

Contents

1	Basic Test Results	2
2	ex4/README.md	4
3	ex4/Activation.h	5
4	ex4/Activation.cpp	6
5	ex4/Dense.h	7
6	ex4/Dense.cpp	8
7	ex4/Matrix.h	9
8	ex4/Matrix.cpp	11
9	ex4/MlpNetwork.h	15
10	ex4/MlpNetwork.cpp	16

1 Basic Test Results

```
1
2 Running presubmission script...
3
4
5 Opening tar file
6 OK
7 Tar extracted O.K.
8 For your convenience, the MD5 checksum for your submission is 1318e5a96905e08d61d3876e7e0d726b
9 Checking files...
10 OK
11 Making sure files are not empty...
12 OK
13 Checking CodingStyle...
14 Checking file Matrix.cpp...
15 Problem: in Matrix.cpp, line 116 is longer than 79 characters and will be truncated.
16 Checking file Activation.cpp...
17 Checking file Dense.cpp...
18 Checking file MlpNetwork.cpp...
19 Problem: in MlpNetwork.cpp, line 6 is longer than 79 characters and will be truncated.
20 Checking file Matrix.h...
21 Problem: in Matrix.h, line [42:16], warning: parameter 'm' is const-qualified in the function declaration; const-qualificati
22
23     Matrix dot(const Matrix m) const;
24
25         ^~~~~~
26
27
28 Problem: in Matrix.h, line 35 is longer than 79 characters and will be truncated.
29 Checking file Activation.h...
30 Checking file Dense.h...
31 Checking file MlpNetwork.h...
32 Compilation check...
33 Compiling...
34
35 Compilation looks good!
36
37
38 =====
39 Public test cases
40 =====
41
42 Running test...
43 Checking functions exist and basic functionality:
44 Checking Matrix class:
45     constructors
46     operators () and []
47     get_rows, get_cols
48     vectorize
49     dot
50     norm
51     transpose
52     operator =
53     Matrix multiplication
54     Scalar multiplication on left
55     Scalar multiplication on right
56     Matrix addition
57     Matrix addition accumulation
58     plain_print: Should print 0 to 8
59 0 1 2 3 4 5 6 7 8
```

```

60 Passed: All Matrix functions exist
61
62 Checking Activation class:
63     existence
64     operator ()
65 Passed: All Activation functions exist
66
67 Checking Dense class:
68     constructor
69     get_weights, get_bias, get_activation
70     operator ()
71 Passed: All Dense functions exist
72
73 Checking MlpNetwork, displaying output:
74 Image processed:
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104 Mlp result: 5 at probability: 0.99821
105 All presubmit tests finished!
106
107
108 OK
109 *****
110 *           ***           *
111 *       Passed all tests!! *
112 *       Good Job!         *
113 *           ***           *
114 *****

```

2 ex4/README.md

1 # ex4-shiraneyal

3 ex4/Activation.h

```
1  #include "Matrix.h"
2
3  #ifndef ACTIVATION_H
4  #define ACTIVATION_H
5
6  // Insert Activation class here...
7  namespace activation {
8      // relu function (given matrix, change every negative element to zero)
9      Matrix relu(const Matrix& m);
10     // softmax function (returns exponent vector)
11     Matrix softmax(const Matrix& m);
12 };
13 #endif //ACTIVATION_H
```

4 ex4/Activation.cpp

```
1  #include "Activation.h"
2
3  Matrix activation::relu(const Matrix& m) {
4      Matrix m2(m.get_rows(), m.get_cols());
5      for (int i = 0; i < m.get_rows(); i++){
6          for (int j = 0; j < m.get_cols(); j++) {
7              if (m(i, j) < 0) {
8                  m2(i, j) = 0;
9              } else {
10                 m2(i, j) = m(i, j);
11             }
12         }
13     }
14     return m2;
15 }
16
17 Matrix activation::softmax(const Matrix& m) {
18     Matrix m2(m.get_rows(), m.get_cols());
19     float exp_sum = 0;
20     for (int i = 0; i < m.get_rows(); i++){
21         for (int j = 0; j < m.get_cols(); j++) {
22             exp_sum += (float) std::exp(m(i, j));
23             m2(i, j) = (float) std::exp(m(i, j));
24         }
25     }
26     for (int i = 0; i < m.get_rows(); i++){
27         for (int j = 0; j < m.get_cols(); j++) {
28             m2(i, j) = m2(i, j) / exp_sum;
29         }
30     }
31     return m2;
32 }
```

5 ex4/Dense.h

```
1  #ifndef DENSE_H
2  #define DENSE_H
3
4  #include "Activation.h"
5  #include "Matrix.h"
6
7  typedef Matrix (*activation_func_pointer)(const Matrix& m);
8  // Insert Dense class here...
9  class Dense {
10 public:
11     // constructor
12     Dense(const Matrix w, const Matrix b, activation_func_pointer func):
13         weights(w), bias(b), activation(func) {};
14     // get weights matrix
15     Matrix get_weights() const {return weights;};
16     // get bias matrix
17     Matrix get_bias() const {return bias;};
18     // get activation func
19     activation_func_pointer get_activation() const {return activation;};
20     // parenthesis operator
21     Matrix operator() (const Matrix& m) const;
22 private:
23     Matrix weights;
24     Matrix bias;
25     activation_func_pointer activation;
26 };
27
28 #endif //DENSE_H
```

6 ex4/Dense.cpp

```
1  #include "Dense.h"
2
3  Matrix Dense::operator() (const Matrix& m) const {
4      Matrix m1 = activation((weights * m) + bias);
5      return m1;
6  }
```


7 ex4/Matrix.h

```
1  // Matrix.h
2  #ifndef MATRIX_H
3  #define MATRIX_H
4
5  #include <iostream>
6  #include <fstream>
7  #include <cmath>
8  /**
9   * @struct matrix_dims
10   * @brief Matrix dimensions container. Used in MlpNetwork.h and main.cpp
11   */
12  typedef struct matrix_dims
13  {
14      int rows, cols;
15  } matrix_dims;
16
17  // Insert Matrix class here...
18  class Matrix {
19  public:
20      // constructor that gets matrix dimensions and initializes zeros matrix of that size
21      Matrix(int rows, int cols);
22      // constructor that gets nothing and initializes a 1x1 matrix
23      Matrix(): Matrix(1, 1) {};
24      // constructor that gets matrix and initializes new identical one
25      Matrix(const Matrix& m);
26      // destructor
27      ~Matrix() {
28          delete[] matrix_arr;
29      };
30
31      // get amount of rows
32      int get_rows() const {return dim.rows;};
33      // get amount of cols
34      int get_cols() const {return dim.cols;};
35      // transpose a matrix (returns reference to self so that b.transpose().transpose() is legal)
36      Matrix& transpose();
37      // vectorize a matrix
38      Matrix& vectorize();
39      // print matrix elements
40      void plain_print() const;
41      // return multiplication matrix of current matrix with given one
42      Matrix dot(const Matrix m) const;
43      // return frobenius norm of matrix
44      float norm() const;
45
46      // Matrix addition
47      Matrix operator+ (const Matrix& m) const;
48      // Matrix assignment
49      Matrix& operator= (const Matrix& m);
50      // Matrix multiplication
51      Matrix operator* (const Matrix& m) const;
52      // Scalar multiplication from right
53      Matrix operator* (float c) const;
54      // Scalar multiplication from left
55      friend Matrix operator* (float c, Matrix& m);
56      // Matrix addition accumulation
57      Matrix& operator+= (Matrix& m);
58      // () indexing
59      float &operator() (int i, int j);
```

```

60     float operator() (int i, int j) const;
61     // [] indexing
62     float &operator[] (int i);
63     float operator[] (int i) const;
64     // pretty export of matrix
65     friend std::ostream &operator<< (std::ostream& ostream, Matrix& m);
66     // read file content into matrix
67     friend std::istream &operator>> (std::istream& ifs, Matrix& m);
68 private:
69     matrix_dims dim;
70     float* matrix_arr;
71 };
72 #endif //MATRIX_H

```

8 ex4/Matrix.cpp

```
1  #include "Matrix.h"
2
3  #define LENGTH_ERROR_MESSAGE "error relating to invalid dimensions has occurred"
4  #define OUT_OF_RANGE_ERROR_MESSAGE "error relating to inaccessibility has occurred"
5  #define RUNTIME_ERROR_MESSAGE "error relating to invalid input has occurred"
6  #define DOUBLE_ASTERISK "***"
7  #define DOUBLE_SPACE " "
8  #define ASTERISK_THRESHOLD 0.1
9
10 Matrix::Matrix(int rows, int cols):
11 dim({rows, cols}), matrix_arr(nullptr)
12 {
13     if (rows <= 0 || cols <= 0) {
14         throw std::length_error (LENGTH_ERROR_MESSAGE);
15     }
16     matrix_arr = new float[rows * cols]();
17 }
18
19 Matrix::Matrix(const Matrix& m):
20 dim({m.dim.rows, m.dim.cols}), matrix_arr(new float[m.dim.rows * m.dim.cols])
21 {
22     for (int i = 0; i < dim.rows * dim.cols; i++) {
23         matrix_arr[i] = m[i];
24     }
25 }
26
27 Matrix& Matrix::transpose() {
28     Matrix temp(*this);
29     // swap rows and cols
30     dim.rows = dim.rows + dim.cols;
31     dim.cols = dim.rows - dim.cols;
32     dim.rows = dim.rows - dim.cols;
33     for (int i = 0; i < dim.rows; i++){
34         for (int j = 0; j < dim.cols; j++) {
35             (*this)(i, j) = temp(j, i);
36         }
37     }
38     return *this;
39 }
40
41 Matrix& Matrix::vectorize() {
42     // redefine matrix's rows and cols to be 1 column
43     dim.rows = dim.rows * dim.cols;
44     dim.cols = 1;
45     return *this;
46 }
47
48 void Matrix::plain_print() const {
49     for (int i = 0; i < dim.rows; i++){
50         for (int j = 0; j < dim.cols; j++) {
51             std::cout << matrix_arr[i * dim.cols + j] << " ";
52         }
53         std::cout << "\n";
54     }
55 }
56
57 Matrix Matrix::dot(const Matrix m) const {
58     // verify dimensions validity
59     if (m.dim.rows != dim.rows || m.dim.cols != dim.cols) {
```

```

60         throw std::length_error (LENGTH_ERROR_MESSAGE);
61     }
62     // create dot product matrix
63     Matrix dot_matrix(dim.rows, dim.cols);
64     for (int i = 0; i < dim.rows; i++){
65         for (int j = 0; j < dim.cols; j++) {
66             dot_matrix(i, j) = m(i, j) * matrix_arr[i * dim.cols + j];
67         }
68     }
69     return dot_matrix;
70 }
71
72 float Matrix::norm() const {
73     float sum = 0;
74     for (int i = 0; i < dim.rows; i++){
75         for (int j = 0; j < dim.cols; j++) {
76             sum += matrix_arr[i * dim.cols + j] * matrix_arr[i * dim.cols + j];
77         }
78     }
79     return sqrtf(sum);
80 }
81
82 Matrix Matrix::operator+ (const Matrix& m) const {
83     if (m.dim.rows != dim.rows || m.dim.cols != dim.cols) {
84         throw std::length_error (LENGTH_ERROR_MESSAGE);
85     }
86     // create addition matrix
87     Matrix addition_matrix(dim.rows, dim.cols);
88     for (int i = 0; i < dim.rows; i++){
89         for (int j = 0; j < dim.cols; j++) {
90             addition_matrix(i, j) = m(i, j) + matrix_arr[i * dim.cols + j];
91         }
92     }
93     return addition_matrix;
94 }
95
96 Matrix& Matrix::operator= (const Matrix& m) {
97     delete[] matrix_arr;
98     dim.rows = m.dim.rows;
99     dim.cols = m.dim.cols;
100    matrix_arr = new float[dim.rows * dim.cols];
101    for (int i = 0; i < dim.rows; i++){
102        for (int j = 0; j < dim.cols; j++) {
103            matrix_arr[i * dim.cols + j] = m(i, j);
104        }
105    }
106    return *this;
107 }
108
109 Matrix Matrix::operator* (const Matrix& m) const {
110     if (dim.cols != m.get_rows()) {
111         throw std::length_error (LENGTH_ERROR_MESSAGE);
112     }
113     Matrix matrix_mult(dim.rows, m.get_cols());
114     for (int i = 0; i < dim.rows; i++){
115         for (int j = 0; j < m.get_cols(); j++) {
116             // for each entry in new matrix, sum i'th row of self multiplied with j'th col of m
117             for (int k = 0; k < dim.cols; k++) {
118                 matrix_mult(i, j) += matrix_arr[i * dim.cols + k] * m(k, j);
119             }
120         }
121     }
122     return matrix_mult;
123 }
124
125 Matrix Matrix::operator* (float c) const {
126     Matrix scalar_mult(dim.rows, dim.cols);
127     for (int i = 0; i < dim.rows; i++){

```

```

128         for (int j = 0; j < dim.cols; j++) {
129             scalar_mult(i, j) = c * scalar_mult(i, j);
130         }
131     }
132     return scalar_mult;
133 }
134
135 Matrix operator* (float c, Matrix& m) {
136     return (m * c);
137 };
138
139 Matrix& Matrix::operator+= (Matrix& m) {
140     if (m.get_rows() != dim.rows || m.get_cols() != dim.cols) {
141         throw std::length_error (LENGTH_ERROR_MESSAGE);
142     }
143     for (int i = 0; i < dim.rows; i++){
144         for (int j = 0; j < dim.cols; j++) {
145             matrix_arr[i * dim.cols + j] += m(i, j);
146         }
147     }
148     return *this;
149 }
150
151 float &Matrix::operator() (int i, int j) {
152     if (i >= dim.rows || i < 0 || j >= dim.cols || j < 0) {
153         throw std::out_of_range (OUT_OF_RANGE_ERROR_MESSAGE);
154     }
155     return matrix_arr[i * dim.cols + j];
156 }
157
158 float Matrix::operator() (int i, int j) const {
159     if (i >= dim.rows || i < 0 || j >= dim.cols || j < 0) {
160         throw std::out_of_range (OUT_OF_RANGE_ERROR_MESSAGE);
161     }
162     return matrix_arr[i * dim.cols + j];
163 }
164
165 float &Matrix::operator[] (int i) {
166     if (i >= dim.rows * dim.cols || i < 0) {
167         throw std::out_of_range (OUT_OF_RANGE_ERROR_MESSAGE);
168     }
169     return matrix_arr[i];
170 }
171
172 float Matrix::operator[] (int i) const {
173     if (i >= dim.rows * dim.cols || i < 0) {
174         throw std::out_of_range (OUT_OF_RANGE_ERROR_MESSAGE);
175     }
176     return matrix_arr[i];
177 }
178
179 std::ostream &operator<< (std::ostream& ostream, Matrix& m) {
180     for (int i = 0; i < m.get_rows(); i++){
181         for (int j = 0; j < m.get_cols(); j++) {
182             if (m(i, j) > ASTERISK_THRESHOLD) {
183                 ostream << DOUBLE_ASTERISK;
184             } else {
185                 ostream << DOUBLE_SPACE;
186             }
187         }
188         ostream << "\n";
189     }
190     return ostream;
191 }
192
193 std::ifstream &operator>> (std::ifstream& ifs, Matrix& m) {
194     ifs.read((char *) m.matrix_arr, sizeof (float) * m.get_rows() * m.get_cols());
195     if (!ifs) {

```

```
196         throw std::runtime_error(RUNTIME_ERROR_MESSAGE);
197     }
198     return ifs;
199 }
200
```

9 ex4/MlpNetwork.h

```
1  //MlpNetwork.h
2
3  #ifndef MLPNETWORK_H
4  #define MLPNETWORK_H
5
6  #include "Dense.h"
7
8  #define MLP_SIZE 4
9
10 /**
11  * @struct digit
12  * @brief Identified (by Mlp network) digit with
13  *       the associated probability.
14  * @var value - Identified digit value
15  * @var probability - identification probability
16  */
17 typedef struct digit {
18     unsigned int value;
19     float probability;
20 } digit;
21
22 const matrix_dims img_dims = {28, 28};
23 const matrix_dims weights_dims[] = {{128, 784},
24                                     {64, 128},
25                                     {20, 64},
26                                     {10, 20}};
27 const matrix_dims bias_dims[] = {{128, 1},
28                                  {64, 1},
29                                  {20, 1},
30                                  {10, 1}};
31
32 // Insert MlpNetwork class here...
33 class MlpNetwork {
34 public:
35     // constructor
36     MlpNetwork(Matrix weights[MLP_SIZE], Matrix biases[MLP_SIZE]);
37     // parenthesis operator
38     digit operator() (const Matrix& m) const;
39 private:
40     Dense dense_arr[MLP_SIZE];
41 };
42 #endif // MLPNETWORK_H
```

10 ex4/MlpNetwork.cpp

```
1  #include "MlpNetwork.h"
2
3  #define LENGTH_ERROR_MESSAGE "error relating to invalid dimensions has occurred"
4
5  MlpNetwork::MlpNetwork (Matrix weights[MLP_SIZE], Matrix biases[MLP_SIZE]):
6      dense_arr{Dense(weights[0], biases[0], (activation::relu)), Dense(weights[1], biases[1], activation::relu),
7                Dense(weights[2], biases[2], activation::relu), Dense(weights[3], biases[3], activation::softmax)}
8  {
9      if (weights[0].get_rows() != weights[1].get_cols() || weights[1].get_rows() != weights[2].get_cols() ||
10         weights[2].get_rows() != weights[3].get_cols()) {
11          throw std::length_error(LENGTH_ERROR_MESSAGE);
12      }
13  }
14
15
16  digit MlpNetwork::operator() (const Matrix& m) const {
17      Matrix output = m;
18      // run all stages
19      for (int i = 0; i < MLP_SIZE; i++) {
20          output = dense_arr[i](output);
21      }
22      // find maximum probability in output
23      digit max = {0, output[0]};
24      for (int j = 0; j < output.get_rows(); j++) {
25          if (output[j] > max.probability) {
26              max = {(unsigned) j, output[j]};
27          }
28      }
29      return max;
30  }
```