中南大學
CENTRAL SOUTH UNIVERSITY

模式识别与机器学习课程设计报告

题　　目：基于 BP 网络进行手写数字识别

学生姓名：饶逸石

指导老师：高琰

学　　院：自动化学院

专业班级：智能 2103

2023 年 7 月

基于 BP 网络进行手写数字识别

摘要

本研究采用了反向传播（Backpropagation，BP）神经网络对 MNIST 手写数字识别任务进行性能优化。实验中，我们使用了动态学习率和多线程训练两种优化策略。结果显示，动态学习率在提升模型准确性上表现优异，多线程训练有效提高了模型训练效率。该神经网络模型经过优化后，在手写数字识别任务上表现出了良好的准确性和性能。

关键词： BP 网络　手写数字识别

目录

第 1 章　概述

1.1　研究内容

在本次实验中，我研究了如何使用简单的反向传播神经网络对 MNIST 手写数字，并进行了计算策略的优化。我设计了一个单隐藏层的神经网络，784 个输入节点对应了 28x28 像素的手写数字图像，30 个隐层节点用于计算，10 个输出节点使用 onehot 表示输出。在进行模型优化的过程中，我前后实验了两种策略：多线程训练和动态学习率调整。多线程训练显著提高了模型的训练效率，有效地节约了训练时间；而动态学习率调整进一步提高了模型的准确性，并有效地避免了过拟合。实验结果显示，经过优化后的神经网络模型在 MNIST 数据集上表现出了高精度的识别能力。然而，此模型在不同数据和任务需求下可能需要进行适当的调整和优化。

1.2　数据集

MNIST 数据集是一个广泛使用的手写数字识别数据集，由美国国家标准与技术研究院 (National Institute of Standards and Technology, NIST) 创建。它是最早的计算机视觉数据集之一，常被用作图像识别算法的基准测试。

该数据集包含 60,000 个训练样本和 10,000 个测试样本，每个样本都是一个 28x28 像素的灰度图像，代表从 0 至 9 的一个手写数字。每个图像都是中心化和规范化的，这意味着数字位于图像中心，且具有相同的尺度和方向。

图像的每个像素值都在 0 到 255 之间，其中 0 表示白色，255 表示黑色，其余数值表示不同的灰度级别。这些像素值可以被视为描述图像的特征，可以直接用于模型的训练。

MNIST 数据集因其规模适中且问题相对简单，常被用作初步测试机器学习和深度学习算法的性能。尽管现在有许多其他更复杂的数据集可供研究，但 MNIST 数据集仍然是验证新算法或理念的重要基准之一。

第 2 章　算法描述

2.1　BP 算法简述

2.1.1　算法思想

反向传播（Backpropagation）是一种应用于多层人工神经网络的梯度下降算法。该算法使用链式法则，以网络每层的权重为变量计算损失函数的梯度，以此更新权重，从而最小化损失函数。

反向传播算法（BP 算法）主要由两个阶段组成：激励传播与权重更新。

第一阶段：激励传播

每次迭代中的传播环节包含两步：

1.（前向传播阶段）将训练输入送入网络以获得预测结果。
2.（反向传播阶段）对预测结果与训练目标进行差值运算（即损失函数的计算）。

第二阶段：权重更新

对于每个突触上的权重，按照以下步骤进行更新：

1. 将输入激励与回应误差相乘，从而计算权重的梯度。
2. 将此梯度乘以一个比例因子并取反，然后加到权重上。

这个比例因子将会影响训练过程的速度和效果，因此将其称为” 训练因子”。由于梯度的方向指向误差增大的方向，在更新权重时需将其取反，以减小权重引起的误差。

这两个阶段可以反复进行，直到网络对输入的响应达到预定的目标范围。

2.1.2　算法描述

1. **权重初始化**：这是神经网络训练的第一步。一般会选择较小的随机值作为初始权重，这种做法有助于神经网络更有效地从数据中学习。
2. **前向传播**：在神经网络的每一层中，输入值会乘以权重并加上一个偏置，然后通过一个激活函数，如 Sigmoid 函数，ReLU 函数等进行非线性变换。具体到每一个神经元，假设其输入为 x，权重为 w，偏置为 b，那么该神经元的加权和可以表示为：

$$z = w \cdot x + b$$

然后，z 被送入激活函数 f，得到该神经元的输出值：

$$y = f(z)$$

3. **误差计算**：通过比较神经网络的预测输出和实际目标值来计算误差。误差通常使用损失函数来计算，例如，最常用的均方误差损失函数可以表示为：

$$E = \frac{1}{2}\sum(target - output)^2$$

4. **反向传播误差**：这是反向传播算法的关键步骤。需要计算损失函数相对于每个权重的梯度。这一步使用了链式法则，根据损失函数的导数和每一层输出的导数，逐层向后计算误差的梯度。误差梯度 δ 可以表示为：

$$\delta = (target - output) \cdot f'(z)$$

其中，$f'(z)$ 是激活函数的导数。

5. **权重更新**：根据计算出的误差梯度，进行权重的更新。权重的更新遵循梯度下降法则，即向误差减小的方向调整权重。新的权重是通过旧的权重减去学习率乘以误差梯度得到的：

$$w_{new} = w_{old} - \text{learning\_rate} \cdot \delta$$

以上步骤会在每个训练样本上重复执行，直到网络的预测误差满足预设的阈值，或者达到预设的迭代次数为止。在每个训练周期内，前向传播和反向传播过程都会执行一次。

第 3 章　实验结果与分析

实验中，首先进行了基于单线程的直接训练作为测试基准，然后，为了提高训练效率，引入了多线程训练方法，这种方法将数据集分成多个部分并同时进行训练，可以显著地减少训练时间，使得神经网络可以更快地收敛。最后，采用了动态学习率调整策略，这种策略将学习率在训练过程中逐渐减小，这样做可以提高模型的准确性并避免过拟合。逐渐减小的学习率也能帮助模型在训练早期快速接近最优解，然后在训练后期更精细地进行调整。

3.1　训练结果

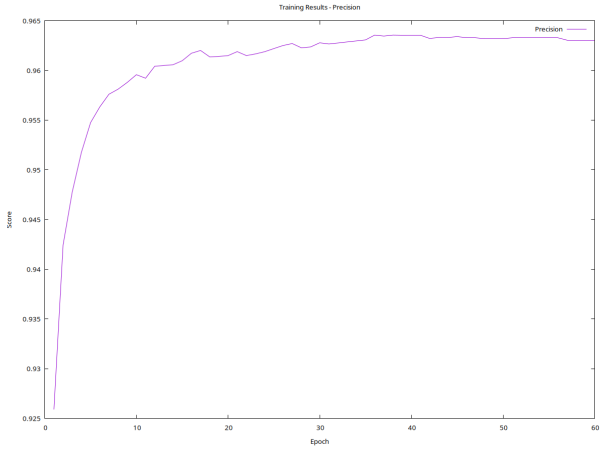
在实验中我们使用精度（Precision），召回率（Recall），以及 F1 作为测试标准。精度是预测为正例的样本中真正正例的比例，召回率是实际正例被预测正确的比例，而 F1 Score 则是精度和召回率的调和平均值，能同时反映模型的精度和召回率。这三个指标可以全面评价模型的性能。

接下来，将通过表格和图表展示实验的结果。我们对模型在测试集上的表现进行了记录，包括各自的精度、召回率和 F1 值，并将这些数据整合表1-0中。

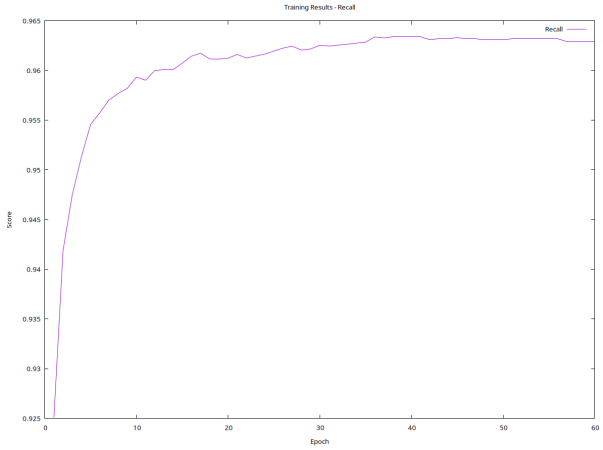
表 1-0　特殊点训练结果表

	epoch	precision	recall	F1 score	time
1	1	0.925898	0.925072	0.924915	2
2	10	0.959579	0.959338	0.959403	19
3	20	0.961497	0.961233	0.96132	38
4	30	0.962786	0.962541	0.962621	57
5	40	0.963542	0.963396	0.963427	76
6	50	0.963211	0.9631	0.963115	96
7	60	0.963025	0.9629	0.962922	116

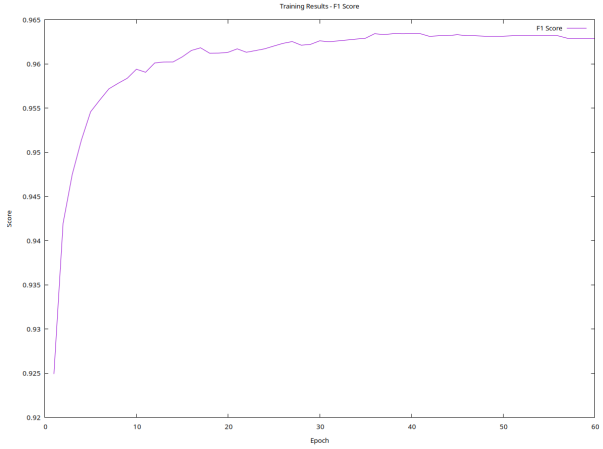
图1-1了给出三种评价指标相对于 epoch 的变化趋势。



(a) precision



(b) recall



(c) f1 score

图 1-1　Result-epoch 图

3.2　结果分析

3.2.1　模型训练结果

(1)　动态学习率

1. 精度：模型的精度在训练过程中逐步提升，并在第 12 轮达到约 0.96 的水平，之后

精度的增长趋于平缓，保持在 0.963 的水平。这可能意味着模型在 12 轮后已经收敛，因为进一步的训练带来的信息增益较小。

2. 时间性能：每轮训练所需的时间大致线性增长，从最初的 2 秒到最后一轮的 116 秒，每轮训练大约需 2 秒。这个表现在合理范围内，并且显著优于单线程训练。

(2) 多线程固定学习率

1. 精度：模型的精度在训练初期迅速上升，然后在第 7 轮时达到峰值，约为 0.9577。然而，随后的训练过程中，精度出现轻微下滑，并保持在一定的波动区间。
2. 时间性能：每轮训练所需的时间也大致线性增长，从最初的 2 秒到最后一轮的 274 秒，每轮训练大约需 2 秒。然而，相同 epoch 下，多线程固定学习率所得到的模型精度相对较低。

3.2.2 综合分析

在动态学习率和多线程固定学习率两种训练策略中，动态学习率在整体的精度上表现更优。它能在较少的 epoch 中达到较高的精度，并且在后续的训练过程中，精度的变动非常小，表现出了很好的稳定性。相比之下，虽然多线程固定学习率在时间性能上与动态学习率相当，但其模型精度较低，在相同的 epoch 下，多线程固定学习率所训练的模型精度略低于动态学习率模型。

第 4 章　实验结论

本次实验设计了一个单一隐藏层的神经网络，通过对其进行深入研究和多次实验，得到了一些重要的结论。首先，网络的设计在图像识别的场景中具有很好的适应性和准确性，输入层的节点数量恰好对应 28x28 像素的图像，使得模型能够充分利用图像信息进行有效学习。

在进行模型优化的过程中，我采用了多线程训练和动态学习率调整两种策略。实验结果表明，多线程训练有效地提升了模型的训练效率，大幅度减少了训练时间，而动态学习率调整策略则进一步提升了模型的准确性并有效避免了过拟合，两者配合使用使得模型达到了最佳的学习效果。

从实验结果来看，我的神经网络模型在经过适当的训练和优化后，表现出了较高的准确性，证明了我们的设计和实施策略是有效的。

然而，值得注意的是，虽然模型在本实验中表现出了良好的性能，但这并不能保证它在所有情况下都能有同样的效果。因此，在将该模型应用到其他场景或问题之前，应该先进行适当的调整和优化，以确保其能够适应不同的数据和任务需求。

第 5 章　参考文献

5.1　参考文章

Principles of training multi-layer neural network using backpropagation: https://wiki.eecs.yorku.ca/course_archive/2011-12/F/4403/_media/backpropagation.pdf

一篇关于神经网络实现的网络博客: https://mp.weixin.qq.com/s?__biz=MzU2MDAyNzk5MA==&mid=2247483953&idx=1&sn=6f09647ba35beaff6ac4965d78f645c7&chksm=fc0f0208cb788b1ee1fard

5.2　参考书籍

周志华《机器学习》

李航《统计学习方法》

附录 A　附录代码

附录中给出了本次实验所使用的代码，本次实验的其他更多内容可以访问:https://github.com/Lambert-Rao/md_notes/tree/main/Homework/ML_homework获取。

A.1　BP 网络 c++ 代码

```
#include <cstdio>
#include <vector>
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <chrono>
#include <thread>
#include <mutex>

using namespace std;

mutex mtx;

vector<vector<double>> train_images;          // 训练集图像
vector<double> train_labels;                    // 训练集标签
vector<vector<double>> test_images;           // 测试集图像
vector<double> test_labels;                     // 测试集标签

const double initial_learning_rate = 0.08; // 初始学习率
const double decay_rate = 0.1; //衰减率
double learning_rate = initial_learning_rate;
void update_learning_rate(int epoch) {
    learning_rate = initial_learning_rate * exp(-decay_rate * epoch);
}

const int epoch = 150;                          // 学习轮次
const int nh = 30;                              // 隐藏单元数量（单隐藏层）
double w1[784][nh];                           // 输入层到隐藏层的权重矩阵
double w2[nh][10];                            // 隐藏层到输出层的权重矩阵
double bias1[nh];                             // 隐藏层的偏置
double bias2[10];                             // 输出层的偏置

int e; //迭代次数

chrono::steady_clock::time_point start_time;

void test();
```

```
int reverse_int(int i)
{
    ///用于读取MNIST数据时，将高位在前的整数转为高位在后的整数。
    unsigned char ch1, ch2, ch3, ch4;
    ch1 = i & 255;
    ch2 = (i >> 8) & 255;
    ch3 = (i >> 16) & 255;
    ch4 = (i >> 24) & 255;
    return ((int)ch1 << 24) + ((int)ch2 << 16) + ((int)ch3 << 8) + ch4;
}

// 载入训练集图像
void read_train_images()
{
    ifstream file("train-images-idx3-ubyte", ios::binary);
    if (file.is_open())
    {
        int magic_number = 0;
        int number_of_images = 0;
        int row = 0;
        int col = 0;

        file.read((char *)&magic_number, sizeof(magic_number));
        file.read((char *)&number_of_images, sizeof(number_of_images));
        file.read((char *)&row, sizeof(row));
        file.read((char *)&col, sizeof(col));

        magic_number = reverse_int(magic_number);
        number_of_images = reverse_int(number_of_images);
        row = reverse_int(row);
        col = reverse_int(col);

        for (int i = 0; i < number_of_images; i++)
        {
            vector<double> this_image;
            for (int r = 0; r < row; r++)
            {
                for (int c = 0; c < col; c++)
                {
                    unsigned char pixel = 0;
                    file.read((char *)&pixel, sizeof(pixel));
                    this_image.push_back(pixel);
                    this_image[r * 28 + c] /= 255; // 像素值归一化处理
                }
            }
            train_images.push_back(this_image);
        }
        printf("%d, train images success\n", train_images.size());
    }
```

```
    else
    {
        printf("open train images failed\n");
        exit(1);
    }
}

// 载入训练集标签
void read_train_labels()
{
    ifstream file;
    file.open("train-labels-idx1-ubyte", ios::binary);
    if (file.is_open())
    {
        int magic_number = 0;
        int number_of_images = 0;

        file.read((char *)&magic_number, sizeof(magic_number));
        file.read((char *)&number_of_images, sizeof(number_of_images));

        magic_number = reverse_int(magic_number);
        number_of_images = reverse_int(number_of_images);

        for (int i = 0; i < number_of_images; i++)
        {
            unsigned char label = 0;
            file.read((char *)&label, sizeof(label));
            train_labels.push_back((double)label);
        }
        printf("%d, train labels success\n", train_labels.size());
    }
    else
    {
        printf("open train labels failed\n");
        exit(1);
    }
}

// 载入测试集图像
void read_test_images()
{
    ifstream file("t10k-images-idx3-ubyte", ios::binary);
    if (file.is_open())
    {
        int magic_number = 0;
        int number_of_images = 0;
        int row = 0;
        int col = 0;

        file.read((char *)&magic_number, sizeof(magic_number));
```

```
        file.read((char *)&number_of_images, sizeof(number_of_images));
        file.read((char *)&row, sizeof(row));
        file.read((char *)&col, sizeof(col));

        magic_number = reverse_int(magic_number);
        number_of_images = reverse_int(number_of_images);
        row = reverse_int(row);
        col = reverse_int(col);

        for (int i = 0; i < number_of_images; i++)
        {
            vector<double> this_image;
            for (int r = 0; r < row; r++)
            {
                for (int c = 0; c < col; c++)
                {
                    unsigned char pixel = 0;
                    file.read((char *)&pixel, sizeof(pixel));
                    this_image.push_back(pixel);
                    this_image[r * 28 + c] /= 255; // 像素值归一化处理
                }
            }
            test_images.push_back(this_image);
        }
        printf("%d, test images success\n", test_images.size());
    }
    else
    {
        printf("open test images failed\n");
        exit(1);
    }
}

// 载入测试集标签
void read_test_labels()
{
    ifstream file;
    file.open("t10k-labels-idx1-ubyte", ios::binary);
    if (file.is_open())
    {
        int magic_number = 0;
        int number_of_images = 0;

        file.read((char *)&magic_number, sizeof(magic_number));
        file.read((char *)&number_of_images, sizeof(number_of_images));

        magic_number = reverse_int(magic_number);
        number_of_images = reverse_int(number_of_images);

        for (int i = 0; i < number_of_images; i++)
```

```
        {
            unsigned char label = 0;
            file.read((char *)&label, sizeof(label));
            test_labels.push_back((double)label);
        }
        printf("%d, test labels success\n", test_labels.size());
    }
    else
    {
        printf("open test labels failed\n");
        exit(1);
    }
}

// 为权重矩阵和偏置向量随机赋初值
void init_parameters()
{
    for (int i = 0; i < 784; i++)
    {
        for (int j = 0; j < nh; j++)
            w1[i][j] = rand() / (10 * (double)RAND_MAX) - 0.05;
    }

    for (int i = 0; i < nh; i++)
    {
        for (int j = 0; j < 10; j++)
            w2[i][j] = rand() / (10 * (double)RAND_MAX) - 0.05;
    }

    for (int i = 0; i < nh; i++)
        bias1[i] = rand() / (10 * (double)RAND_MAX) - 0.1;
    for (int i = 0; i < 10; i++)
        bias2[i] = rand() / (10 * (double)RAND_MAX) - 0.1;
}

// 激活函数 sigmoid，可以把 x 映射到 0 ~ 1 之间
// s(x) = 1 / (1 + e^(-x))
// s'(x) = s(x) * (1 - s(x))
double sigmoid(double x)
{
    return 1 / (1 + exp(-x));
}

// 通过图像的输入得到隐藏层的输出
vector<double> get_hidden_out(vector<double> &image)
{
    vector<double> hidden_out(nh);
    for (int i = 0; i < nh; i++)
    { // 对于每一个隐藏单元
        double sum = 0.0;
```

```
        for (int j = 0; j < 784; j++)
        {
            sum += image[j] * w1[j][i];
        }
        hidden_out[i] = sigmoid(sum + bias1[i]);
    }
    return hidden_out;
}

// 通过隐藏层的输出得到网络最终的输出
vector<double> get_z(vector<double> &hidden_out)
{
    vector<double> z(10);
    for (int i = 0; i < 10; i++)
    { // 对于每一个输出单元
        double sum = 0.0;
        for (int j = 0; j < nh; j++)
        {
            sum += hidden_out[j] * w2[j][i];
        }
        z[i] = sigmoid(sum + bias2[i]);
    }
    return z;
}

// 计算损失函数（1/2 均方误差）
double get_loss(vector<double> &z, double label)
{
    double loss = 0;
    int true_label = (int)label;
    for (int i = 0; i < 10; i++)
    {
        if (i != true_label)
            loss += z[i] * z[i];
        else
            loss += (1 - z[i]) * (1 - z[i]);
    }
    return loss / 2;
}

void train(vector<vector<double>> &images, vector<double> &labels, int start_index,
    int end_index)
{
    for (int im = start_index; im <= end_index; im++)
    {
        double grad_w1[784][nh];
        double grad_w2[nh][10];
        double grad_bias1[nh];
        double grad_bias2[10];
```

```
vector<double> hidden_out = get_hidden_out(images[im]);
vector<double> z = get_z(hidden_out);
double loss = get_loss(z, labels[im]);
int true_label = (int)labels[im];

// ----------------------------------------计算梯度
    ------------------------------------

for (int i = 0; i < 784; i++)
{
    for (int j = 0; j < nh; j++)
    {
        double sum = 0.0;
        for (int k = 0; k < 10; k++)
        {
            double labelk = (k == true_label) ? 1.0 : 0.0;
            sum += (z[k] - labelk) * z[k] * (1 - z[k]) * w2[j][k] *
                hidden_out[j] * (1 - hidden_out[j]) * images[im][i];
        }
        grad_w1[i][j] = sum;
    }
}

for (int j = 0; j < nh; j++)
{
    for (int k = 0; k < 10; k++)
    {
        double labelk = (k == true_label) ? 1.0 : 0.0;
        grad_w2[j][k] = (z[k] - labelk) * z[k] * (1 - z[k]) * hidden_out[j
            ];
    }
}

for (int j = 0; j < nh; j++)
{
    double sum = 0.0;
    for (int k = 0; k < 10; k++)
    {
        double labelk = (k == true_label) ? 1.0 : 0.0;
        sum += (z[k] - labelk) * z[k] * (1 - z[k]) * w2[j][k] * hidden_out[
            j] * (1 - hidden_out[j]);
    }
    grad_bias1[j] = sum;
}

for (int k = 0; k < 10; k++)
{
    double labelk = (k == true_label) ? 1.0 : 0.0;
    grad_bias2[k] = (z[k] - labelk) * z[k] * (1 - z[k]);
}
```

```
        // ——————————————————————————————更新权与偏置
            ————————————————————————————
        mtx.lock();
        for (int i = 0; i < 784; i++)
        {
            for (int j = 0; j < nh; j++)
            {
                w1[i][j] -= learning_rate * grad_w1[i][j];
            }
        }

        for (int i = 0; i < nh; i++)
        {
            for (int j = 0; j < 10; j++)
            {
                w2[i][j] -= learning_rate * grad_w2[i][j];
            }
        }

        for (int i = 0; i < nh; i++)
        {
            bias1[i] -= learning_rate * grad_bias1[i];
        }

        for (int i = 0; i < 10; i++)
        {
            bias2[i] -= learning_rate * grad_bias2[i];
        }
        mtx.unlock();
    }
}

void train_and_test()
{
    for (e = 1; e <= epoch; e++)
    {
        update_learning_rate(e);
        // 平均分割训练集为16份
        int batch_size = train_images.size() / 16;

        vector<thread> threads;
        for (int i = 0; i < 15; i++)
        {
            threads.emplace_back(train, ref(train_images), ref(train_labels), i *
                batch_size, (i + 1) * batch_size - 1);
        }
        threads.emplace_back(train, ref(train_images), ref(train_labels), 15 *
            batch_size, train_images.size() - 1);
```

```
        for (auto &thread : threads)
        {
            thread.join();
        }

        test();
    }
}

void test()
{
    vector<int> true_positive(10, 0), false_positive(10, 0), false_negative(10, 0);
    for (int i = 0; i < test_images.size(); i++)
    {
        int true_label = (int)test_labels[i];
        vector<double> hidden_out = get_hidden_out(test_images[i]);
        vector<double> z = get_z(hidden_out);
        int recognize = -1;
        double max = 0;
        for (int i = 0; i < 10; i++)
        {
            if (z[i] > max)
            {
                max = z[i];
                recognize = i;
            }
        }

        if (recognize == true_label)
            true_positive[recognize]++;
        else
        {
            false_positive[recognize]++;
            false_negative[true_label]++;
        }
    }

    double total_precision = 0, total_recall = 0, total_f1 = 0;
    for (int i = 0; i < 10; i++)
    {
        double precision = true_positive[i] * 1.0 / (true_positive[i] +
            false_positive[i]);
        double recall = true_positive[i] * 1.0 / (true_positive[i] + false_negative
            [i]);
        double f1 = 2 * precision * recall / (precision + recall);
        total_precision += precision;
        total_recall += recall;
        total_f1 += f1;
    }
```

```
    cout << "epoch = " << e << ", average precision = " << total_precision / 10.0
         << ", average recall = " << total_recall / 10.0
         << ", average F1 score = " << total_f1 / 10.0
         << ", time:" << chrono::duration_cast<chrono::seconds>(chrono::
             steady_clock::now() - start_time).count() << endl;

}

int main()
{
    start_time = chrono::steady_clock::now();
    read_train_images();        // 载入训练集图像
    read_train_labels();        // 载入训练集标签
    read_test_images();         // 载入测试集图像
    read_test_labels();         // 载入测试集标签
    init_parameters();          // 为权重矩阵和偏置向量随机赋初值
    train_and_test();
    return 0;
}
```