Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский Авиационный Институт»
(Национальный Исследовательский Университет)

Факультет №8 «Информационные технологии и прикладная математика»
Кафедра 805 «Прикладная математика»

Реферат

по курсу «Языки и методы программирования»

2 семестр

Тема:

Создание нейронной сети на языке C++

Автор:

Студент 1 курса, гр. М80-103Б-21

Белоносов Кирилл Алексеевич

Руководитель:

Севастьянов Виктор Сергеевич

Москва, 2022

Оглавление

Введение 3

Знакомство с нейронными сетями 4

Написание простейшей нейронной сети с использованием сигмоидальной функции 6

Написание классификатора для распознавания цифр 9

Написание нейронной сети на Python с использованием библиотеки Keras 12

Итоги 14

Список литературы: 15

Ссылки 16

Введение

Основной целью данного проекта является знакомство с архитектурой нейронных сетей, написание простейшей нейронной сети для распознавания цифр.

Основные задачи:

1. Знакомство с устройством нейронных сетей
2. Написание простейшей нейронные сети с использованием сигмоидной функции активации
3. Написание классификатора для распознавания цифр
4. Написание нейронной сети на Python с использованием библиотеки Keras
5. Подвести итоги

Знакомство с нейронными сетями

Нейронная сеть (также искусственная нейронная сеть, ИНС) — математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма. Это понятие возникло при изучении процессов, протекающих в мозге, и при попытке смоделировать эти процессы.

ИНС представляет собой систему соединённых и взаимодействующих между собой простых процессоров (искусственных нейронов). Такие процессоры обычно довольно просты (особенно в сравнении с процессорами, используемыми в персональных компьютерах). Каждый процессор подобной сети имеет дело только с сигналами, которые он периодически получает, и сигналами, которые он периодически посылает другим процессорам. И, тем не менее, будучи соединёнными в достаточно большую сеть с управляемым взаимодействием, такие по отдельности простые процессоры вместе способны выполнять довольно сложные задачи.

Несмотря на большое разнообразие вариантов нейронных сетей, все они имеют общие черты. Так, все они, так же, как и мозг человека, состоят из большого числа связанных между собой однотипных элементов – нейронов, которые имитируют нейроны головного мозга. Нейрон состоит из синапсов, связывающих входы нейрона с ядром; ядра нейрона, которое осуществляет обработку входных сигналов и аксона, который связывает нейрон с нейронами следующего слоя. Каждый синапс имеет вес, который определяет, насколько соответствующий вход нейрона влияет на его состояние. Состояние нейрона определяется по формуле:

$$S = \sum_{i=1}^{n} x_i w_i$$

Затем определяется значение аксона нейрона по формуле:

$$Y = f(S)$$

Где f – некоторая функция активации

Нейронные сети обратного распространения – это мощнейший инструмент поиска закономерностей, прогнозирования, качественного анализа. Такое название – сети обратного распространения (back propagation) они получили из-за используемого алгоритма обучения, в котором ошибка распространяется от выходного слоя к входному, т. е. в направлении, противоположном направлению распространения сигнала при нормальном функционировании сети.

Нейронная сеть обратного распространения состоит из нескольких слоев нейронов, причем каждый нейрон слоя i связан с каждым нейроном слоя i+1, т. е. речь идет о полносвязной НС.

В общем случае задача обучения НС сводится к нахождению некой функциональной зависимости Y=F(X) где X – входной, а Y – выходной векторы. В общем случае такая задача, при ограниченном наборе входных данных, имеет бесконечное множество решений. Для ограничения пространства поиска при обучении ставится задача минимизации целевой функции ошибки НС.

Обучение нейросети производится методом градиентного спуска, т. е. на каждой итерации изменение веса производится по формуле:

$$\Delta w_{ij} = -\eta * \frac{\partial E}{\partial w_{ij}}$$

Где η – параметр отвечающий за скорость обучения

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i} * \frac{dy_i}{dS_j} * \frac{\partial S_j}{\partial w_{ij}}$$

При этом множитель:

$$\frac{\partial S_j}{\partial w_{ij}} = x_i$$

Введем вспомогательную переменную:

$$\delta_j^n = \left[\sum_k \delta_k^{n+1} * w_{jk}^{n+1}\right] * \frac{\delta y_j}{\delta S_j}$$

Тогда изменение веса равно:

$$\Delta w_{ij}^n = -\eta * \delta_j^n * x_i^n$$

Алгоритм обучения нейронной сети:

1. подать на вход НС один из требуемых образов и определить значения выходов нейронов нейросети
2. рассчитать для выходного слоя НС по формуле (12) и рассчитать изменения весов выходного слоя N по формуле (13)
3. Рассчитать по формулам (11) и (13) соответственно и Δw(N)ij для остальных слоев НС, n = N-1..1
4. Скорректировать все веса НС
5. Если ошибка существенна, то перейти на шаг 1

Написание простейшей нейронной сети с использованием сигмоидальной функции

Функция активации — это способ нормализации входных данных То есть, если на входе у нас будет большое число, пропустив его через функцию активации, мы получите выход в нужном вам диапазоне.

В данном сети я использую сигмоидальную (логистическую) функцию

$$f(x) = \frac{1}{1 + e^{-x}}$$

Для начала создам класс нейрона и Нейронной сети.

```
class Neuron {
public:
    void sigmoid();
    void sum();
    std::vector<Neuron *> inputs;
    std::vector<double> weight;
    std::vector<double> changeWeight;
    double zi;
    double hi;
    double delta;
};
```

```
class NeuronSystem {
public:
    NeuronSystem(std::vector<int> NeuronCount, double E, double A);
    void neuronBias(int CurrentLay);
    void resizeLays(int numberLays, int countNeuron);
    void inputWeight (int layNumber, std::ifstream * fin);
    void forwardfeed(std::string activationFunction);
    void IterationSigmoid(int CurrentLay);
    void IterationRelu(int CurrentLay);
    void IterationEndSigmoid();
    void Softmax();
    double ErrorQuad(std::vector<double> output);
    double CrossEntropy(std::vector<double> answers);
    void backpropagation(std::string activationFunction);
    void deltaEndSigmoid(std::vector<double> output);
    void deltaEndRelu(std::vector<double> answers);
    void deltaSigmoid(int CurrentLay);
    void deltaRelu(int CurrentLay);
    void ChangeWeight(int CurrentLay);
    void NeuronSystemStart(std::vector<std::vector<double>> input, std::vector<double> output, std::string activat
    double Is(int i, int j);
    void currentWeight (int currentLay, std::ofstream * fout);
    void coutWeight ();
    void NeuralSystemAnswer(std::vector<std::vector<double>> input, std::string activatonFunction);
    double E;
    double A;
private:
    std::vector<std::vector<Neuron>> Lays;
};
```

Создам конструктор нейронной сети:

```
NeuronSystem::NeuronSystem(std::vector<int> NeuronCount, double E, double A) : E(E), A(A) {
    int NeuronCounts = NeuronCount.size();
    Lays.resize(NeuronCounts);
    for (int i = 0; i != NeuronCounts; ++i) {
        resizeLays(i, NeuronCount[i]);
    }
    for (int i = 0; i != NeuronCounts - 1; ++i) {
        neuronBias(i);
    }
    std::ifstream fin;
    fin.open("input.txt");
    for (int i = 1; i != NeuronCounts; ++i) {
        inputWeight(i, &fin);
    }
    fin.close();
}
```

При создании нейронной сети количество слоев и нейронов будет импортироваться из вектора NeuronCount. Переменные E и A будут задавать гиперпараметры сети. В результате создается структура нейронной сети, где синапсы имеют веса из файла.

Далее создам функцию старта нейронной сети:

```
void NeuronSystem::NeuronSystemStart(std::vector<std::vector<double>> input, std::vector<double> output, std::string activatonFunction) {
    if (activatonFunction == "sigmoid") {
        double ErrorSystem;
        int Epoch = 0;
        while (Epoch != 1000000) {
            double InError = 0;
            int inputSize = input.size();
            int inputSize1 = input[0].size();
            for (int i = 0; i != inputSize; ++i) {
                for (int j = 0; j != inputSize1; ++j) {
                    Lays[0][j].hi = input[i][j];
                }
                forwardfeed("sigmoid");
                InError = ErrorQuad(output);
                deltaEndSigmoid(output);
                backpropagation("sigmoid");

            }
            ErrorSystem = InError / input.size();
            std::cout << "Error: "<< ErrorSystem<<"\n";
            std::cout << "\n";
            ++Epoch;
        }
        std::cout << "Complete" <<"\n";
    }
```

Происходит установка количества эпох и в цикле происходит полный алгоритм обучения.

В функции forwardfeed происходит прямой проход по нейронной сети.

```
void NeuronSystem::forwardfeed(std::string activationFunction) {
    if (activationFunction == "sigmoid") {
        int CountOfLays = Lays.size();
        for (int i = 1; i != CountOfLays - 1; ++i) {
            IterationSigmoid(i);
        }
        IterationEndSigmoid();
```

В функции ErrorQuad вычисляется среднеквадратичная ошибка:

```
double NeuronSystem::ErrorQuad(std::vector<double> output) {
    int LastLayer = Lays.size() - 1;
    int CountNeuron = Lays[LastLayer].size();
    double Error = 0;
    for (int i = 0; i != CountNeuron; ++i) {
        Error += pow(Lays[LastLayer][i].hi - output[i], 2);
    }
    return Error;
}
```

Функция deltaEndSigmoid вычисляет значение δ на выходных нейронных

```
void NeuronSystem::deltaEndSigmoid(std::vector<double> output) {
    int LastLayer = Lays.size() - 1;
    int CountNeuron = Lays[LastLayer].size();
    for (int i = 0; i != CountNeuron; ++i) {
        Lays[LastLayer][i].delta = (output[i] - Lays[LastLayer][i].hi)*((1 - Lays[LastLayer][i].hi)*Lays[LastLayer][i].hi);
    }
}
```

В функции backpropagation происходит вычисление ошибок на каждом нейроне и вычисление изменения весов.

```
void NeuronSystem::backpropagation(std::string activationFunction) {
    if (activationFunction == "sigmoid") {
        int CountOfLay = Lays.size();
        for (int i = CountOfLay - 2; i != 0; --i) {
            deltaSigmoid(i);
        }
        for (int i = CountOfLay - 1; i != 0; --i) {
            ChangeWeight(i);
        }
    }
```

В итоге полученная нейронная сеть может классифицировать 2 вида предмета. Например, её можно использовать для XOR сети.

Данная сеть имеет мало практического применения и служит чаще всего для ознакомления с устройством нейронной сети.

Написание классификатора для распознавания цифр

Данная нейронная сеть использует 2 функции активации relu и softmax

ReLU:

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

Softmax:

$$f_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_i}}$$

На вход нейронной сети будет поступать изображение 28x28, а на выходном слое будет 10 нейронов, где будет находиться вероятность принадлежности к определенной цифре.

Конструктор нейронной сети останется из предыдущей сети

Напишем функцию старта нейронной сети

```
else if(activatonFunction == "relu") {
    double ErrorSystem;
    int Epoch = 0;
    while (Epoch != 20) {
        double InError = 0;
        int inputSize = input.size();
        int inputSize1 = input[0].size();
        for (int i = 0; i != inputSize; ++i) {
            for (int j = 0; j != inputSize1; ++j) {
                Lays[0][j].hi = input[i][j];
            }
            int LastLayer = Lays.size() - 1;
            std::vector<double> answers(Lays[LastLayer].size(), 0);
            for (int j = 0; j != Lays[LastLayer].size(); ++j) {
                answers[output[i]] = 1;
            }
            double AtError;
            forwardfeed("relu");
            AtError = CrossEntropy(answers);
            deltaEndRelu(answers);
            backpropagation("relu");
            InError += AtError;
        }
        ErrorSystem = InError / input.size();
        std::cout << "----------Epoch Error: "<< ErrorSystem<<"\n";
        std::cout << "\n";
        ++Epoch;
    }
}
```

Перепишем forwardfeed для функции ReLU, на выходных нейронах вызывается функция Softmax

```
else if (activationFunction == "relu") {
    int CountOfLays = Lays.size();
    for (int i = 1; i != CountOfLays - 1; ++i) {
        IterationRelu(i);
    }
    Softmax();
}
```

Реализую логарифмическую функцию потерь (кросс-энтропию)

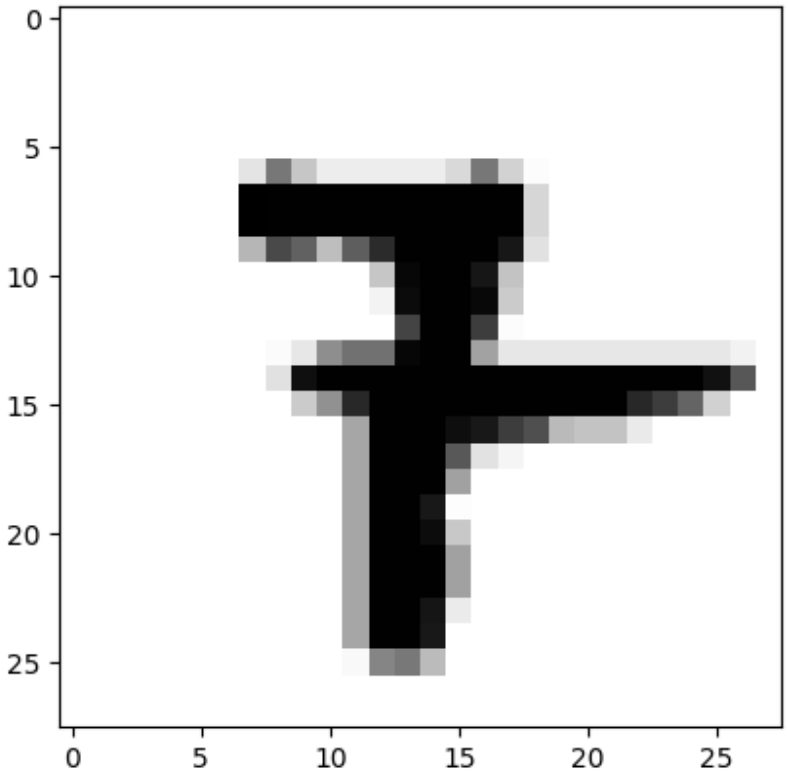
```
double NeuronSystem::CrossEntropy(std::vector<double> answers) {
    int LastLayer = Lays.size() - 1;
    int CountNeuron = Lays[LastLayer].size();
    double Error = 0;
    for (int i = 0; i != CountNeuron; ++i) {
        Error += answers[i]* log(Lays[LastLayer][i].hi);
    }
    return -Error;
}
```

Также перепишу backpropagation и напишу deltaRelu

```
    else if (activationFunction == "relu") {
        int CountOfLay = Lays.size();
        for (int i = CountOfLay - 2; i != 0; --i) {
            deltaRelu(i);
        }
        for (int i = CountOfLay - 1; i != 0; --i) {
            ChangeWeight(i);
        }
    }
}
```

```
void NeuronSystem::deltaRelu(int CurrentLay) {
    int CountNeuron = Lays[CurrentLay].size();
    for (int i = 0; i != CountNeuron; ++i) {
        int CountNeuronNext = Lays[CurrentLay + 1].size();
        double sum = 0;
        for (int j = 0; j != CountNeuronNext; ++j) {
            sum += Lays[CurrentLay + 1][j].delta * Lays[CurrentLay + 1][j].weight[i];
        }
        Lays[CurrentLay][i].delta = ((Lays[CurrentLay][i].zi > 0) ? 1 : 0) * sum;
    }
}
```

Для обучения нейронной сети используем dataset mnist содержащий изображения рукописных цифр.



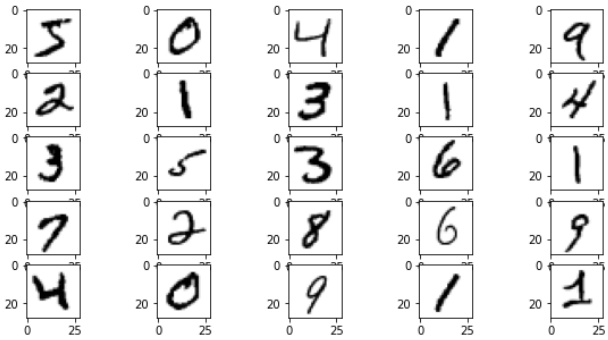
Погрешность данной нейронной сети составляет примерно 97%.

Написание нейронной сети на Python с использованием библиотеки Keras

Реализуем данную нейронную сеть на языке Python, используя библиотеку Keras. Напишу программу используя Jupyter Notebook

Ввод [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.python import keras
from keras.datasets import mnist
from tensorflow import keras
from keras.layers import Dense, Flatten

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train/255
x_test = x_test/255

y_train_cat = keras.utils.to_categorical(y_train, 10)
y_test_cat = keras.utils.to_categorical(y_test, 10)
plt.figure(figsize=(10, 5))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks()
    plt.yticks()
    plt.imshow(x_train[i], cmap=plt.cm.binary)

plt.show()

model = keras.Sequential([])
```



Ввод [2]:

```
model = keras.Sequential([Flatten(input_shape=(28, 28, 1)), Dense(128, activation='relu'), Dense(10, activation='softmax')])

print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
None
```

Ввод [3]:

```
model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Ввод [4]:
```
model.fit(x_train, y_train_cat, batch_size=32, epochs=5, validation_split=0.2)
```

```
Epoch 1/5
1500/1500 [==============================] - 6s 2ms/step - loss: 0.2846 - accuracy: 0.9197 - val_loss: 0.1501 - val_accuracy:
0.9580
Epoch 2/5
1500/1500 [==============================] - 3s 2ms/step - loss: 0.1233 - accuracy: 0.9636 - val_loss: 0.1156 - val_accuracy:
0.9657
Epoch 3/5
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0840 - accuracy: 0.9759 - val_loss: 0.0952 - val_accuracy:
0.9707
Epoch 4/5
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0628 - accuracy: 0.9814 - val_loss: 0.0999 - val_accuracy:
0.9690
Epoch 5/5
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0495 - accuracy: 0.9854 - val_loss: 0.0835 - val_accuracy:
0.9742
```

Out[4]:
```
<keras.callbacks.History at 0x23f26882c40>
```

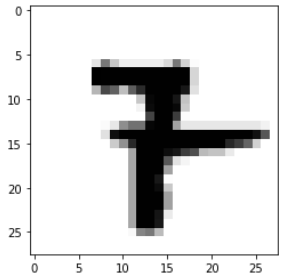
Ввод [5]:
```
model.evaluate(x_test, y_test_cat)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0822 - accuracy: 0.9749
```

Out[5]:
```
[0.08218580484390259, 0.9749000072479248]
```

Ввод [6]:
```
n = 97
x = np.expand_dims(x_test[n], axis=0)
res = model.predict(x)
print(res)
print(f"Распознаная цифра: {np.argmax(res)}")

plt.imshow(x_test[n], cmap=plt.cm.binary)
plt.show()
```

```
[[4.8874012e-09 2.8235360e-07 3.5314388e-06 2.9772744e-04 7.0396467e-08
  4.2646703e-10 1.4662250e-12 9.9969792e-01 4.5509672e-08 3.8343919e-07]]
Распознаная цифра: 7
```



Точность также составила примерно 97%

Итоги

Данная работа была очень интересной, благодаря ей я познакомился с основами нейронных сетей, реализовал простейшие нейронные сети, также я познакомился с библиотекой keras для разработки нейронных сетей. Как можно заметить, скорость разработки НС намного выше при использовании библиотек. Работа имеет пути дальнейшего развития, например, создание сверточной нейронной сети.

Список литературы:

1. https://habr.com/ru/post/312450/
2. https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C
3. https://basegroup.ru/community/articles/math
4. https://www.monographies.ru/ru/book/section?id=2465
5. http://neuralnetworksanddeeplearning.com/
6. https://www.youtube.com/watch?v=5031rWXgdYo

Ссылки

1. https://github.com/KiRiLLBEL/Projects