CS4248
AY 2024/25 Semester 1
Assignment 2

Due Date

Monday 14 October 2024 at 9am. This assignment consists of two parts, and both parts need to be submitted through CodeCrunch before the due date. Late assignments will not be accepted and will receive ZERO marks.

PART 1

Objective

The first part of this assignment will guide you through PyTorch tensors. Your implementation of the functions needs to be done exclusively in PyTorch and basic Python 3 functions, i.e., the only import statement is `import torch`. You are provided with skeletal code `a2part1.py`. Please **do not** change the functions' **identifiers** (names, parameters, etc.) and do not change the **file name** to make sure that your assignment can be graded. Failure to do so will give you ZERO marks.

1. Given a tensor t and a value v, generate a new tensor with the same shape as t in which all elements are v.
```
Q1 example input:
t: tensor([[1, 2, 3, 4],
           [5, 6, 7, 8]])
v: 10

Q1 example output:
tensor([[10., 10., 10., 10.],
        [10., 10., 10., 10.]])
```

2. Define a function to get the second last column from a 2-dimensional tensor.
```
Q2 example input:
tensor([[1, 2, 3, 4],
        [5, 6, 7, 8]])

Q2 example output:
tensor([3, 7])
```

3. Define a function that accepts a tensor $t \in \mathbb{R}^{n \times 2d}$and multiplies the second half of t along the dimension of size $2d$ by -1. Do not use any loops.
```
Q3 example input:
tensor([[1, 2, 3, 4],
        [5, 6, 7, 8]])

Q3 example output:
tensor([[1, 2, -3, -4],
        [5, 6, -7, -8]])
```

4. Define a function that accepts a tensor $t \in \mathbb{R}^{n\times n}$ and returns a tensor $t_{cat} \in \mathbb{R}^{n\times 2n}$ which is the result of concatenating tensor t to the transpose of t along the last dimension. Do not use any loops.
```
Q4 example input:
tensor([[0, 1],
        [0, 2]]),

Q4 example output:
tensor([[ 0, 1, 0, 0],
        [ 0, 2, 1, 2]])
```

5. All elements in a tensor must have the same length. When combining tensors with different lengths, there are typically two different ways to achieve this: we can either pad the shorter tensors with padding elements, or truncate the longer tensors to match the length of the shortest tensor. Define a function to combine a list of 1-D tensors with different lengths into a new tensor by **truncating** the longer tensors.
```
Q5 example input:
[tensor([1, 2, 3]), tensor([4, 5]), tensor([6, 7, 8, 9])]

Q5 example output:
tensor([[1, 2],
        [4, 5],
        [6, 7]])
```

6. Given the input $\boldsymbol{X}$, $\boldsymbol{Q}$, and $\boldsymbol{K}$ in the type of PyTorch two-dimensional tensors, define a function that calculates $(\boldsymbol{X} \times \boldsymbol{Q}) \times (\boldsymbol{X} \times \boldsymbol{K})^T$. Implement the function in a PyTorch multiplication and transpose function without any loops.
```
Q6 example input:
X: tensor([[1, 2, 3],
           [4, 5, 6]])

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

K: tensor([[ 0.0200, -0.0300],
           [ 0.0300,  0.0200],
           [ 0.0200,  0.0000]]

Q6 example output:
tensor([[0.0900, 0.1980],
        [0.3510, 0.8100]])
```

7. To improve efficiency, matrix computation is usually done in a batch consisting of b tensors at the same time. Given the input $\mathbf{X} \in \mathbb{R}^{b\times n\times m}, \mathbf{Q} \in \mathbb{R}^{m\times d}, \mathbf{K} \in \mathbb{R}^{m\times d}$, calculate $(\mathbf{X}_i \times \mathbf{Q}) \times (\mathbf{X}_i \times \mathbf{K})^T$ for $i \in \{1, 2, \ldots, b\}$. Implement the function in a PyTorch multiplication and transpose function without any loops.
```
Q7 example input:
X: tensor([[[1, 2, 3],
            [4, 5, 6]],
           [[2, 1, 2],
            [3, 2, 3]]])
```

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

K: tensor([[ 0.0200, -0.0300],
           [ 0.0300,  0.0200],
           [ 0.0200,  0.0000]])

Q7 example output:
tensor([[[0.0900, 0.1980],
         [0.3510, 0.8100]],
        [[0.0380, 0.0730],
         [0.0920, 0.1660]]])
```

8. When working with tensors with padding, we often want to apply a mask to ensure that these padding values do not influence a model's predictions. Given a 3-D tensor $\boldsymbol{X}$ and a padding value v, apply a mask to $\boldsymbol{X}$ by setting any value v in $\boldsymbol{X}$ to 0.

```
Q8 example input:
X: torch.tensor([[[ 1, 2],
                  [-1,-1],
                  [-1,-1]],
                 [[ 2, 2],
                  [ 2, 1],
                  [-1,-1]],
                 [[ 3, 4],
                  [ 1, 3],
                  [ 3, 6]]])
v: -1

Q8 example output:
X: torch.tensor([[[ 1, 2],
                  [ 0, 0],
                  [ 0, 0]],
                 [[ 2, 2],
                  [ 2, 1],
                  [ 0, 0]],
                 [[ 3, 4],
                  [ 1, 3],
                  [ 3, 6]]])
```

9. Given a 3-D tensor $\boldsymbol{X} \in \mathbb{R}^{b \times n \times m}$ that contains paddings (-1 values), calculate the average of the tensor elements along the second dimension (of size n) ignoring the paddings.

```
Q9 example input:
tensor([[[ 1,  2],
         [-1, -1],
         [-1, -1]],
        [[ 2,  2],
         [ 2,  1],
         [-1, -1]],
        [[ 3,  4],
         [ 1,  3],
         [ 3,  6]]])
```

```
Q9 example output:
tensor([[1.0000, 2.0000],
        [2.0000, 1.5000],
        [2.3333, 4.3333]])
```

10. One of the commonly used loss functions is the squared loss function. Given a list of n pairs of vectors of the same length (in the type of Python list) $[(\hat{\boldsymbol{y}}_1, \boldsymbol{y}_1), \cdots, (\hat{\boldsymbol{y}}_n, \boldsymbol{y}_n)]$ where $\hat{\boldsymbol{y}}_1, \boldsymbol{y}_1, \ldots, \hat{\boldsymbol{y}}_n, \boldsymbol{y}_n \in \mathbb{R}^d$, sum the squared loss between each pair of vectors and return it as a 0-D tensor $t \in \mathbb{R}$. The squared loss of $\hat{\boldsymbol{y}}_i$ and $\boldsymbol{y}_i$ is $(\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i)^2$.

```
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(11.0954)
```

PART 2

Objective

In this assignment, you are to write a program in Python 3 to perform sentiment classification. Given a text, the task is to assign a class (positive or negative) denoting positive or negative sentiment expressed by the text.

You are to implement a neural network model with bigram embeddings, a hidden layer, and an output layer, using PyTorch. You are not allowed to use external libraries; only PyTorch and Python standard library are allowed. This part of the assignment comes with skeletal code in `a2part2.py`. You are required to use the skeletal code and make sure that your code runs correctly on CodeCrunch. For this part of the assignment, you are allowed to add your own functions to the skeletal code. You can test your code before submitting it to CodeCrunch on any of the SOC compute cluster GPU servers.

Approach

In this assignment, you are to implement a feed-forward neural network with a non-linear activation function in the hidden layer. The neural network uses bigram features to predict the sentiment of the text. Given a text with n tokens $t = \{t_1, \dots, t_n\}$, where t_i is the token in the i-th position, the model will first generate bigrams $x_{1:n-1} = \{x_1, \dots, x_{n-1}\}$ where $x_i = t_i t_{i+1}$, then assign a class y with the probability of $p(y|x_{1:n-1})$.

Each bigram x_i is represented by an embedding vector $\mathbf{x}_i = \mathbf{E}_{[\mathrm{idx}(x_i)]} \in \mathbb{R}^d$, obtained from a lookup table (embedding matrix) $\mathbf{E} \in \mathbb{R}^{|V| \times d}$, where d is the vector dimension, V is the vocabulary of bigrams, and the function $\mathrm{idx}(x)$ 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 bigram sequence of length k ($k = n - 1$) is represented as

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

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

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

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

We apply dropout to $\mathbf{h}_1$ during training and calculate the probability of the sentiment class through a linear layer ($\mathbf{w}_2 \in \mathbb{R}^{m \times 1}, b_2 \in \mathbb{R}^1$). Lastly, we apply sigmoid to the linear layer output to compute the probability p of the text having a positive sentiment.

$$p = \sigma(\mathbf{h}_1 \times \mathbf{w}_2 + b_2)$$

The loss function to use is binary cross-entropy loss.

Training and Testing

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

```
python3 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 weights from training as well as other information that you need to save for testing.

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

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

Your output file `out.txt` must contain the sentiment classes of the test texts, in the same format as the `y_train.txt` file.

The following command calculates the accuracy of your sentiment classification model, where `out.txt` is the output file of your code and `y_test.txt` contains the human-annotated sentiment classes of `x_test.txt.`

```
python3 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 (24/25 Sem 1) A2 part 1' under the CS4248 module to submit your `a2part1.py` and 'CS4248 (24/25 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. You can test your code on these nodes before submitting it to CodeCrunch. You can login 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 10GB of GPU memory and no more than **5 minutes** to complete execution. Your code will be terminated by CodeCrunch if it takes more than the specified restrictions.

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 Python 3.10 and PyTorch 2.3.0 to run your code. Make sure your code can **run correctly** with the specified requirements.

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 sentiment classification accuracy of your implementation when evaluated on a blind test set of texts.