Project 2: CS61Classify

Part A

Task 1: ReLU

什么是`ReLU`？ $ReLU(a) = max(a, 0)$

```
for (int i =0;i<a1;i++){
    if (a[i]<0) a[i]=0;
}
```

`任务要求`：`a0` 是数组的指针，`a1` 是数组的长度, 如果数组的长度小于1，返回`error code 8`

```
.globl relu

.text
# ================================================================================
# FUNCTION: Performs an inplace element-wise ReLU on an array of ints
# Arguments:
# a0 (int*) is the pointer to the array
#   a1 (int)  is the # of elements in the array
# Returns:
#   None
#
# If the length of the vector is less than 1,
# this function exits with error code 8.
#
# ================================================================================
relu:
loop_start:

    add t0, x0, x0
    addi t0, x0, 1
    blt a1, t0, loop_error

    # t0 is used as the current length
    add t0, x0, x0
    # t1 is used as the offset
    add t1, x0, x0
    add t1, x0, a0


loop_continue:
    lw t2, 0(t1)
    bge t2, x0, great_than_0
    add t3, x0, x0
    sw t3, 0(t1)
```

```
  great_than_0:
    addi t0, t0, 1
    addi t1, t1, 4
    blt t0, a1, loop_continue
loop_end:
      ret
loop_error:
    add a0, x0, x0
    addi a0, x0, 8
    ret
```

测试

```
python3 runner.py relu
```

Task 2: ArgMax

`任务要求`: 找出一个数组中的最大数的下标,如果最大数有多个，返回下标最小的那个。如果数组长度小于1，返回`code error 7`

```
int idx=0;
int cur=a[0];
for (int i=0;i<a1;i++){
    if (a[i]>cur){
        idx=i;
        cur=a[i];
    }
}
```

```
.globl argmax

.text
# =============================================================
# FUNCTION: Given a int vector, return the index of the largest
#   element. If there are multiple, return the one
#   with the smallest index.
# Arguments:
# a0 (int*) is the pointer to the start of the vector
#   a1 (int)  is the # of elements in the vector
# Returns:
#   a0 (int)  is the first index of the largest element
#
# If the length of the vector is less than 1,
# this function exits with error code 7.
# =============================================================
argmax:
loop_start:
    add t0, x0, x0
```

```
    addi t0, x0, 1
    blt a1, t0, loop_error
    # t0 is used as the current length
    add t0, x0, x0
    # t1 is used as the offset
    add t1, x0, x0
    add t1, x0, a0
    # t2 is used as the index stored
    add t2, x0, x0
    # t3 is used as the minimum
    lw t3, 0(t1)

loop_continue:
    lw t4, 0(t1)
    bge t3, t4, greater_equal_min
    add t2, t0, x0
    add t3, t4, x0

greater_equal_min:
    addi t0, t0, 1
    addi t1, t1, 4
    blt t0, a1, loop_continue

loop_end:
    add a0, t2, x0
    ret

loop_error:
    add a0, x0, x0
    addi a0, x0, 7
    ret
```

测试

```
python3 runner.py argmax
```

Task 3.1: Dot Product

```
.globl dot

.data
vector0: .word 1 2 3 4 5 6 7 8 9
vector1: .word 1 2 3 4 5 6 7 8 9

.text
# =======================================================
# FUNCTION: Dot product of 2 int vectors
# Arguments:
#   a0 (int*) is the pointer to the start of v0
#   a1 (int*) is the pointer to the start of v1
```

```
#   a2 (int)  is the length of the vectors
#   a3 (int)  is the stride of v0
#   a4 (int)  is the stride of v1
# Returns:
#   a0 (int)  is the dot product of v0 and v1
#
# If the length of the vector is less than 1,
# this function exits with error code 5.
# If the stride of either vector is less than 1,
# this function exits with error code 6.
# ========================================================
dot:
loop_start:
    la a0,vector0
    la a1,vector1
    addi a2,x0,10
    addi a3,x0,1
    addi a4,x0,1
    addi t0, x0, 1
    blt a2, t0, loop_error1
    blt a3, t0, loop_error2
    blt a4, t0, loop_error2

    addi t0, x0, 4
    mul a3, a3, t0
    mul a4, a4, t0
    mul a5, a2, t0
    mul a6, a2, t0

    add t1, x0, a0
    add t2, x0, a1
    add t3, x0, x0

    add a5,a5,t1
    add a6,a6,t2

loop_continue:
    lw t4, 0(t1)
    lw t5, 0(t2)
    mul t4, t4, t5
    add t3, t3, t4
    add t1, t1, a3
    add t2, t2, a4
    blt t1,a5,check
    j loop_end

check:
    blt t2,a6,loop_continue


loop_end:
    add a0, t3, x0
```

```
    ret

loop_error1:
    addi a0, x0, 5
    ret

loop_error2:
    addi a0, x0, 6
    ret
```

测试

注意本题我们需要自行修改文件`test_dot.s`，`test_dot.json`来测试。

Task 3.2: Matrix Multiplication

我们拿两个指针，第一个指向左边那个矩阵的行，第二个指向右边矩阵的列。

两层循环，外层循环遍历左边矩阵的行，内层循环遍历右边矩阵的列。然后需要注意步长。

```
.globl matmul

.text
# =======================================================
# FUNCTION: Matrix Multiplication of 2 integer matrices
#   d = matmul(m0, m1)
#   The order of error codes (checked from top to bottom):
#   If the dimensions of m0 do not make sense,
#   this function exits with exit code 2.
#   If the dimensions of m1 do not make sense,
#   this function exits with exit code 3.
#   If the dimensions don't match,
#   this function exits with exit code 4.
# Arguments:
#   a0 (int*)  is the pointer to the start of m0
#   a1 (int)   is the # of rows (height) of m0
#   a2 (int)   is the # of columns (width) of m0
#   a3 (int*)  is the pointer to the start of m1
#   a4 (int)   is the # of rows (height) of m1
#   a5 (int)   is the # of columns (width) of m1
#   a6 (int*)  is the pointer to the the start of d
# Returns:
#   None (void), sets d = matmul(m0, m1)
# =======================================================
# C=A*B, A=N*M B = M*K
# N=a1, M=a2 K=a5
matmul:

    addi t0,x0,1
    blt a1,t0,loop_error1
    blt a2,t0,loop_error1
    blt a4,t0,loop_error2
```

```
    blt a5,t0,loop_error2
    bne a2,a4, loop_error3

    addi sp, sp, -44
    sw s0 0(sp)
    sw s1 4(sp)
    sw s2 8(sp)
    sw s3 12(sp)
    sw s4 16(sp)
    sw s5 20(sp)
    sw s6 24(sp)
    sw s7 28(sp)
    sw s8 32(sp)
    sw s9 36(sp)
    sw ra 40(sp)

    add s0,x0,a0 #s0为A的指针
    add s1,x0,a3 #s1为B的指针
    add s2,x0,a6 #s2为C的指针
    add t0,x0,a2 #M
    addi t1,x0,4
    mul t0,t0,t1
    add s3,t0,x0 #4*M
    add s4,x0,a1 #N
    add s5,x0,a5 #K
    add s6,x0,a2 #M
    add s7,x0,x0 #index


outer_loop_start:
    add s8,x0,x0 #枚举B的列的循环指标
    add s9,x0,s1 #B的指针


inner_loop_start:
    add a0,x0,s0 #A的指针
    add a1,x0,s9 #B的指针
    add a2,x0,s6 #C的指针
    addi a3,x0,1 #A的步长为1
    add a4,x0,s5
    jal ra,dot #调用rot
    sw a0,0(s2) #算出的A的第一行和B的第一列的点乘
    addi s8,s8,1 #index+1
    addi s9,s9,4 #考虑B的下一列
    addi s2,s2,4
    blt s8,s5,inner_loop_start #B的列枚举完了


inner_loop_end:
    addi s7,s7,1 #A的下一行
    add s0,s0,s3 #A的指针指向下一行的起点
    blt s7,s4,outer_loop_start #A的行还没枚举完
```

```
85
86
87
88  outer_loop_end:
89      lw s0 0(sp)
90      lw s1 4(sp)
91      lw s2 8(sp)
92      lw s3 12(sp)
93      lw s4 16(sp)
94      lw s5 20(sp)
95      lw s6 24(sp)
96      lw s7 28(sp)
97      lw s8 32(sp)
98      lw s9 36(sp)
99      lw ra 40(sp)
100     addi sp, sp, 44
101     ret
102
103 loop_error1:
104     addi a0, x0, 2
105     ret
106 loop_error2:
107     addi a0, x0, 3
108     ret
109 loop_error3:
110     addi a0, x0, 4
111     ret
112
113
```

Part B: File Operations and Main

Task 1: Read Matrix

在本任务中，我们需要读入一个二进制矩阵文件，并为其安排存储空间。

官方给我们提供了一些文件操作相关的函数：

- `fopen`：打开一个文件，其参数为：
  - `a1`是一个指向文件名的指针
  - `a2` 代表权限
  - 返回值为一个`file descriptor`
- `fread`：读取指定数量的bytes到`buffer`从一个文件，参数：
  - `a1`：`file desciptor`
  - `a2`：指向`buffer` 的一个指针
  - `a3`：读取多少个bytes
  - 返回一个数，其表示真正从文件读了多少个bytes

- `fwrite`：向文件中写入某种元素。参数
  - `a1`：`file desciptor`
  - `a2`：指向`buffer`的指针，`buffer`里面装我们想写的东西
  - `a3`写入多少个元素
  - `a4` `buffer`中每个元素的大小
  - 返回一个数，其表示真正写入的元素的个数
- `fclose`关闭文件，并保存我们对其的写入， 参数
  - `a1`：`file desciptor`
  - 返回0代表成功，-1代表失败

```
.globl read_matrix

.text
# ==============================================================================
# FUNCTION: Allocates memory and reads in a binary file as a matrix of integers
#   If any file operation fails or doesn't read the proper number of bytes,
#   exit the program with exit code 1.
# FILE FORMAT:
#   The first 8 bytes are two 4 byte ints representing the # of rows and columns
#   in the matrix. Every 4 bytes afterwards is an element of the matrix in
#   row-major order.
# Arguments:
#   a0 (char*) is the pointer to string representing the filename
#   a1 (int*)  is a pointer to an integer, we will set it to the number of rows
#   a2 (int*)  is a pointer to an integer, we will set it to the number of columns
# Returns:
#   a0 (int*)  is the pointer to the matrix in memory
#
# If you receive an fopen error or eof,
# this function exits with error code 50.
# If you receive an fread error or eof,
# this function exits with error code 51.
# If you receive an fclose error or eof,
# this function exits with error code 52.
# ==============================================================================
read_matrix:

    addi sp, sp, -32
    sw s0 0(sp)
    sw s1 4(sp)
    sw s2 8(sp)
    sw s3 12(sp)
```

```
    sw s4 16(sp)
    sw s5 20(sp)
    sw s6 24(sp)
    sw ra 28(sp)

    add s0,x0,a0 #s0是指向文件名的指针
    add s1,x0,a1 #s1是指向行数的指针
    add s2,x0,a2 #s2是指向列数的指针
    add s3,x0,x0 #s3是file descriptor
    add s4,x0,x0
    add s5,x0,x0
    add a1,x0,s0
    addi a2,x0,0
    jal ra,fopen

    add s3,x0,a0 #s3是file descriptor
    addi to,x0,-1
    beq s3,t0,open_error #异常

    add a1,x0,s3
    add a2,x0,s1 #s1表示行数
    addi a3,x0,4
    jal ra,fread #读取行数

    addi t0,x0,4
    ben a0,t0,read_error #异常

    add a1,x0,s3
    add a2,x0,s2 #s2表示列数
    addi a3,x0,4
    jal ra,fread

    addi t0, x0, 4
    bne a0, t0, read_error #一场

    lw t0,0(s1)
    lw t1,0(s2)
    mul s4, t0,t1 #矩阵元素总数

    addi t0,x0,4
    mul a0,t0,s4
    jal ra,malloc #分配大小为s4的空间

    add s5,x0,a0 # s5为指向分配空间的指针
    add t0,x0,x0
    beq s5,t0,mallo_error

    add a1,x0,s3
    add a2,x0,s5
    addi t0,x0,4
    mul a3,t0,s4
    jal ra,fread #把矩阵数据读给s5
```

```
    addi t0,x0,4
    mul t0,t0,s4
    bne a0,t0,read_error

    add s6,x0,s5
    j end


open_error:
    addi s6, x0, 50
    j end

read_error:
    addi s6, x0, 51
    j end

malloc_error:
    addi s6, x0, 48
    j end


end:
    add a1,x0,s3
    jal ra,fclose

    add t0,x0,x0
    beq a0,t0,restore
    addi s6,x0,52

restore:
    add a0,x0,s6 #设置返回值为指向矩阵的指针
    lw s0 0(sp)
    lw s1 4(sp)
    lw s2 8(sp)
    lw s3 12(sp)
    lw s4 16(sp)
    lw s5 20(sp)
    lw s6 24(sp)
    lw ra 28(sp)
    addi sp, sp, 32

    ret

```

Task 2: Write Matrix

```
.globl write_matrix

.text
```

```
#
================================================================================
# FUNCTION: Writes a matrix of integers into a binary file
#   If any file operation fails or doesn't write the proper number of bytes,
#   exit the program with exit code 1.
# FILE FORMAT:
#   The first 8 bytes of the file will be two 4 byte ints representing the
#   numbers of rows and columns respectively. Every 4 bytes thereafter is an
#   element of the matrix in row-major order.
# Arguments:
#   a0 (char*) is the pointer to string representing the filename
#   a1 (int*)  is the pointer to the start of the matrix in memory
#   a2 (int)   is the number of rows in the matrix
#   a3 (int)   is the number of columns in the matrix
# Returns:
#   None
#
# If you receive an fopen error or eof,
# this function exits with error code 53.
# If you receive an fwrite error or eof,
# this function exits with error code 54.
# If you receive an fclose error or eof,
# this function exits with error code 55.
#
================================================================================
write_matrix:

    addi sp, sp, -28
    sw s0 0(sp)
    sw s1 4(sp)
    sw s2 8(sp)
    sw s3 12(sp)
    sw s4 16(sp)
    sw s5 20(sp)
    sw ra 24(sp)

    add s0,x0,a0 #s0是指向文件名的指针
    add s1,x0,a1 #s1是指向存在内存中的矩阵的指针
    add s2,x0,a2 #s2是行数
    add s3,x0,a3 #s3是列数
    add s4,x0,x0 #s4是file descriptor
    add s5,x0,x0 #s5是返回值

    add a1,x0,s0
    addi a2,x0,1
    jal ra,fopen

    add s4,x0,a0 #file descriptor

    addi t0,x0,-1
    beq s4,t0,open_error

```

```
    #写入行数
    add a1,x0,s4
    la a2,row
    sw s2,0(a2)
    addi a3,x0,1 #写入一个元素
    addi a4,x0,4 #一个元素的大小是4B
    jal ra,fwrite

    addi t0,x0,1
    bne a0,t0,write_error

    add a1,x0,s4
    la a2,col
    sw s3,0(a2)
    addi a3,x0,1
    addi a4,x0,4
    jal ra,fwrite

    addi t0,x0,1
    bne a0,t0,write_error

    mul t0,s2,s3 #总元素个数
    add a1,x0,s4
    add a2,x0,s1
    add a3,x0,t0
    addi a4,x0,4
    jal ra,fwrite

    mul t0, s2, s3
    bne a0, t0, write_error

    j end

    open_error:
    addi s5, x0, 53
    j end

write_error:
    addi s5, x0, 54
    j end

end:

    add a1, x0, s4
    jal ra, fclose

    add t0, x0, x0
    beq a0, t0, restore
    addi s6, x0, 55

restore:
    lw s0 0(sp)
```

```
    lw s1 4(sp)
    lw s2 8(sp)
    lw s3 12(sp)
    lw s4 16(sp)
    lw s5 20(sp)
    lw ra 24(sp)
    addi sp, sp, 28

    ret
```

Task 3: Putting it all Together

我们的神经网络的计算方式如下:

```
hidden_layer = matmul(m0, input)
relu(hidden_layer) # Recall that relu is performed in-place
scores = matmul(m1, hidden_layer)
```

把我们之前所完成的函数组合起来就行了。

```
.globl classify

.data
m0: .word 0 0
m1: .word 0 0
input: .word 0 0

.text
classify:
    # =====================================
    # COMMAND LINE ARGUMENTS
    # =====================================
    # Args:
    #   a0 (int)    argc
    #   a1 (char**) argv
    #   a2 (int)    print_classification, if this is zero,
    #               you should print the classification. Otherwise,
    #               this function should not print ANYTHING.
    # Returns:
    #   a0 (int)    Classification
    #
    # If there are an incorrect number of command line args,
    # this function returns with exit code 49.
    #
    # Usage:
    #   main.s -m -1 <M0_PATH> <M1_PATH> <INPUT_PATH> <OUTPUT_PATH>

    addi t0, x0, 5
    bne a0, t0, arg_error

    addi sp, sp, -40
```

```
    sw s0 0(sp)
    sw s1 4(sp)
    sw s2 8(sp)
    sw s3 12(sp)
    sw s4 16(sp)
    sw s5 20(sp)
    sw s6 24(sp)
    sw s7 28(sp)
    sw s8 32(sp)
    sw ra 36(sp)

    # s0 is argc
    add s0, x0, a0
    # s1 is argv
    add s1, x0, a1
    # s2 is a flag to indicate whether print
    add s2, x0, a2
    # s3 is a pointer to point to the m0
    add s3, x0, x0
    # s4 is a pointer to point to the m1
    add s4, x0, x0
    # s5 is a poiner to point to the input matrix
    add s5, x0, x0
    # s6 is the dynamic memory in the heap
    add s6, x0, x0
    # s7 is the dynamic memory in the heap
    add s7, x0, x0
    # s8 is the return value
    add s8, x0, x0

      # ====================================
    # LOAD MATRICES
    # ====================================

    # Load pretrained m0

    lw a0, 4(s1)
    la t0, m0
    addi a1, t0, 0
    addi a2, t0, 4
    jal ra, read_matrix

    add s3, a0, x0

    # Load pretrained m1

    lw a0, 8(s1)
    la t0, m1
    addi a1, t0, 0
    addi a2, t0, 4
    jal ra, read_matrix

```

```
    add s4, a0, x0

    # Load input matrix

    lw a0, 12(s1)
    la t0, input
    addi a1, t0, 0
    addi a2, t0, 4
    jal ra, read_matrix

    add s5, a0, x0

    # ====================================
    # RUN LAYERS
    # ====================================
    # 1. LINEAR LAYER:    m0 * input
    # 2. NONLINEAR LAYER: ReLU(m0 * input)
    # 3. LINEAR LAYER:    m1 * ReLU(m0 * input)

    # Here, we need to allocate the memory
    la t0, m0
    la t1, input
    lw t2, 0(t0)
    lw t3, 4(t1)
    mul t4, t2, t3
    addi t5, x0, 4
    mul a0, t4, t5

    jal ra, malloc
    add s6, a0, x0

    # m0 * input
    add a0, x0, s3
    la t0, m0
    lw a1, 0(t0) # m0 row
    lw a2, 4(t0) # m0 col
    add a3, x0, s5
    la t0, input
    lw a4, 0(t0) # input row
    lw a5, 4(t0) # input col
    add a6, x0, s6
    jal ra, matmul

    # ReLU(m0 * input)
    add a0, x0, s6
    la t0, m0
    la t1, input
    lw t2, 0(t0)
    lw t3, 4(t1)
    mul a1, t2, t3
    jal ra, relu

```

```
    # m1 * ReLU(m0 * input)
    la t0, m1
    la t1, input
    lw t2, 0(t0)
    lw t3, 4(t1)
    mul t4, t2, t3
    addi t5, x0, 4
    mul a0, t4, t5

    jal ra, malloc
    add s7, a0, x0

    add a0, s4, x0
    la t0, m1
    lw a1, 0(t0) # m1 row
    lw a2, 4(t0) # m1 col
    add a3, s6, x0
    la t0, m0
    la t1, input
    lw a4, 0(t0)
    lw a5, 4(t1)
    add a6, x0, s7

    jal ra, matmul

    # ====================================
    # WRITE OUTPUT
    # ====================================
    # Write output matrix

    lw a0, 16(s1)
    add a1, s7, x0
    la t0, m1
    la t1, input
    lw a2, 0(t0)
    lw a3, 4(t1)

    jal ra, write_matrix

    # ====================================
    # CALCULATE CLASSIFICATION/LABEL
    # ====================================
    # Call argmax

    la t0, m1
    la t1, input
    lw t2, 0(t0)
    lw t3, 4(t1)
    mul t4, t2, t3

    add a0, x0, s7
    add a1, x0, t4
```

```
    jal ra, argmax

    # Print classification

    add s8, x0, a0
    bne s2, x0, end
    add a1, x0, s8
    jal ra, print_int

    # Print newline afterwards for clarity
    li a1 '\n'
    jal ra, print_char

end:

    add a0, s3, x0
    jal ra, free
    add a0, s4, x0
    jal ra, free
    add a0, s5, x0
    jal ra, free
    add a0, s6, x0
    jal ra, free
    add a0, s7, x0
    jal ra, free


    add a0, s8, x0
    lw s0 0(sp)
    lw s1 4(sp)
    lw s2 8(sp)
    lw s3 12(sp)
    lw s4 16(sp)
    lw s5 20(sp)
    lw s6 24(sp)
    lw s7 28(sp)
    lw s8 32(sp)
    lw ra 36(sp)
    addi sp, sp, 40

    ret
arg_error:
    addi a0, x0, 49
    ret
```