

Персептрон

Замирайлов Алексей
s21: maykitbo
Telegram: @zamyrailov

Тебе всё-таки придётся написать 1000+ строк на C++ вместо 10 на Python



5.11.2023

Перспетрон

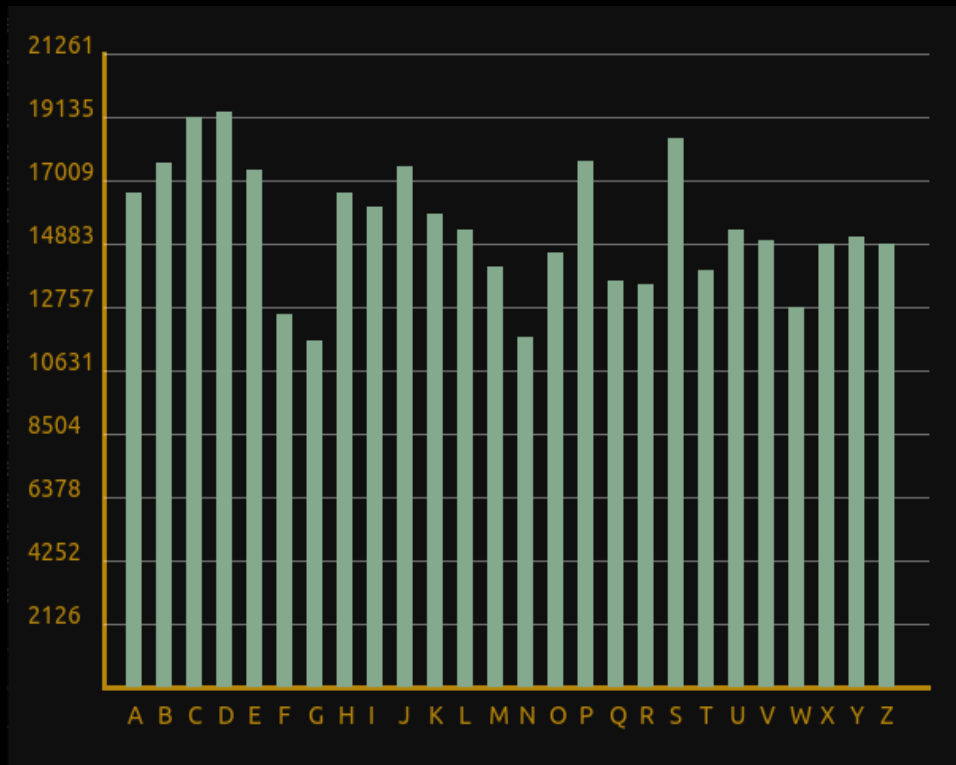
Работа с данными

- Сбор
- Подготовка
- Перемешивание
- Передача

Нейросеть

- Матрица весов
- Forward
- Backward
- Ускорение
- Примеры

Сбор данных

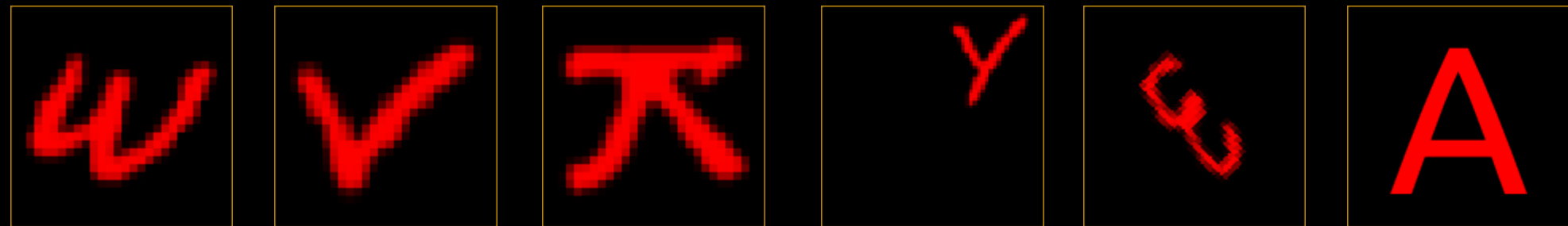
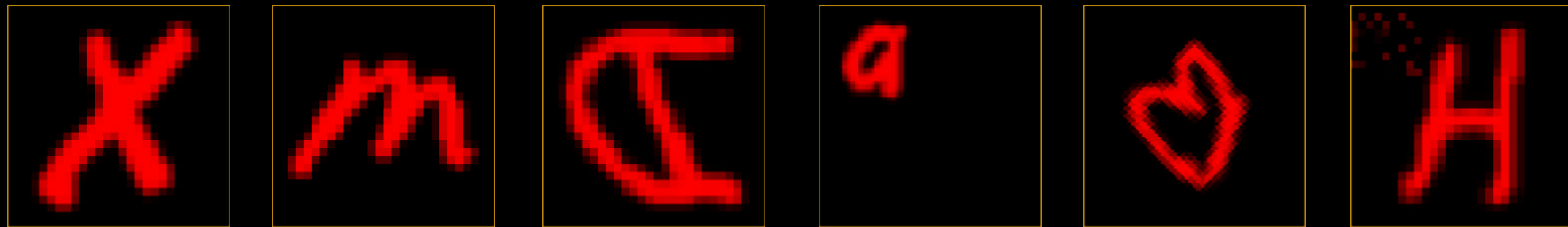


Плохое распределение



Хорошее распределение

Сбор и обработка данных



1

2

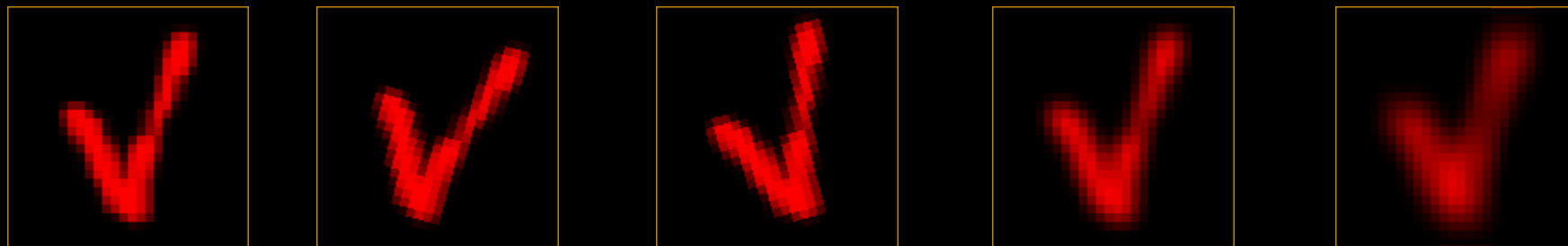
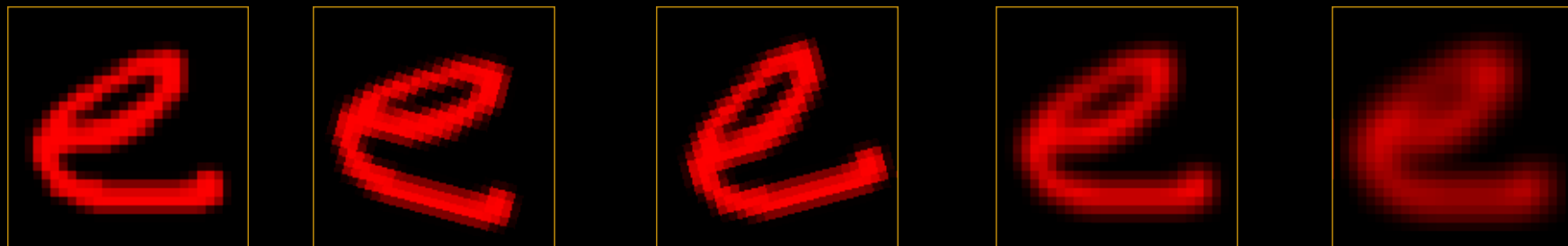
3

4

5

4

Обработка данных



original

Rotate 15°

Rotate -15°

Blure R1

Blure R2

Передача данных в нейрость

p 1 1	p 1 2	...	p 1 n
p 2 1	p 2 2	...	p 2 n
...
p n 1	p n 2	...	p n n

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

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

Один раз:

```
random_shuffle(data.begin(), data.end())
```

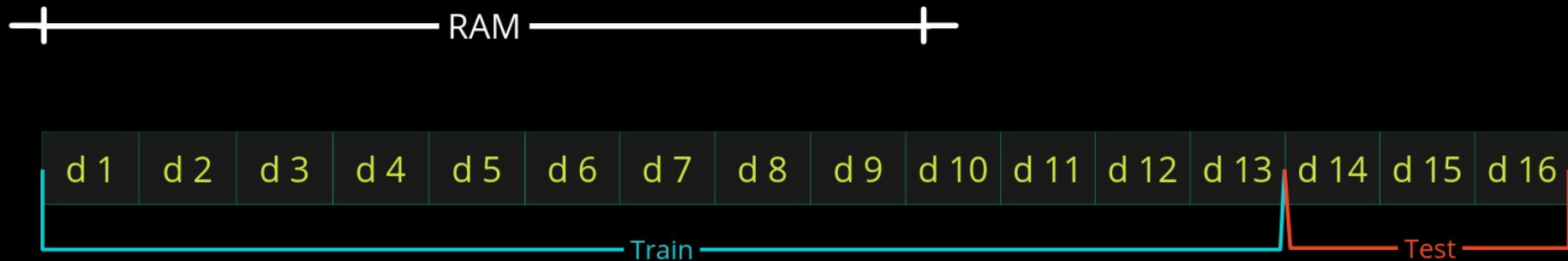


Перемешиваем тренировочную выборку перед каждой эпохой:

```
random_shuffle(data.begin(), data.begin() + train_size)
```

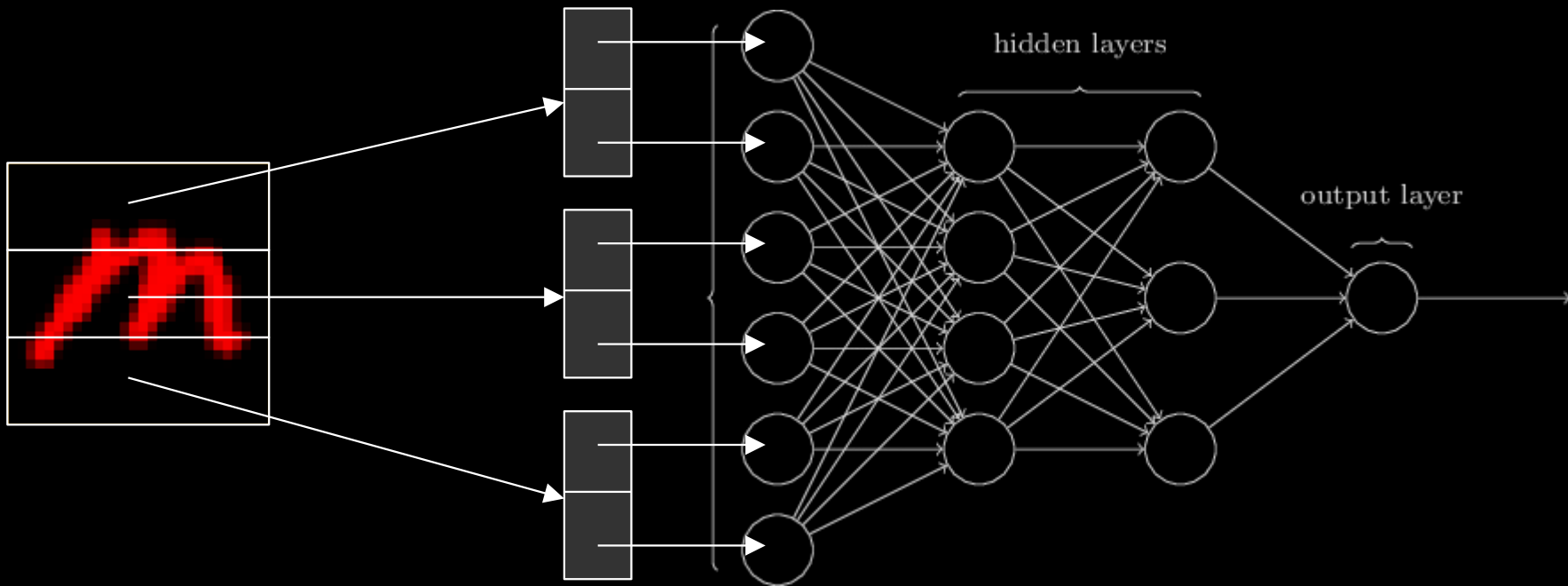
miro

Проблемы RAM



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

Персептрон



Персептрон. Структура проекта

```
typedef vector<float> vector_t;
typedef Matrix<float> matrix_t;

struct Layer
{
    vector_t output;
    vector_t biases;
    matrix_t weights;

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

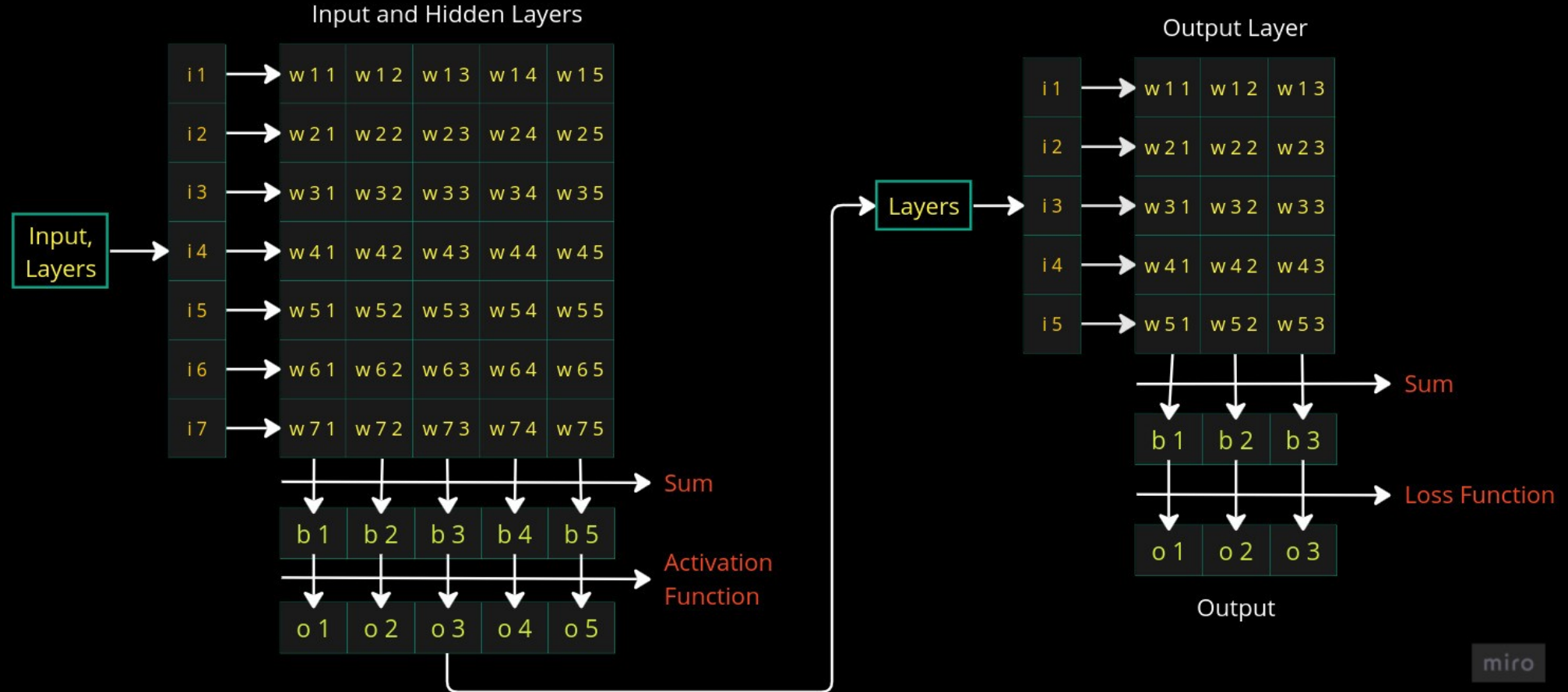
```
struct Settings
{
    vector<size_t> layers;
    float learning_rate;
    function<float(float)> activation;
    function<float(float)> derivative_activation;
};

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

Матрица весов. Forward



Forward



Forward

```
void NeuronNetwork::Forward(vector_t &letter)
{
    // вход первого слоя - входной вектор
    vector_t input = &letter;
    // для каждого слоя кроме последнего
    for (size_t i = 0; i < layers.size() - 1; ++i)
    {
        layers[i].HiddenSignal(input);
        // выход текущего слоя - вход следующего
        letter = &layers[i].output;
    }
    // последний слой
    layers.back().OutputSignal(input);
}
```

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

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

Backward

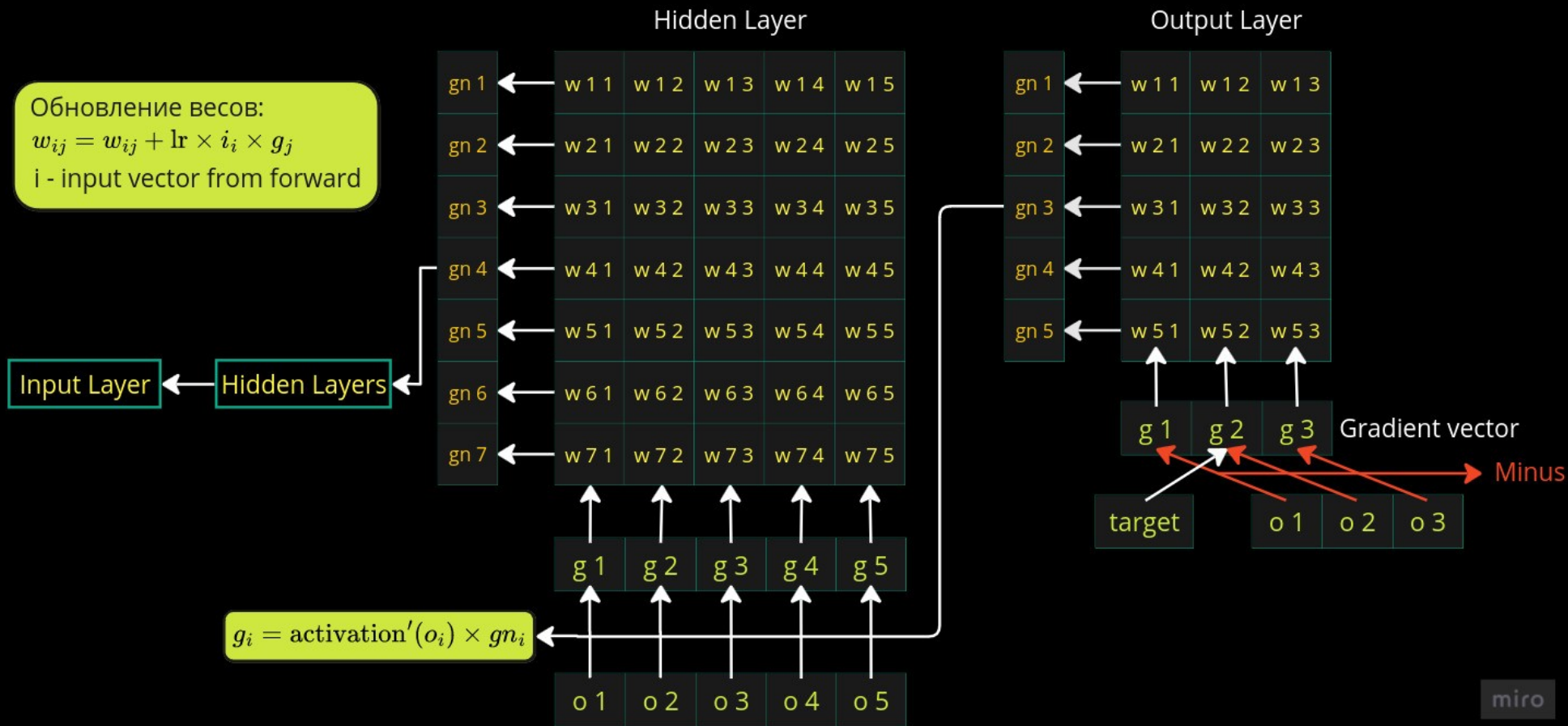
```
struct Layer
{
    vector_t output;
    vector_t biases;
    matrix_t weights;

    vector_t gradient;
    vector_t gradient_sum;
    matrix_t gradient_matrix;
};
```

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

    , gradient(cols)
    , gradient_sum(cols)
    , gradient_matrix(rows, cols)
{ }
```

Backward



Backward

```
void NeuronNetwork::Backward(vector_t &input
                             , uint target)
{
    size_t 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()
{
    // matrix = matrix + vector^T * vector
    gradient_matrix += gradient * prev_layer.output;
    gradient_sum += gradient;
}

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

void Layer::ProcessGradient()
{
    gradient *= DerivativeActivation(output);
}

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

Обновление весов

```
// после каждого батча:  
void NeuronNetwork::UpdateWeights()  
{  
    for (auto &layer : layers)  
    {  
        layer.UpdateWeights();  
    }  
}  
  
void Layer::UpdateWeights()  
{  
    biases += learning_rate * gradient_sum;  
    weights += learning_rate * gradient_matrix;  
  
    // обнуляем градиенты  
    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...
4. Профиляторы: perf...
3. cache friendly Пример:

```
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);
    }
}
```

```
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_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 $+9^\circ$



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):

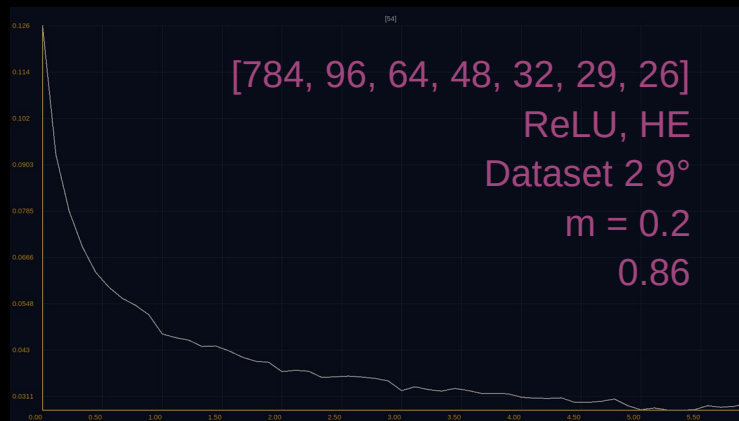
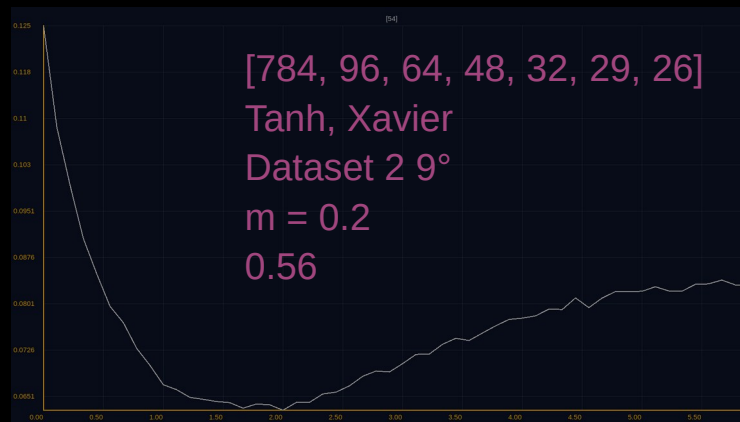
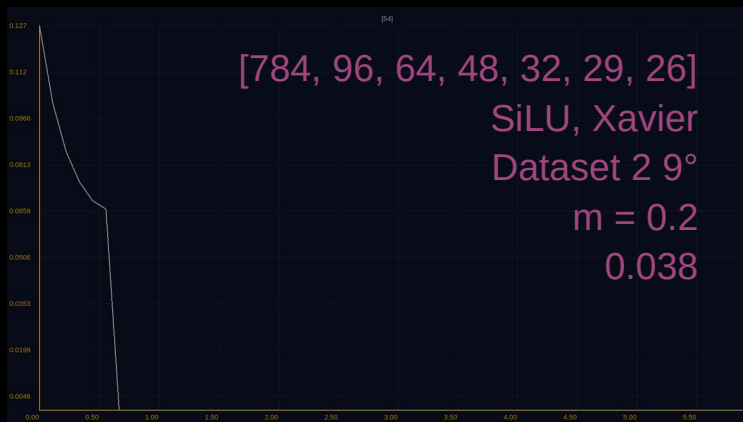
L1d:	192 KiB (4 instances)
L1i:	128 KiB (4 instances)
L2:	2 MiB (4 instances)
L3:	6 MiB (1 instance)

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

miro

Примеры



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

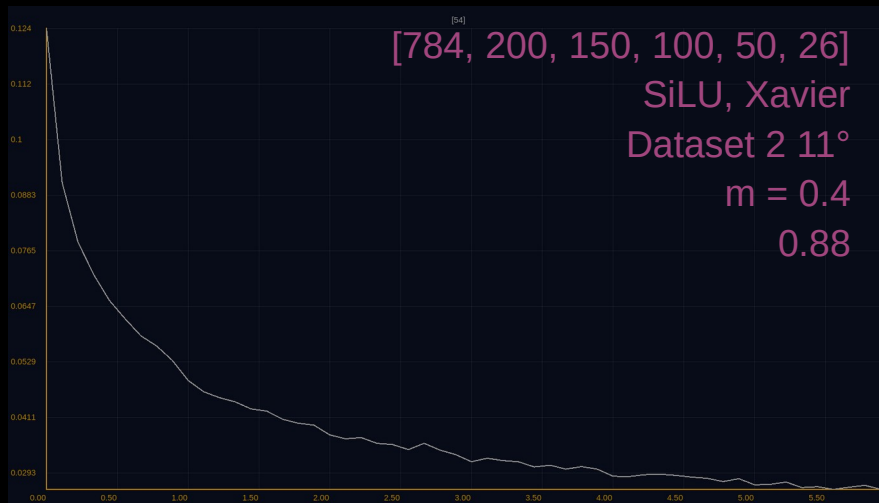


Precision: 0.880533 +- 0.060524
Recall: 0.879685 +- 0.0702725
F1: 0.879176 +- 0.0600266



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