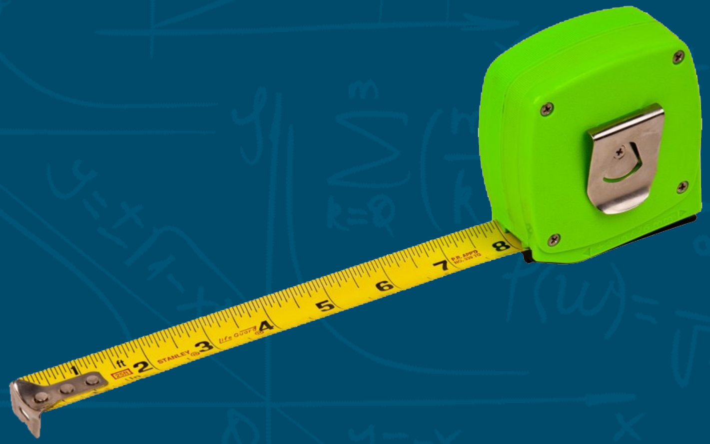
Operator Tolerance Conformance Considerations

DWAYNE ROBINSON
2022-10-06

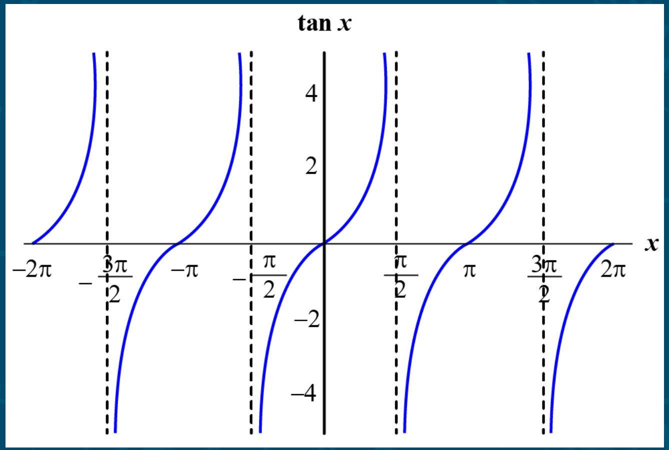


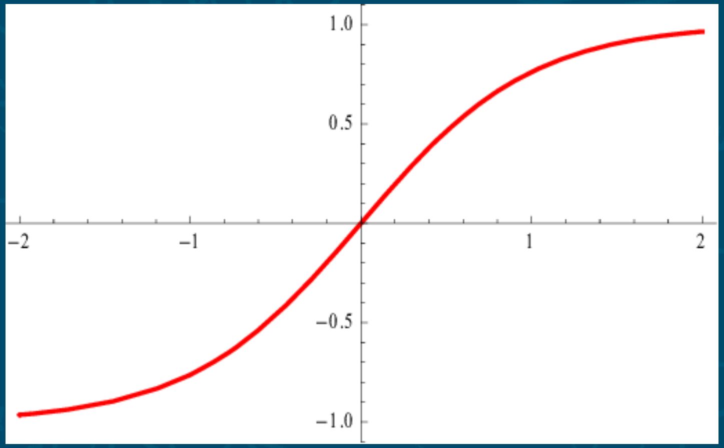
Operator Categories

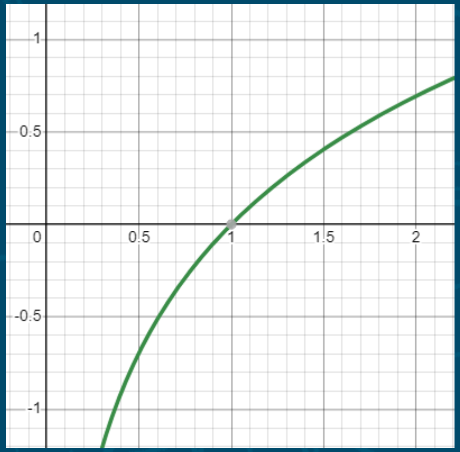
- Grouping them simplifies the problem:
  - **Data movement**: slice, pad, concat, split, reshape, squeeze, unsqueeze, transpose, gather, scatter, padding, depthToSpace, spaceToDepth, topK...
  - **Data generation**: diagonalMatrix, trilu, fillValueSequence...
  - **Exact math**: abs, neg, clamp, ceil, floor, min, max, relu, reduceMin/Max, maxpoolNd...
  - **Simple math**: add, subtract, multiply, divide, linear, leakyRelu, hardSigmoid...
  - **Complex math**: exp, log, pow, softsign, softmax, softplus, sigmoid, sqrt...
  - **Trigonometric functions**: sin, sinh, cos, cosh, tan, tanh...
  - **Lossy accumulation**: convNd, gemm/matmul, batch/instanceNormalization, reduceSum, averagePoolNd, resampleNd...
  - **Very complex iterative**: gru, gruCell, lstm, rnn...

Testing

- Verifying operator *behavior* conformance vs device *precision*?
- Curated input data can help control problematic outliers
  - Selected ranges (to avoid asymptotes)
  - Integers and powers of two rather than purely random inputs
  - Same sign (avoid catastrophic cancellation e.g. negative GEMM bias)



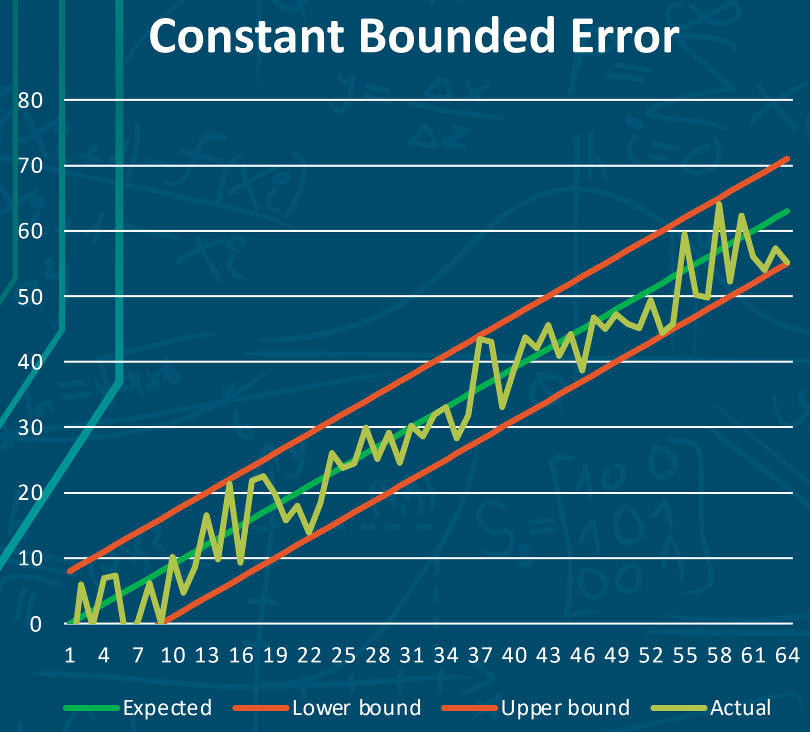


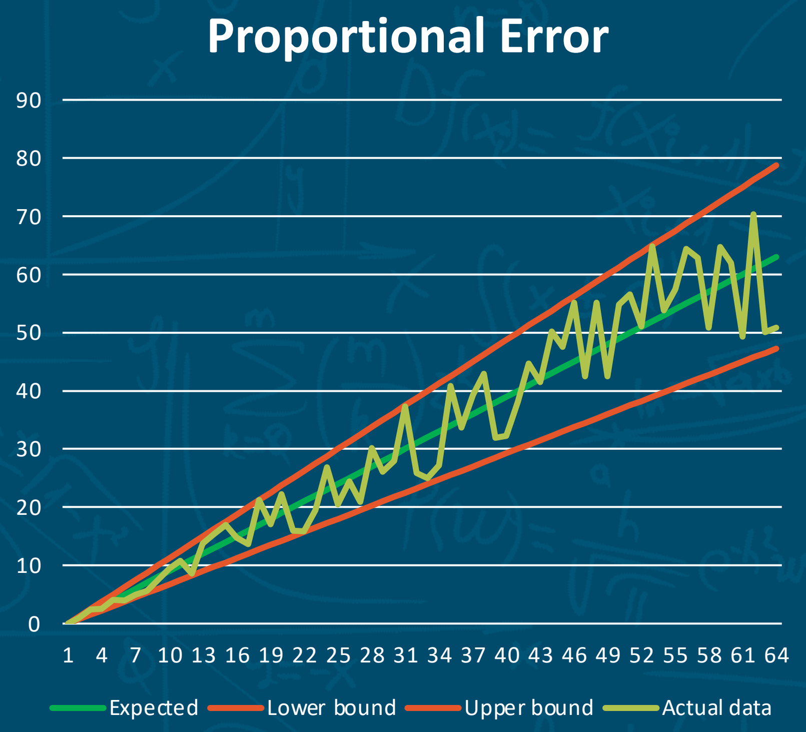


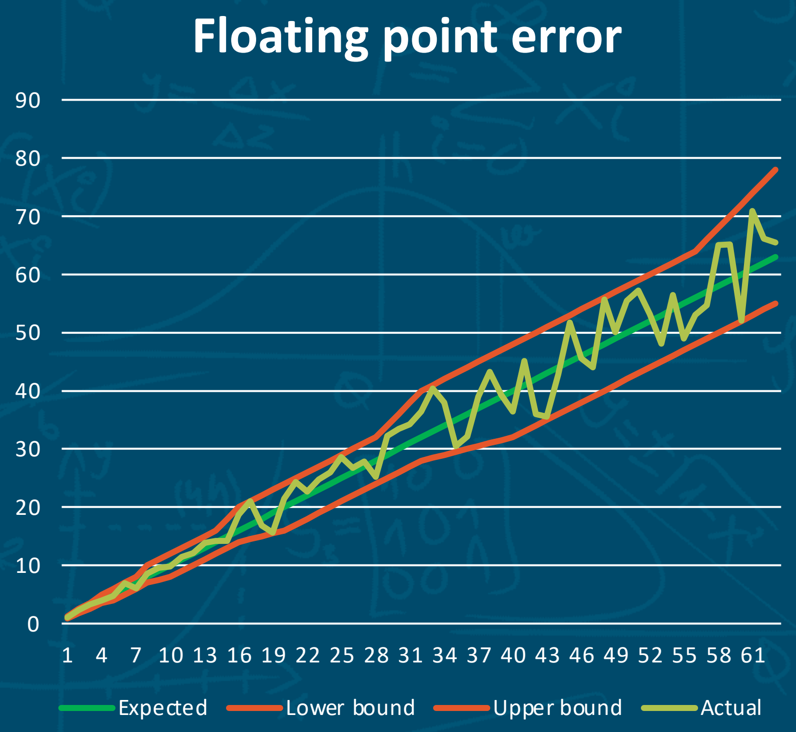
Precision issues and gotchas

- Subtraction of nearly equal numbers (catastrophic cancellation)
- Division by very small numbers (magnifies earlier errors)
- Asymptotes of trigonometric and nonlinear functions
- Subnormals, infinities, near infinities, NaNs
- Differing compute precision vs tensor precision
  - Higher precision computation than tensor type (a final round off into tensor)
  - Lower precision computation than tensor type (e.g. float32 input/output with float32x13f10e8s1 or fixed24f12i11s1 compute)

Ideal (expected) vs Actual Signal Behavior







Measurement methods

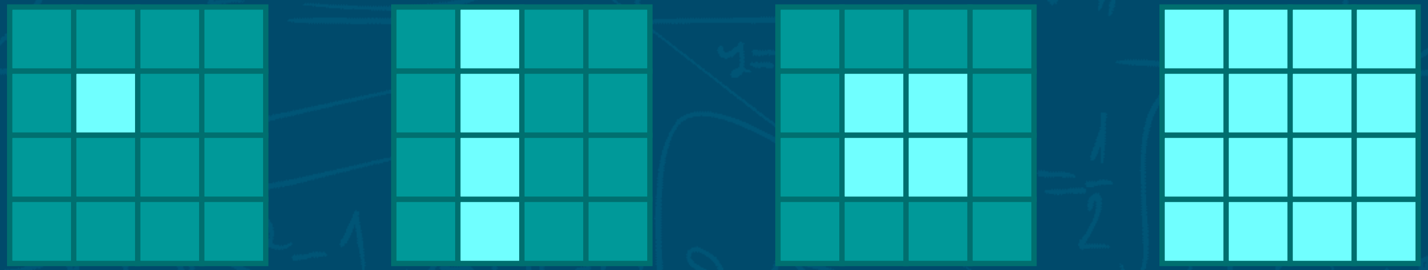
- Methods of tolerance ("fundamental deviation")
  - Absolute tolerance - expected within [actual - ATOL, actual + ATOL]
  - Relative tolerance - expected within [actual - (actual*RTOL), actual + (actual*RTOL)]
  - Unit last place – expected.rawbits within [actual.rawbits - ULP, actual.rawbits + ULP]
- Want tight bounds matching the error distribution
- No single method sufficient for *all* cases, and so choose the appropriate ones for the operator (e.g. legitimate points on functions like log at x=1 and atan at x=0 have denominator issues with RTOL and ULP)
- Note relative tolerances can be expressed within ULP (eliminating one error inducing multiplication), making ATOL and ULP sufficient

Contributing error factors

- Number of calculations
  - Input elements per output element (IEPOE)
  - Total lossy math operations
- Device-specific differences for compute precision and floating-point behavior
- Nature of data values
  - large/small, homogeneous, varied, integral, pow2…
- Algorithm used
  - e.g. summation order of sequential reduction vs iterative pairwise reduction
- Operation fusions
  - They complicate error tolerance because of the error magnification effects
  - No longer about *operator* tolerance but rather that of a miniature graph
  - In the rare cases where these implementation optimizations are exposed at an API level, they should have less or equal error than each operator chained

Contributing error factors – IEPOE (input elements per output element)

- Not used directly, but conceptually tells degree of complexity, as the potential error often proportional to number of input elements
  - Elementwise = 1
  - GEMM = a.sizes.width (or equivalently b.sizes.height)
  - Conv2D = filter.sizes.width * filter.sizes.height * (input.sizes.channel / groupCount)
  - Reduction = input sizes multiplied for each reduction active axis
  - Pooling = window size



Contributing error factors – lossy math count

- Intuitively, more lossy math ops yields greater potential error. e.g. compare
  - simple linear activation = just 2 math ops, vs
  - softmax = elementsToReduce * 3 + 3 math ops
    exp_e(a - reduceMax(A, axes)) / reduceSum(exp_e(A - reduceMax(A, axes)), axes);
- The lossy op count establishes a sensible upper bound for the worst serial ordering.
  - A value-increasing operation (e.g. + or *) with 1 ULP of error repeated 100x yields <= 1*100 ULP.
  - Experiments demonstrate such serial operations (e.g. ReduceSum, ReduceProd, DotProduct) yield ULP <= n/~3 even when rounding is always forced toward zero or always toward infinity.
  - And in practice, error is *much less* due to nearest even rounding balancing the deviations (but don't let that false comfort fool you into thinking the worst case can't happen).
  - Note any *exact* operations are ignorable along the way (e.g. min, max, *2, /4)
- Adding respective ULP's tolerances for operators (and even fusions) sets a sensible upper bound - *disclaimer: not a rigorous mathematical proof, but it works in practice and beats pulling numbers out of thin air, or picking arbitrary implementations for reference*

Contributing error factors – device specific differences

- Compute precision and tensor data type
  - float16 vs float32 vs non-standard types (float19of32, bfloat16)
  - rounding modes (toward zero, toward infinity, to nearest even)
  - subnormal flushing (to flush or not to flush, that is the question)
  - different NaN bit patterns (not all not-a-numbers are equal)
  - saturation differences (some GPU's/NPU's may saturate to maximum positive number, whereas others saturate to infinity)
- Algorithms used (e.g. summation order)
  - Device driver implementation specifics (e.g. calculation vs table interpolation lookups)

Questions?