HW 4 – Multi-label Classification (20 Points)

Required Submission files:

Submit the following two files:

1. **FirstName_LastName_HW4A.ipynb**
2. **FirstName_LastName_HW4A.pdf (pdf version of the above file)**
3. **FirstName_LastName_HW4B.ipynb**
4. **FirstName_LastName_HW4B.pdf (pdf version of the above file)**

TASK: Multilabel Classification

In this HW, you will identify tags for stack exchange Questions. This data is a subset of data available in Kaggle Competitions. The given dataset actually contains the different questions asked in the StackExchange website for various tech domains. We have fetched only those question that contains top 10 individual tags. Each question can have more than one tag. **This means that this is a multi-label classification problem.** These are the ten categories for tags in the data.

0	c#
1	java
2	php
3	javascript
4	android
5	jquery
6	c++
7	python
8	iphone
9	asp.net

Data:

- The dataset is available from eLearning : **0_Data folder** from the file: **df_multilabel_hw_cleaned.joblib**.
- The dataset is a pandas dataframe. You can use joblib.load(file) to load the dataset.
- No preprocessing is required, you are provided with the cleaned dataset.
- You will use this data set to train your classification model.
- Hint1: The Tag_Number column is your labels. However, the data type of list elements is object. To convert it to integers; use ast.literatl_eval() function from ast library. You will need to import this library first.
- Hint2: Since this is a multilabel dataset – you will need to do the one-hot encoding of your dependent variable. You can use MultiLabelBinarizer from sklearn.

- **You will create train/valid/test splits from this data set.** Use 60% for train, 20 % for the test, and the remaining 20% for the validation dataset.

Part A (Neural Network in PyTorch using Dense Embeddings) (10 Points)

- Your Network should have the following layers
  EmbeddingBag → Hidden_Layer1 →ReLU → Dropout_Layer1 → BatchNorn_Layer1 → Hidden_Layer2 → ReLU → DropoutLayer2 → BatchNorm_Layer2 → Output Layer
- You will use the following hyperparameters to train our model:
  - Neurons for first hidden layer = 200
  - Neurons for second hidden layer = 100
  - Embedding Dimension = 300
  - EPOCHS=5
  - BATCH_SIZE=128
  - LEARNING_RATE=0.001
  - WEIGHT_DECAY=0.000
  - CLIP_TYPE = 'value'
  - CLIP_VALUE = 10
  - PATIENCE=5
  - Optimizer = AdamW

Notes:

- For Gradient Clipping. you will need to add the following line in the step function we discussed in class.

```
loss.backward()

clip_grad_value_(model.parameters(), clip_value=10.0)

optimizer.step()
```

- Instead of accuracy use Hamming Loss as a metric, since this is a multilabel classification problem. **Hint (see Task 4 of HW3 for reference).**
- You can **use 7_imdb_complete_example.ipynb** from Lecture 4 as a starting point. ***You will need to make changes to the Training Function to adapt it to the multilabel Classification problem. The accuracy and prediction should be calculated differently (see Task 4 of HW3 for reference).***
- Your code should have all the components –(1) Model Training (Training and evaluation on Validation set), (2) Model Testing (evaluation on the test set), and (3) Inference

Part B (Neural Network in PyTorch using Sparse Embeddings) (10 Points)

- **You will create train/valid/test splits from this data set.** Use 60% for train, 20 % for the test, and the remaining 20% for the validation dataset.
- In this part, you will use the bag of word approach (tfidf vectorizer) to convert text to tensors. You can use max_features = 5000 for tfidf. Use fit on Train and transform on valid and test.
- The tfidf vectors are your X features.
- You will need to do the one-hot encoding of your dependent variable to get y.
- Create Pytorch Datasets using X and y (create three dataset train/test/valid using corresponding X and y for each split).
- Create Data loaders.

Model

- Your Network should have the following layers

  Hidden_Layer1 →ReLU → Dropout_Layer1 → BatchNorn_Layer1 → Hidden_Layer2 → ReLU → DropoutLayer2 → BatchNorm_Layer2 → Output Layer

Since we have created sparse embeddings using tfidf vectorizer. *We do no need embedding bag layer. We will also do not need a collate function.*

- **In the first cell of the notebook, explain why we do not need a collate function.**

- You will use the following hyperparameters to train our model:

  - `Neurons for first hidden layer = 200`
  - `Neurons for second hidden layer = 100`
  - `EPOCHS=5`
  - `BATCH_SIZE=128`
  - `LEARNING_RATE=0.001`
  - `WEIGHT_DECAY=0.000`
  - `CLIP_TYPE = 'value'`
  - `CLIP_VALUE = 10`
  - `PATIENCE=5`
  - `Optimizer =` AdamW

Notes:

- For Gradient Clipping. you will need to add the following line in the step function we discussed in class.

```
loss.backward()

clip_grad_value_(model.parameters(), clip_value=10.0)
```

```
optimizer.step()
```

- Instead of accuracy use Hamming Loss as a metric, since this is a multilabel classification problem. **Hint (see Task 4 of HW3 for reference).**

- You can **use 7_imdb_complete_example.ipynb** from Lecture 4 as a starting point. ***You will need to make changes to the Training Function to adapt it to the multilabel Classification problem. The accuracy and prediction should be calculated differently (see Task 4 of HW3 for reference).***

- Your code should have all the components –(1) Model Training (Training and evaluation on the Validation set), (2) Model Testing (evaluation on the test set), and (3) Inference