Персептрон

Замирайлов Алексей
s21: maykitbo
Telegram: @zamyrailov

5.11.2023

Перспетрон

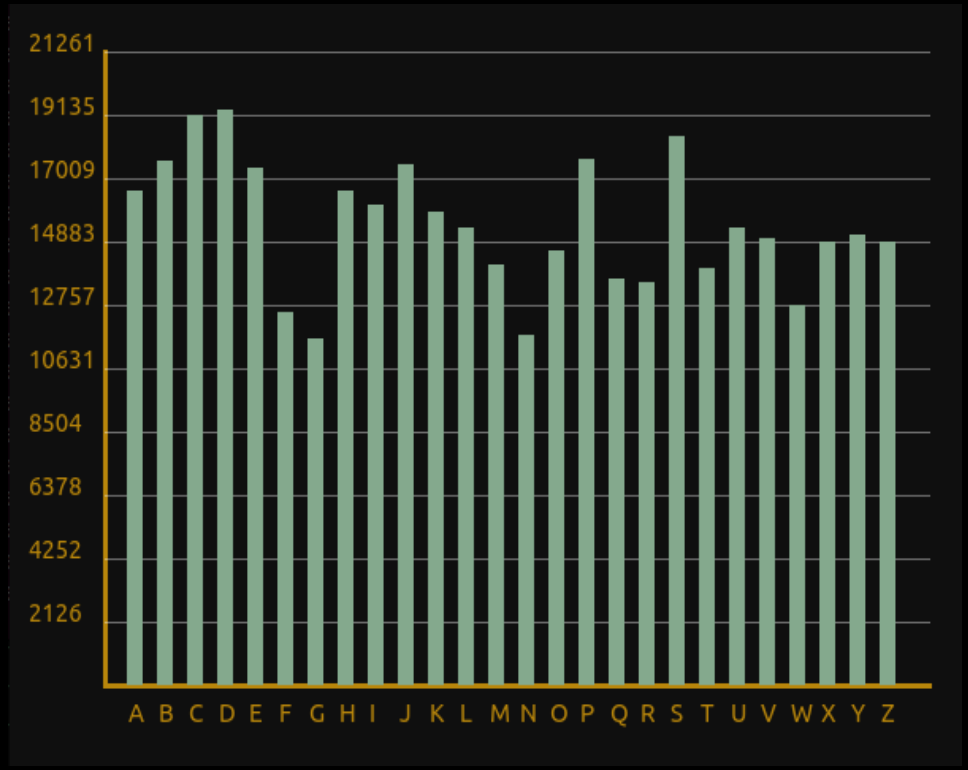
Работа с данными

- Сбор
- Подготовка
- Перемешивание
- Передача

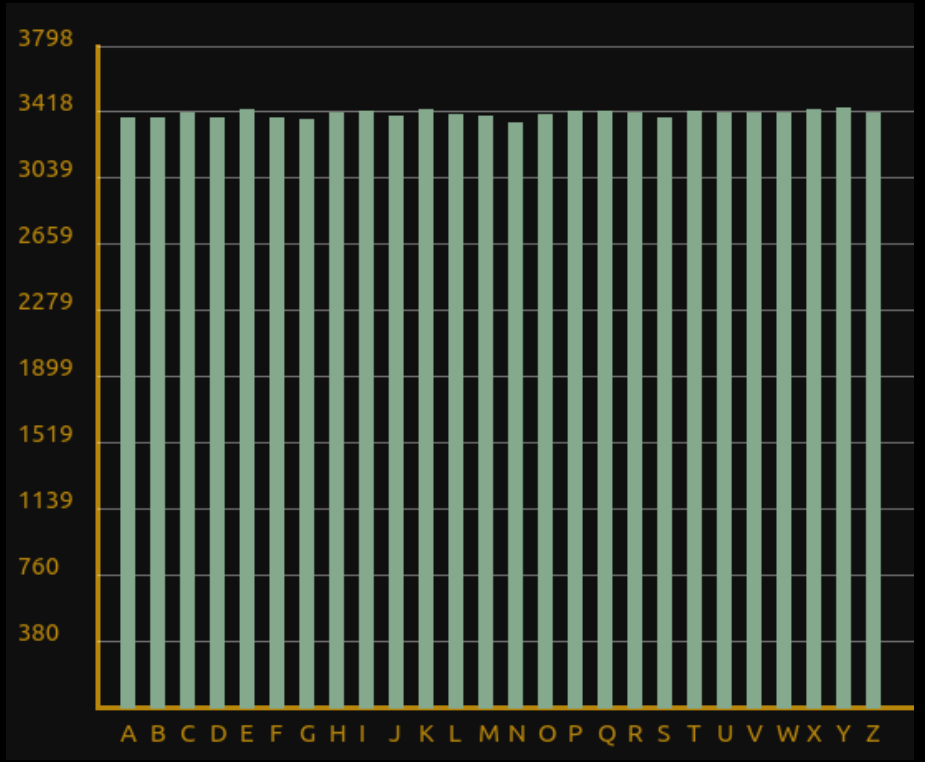
Нейросеть

- Матрица весов
- Forward
- Backward
- Ускорение
- Примеры

Сбор данных

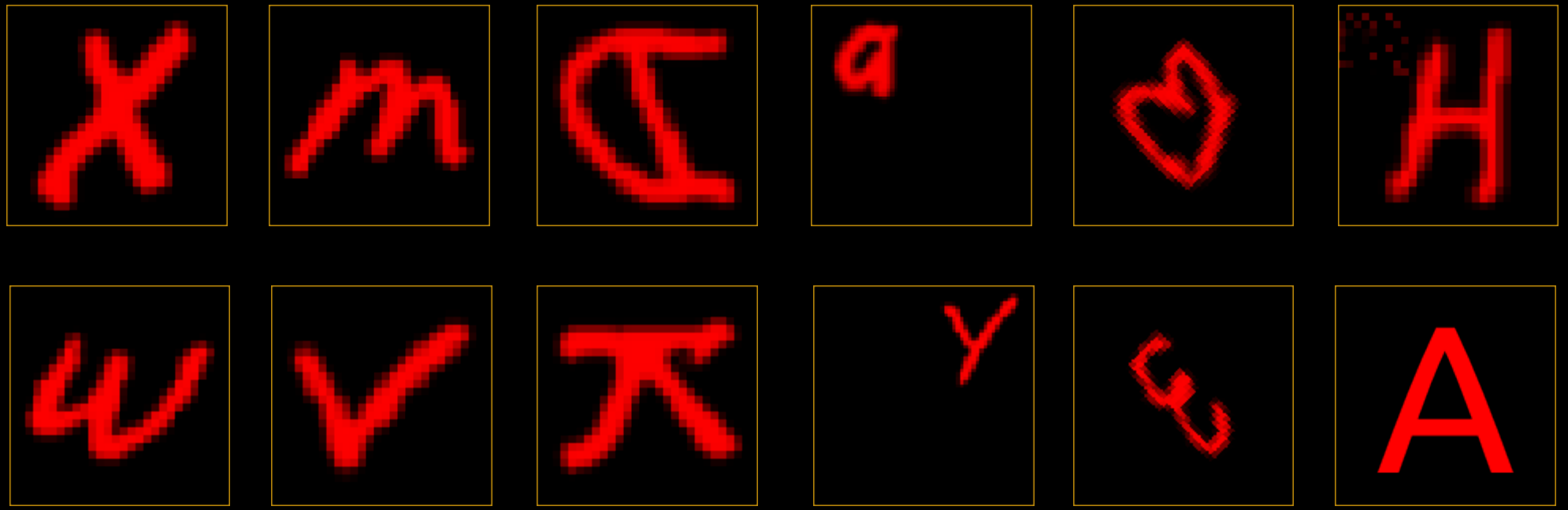


Плохое распределение



Хорошее распределение

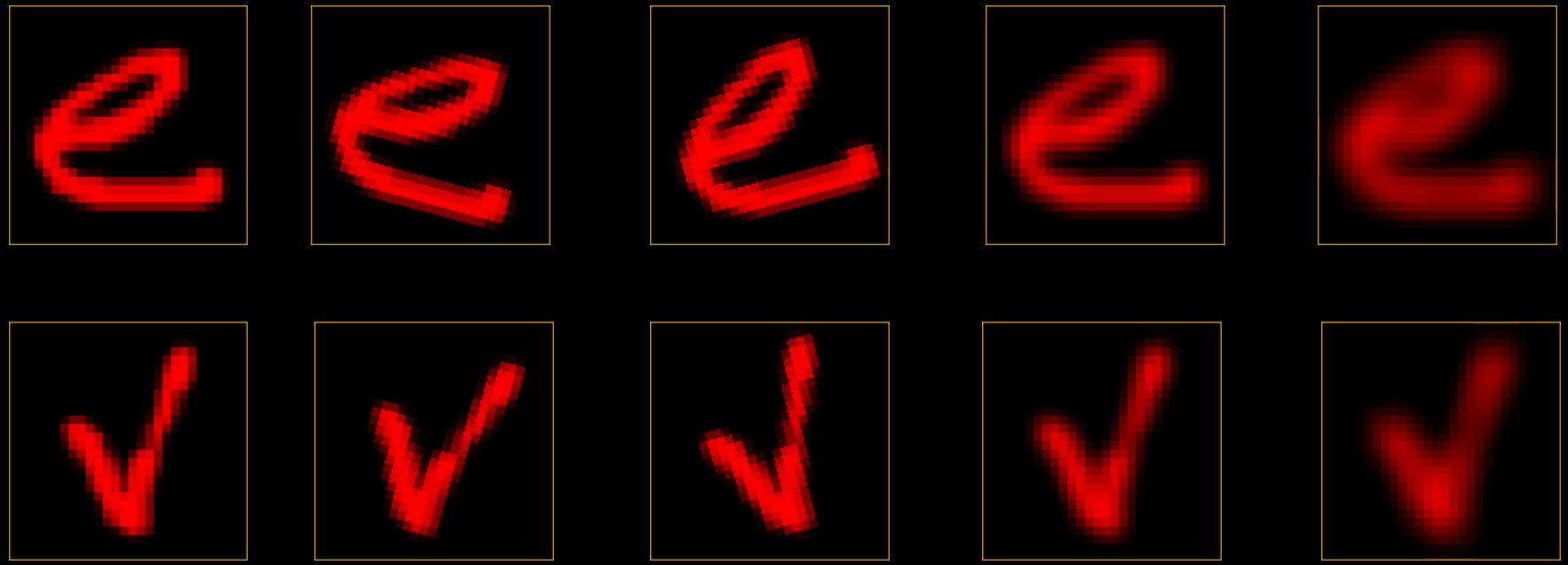
Сбор и обработка данных



1 2 3 4 5

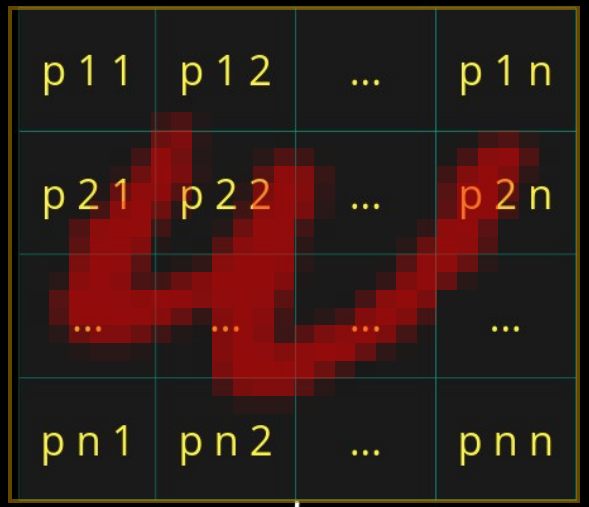
4

Обработка данных



original | Rotate 15° | Rotate -15° | Blure R1 | Blure R2

Передача даных в нейрость

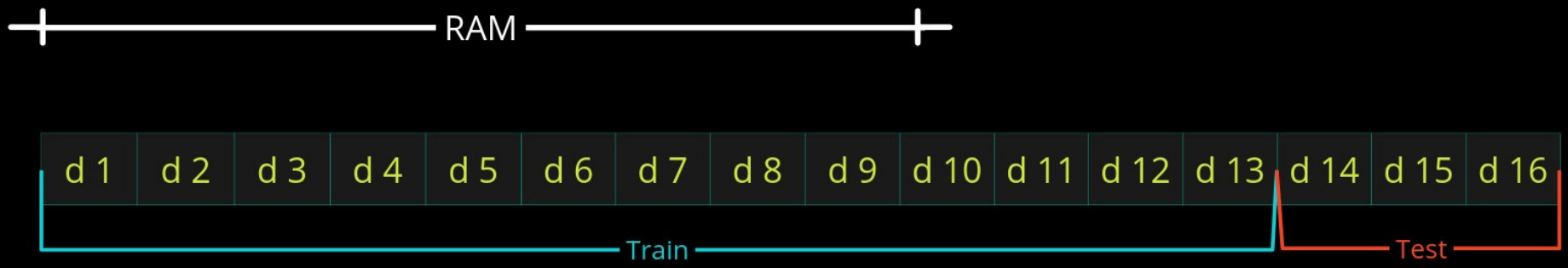


$$p_{ij} \in [0, 255] \Rightarrow p_{ij} = \frac{p_{ij}}{127.5} - 1$$

$$p_{ij} \in [-1, 1]$$

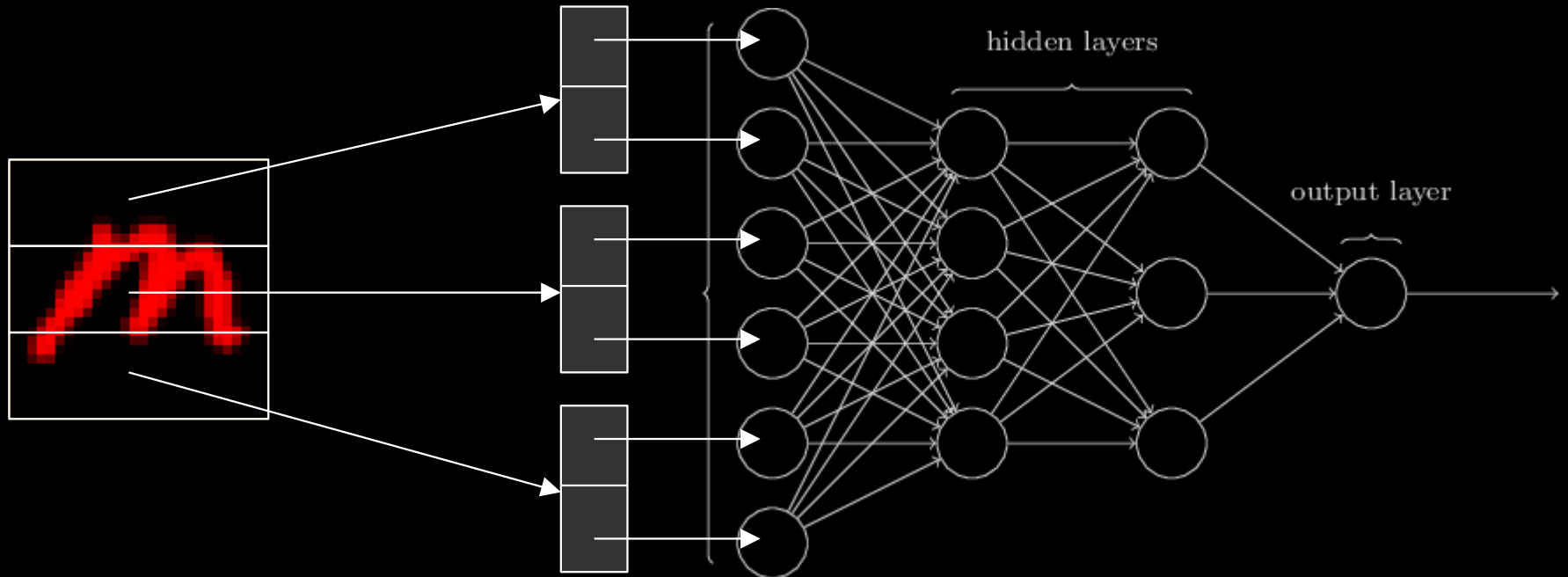
Перемешиваем тренировочную выборку перед каждой эпохой:
random_shuffle(data.begin(), data.begin() + train_size)

Проблемы RAM



1. Разбиение данных
2. База данных
3. Генератор
4. Семплирование

Персептрон



Персептрон. Структура проекта

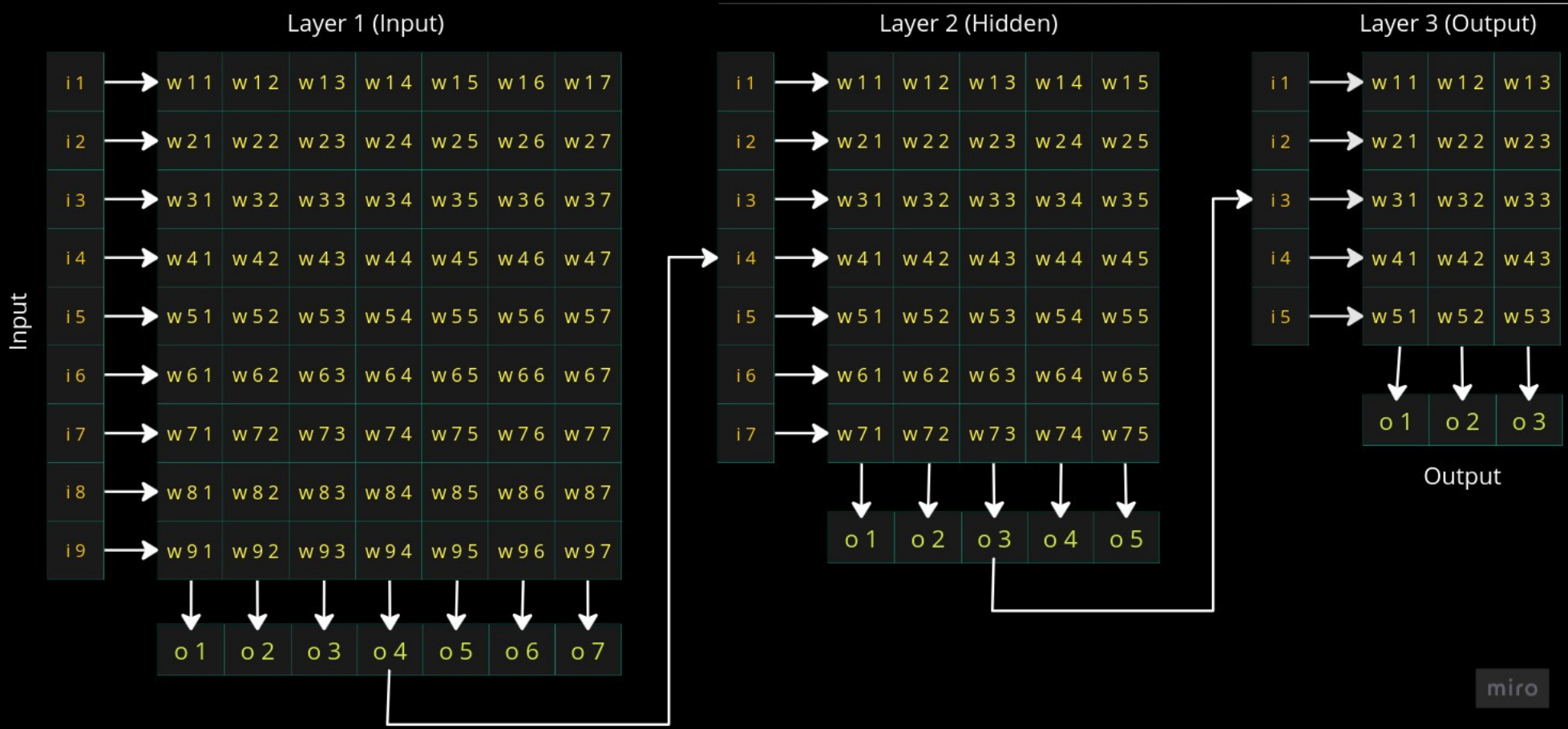
```cpp
struct Layer
{
    vector<float> output;
    vector<float> biases;
    matrix<float> weights;

    Layer(int cols, int rows)
        : output(cols)
        , biases(cols)
        , weights(cols, rows)
    {}
};
```

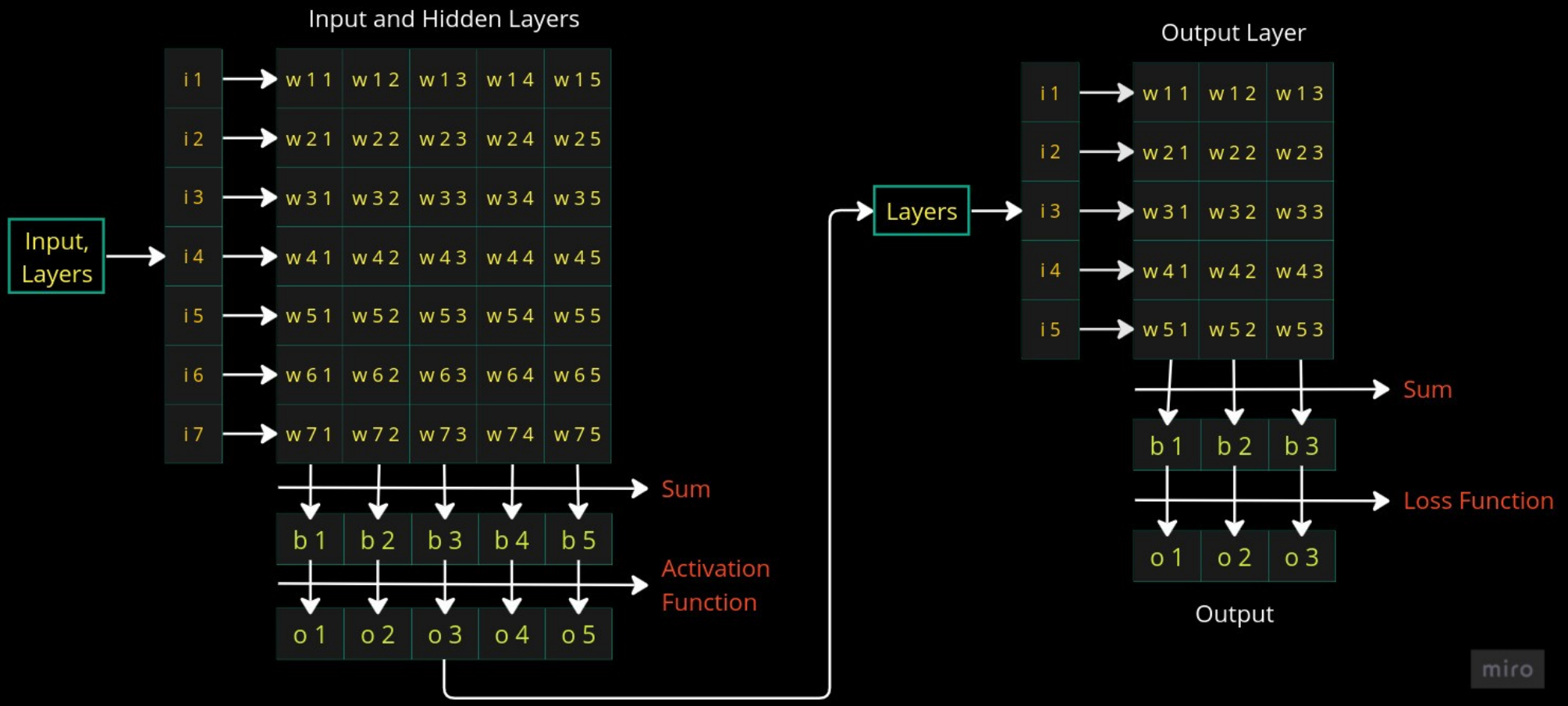
```cpp
struct Settings
{
    float learning_rate;
    float Activation(float x);
    float DerivativeActivation(float x);
    /* ... */
};

struct NeronNetwork
{
    vector<Layer> layers;
    Settings settings;
};
```

Матрица весов. Forward



Forward


Input and Hidden Layers
Input, Layers
i 1
i 2
i 3
i 4
i 5
i 6
i 7
w 1 1 w 1 2 w 1 3 w 1 4 w 1 5
w 2 1 w 2 2 w 2 3 w 2 4 w 2 5
w 3 1 w 3 2 w 3 3 w 3 4 w 3 5
w 4 1 w 4 2 w 4 3 w 4 4 w 4 5
w 5 1 w 5 2 w 5 3 w 5 4 w 5 5
w 6 1 w 6 2 w 6 3 w 6 4 w 6 5
w 7 1 w 7 2 w 7 3 w 7 4 w 7 5
Sum
b 1 b 2 b 3 b 4 b 5
Activation Function
o 1 o 2 o 3 o 4 o 5
Layers
Output Layer
i 1
i 2
i 3
i 4
i 5
w 1 1 w 1 2 w 1 3
w 2 1 w 2 2 w 2 3
w 3 1 w 3 2 w 3 3
w 4 1 w 4 2 w 4 3
w 5 1 w 5 2 w 5 3
Sum
b 1 b 2 b 3
Loss Function
o 1 o 2 o 3
Output
miro

Forward

```
void NeronNetwork::Forward(vector &input)
{
    // для каждого слоя кроме последнего
    for (int i = 0; i < layers.size() - 1; ++i)
    {
        // input первого слоя - входные данные
        layers[i].HiddenSignal(input);
        // output слоя - input следующего слоя
        input = layers[i].output;
    }
    // последний слой
    layers.back().OutputSignal(input);
}
```

```
void Layer::HiddenSignal(vector &input)
{
    // vector = vector * matrix
    output = input * weights;
    // vector = f(vector + vector)
    output = activation(output + biases);
}

void Layer::OutputSignal(vector &input)
{
    // vector = vector * matrix
    output = input * weights;
    // vector = f(vector + vector)
    output = lossfunc(output + biases);
}
```

Backward

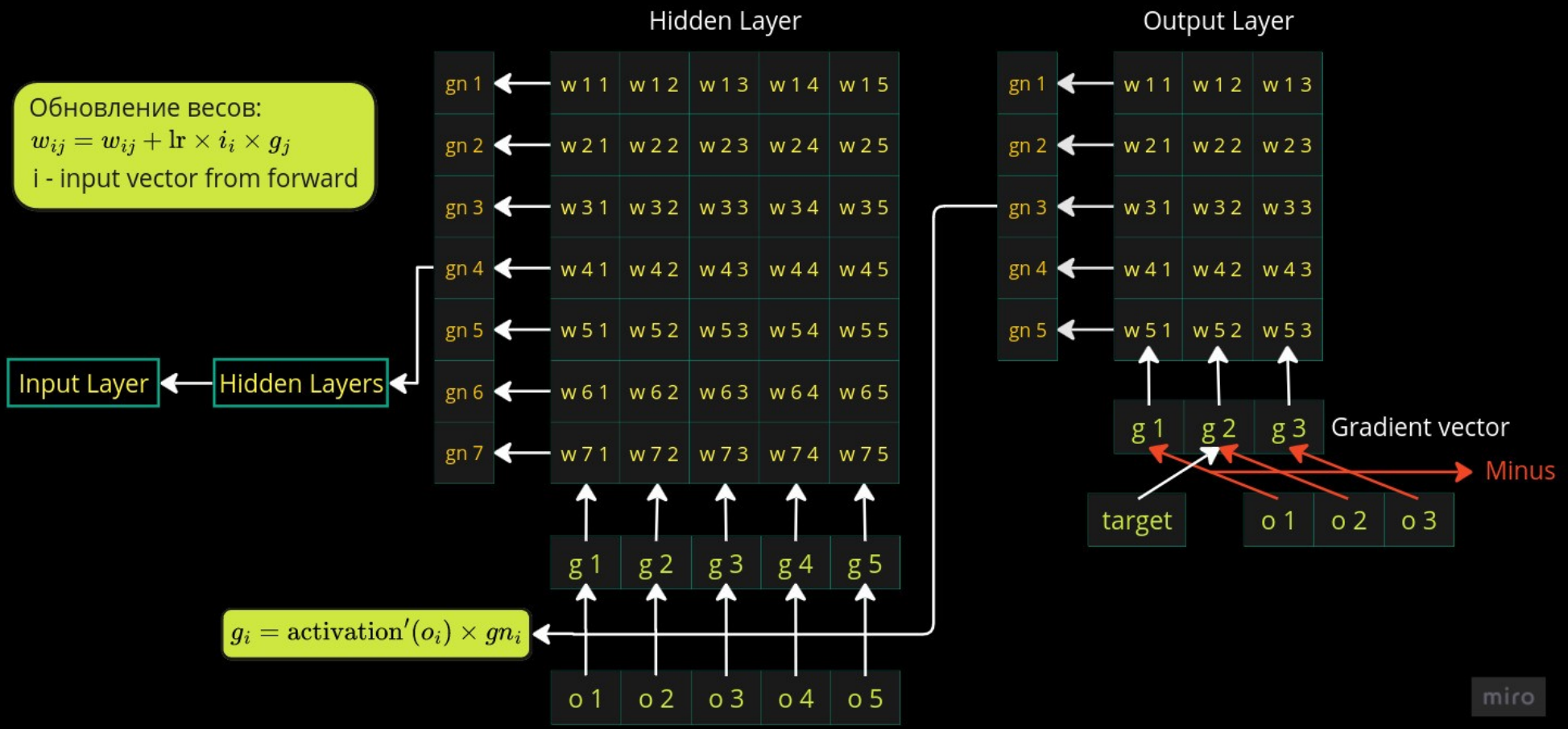
```
struct Layer
{
    vector<float> output;
    vector<float> biases;
    matrix<float> weights;

    matrix<float> gradients_matrix;
    vector<float> gradients_sum;
    vector<float> gradients;
};
```

```
Layer::Layer(int cols, int rows)
    : output(cols)
    , biases(cols)
    , weights(cols, rows)

    , gradients_matrix(cols, rows)
    , gradients_sum(cols)
    , gradients(cols)
{}
```

Backward


Hidden Layer
Output Layer
Обновление весов:
$w_{ij} = w_{ij} + \mathrm{lr} \times i_i \times g_j$
i - input vector from forward
Input Layer
Hidden Layers
gn 1
gn 2
gn 3
gn 4
gn 5
gn 6
gn 7
w 1 1 w 1 2 w 1 3 w 1 4 w 1 5
w 2 1 w 2 2 w 2 3 w 2 4 w 2 5
w 3 1 w 3 2 w 3 3 w 3 4 w 3 5
w 4 1 w 4 2 w 4 3 w 4 4 w 4 5
w 5 1 w 5 2 w 5 3 w 5 4 w 5 5
w 6 1 w 6 2 w 6 3 w 6 4 w 6 5
w 7 1 w 7 2 w 7 3 w 7 4 w 7 5
g 1 g 2 g 3 g 4 g 5
$g_i = \mathrm{activation}'(o_i) \times gn_i$
o 1 o 2 o 3 o 4 o 5
w 1 1 w 1 2 w 1 3
w 2 1 w 2 2 w 2 3
w 3 1 w 3 2 w 3 3
w 4 1 w 4 2 w 4 3
w 5 1 w 5 2 w 5 3
g 1 g 2 g 3
Gradient vector
Minus
target
o 1 o 2 o 3
miro

Backward

```
void NeronNetwork::Backward(vector &input, int target)
{
    int k = layers.size() - 1;
    // последний слой
    layers[k].ProcessTarget(target);
    layers[k].PassGradient(layers[k - 1]);
    layers[k].CollectGradients();

    // для каждого скрытого слоя
    for (--k; k >= 1; --k)
    {
        layers[k].ProcessGradient();
        layers[k].PassGradient(layers[k - 1]);
        layers[k].CollectGradients();
    }

    // первый слой
    layers[0].ProcessGradient();
    layers[0].CollectGradients();
}
```

```
void Layer::CollectGradients(vector &a, vector &b)
{
    // matrix = matrix + vector^T * vector
    gradients_matrix += gradients * prev_layer.output;
    gradients_sum += gradients;
}

void Layer::ProcessTarget(int target)
{
    gradients = -output;
    gradients[target] += 1.0;
}

void Layer::ProcessGradient()
{
    gradients *= derivative_activation(output);
}

void Layer::PassGradient(Layer &prev_layer)
{
    // vector = vector * matrix^T
    prev_layer.gradients = MulABT(gradients, weights);
}
```

Обновление весов

```cpp
void NeronNetwork::UpdateWeights()
{
    for (auto &layer : layers)
    {
        layer.UpdateWeights();
    }
}

void Layer::UpdateWeights()
{
    biases += learning_rate * gradients_sum;
    weights += learning_rate * gradients_matrix;

    fill(gradients_sum.begin(), gradients_sum.end(), 0.0);
    gradients_matrix.fill(0.0);
}
```

Настройки

```
struct Settings
{
    vector<size_t> layers;          ///<  Размеры слоёв
    float learning_rate;            ///<  Скорость обучения (0.001, 0.002)
    float momentum = 0.0;           ///<  Коэффициент инерции (0.0, 0.9)
    float lr_epoch_k = 1.0;         ///<  Изменение lr с каждой эпохой
    float lr_layers_k = 1.0;        ///<  Изменение lr по слоям
    float weight_mean = 0.0;        ///<  Среднее значение весов
    float weight_sd = 0.01;         ///<  Среднее отклонение весов
    /* ... */

    function<float(float)> activation;                    ///<  Функция активации
    function<float(float)> derivative_activation;         ///<  Производная активации
    function<float(int, int)> weight_init;                ///<  Функция инициализации весов
    function<vector<float>(vector<float>)> loss_function; ///<  Функция потерь
};
```

Ускорение

1. Флаги оптимизации "-O3 -march=native"
2. Библиотеки OpenBLAS, MKL…
3. cache friendly Пример:

```
const unsigned N = 784;
const unsigned M = 1024;
float momentum = 0.2;

Matrix<double> gradients_matrix(N, M);
Matrix<double> delta_weights_1(N, M);
Matrix<double> delta_weights_2(N, M);
Matrix<double> weights_1(N, M);
Matrix<double> weights_2(N, M);
Matrix<double> weights_3(N, M);
```

```
for (unsigned k = 0; k < N; ++k)
{
    for (unsigned g = 0; g < M; ++g)
    {
        delta_weights_2(k, g) = delta_weights_2(k, g) * momentum
            + gradients_matrix(k, g) * (1.0 - momentum);
    }
}
for (unsigned k = 0; k < N; ++k)
{
    for (unsigned g = 0; g < M; ++g)
    {
        weights_2(k, g) += delta_weights_2(k, g);
    }
}
```

```
for (unsigned k = 0; k < N; ++k)
{
    for (unsigned g = 0; g < M; ++g)
    {
        delta_weights_1(k, g) = delta_weights_1(k, g) * momentum
            + gradients_matrix(k, g) * (1.0 - momentum);
        weights_1(k, g) += delta_weights_1(k, g);
    }
}
```

Примеры

Layers [784, 200, 150, 100, 26]
learning rate 0.0005
6 эпох с размером батча 64
Активационная функция ReLU
Инициализация весов HE
Инерция 0.2
Данные 266400 букв, 1.68 Гбайт, поворот +-9°

Accuracy: 0.89238
Precision: 0.895638 +- 0.0705589
Recall: 0.892772 +- 0.0727184
F1: 0.892666 +- 0.0647143
Время одной эпохи 27787 ms

Параметры

Активационные функции:

1. ReLU
2. tanh
3. SiLU
4. Sigmoid

Инициализация весов:

1. Xavier
2. HE

Данные:

1. 88800 букв 560 Мбайт
2. 266400 букв, 1.68 Гбайт, поворот

Инерция = m = 0.2 – m 0.8

Скорость обучения = 0.0005

RAM speed: 2667 MT/s

Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz

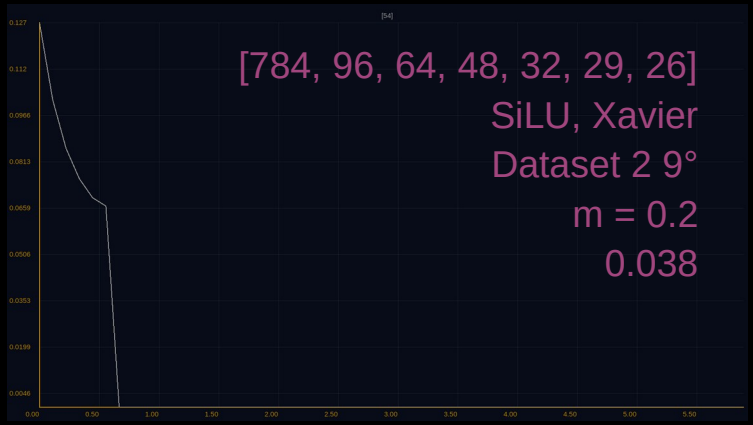
Caches (sum of all):

L1d:	192 KiB (4 instances)
L1i:	128 KiB (4 instances)
L2:	2 MiB (4 instances)
L3:	6 MiB (1 instance)

Параметры

Форма	Размер сети, Мбайт	Скорость обработки данных, Forward + Backward букв/мс
[784, 200, 150, 100, 26]	4.9208	9.0861
[784, 300, 26]	5.8424	7.4774
[784, 300, 300, 26]	8.012	3.3015
[784, 64, 32, 26]	1.2772	35.8830
[784, 1024, 256, 64, 26]	26.036	0.4568
[784, 96, 64, 48, 32, 29, 26]	2.1142	17.6647
[784, 200, 150, 100, 50, 26]	5.0112	6.4937

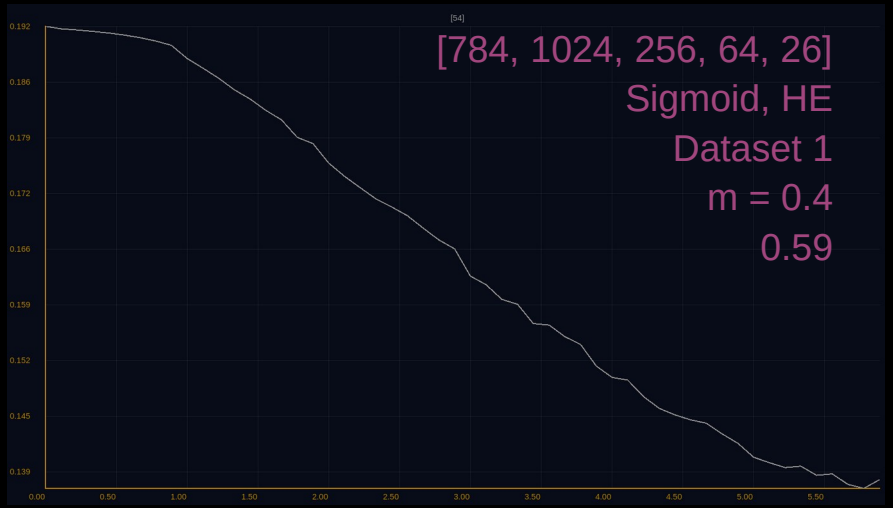
Примеры


[784, 96, 64, 48, 32, 29, 26]
SiLU, Xavier
Dataset 2 9°
m = 0.2
0.038


[784, 96, 64, 48, 32, 29, 26]
Tanh, Xavier
Dataset 2 9°
m = 0.2
0.56


[784, 96, 64, 48, 32, 29, 26]
ReLU, HE
Dataset 2 9°
m = 0.2
0.86

Примеры



Precision: 0.901 +- 0.0695
Recall: 0.900 +- 0.0680
F1: 0.900 +- 0.0643

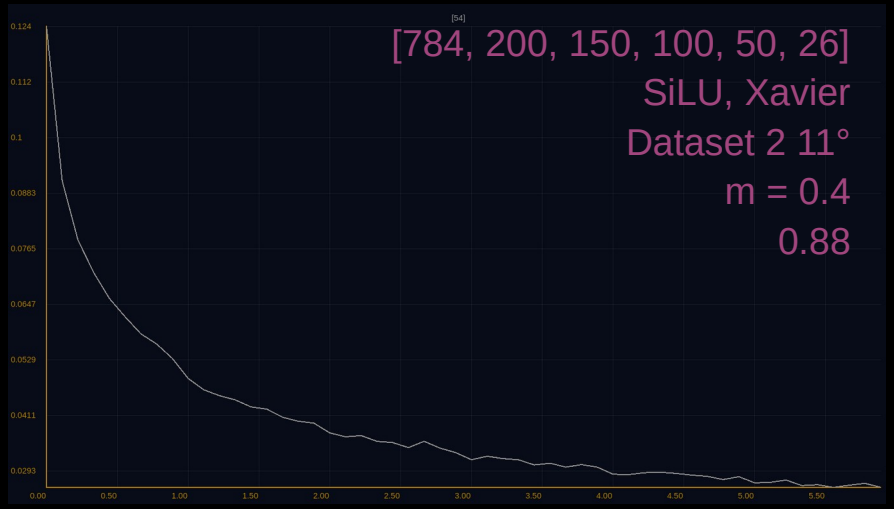
Precision: 0.588 +- 0.0876
Recall: 0.591 +- 0.138
F1: 0.590 +- 0.091

Примеры

Precision: 0.880533 +- 0.060524
Recall: 0.879685 +- 0.0702725
F1: 0.879176 +- 0.0600266

Precision: 0.861502 +- 0.075181
Recall: 0.860129 +- 0.0713544
F1: 0.859708 +- 0.0668606

Примеры

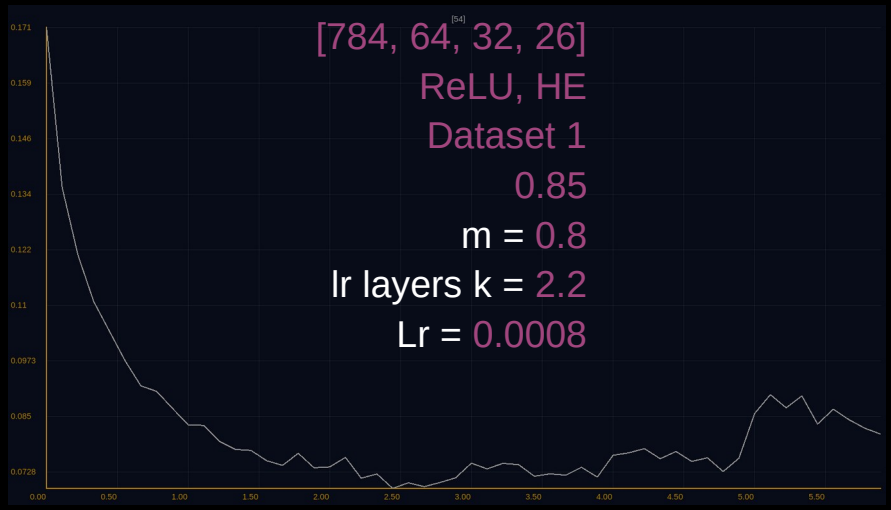


Precision: 0.884652 +- 0.0617349
Recall: 0.882515 +- 0.0715337
F1: 0.882424 +- 0.0607654

Precision: 0.862802 +- 0.073998
Recall: 0.859704 +- 0.0736224
F1: 0.859865 +- 0.0659351

Примеры

Precision: 0.851285 +- 0.084603
Recall: 0.844849 +- 0.0916267
F1: 0.844958 +- 0.0796247



Precision: 0.751197 +- 0.12285
Recall: 0.732319 +- 0.13682
F1: 0.730512 +- 0.10716