

Project 2: CS61Classify

Part A

Task 1: ReLU

什么是 ReLU? $ReLU(a) = \max(a, 0)$

```
1  for (int i =0;i<a1;i++){
2      if (a[i]<0) a[i]=0;
3  }
```

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

```
1  .globl relu
2
3  .text
4  #
5  =====
6  # FUNCTION: Performs an inplace element-wise ReLU on an array of ints
7  # Arguments:
8  # a0 (int*) is the pointer to the array
9  # a1 (int) is the # of elements in the array
10 # Returns:
11 # None
12 # If the length of the vector is less than 1,
13 # this function exits with error code 8.
14 #
15 =====
16 relu:
17 loop_start:
18     add t0, x0, x0
19     addi t0, x0, 1
20     blt a1, t0, loop_error
21
22     # t0 is used as the current length
23     add t0, x0, x0
24     # t1 is used as the offset
25     add t1, x0, x0
26     add t1, x0, a0
27
28
29 loop_continue:
30     lw t2, 0(t1)
31     bge t2, x0, great_than_0
32     add t3, x0, x0
33     sw t3, 0(t1)
```

```

34     great_than_0:
35         addi t0, t0, 1
36         addi t1, t1, 4
37         blt t0, a1, loop_continue
38     loop_end:
39         ret
40     loop_error:
41         add a0, x0, x0
42         addi a0, x0, 8
43         ret

```

测试

```
1 | 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

```

1  int idx=0;
2  int cur=a[0];
3  for (int i=0;i<a1;i++){
4      if (a[i]>cur){
5          idx=i;
6          cur=a[i];
7      }
8  }

```

```

1  .globl argmax
2
3  .text
4  # =====
5  # FUNCTION: Given a int vector, return the index of the largest
6  #   element. If there are multiple, return the one
7  #   with the smallest index.
8  # Arguments:
9  # a0 (int*) is the pointer to the start of the vector
10 # a1 (int) is the # of elements in the vector
11 # Returns:
12 # a0 (int) is the first index of the largest element
13 #
14 # If the length of the vector is less than 1,
15 # this function exits with error code 7.
16 # =====
17 argmax:
18 loop_start:
19     add t0, x0, x0

```

```

20     addi t0, x0, 1
21     blt a1, t0, loop_error
22     # t0 is used as the current length
23     add t0, x0, x0
24     # t1 is used as the offset
25     add t1, x0, x0
26     add t1, x0, a0
27     # t2 is used as the index stored
28     add t2, x0, x0
29     # t3 is used as the minimum
30     lw t3, 0(t1)
31
32 loop_continue:
33     lw t4, 0(t1)
34     bge t3, t4, greater_equal_min
35     add t2, t0, x0
36     add t3, t4, x0
37
38 greater_equal_min:
39     addi t0, t0, 1
40     addi t1, t1, 4
41     blt t0, a1, loop_continue
42
43 loop_end:
44     add a0, t2, x0
45     ret
46
47 loop_error:
48     add a0, x0, x0
49     addi a0, x0, 7
50     ret

```

测试

```
1 | python3 runner.py argmax
```

Task 3.1: Dot Product

```

1  .globl dot
2
3  .data
4  vector0: .word 1 2 3 4 5 6 7 8 9
5  vector1: .word 1 2 3 4 5 6 7 8 9
6
7  .text
8  # =====
9  # FUNCTION: Dot product of 2 int vectors
10 # Arguments:
11 #   a0 (int*) is the pointer to the start of v0
12 #   a1 (int*) is the pointer to the start of v1

```

```

13 #   a2 (int)   is the length of the vectors
14 #   a3 (int)   is the stride of v0
15 #   a4 (int)   is the stride of v1
16 # Returns:
17 #   a0 (int)   is the dot product of v0 and v1
18 #
19 # If the length of the vector is less than 1,
20 # this function exits with error code 5.
21 # If the stride of either vector is less than 1,
22 # this function exits with error code 6.
23 # =====
24 dot:
25 loop_start:
26     la a0,vector0
27     la a1,vector1
28     addi a2,x0,10
29     addi a3,x0,1
30     addi a4,x0,1
31     addi t0, x0, 1
32     blt a2, t0, loop_error1
33     blt a3, t0, loop_error2
34     blt a4, t0, loop_error2
35
36     addi t0, x0, 4
37     mul a3, a3, t0
38     mul a4, a4, t0
39     mul a5, a2, t0
40     mul a6, a2, t0
41
42     add t1, x0, a0
43     add t2, x0, a1
44     add t3, x0, x0
45
46     add a5,a5,t1
47     add a6,a6,t2
48
49 loop_continue:
50     lw t4, 0(t1)
51     lw t5, 0(t2)
52     mul t4, t4, t5
53     add t3, t3, t4
54     add t1, t1, a3
55     add t2, t2, a4
56     blt t1,a5,check
57     j loop_end
58
59 check:
60     blt t2,a6,loop_continue
61
62
63 loop_end:
64     add a0, t3, x0

```

```

65     ret
66
67 loop_error1:
68     addi a0, x0, 5
69     ret
70
71 loop_error2:
72     addi a0, x0, 6
73     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。

```

1  .globl matmul
2
3  .text
4  # =====
5  # FUNCTION: Matrix Multiplication of 2 integer matrices
6  #   d = matmul(m0, m1)
7  #   The order of error codes (checked from top to bottom):
8  #   If the dimensions of m0 do not make sense,
9  #   this function exits with exit code 2.
10 #   If the dimensions of m1 do not make sense,
11 #   this function exits with exit code 3.
12 #   If the dimensions don't match,
13 #   this function exits with exit code 4.
14 # Arguments:
15 #   a0 (int*) is the pointer to the start of m0
16 #   a1 (int)  is the # of rows (height) of m0
17 #   a2 (int)  is the # of columns (width) of m0
18 #   a3 (int*) is the pointer to the start of m1
19 #   a4 (int)  is the # of rows (height) of m1
20 #   a5 (int)  is the # of columns (width) of m1
21 #   a6 (int*) is the pointer to the the start of d
22 # Returns:
23 #   None (void), sets d = matmul(m0, m1)
24 # =====
25 # C=A*B, A=N*M B = M*K
26 # N=a1, M=a2 K=a5
27 matmul:
28
29     addi t0,x0,1
30     blt a1,t0,loop_error1
31     blt a2,t0,loop_error1
32     blt a4,t0,loop_error2

```

```

33     blt a5,t0,loop_error2
34     bne a2,a4, loop_error3
35
36     addi sp, sp, -44
37     sw s0 0(sp)
38     sw s1 4(sp)
39     sw s2 8(sp)
40     sw s3 12(sp)
41     sw s4 16(sp)
42     sw s5 20(sp)
43     sw s6 24(sp)
44     sw s7 28(sp)
45     sw s8 32(sp)
46     sw s9 36(sp)
47     sw ra 40(sp)
48
49     add s0,x0,a0 #s0为A的指针
50     add s1,x0,a3 #s1为B的指针
51     add s2,x0,a6 #s2为C的指针
52     add t0,x0,a2 #M
53     addi t1,x0,4
54     mul t0,t0,t1
55     add s3,t0,x0 #4*M
56     add s4,x0,a1 #N
57     add s5,x0,a5 #K
58     add s6,x0,a2 #M
59     add s7,x0,x0 #index
60
61
62     outer_loop_start:
63         add s8,x0,x0 #枚举B的列的循环指标
64         add s9,x0,s1 #B的指针
65
66
67     inner_loop_start:
68         add a0,x0,s0 #A的指针
69         add a1,x0,s9 #B的指针
70         add a2,x0,s6 #C的指针
71         addi a3,x0,1 #A的步长为1
72         add a4,x0,s5
73         jal ra,dot #调用rot
74         sw a0,0(s2) #算出的A的第一行和B的第一列的点乘
75         addi s8,s8,1 #index+1
76         addi s9,s9,4 #考虑B的下一列
77         addi s2,s2,4
78         blt s8,s5,inner_loop_start #B的列枚举完了
79
80
81     inner_loop_end:
82         addi s7,s7,1 #A的下一行
83         add s0,s0,s3 #A的指针指向下一行的起点
84         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 descriptor`
 - `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 descriptor
 - `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 descriptor
 - 返回0代表成功, -1代表失败

```

1  .globl read_matrix
2
3  .text
4  #
=====
5  # FUNCTION: Allocates memory and reads in a binary file as a matrix of
   integers
6  #   If any file operation fails or doesn't read the proper number of bytes,
7  #   exit the program with exit code 1.
8  # FILE FORMAT:
9  #   The first 8 bytes are two 4 byte ints representing the # of rows and
   columns
10 #   in the matrix. Every 4 bytes afterwards is an element of the matrix in
11 #   row-major order.
12 # Arguments:
13 #   a0 (char*) is the pointer to string representing the filename
14 #   a1 (int*) is a pointer to an integer, we will set it to the number of
   rows
15 #   a2 (int*) is a pointer to an integer, we will set it to the number of
   columns
16 # Returns:
17 #   a0 (int*) is the pointer to the matrix in memory
18 #
19 # If you receive an fopen error or eof,
20 # this function exits with error code 50.
21 # If you receive an fread error or eof,
22 # this function exits with error code 51.
23 # If you receive an fclose error or eof,
24 # this function exits with error code 52.
25 #
=====
26 read_matrix:
27
28     addi sp, sp, -32
29     sw s0 0(sp)
30     sw s1 4(sp)
31     sw s2 8(sp)
32     sw s3 12(sp)

```



```

33     sw s4 16(sp)
34     sw s5 20(sp)
35     sw s6 24(sp)
36     sw ra 28(sp)
37
38     add s0,x0,a0 #s0是指向文件名的指针
39     add s1,x0,a1 #s1是指向行数的指针
40     add s2,x0,a2 #s2是指向列数的指针
41     add s3,x0,x0 #s3是file descriptor
42     add s4,x0,x0
43     add s5,x0,x0
44     add a1,x0,s0
45     addi a2,x0,0
46     jal ra,fopen
47
48     add s3,x0,a0 #s3是file descriptor
49     addi t0,x0,-1
50     beq s3,t0,open_error #异常
51
52     add a1,x0,s3
53     add a2,x0,s1 #s1表示行数
54     addi a3,x0,4
55     jal ra,fread #读取行数
56
57     addi t0,x0,4
58     ben a0,t0,read_error #异常
59
60     add a1,x0,s3
61     add a2,x0,s2 #s2表示列数
62     addi a3,x0,4
63     jal ra,fread
64
65     addi t0, x0, 4
66     bne a0, t0, read_error #一场
67
68     lw t0,0(s1)
69     lw t1,0(s2)
70     mul s4, t0,t1 #矩阵元素总数
71
72     addi t0,x0,4
73     mul a0,t0,s4
74     jal ra,malloc #分配大小为s4的空间
75
76     add s5,x0,a0 # s5为指向分配空间的指针
77     add t0,x0,x0
78     beq s5,t0,mallo_error
79
80     add a1,x0,s3
81     add a2,x0,s5
82     addi t0,x0,4
83     mul a3,t0,s4
84     jal ra,fread #把矩阵数据读给s5

```

```

85
86     addi t0,x0,4
87     mul t0,t0,s4
88     bne a0,t0,read_error
89
90     add s6,x0,s5
91     j end
92
93
94 open_error:
95     addi s6, x0, 50
96     j end
97
98 read_error:
99     addi s6, x0, 51
100    j end
101
102 malloc_error:
103     addi s6, x0, 48
104     j end
105
106
107 end:
108     add a1,x0,s3
109     jal ra,fclose
110
111     add t0,x0,x0
112     beg a0,t0,restore
113     addi s6,x0,52
114
115 restore:
116     add a0,x0,s6 #设置返回值为指向矩阵的指针
117     lw s0 0(sp)
118     lw s1 4(sp)
119     lw s2 8(sp)
120     lw s3 12(sp)
121     lw s4 16(sp)
122     lw s5 20(sp)
123     lw s6 24(sp)
124     lw ra 28(sp)
125     addi sp, sp, 32
126
127     ret
128

```

Task 2: Write Matrix

```

1  .globl write_matrix
2
3  .text

```

```

4  #
=====
5  # FUNCTION: Writes a matrix of integers into a binary file
6  #   If any file operation fails or doesn't write the proper number of bytes,
7  #   exit the program with exit code 1.
8  # FILE FORMAT:
9  #   The first 8 bytes of the file will be two 4 byte ints representing the
10 #   numbers of rows and columns respectively. Every 4 bytes thereafter is an
11 #   element of the matrix in row-major order.
12 # Arguments:
13 #   a0 (char*) is the pointer to string representing the filename
14 #   a1 (int*)  is the pointer to the start of the matrix in memory
15 #   a2 (int)   is the number of rows in the matrix
16 #   a3 (int)   is the number of columns in the matrix
17 # Returns:
18 #   None
19 #
20 # If you receive an fopen error or eof,
21 # this function exits with error code 53.
22 # If you receive an fwrite error or eof,
23 # this function exits with error code 54.
24 # If you receive an fclose error or eof,
25 # this function exits with error code 55.
26 #
=====
27 write_matrix:
28
29     addi sp, sp, -28
30     sw s0 0(sp)
31     sw s1 4(sp)
32     sw s2 8(sp)
33     sw s3 12(sp)
34     sw s4 16(sp)
35     sw s5 20(sp)
36     sw ra 24(sp)
37
38     add s0,x0,a0 #s0是指向文件名的指针
39     add s1,x0,a1 #s1是指向存在内存中的矩阵的指针
40     add s2,x0,a2 #s2是行数
41     add s3,x0,a3 #s3是列数
42     add s4,x0,x0 #s4是file descriptor
43     add s5,x0,x0 #s5是返回值
44
45     add a1,x0,s0
46     addi a2,x0,1
47     jal ra,fopen
48
49     add s4,x0,a0 #file descriptor
50
51     addi t0,x0,-1
52     beq s4,t0,open_error
53

```

```

54     #写入行数
55     add a1,x0,s4
56     la a2,row
57     sw s2,0(a2)
58     addi a3,x0,1 #写入一个元素
59     addi a4,x0,4 #一个元素的大小是4B
60     jal ra,fwrite
61
62     addi t0,x0,1
63     bne a0,t0,write_error
64
65     add a1,x0,s4
66     la a2,col
67     sw s3,0(a2)
68     addi a3,x0,1
69     addi a4,x0,4
70     jal ra,fwrite
71
72     addi t0,x0,1
73     bne a0,t0,write_error
74
75     mul t0,s2,s3 #总元素个数
76     add a1,x0,s4
77     add a2,x0,s1
78     add a3,x0,t0
79     addi a4,x0,4
80     jal ra,fwrite
81
82     mul t0, s2, s3
83     bne a0, t0, write_error
84
85     j end
86
87     open_error:
88     addi s5, x0, 53
89     j end
90
91     write_error:
92     addi s5, x0, 54
93     j end
94
95     end:
96
97     add a1, x0, s4
98     jal ra, fclose
99
100    add t0, x0, x0
101    beq a0, t0, restore
102    addi s6, x0, 55
103
104    restore:
105    lw s0 0(sp)

```

```

106     lw s1 4(sp)
107     lw s2 8(sp)
108     lw s3 12(sp)
109     lw s4 16(sp)
110     lw s5 20(sp)
111     lw ra 24(sp)
112     addi sp, sp, 28
113
114     ret

```

Task 3: Putting it all Together

我们的神经网络的计算方式如下:

```

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

```

把我们之前所完成的函数组合起来就行了。

```

1  .globl classify
2
3  .data
4  m0: .word 0 0
5  m1: .word 0 0
6  input: .word 0 0
7
8  .text
9  classify:
10     # =====
11     # COMMAND LINE ARGUMENTS
12     # =====
13     # Args:
14     #   a0 (int)   argc
15     #   a1 (char**) argv
16     #   a2 (int)   print_classification, if this is zero,
17     #               you should print the classification. Otherwise,
18     #               this function should not print ANYTHING.
19     # Returns:
20     #   a0 (int)   Classification
21     #
22     # If there are an incorrect number of command line args,
23     # this function returns with exit code 49.
24     #
25     # Usage:
26     #   main.s -m -l <M0_PATH> <M1_PATH> <INPUT_PATH> <OUTPUT_PATH>
27
28     addi t0, x0, 5
29     bne a0, t0, arg_error
30
31     addi sp, sp, -40

```

```

32     sw s0 0(sp)
33     sw s1 4(sp)
34     sw s2 8(sp)
35     sw s3 12(sp)
36     sw s4 16(sp)
37     sw s5 20(sp)
38     sw s6 24(sp)
39     sw s7 28(sp)
40     sw s8 32(sp)
41     sw ra 36(sp)
42
43     # s0 is argc
44     add s0, x0, a0
45     # s1 is argv
46     add s1, x0, a1
47     # s2 is a flag to indicate whether print
48     add s2, x0, a2
49     # s3 is a pointer to point to the m0
50     add s3, x0, x0
51     # s4 is a pointer to point to the m1
52     add s4, x0, x0
53     # s5 is a pointer to point to the input matrix
54     add s5, x0, x0
55     # s6 is the dynamic memory in the heap
56     add s6, x0, x0
57     # s7 is the dynamic memory in the heap
58     add s7, x0, x0
59     # s8 is the return value
60     add s8, x0, x0
61
62     # =====
63     # LOAD MATRICES
64     # =====
65
66     # Load pretrained m0
67
68     lw a0, 4(s1)
69     la t0, m0
70     addi a1, t0, 0
71     addi a2, t0, 4
72     jal ra, read_matrix
73
74     add s3, a0, x0
75
76     # Load pretrained m1
77
78     lw a0, 8(s1)
79     la t0, m1
80     addi a1, t0, 0
81     addi a2, t0, 4
82     jal ra, read_matrix
83

```

```

84     add s4, a0, x0
85
86     # Load input matrix
87
88     lw a0, 12(s1)
89     la t0, input
90     addi a1, t0, 0
91     addi a2, t0, 4
92     jal ra, read_matrix
93
94     add s5, a0, x0
95
96     # =====
97     # RUN LAYERS
98     # =====
99     # 1. LINEAR LAYER:    m0 * input
100    # 2. NONLINEAR LAYER: ReLU(m0 * input)
101    # 3. LINEAR LAYER:    m1 * ReLU(m0 * input)
102
103    # Here, we need to allocate the memory
104    la t0, m0
105    la t1, input
106    lw t2, 0(t0)
107    lw t3, 4(t1)
108    mul t4, t2, t3
109    addi t5, x0, 4
110    mul a0, t4, t5
111
112    jal ra, malloc
113    add s6, a0, x0
114
115    # m0 * input
116    add a0, x0, s3
117    la t0, m0
118    lw a1, 0(t0) # m0 row
119    lw a2, 4(t0) # m0 col
120    add a3, x0, s5
121    la t0, input
122    lw a4, 0(t0) # input row
123    lw a5, 4(t0) # input col
124    add a6, x0, s6
125    jal ra, matmul
126
127    # ReLU(m0 * input)
128    add a0, x0, s6
129    la t0, m0
130    la t1, input
131    lw t2, 0(t0)
132    lw t3, 4(t1)
133    mul a1, t2, t3
134    jal ra, relu
135

```

```

136     # m1 * ReLU(m0 * input)
137     la t0, m1
138     la t1, input
139     lw t2, 0(t0)
140     lw t3, 4(t1)
141     mul t4, t2, t3
142     addi t5, x0, 4
143     mul a0, t4, t5
144
145     jal ra, malloc
146     add s7, a0, x0
147
148     add a0, s4, x0
149     la t0, m1
150     lw a1, 0(t0) # m1 row
151     lw a2, 4(t0) # m1 col
152     add a3, s6, x0
153     la t0, m0
154     la t1, input
155     lw a4, 0(t0)
156     lw a5, 4(t1)
157     add a6, x0, s7
158
159     jal ra, matmul
160
161     # =====
162     # WRITE OUTPUT
163     # =====
164     # Write output matrix
165
166     lw a0, 16(s1)
167     add a1, s7, x0
168     la t0, m1
169     la t1, input
170     lw a2, 0(t0)
171     lw a3, 4(t1)
172
173     jal ra, write_matrix
174
175     # =====
176     # CALCULATE CLASSIFICATION/LABEL
177     # =====
178     # Call argmax
179
180     la t0, m1
181     la t1, input
182     lw t2, 0(t0)
183     lw t3, 4(t1)
184     mul t4, t2, t3
185
186     add a0, x0, s7
187     add a1, x0, t4

```



```

188     jal ra, argmax
189
190     # Print classification
191
192     add s8, x0, a0
193     bne s2, x0, end
194     add a1, x0, s8
195     jal ra, print_int
196
197     # Print newline afterwards for clarity
198     li a1 '\n'
199     jal ra, print_char
200
201 end:
202
203     add a0, s3, x0
204     jal ra, free
205     add a0, s4, x0
206     jal ra, free
207     add a0, s5, x0
208     jal ra, free
209     add a0, s6, x0
210     jal ra, free
211     add a0, s7, x0
212     jal ra, free
213
214
215     add a0, s8, x0
216     lw s0 0(sp)
217     lw s1 4(sp)
218     lw s2 8(sp)
219     lw s3 12(sp)
220     lw s4 16(sp)
221     lw s5 20(sp)
222     lw s6 24(sp)
223     lw s7 28(sp)
224     lw s8 32(sp)
225     lw ra 36(sp)
226     addi sp, sp, 40
227
228     ret
229 arg_error:
230     addi a0, x0, 49
231     ret

```