**3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе мы рассмотрим основные принципы работы приложения.

**3.1** Модуль библиотеки нейронных сетей

Центральная часть программного комплекса представляет собой набор компонентов, с помощью которых можно создавать различные типы нейронных сетей и проводить их обучение. Пользователю приложения предоставляется гибкое API для задания конфигурации нейронной сети и ее обучения. В состав библиотеки нейронных сетей входят:

* реализации различных типов слоев нейронной сети
* компоненты для представления входных и выходных данных сети
* модуль обучения сети
* компоненты клиентского интерфейса для взаимодействия с библиотекой
* модуль параллельных вычислений

Рассмотрим каждый из описанных выше модулей

Структура нейронной сети представляет собой несколько слоев нейронов, соединенных последовательно. Каждый слой состоит из определенного количества нейронов. Но на этапе проектирования было принято решение не реализовывать отдельные нейроны как простейшие элементы сети. Это значит, что базовым компонентом для построения структуры сети является слой нейронов. Такой подход позволяет ускорить разработку, а также повысить быстродействие приложения и уменьшить используемую память, так как сокращается количество создаваемых в ходе работы объектов.

**3.1.1** Во время работы сети, данные, которые попадают на вход поочередно проходят через каждый слой сети и в итоге преобразуются в выходной набор данных который является ответом сети на входные параметры. При прохождении данных через сеть размерность данных меняется. Например, на вход сети может подаваться цветное RGB изображение (размерность массива данных 25х25х3), а на выходе мы получаем вектор предсказаний размерностью 10. Поэтому модель, которая используется для хранения данных должна иметь возможность менять размерность данных по ходу работы. Было принято решение использовать для хранения данных внутри модели простой одномерный массив, но предоставить клиенту интерфейс, с помощью которого с объектом можно вести работу как с многомерным массивом.

Базовой моделью для представления массива данных в нейронной сети является класс DataSet. Данный класс может представлять одномерный, двумерный либо трехмерный массив и предоставляет набор операций над данными которые необходимы при работе с нейронными сетями.

public class DataSet {

private final List<Double> data = new ArrayList<>();

private Dimension dimension;

public List<Double> getData() {}

public DoubleStream getStreamData() {}

public Double[] getWrapperArrayData() {}  
  
public int getSize() {}  
  
public Dimension getDimension() {}  
  
public DataSet update(final List<Double> data, final

Dimension dimension) {}

public DataSet update(final DataSet dataSet) {}

public DataSet rotate() {}

public DataSet merge(final DataSet other, final DoubleBinaryOperator operator) {

public double[][] get2DArrayData() {}  
  
public double[] getArrayData() {}  
  
public double get(int idx) {}  
  
public double get(int widthIdx, int heightIdx, int

depthIdx){}  
  
public void set(int widthIdx, int heightIdx, int depthIdx,

double value) {}

Класс содержит два поля:

* data – массив данных типа double;
* dimension – объект класса, который хранит информацию о размерности данных (width, height, depth).

Методы класса позволяют проводить различные операции с данными. Перечислим некоторые из них:

* List<Double> getData() – возвращает одномерный массив данных, которые хранятся в модели.
* void rotate() – производит поворот массива данных на 180 градусов (rot180). Данная операция необходима для реализации двумерной свертки в сверточных нейронных сетях.
* double get(int widthIdx, int heightIdx, int depthIdx)- позволяет получить единичный элемент массива по заданным координатам. Во время вызова метода происходит проверка на корректность введенных координат чтобы исключить возможность обращения к несуществующим данным.
* DataSet merge(final DataSet other, final DoubleBinaryOperator operator) – в качестве аргумента принимает другой экземпляр класс DataSet и применяет функцию operator попарно к каждым двум элементам массивов данных если размерности массивов данных совпадают.

**3.1.2** Как было сказано выше, ключевым элементом в нейронной сети является слой нейронов. Данные попадают на вход слоя и после обработки переходят на следующий слой либо на выход сети (если этот слой оказался последним) В общем случае слой обладает такими характеристиками, как:

* размерность входных и выходных данных;
* набор весов (параметров) слоя, которые изменяются во время обучения сети и с помощью которых происходит преобразование данных (в сверточных и полносвязных слоях);
* функция активации, которая применяется к данным на выходе из слоя.

Базовая структура слоя описана в классе NetLayer. Все существующие типы слоев так или иначе являются наследниками этого класса. Класс является абстрактным, так как некоторые методы должны быть определены в классах наследниках.

public abstract class NetLayer {  
  
 final LayerDimensions layerDimensions;  
 final DataSet deltas;  
 final DataSet weights;  
 final DataSet prevOutputs;  
 final DataSet selfOutputs;  
 final ActivationFunction activationFunction;  
  
 abstract void forward(final DataSet dataSet);  
  
 abstract void backward(final DataSet deltas, final DataSet childrenWeights);  
  
 abstract void lastLayerBackward(final DataSet deltas, final DataSet y, final DataSet outputs);  
}

Рассмотрим поля и методы этого класса:

* LayerDimensions layerDimensions – входные и выходные размерности для слоя. Определяют то, в каком формате данные должны подаваться на вход и выход слоя;
* DataSet deltas – массив ошибок в нейронах слоя. Используется во время работы алгоритма градиентного спуска;
* DataSet weights – параметры слоя, которые используются для преобразования данных;
* DataSet prevOutputs – сохраненные значения выходов предыдущего слоя. Используются во время работы алгоритма градиентного спуска;
* DataSet selfOutputs – Собственные выходы слоя (до применения фунции активации). Используются во время работы алгоритма градиентного спуска;
* ActivationFunction activationFunction – функция активации слоя. Во время работы применяется к каждому выходу слоя.

Методы класса:

* abstract void forward(final DataSet dataSet) – данный метод объявлен абстрактным и должен быть переопределен в классе-наследнике. Метод forward вызывается по очереди для каждого слоя сети от первого к последнему. Именно в этом методе происходит преобразование данных, специфичное для данного типа слоя. В качестве аргумента методу передается ссылка на объект DataSet, который является массивом данных обрабатываемым в данный момент нейронной сетью. Каждый из различных типов слоев нейронной сети обладает собственной логикой преобразования данных;
* abstract void backward(final DataSet deltas, final DataSet childrenWeights) – метод обратного прохода вызывается во время обучения сети поочередно для каждого слоя сети, начиная от последнего. Во время работы этого метода происходит вычисление ошибок нейронов для данного слоя, а также изменение параметров слоя. В качестве аргументов в метод передается массив ошибок предыдущего слоя, а также параметры предыдущего слоя.

На данный момент в системе присутствует реализация следующих

типов слоев нейронной сети:

* полносвязный слой (FullyConnectedNetLayer);
* сверточный слой (ConvolutionNetLayer);
* softmax-слой (SoftmaxNetLayer);
* слой пулинга (PoolingNetLayer).

Все вышеперечисленные классы являются наследниками класса NetLayer и отличаются лишь реализацией методов forward() и backward(). Рассмотрим подробнее принцип работы каждого типа слоя.

**3.1.3** Полносвязный слой является самым известным и часто используемым типом слоя в нейронных сетях. В этом слое выход каждого нейрона предыдущего слоя поступает на вход всех нейронов текущего слоя. Размерность массива параметров слоя равна , где m – количество нейронов предыдущего слоя, n – количество нейронов текущего слоя. Имея реализацию для слоев только данного типа возможно построение простых нейронных сетей, например, многослойных персептронов.

При прохождении через полносвязный слой происходит матричное умножение данных, попадающих на вход слоя на матрицу параметров (весов). После этого к данным применяется функция активации и результат передается на вход следующего слоя. Математически это преобразование описывается формулой (3.1):

|  |  |
| --- | --- |
|  | (3.1) |

где, al – выход текущего слоя;

al-1 – выход предыдущего слоя;

wl – матрица параметров слоя;

σ – функция активации слоя;

bl – смещение слоя.

**3.1.4** Сверточный слой используется для построения сверточных нейронных сетей -   архитектуры [искусственных нейронных сетей](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BA%D1%83%D1%81%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C), нацеленная на эффективное [распознавание изображений](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D1%80%D0%B0%D1%81%D0%BF%D0%BE%D0%B7%D0%BD%D0%B0%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2), входящей в состав технологий [глубинного обучения](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D1%83%D0%B1%D0%B8%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Главным отличием сверточного слоя от полносвязного является то, что не все нейроны предыдущего слоя связаны с нейронами сверточного слоя. Параметры слоя представляют собой ядро свертки, с помощью которого (см. рисунок 3.1) из входных данных мы получаем карту признаков, которую определяет данный слой.

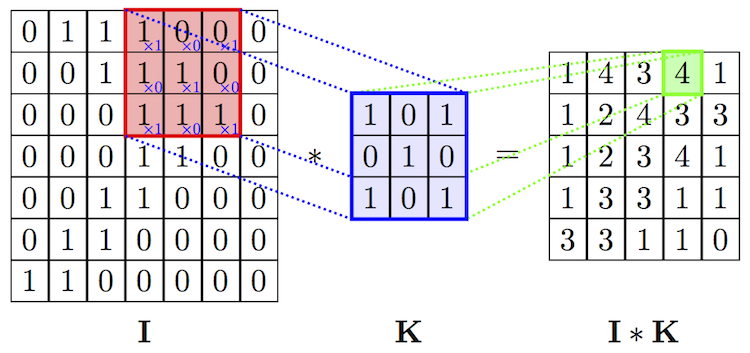


Рисунок 3.1 – операция двумерной свертки

Работа сверточных слоев обычно интерпретируется как переход от конкретных особенностей входного сигнала (изображения) к более абстрактным деталям, и далее к ещё более абстрактным деталям вплоть до выделения понятий высокого уровня. При этом в процессе обучения сверточный слой самонастраивается и вырабатывает сам необходимую иерархию абстрактных признаков (карт признаков), фильтруя маловажные детали и выделяя существенное. Ниже представлена реализация метода forward() для сверточного слоя.

@Override  
void forward(DataSet dataSet) {  
 this.prevOutputs.update(dataSet);  
 Validate.isTrue(dataSet.getDimension().equals(getLayerDimensions().getInputDimension()),  
 "DataSet must match input dimension");  
 dataSet.update(mathOperations.convolve(weights, dataSet));  
 this.selfOutputs.update(dataSet);  
 dataSet.update(this.activationFunction.getForwardOperator());  
}

Стоит обратить внимание на то, что после операции свертки происходит сохранение результата в поле selfOutputs слоя. Эти данные необходимы для работы алгоритма обратного распространения ошибки с помощью которого происходит обучение сети.

**3.1.5** В сверточных нейронных сетях между слоями свертки присутствуют так называемые слои субдискретизации, целью которых является уплотнение карты признаков, полученной от предыдущего слоя свертки. Эти слои не обладают параметрами, которые изменяются в процессе обучения. Операция субдискретизации выполняет уменьшение размерности сформированных карт признаков. В архитектуре сверточной сети считается, что информация о факте наличия искомого признака важнее точного знания его координат, поэтому из нескольких соседних нейронов карты признаков выбирается максимальный и принимается за один нейрон уплотнённой карты признаков меньшей размерности. За счёт данной операции, помимо ускорения дальнейших вычислений, сеть становится более инвариантной к масштабу входного изображения. Чаще всего для субдискретизации применяется функция максимума, когда из определенной части входного сигнала (например, квадрата 2 на 2) выбирается элемент с максимальным значением, который и передается на выход (см. рисунок 3.2). Кроме субдискретизации с функцией максимума можно использовать и другие функции — например, среднего значения или L2-нормирования. Клиент может создать слой субдискретизации с собственной функцией путем наследования от класса PoolingNetLayer.

Однако стоит помнить о том, что во время обучения сети происходит обратный проход – то есть движение по сети от последнего слоя к первому. Поэтому слои субдискретизации должны обладать возможностью восстановить состояние сигнала до прохождения слоя. Например, при использовании max-пулинга сохраняется индекс элемента, который оказался максимальным в каждой части входного сигнала.

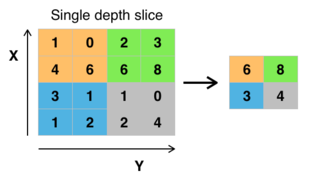


Рисунок 3.2 – операция субдискретизации