Влияние различных представлений чисел на эффективность и устойчивость CNN моделей

Гринченко Торбин Лановенко Исмаилов

Постановка задачи

Для обучения более крупных моделей обычно требуется больше вычислительных ресурсов и ресурсов памяти. Основная идея состоит в том, что пониженная точность устраняет проблему памяти, а также уменьшает время выполнения программ. Нагрузка на пропускную способность памяти снижается за счет использования меньшего количества битов для хранения того же количества значений. Время арифметических вычислений также можно сократить на процессорах, которые обеспечивают более высокую пропускную способность для снижения точности математических вычислений. Поэтому наша задача состоит в том, чтобы предъявить эффективную реализацию арифметики с урезанной точностью.

Общая схема работы смешанного подхода

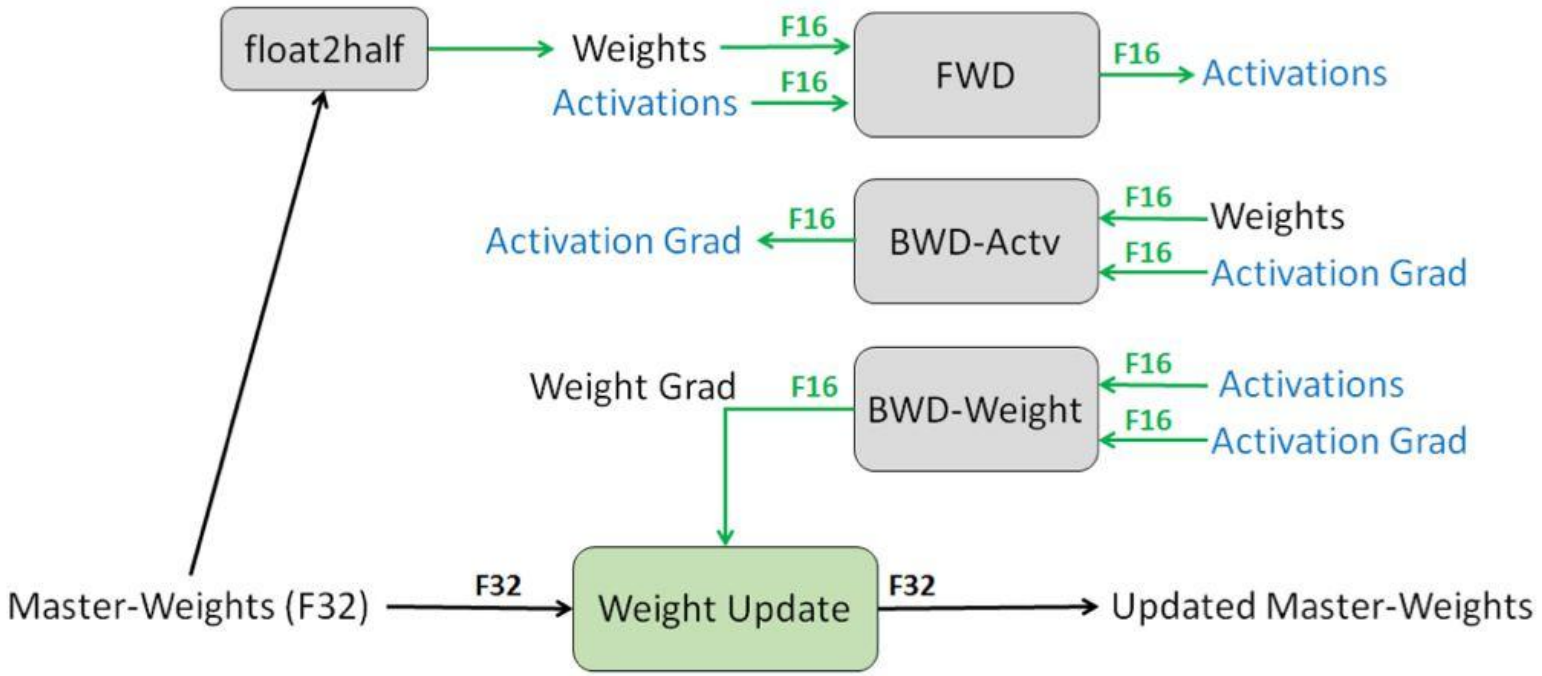
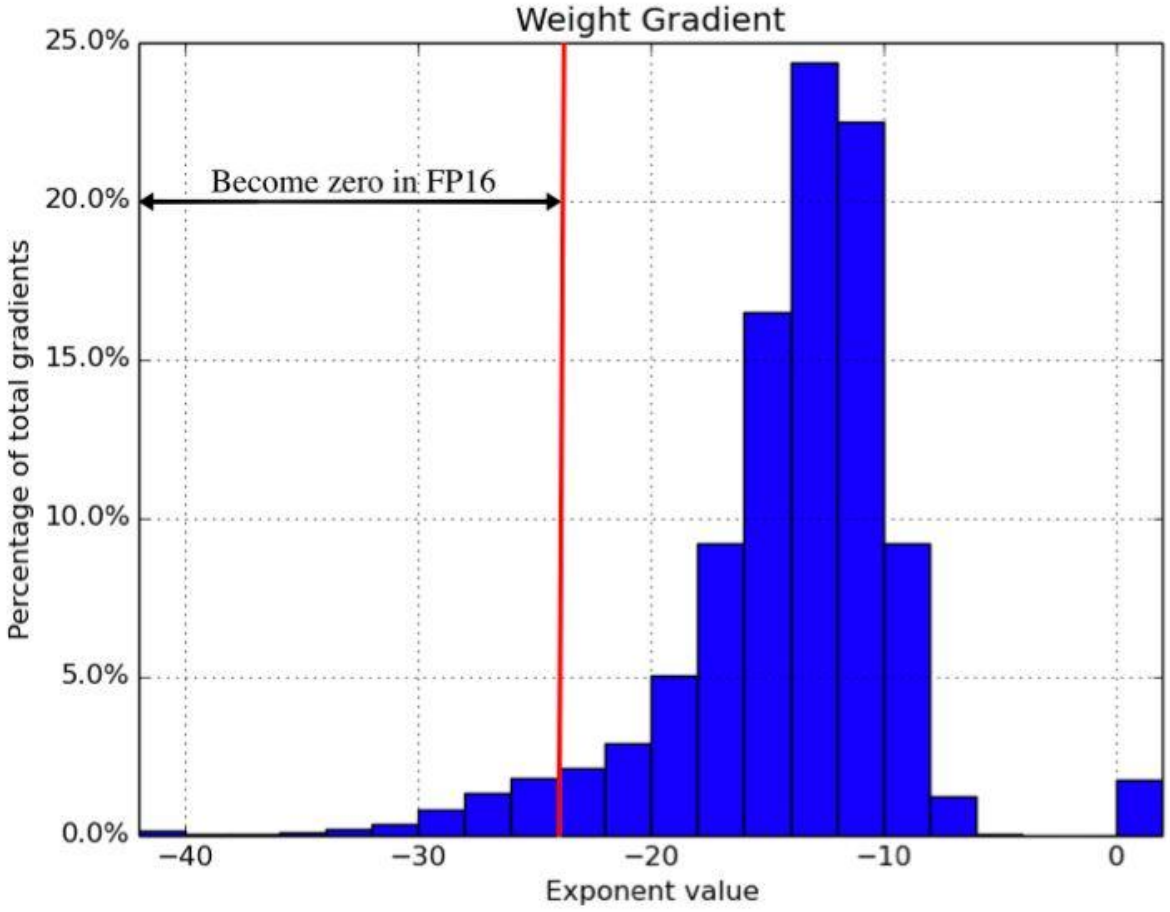


Figure 1: Mixed precision training iteration for a layer.

scaling



Значения градиентов обычно довольно маленькие, в сравнении с размерностью FP16, поэтому перед началом обратного распространения разумно скейлить их, чтобы не упустить действительно важные значения.

Результаты

Тип	Batch Size	epoch	С apex	Accuracy, %	Время работы, sec	Память, MB
Standard	100	5	-	32	63.054	159.1
Mixed precision	100	5	-	32	66.776	132.78
F16 precision	100	5	-	23	60.602	88.54
Mixed precision	100	5	+	32	64.218	154.81
F16 precision	100	5	+	22	58.409	95.2
Standard	32	5	-	41	106.870	181.25
Mixed Precision	32	5	-	42	123.817	159.79
F16 precision	32	5	-	31	107.285	116.19
Mixed Precision	32	5	+	39	137.804	153.8
F16 precision	32	5	+	31	121.288	74.9
Mixed precision	100	15	+	42	204.572	155.19
F16 precision	100	15	+	28	187.183	94.13
Standard	100	15	-	41	186.845	160.04
Mixed precision	100	15	-	41	208.011	133.24
F16 precision	100	15	-	28	189.080	88.54

для cifar10 использовался resnet18

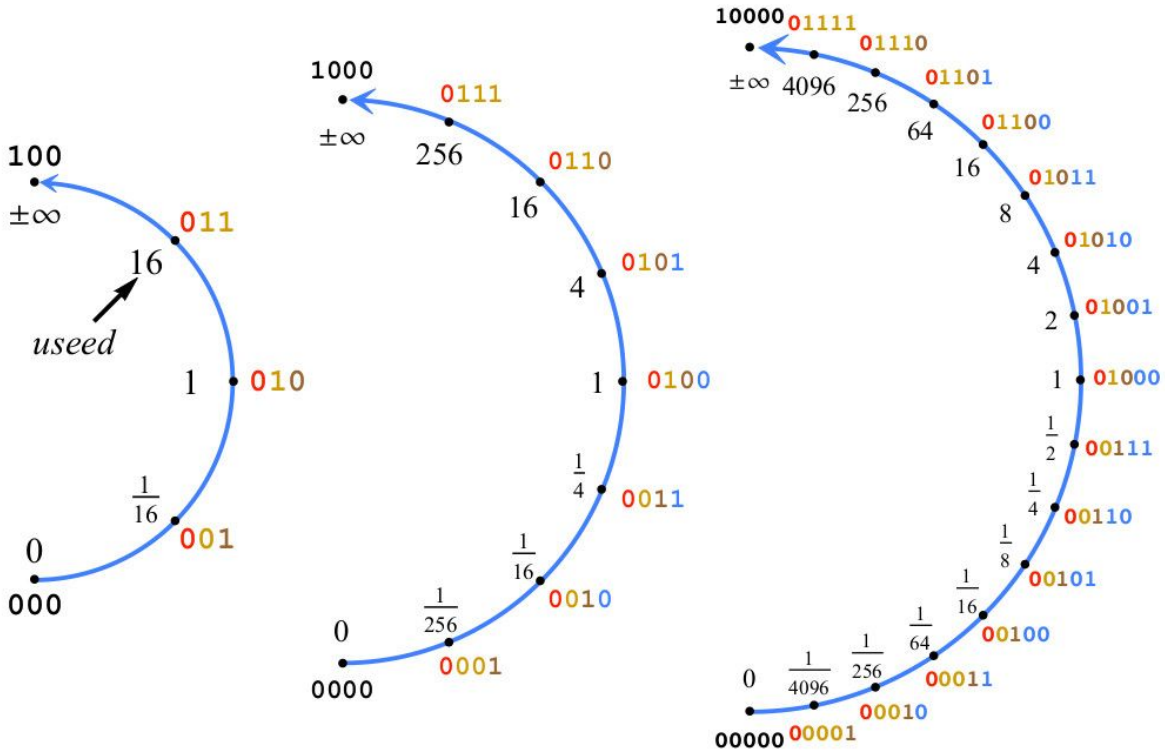
Анализ результатов

Смешанная арифметика действительно позволяет уменьшить память, используемую при обучении, не ухудшая точность. Однако, время обучения не уменьшается как изначально предполагалось. Также как видно из тестов, встроенные методы PyTorch дают лучшую оценку, чем библиотека апекс или примерно такую же. Особенно это заметно при батче большого размера.

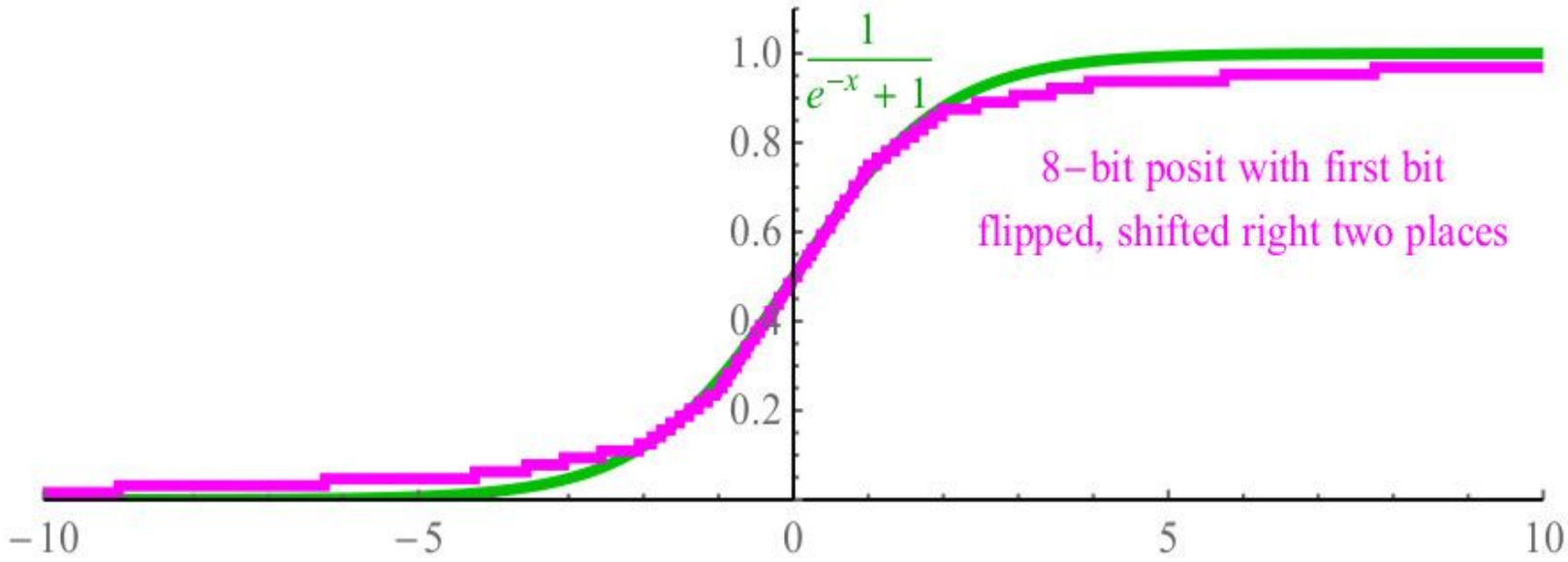
Формат Posit

Число в формате Posit вычисляется следующим образом:

$$x = \begin{cases} 0, & p = 0, \\ \pm\infty, & p = -2^{n-1}, \\ sign(p) \times useed^k \times 2^e \times f & \text{любое другое } p \end{cases}$$

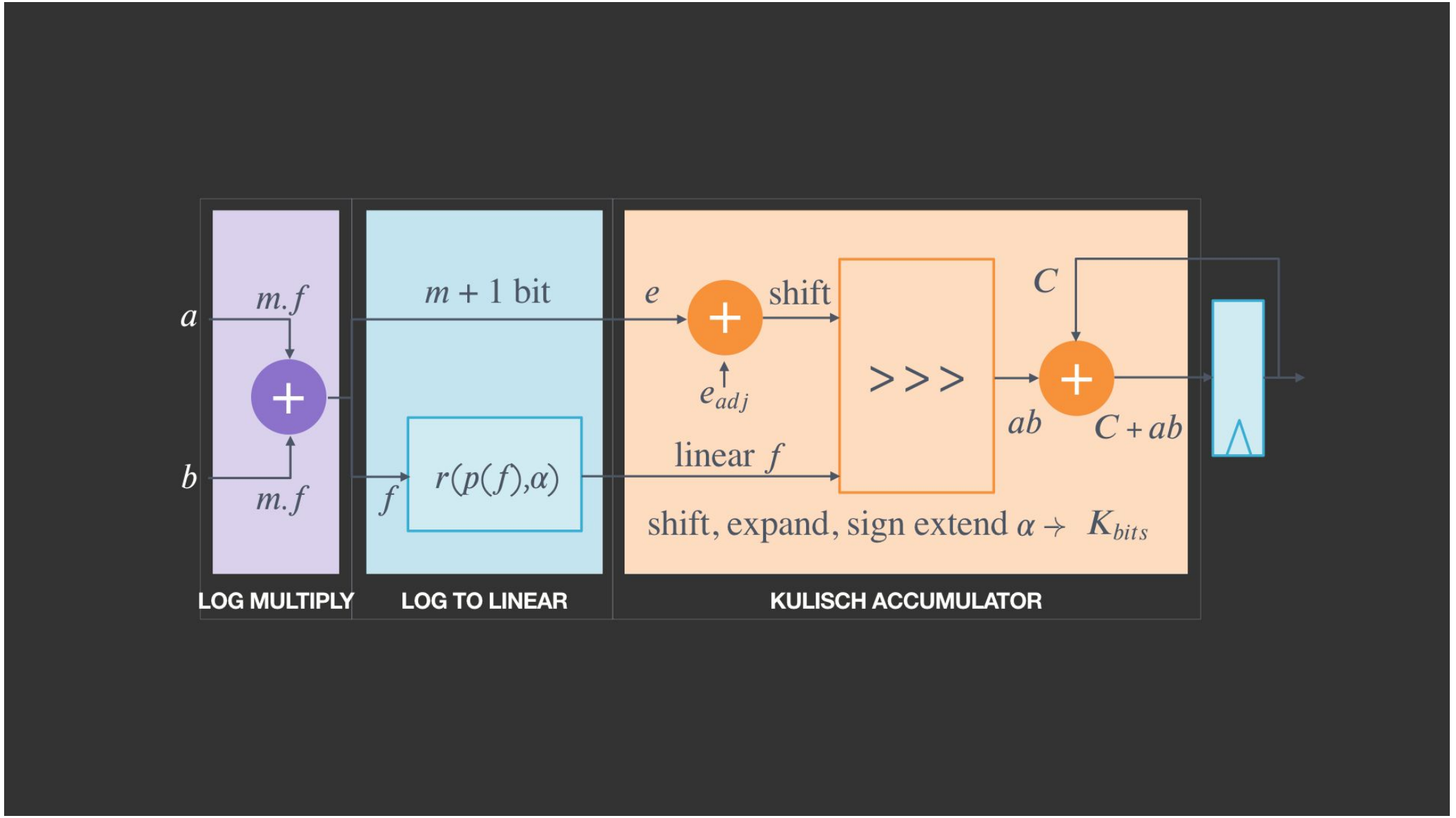


Получение сигмоиды с помощью элементарных битовых операций

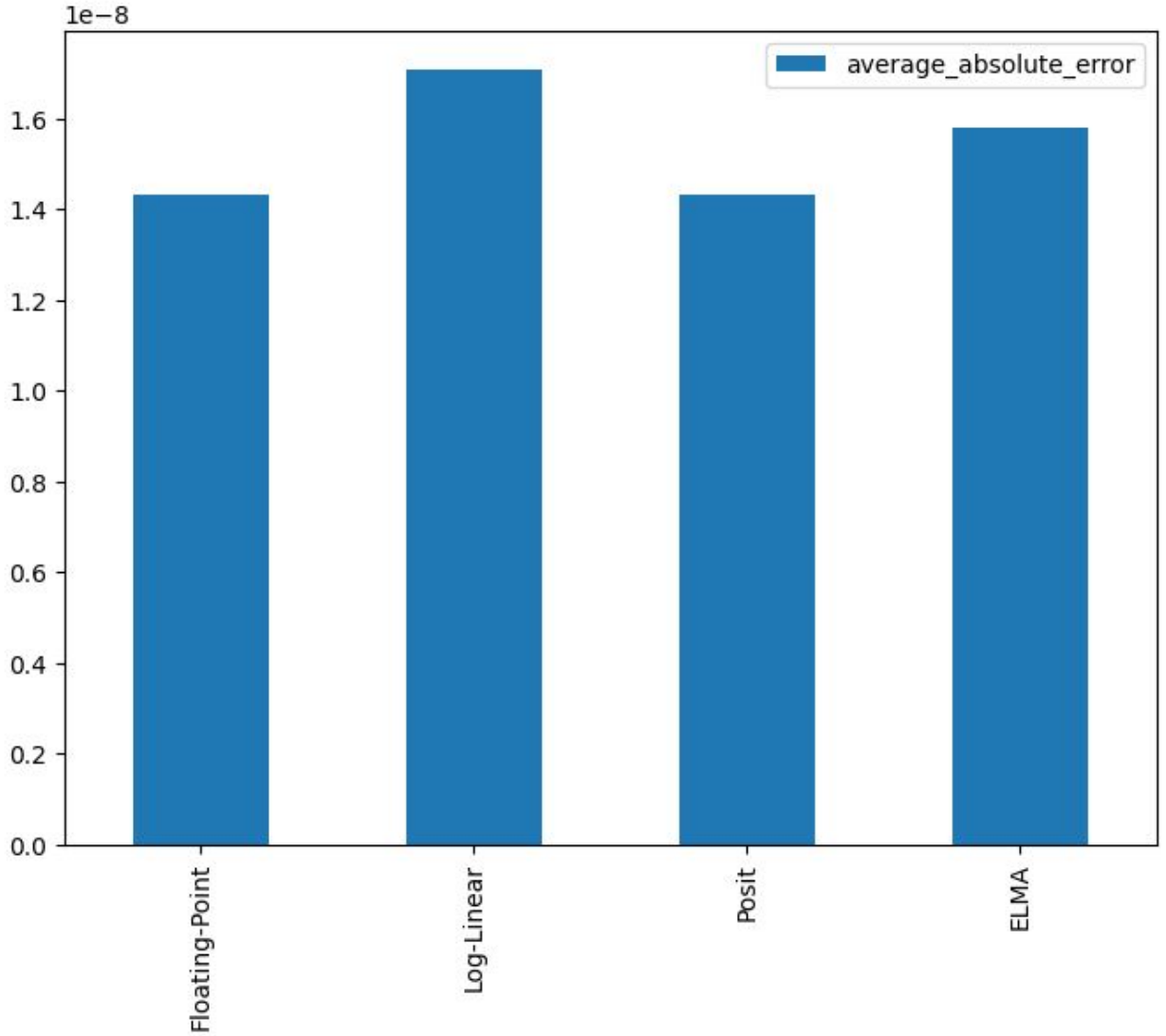


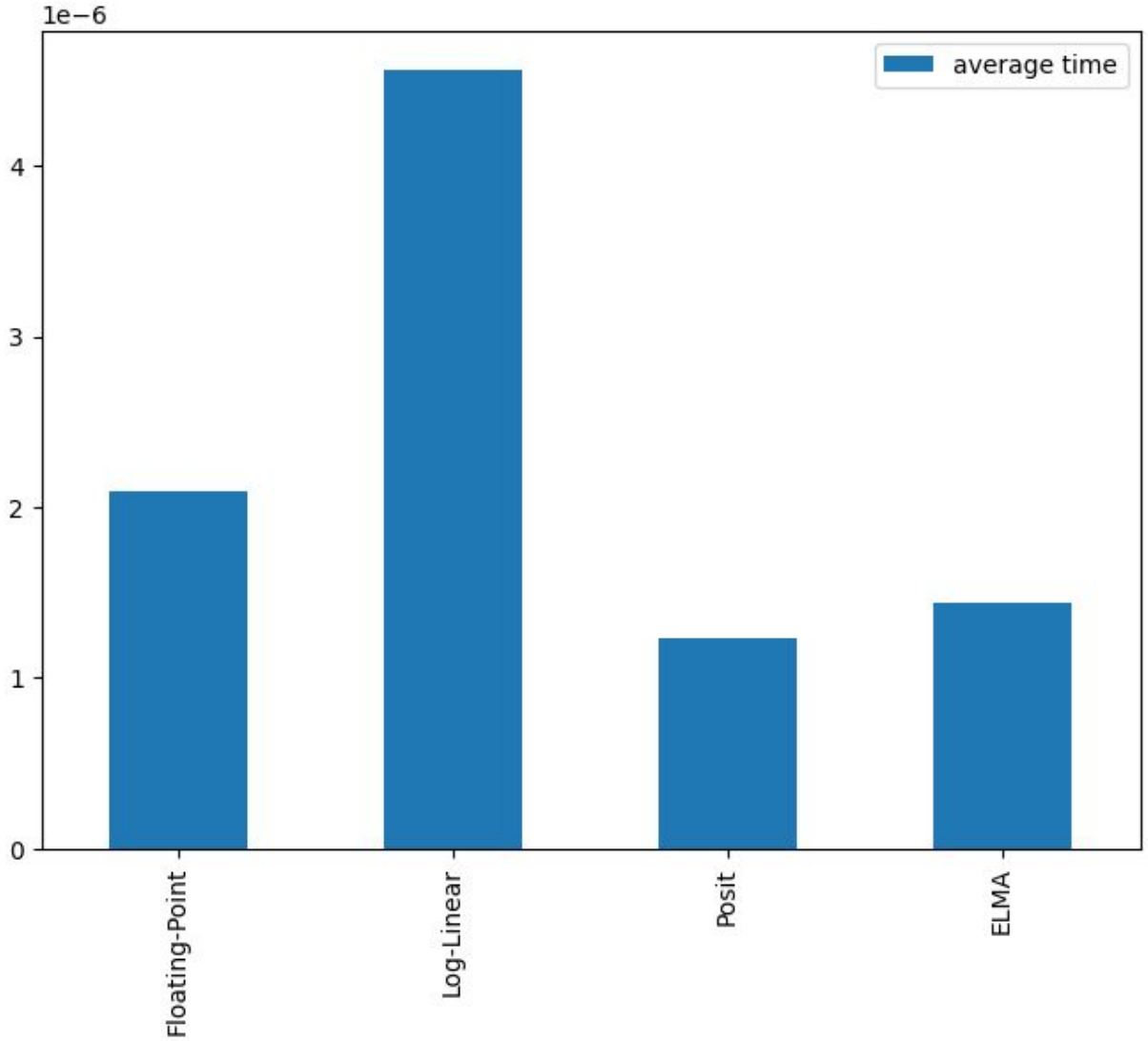
Для получения функции сигмоиды с высокой точностью достаточно пары битовых операций: инвертируем первый бит числа и сдвиг на 2 бита вправо с заполнением пропусков нулями.

ELMA (Exact Log-linear Multiply-Add)



Подсчёт ошибки f(a,b) = a*b - c, где b = c / a при n = 10^6 итерациях





Область развития

1) с появлением возможностей поддержки Posit формата обучить модель с его использованием.
2) при наличии достаточной вычислительной мощности рассмотреть задачу на датасетах большего размера и узнать выйгрыш по времени в этом случае.
3) рассмотреть работу Nvidia AMP для параллельной работы нескольких видеокарт

Ссылки

- https://github.com/228egger/NLA-projects
- https://colab.research.google.com/drive/152Y_Eqm7qfob6dEUsle-mBiiPYQWLX7x#scrollTo=KrziQH4LKuhg
- https://github.com/milankl/SoftPosit.jl/blob/main/docs/softposit_examples.ipynb
- https://arxiv.org/pdf/1710.03740.pdf
- https://colab.research.google.com/drive/16CJu3Wy0t3wwvmZ_UMdTECZGXJQEloDd?usp=sharing
- https://arxiv.org/pdf/1811.01721.pdf
- https://superfri.org/index.php/superfri/article/view/137/232
- https://github.com/NVIDIA/apex/