Лабораторная работа 1
Реализация метода обратного распространения ошибки для двухслойной полностью связанной нейронной сети

Цель данной работы заключается в освоении принципов метода обратного распространения ошибки и его практической реализации для обучения двухслойной полностью связанной нейронной сети. В данной лабораторной работе реализация выполняется "с нуля" на языке MATLAB, что позволяет глубже понять механизм обучения нейронных сетей.

В ходе работы изучается общая схема метода обратного распространения ошибки, включая прямой проход, вычисление функции потерь и обратный проход для корректировки весов. Особое внимание уделяется математическому обоснованию алгоритма: выводятся формулы для вычисления градиентов функции ошибки по весам и смещениям, а также правила их обновления с использованием градиентного спуска.

На основе полученных теоретических знаний разрабатывается программная реализация двухслойной нейронной сети в среде MATLAB. Проектирование включает инициализацию параметров, прямой и обратный ход, а также процедуру обновления весов. Для проверки корректности реализации проводится тестирование на стандартном наборе MNIST. В процессе тестирования оцениваются точность модели и скорость обучения.

Описание метода обратного распространения ошибки

В задаче реализуется классификация изображений из датасета MNIST.

x —входной вектор из 784 значений, является преобразованным изображением из датасета 28 на 28 пикселей;

y — вектор из истинных значений;

$\hat{y}$ — вектор предсказанных значений;

w_1 – матрица весов между входным и скрытым слоями;

b_1 – смещение для весов скрытого слоя;

w_2 – матрица весов между скрытым и выходным слоями;

b_2 – смещение для весов скрытого слоя.

Использовалась функция активации Softmax

$$Softmax(x) = \frac{e^{y_i}}{\sum_j^N e^{y_i}}$$

и функция потерь категориальная кросс-энтропия

$$Categorical\_cross\_entropy = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(\hat{y}_i)$$

Алгоритм работы сети обратного распространения ошибки:

1. Прямое распространение.

$$z_1 = xw_1 + b_1$$
$$a_1 = \sigma(z_1)$$
$$z_2 = a_1 w_2 + b_2$$
$$\hat{y} = softmax(z_2)$$

2. Вычисление потерь.
3. Обратное распространение ошибки

$\delta_2 = \hat{y} - y$ – ошибка на выходе

$$\frac{\partial L}{\partial w_2} = a_1 \delta_2$$
$$\frac{\partial L}{\partial b_2} = \sum \delta_2$$

– градиенты для весов и смещений выходного слоя

$\delta_1 = (\delta_2 w_2) * \sigma'(z_1)$ – ошибка на скрытом слое

$$\frac{\partial L}{\partial w_1} = x^T \delta_1$$
$$\frac{\partial L}{\partial b_1} = \sum \delta_1$$

– градиенты для весов и смещений скрытого слоя

4. Обновление весов.

В силу специфики MATLAB выходной вектор содержит 11 нейронов. Предлагаемая программная реализация включает пример, для сети с одним скрытым слоем (в соответствии с заданием). Однако возможно и расширение на большее количество скрытых слоев.

Описание используемых метрик

Для оценки качества решения задачи бинарной классификации используется метрика точности, которая показывает, какую долю примеров модель правильно отнесла к своему классу. При этом классификация каждого примера может быть одной из четырёх ситуаций: правильное положительное предсказание (TP), когда модель верно определила положительный класс; правильное отрицательное предсказание (TN), когда модель верно распознала отрицательный класс; ложноположительное срабатывание (FP), когда пример ошибочно отнесён к положительному классу; и ложноотрицательное (FN) когда модель пропустила положительный пример и отнесла его к отрицательному классу.

Точность вычисляется как отношение суммы правильно классифицированных примеров (как положительных, так и отрицательных) к общему числу всех примеров:

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N} \mathrm{I}[y_i = y_{pred}].$$

Для оптимизации модели используется функция потерь — бинарная кросс-энтропия. Она измеряет расхождение между истинными метками и предсказанными вероятностями принадлежности к положительному классу, помогая модели постепенно улучшать свои прогнозы. В процессе обучения и проверки на валидационных данных вычисляются значения как функции потерь, так и точности, что позволяет отслеживать качество и эффективность обучения.

$$Loss = \ -\frac{1}{N}\sum_{i=1}^{N}\left[y_{true_i}\, log\left(y_{pred_i}\right) + \left(1 - y_{true_i}\right) log\left(1 - y_{pred_i}\right)\right].$$

Для случая небинарной классификации (тестирование на наборе MNIST) использовалась аналогичная метрика, категориальная кросс-энтропия.

$$Categorical\_cross\_entropy = -\frac{1}{N}\sum_{i=1}^{N} y_{true\ i} \cdot log(y_{pred_i})$$

Реализация программы

Для начала загрузим необходимые для выполнения работы данные из набора MNIST. Важно отметить, что для работы необходим пакет *Deep Learning Toolbox.*

```
function [X_train, y_train, X_test, y_test] = load_mnist()
```

```
    digitDatasetPath = fullfile(matlabroot, 'toolbox', 'nnet',
'nndemos', ...
        'nndatasets', 'DigitDataset');
    imds = imageDatastore(digitDatasetPath, ...
        'IncludeSubfolders', true, 'LabelSource', 'foldernames');

    % Разделение на train/test
    [imdsTrain, imdsTest] = splitEachLabel(imds, 0.8, 'randomized');

    % Вытягивание в матрицы
    X_train = zeros(numel(imdsTrain.Files), 28*28);
    for i = 1:numel(imdsTrain.Files)
        img = readimage(imdsTrain, i);
        X_train(i,:) = double(img(:))'/255;
    end
    y_train = double(imdsTrain.Labels);

    X_test = zeros(numel(imdsTest.Files), 28*28);
    for i = 1:numel(imdsTest.Files)
        img = readimage(imdsTest, i);
        X_test(i,:) = double(img(:))'/255;
    end
    y_test = double(imdsTest.Labels);

    % One-hot кодирование
    y_train = ind2vec(y_train' + 1)';
    y_test = ind2vec(y_test' + 1)';
end
```

Реализуем функцию обучения модели

```
function [res, train_hist] = train(Model, dataset, y_true, loss_func,
batch_size, lr, num_epochs)
    num_samples = size(dataset, 1); % Количество примеров
    num_batches = ceil(num_samples / batch_size); % Количество батчей
    train_hist = [];
    h = waitbar(0, 'Processing...');

    for epoch = 1:num_epochs
        % Перемешивание данных на каждой эпохе
        shuffled_indices = randperm(num_samples);
        shuffled_data = dataset(shuffled_indices,:);
        shuffled_labels = y_true(shuffled_indices,:);

        total_epoch_loss = 0;
        for batch=1:num_batches
            % Выделение текущего батча
            start_idx = (batch - 1) * batch_size + 1;
            end_idx = min(batch * batch_size, num_samples);
            X_batch = shuffled_data(start_idx:end_idx, :);
            y_batch = shuffled_labels(start_idx:end_idx, :);

            [y_pred, hist] = forward(Model, X_batch); % Прямой проход
            Loss = loss_func(y_pred, y_batch); % Вычисление ошибки на
батче
            total_epoch_loss = total_epoch_loss + Loss*length(X_batch);
```

```
            Model = backward(Model, X_batch, y_batch, hist); % Обратный
проход

            % Коррекция весов
            for i = 1:Model.hiden+1
                Model.("W"+int2str(i)) = Model.("W"+int2str(i)) - lr *
Model.("dW"+int2str(i));
                Model.("b"+int2str(i)) = Model.("b"+int2str(i)) - lr *
Model.("db"+int2str(i));
            end
        end
        total_epoch_loss = total_epoch_loss/num_samples;
        train_hist = [train_hist, total_epoch_loss];
        waitbar(epoch/num_epochs,   h,   sprintf('Epoch   (%d%%)   Loss:
%f',ceil(epoch/num_epochs*100), total_epoch_loss));

    end
    res = Model;
end
```

Внутри функции обучения используются следующие вспомогательные функции прямого и обратного прохода

```
function [res, his] = forward(Model, input)
    X = input;

    for i=1:Model.hiden+1
        hist.("W"+int2str(i))                                          =
X*Model.("W"+int2str(i))+Model.("b"+int2str(i));

        X                                                              =
Model.("f"+int2str(i))(X*Model.("W"+int2str(i))+Model.("b"+int2str(i))
);

        hist.("F"+int2str(i)) = X;
    end
    res = X;
    his = hist;
end

function Model = backward(Model, X_batch, y_batch, hist)
    m = size(X_batch, 1); % Количество примеров в батче
    layers = Model.hiden + 1; % Общее количество слоёв

    % Инициализация градиента выходного слоя (softmax + cross-entropy)
    dZ = hist.("F"+int2str(Model.hiden + 1)) - y_batch;

    % Обратное распространение по слоям
    for i = (Model.hiden + 1):-1:1
        % Получаем активации предыдущего слоя
        if i == 1
            A_prev = X_batch;
        else
            A_prev = hist.("F"+int2str(i-1));
        end

        % Сохраняем градиенты в структуру Model
```

```
        Model.("dW"+int2str(i)) = (A_prev' * dZ) / m;
        Model.("db"+int2str(i)) = sum(dZ, 1) / m;

        % Если не входной слой, вычисляем градиент для предыдущего
        if i > 1
            W = Model.("W"+int2str(i));
            Z = hist.("W"+int2str(i-1));
            dA = dZ * W';
            dZ = dA .* Model.("df"+int2str(i-1))(Z);
        end
    end
end
```

Также для анализа результатов обучения понадобятся еще две функции, это функция для вывода матрицы ошибок. Также для определения функции ошибки и использования ее при инициализации модели используется соответствующая функция

```
function res = LosFunction(y_pred, y_true)
    eps = 1e-15;
    res = mean(-sum(y_true.*log(max(eps, y_pred)), 2));
end

function stats = confusionmatStats(C)
    stats.accuracy = sum(diag(C)) / sum(C(:));
    stats.precision = diag(C) ./ sum(C, 2);
    stats.recall = diag(C) ./ sum(C, 1)';
    stats.F1 = 2 * (stats.precision .* stats.recall) ./ (stats.precision
+ stats.recall);
end
```

Далее приведен пример загрузки набора данных и предобработки. При этом загрузку набора данных необходимо выполнить только один раз в случае локальной среды.

```
% требуется Deep Learning Toolbox
function [X_train, y_train, X_test, y_test] = load_mnist()

    digitDatasetPath   =   fullfile(matlabroot,   'toolbox',   'nnet',
'nndemos', ...
        'nndatasets', 'DigitDataset');
    imds = imageDatastore(digitDatasetPath, ...
        'IncludeSubfolders', true, 'LabelSource', 'foldernames');

    % Разделение на train/test
    [imdsTrain, imdsTest] = splitEachLabel(imds, 0.8, 'randomized');

    % Вытягивание в матрицы
    X_train = zeros(numel(imdsTrain.Files), 28*28);
    for i = 1:numel(imdsTrain.Files)
        img = readimage(imdsTrain, i);
        X_train(i,:) = double(img(:))'/255;
    end
    y_train = double(imdsTrain.Labels);
```

```
    X_test = zeros(numel(imdsTest.Files), 28*28);
    for i = 1:numel(imdsTest.Files)
        img = readimage(imdsTest, i);
        X_test(i,:) = double(img(:))'/255;
    end
    y_test = double(imdsTest.Labels);

    % One-hot кодирование
    y_train = ind2vec(y_train' + 1)';
    y_test = ind2vec(y_test' + 1)';
end

% 1. Загрузка данных
[X_train, y_train, X_test, y_test] = load_mnist();
```

Для теста работоспособности использовалась следующая модель

```
input_size = 784;  % 28x28
hidden_size = 128; % Выбрано случайное значение
output_size = 11;  % Цифры 0-9

% Количество скрытых слоев
Model.hiden = 1;

% Инициализация первого слоя (ReLU)
Model.W1 = randn(input_size, hidden_size) * sqrt(2/input_size);
Model.b1 = zeros(1, hidden_size);
Model.f1 = @(x) max(0, x);
Model.df1 = @(x) x>0;

% Инициализация второго слоя
Model.W2 = randn(hidden_size, output_size) * 0.01;  % Малое случайное значение
Model.b2 = zeros(1, output_size);
Model.f2 = @(x) exp(x - max(x, [], 2)) ./ sum(exp(x - max(x, [], 2)), 2);
```

Модель обучалась со следующими параметрами.

```
% 2. Обучение
batch_size = 10;
learning_rate = 0.001;
epochs = 2000;
[Model, hist] = train(Model, X_train, y_train, @(x, y) LosFunction(x, y) ,batch_size, learning_rate, epochs);
plot(hist)
```

Ниже приведены кривые обучения на данных с набором рукописных цифр MNIST и матрица ошибок. Итоговая точность на тестовом наборе составила 99%. В данном случае использовался не предлагаемый в TensorFlow тестовый набор данных, а проводилось разделение обучающего набора на обучение/тест, в связи с чем модель и показала высокую точность на тестовом наборе. Несмотря на это реализованная модель успешно обучается, что подтверждается метриками. Итоговый код в формате “.mlx” загружен на репозиторий https://github.com/U-yun/LAB_KICHEEV. Для запуска рекомендуется MATLAB как минимум версии R2022b.

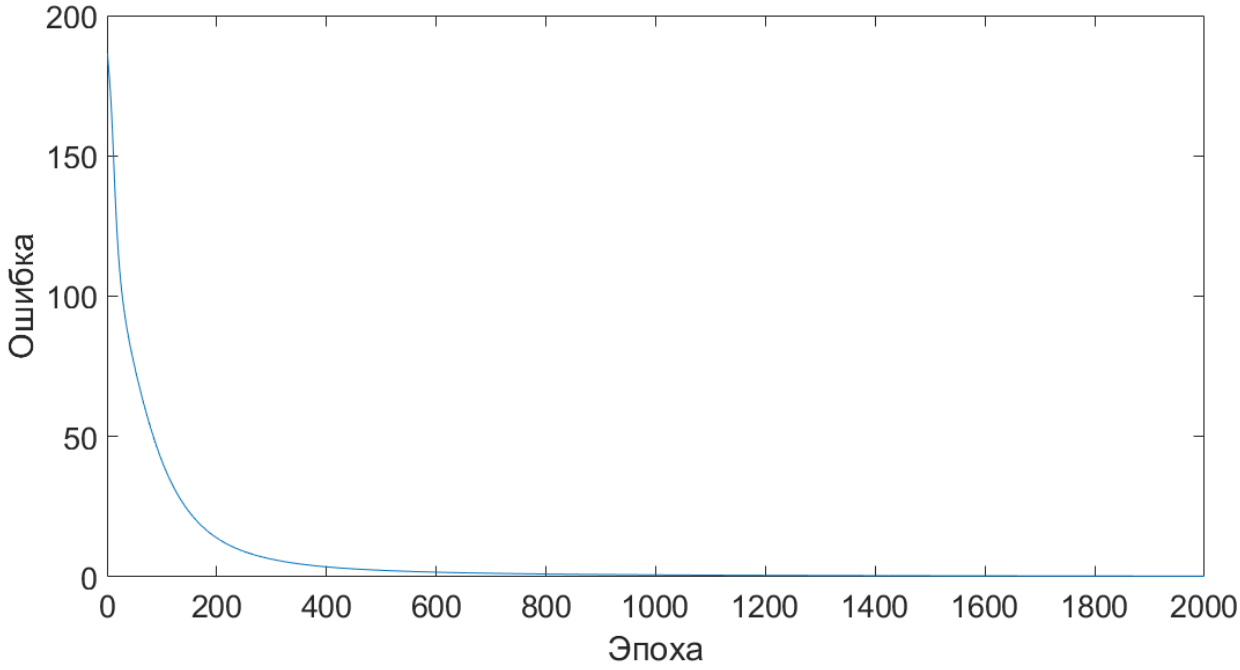

Рисунок 1 – Кривая ошибки при обучении модели на наборе рукописных цифр MNIST

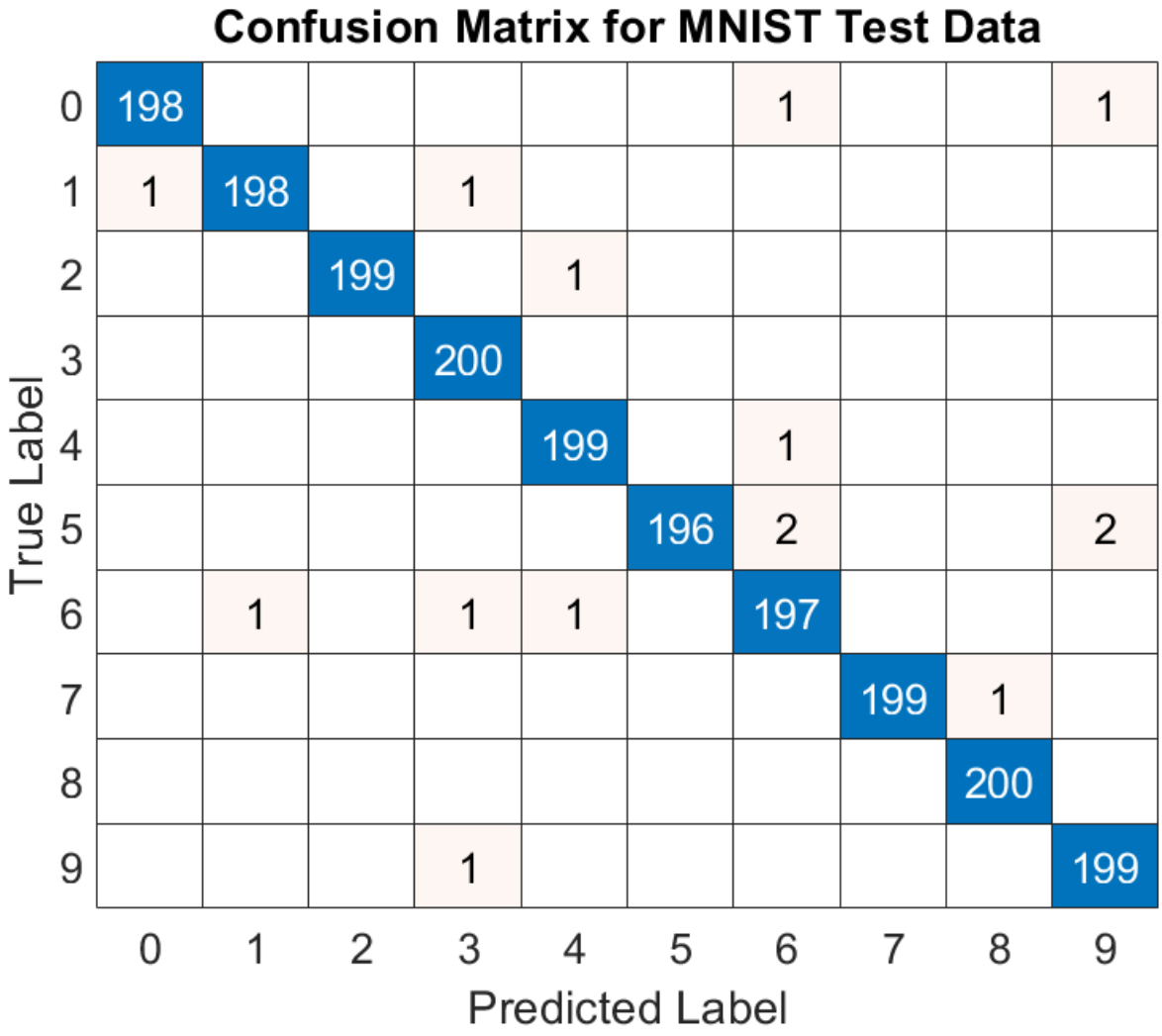

Рисунок 2 – Результат работы модели на тестовых данных