УДК 519.688, 004.8

ИНСТРУМЕНТЫ PYTHON ДЛЯ РАБОТЫ С БАТЧАМИ

Мантусов Владимир Анатольевич
mantab@yandex.ru
Студент 3 курса образовательной программы «Документоведение»
ФГОУ ВО Калмыцкий государственный универсистет имени Б. Б. Городовикова
г.Элиста, Российская Федерация
Научный руководитель – **Мантусов А. Б.**
к.п.н., доцент

Ключевые слова: пакетное умножение матриц, батч, einsum, NumPy, PyTorch, TensorFlow, производительность, GPU.

Введение

Современные задачи машинного обучения, компьютерного зрения и научных вычислений часто требуют обработки больших объёмов данных с высокой скоростью. Одной из ключевых операций в этих областях является **матричное умножение**, лежащее в основе многих алгоритмов — от линейных преобразований до сложных нейронных сетей. Однако при работе с огромными наборами данных (например, батчами изображений или временными рядами) поочередное перемножение отдельных матриц становится неэффективным.

Пакетное (батчевое) умножение матриц решает эту проблему, позволяя выполнять массовые вычисления за один вызов оптимизированных функций. Этот подход обеспечивает:

- **Высокую производительность** за счёт параллелизации вычислений на CPU/GPU.
- **Удобство работы с тензорами** в библиотеках, таких как NumPy, PyTorch и TensorFlow.
- **Эффективную обработку данных** в нейронных сетях, где операции часто выполняются над мини-батчами.

Материал рассчитан на читателей, знакомых с основами линейной алгебры и Python, но стремящихся глубже разобраться в оптимизации матричных операций для задач Data Science и ML.

Основная часть

Батч (от англ. "batch" — пакет, партия) — это группа нескольких матриц, объединённых вместе для одновременной обработки. Это фундаментальное понятие в вычислительной математике и глубоком обучении.

В терминах размерностей можно дать следующее определение:

если одиночная матрица: представляет собой 2D массив (m, n), то батч матриц является 3D массивом (batch_size, m, n), где:

batch_size — количество матриц в пакете

m, n — строки и столбцы каждой матрицы

3. Пример батчей

Батч из 3 матриц 2×3

```
python
Copy
import numpy as np

batch = np.array([
   [[1, 2, 3],  # Первая матрица 2x3
    [4, 5, 6]],

   [[7, 8, 9],  # Вторая матрица 2x3
    [10, 11, 12]],

   [[13, 14, 15],  # Третья матрица 2x3
    [16, 17, 18]]
])
print(batch.shape)  # (3, 2, 3) ← 3 матрицы, каждая 2x3
```

Батчи имеют широкое применение по следующим причинам

1. Эффективность вычислений:

- Современные процессоры и GPU оптимизированы для параллельных операций
- Один вызов функции для батча быстрее, чем многократные вызовы для отдельных элементов

2. Ускорение обучения моделей:

- В машинном обучении градиенты обычно вычисляются для всего батча
- Позволяет лучше использовать ресурсы видеокарт

3. Удобство обработки:

- Данные уже подготовлены для параллельной обработки
- Упрощается работа с библиотеками (TensorFlow, PyTorch и др.)

Батчинг (batch processing) — это метод оптимизации вычислений, при котором данные объединяются в группы (батчи) для одновременной обработки. В отличие от последовательного выполнения операций над отдельными элементами, модель обрабатывает целый пакет данных за один проход, что значительно повышает эффективность использования вычислительных ресурсов [1].

Практическое применение:

Например, при генерации векторных представлений (эмбеддингов) для текстовых данных обработка 100 документов в одном батче выполняется существенно быстрее, чем последовательное вычисление для каждого документа по отдельности. Такой подход позволяет максимально задействовать параллельные вычисления на CPU/GPU и сократить общее время обработки.

Одной из областей, где батчи очень удобны в применении этопакетное умножение матриц (batch matrix multiplication). Пакетное умножение матриц - это операция одновременного умножения нескольких пар матриц, что особенно полезно в машинном обучении и обработке больших данных. Пакетное умножение обычно быстрее, чем последовательное умножение отдельных матриц, так как:

Лучше использует кэш процессора

Позволяет векторизованные операции

Эффективно работает на GPU.[2]

Приведем основные понятия

При обычном умножении матриц ($A \times B = C$):

- A имеет размерность (m, n)
- B имеет размерность (n, p)

- Результат C будет размерности (m, p)

При пакетном умножении мы работаем с "батчами" (пакетами) матриц:

- A имеет размерность (batch, m, n)
- B имеет размерность (batch, n, p)
- Результат C будет размерности (batch, m, p)

В Python имеется достаточно гибкий набор инструментов для реализации пакетного умножения матриц, предусматривабщий использование различных пакетов.[3]

1. Использование NumPy

python
Copy

```
import numpy as np

# Создаём батч из 3 матриц 2x3
batch_A = np.array([
    [[1, 2, 3],
     [4, 5, 6]],

    [[7, 8, 9],
     [10, 11, 12]],

    [[13, 14, 15],
     [16, 17, 18]]
])

# Создаём батч из 3 матриц 3x2
batch_B = np.array([
    [[1, 2],
     [3, 4],
     [5, 6]],

    [[7, 8],
     [9, 10],
     [11, 12]],

    [[13, 14],
     [15, 16],
     [17, 18]]
])

# Пакетное умножение
result = np.matmul(batch_A, batch_B)
print(result.shape)  # (3, 2, 2)
```

2. Использование PyTorch

python
Copy

```
import torch

# Создаём тензоры
batch_A = torch.tensor([
    [[1, 2, 3], [4, 5, 6]],
    [[7, 8, 9], [10, 11, 12]]
], dtype=torch.float32)
```

```
batch_B = torch.tensor([
    [[1, 2], [3, 4], [5, 6]],
    [[7, 8], [9, 10], [11, 12]]
], dtype=torch.float32)

# Пакетное умножение
result = torch.bmm(batch_A, batch_B)
print(result)
```

3. Использование TensorFlow

```
python
Copy
import tensorflow as tf

batch_A = tf.constant([
    [[1, 2, 3], [4, 5, 6]],
    [[7, 8, 9], [10, 11, 12]]
], dtype=tf.float32)

batch_B = tf.constant([
    [[1, 2], [3, 4], [5, 6]],
    [[7, 8], [9, 10], [11, 12]]
], dtype=tf.float32)

result = tf.matmul(batch_A, batch_B)
print(result)
```

Использование numpy.einsum (Эйнштейновская нотация)

numpy.einsum - это мощная функция, которая позволяет выражать различные операции над многомерными массивами, включая пакетное умножение матриц, используя компактную Эйнштейновскую нотацию.[4]

Эйнштейновская нотация предполагает:

1. Индексы для осей входных массивов
2. Стрелку -> и индексы для выходного массива
3. Повторяющиеся индексы в входных массивах означают суммирование по этим осям [7]

Для пакетного умножения матриц размерностью (batch, m, n) × (batch, n, p):

```
python
Copy
import numpy as np

batch_A = np.random.rand(10, 3, 4)  # 10 матриц 3x4
batch_B = np.random.rand(10, 4, 5)  # 10 матриц 4x5

# Эйнштейновская нотация: "bij,bjk->bik"
result = np.einsum('bij,bjk->bik', batch_A, batch_B)
print(result.shape)  # (10, 3, 5)
```

Разберём нотацию:

- b - ось батча (сохраняется)
- i,j,k - индексы для строк и столбцов матриц
- bij,bjk->bik означает: суммировать по оси j (последняя ось первой матрицы и первая ось второй матрицы) [5]

Приведем пример того, как пакетная обработка позволяет эффективно применять операции ко многим данным одновременно Пусть у нас есть батч из 16 изображений (16, 64, 64), и мы хотим применить к каждому одно и то же преобразование (фильтр 3x3):

```
python
Copy
images = torch.randn(16, 64, 64)  # 16 изображений 64x64
filters = torch.randn(3, 3)       # один фильтр 3x3

# Чтобы применить фильтр ко всем изображениям, можно использовать
# групповую свёртку или "размножить" фильтр
result = torch.nn.functional.conv2d(
   images.unsqueeze(1),  # добавляем размерность канала (16, 1, 64, 64)
   filters.unsqueeze(0).unsqueeze(0).repeat(16, 1, 1, 1),  # (16, 1, 3, 3)
   groups=16
)
```

Области использования пакетного умножения.

1. **Для обработка мини-батчей в нейронных сетях**: Когда вы подаёте на вход сети несколько образцов одновременно.

```
python
Copy
# Например: 32 образца, каждый с 784 признаками
inputs = torch.randn(32, 784)
# Веса слоя: 784 входа, 256 выходов
weights = torch.randn(784, 256)
# Матричное умножение для всех образцов
outputs = torch.mm(inputs, weights)  # автоматически батчевое
```

2. **Трёхмерные данные**: Например, обработка временных рядов или видеоданных.
3. **Трансформеры и внимание**: В механизмах внимания часто используются пакетные умножения. [6]

Таблица 1 – Сравнение методов

Метод	Преимущества	Недостатки
np.matmul/@	Простота, читаемость	Менее гибкий
np.einsum	Максимальная гибкость	Более сложный синтаксис
torch.bmm	Хорошо оптимизирован для GPU	Только для PyTorch

Заключение

Батч — это:

1. Способ организации данных для эффективной параллельной обработки
2. Основная единица работы в современных ML-фреймворках
3. Техника, позволяющая максимально использовать возможности железа
4. Концепция, которая применяется не только к матрицам, но и к тензорам любой размерности

Python предлагает мощные инструменты для работы с батчами:

1. NumPy:

- Функции np.matmul() и @ для пакетного умножения
- Оптимизированные операции с использованием np.einsum()
- Поддержка broadcasting для автоматического расширения размерностей

2. PyTorch:

- Специализированные функции torch.bmm() и torch.matmul()
- Автоматическое дифференцирование для батчей

- Эффективная работа на GPU через CUDA

3. TensorFlow/Keras:

- Встроенная поддержка батчей в слоях нейронных сетей
- Оптимизированные операции tf.matmul()
- Поддержка распределенных вычислений

4. Специализированные библиотеки:

- CuPy для GPU-ускорения numpy-подобных операций
- JAX для автоматической векторизации и параллелизации
- Dask для работы с большими батчами, не помещающимися в память

Батчи предоставляют значимые преимущества в области программирования, машинного обучения и научных вычислений, а современные Python-инструменты делают работу с ними максимально эффективной и удобной. Ключевые факторы успеха - правильный выбор инструментария и понимание принципов параллельной обработки данных.

Освоение этих технологий позволяет:

- Ускорить вычисления в десятки и сотни раз
- Эффективно использовать ресурсы GPU/TPU
- Масштабировать алгоритмы на большие объемы данных
- Упрощать код, сохраняя его производительность

Список использованной литературы

1. Что такое батчинг и почему это важно в ML Электронный ресурс https://tenchat.ru/media/2976932-chto-takoye-batching-i-pochemu-eto-vazhno-v-ml Дата обращения - 14/03/2025
2. Мантусов А.Б., Мантусов В.А. ОДИН ПРИМЕР ПРИМЕНЕНИЯ БИБЛИОТЕК TENSORFLOW И KERAS ДЛЯ СОЗДАНИЯ И ОБУЧЕНИЯ НЕЙРОННЫХ СЕТЕЙ НА PYTHON ДЛЯ РАСПОЗНАВАНИЯ ЗНАКОВ // В сборнике: Традиции и инновации в науке и образовании в аспекте цифровизации. Всероссийская научно-практическая конференция с международным участием. Элиста, 2023. С. 51-58.
3. Эпоха, батч, итерация - в чем различия? Электронный ресурс https://neurohive.io/ru/osnovy-data-science/jepoha-razmer-batcha-iteracija/Дата обращения - 16/03/2025
4. Мантусов А.Б. ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В НАУЧНЫХ ИССЛЕДОВАНИЯХ -ПРИМЕНЕНИЕ ЯЗЫКА ПРОГРАММИРОВАНИЯ PYTHON // В сборнике: Адаптация человека к вызовам цифровой эпохи. материалы круглого стола. Элиста, 2023. С. 68-70.
5. Захаров Виктор Николаевич, Мунерман Виктор Иосифович Модели и методы параллельной обработки структурированных больших данных // Современные информационные технологии и ИТ-образование. 2014. №10. URL: https://cyberleninka.ru/article/n/modeli-i-metody-parallelnoy-obrabotki-strukturirovannyh-bolshih-dannyh (дата обращения: 20/03/2025).
6. Мантусов А. Б. ОБРАБОТКА ЕСТЕСТВЕННОГО ЯЗЫКА С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА ПРОГРАММИРОВАНИЯ PYTHON // Учебное пособие / Том Часть 1. Элиста, 2022.
7. Understanding Numpy's einsum Электронный

ресурс https://eli.thegreenplace.net/2025/understanding-numpys Дата обращения - 22/03/2025