torch.optim.Adam(self.parameters())
        return optim


# Define hyperparameters
size_of_vocab = len(Sentence.vocab)
embedding_dim = 100
num_hidden_nodes = 20
num_output_nodes = 5
num_layers = 4
dropout = 0.4

# Instantiate the model
model = classifier(size_of_vocab, embedding_dim, num_hidden_nodes, num_output_node
```

Model Checkpoint

This saves the best model (best => model with lowest val loss)

```
checkpoint_callback = ModelCheckpoint(
    monitor='val_loss',
    dirpath='/content',
    filename='sst-{epoch:02d}-{val_loss:.2f}',
    mode='min'
)
```

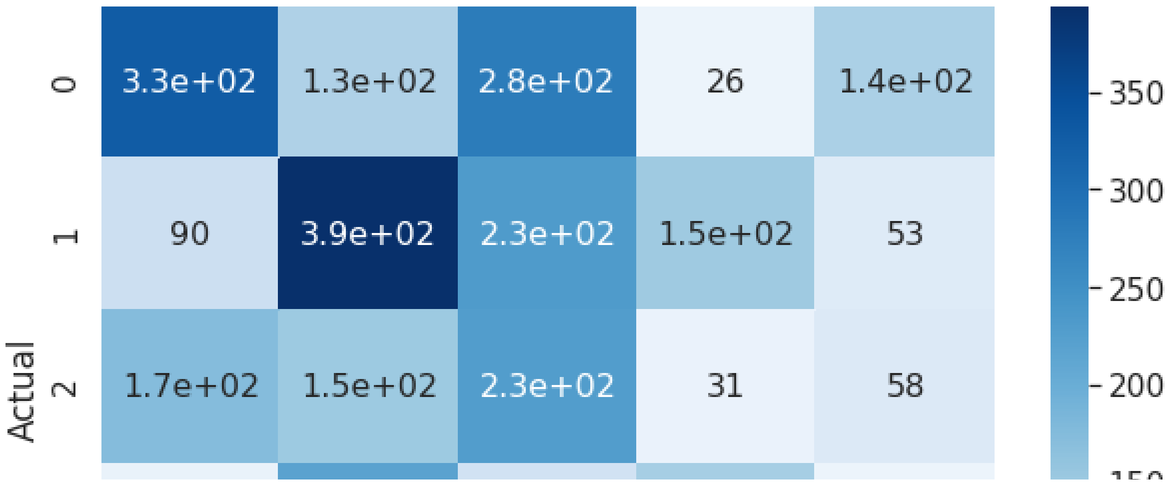
```
!rm -rf csv_logs
csvlogger = CSVLogger('csv_logs', name='END2 Assign 7_1_TL', version=0)
trainer = pl.Trainer(max_epochs=20, num_sanity_val_steps=1, logger=csvlogger, gpus
trainer.fit(model, train_dataloader=train_iterator, val_dataloaders=test_iterator)
checkpoint_callback.best_model_path
```

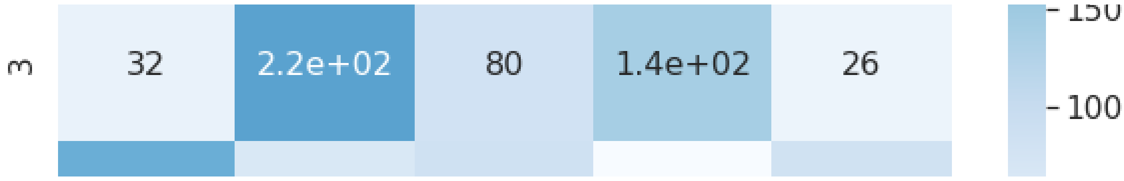
```
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name       | Type             | Params
------------------------------------------------
0 | train_accm | Accuracy         | 0
1 | valid_accm | Accuracy         | 0
2 | loss       | CrossEntropyLoss | 0
3 | embedding  | Embedding        | 1.6 M
4 | encoder    | LSTM             | 19.8 K
5 | fc         | Linear           | 105
------------------------------------------------
1.7 M     Trainable params
0         Non-trainable params
1.7 M     Total params
6.631     Total estimated model params size (MB)
```

Validation sanity check: 0% 0/1 [00:00<?, ?it/s]

Epoch 19: 100% 353/353 [00:03<00:00, 102.39it/s, loss=0.203, v_num=0]

epoch	Train Acc	Train Loss	Valid Acc	Valid Loss
1	26.38	1.5746	30.018	1.5603
2	30.886	1.5308	32.487	1.51
3	37.709	1.4197	34.879	1.4888
4	44.81	1.2915	36.799	1.4885
5	50.962	1.1527	34.82	1.562
6	57.532	1.0284	36.031	1.6687
7	63.532	0.90568	35.204	1.85
8	68.924	0.80161	35.145	1.9615
9	73.595	0.70365	34.495	2.1086
10	77.354	0.62018	35.263	2.1964
11	80.241	0.55864	34.79	2.3029
12	83.038	0.49319	34.672	2.4138
13	85.848	0.43099	35.351	2.5511
14	86.911	0.38719	33.579	2.6121
15	88.481	0.35891	34.997	2.7361
16	89.418	0.32695	34.406	2.8067
17	91.025	0.29164	34.702	2.8948
18	91.797	0.26694	34.377	2.9899
19	92.253	0.25037	34.584	3.0636
20	92.962	0.22841	34.584	3.0698





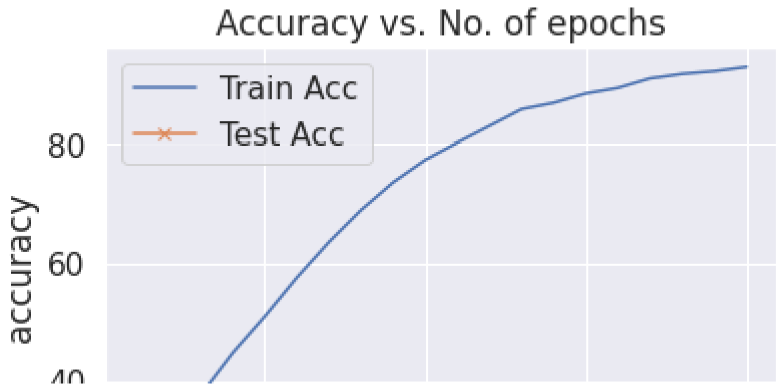
Model Training and Evaluation

First define the optimizer and loss functions

```
root='./csv_logs/' + 'END2 Assign 7_1_TL' + '/'
dirlist = [ item for item in os.listdir(root) if os.path.isdir(os.path.join(root, 
metricfile = root + dirlist[-1:][0] + '/metrics.csv'
metrics = pd.read_csv(metricfile)
```

```
plt.plot(metrics['epoch'], metrics['Train Loss'], label="Train Loss")
plt.plot(metrics['epoch'], metrics['Valid Loss'], '-x', label="Test Loss")
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.title('Loss vs. No. of epochs');
```


```
plt.plot(metrics['epoch'], metrics['Train Acc'], label="Train Acc")
plt.plot(metrics['epoch'], metrics['Valid Acc'], '-x', label="Test Acc")
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.legend()
plt.title('Accuracy vs. No. of epochs');
```



Model Testing

```
#load weights and tokenizer

model = model.to(device)
model.eval()
tokenizer_file = open('./tokenizer.pkl', 'rb')
tokenizer = pickle.load(tokenizer_file)

#inference
def classify_sentence(sentence):

    categories = {0: "Worst", 1:"Negative", 2:"Neutral", 3:"Positive", 4:"Great"}

    # tokenize the sentence
    tokenized = [tok.text for tok in nlp.tokenizer(sentence)]
    # convert to integer sequence using predefined tokenizer dictionary
    indexed = [tokenizer[t] for t in tokenized]
    # compute no. of words
    length = [len(indexed)]
    # convert to tensor
    tensor = torch.LongTensor(indexed).to(device)
    # reshape in form of batch, no. of words
    tensor = tensor.unsqueeze(1).T
    # convert to tensor
    length_tensor = torch.LongTensor(length).to(device)
    # Get the model prediction
    with torch.no_grad():
      prediction = model(tensor, length_tensor)

      _, pred = torch.max(prediction, 1)

    # return categories[pred.item()]
    return pred.item()


classify_sentence("This is something you will regret.")
```

```
1
```

```
for i in np.random.randint(0,len(test),10):
  sent = " ".join((vars(test.examples[i]))['sentence'])
```

```
    pred = classify_sentence(sent)
    label = (vars(test.examples[i]))['label']
    print(f'Sentence: {sent[:60]} \t Predicted: {pred} \t Actual: {label}')
```

```
Sentence: Terrific casting and solid execution give all three stories    Pred
Sentence: If it seems like a minor miracle that its septuagenarian sta   Pred
Sentence: Like its script , which nurses plot holes gaping enough to p   Pred
Sentence: Murder by Numbers ' is n't a great movie , but it 's a perfe   Pred
Sentence: The weird thing about The Santa Clause 2 , purportedly a chi   Pred
Sentence: Allen shows he can outgag any of those young whippersnappers   Pred
Sentence: I 've had more interesting -- and , dare I say , thematicall   Pred
Sentence: Miller has crafted an intriguing story of maternal instincts   Pred
Sentence: that it 'll probably be the best and most mature comedy of t   Pred
Sentence: A bittersweet contemporary comedy about benevolent deception   Pred
```

0s completed at 9:34 AM