CSC 583 HW#3-1: Text Classification using RNNs and PyTorch (Fall 2025)

Start-up Code with some baked outputs

In [1]:
```
## Code piece to mount my Google Drive
from google.colab import drive
drive.mount("/content/drive") # my Google Drive root directory will be mapped here
```

```
Mounted at /content/drive
```

In [ ]:
```
# Change the working directory to your own work directory (where the code file is).
import os
thisdir = '/content/drive/My Drive/CSC583_Fall2025/HW#3-1'
os.chdir(thisdir)

# Ensure the files are there (in the folder)
!pwd
```

Check for GPU's

In [ ]:
```
import torch

# If there's a GPU available...
if torch.cuda.is_available():
    # Tell PyTorch to use the GPU.
    device = torch.device("cuda")
    print ('There are %d GPU(s) available.' % torch.cuda.device_count())
    print ('We will use the GPU:', torch.cuda.get_device_name(0))
# If not...
else:
    print ('No GPU available, using the CPU instead.')
    device = torch.device("cpu")
```

Some important import's

In [2]:
```
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
import string
import re
from tqdm import tqdm
import matplotlib.pyplot as plt
from torch.utils.data import TensorDataset, DataLoader
```

(1) Load datasets

In [3]:
```
# TO-DO (0):
# Load train and test data into pandas dataframe (separately).
# Be sure to specify the encoding to be 'utf-8' and the
# delimiter to be '\t' (tab).









#----------------
# a test call
print (df_train.shape)
print (df_test.shape)
df_train.tail()
```

```
(4447, 3)
(1112, 2)
```

Out[3]:

	id	content	labels
4442	1864	Don’t forget to use N95 mask https://t.co/W5RP...	1
4443	772	President Trump congratulated Japanese Prime M...	1
4444	2454	BREAKING: CUNY announces student at @JohnJayCo...	1
4445	5710	CDC confirms Chinese coronavirus has arrived i...	1
4446	6334	Chinese Premier Li Keqiang arrives in #coronav...	1

Inspect some properties of the datasets

In [24]:
```
# Check the class distribution (Fake/True; stratified) in train and test sets
def binary_ratio(df):
    shape = df.shape
    fakecount = df[(df['labels'] == 0)].shape[0]  # count of fake entries
    print (f'Fake ratio: shape={df.shape} -- fake {fakecount}/{shape[0]} = {fa
kecount/shape[0]}')

binary_ratio(df_train)
```

```
Fake ratio: shape=(4447, 6) -- fake 2945/4447 = 0.6622442095794918
```

In [4]:
```
import statistics

# sentence lengths for the training set
df_train['content_length'] = df_train['content'].apply(lambda x: len(x.split
())) # simple white-space delimiter

# mean and stdev of lengths
lengths = df_train['content_length'].tolist()
print (f'Mean: {statistics.mean(lengths)}, Stdev: {statistics.stdev(lengths)},
Max: {max(lengths)}')

df_train.head()
```

```
Mean: 19.773105464357993, Stdev: 13.641496714680176, Max: 140
```

Out[4]:

	id	content	labels	content_length
0	466	Coronavirus patients are being "cremated alive...	0	8
1	1823	A video shows a creature on top of a dome-like...	0	22
2	4708	A video showing an anti-China protest amid the...	0	12
3	4740	Article suggests African skin and blood is res...	0	10
4	3294	The Brazilian Government is handling a 600 bra...	0	31

(2) Build vocabulary

We build the vocabulary from words/tokens in the training set.

First we define/obtain a tokenizer. We will use a simple white-space-based tokenizer (used in GloVe), which is essentially what NLTK's word_tokenize() does. After tokenization, we will convert text into **lower case** and **remove punctuations and numbers** in addition.

In [5]:
```
!pip install nltk
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: nltk in c:\programdata\anaconda3\lib\site-pack
ages (3.7)
Requirement already satisfied: click in c:\programdata\anaconda3\lib\site-pac
kages (from nltk) (8.0.4)
Requirement already satisfied: tqdm in c:\users\ntomuro\appdata\roaming\pytho
n\python39\site-packages (from nltk) (4.66.2)
Requirement already satisfied: joblib in c:\programdata\anaconda3\lib\site-pa
ckages (from nltk) (1.1.0)
Requirement already satisfied: regex>=2021.8.3 in c:\programdata\anaconda3\li
b\site-packages (from nltk) (2022.3.15)
Requirement already satisfied: colorama in c:\programdata\anaconda3\lib\site-
packages (from click->nltk) (0.4.4)
```

In [6]:
```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ntomuro\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[6]:
```
True
```

In [7]:
```
# Check NLTK's word_tokenize() function.
from nltk import word_tokenize

sent = "Congress has passed a US$8.3 billion coronavirus response bill, which
includes $2.2 billion for the CDC to â€œprevent, prepare for, and respond to c
oronavirus, domestically or internationally.â€"
#"Text classification is a fundamental natural language processing (NLP) tas
k."

tokens = word_tokenize(sent)
print (tokens)
```

```
['Congress', 'has', 'passed', 'a', 'US', '$', '8.3', 'billion', 'coronaviru
s', 'response', 'bill', ',', 'which', 'includes', '$', '2.2', 'billion', 'fo
r', 'the', 'CDC', 'to', 'â€œprevent', ',', 'prepare', 'for', ',', 'and', 'res
pond', 'to', 'coronavirus', ',', 'domestically', 'or', 'internationally.â€\x9
d']
```

In [8]:
```
# Our tokenizer function
def tokenize (text):
    # first clean up the text by replacing non-ascii characters to a space
    text = re.sub(r"[^\x00-\x7F]+", " ", text)

    # TO-DO (1):
    # Continue to tokenize text. You do these in ANY ORDER: 1. removing punctu
ations,
    # 2. removing numbers, 3. changing text to lower case, 4. tokenize text in
to tokens
    # (by word_tokenize()).  Return the tokens in a list.




#----------------
# a test call
print (tokenize(sent))
```

```
['congress', 'has', 'passed', 'a', 'us', 'billion', 'coronavirus', 'respons
e', 'bill', 'which', 'includes', 'billion', 'for', 'the', 'cdc', 'to', 'preve
nt', 'prepare', 'for', 'and', 'respond', 'to', 'coronavirus', 'domestically',
'or', 'internationally']
```

Tokenize each text and save results in a new column 'content_tokenized' in the dataframe

In [9]:
```
# TO-DO (2):
# Apply tokenizer to each text in 'content' in df_train.
# Store the results in a new column 'content_tokenized'.
# (*) Be sure to nest the output in np.array (to make a list of one element)
# because pandas df does not accept arrays of different length (i.e., jagged
# arrays) or arrays of strings (in our case, tokens).




#----------------
# a test call
df_train.head()
```

Out[9]:

	id	content	labels	content_length	content_tokenized
0	466	Coronavirus patients are being "cremated alive...	0	8	[[coronavirus, patients, are, being, cremated,...
1	1823	A video shows a creature on top of a dome-like...	0	22	[[a, video, shows, a, creature, on, top, of, a...
2	4708	A video showing an anti-China protest amid the...	0	12	[[a, video, showing, an, anti, china, protest,...
3	4740	Article suggests African skin and blood is res...	0	10	[[article, suggests, african, skin, and, blood...
4	3294	The Brazilian Government is handling a 600 bra...	0	31	[[the, brazilian, government, is, handling, a,...

Collect tokens and store them in NLTK's FreqDist dictionary

In [14]:
```
# function to flatten a nested list to a flat list
def flatten(sents):
  # assuming the nesting level of 2..
  return [token for sent in sents for token in sent]
```

In [15]:
```
# Collect tokenized results into a list
all_tokens_list = [wlist[0] for wlist in df_train['content_tokenized'].tolist
()]
token_list = flatten(all_tokens_list)

# NLTK's FreqDist
fdist = nltk.probability.FreqDist(token_list)
print (fdist)
```

```
<FreqDist with 10272 samples and 89673 outcomes>
```

Finalize vocabulary as tokens that occurred >= 2 times, plus "" and 'UNK'

In [16]:
```
# TO-DO (3):
# Select tokens that appeared >= 2 times, and sort them.
# Merge them with the list ["", 'UNK'].  Note "" for padding
# and 'UNK' for unknown tokens.  Name the final vocabulary as 'voc'.









#----------------
# a test call
vocab_size = len(vocab) # this variable will be used later
print (f'vocabulary_size: {vocab_size}')
print (vocab[:10])
```

```
vocabulary_size: 4922
['', 'UNK', 'a', 'aa', 'ab', 'ababa', 'abandon', 'abandoned', 'abascal', 'abb
ott']
```

Create vocabulary lookup tables as well

In [17]:
```
# Vocabulary lookup tables
vocab2index = {} # token to index lookup
index2vocab = {} # index to token (reverse) lookup

for idx, token in enumerate(vocab):
    vocab2index[token] = idx
    index2vocab[idx] = token
```

Encode each text (token -> idex) and save results in a new column in the dataframe.

Text is truncated to the maximum input length (**max_input_len**). Also, tokens that are not in the vocabulary are indicated with 'UNK'.

In [18]:
```
max_input_len = 50 # this variable will be used later too

# Returns a numpy array of tokens of a _fixed_ size (N -- defaults to 'max_inp
ut_len')
def encode_sentence(tokenized_text, vocab2index, N=max_input_len):
    # TO-DO (4):
    # Create a list of vocabulary indices for the tokens in 'tokenized_text'
(NOT nested)
    # (e.g. ['a' 'video' 'showing' 'an' 'anti' 'china' 'protest' 'amid' 'th
e'..])
    # and return the list (in a non-nested, fixed size (N) numpy array).
    # Assume the token indices are recorded in 'vocab2index' dictionary.
    # (*) If a token is not in the vocabulary, the index associated with 'UNK'
    # should be selected for the token.
    # (*) If the length of the 'tokenized_text' is longer than N, it will be t
runcated.
    # Or if the length is shorter, the remaining slots in the resulting index
list
    # should be padded with 0's.
    # Return the vocabulary index list (i.e., encoded list) and it's length.
```

In [19]:
```
# each entry in df_train['encoded'] has the same/fixed length of 'max_input_le
n' (and the
#  remainders are filled with index 0 -- the padding character)
df_train['encoded'] = df_train['content_tokenized'].apply(lambda x: encode_sen
tence(x.tolist()[0], vocab2index)[0])
print (f'{df_train.loc[2].encoded}, \n{df_train.loc[2].content_tokenized}')
print (f'{df_train.loc[4].encoded}, \n{df_train.loc[4].content_tokenized}')
df_train.head()
```

```
[   2 4668 3940  192  215  748 3381  186 4365 1019 3033 2121 2252    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0],
[['a' 'video' 'showing' 'an' 'anti' 'china' 'protest' 'amid' 'the'
  'covid' 'outbreak' 'in' 'italy']]
[4365  545 1848 2231 1911    2  545 3499 2755 1019 1409    1 4428 3240
  775 4428   31 2248 4901 2857 4428 1638 3031  192    1 1712 2464 4428
 4365 1712    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0],
[['the' 'brazilian' 'government' 'is' 'handling' 'a' 'brazilian' 'reais'
  'monthly' 'covid' 'emergency' 'allowance' 'to' 'poor' 'citizens' 'to'
  'access' 'it' 'you' 'need' 'to' 'fill' 'out' 'an' 'oficial' 'form'
  'link' 'to' 'the' 'form']]
```

Out[19]:

	id	content	labels	content_length	content_tokenized	encoded
0	466	Coronavirus patients are being "cremated alive...	0	8	[[coronavirus, patients, are, being, cremated,...	[974, 3113, 254, 421, 1035, 153, 2121, 748, 0,...
1	1823	A video shows a creature on top of a dome-like...	0	22	[[a, video, shows, a, creature, on, top, of, a...	[2, 4668, 3942, 2, 1, 2985, 4442, 2956, 2, 1, ...
2	4708	A video showing an anti-China protest amid the...	0	12	[[a, video, showing, an, anti, china, protest,...	[2, 4668, 3940, 192, 215, 748, 3381, 186, 4365...
3	4740	Article suggests African skin and blood is res...	0	10	[[article, suggests, african, skin, and, blood...	[277, 4213, 101, 3982, 196, 491, 2231, 3625, 4...
4	3294	The Brazilian Government is handling a 600 bra...	0	31	[[the, brazilian, government, is, handling, a,...	[4365, 545, 1848, 2231, 1911, 2, 545, 3499, 27...

(**) Do the same preprocessing steps for the test set (using the vocabulary constructed from the training set)

In [20]:
```
# TO-DO (5):
# First obtain sentence lengths for each content entry and assign to a new col
umn 'content_length'



# Next tokenize each content and save the tokenized tokens in a new column 'co
ntent_tokenized'



# Then encode the text (into indices)




#----------------
# a test call
df_test.head()
```

Out[20]:

	id	content	content_length	content_tokenized	encoded
0	2	The health experts had predicted the virus cou...	15	[[the, health, experts, had, predicted, the, v...	[4365, 1952, 1529, 1895, 3289, 4365, 4689, 998...
1	11	Japanese doctors advice that taking a few sips...	24	[[japanese, doctors, advice, that, taking, a, ...	[2269, 1280, 85, 4364, 4294, 2, 1627, 3969, 29...
2	16	Gargling with salt water or Vinegar 'eliminate...	18	[[gargling, with, salt, water, or, vinegar, el...	[1775, 4832, 3759, 4753, 3007, 4678, 1400, 436...
3	20	Washing your hands decreases the number of mic...	19	[[washing, your, hands, decreases, the, number...	[4746, 4904, 1912, 1, 4365, 2929, 2956, 1, 298...
4	46	The fictional "Umbrella Corporation" from the ...	31	[[the, fictional, umbrella, corporation, from,...	[4365, 1, 4560, 988, 1744, 4365, 1768, 3620, 1...

(3) Create PyTorch Datasets and DataLoaders

We first define a custom 'MyDataset' class

In [21]:
```
from torch.utils.data import Dataset, DataLoader

class MyDataset(Dataset):
    def __init__(self, X, Y):
        self.X = X
        self.y = Y

    def __len__(self):
        return len(self.y)

    def __getitem__(self, idx):
        # returns a torch tensor (possibly from a numpy array)
        return torch.from_numpy(self.X[idx].astype(np.int32)), self.y[idx]
```

In [26]:
```
# Prepare train/validation/test data
x = df_train['encoded'].tolist()
y = df_train['labels'].tolist()
x_test = df_test['encoded'].tolist()

# TO-DO (6):
# Split the training ('x' and 'y' above) into train and validation, with 20% f
or validation.
# Be sure to split using stratification.
# Name the resulting variables as x_train, x_valid, y_train, y_valid.




#----------------
# a test call
print (f'Training contains {len(x_train)} instances; Validation contains {len
(x_valid)} instances')
```

```
Training contains 3557 instances; Validation contains 890 instances
```

In [28]:
```
# Then create custom Datasets
train_ds = MyDataset(x_train, y_train)
valid_ds = MyDataset(x_valid, y_valid)
```

(*) Dataloaders

In [29]:
```
batch_size = 64

train_dataloader = DataLoader(train_ds, batch_size=batch_size, shuffle=True)
valid_dataloader = DataLoader(valid_ds, batch_size=batch_size, shuffle=True)
```

(4) Model

In [30]:
```
class MyLSTM(torch.nn.Module) :
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim, n_la
yers=1,
                 bidirectional=False, dropout=0.0):
        super().__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim, padding_idx=
0)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers=n_layers,
                            bidirectional=bidirectional, dropout=dropout, batch
_first=True)
        self.linear = nn.Linear(hidden_dim, output_dim)
        self.dropout = nn.Dropout(0.3)
        # activation function for the output layer -- for binary/logistic clas
sification
        self.act = nn.Sigmoid()

    def forward(self, x):
        x = self.embeddings(x)
        lstm_out, (ht, ct) = self.lstm(x)
        out = self.linear(ht[-1])
        out = self.dropout(out)
        return self.act(out)
```

(5) Training -- train and eval functions

In [31]:
```
# function to predict accuracy (or number of correctly classified instances)
def acc(pred,label):
    pred = torch.round(pred.squeeze())
    return torch.sum(pred == label.squeeze()).item()

#----------------------------
# function to train the model
#----------------------------
def train_model(model, epochs=10, lr=0.001, weight_decay=1e-5):
    # define optimizer (Adam, for parameters that require gradient)
    parameters = filter(lambda p: p.requires_grad, model.parameters())
    optimizer = torch.optim.Adam(parameters, lr=lr, weight_decay=weight_decay)
    #
    valid_loss_min = np.Inf

    # save the initial model
    torch.save(model.state_dict(), init_model_path) # current best model

    ## training loop - for each epoch
    for epoch in range(epochs):
        ##======== (1) Training ========
        # (*) set the mode to train
        model.train()
        # results accumulator variables
        train_losses = [] # trace of losses (over batches)
        train_acc = 0.0   # total number of correctly classified instances

        # iterate over mini-batches
        for inputs, labels in train_dataloader:
            # push them to the GPU
            inputs, labels = inputs.to(device), labels.to(device)
            # (*) clear the gradients
            optimizer.zero_grad()
            # forward propagate to obtain prediction
            output = model(inputs)

            # compute loss
            loss = criterion(output.squeeze(), labels.float())
            # backward propagation
            loss.backward()

            # record the loss (by appending the value to the list of losses)
            train_losses.append(loss.item())
            # calculating accuracy (accumulate correct count)
            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)
            # update the weights
```

```
            optimizer.step()

        ##======== (2) Evaluation ========
        val_losses, val_acc = evaluate(model, valid_dataloader)

        ##======== (3) Reporting ========
        epoch_train_loss = np.mean(train_losses)
        epoch_val_loss = np.mean(val_losses)
        epoch_train_acc = train_acc/len(train_dataloader.dataset)
        epoch_val_acc = val_acc/len(valid_dataloader.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_v
al_acc*100}')

        if epoch_val_loss <= valid_loss_min:
            torch.save(model.state_dict(), best_model_path) # current best mod
el
            print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving mode
l ...'.format(valid_loss_min,epoch_val_loss))
            valid_loss_min = epoch_val_loss
        print(25*'==')

#----------------------------
# function to evaluate the model
#----------------------------
def evaluate(model, valid_dl):
    # (*) set the mode to evaluation
    model.eval()
    #
    val_losses = [] # trace of losses (over batches)
    val_acc = 0.0   # total number of correctly classified instances

    #deactivate autograd since it's not needed during evaluation
    with torch.no_grad():
        # TO-DO (7):
        # Evaluate the model with respect to the validation dataset ('valid_d
l').









    return val_losses, val_acc
```

Define the loss function

In [32]:
```
import torch.optim as optim

#define the loss function -- binary cross-entropy
# (https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html#torch.nn.BCELoss)
criterion = nn.BCELoss()

#push to cuda if available
criterion = criterion.to(device)
```

Create a model and set up other parameters

In [33]:
```
embedding_dim = 50
output_dim = 1
hidden_dim = 128
clip = 5

# train for some number of epochs
epoch_tr_loss,epoch_vl_loss = [],[]
epoch_tr_acc,epoch_vl_acc = [],[]

# paths to save models
init_model_path ='./saved/init_model.pt'
best_model_path ='./saved/best_state_model.pt'

# Create a model
model = MyLSTM(vocab_size, embedding_dim, hidden_dim, output_dim, dropout=0.2)

#moving the model to gpu
model.to(device)

print(model)
```

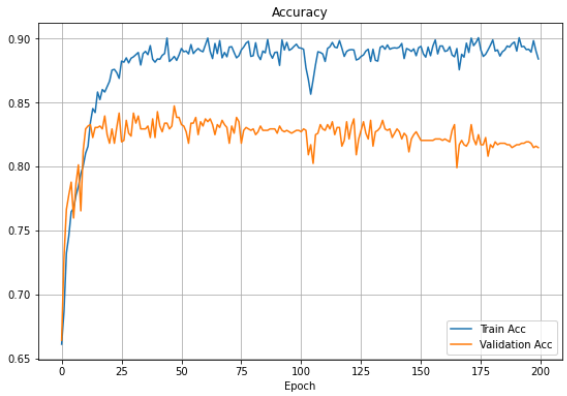
```
C:\ProgramData\Anaconda3\lib\site-packages\torch\nn\modules\rnn.py:62: UserWarning: dropout option adds dropout after all but last recurrent layer, so non-zero dropout expects num_layers greater than 1, but got dropout=0.2 and num_layers=1
  warnings.warn("dropout option adds dropout after all but last "

MyLSTM(
  (embeddings): Embedding(4922, 50, padding_idx=0)
  (lstm): LSTM(50, 128, batch_first=True, dropout=0.2)
  (linear): Linear(in_features=128, out_features=1, bias=True)
  (dropout): Dropout(p=0.3, inplace=False)
  (act): Sigmoid()
)
```

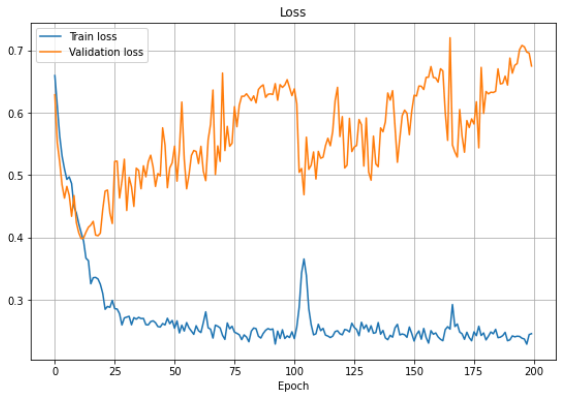
Finally train the model

In [ ]:
```
# TO-DO (8):
# Try various number of epochs and lr, as well as model architecture
# parameters (e.g. number of layers, bidirectional, recurrent drop-out etc.)
train_model(model, epochs=200, lr=0.0005)
```

Visualize training

In [35]:
```
# TO-DO (9):
# Visualize the training results.  You can do more epochs, but be sure to
# compare training and validation accuracies and losses.
```





(6) Prediction/Inference with the test set

First load the saved best model and define the inference function that accepts the user defined input and make predictions ('./RNN-references/Text%20Classification%20Pytorch%20_%20Build%20Text%20Classifica

In [36]:
```
#load weights from the saved best model
model = MyLSTM(vocab_size, embedding_dim, hidden_dim, output_dim)
model.to(device)

model.load_state_dict(torch.load(best_model_path))
model.eval()  # set the mode to eval (i.e., no gradient)
```

Out[36]:
```
MyLSTM(
  (embeddings): Embedding(4922, 50, padding_idx=0)
  (lstm): LSTM(50, 128, batch_first=True)
  (linear): Linear(in_features=128, out_features=1, bias=True)
  (dropout): Dropout(p=0.3, inplace=False)
  (act): Sigmoid()
)
```

In [30]:
```
#----------------------------
# function to generate predictions for the testset
#----------------------------
def predict(model, test_list):
    # (*) set the mode to evaluation
    model.eval()
    #
    prediction_list = [] # store predictions

    with torch.no_grad(): #deactivates autograd
        # TO-DO (10):
        # Obtain prediction for each instance in the test set/list
        # and accumulate them in 'prediction_list'.









    # Return 'prediction_list'
    return prediction_list
```

In [ ]:
```
##------------------------------
## Inference/generate predictions
##------------------------------
predictions = predict(model, x_test)
```

Write test predictions to a csv file (for Kaggle submission)

In [34]:
```
# TO-DO (11):
# Write your own code.
```