Руководство програмииста

Документация Neural Network Constructor

Общая информация

Проект представляет из себя приложение для наглядного обучения нейросети, в котором пользователь может посмотреть, как изменяются гиперпараметры сети во время ее обучения. Проект написан на C++, для написания фронта также использовалась библиотека [QT](Qt | Tools for Each Stage of Software Development Lifecycle).

Сборка и запуск

Зависимости:

1. *CMake* ≥ v3.1.0
2. *Qt6* ≥ v6.2.4
3. *C++* ≥ 17

```
git clone https://github.com/Neural-Network-Constructor/Neural-Network-Constructor.git
cd Neural-Network-Constructor
mkdir build; cd build
cmake ..; make
```

Backend

Функции активации

Прототипы стандартных функций активации находятся в файле `backend/Model/Activation.h` и лежат в неймспейсе `activation`.

В проекте представлены следующие функции активации:

ReLU

Функция

$$ReLU(x) = max(0, x)$$

Производная

$$ReLU'(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

Sigmoid

Функция

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

Производная

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Tanh

Функция

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

Производная

$$Tanh'(x) = 1 - Tanh^2(x)$$

Softmax

Функция

$$Softmax(z_i) = \frac{e^{z_i}}{\Sigma_{j=1}^{N} e^{z_j}}$$

Производная

$$\frac{\delta S(z_i)}{\delta z_j} = \begin{cases} S(z_i) * (1 - S(z_i)) & i = j \\ -S(z_i) * S(z_(j) & i \neq j \end{cases}$$

Слои

Все слои, имеющиеся в проекте (`InputLayer`, `FullyConnectedLayer`) наследуются от класса `Layer`.

Прототипы полей класса `Layer`:

```
class Layer {
protected:
  bool is_output = 0;        // Являлется ли слой выходным
  uint32_t inputs_num = 0;   // Количество входный слоев
  uint32_t outputs_num = 0; // Количество выходный слоев
  uint32_t values_num = 0;   // Количество нейронов в слое
  std::vector<Layer *> inputs; // Указатели на входные слои
  std::vector<Layer *> outputs; // Указатели на выходные слои
  std::vector<double> values; // Значения нейронов в слое
  std::function<double(double)> activation; // Функция активации
  std::function<double(double)>
      activation_derivative; // Приозводная функции активации

  // Создает связь между слоем и его выходом
  // (вызывается из экземляра слоя и передается выходной слой)
  virtual void AddOutput(Layer *&);

public:
  Layer() = default;
  ~Layer() = default;

  // Конструктор от количества нейронов в слое, функции активации и
ее
  // производной
  Layer(const uint32_t &, const std::function<double(double)> &,
        const std::function<double(double)> &);
  // Конструктор от количества нейронов в слое, функции активации,
ее
  // производной и переменной, которая показывает является ли слой
выходным
  Layer(const uint32_t &, const std::function<double(double)> &,
```

```
        const std::function<double(double)> &, const bool &);

  // Функция, которая считает значения нейронов в слое и сохраняет
их в values
  virtual void Predict();

  // Зануление значений нейронов в слое
  virtual void ClearValues();

  // Геттер для values
  virtual std::vector<double> GetValues();

  // Создает связь между слоем и его входом
  // (вызывается из экземляра слоя и передается входной слой)
  // а также инициализирует все гиперпараметры для обучения
  virtual void ConnectTo(Layer *&);

  // Геттер для values_num
  uint32_t GetValuesNum() const;
};
```

Прототипы полей класса `FullyConnectedNeuralNetwork`:

```
class FullyConnectedLayer : public Layer {
private:
  std::map <Layer*, std::vector <double>> weights; // Список весов
нейронов (ключ - указатель на слой, значение - вектор весов)
  std::map <Layer*, std::vector <double>> biases; // Список смещений
нейронов (ключ - указатель на слой, значение - вектор смещений)

public:
  // Конструктор от количества нейронов в слое, функции активации и
ее
  // производной
  FullyConnectedLayer(const uint32_t &, const
std::function<double(double)> &, const std::function<double(double)>
&);
  // Конструктор от количества нейронов в слое, функции активации,
ее
  // производной и переменной, которая показывает является ли слой
выходным
```

```
  FullyConnectedLayer(const uint32_t &, const
std::function<double(double)> &, const std::function<double(double)>
&, const bool &);

  // Функция, которая считает значения нейронов в слое и сохраняет
их в values
  void Predict();
};
```

Прототипы полей класса `InputLayer`:

```
class InputLayer : public Layer {
private:
public:
  // Конструктор от количества нейронов в слое
  InputLayer(const uint32_t &);
  // Конструктор от количества нейронов в слое и
  // переменной, которая показывает является ли слой выходным
  InputLayer(const uint32_t &, const bool &);

  void SetValues(const std::vector <double> &);
};
```

Алгоритм обучения

Для обучения модели мы используем [метод обратного распространения ошибки](Метод обратного распространения ошибки — Википедия (wikipedia.org)) и [стохастический градиентный спуск](Стохастический градиентный спуск — Википедия (wikipedia.org)). При этом над надо высчитывать частные производные функции ошибки по каждому весу и смещению.

Производная функции ошибки по смещению нейрона

$$\frac{\delta C_0}{\delta b^{(L)}} = 2(a^{(L)} - y)\sigma'(Z^{(L)})$$

Здесь

- C_0 – значение ошибки

- $b^{(L)}$ – смещение нейрона
- $a^{(L)}$ – значение текущего нейрона
- y – требуемое значение нейрона
- $\sigma'(x)$ – производная функции активации
- $Z^{(L)}$ – значение текущего нейрона до применения функции активации

Производная функции ошибки по весу нейрона

$$\frac{\delta C_0}{\delta w^{(L)}} = 2(a^{(L)} - y)\sigma'(Z^{(L)})a^{(L-1)}$$

Здесь

- C_0 – значение ошибки
- $w^{(L)}$ – значение веса
- $a^{(L)}$ – значение текущего нейрона
- $a^{(L-1)}$ – значение нейрона, с которым соединяется текущий нейрон данным весом
- y – требуемое значение нейрона
- $\sigma'(x)$ – производная функции активации
- $Z^{(L)}$ – значение текущего нейрона до применения функции активации

Frontend

Написаны основные классы: **Node** – слои нейронов, **Edge** – соединения между слоями, **App** – само приложение.

Прототипы полей класса Node

```
class Node : public QGraphicsItem
{
    friend Edge::Edge(Node *sourceNode, Node *destNode);
    friend Edge::~Edge();
public:
    Node(int type); // конструктор
    ~Node() override; // деструктор
    QList<Edge *> edges() const; // массив рёбер

    enum { Type = UserType + 1 };
    int type() const override { return Type; }
```

```
    static const int RADIUS = 30; // базовый радиус
    QRectF boundingRect() const override;
    QPainterPath shape() const override;
    void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, // рисование вершины
               QWidget *widget) override;

    bool mark() const; // проверка на выделенность
    void addEdge(Edge *edge); // добавление ребра
    void setMark(bool mark); // установка выделенности

protected:
    QVariant itemChange(GraphicsItemChange change, // смена фокуса
на вершине
                        const QVariant &value) override;
    void mousePressEvent(QGraphicsSceneMouseEvent *event) override;
// нажатие клавиши мыши
    void mouseReleaseEvent(QGraphicsSceneMouseEvent *event)
override; // удержание клавиши мыши
    void removeEdge(Edge *edge); // удаление ребра
private:
    QList<Edge *> edgeList; // массив ребёр, связанных с ним
    bool _mark; // метка выделенности
    int _type; // тип нейрона
};
```

Класс полей **Node** наследуется от встроенного в Qt класса **QGraphicsItem**, позволяющего использовать графику и хранить граф слоёв нейронов. Созданные функции позволяют рисовать, перемещать и соединять рёбрами отдельные экземпляры класса.
Также реализована реакция нейронов на нажатие.

Прототипы полей класса Edge

```
class Edge : public QGraphicsItem
{
public:
    Edge(Node *sourceNode, Node *destNode); // конструктор
    ~Edge() override; // деструктор
    Node *sourceNode() const; // исходная вершина
```

```
    Node *destNode() const; // целевая вершина

    void adjust(); //

    enum { Type = UserType + 2 };
    int type() const override { return Type; }

protected:
    QRectF boundingRect() const override;
    void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget) override;

private:
    Node *source, *dest;

    QPointF sourcePoint;
    QPointF destPoint;
    qreal arrowSize;
};
```

Прототипы полей класса App

```
class App : public QMainWindow {
    Q_OBJECT

public:
    App(int width, int height); // конструктор
    ~App() override; // деструктор

    void render() { // создание окна программы
        window_->show();
    }

private slots:
    void loadFromFile(); // загрузка из файла
    void gotoBegin(); // возвращение к стартовому экрану
    void gotoEditor(); // возвращение к редактору
    void gotoSimulator(); // возвращение к симуляции

    void drawInNeuron(); // рисование входного нейрона
    void drawFCNNeuron(); // рисование основного нейрона
```

```
    void drawOutNeuron(); // рисование выходного нейрона
    void deleteNeuron(); // удаление нейрона
    void graphWidgetClicked(QMouseEvent *event); // создание ребра

private:
    int connProcess;
    std::vector<Node*> nodes_;
    QWidget* window_;
    QLabel* base_line_;
    QPushButton* begin_button_;
    QPushButton* editor_button_;
    QPushButton* simulation_button_;


    QLabel* begin_;
    QPushButton* load_button_;
    QPushButton* create_button_;

    QLabel* editor_;
    QPushButton* in_neuron_;
    QPushButton* fcn_neuron_;
    QPushButton* out_neuron_;
    QPushButton* delete_neuron_btn_;
    QPushButton* add_edge_btn_;
    QPushButton* start_simulating_btn_;

    QGraphicsView* edit_tablet_;
    QGraphicsScene* edit_scene_;
    QPainter* painter_;

    QWidget* creating_tablet_;
    QLineEdit* x_coord_;
    QLineEdit* y_coord_;
    QPushButton* neuron_painter_;

    QTabBar* tabbar_;
    QLabel* logtext_;
    QTabWidget* maintab_;
};
```

Класс App является основным классом приложения, в нём реализована основная его структура: кнопки и все интерактивные способы взаимодействия пользователя. Кнопки сохранения, открытия файлов, перехода между вкладками приложения, редактирования и создания нейронов, проигрывание симуляции обучения. Весь класс реализован с помощью CSS встроенных классов Qt6.