fast exp and tanh for simpleNN

Manuel Schiller

University of Glasgow

June 18th, 2020

intro

- last time, introduced `simpleNN` for fast NN evaluations
- today:
  - after (auto-)vectorizing matrix multiplication, it's time to turn to activation functions which contain $\exp(x)$ and $\tanh(x)$
  - micro-bechmark time and accuracy behaviour for long vectors of floating point numbers

benchmarking method

- time benchmarking:
  - time the following code fragment for cacheline-aligned source and destination

```
template <typename T, typename FN>
[[gnu::noinline]] void bench(const T * first, const T * last, T * dest, FN&& fn) noexcept
{
     for (; last != first; ++first, ++dest) *dest = fn(*first);
}
```

  - time from `std::chrono::high\_resolution\_clock::now()`
  - array size 65536, take minimum time of 128 trials
  - subtract time it takes to copy source to destination
- accuracy benchmarking
  - ULP (Unit in the Last Place): $x = 1.\mathtt{mmm}\ldots\mathtt{mmm}\mathtt{u} \times 2^{\mathtt{exp}}$ (where m, ..., u: mantissa bits)
- compiler flags: `-std=c++14 -pedantic -Wall -Wextra -g -O2 -march=native -ftree-vectorize`

computation method

- generalities:
  - to vectorize well, control flow must not depend on input data
  - therefore: branchless if, abs, copysign, ...

```
// almost (cond ? a : b)
float sel(bool cond, float a, float b)
{
    int32_t mask = -int32_t(cond);
    return bit_cast<float>((bit_cast<int32_t>(x) & mask) | (bit_cast<int32_t>(b) & ~mask))
}
```

- $\exp(x)$
  - use $\exp(x) = \exp(n/\ln(2) + r) = 2^n P(r)/Q(r)$
- $\tanh(x)$
  - use $x' = x/8$ to reduce argument magnitude
  - use $\tanh(x') \sim P(x')/Q(x')$
  - use argument doubling formula $\tanh(2x) = \frac{2\tanh(x)}{1+(\tanh(x))^2}$ three times to undo argument reduction

results

- time for the float version:

$[ns]$	Core i7-2640 gcc 10	Core i7-2640 clang 10	Ryzen 7 2700U gcc 8	Ryzen 7 2700U clang 10
`std::exp`	6.84	6.87	2.41	2.39
`vdt::fast_exp`	10.55	2.48	4.14	1.09
`my::exp`	2.16	1.47	1.25	0.96
`std::tanh`	30.34	30.35	15.97	15.72
`vdt::fast_tanh`	22.77	5.30	4.16	1.14
`my::tanh`	5.44	5.31	1.07	0.84

- accuracy:

$[ULP]$	min.	max.	RMS
`vdt::exp`	-1	4.9×10^6	185
`my::fast_exp`	-2	2	0.35
`vdt::tanh`	-6	6	0.83
`my::fast_tanh`	-6	7	0.83

- VDT uses my tanh code anyway
- `vdt::exp` has a problem for large x where it gets quite inaccurate

next steps

- nice speedup possible for activation functions
- complete the tables for doubles, and for different CPUs/platforms (log files exist, I just need to extract this)
- put this into simpleNN, and measure impact
- possibly: feed that back into some library