Математические основы информационной безопасности

Груздев Дмитрий Николаевич

Ускорение обучения моделей

Как можно ускорить обучение

Ускорение вычислений

- распараллеливание вычислений
- специальные вычислители

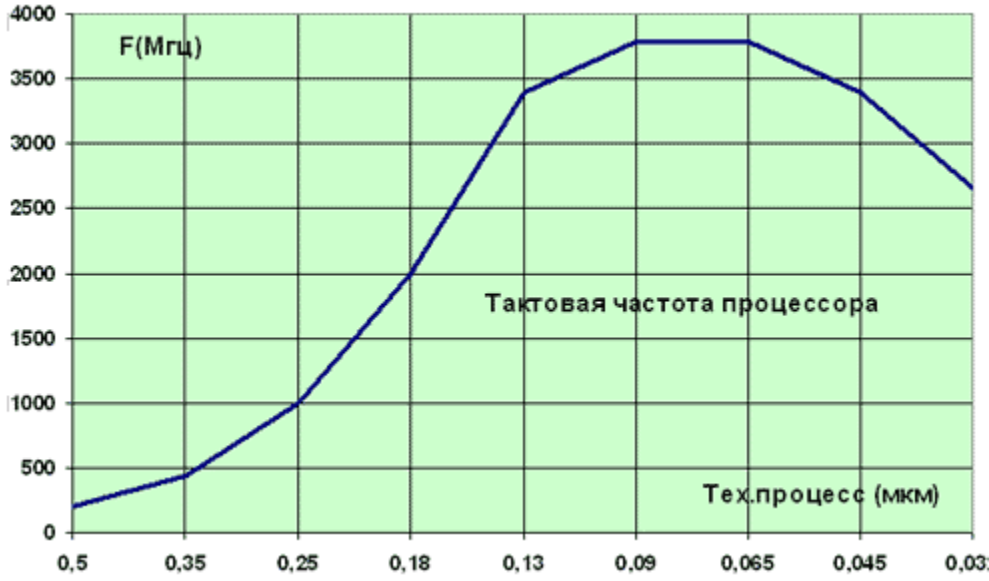
Улучшенные алгоритмы обучения

- стохастический градиент
- метод моментов
- метод Нестерова
- оптимизаторы: adam, adagrad

Распараллеливание вычислений

Типы распараллеливания:

- Несколько ядер и процессоров на одном устройстве.
- Несколько вычислителей, соединенных в сеть.



CPU vs GPU

	CPU	GPU
Название	Central processing unit	Graphics processing unit
Предназначение	любые задачи	обработка графики
Набор команд	расширенный	специализированный
Количество ядер	~ от 2 до 8	~3000
Тактовая частота	2500 – 4000 МГц	900 – 1100 МГц
Вычислительная мощность	~10GFLOP	~100GFLOP
Скорость доступа к памяти	~6 GB/s	~40GB/s

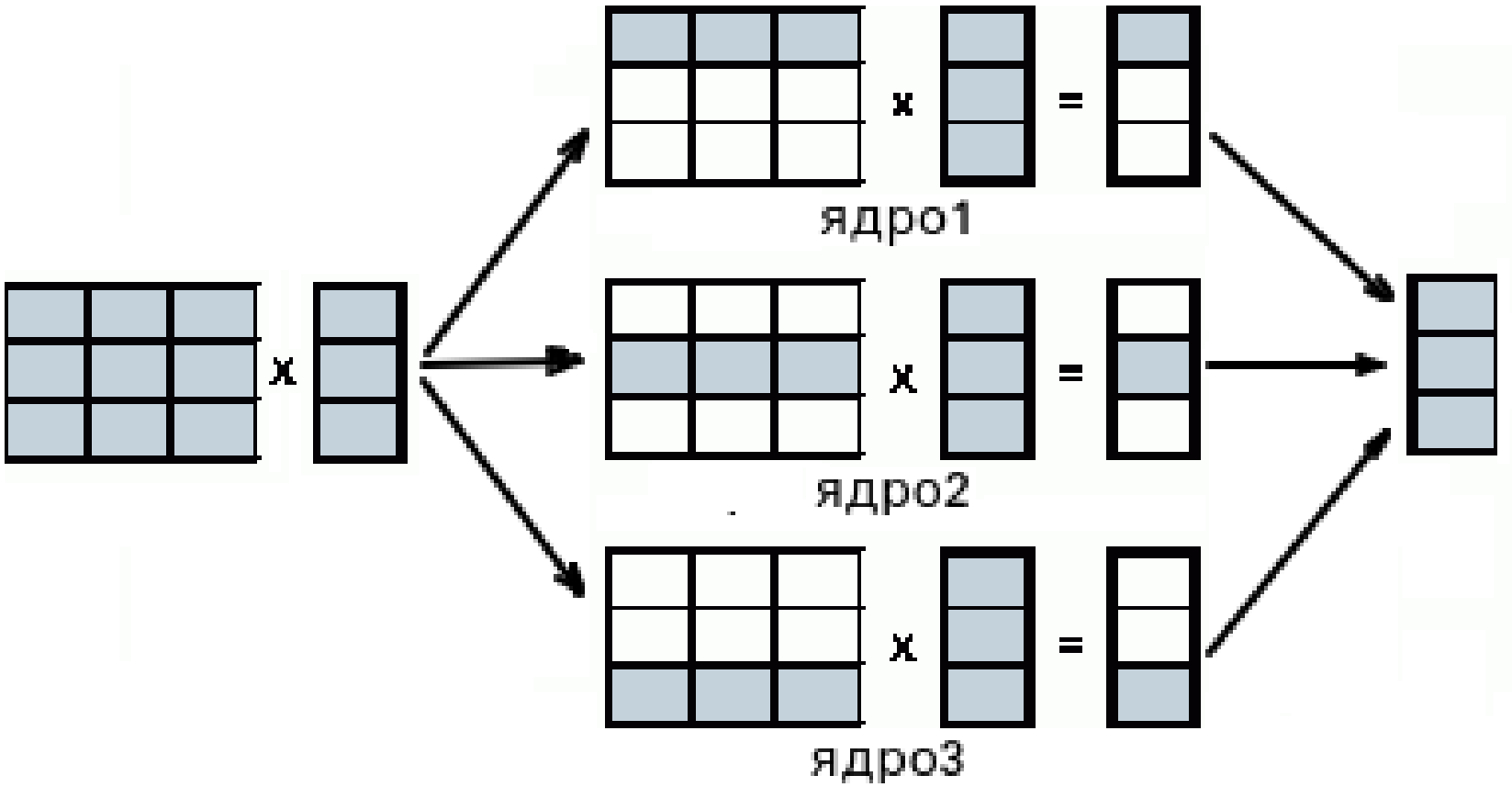
Работа с GPU

Исходная задача разбивается на ряд подзадач, каждая из которых решается абсолютно независимо (т.е. никакого взаимодействия между подзадачами нет) и в произвольном порядке.

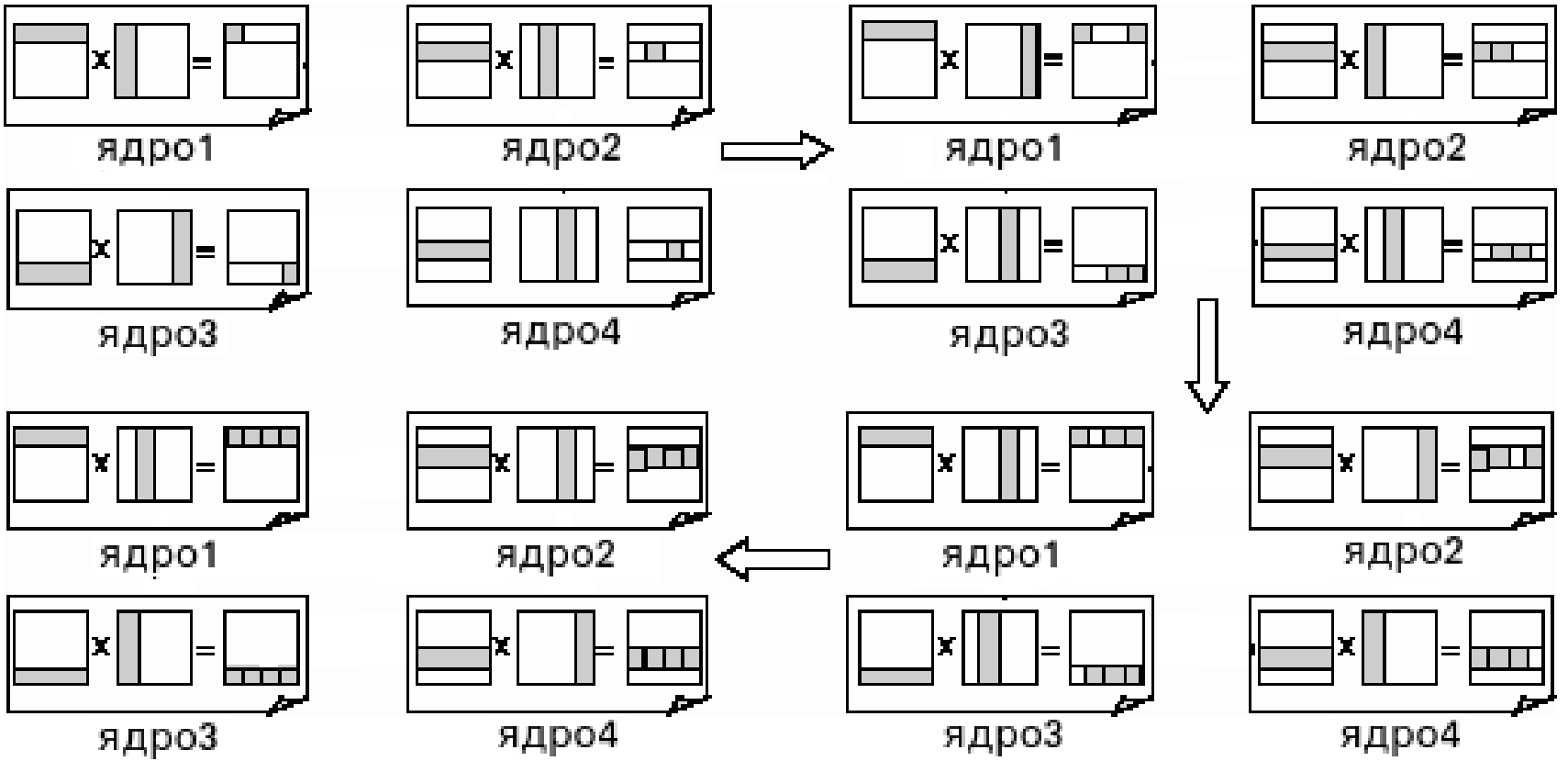
Программные технологии для работы с GPU:

- CUDA – для работы с NVIDEA
- OpenCL – кросплатформенная

Умножение вектора на матрицу



Параллельное умножение матриц



Ленточная схема разделения данных.

Переход к матричным вычислениям

Matlab, tensorflow и другие вычислительные среды не могут распараллелить вычисления значений в цикле.

По возможности необходимо представлять вычисления в матричном виде.

Примеры:

1. Линейная регрессия, метод градиентного спуска:

   for ($j = 1; j \le n; j++$)
   $\Delta\theta_j = -\alpha * \Sigma_{1 \le i \le m}(\langle\theta, x_i\rangle - y_i) * x_i^{(j)}$; - плохо

   $\Delta\theta = X^T(X*\theta - y)$; – хорошо ($X_{mxn}$ – матрица объекты-признаки)

2. Нейронные сети, прямое распространение сигнала:

   for ($j = 1; j \le H_{k+1}; j++$)
   $y_j^{(k+1)} = \sigma_j^{(k+1)}(\Sigma_{0 \le i \le Hk}\Theta_{ij}^{(k)} * y_i^{(k)})$; - плохо

   $y^{(k+1)} = \sigma^{(k+1)}(\Theta^{(k)} * y^{(k)})$; - хорошо ($\Theta^{(k)}$ – матрица весов k-го слоя)

Оптимизация в программных пакетах

	Параллельные вычисления	Работа с GPU
Numpy	-	-
SciPy	-	-
Skikit-learn	+/-	-
PyTorch	+	+
Tensorflow	+	+
Matlab, Octave	+	-

Нейросети на ПЛИС

Программируемая логическая интегральная схема:

- Высокое быстродействие.
- Низкое энергопотребление.
- Подходит только для использования нейросетей (обучение проводится в другом месте).
- Сложность программирования.

Нейросети на ASIC

Application-Specific Integrated Circuit:

- Самая низкая стоимость чипа.
- Самое низкое энергопотреблении.
- Высокая скорость работы.
- Только использование нейросетей (без обучения).
- После изготовления нельзя вносить изменения.
- Сложный и дорогой техпроцесс разработки.

TPU

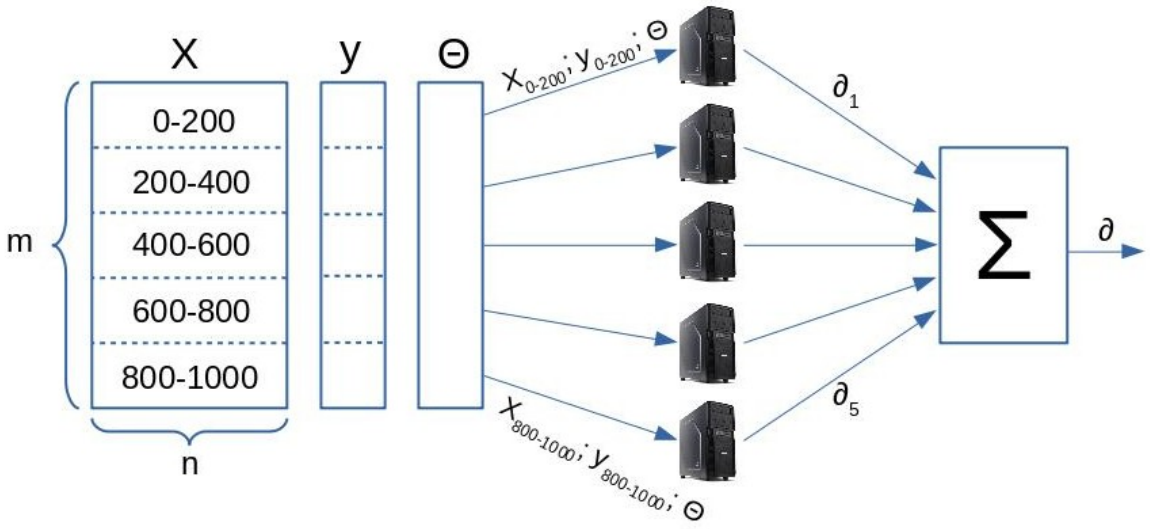
Tensor Processing Unit

- Аппаратное ускорение скорости обучения нейросети.
- Высокая скорость работы нейросети.
- Оптимизированы для работы в кластерах.
- Низкое энергопотребление (по сравнению с GPU).
- В основном используются через облачные вычисления, т.к. требуется специальное оборудование.
- Молодая технология (представлена с 2017 г.)

Сетевые распределенные вычисления

Линейная регрессия:

- $\Theta = \Theta - \alpha * X^T * (X * \Theta - y)$
- $X^T * (X * \Theta - y) = \partial$
- m = 1000
- Разобьем X и y на части в соответствии с количеством компьютеров в сети.
- Каждый компьютер сделает свою часть вычислений.



$$\partial_i = X_i^T * (X_i * \Theta - y_i)$$

$$\partial = \Sigma \partial_i$$

Заблуждения распределенных вычислений

- Сеть надежна.
- Задержка нулевая.
- Пропускная способность бесконечна.
- Сеть безопасна.
- Топология не меняется.
- Администратор только один.
- Стоимость пересылки нулевая.
- Сеть однородна.

Закон Амдала

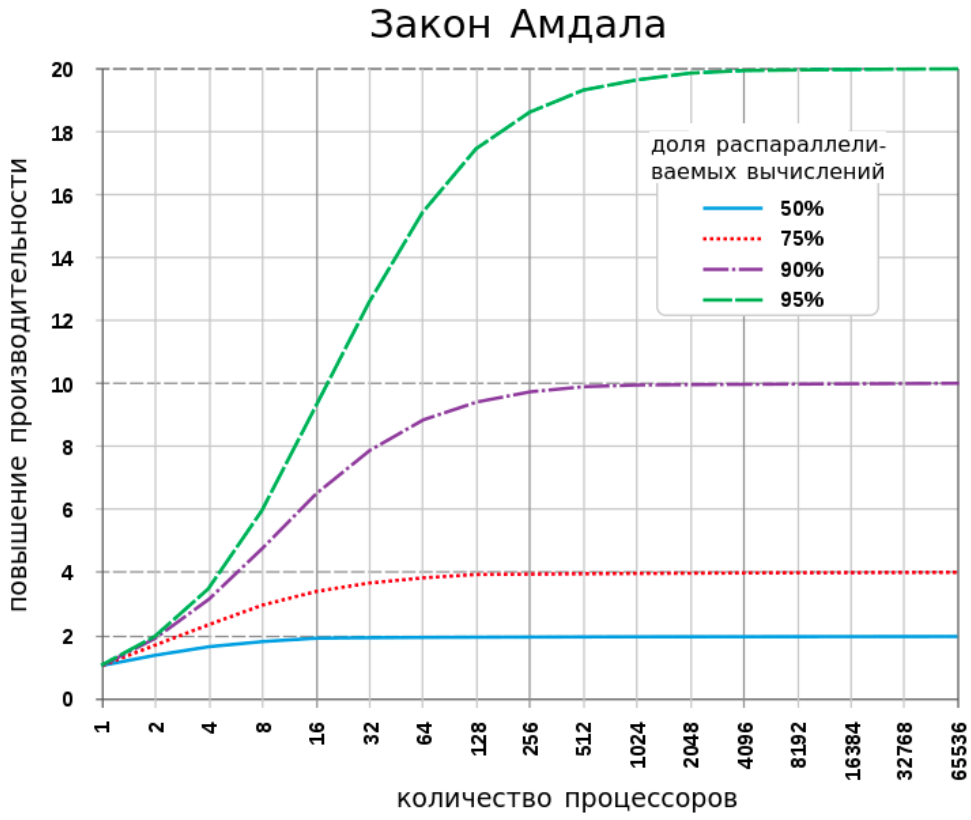
«В случае, когда задача разделяется на несколько частей, суммарное время её выполнения на параллельной системе не может быть меньше времени выполнения самого медленного фрагмента»

α – доля вычислений, во время выполнения которых невозможно распараллеливание

p – количество вычислительных узлов

Ускорение производительности не превышает

$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}}$$



Стохастический градиентный спуск

В методе градиентного спуска минимизируется функция ошибки по всей обучающей выборке. Это позволяет вычислить точное значение градиента, но требует больших вычислений.

Небольшая случайная подвыборка обучающей выборки даст примерно то же направление, но потребует гораздо меньшее количество вычислений.

Стохастический градиентный спуск:

- разбить обучающую выборку на части (minibatch) и осуществить по одной итерации градиентного спуска для каждой из частей (проход всех подвыборок обучающей выборки называется эпохой (epoch) обучения)
- повторять этот процесс, переразбивая обучающую выборку случайным образом

Размер минибатчей обычно от 1 до 100 элементов.

Градиент с моментами

$$\Delta\theta_t = \gamma * \Delta\theta_{t-1} - \eta * dE/d\theta$$

- Сохраняет направление изменения весов с инерцией γ.
- При малом γ совпадает со стахостическим градиентом.
- Позволяет преодолевать небольшие локальные минимумы.
- Обычно γ ~ 0.9

Ускоренные градиенты Нестерова

$$\Delta\theta_t = \gamma*\Delta\theta_{t-1} - \eta*dE(\theta_{t-1}+\gamma*\Delta\theta_{t-1})/d(\theta_{t-1}+\gamma*\Delta\theta_{t-1})$$

- Градиент вычисляется не в точке, где сейчас находимся, а в точке по направлению изменения весов.
- Позволяет быстрее двигаться, если производная в направлении движения увеличивается.
- Можно так ускориться, что перескочить область минимума.

Adagrad

$g_t \equiv dE(\theta_t)/d\theta_t$; $G_t = G_t + g^2_t$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

- Алгоритм накапливает информацию о том, как часто изменяется соединение в величине G_t.
- После продолжительного обучения эти связи почти не изменяются, а изменяются редко задействованные связи.
- Благодаря этой идее соразмерный вклад в обучение вносят редко встречающиеся, но важные признаки.

Adam

Сочетает в себе идею накопления движения и идею более слабого обновления весов типичных признаков.

$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$

$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g^2_t$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}} m_t$$

https://sesc-infosec.github.io/