Санкт–Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Минкашев Талгат Наильевич

Создание инструментов для машинного обучения на MPL

Отчет по учебной практике

Научный руководитель:
к.т.н., доцент Т. А. Брыксин

Консультант:
технический директор ООО «ГравиЛинк» Р. С. Зубаревич

Санкт-Петербург
2021 г.

Содержание

Введение

С развитием искусственного интеллекта возникает все больше возможностей для автоматизации всевозможных процессов, ранее выполняемых человеком. Так, например, для решения задачи имперсонации, то есть эмуляции поведения конкретного человека в интернете, была разработана специальная интеллектуальная система под названием Lei, нашедшая свое применение в сфере IT-безопасности. Изначально система была написана на C++ и содержала большое количество шаблонизированного кода, который на C++ оказалось трудоемко поддерживать. Впоследствии шаблонизированное ядро было переписано на язык программирования MPL и многократно ускорено. MPL — это развивающийся язык общего назначения. Именно после этого были проведены исследования и выяснилось, что MPL действительно порождает более быстрый код по сравнению с C++ благодаря лучшей утилизации известной во время компиляции информации о коде. После этого Lei был переписан на MPL целиком.

На данный момент интеллектуальной системе необходимо умение определять контекст изображений, чтобы эффективно строить базу знаний. Из-за этого потребовалось реализовать набор инструментов, с помощью которых можно было бы создавать модели для классификации изображений.

Одними из наиболее эффективных систем, решающих задачу классификации изображений являются сверточные нейронные сети [1]. Искусственные нейронные сети представляют собой структуру, состоящую из искусственных нейронов, некоторым образом связанных друг с другом и входными данными с помощью связей, каждая из которых имеет определенный коэффициент, умножаемый на поступающее через него значение — вес. Нейронные сети приобретают необходимые свойства в процессе обучения, который, в случае с классификацией изображений, заключается в минимизации функции потерь на обучающей выборке уже размеченных изображений.

Функция потерь — функция, которая характеризует потери при принятии сетью неверного решения на основе входных данных.

Оптимизация, как правило, происходит благодаря вычислению част-

ных производных функции потерь относительно весов сети и последующего их изменения для приближения к локальному минимуму функции потерь.

Свертка выполняет фильтрацию значений пикселов в изображении. В частности, поэтапная свертка позволяет выделять определенные признаки изображения, уменьшая количество данных, подаваемых на вход нейронной сети, сохраняя при этом информацию о структуре исходного изображения.

Таким образом, сверточная нейронная сеть может быть обучена на большом наборе размеченных изображений и впоследствии использована для классификации новых изображений, используя найденные во время обучения закономерности, выраженные в значениях весов сети. Именно такая функциональность требуется для Lei.

Постановка цели и задач

Целью данной работы является реализация части инструментов для создания нейронных сетей на MPL с более читаемой структурой, компактным и модифицируем кодом и меньшим количеством внешних зависимостей.

В рамках данной работы были поставлены следующие задачи:

1) изучить MPL и испытать его на новом классе задач;
2) проанализировать существующие решения, выделить основные концепции;
3) реализовать структуру для хранения данных, используемых нейронной сетью;
4) реализовать преобразование изображений в удобную для взаимодействия с нейронной сетью структуру;
5) спроектировать модуль автоматического дифференцирования — набор механизмов, позволяющих автоматически вычислять частные производные функций и реализовать построение графа вычислений дифференцируемой функции.

Глава 1. Обзор

Далее представлен обзор аналогов и распространенных методик для решения поставленных задач.

1.1 Аналоги

Для решения задач машинного обучения существует широкий выбор инструментов. Одними из наиболее популярных в индустрии являются TensorFlow [2] и PyTorch [3] — большие, комплексные многофункциональные фреймворки, предоставляющие готовые реализации самых распространенных алгоритмов машинного обучения. Общим подходом данных библиотек является наличие простого, понятного API на таких языках, как, например, Python, и реализация большей части ресурсоёмких вычислений на языках C, C++ и CUDA с использованием производительных библиотек — к примеру, Eigen [4] в случае с TensorFlow. Такой подход обеспечивает не только быстродействие, но и простоту использования для конечного пользователя. Неоспоримыми плюсами данных платформ является их доступность и универсальность. Однако, такой широкий набор возможностей неизбежно влечет частичную потерю в производительности, множество зависимостей и усложнение поддержания кода ввиду его сложности.

Существуют также и более специализированные инструменты. Например, библиотека компьютерного зрения OpenCV [5] вместе с модулем DNN, предназначенного для работы с нейросетями, или OpenNN [6] — библиотека, реализующая нейронные сети. Данные библиотеки написаны на языке C++, хорошо оптимизированы и опережают конкурентов в решении задач, на которых специализируются [7]. Однако, OpenCV не поддерживает возможности обучения моделей, а только возможность запуска моделей из некоторых популярных фреймворков. OpenNN предоставляет набор готовых классов для создания нейронных сетей, но не обеспечивает инструментов для создания своих алгоритмов.

Таким образом, разработка нового решения на MPL позволит получить более легковесное решение, с простым кодом и потенциалом для оптимизации и реализации гибридных алгоритмов.

1.2 Тензоры

Широко распространенной структурой данных, используемой при решении задач машинного обучения, является тензор [8]. Тензоры — это алгебраические объекты, заданные на векторном пространстве конечной размерности, которые обобщают скаляры, векторы и матрицы, представляя собой многомерные структуры. Современные задачи машинного обучения предполагают многомерность данных, что и мотивирует использование тензоров для хранения и обработки информации. TensorFlow, PyTorch и многие другие подобные библиотеки используют тензоры для представления данных.

1.3 Автоматическое дифференцирование

Производные, в основном в виде градиентов и, реже, гессианов, повсеместно используются в машинном обучении, являясь основой многих алгоритмов оптимизации. Автоматическое дифференцирование в смысле алгоритмического дифференцирования представляет собой совокупность методик для эффективного и точного вычисления производных функций, представленных в виде компьютерных программ. Автоматическое дифференцирование можно рассматривать как частный случай семантической трансформации: необходимо из кода исходной функции получить код, вычисляющий производные этой функции. Причем дифференцирование должно быть быстрым, точным, масштабируемым и удобным в использовании. Как правило, для решения этой задачи конструируется граф вычислений дифференцируемой функции, состоящий из примитивных операций, для которых алгоритм вычисления производных известен и задан программно. Впоследствии выполняется обход графа вычислений с поочередным применением цепного правила [9] для вычисления производных. Распространены два способа реализации данной идеи: с помощью перегрузки операторов и преобразования исходного кода. Tensorflow использует преобразование исходного кода и требует явного задания графа вычислений с помощью модуля GradientTape [10], PyTorch использует перегрузку операторов и конструирует граф вычислений неявно. Подробнее об автоматическом диффе-

ренцировании: [11], [12].

1.4 Использованные инструменты

Для реализации был использован язык программирования MPL, так как его лучшее освоение является одной из задач данной работы. Для работы с изображениями была использована популярная библиотека `stb_image` [13]. Данная библиотека имеет удобный интерфейс и просто интегрируется в проект.

Глава 2. Реализация

Для удобства реализации и использования, библиотека была разделена на две части: mlLibrary — непосредственно библиотеку, содержащую основную функциональность, и mlTester — приложение со всеми дополнительными инструментами для тестирования. Далее представлен обзор компонентов.

2.1 Реализация тензоров

В mlLibrary тензор представляется параметризуемым по типу хранимых данных классом Tensor.

Внутреннее хранение информации в Tensor реализовано с помощью обычного одномерного массива, а индексирование обеспечивается благодаря массивам blocks и shape, где shape представляет собой форму тензора, а blocks для каждого измерения содержит произведение размеров всех следующих за ним измерений и единицу для последнего измерения. Таким образом, используя blocks и shape можно из мультииндекса получать индекс элемента во внутреннем одномерном массиве. Более того, такая структура позволяет производить операции транспонирования, сечения и изменения формы за константное время. Для этого изменяется не внутренний массив, а только массивы blocks и shape, что приводит к изменению индексации и обеспечивает необходимый результат. Для работы с тензорами были также реализованы:

- поэлементные операции;
- преобразование тензора в строку;
- получение следующего мультииндекса из исходного;
- получение информации о тензоре: его ранга, количества элементов и прочей.

Таким образом, были реализованы наиболее часто используемые операции и было обеспечено их быстродействие.

2.2 Модуль imageLoader

Одним из наиболее широко распространенных видов информации являются изображения всевозможных форматов. Для преобразования изображений в тензор был реализован модуль imageLoader, использующий библиотеку `stbi_image`. В модуле были реализованы:

- структура Image, описывающая характеристики изображения: высоту, ширину, количество каналов и массив, содержащий значения пикселов;
- функция loadImage, оборачивающая `stbi_load_image`, для загрузки изображения из файла по указанному пути;
- функцию преобразования Image в Tensor.

2.3 Построение графа вычислений

В mlLibrary для автоматического дифференцирования был выбран метод перегрузки операторов, так как это более простой в реализации и дальнейшей эксплуатации метод. Для вычисления градиентов функций был выбран метод обратного распространения ошибки, подразумевающий обход графа вычислений в направлении от выходных значений к входным, так как данный метод является более производительным при большом количестве входных аргументов дифференцируемой функции и небольшом количестве выходных значений по сравнению с методом построения градиента во время прямого обхода графа вычислений [12]. На данный момент был реализован модуль tracer, предназначенный для построения графа вычислений дифференцируемой функции. Модуль содержит:

- класс Box для представления данных и структура Node для представления вершин в графе вычислений;
- функции для работы с Box и Node;
- функция trace, конструирующая граф вычислений для входной функции;

- перегруженные варианты функций обращения по индексу и сложения для проверки функциональности модуля.

Функция trace принимает в качестве аргументов дифференцируемую функцию, представляющую собой композицию из примитивных функций, имеющих перегруженную версию, принимающую Box, и ее аргумент, относительно которого в дальнейшем будет вычисляться производная. Для построения графа вычислений исходные аргументы оборачиваются в Box, после чего исходная функция вызывается с переданным аргументом уже типа Box, что приводит к вызову перегруженных версий примитивных функций по мере исполнения исходной функции. Перегруженные версии примитивных функций помимо вычислений результатов применения исходных примитивных функций обеспечивают создание графа вычислений, инициализируя новые экземпляры Box.

Представление графа вычислений продемонстрировано на Рис. 1. Все экземпляры класса Box хранятся в отдельном массиве в единственном экземпляре. Каждому Box соответствует Node. Box может содержать как явные значения, так и другой Box. Для обеспечения такой функциональности используется тип Variant, представляющий собой размеченное объединение. Argument же используется для сокращенной записи. Среди допустимых типов Argument отсутствует Box, вместо этого используется структура Index, хранящая индекс необходимого Box в массиве boxes. Такая структура позволяет избежать копирования данных в полях arguments у Node и value у Box, а также обеспечивает возможность вложенных вызовов дифференцирования.

Node содержит всю необходимую информацию для обратного обхода графа вычислений в дальнейшем: аргументы, номера аргументов, относительно которых необходимо вычислить производные, идентификатор функции, вычисленный во время прямого прохода результат и индексы родительских Box, из которых впоследствии извлекаются необходимые Node.

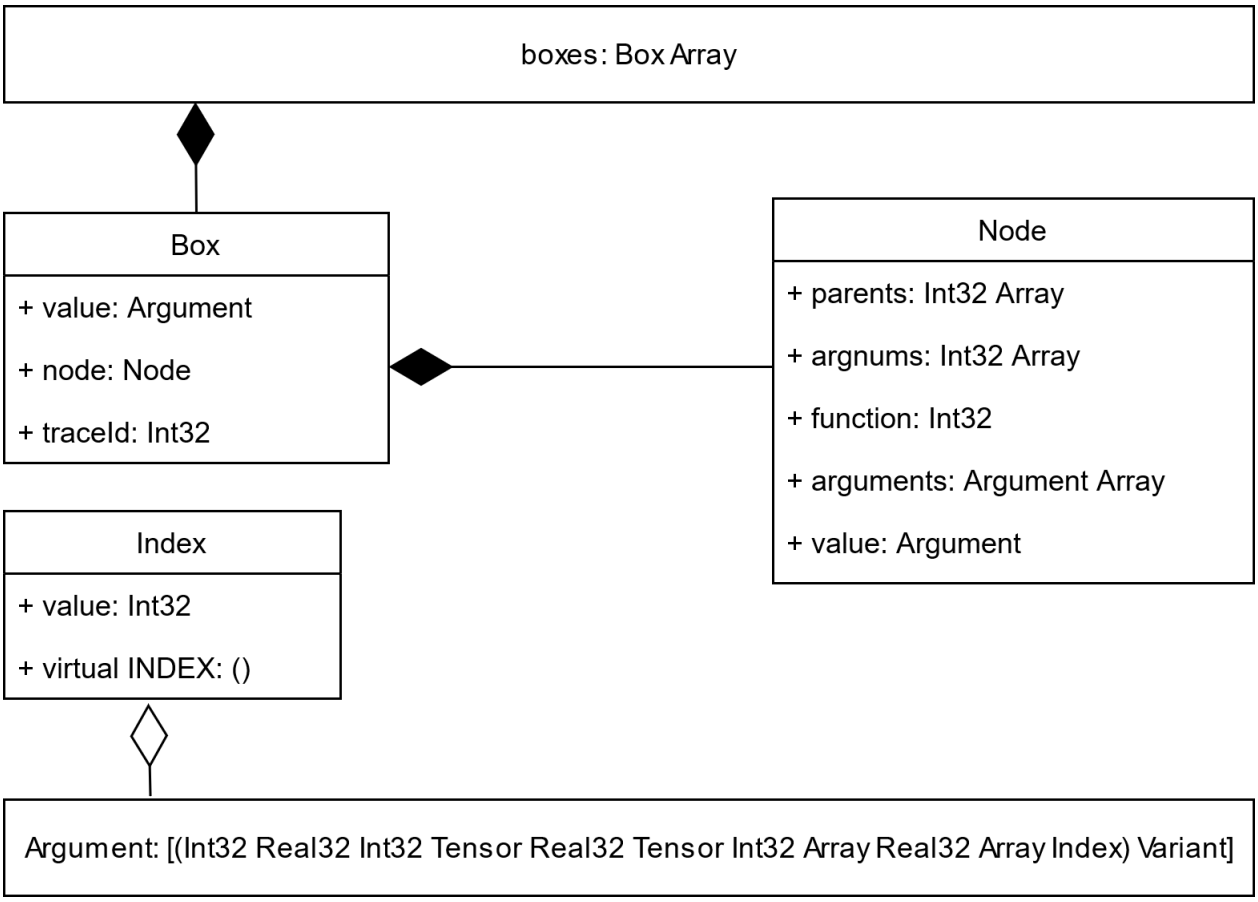

Рис. 1: Устройство графа вычислений.

Глава 3. Тестирование

Вся функциональность класса Tensor была протестирована с помощью юнит-тестов. Модуль tracer был протестирован на нескольких основных сценариях использования. Модуль imageLoader был протестирован вручную, так как функции библиотеки stb уже протестированы.

Заключение

В результате работы были реализованы и протестированы компактные инструменты для создания нейронных сетей на MPL, а именно:

1) класс Tensor для хранения данных;

2) модуль imageLoader для загрузки изображений;

3) модуль tracer для построения графа вычислений функции.

Также были рассмотрены существующие подходы к автоматическому дифференцированию, из которых были выбраны наиболее подходящие для дальнейшего развития библиотеки методики.

Код данной работы является закрытым, так как MPL в любом случае используется только в пределах компании ГравиЛинк и имеет необычный по сравнению с популярными языками программирования синтаксис. Однако по мере развития проекта планируется сделать код проекта откртым.

Список литературы

[1] Zheng Yi. Evaluation and Implementation of Convolutional Neural Networks in Image Recognition // Journal of Physics: Conference Series. — 2018. — Vol. 1087, no. 6.

[2] Репозиторий TensorFlow. — URL: `https://github.com/tensorflow/tensorflow` (online, accessed: 2021-05-15).

[3] Репозиторий PyTorch. — URL: `https://github.com/pytorch/pytorch` (online, accessed: 2021-05-15).

[4] Репозиторий Eigen. — URL: `https://gitlab.com/libeigen/eigen` (online, accessed: 2021-05-15).

[5] Репозиторий OpenCV. — URL: `https://github.com/opencv/opencv` (online, accessed: 2021-05-17).

[6] Репозиторий OpenNN. — URL: `https://github.com/Artelnics/opennn` (online, accessed: 2021-05-17).

[7] High performance optimization algorithms for neural networks. — URL: `https://www.opennn.net/files/high_performance_optimization_algorithms_for_neural_networks.pdf` (online, accessed: 2021-06-02).

[8] Introduction to Tensors. — URL: `https://www.tensorflow.org/guide/tensor` (online, accessed: 2021-05-15).

[9] Apostol T. M., «Calculus, 2nd ed., Vol. 1: One-Variable Calculus, with an Introduction to Linear Algebra». Waltham, MA: Blaisdell, pp. 174-179, 1967.

[10] tf.GradientTape. — URL: `https://www.tensorflow.org/api_docs/python/tf/GradientTape` (online, accessed: 2021-05-17).

[11] Atilim Baydin, Barak Pearlmutter, Alexey Radul et al. Automatic differentiation in machine learning: A survey // Journal of Machine Learning Research. — 2018. — Vol. 18, no. 1. — P. 1-43.

[12] Christian Bischof, Martin Bucker. Computing Derivatives of Computer Programs // NIC Series. — 2000. — Vol. 3. — P. 315-327.

[13] stb. — URL: https://github.com/nothings/stb/blob/master/stb_image.h (online, accessed: 2021-05-03).