МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«**Вятский государственный университет**»

**(«ВятГУ»)**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Отчет по лабораторной работе №2

по дисциплине «Параллельные вычисления на графических процессорах»

Выполнил студент группы ИВТ-32 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Щесняк Д. С./

Проверил доцент кафедры ЭВМ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Вожегов Д. В./

Киров 2017

1. Задание на лабораторную работу

Целью лабораторной работы является разработка линейного программного кода на основе алгоритма, разработанного в ходе выполнения предыдущей лабораторной работы.

1. Блок-схемы алгоритма

Блок-схемы алгоритма представлены на рисунках 1, 2, 3.



Рисунок 1 – Блок-схема основного алгоритма



Рисунок 2 – Блок-схема функции обучения



Рисунок 3 – Блок-схема функции поиска индекса максимального элемента

1. Исходный код

Исходный код модуля Matrix, в котором описываются все операции над матрицами представлен на рисунке 4. Исходный код модуля LogisticRegression, в котором описывается класс логистической регрессии представлен на рисунке 5. Исходный код основной программы представлен на рисунке 6.

|  |
| --- |
| #pragma once  #include <cstdlib>  #include <iostream>  namespace matrix {  struct Matrix  {  float\*\* value;  int lines;  int cols;  };  typedef float mapFunc(float arg);  Matrix\* zeros(int lines, int cols, float);  Matrix\* ones(int lines, int cols, float);  Matrix\* randoms(int lines, int cols, float min, float max, int seed);  template <typename t>  Matrix\* mul(Matrix\* a, t b, int clr);  Matrix\* sum(Matrix\* a, Matrix\* b, int clear);  Matrix\* sub(Matrix\* a, Matrix\* b, int clear);  Matrix\* mul(Matrix\* a, Matrix\* b, int clear);  Matrix\* matmul(Matrix\* a, Matrix\* b, int clear);  Matrix\* transpose(Matrix\* a, int clear);  Matrix\* map(Matrix\* a, mapFunc f, int clear);  Matrix\* getLine(Matrix\* a, int line);  Matrix\* getMax(Matrix\* a);  Matrix\* print(Matrix\* a);  void clear(Matrix\* a);    bool cmp(Matrix\* a, Matrix\* b) {  for (int i = 0; i < a->lines; i++) {  for (int j = 0; j < a->cols; j++) {  if (a->value[i][j] != b->value[i][j]) return false;  }  }  return true;  }  Matrix\* zeros(int lines, int cols, float num=0) {  Matrix\* m = new Matrix();  m->lines = lines;  m->cols = cols;  m->value = new float\*[lines];  for (int i = 0; i < lines; i++) {  m->value[i] = new float[cols];  for (int j = 0; j < cols; j++) {  m->value[i][j] = num;  }  }  return m;  }  Matrix\* ones(int lines, int cols, float num=1) {  return zeros(lines, cols, num);  }  Matrix\* randoms(int lines, int cols, float min=-1, float max=1, int seed=0) {  Matrix\* m = new Matrix();  m->lines = lines;  m->cols = cols;  srand(seed);  m->value = new float\*[lines];  for (int i = 0; i < lines; i++) {  m->value[i] = new float[cols];  for (int j = 0; j < cols; j++) {  m->value[i][j] = ((float) rand() / RAND\_MAX) \* (abs(max) + abs(min)) + min;  }  }  return m;  }  template <typename t>  Matrix\* mul(Matrix\* a, t b, int clr=3) {  Matrix\* m = zeros(a->lines, a->cols);  #pragma omp parallel for  for (int i = 0; i < a->lines; i++) {    for (int j = 0; j < a->cols; j++) {  m->value[i][j] = a->value[i][j] \* b;  }  }  if (clr & 1 == 1) {  clear(a);  }  return m;  }  Matrix\* sum(Matrix\* a, Matrix\* b, int clr=3) {  if (a->cols != b->cols) {  return NULL;  }  Matrix\* m;  if (clr & 1) m = a;  else if (clr & 2) m = b;  else m = zeros(a->lines, a->cols);  #pragma omp parallel for  for (int i = 0; i < a->lines; i++) {    for (int j = 0; j < a->cols; j++) {  m->value[i][j] = a->value[i][j] + b->value[i % b->lines][j];  }  }  if ((clr & 1) && (clr & 2)) {  clear(b);  }  return m;  }  Matrix\* sub(Matrix\* a, Matrix\* b, int clr=3) {  if (a->cols != b->cols) {  return NULL;  }    Matrix\* m;  if (clr & 1) m = a;  else if (clr & 2) m = b;  else m = zeros(a->lines, a->cols);  #pragma omp parallel for  for (int i = 0; i < a->lines; i++) {    for (int j = 0; j < a->cols; j++) {  m->value[i][j] = a->value[i][j] - b->value[i % b->lines][j];  }  }              if ((clr & 1) && (clr & 2)) {  clear(b);  }  return m;  }  Matrix\* mul(Matrix\* a, Matrix\* b, int clr=3) {  if (a->cols != b->cols) {  return NULL;  }    Matrix\* m;  if (clr & 1) m = a;  else if (clr & 2) m = b;  else m = zeros(a->lines, a->cols);  #pragma omp parallel for  for (int i = 0; i < a->lines; i++) {    for (int j = 0; j < a->cols; j++) {  m->value[i][j] = a->value[i][j] \* b->value[i % b->lines][j];  }  }              if ((clr & 1) && (clr & 2)) {  clear(b);  }  return m;  }  Matrix\* matmul(Matrix\* a, Matrix\* b, int clr=3) {  if (a->cols != b->lines) {  return NULL;  }  Matrix\* m = zeros(a->lines, b->cols);  for (int k = 0; k < b->cols; k++) {  for (int i = 0; i < a->lines; i++) {  #pragma omp parallel for  for (int j = 0; j < a->cols; j++) {  m->value[i][k] += a->value[i][j] \* b->value[j][k];  }  }  }  if (clr & 1) {  clear(a);  }  if (clr & 2) {  clear(b);  }  return m;  }  Matrix\* transpose(Matrix\* a, int clr=3) {  Matrix\* m = zeros(a->cols, a->lines);  for (int i = 0; i < a->lines; i++) {  for (int j = 0; j < a->cols; j++) {  m->value[j][i] = a->value[i][j];  }  }  if (clr & 1) {  clear(a);  }  return m;  }  Matrix\* map(Matrix\* a, mapFunc f, int clr=3) {  Matrix\* m = zeros(a->lines, a->cols);  for (int i = 0; i < a->lines; i++) {  for (int j = 0; j < a->cols; j++) {  m->value[i][j] = f(a->value[i][j]);  }  }  if (clr & 1) {  clear(a);  }  return m;  }  Matrix\* getLine(Matrix\* a, int line) {  Matrix\* m = zeros(1, a->cols);  for (int i = 0; i < a->cols; i++) {  m->value[0][i] = a->value[line][i];  }  return m;  }  Matrix\* getMax(Matrix\* a) {  Matrix\* m = zeros(a->lines, 1);  for (int i = 0; i < a->lines; i++){  int k = 0;  for (int j = 1; j < a->cols; j++) {  if (a->value[i][k] < a->value[i][j]) k = j;  }  m->value[i][0] = k;  }  return m;  }  Matrix\* print(Matrix\* a) {  for (int i = 0; i < a->lines; i++) {  for (int j = 0; j < a->cols; j++) {  std::cout << a->value[i][j] << "\t";  }  std::cout << std::endl;  }  std::cout << std::endl;  return a;  }  void clear(Matrix\* a) {  for (int i = 0; i < a->lines; i++) {  delete[] a->value[i];  }  delete[] a->value;  delete a;  }  } |

Рисунок 4 – Исходный код модуля Matrix

|  |
| --- |
| #pragma once  #include "Matrix.h"  using namespace matrix;  typedef float lFunc(int inter);  float sigmoid(float a) {  return 1.0 / (1.0 + exp(-a));  }  float lin(float a) {  return a > 0 ? 1 : 0;  }  float th(float a) {  return (tanh(a) + 1) / 2;  }  float a1(int i) {  return 1.0 / i;  }  float a0(int i) {  return 1.0;  }  class LogisticRegression  {  private:  Matrix\* W;  Matrix\* b;  mapFunc\* activate;  lFunc\* learnCoeff;  public:  LogisticRegression(int features, int classes, mapFunc\* activ, lFunc\* learnCoeff) {  this->W = randoms(features, classes);  this->b = randoms(1, classes);  this->activate = activ;  this->learnCoeff = learnCoeff;  }  ~LogisticRegression();  Matrix\* predict(Matrix\* X) {  return map(sum(matmul(X, W, false), b, false), \*this->activate);  }  void fit(Matrix\* X, Matrix\* Y, int epoch=10) {  for (int e = 0; e < epoch; e++) {  for (int i = 0; i < X->lines; i++) {  Matrix\* x = getLine(X, i);  Matrix\* y = getLine(Y, i);  Matrix\* d = sub(this->predict(x), y, 1);  Matrix\* t\_x = transpose(x, 0);  this->W = sub(this->W, mul(matmul(t\_x, d, 1), learnCoeff(e + 1)));  this->b = sub(this->b, mul(matmul(ones(1, x->lines), d), learnCoeff(e + 1)));  clear(x);  clear(y);  }  }  }  void less(Matrix\* X, Matrix\* y) {  Matrix\* h = predict(X);  Matrix\* one = ones(y->lines, y->cols);  Matrix\* l = matmul(transpose(mul(y, -1)), map(h, log));  Matrix\* r = matmul(transpose(sub(one, y)), sub(one, map(h, log)));  Matrix\* res = mul(sub(l, r), 1 / y->lines);  print(res);  }  }; |

Рисунок 5 – Исходный код модуля LogisticRegression

|  |
| --- |
| #include "MnistReader.h"  #include "Matrix.h"  #include "LogisticRegression.h"  #include <ctime>  #define TRAIN 10000  #define TEST 1000  #define EPOCHES 4  using namespace matrix;  int metric(Matrix\* Y, Matrix\* pred) {    Matrix\* ans = getMax(Y);  Matrix\* predMax = getMax(pred);  int k = 0;  for (int i = 0; i < pred->lines; i++) {  if (predMax->value[i][0] == ans->value[i][0]) k++;  }  clear(ans);  clear(predMax);  return k;  }  float f(float a) {  return a\*a;  }  int main() {  MnistReader\* mr = new MnistReader("./trainBase/images.idx3-ubyte", "./trainBase/labels.idx1-ubyte");  clock\_t t;  Matrix\* X = zeros(TRAIN, 28 \* 28);  Matrix\* Y = zeros(TRAIN, 10);  for (int i = 0; i < TRAIN; i++) {  X->value[i] = mr->getMatrix();  Y->value[i][mr->label] = 1;  mr->next();  }    LogisticRegression\* lgSigm = new LogisticRegression(28 \* 28, 10, sigmoid, a0);  LogisticRegression\* lgLin = new LogisticRegression(28 \* 28, 10, lin, a0);  LogisticRegression\* lgTh = new LogisticRegression(28 \* 28, 10, th, a0);  LogisticRegression\* lgSigmA1 = new LogisticRegression(28 \* 28, 10, sigmoid, a1);  LogisticRegression\* lgLinA1 = new LogisticRegression(28 \* 28, 10, lin, a1);  LogisticRegression\* lgThA1 = new LogisticRegression(28 \* 28, 10, th, a1);    t = clock();  lgSigm->fit(X, Y, EPOCHES);  std::cout << "LogReg with Sigmoid." << " fit time = " << (float) (clock() - t) / CLOCKS\_PER\_SEC << " sec" << std::endl;  t = clock();  lgLin->fit(X, Y, EPOCHES);  std::cout << "LogReg with Linear." << " fit time = " << (float)(clock() - t) / CLOCKS\_PER\_SEC << " sec" << std::endl;  t = clock();  lgTh->fit(X, Y, EPOCHES);  std::cout << "LogReg with Th." << " fit time = " << (float)(clock() - t) / CLOCKS\_PER\_SEC << " sec" << std::endl;  t = clock();  lgSigmA1->fit(X, Y, EPOCHES);  std::cout << "LogReg with Sigmoid and learn rate a1." << " fit time = " << (float)(clock() - t) / CLOCKS\_PER\_SEC << " sec" << std::endl;  t = clock();  lgLinA1->fit(X, Y, EPOCHES);  std::cout << "LogReg with Linear and learn rate a1." << " fit time = " << (float)(clock() - t) / CLOCKS\_PER\_SEC << " sec" << std::endl;  t = clock();  lgThA1->fit(X, Y, EPOCHES);  std::cout << "LogReg with Th and learn rate a1." << " fit time = " << (float)(clock() - t) / CLOCKS\_PER\_SEC << " sec" << std::endl;  clear(X);  clear(Y);  X = zeros(TEST, 28 \* 28);  Y = zeros(TEST, 10);  for (int i = 0; i < TEST; i++) {  X->value[i] = mr->getMatrix();  Y->value[i][mr->label] = 1;  mr->next();  }  std::cout << "LogReg with Sigmoid." << " Accuracy = " << (float)metric(Y, lgSigm->predict(X)) / X->lines \* 100 << "%" << std::endl;  std::cout << "LogReg with Linear." << " Accuracy = " << (float)metric(Y, lgLin->predict(X)) / X->lines \* 100 << "%" << std::endl;  std::cout << "LogReg with Th." << " Accuracy = " << (float)metric(Y, lgTh->predict(X)) / X->lines \* 100 << "%" << std::endl;  std::cout << "LogReg with Sigmoid and learn rate a1." << " Accuracy = " << (float)metric(Y, lgSigmA1->predict(X)) / X->lines \* 100 << "%" << std::endl;  std::cout << "LogReg with Linear and learn rate a1." << " Accuracy = " << (float)metric(Y, lgLinA1->predict(X)) / X->lines \* 100 << "%" << std::endl;  std::cout << "LogReg with Th and learn rate a1." << " Accuracy = " << (float)metric(Y, lgThA1->predict(X)) / X->lines \* 100 << "%" << std::endl;    system("pause");  return 0;  } |

Рисунок 6 – Исходный код основной программы

1. Оценка времени выполнения

Среднее время выполнения обучения одной логистической регрессии 109 секунд. Результаты измерения времени представлены на рисунке 7

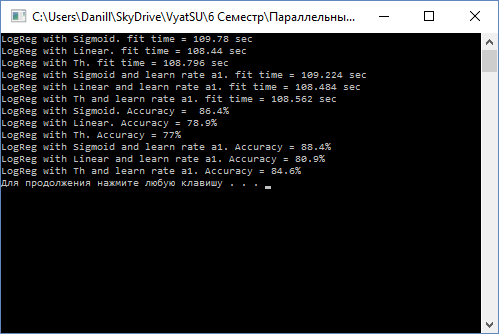


Рисунок 7 – Результаты измерения времени

1. Вывод

В ходе выполнения лабораторной работы был реализованы алгоритмы обучения и тестирования модели логистической регрессии, так же были разработаны функции выполняющие основные операции над матрицами: сложение, умножение, вычитание и тд. Было измерено время выполнения линейной программы без использования средств распараллеливания. Среднее время обучения одной логистической регрессии получилось равным 109 секундам. Следующим этапом выполнения курса лабораторных работ будет уменьшение времени работы программы за счет введения параллелизма.