ярлык. Допустим, первое обучающее изображение — это цифра 2. Ярлыком изображения будет [0 0 1 0 0 0 0 0 0 0]. Функция потери может быть выражена по-разному. В нашем случае применяется среднеквадратическая ошибка, это 0.5 умножить на (реальность — предсказание) в квадрате.

Примем этот параметр за переменную *L*. Как можно догадаться, потеря будет иметь очень высокое значение для первых двух обучающих образов. Теперь давайте подумаем об этом интуитивно. Мы хотим добиться того, чтобы предсказанный класс был таким же, как класс обучающего изображения. Чтобы добиться этого, нужно привести к минимуму количество потерь, которое имеется.

Скорость обучения — это параметр, который выбирается при разработке сети. Высокая скорость обучения означает, что в обновлениях весов выполнялись большие шаги, поэтому образу может понадобиться меньше времени, для того что бы набрался верный набор весов. Но слишком высокая скорость обучения может привести к очень крупным и неточным шагам, которые могут помешают набрать оптимальные показатели. Процесс прямого распространения, функцию потери, обратное распространение и обновление весов, обычно называют одной эпохой. Программа будет повторять эпоху определенное количество раз для каждого обучающего образа. После того, как завершится обновление свойств на последнем обучающем образце, сеть предположительно должна быть достаточно хорошо обучена и веса слоёв настроены верно.

Для построения модели нейронной сети было использована библиотека Tensor Flow. Было протестировано на базе данных MNIST. Результат зависел от количества эпох для обучения нейронной сети. К примеру, если использовать 10 эпох, то процент распознавания будет 60%. С увеличением количества эпох до 100 к примеру процент распознавания поднялось до 85%. Так же стоит учитывать количество классов для обучения. Чем больше классов, тем меньше процент распознавания.

Список использованных источников

1. Шапиро, Л. Компьютерное зрение / Л. Шапиро, Дж. Стокман. – М. // БИНОМ. Лаборатория знаний, 2006. – 762 с.
2. Гонсалес, Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. // М. : Техносфера, 2005. – 1073 с.

ПРИНЦИПЫ РАБОТЫ ДИНАМИЧЕСКОЙ ДИСПЕТЧЕРИЗАЦИИ МЕТОДОВ В РАЗЛИЧНЫХ ЯЗЫКАХ ПРОГРАММИРОВАНИЯ

Ганнусенко Н.Н.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Иванов Н.Н. – к.ф.-м.н., доцент

При разработке программного обеспечения можно использовать различные парадигмы программирования. И часто для их применения используются полиморфные объекты – это объекты, которые меняют своё поведение в зависимости от обстоятельств. И динамическая диспетчеризация методов – один из способов употребления таких объектов. Динамическая диспетчеризация методов – это механизм выбора реализации полиморфной операции на этапе времени выполнения программы. А механизмы их реализации уже представлены по-разному в каждом из языков программирования.

В ходе изучения способов реализации динамической диспетчеризации методов в разных языках программирования эти механизмы можно разделить на подходы:

- применение виртуальных таблиц методов;
- применение толстых указателей;
- обращение через таблицы поиска;
- поиск посредством пересылки сообщений;
- переходы через указатели на функции.

В таких языках программирования, как C++, Java, C#, Swift, используются виртуальные таблицы методов. Хоть их реализации и различаются в зависимости от наличия множественного наследования [1], от наличия интерфейсов [2] или протоколов [3, 4], но у реализаций есть общее – выделение вызовов динамического поведения в отдельные виртуальные таблицы методов [5], а доступ к ним осуществляется посредством обращения к указателям на связанные таблицы, ссылки которых хранятся у каждого объекта класса. Пример виртуальной таблицы методов представлен на рисунке 1.

В таких языках программирования, как Rust или Go, используется концепция толстых указателей. Принцип её работы схож с подходом использования виртуальных таблиц методов, но имеется разница в том, где конкретно хранится указатель на виртуальную таблицу методов. В толстых указателях ссылки на соответствующие виртуальные таблицы методов хранятся вместе

указателем на сам объект, а не внутри его данных [6, 7]. Пример толстого указателя представлен на рисунке 2.

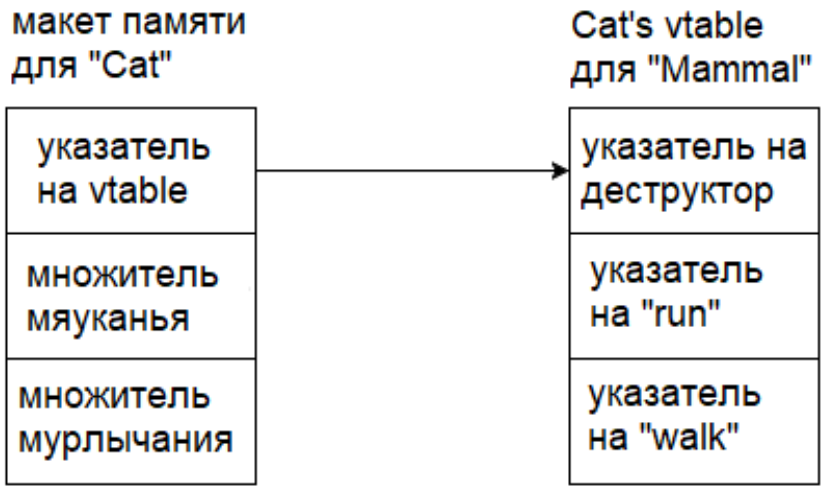

Рисунок 1 – Пример виртуальной таблицы методов для “Cat”

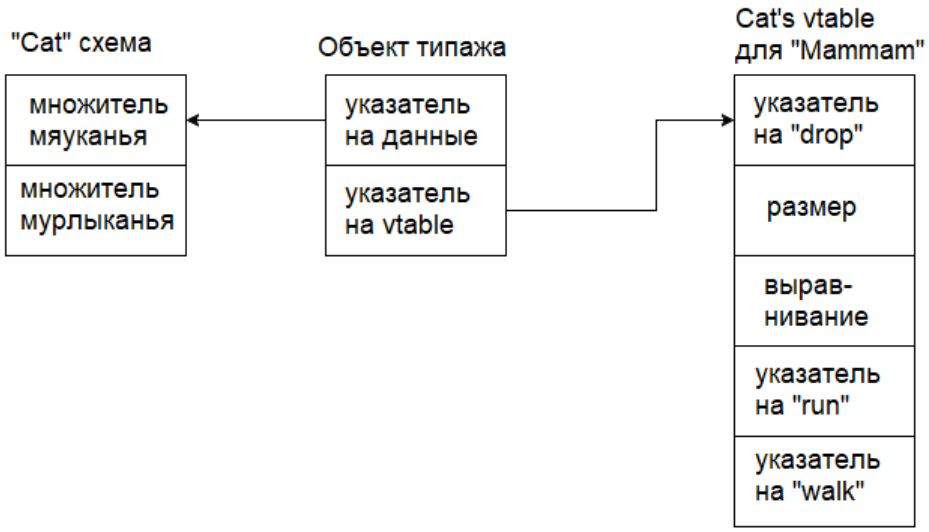

Рисунок 2 – Пример толстого указателя для “Cat”

В интерпретируемых языках программирования, таких как Python, Ruby, Javascript, разрешается поиск методов посредством обращения к ним через таблицы поиска [8]. При таком подходе в каждом объекте хранится ссылка на свою таблицу поиска методов и поиск конкретной реализации происходит через обход таблицы поиска в текущем объекте и в иных связанных местах в зависимости от определенного порядка разрешения методов в конкретном языке программирования.

В языке программирования Objective-C, а также в рамках обратной совместимости и в Swift, методы разрешаются посредством механизма пересылки сообщений [9]. В реализации этого механизма в данных языках программирования в каждом объекте хранится ссылка на метаданные, в которых есть указатель на все реализованные методы. И поиск метода происходит путем конвертирования имени методов и соответствующие индексы методов класса, по которым потом идет обращение в список методов метакласса объекта. Пример заполнения методов метаклассов при использовании механизма пересылки сообщений представлен на рисунке 3.

В языке программирования C также имеется механизм динамической диспетчеризации методов, и достигается он посредством манипулирования указателями на функции [10]. Для того, чтобы этот механизм использовать, определяется структура данных со всеми связанными указателями на функции и далее заполняется ссылками на необходимые указатели на функции в зависимости от требуемого поведения в конкретной задаче.

Таким образом, были выделены различные подходы к реализации механизма динамической диспетчеризации методов, а также кратко рассмотрены их конкретные реализации в различных языках программирования.

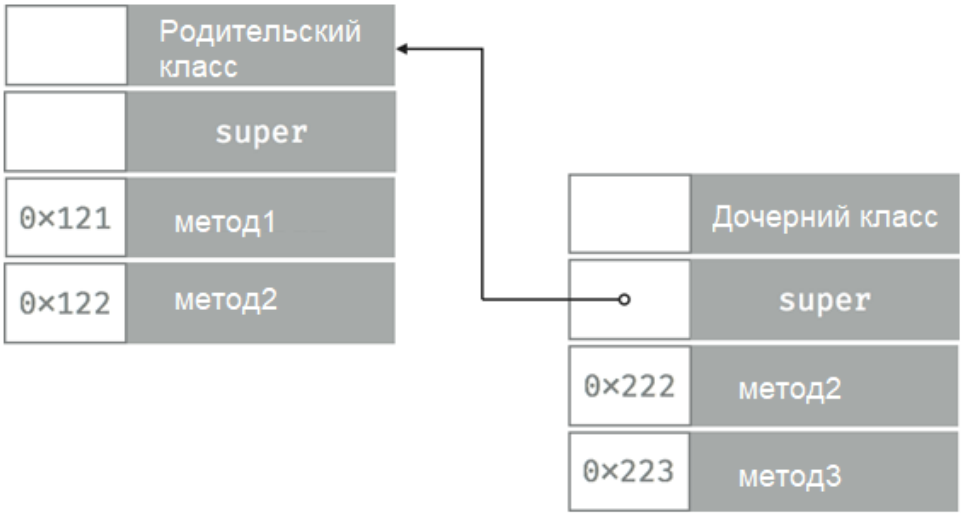

Рисунок 3 – Пример реализации механизма пересылки сообщений

Список использованных источников:

1. Understanding Virtual Tables in C++ [Электронный ресурс]. – Режим доступа: https://pabloariasal.github.io/2017/06/10/understanding-virtual-tables/ – Дата доступа: 19.03.2019.
2. The Black Magic of (Java) Method Dispatch [Электронный ресурс]. – Режим доступа: https://shipilev.net/blog/2015/black-magic-method-dispatch/ – Дата доступа: 19.03.2019.
3. Method Dispatch in Swift [Электронный ресурс]. – Режим доступа: https://www.raizlabs.com/dev/2016/12/swift-method-dispatch/ – Дата доступа: 20.03.2019.
4. Secrets of Swift's Speed [Электронный ресурс]. – Режим доступа: https://www.mikeash.com/pyblog/friday-qa-2014-07-04-secrets-of-swifts-speed.html – Дата доступа: 20.03.2019.
5. Interface dispatch [Электронный ресурс]. – Режим доступа: https://lukasatkinson.de/2018/interface-dispatch/ – Дата доступа: 20.03.2019.
6. Exploring Dynamic Dispatch in Rust [Электронный ресурс]. – Режим доступа: https://alschwalm.com/blog/static/2017/03/07/exploring-dynamic-dispatch-in-rust/ – Дата доступа: 21.03.2019.
7. Блэнди Дж., Орендорф Дж., Программирование на языке Rust / пер. с анг. А. А. Слинкина. – М.: ДМК, Пресс, 2018. – 550 с.: ил, стр. 227 – 228.
8. Method resolution order in Python Inheritance [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/method-resolution-order-in-python-inheritance/ – Дата доступа: 21.03.2019.
9. Objective-C vs Swift messages dispatch [Электронный ресурс]. – Режим доступа: https://blog.untitledkingdom.com/objective-c-vs-swift-messages-dispatch-9d5b7fd58327 – Дата доступа: 21.03.2019.
10. Мартин Р., Чистая архитектура. Искусство разработки программного обеспечения. – СПб.: Питер, 2018. – 352 с.: ил., стр. 60 – 63.

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ ХРАНЕНИЯ ОБЪЕКТОВ АВТОРСКИХ ПРАВ НА ОСНОВЕ ТЕХНОЛОГИИ БЛОКЧЕЙН

Глоба А.А.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Ганжа В.А. – к.физ.-мат.н.,доцент

Разрабатываемая система служит для хранения объектов авторских прав с использованием технологии блокчейн. Блокчейн в общем виде – это выстроенная в соответствии с определенными правилами цепочка блоков, содержащих какую-либо информацию. Обычно копии цепочек хранятся распределённо на разных устройствах в пределах одной сети. С этой базы данных невозможно ничего удалить или провести замену/подмену блока, доступно только чтение.

В данный момент разрабатываемая БД представляет собой структуру, представленную на рис.1:

В ходе разработки системы для хранения авторских прав, было принято решение разделить БД на несколько цепочек, где каждому типу данных (текстовые, аудио-, изображения) соответствовала своя цепочка. Более того, на каждую отдельную цепочку накладывались свои правила для прохождения валидации. Такой подход обеспечивает гибкость и относительно легкую