CS4248
AY 2022/23 Semester 1
Assignment 2

Due Date

Friday 14 October 2022 at 9am. This assignment consists of two parts, and both need to be submitted through CodeCrunch before the due date. Late assignments will not be accepted and will receive ZERO marks. This is an individual assignment, do not copy your code from any sources.

PART 1

Objective

The first part of this assignment will guide you through PyTorch tensors. Your implementation will be carried out to meet the goal of the function described in this assignment. Your implementation of the functions needs to be done exclusively in PyTorch and basic Python operations, i.e., the only import statement is `import torch`. You are provided with skeletal code `a2part1.py`. Please **do not** change the functions' **identifiers** (name, parameters, etc.), and do not change the **file name** to make sure your assignment can be graded. Failure to do so will give you ZERO mark.

1. Given the shape (dimension) of the tensor, generate a tensor where all its elements are the number one.

```
Q1 example input:(2, 3)

Q1 example output:
tensor([[1., 1., 1.],
        [1., 1., 1.]])
```

2. PyTorch tensors can have different kinds of data types. Define a function to convert a Python list into a PyTorch tensor with torch.long data type.

```
Q2 example input:
[[1., 2.1, 3.0], [4., 5., 6.2]]

Q2 example output:
tensor([[1, 2, 3],
        [4, 5, 6]])
```

3. Define a function that accepts two tensors *a* and *b* with the same shape, and produce a new tensor that represents the result of *3 * a + 2 * b*.

```
Q3 example input:
a = tensor([[1, 2, 3],
            [4, 5, 6]])
b = tensor([[1, 1, 1],
            [1, 1, 1]])
Q3 example output:
tensor([[ 5,  8, 11],
        [14, 17, 20]])
```

4. Define a function to get the last column from a 2-dimensional tensor.

```
Q4 example input:
tensor([[1, 2, 3],
        [4, 5, 6]])

Q4 example output:
tensor([3, 6])
```

5. Sometimes you need to combine many tensors into a single tensor at a new dimension, especially in batched neural network training. Define a function to combine a list of n tensors $t_1, t_2, \cdots, t_n \in \mathbb{R}^{d_1 \times d_2 \times \ldots \times d_m}$, into a single tensor $t \in \mathbb{R}^{d_1 \times d_2 \times \ldots \times d_m \times n}$ at the last dimension. Afterward, expand the tensor dimension on the first dimension, making $t \in \mathbb{R}^{1 \times d_1 \times d_2 \times \ldots \times d_m \times n}$.

```
Q5 example input:
[tensor([1, 1]), tensor([3, 6]), tensor([11, 20])]

Q5 example output:
tensor([[[ 1,  3, 11],
         [ 1,  6, 20]]])
```

6. All elements in a tensor must have the same length. When combining tensors with different lengths, we usually pad the shorter tensors with padding elements so that all tensors have the same length. Define a function to combine a list of 1-D tensors with different lengths into a new tensor by padding the shorter tensors with 0 on the left side.

```
Q6 example input:
[tensor([1]), tensor([2, 2]), tensor([3, 3, 3])]

Q6 example output:
tensor([[0, 0, 1],
        [0, 2, 2],
        [3, 3, 3]])
```

7. A neural network contains many linear operations which are usually computed through matrix multiplications. Given the input **Y**, **W**, and **B** in the type of PyTorch tensors, define a function that calculates $\mathbf{W^T} \times \mathbf{(Y - B)}$.

```
Q7 example input:
y: tensor([[1.1200, 1.5700, 2.1100],
           [0.2300, 0.7200, 1.1900],
           [0.5200, 0.4000, 0.3100]])

w: tensor([[ 0.3000,  0.2000],
           [ 0.6000, -0.1000],
           [-0.3000,  0.2000]])

b: tensor([[ 0.0200, -0.0300,  0.0100],
           [ 0.0300,  0.0200, -0.0100],
           [ 0.0200,  0.0000,  0.0100]])

Q7 example output:
tensor([[0.3000, 0.7800, 1.2600],
        [0.3000, 0.3300, 0.3600]])
```

8. To improve efficiency, matrix computation is usually done in a batch consist of b tensors at the same time. Given the input $\mathbf{Y} \in \mathbb{R}^{b\times n\times k}$, $\mathbf{W} \in \mathbb{R}^{b\times n\times m}$, , $\mathbf{B} \in \mathbb{R}^{b\times n\times k}$, calculate $\mathbf{W_i^T} \times (\mathbf{Y_i} - \mathbf{B_i})\ for\ i \in \{1, 2, \dots, b\}$. As the first dimension denotes the batch, $\mathbf{W^T} \in \mathbb{R}^{b\times m\times n}$. Implement the function in a single PyTorch multiplication and subtraction step without any loop.

```
Q8 example input:
y: tensor([[[ 1.1200,  1.5700,  2.1100],
            [ 0.2300,  0.7200,  1.1900],
            [ 0.5200,  0.4000,  0.3100]],

           [[ 0.3100,  0.3600,  0.3000],
            [ 0.8200,  0.5100,  0.7800],
            [ 0.2100, -0.0100,  0.2200]]])

w: tensor([[[ 0.3000,  0.2000],
            [ 0.6000, -0.1000],
            [-0.3000,  0.2000]],

           [[-0.6000,  0.5000],
            [ 0.1000,  0.2000],
            [ 0.4000, -0.2000]]])
b: tensor([[[ 0.0200, -0.0300,  0.0100],
            [ 0.0300,  0.0200, -0.0100],
            [ 0.0200,  0.0000,  0.0100]],

           [[ 0.0100, -0.0400,  0.0000],
            [ 0.0200,  0.0100, -0.0200],
            [ 0.0100, -0.0100,  0.0200]]])

Q8 example output:
tensor([[[ 0.3000,  0.7800,  1.2600],
         [ 0.3000,  0.3300,  0.3600]],

        [[-0.0200, -0.1900, -0.0200],
         [ 0.2700,  0.3000,  0.2700]]])
```

9. Given a 3-D tensor $\mathbf{X} \in \mathbb{R}^{b\times n\times m}$ that contains paddings, calculate the average of the tensor elements along the last dimension without accounting for the paddings (0 values).

```
Q9 example input:
tensor([[[1.0000, 0.0000, 0.0000],
         [1.2000, 0.0000, 0.0000]],

        [[2.0000, 2.2000, 0.0000],
         [2.2000, 2.6000, 0.0000]],

        [[3.0000, 3.2000, 3.1000],
         [3.2000, 3.4000, 3.6000]]])
Q9 example output:
tensor([[1.0000, 1.2000],
        [2.1000, 2.4000],
        [3.1000, 3.4000]])
```

10. Given a list of n pairs of vectors with the same length (in the type of Python list) $[(t_{11}, t_{12}), \cdots, (t_{n1}, t_{n2})]$ where $t_{11}, \ldots, t_{n2} \in \mathbb{R}^d$, define a function that calculates the Euclidean distance of each vector pair and return them as a 1-D tensor $t \in \mathbb{R}^n$

```
Q10 example input:
[([1, 1, 1], [2, 2, 2]),
 ([1, 2, 3], [3, 2, 1]),
 ([0.1, 0.2, 0.3], [0.33, 0.25, 0.1])]

Q10 example output:
tensor([1.7321, 2.8284, 0.3089])
```

PART 2

Objective

In this assignment, you are to write a program in Python 3.8 to perform a language identification task. Given a text, the task is to assign a language ID, one of eng (English), deu (German), fra (French), ita (Italian), or spa (Spanish).

For example:

Input	Output
This arrondissement is the least geographically extended	eng
Der Rheinradweg ist ein Radfernweg, der durch vier Länder auf 1300 km	deu
À ce jour, cet immeuble emblématique géré par la SA d'HLM	fra
New York è stata descritta, dai consolati diplomatici islandesi e lettoni	Ita
La tarde fue pasada por Nerón recitando un poema	spa

You are to implement a neural network model with bigram embeddings, a hidden layer, and an output layer, using PyTorch version 1.9.0. This part of the assignment comes with a skeletal code `a2part2.py`. You are required to use the skeletal code and make sure that your code runs correctly on any of the xgpg0 to xgpg2 servers.

Approach

In this assignment, you are to implement a feed-forward network with a non-linear activation function in the hidden layer. The neural network uses bigram features to predict the language of the text. Given a text with *n* characters, $c_{1:n} = \{c_1, \dots, c_n\}$, where c_i is the character in the *i*-th position, the model will first generate bigrams $x_{1:n-1} = \{x_1, \dots, x_{n-1}\}$ where $x_i = c_i c_{i+1}$ then assign a label *y* with the probability of $p(y|x_{1:n-1})$.

Each bigram x_i is represented by an embedding vector $\mathbf{x}_i = \mathbf{E}_{[\text{idx}(x_i)]} \in \mathbb{R}^d$, obtained from a lookup table (embedding matrix) $\mathbf{E} \in \mathbb{R}^{|V| \times d}$, in which d is the vector dimension, V is the vocabulary, and the operator $\text{idx}(x)$ maps a bigram x to a unique integer index, which corresponds to the row in $\mathbf{E}$ that contains the embedding vector for x. Thus, after the embedding operation, the text with *k*-length bigram sequence ($k = n - 1$) is represented as

$$\mathbf{x} = [\mathbf{x}_1; \mathbf{x}_2; \dots; \mathbf{x}_k]$$

Next, we average the bigram embeddings to get a single vector $\mathbf{h}_0 = \frac{\mathbf{1}}{\boldsymbol{k}} \sum_{j=1}^{k} \mathbf{x}_j \in \mathbb{R}^d$.

Then, we project this vector to an m-dimension vector through a linear layer ($\mathbf{W}_1 \in \mathbb{R}^{d \times m}$, $\boldsymbol{b}_1 \in \mathbb{R}^m$) with Rectified Linear Unit (ReLU) activation function to acquire $\mathbf{h}_1$.

$$\mathbf{h}_1 = ReLU(\mathbf{h}_0 \times \mathbf{W}_1 + \boldsymbol{b}_1)$$

We apply dropout to $\mathbf{h}_1$ during training, and calculate the probability of each language class through a linear layer ($\mathbf{W}_2 \in \mathbb{R}^{m \times |L|}$, $\boldsymbol{b}_2 \in \mathbb{R}^{|L|}$) with L is the set of language IDs. Lastly, we apply softmax to the linear layer output to compute the class probability distribution.

$$\mathbf{p} = \text{softmax}(\mathbf{h}_1 \times \mathbf{W}_2 + \boldsymbol{b}_2)$$

The loss function to use is cross-entropy loss.

Training and Testing

You are provided with a training set of sentences (`x_train.txt`) and the respective language IDs (`y_train.txt`). You will train your neural network using this training set. The command for training the model is:

```
python3.8 a2part2.py --train --text_path x_train.txt --label_path y_train.txt --model_path model.pt
```

The file `model.pt` is the output of the training process that contains the model weight from training as well as other information that you need to save for testing.

You are also supplied with a test set (`x_test.txt`). The command to test on this test file and generate an output file `out.txt` is:

```
python3.8 a2part2.py --test --text_path x_test.txt --model_path model.pt --output_path out.txt
```

Your output file `out.txt` must contain the language ID in the same format as the `y_train.txt` file.

The following command calculates the accuracy of your language identification model, where `out.txt` is the output file of your code and `y_test.txt` is the actual language id of `x_test.txt`

```
python3.8 eval.py out.txt y_test.txt
```

You are provided with the scoring code `eval.py`.

DELIVERABLES

The same commands described above will be executed to evaluate your model. Grading will be done after the submission deadline, by testing your model on a set of new, blind test texts.

You will need to submit your files, `a2part1.py` and `a2part2.py`, via CodeCrunch. Please do not change the file names and do not submit any other files. Use the skeletal code `a2part1.py` and `a2part2.py` released together with this assignment.

The URL of CodeCrunch is as follows:
https://codecrunch.comp.nus.edu.sg/index.php

Login to CodeCrunch with your NUSNET ID and choose the task 'CS4248 (22/23 Sem 1) A2 Part 1' under the CS4248 module to submit your `a2part1.py` and 'CS4248 (22/23 Sem 1) A2 Part 2' to submit your `a2part2.py`.

After submission, you can find the accuracy of your part 2 model on the 'My Submissions' tab after execution is completed. The test file used in CodeCrunch is `x_test.txt`. We will run your code on the SoC cluster nodes xgpg0 to xgpg2. You can test your code on these nodes

before submitting it to CodeCrunch. You can log in to the cluster nodes using your SoC Unix ID. Details of these computing nodes can be found at the following URL:

https://dochub.comp.nus.edu.sg/cf/guides/compute-cluster/hardware

Some Points to Note:

1. Your `a2part1.py` should not take more than **1 minute** to complete execution on CodeCrunch and your `a2part2.py` should not take more than **2 minutes** to complete execution. Your code will be terminated by CodeCrunch if it takes more than the specified time limit.

2. Arguments to the Python files are absolute paths, not relative paths. They will be passed as arguments to the Python files, so please **do not hard code** any file paths in your code.

3. We will use python3.8 and PyTorch 1.9.0 to run your code. Make sure your code can **run correctly** with the specified requirements. Please test your code on xgpg0-2 before submitting them.

4. You can make a maximum of **99 submissions** to CodeCrunch for each part of this assignment.

Grading

Part 1 constitutes 30% of this assignment (3% per question). Part 2 constitutes 70% of this assignment. The marks awarded in Part 1 will be based on the correctness of your implementation. The marks awarded in Part 2 will be based on the language identification accuracy of your implementation when evaluated on a blind test set of texts.