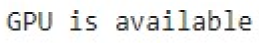
```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import torch
import torch.nn as nn
import torch.nn.functional as F
from nltk.corpus import stopwords
from collections import Counter
import string
import re
import seaborn as sns
from tqdm import tqdm
import matplotlib.pyplot as plt
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split
```

##

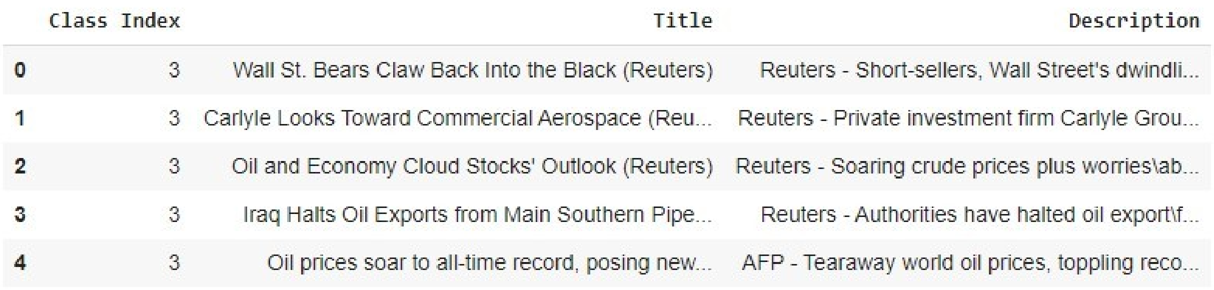
```python
import torch
is_cuda = torch.cuda.is_available()

# If we have a GPU available, we'll set our device to GPU. We'll use this device variable later in our code.
if is_cuda:
  device = torch.device("cuda")
  print("GPU is available")
else:
  device = torch.device("cpu")
  print("GPU not available, CPU used")
```



```
GPU is available
```

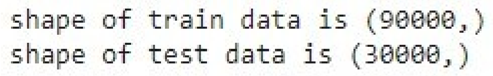
##

```python
base_csv = 'agnews.csv'
df = pd.read_csv(base_csv)
df.head()
```



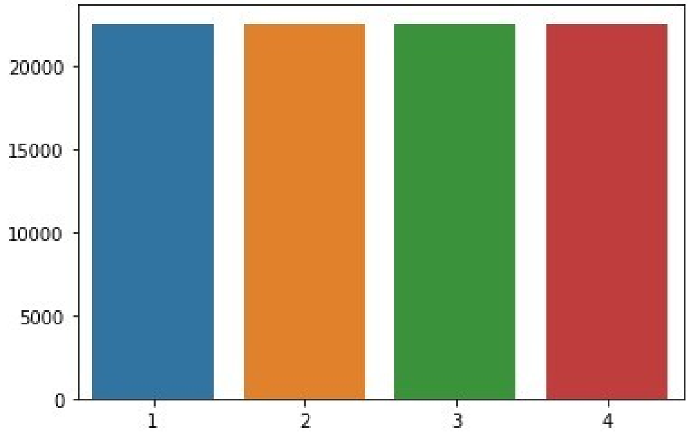
	Class Index	Title	Description
0	3	Wall St. Bears Claw Back Into the Black (Reuters)	Reuters - Short-sellers, Wall Street's dwindli...
1	3	Carlyle Looks Toward Commercial Aerospace (Reu...	Reuters - Private investment firm Carlyle Grou...
2	3	Oil and Economy Cloud Stocks' Outlook (Reuters)	Reuters - Soaring crude prices plus worries\ab...
3	3	Iraq Halts Oil Exports from Main Southern Pipe...	Reuters - Authorities have halted oil export\f...
4	3	Oil prices soar to all-time record, posing new...	AFP - Tearaway world oil prices, toppling reco...

```
######################################################################################

X,y = df['Description'].values,df['Class Index'].values
x_train,x_test,y_train,y_test = train_test_split(X,y,stratify=y)
print(f'shape of train data is {x_train.shape}')
print(f'shape of test data is {x_test.shape}')
```



```
shape of train data is (90000,)
shape of test data is (30000,)
```

```
######################################################################################

dd = pd.Series(y_train).value_counts()
sns.barplot(x=np.array(['1','2','3','4']),y=dd.values)
plt.show()
```



```
######################################################################################

def preprocess_string(s):
  # Remove all non-word characters (everything except numbers and letters)
  s = re.sub(r"[^\w\s]", '', s)
  # Replace all runs of whitespaces with no space
  s = re.sub(r"\s+", '', s)
  # replace digits with no space
  s = re.sub(r"\d", '', s)

  return s

def tockenize(x_train,y_train,x_val,y_val):
  word_list = []

  stop_words = set(stopwords.words('english'))
  for sent in x_train:
    for word in sent.lower().split():
      word = preprocess_string(word)
      if word not in stop_words and word != '':
        word_list.append(word)

  corpus = Counter(word_list)
  # sorting on the basis of most common words
  corpus_ = sorted(corpus,key=corpus.get,reverse=True)[:1000]
  # creating a dict
  onehot_dict = {w:i+1 for i,w in enumerate(corpus_)}

  # tockenize
  final_list_train,final_list_test = [],[]
  for sent in x_train:
      final_list_train.append([onehot_dict[preprocess_string(word)] for word in sent.l
ower().split()
                    if preprocess_string(word) in onehot_dict.keys()])
  for sent in x_val:
      final_list_test.append([onehot_dict[preprocess_string(word)] for word in sent.lo
wer().split()
                    if preprocess_string(word) in onehot_dict.keys()])

  return np.array(final_list_train), np.array(y_train),np.array(final_list_test), np.a
rray(y_val),onehot_dict

#########################################################################################
```

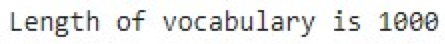
```
 >>> import nltk
 >>> nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
#########################################################################################

x_train,y_train,x_test,y_test,vocab = tockenize(x_train,y_train,x_test,y_test)

#########################################################################################
```

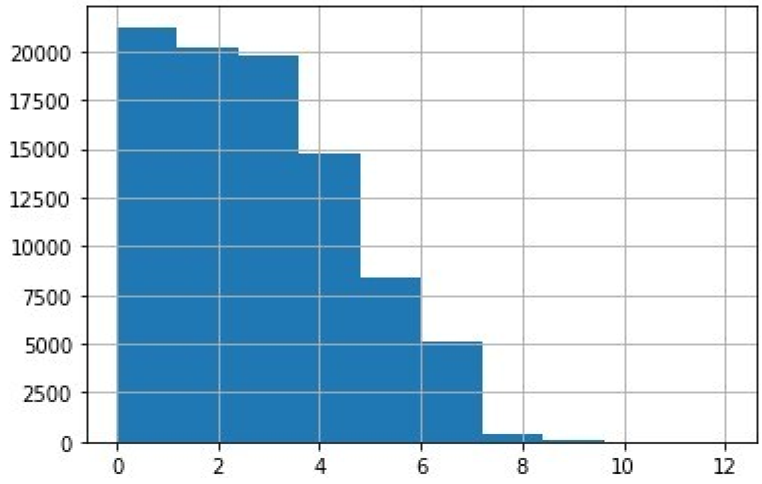
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:36:
VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray
```

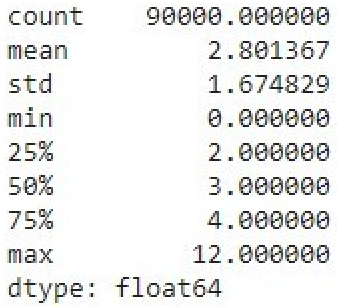
```
#########################################################################################

print(f'Length of vocabulary is {len(vocab)}')
```



```
Length of vocabulary is 1000
```

```
#########################################################################################

rev_len = [len(i) for i in x_train]
pd.Series(rev_len).hist()
plt.show()
pd.Series(rev_len).describe()
```

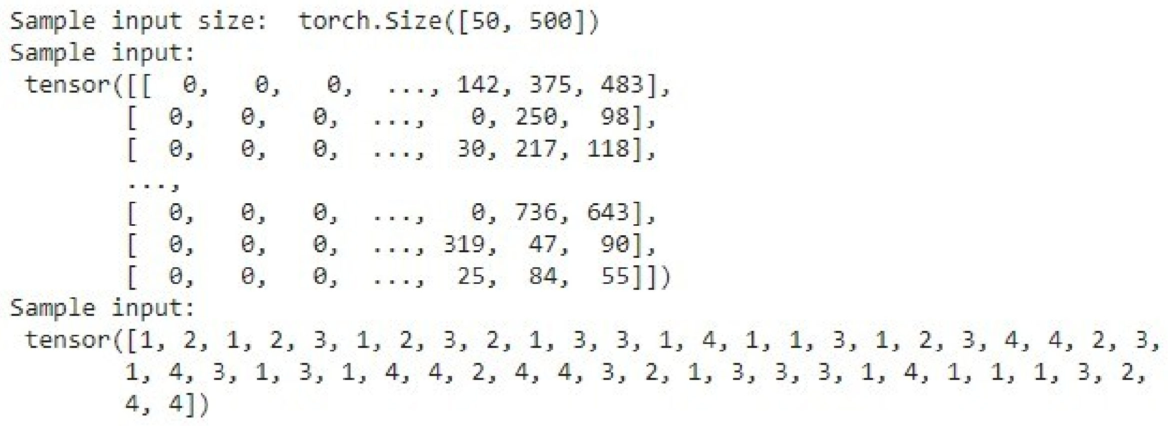




```
count    90000.000000
mean         2.801367
std          1.674829
min          0.000000
25%          2.000000
50%          3.000000
75%          4.000000
max         12.000000
dtype: float64
```

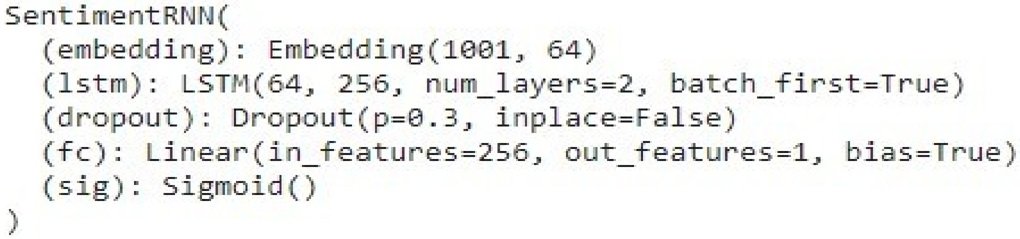
```
#########################################################################################

def padding_(sentences, seq_len):
  features = np.zeros((len(sentences), seq_len),dtype=int)
  for ii, review in enumerate(sentences):
    if len(review) != 0:
      features[ii, -len(review):] = np.array(review)[:seq_len]
  return features



#we have very less number of reviews with length > 500.
#So we will consideronly those below it.
```

```
x_train_pad = padding_(x_train,500)
x_test_pad = padding_(x_test,500)


######################################################################################


# create Tensor datasets
train_data = TensorDataset(torch.from_numpy(x_train_pad), torch.from_numpy(y_train))
valid_data = TensorDataset(torch.from_numpy(x_test_pad), torch.from_numpy(y_test))

# dataloaders
batch_size = 50

# make sure to SHUFFLE your data
train_loader = DataLoader(train_data, shuffle=True, batch_size=batch_size)
valid_loader = DataLoader(valid_data, shuffle=True, batch_size=batch_size)



# obtain one batch of training data
dataiter = iter(train_loader)
sample_x, sample_y = dataiter.next()

print('Sample input size: ', sample_x.size()) # batch_size, seq_length
print('Sample input: \n', sample_x)
print('Sample input: \n', sample_y)
```



```
Sample input size:  torch.Size([50, 500])
Sample input:
 tensor([[  0,   0,   0,  ..., 142, 375, 483],
        [  0,   0,   0,  ...,   0, 250,  98],
        [  0,   0,   0,  ...,  30, 217, 118],
        ...,
        [  0,   0,   0,  ...,   0, 736, 643],
        [  0,   0,   0,  ..., 319,  47,  90],
        [  0,   0,   0,  ...,  25,  84,  55]])
Sample input:
 tensor([1, 2, 1, 2, 3, 1, 2, 3, 2, 1, 3, 3, 1, 4, 1, 1, 3, 1, 2, 3, 4, 4, 2, 3,
        1, 4, 3, 1, 3, 1, 4, 4, 2, 4, 4, 3, 2, 1, 3, 3, 3, 1, 4, 1, 1, 1, 3, 2,
        4, 4])
```

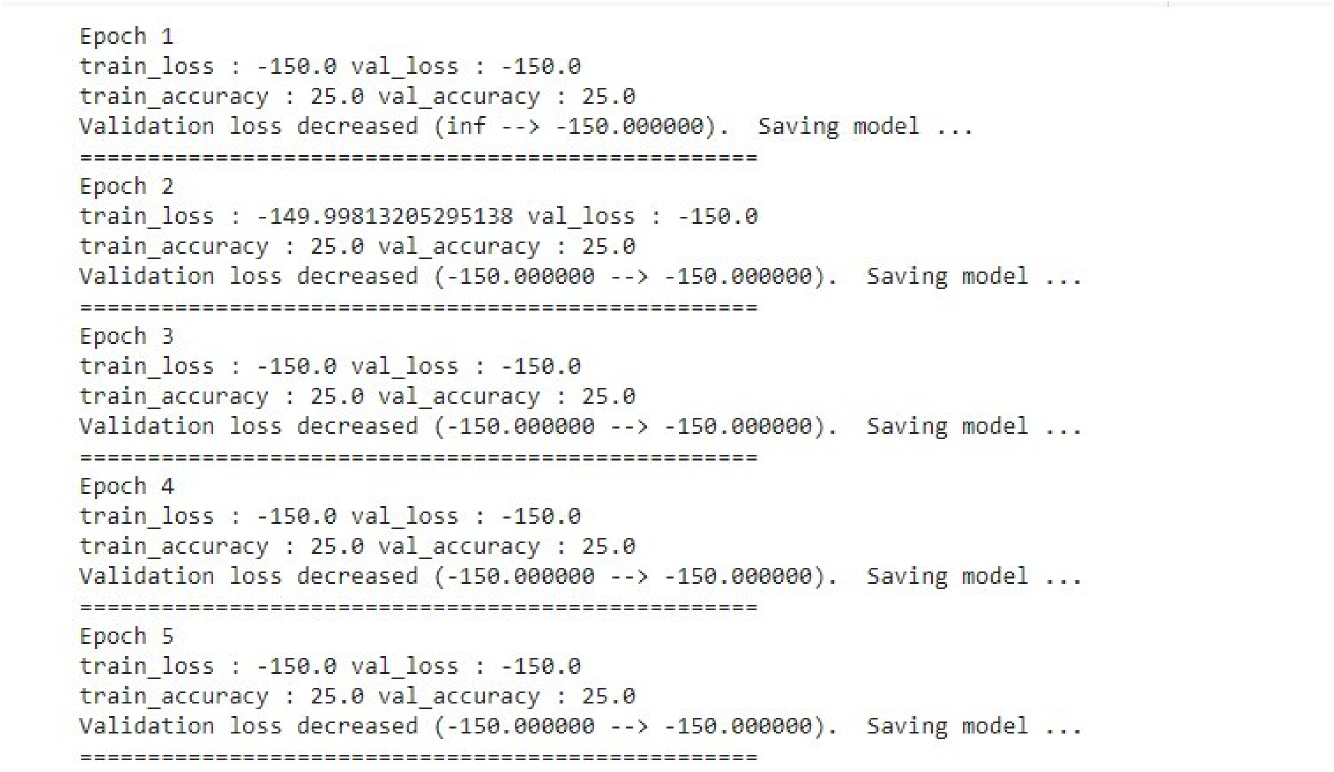
```
######################################################################################

class SentimentRNN(nn.Module):
  def __init__(self,no_layers,vocab_size,hidden_dim,embedding_dim,drop_prob=0.5):
    super(SentimentRNN,self).__init__()

    self.output_dim = output_dim
    self.hidden_dim = hidden_dim

    self.no_layers = no_layers
    self.vocab_size = vocab_size

    # embedding and LSTM layers
    self.embedding = nn.Embedding(vocab_size, embedding_dim)

    #lstm
    self.lstm = nn.LSTM(input_size=embedding_dim,hidden_size=self.hidden_dim,
              num_layers=no_layers, batch_first=True)


    # dropout layer
    self.dropout = nn.Dropout(0.3)

    # linear and sigmoid layer
    self.fc = nn.Linear(self.hidden_dim, output_dim)
    self.sig = nn.Sigmoid()

  def forward(self,x,hidden):
    batch_size = x.size(0)
    # embeddings and lstm_out
    embeds = self.embedding(x)  # shape: B x S x Feature   since batch = True
    #print(embeds.shape)  #[50, 500, 1000]
    lstm_out, hidden = self.lstm(embeds, hidden)

    lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)

    # dropout and fully connected layer
    out = self.dropout(lstm_out)
    out = self.fc(out)

    # sigmoid function
    sig_out = self.sig(out)

    # reshape to be batch_size first
    sig_out = sig_out.view(batch_size, -1)

    sig_out = sig_out[:, -1] # get last batch of labels

    # return last sigmoid output and hidden state
    return sig_out, hidden



  def init_hidden(self, batch_size):
    ''' Initializes hidden state '''
    # Create two new tensors with sizes n_layers x batch_size x hidden_dim,
    # initialized to zero, for hidden state and cell state of LSTM
    h0 = torch.zeros((self.no_layers,batch_size,self.hidden_dim)).to(device)
    c0 = torch.zeros((self.no_layers,batch_size,self.hidden_dim)).to(device)
    hidden = (h0,c0)
    return hidden

#######################################################################################

no_layers = 4
vocab_size = len(vocab) + 1 #extra 1 for padding
embedding_dim = 64
output_dim = 1
hidden_dim = 256


model = SentimentRNN(no_layers,vocab_size,hidden_dim,embedding_dim,drop_prob=0.5)

#moving to gpu
model.to(device)

print(model)



# loss and optimization functions
lr=0.001
```



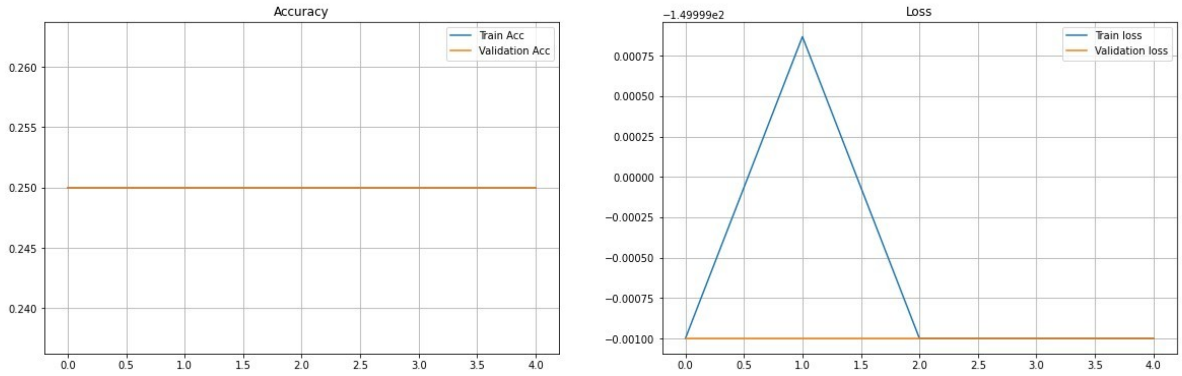
```
SentimentRNN(
  (embedding): Embedding(1001, 64)
  (lstm): LSTM(64, 256, num_layers=2, batch_first=True)
  (dropout): Dropout(p=0.3, inplace=False)
  (fc): Linear(in_features=256, out_features=1, bias=True)
  (sig): Sigmoid()
)
```

```
#######################################################################################

criterion = nn.BCELoss()

optimizer = torch.optim.Adam(model.parameters(), lr=lr)

# function to predict accuracy
def acc(pred,label):
  pred = torch.round(pred.squeeze())
  return torch.sum(pred == label.squeeze()).item()
```

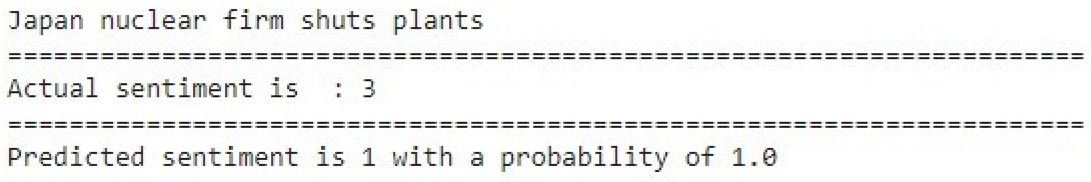
```
clip = 5
epochs = 5
valid_loss_min = np.Inf
# train for some number of epochs
epoch_tr_loss,epoch_vl_loss = [],[]
epoch_tr_acc,epoch_vl_acc = [],[]

for epoch in range(epochs):
  train_losses = []
  train_acc = 0.0
  model.train()
  # initialize hidden state
  h = model.init_hidden(batch_size)
  for inputs, labels in train_loader:

    inputs, labels = inputs.to(device), labels.to(device)
    # Creating new variables for the hidden state, otherwise
    # we'd backprop through the entire training history
    h = tuple([each.data for each in h])

    model.zero_grad()
    output,h = model(inputs,h)

    # calculate the loss and perform backprop
    loss = criterion(output.squeeze(), labels.float())
    loss.backward()
    train_losses.append(loss.item())
    # calculating accuracy
    accuracy = acc(output,labels)
    train_acc += accuracy
    #`clip_grad_norm` helps prevent the exploding gradient problem in RNNs / LSTMs.
    nn.utils.clip_grad_norm_(model.parameters(), clip)
    optimizer.step()



  val_h = model.init_hidden(batch_size)
  val_losses = []
  val_acc = 0.0
  model.eval()
  for inputs, labels in valid_loader:
      val_h = tuple([each.data for each in val_h])

      inputs, labels = inputs.to(device), labels.to(device)

      output, val_h = model(inputs, val_h)
      val_loss = criterion(output.squeeze(), labels.float())

      val_losses.append(val_loss.item())

      accuracy = acc(output,labels)
      val_acc += accuracy

  epoch_train_loss = np.mean(train_losses)
  epoch_val_loss = np.mean(val_losses)
  epoch_train_acc = train_acc/len(train_loader.dataset)
  epoch_val_acc = val_acc/len(valid_loader.dataset)
  epoch_tr_loss.append(epoch_train_loss)
  epoch_vl_loss.append(epoch_val_loss)
  epoch_tr_acc.append(epoch_train_acc)
  epoch_vl_acc.append(epoch_val_acc)
  print(f'Epoch {epoch+1}')
  print(f'train_loss : {epoch_train_loss} val_loss : {epoch_val_loss}')
  print(f'train_accuracy : {epoch_train_acc*100} val_accuracy : {epoch_val_acc*100}')
  if epoch_val_loss <= valid_loss_min:
    #torch.save(model.state_dict(), '../working/state_dict.pt')
    print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...
'.format(valid_loss_min,epoch_val_loss))
    valid_loss_min = epoch_val_loss
  print(25*'==')
```



```
Epoch 1
train_loss : -150.0 val_loss : -150.0
train_accuracy : 25.0 val_accuracy : 25.0
Validation loss decreased (inf --> -150.000000).  Saving model ...
==================================================
Epoch 2
train_loss : -149.99813205295138 val_loss : -150.0
train_accuracy : 25.0 val_accuracy : 25.0
Validation loss decreased (-150.000000 --> -150.000000).  Saving model ...
==================================================
Epoch 3
train_loss : -150.0 val_loss : -150.0
train_accuracy : 25.0 val_accuracy : 25.0
Validation loss decreased (-150.000000 --> -150.000000).  Saving model ...
==================================================
Epoch 4
train_loss : -150.0 val_loss : -150.0
train_accuracy : 25.0 val_accuracy : 25.0
Validation loss decreased (-150.000000 --> -150.000000).  Saving model ...
==================================================
Epoch 5
train_loss : -150.0 val_loss : -150.0
train_accuracy : 25.0 val_accuracy : 25.0
Validation loss decreased (-150.000000 --> -150.000000).  Saving model ...
==================================================
#######################################################################################

fig = plt.figure(figsize = (20, 6))
plt.subplot(1, 2, 1)
plt.plot(epoch_tr_acc, label='Train Acc')
plt.plot(epoch_vl_acc, label='Validation Acc')
plt.title("Accuracy")
plt.legend()
plt.grid()

plt.subplot(1, 2, 2)
plt.plot(epoch_tr_loss, label='Train loss')
plt.plot(epoch_vl_loss, label='Validation loss')
plt.title("Loss")
plt.legend()
plt.grid()

plt.show()
```



```
#######################################################################################

index = 30
print(df['Title'][index])
print('='*70)
print(f'Actual sentiment is  : {df["Class Index"][index]}')
print('='*70)
pro = predict_text(df['Title'][index])
status = "1" if pro > 0.5 else "2"
pro = (1 - pro) if status == "negative" else pro
print(f'Predicted sentiment is {status} with a probability of {pro}')
```



```
Japan nuclear firm shuts plants
======================================================================
Actual sentiment is  : 3
======================================================================
Predicted sentiment is 1 with a probability of 1.0
```

Referanslar

[1] https://www.kaggle.com/arunmohan003/sentiment-analysis-using-lstm-pytorch