A Numerically Efficient Algorithm for Computing the Softmax Function

Daniel Eftekhari

Let $x \in \mathbb{R}^n$ be the input to a softmax layer, which is defined as:

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{1}$$

with $\sigma(x_i) \in [0, 1]$. On low-floating point precision hardware, conventional algorithms for computing the softmax function are prone to numerical errors. Here I present an alternative approach to computing the softmax function.

We begin by decomposing the exponential function e^x for some input x as:

$$e^x = 2^k e^r, \tag{2}$$

which follows when $x = k \ln 2 + r$, with (usefully) k only taking on integer values, thereby only requiring r to take on fractional values.

Even with this decomposition, we are still left with the task of computing the exponential function. However now we need consider only a much smaller range of input values, in the range $-\frac{1}{2} \ln 2 \leq r \leq +\frac{1}{2} \ln 2$,[1] rather than the original range of $-\infty < x < +\infty$.

The advantages of restricting ourselves to this input interval for r are as follows:

- With a smaller range of input values, a Taylor series with a few terms suffices in near-exactly computing the exponential function. This is depicted graphically below.
- Because the input range is now symmetric around zero, low-order Taylor approximations of the exponential function $f(x) = e^x$ around $x \approx 0$, such as $T_1(f(x)) \approx 1 + x$, better approximate the exponential function. This is important because a) the exponential function can be more accurately described using polynomial approximations near $x \approx 0$, and b) low-order polynomials can be computed more quickly than higher-order polynomials.

Next we solve for k and r. First solving for k, we have $k = \frac{x}{\ln 2} - \frac{r}{\ln 2}$. Because $-\frac{1}{2} \ln 2 \leq r \leq +\frac{1}{2} \ln 2$ we have $-\frac{1}{2} \leq \frac{r}{\ln 2} \leq +\frac{1}{2}$, and since k is an integer by assumption, we may simply re-write the expression as $k = \lfloor \frac{x}{\ln 2} + \frac{1}{2} \rfloor$. Having computed k, we are left with $r = x - k \ln 2$. Computing k and r requires only basic arithmetic operations (multiply and accumulate), and a single instance of the floor function, all of which can be done efficiently on hardware, in particular with low-floating

[1] This follows because x is written in multiples of $\ln 2$, and therefore constraining r to half this interval suffices.

point precision.

Figure 1 exemplifies the significance of restricting the input range to $\left[-\frac{1}{2}\ln 2, +\frac{1}{2}\ln 2\right]$ when computing e^r, as each of the Taylor approximations closely approximate the function in this interval. Contrasting this with the approximation of e^r in the larger interval $[-2, +2]$ (Figure 2) further demonstrates the value of restricting the input range.

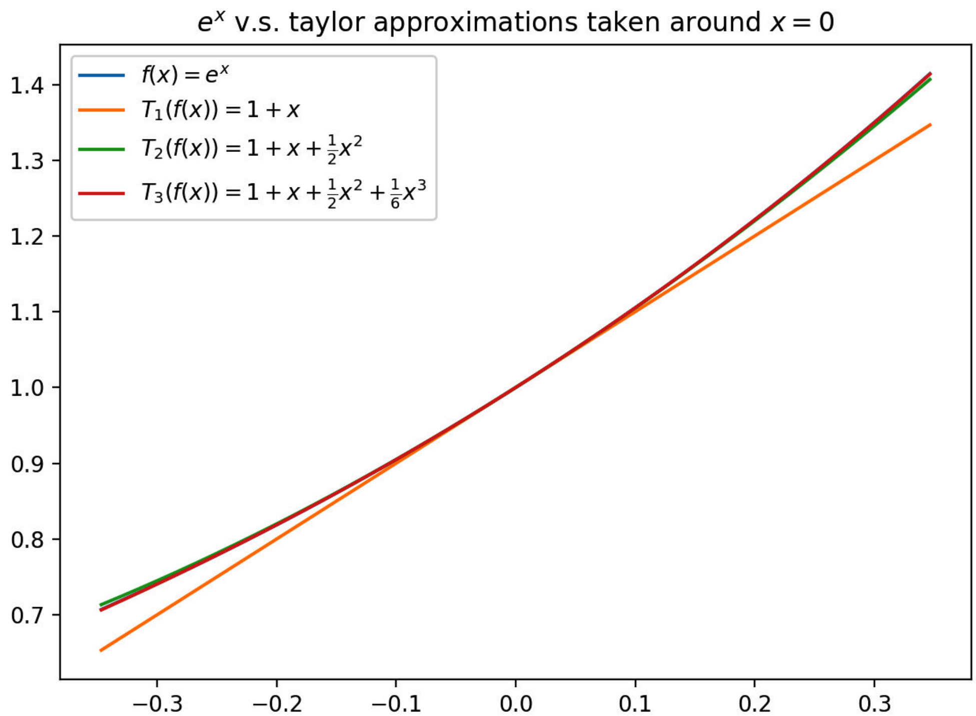

Figure 1: Taylor approximation of $f(r) = e^r$ around $r = 0$, for polynomials of order one to three, in the interval $\left[-\frac{1}{2}\ln 2, +\frac{1}{2}\ln 2\right]$.

Figure 3 and Figure 4 demonstrate the error and relative error, between the Taylor approximation to the exponential function, for polynomials of order one to three.

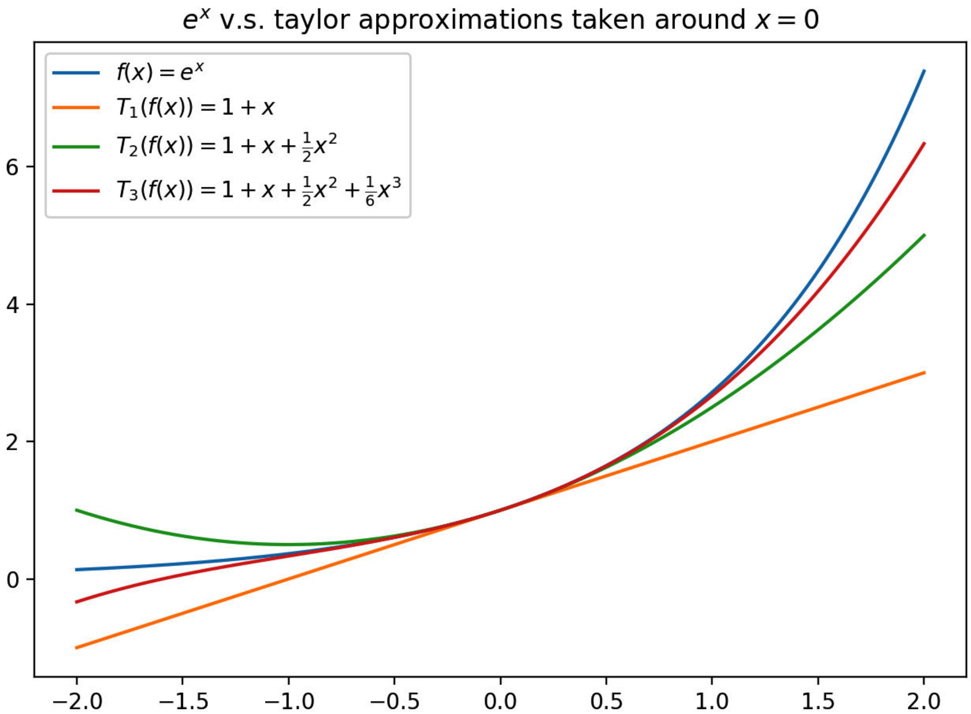

Figure 2: As in Figure 1, but with a larger input range of $[-2, +2]$. Note how the low-order polynomials' approximations quickly degrade away from zero.

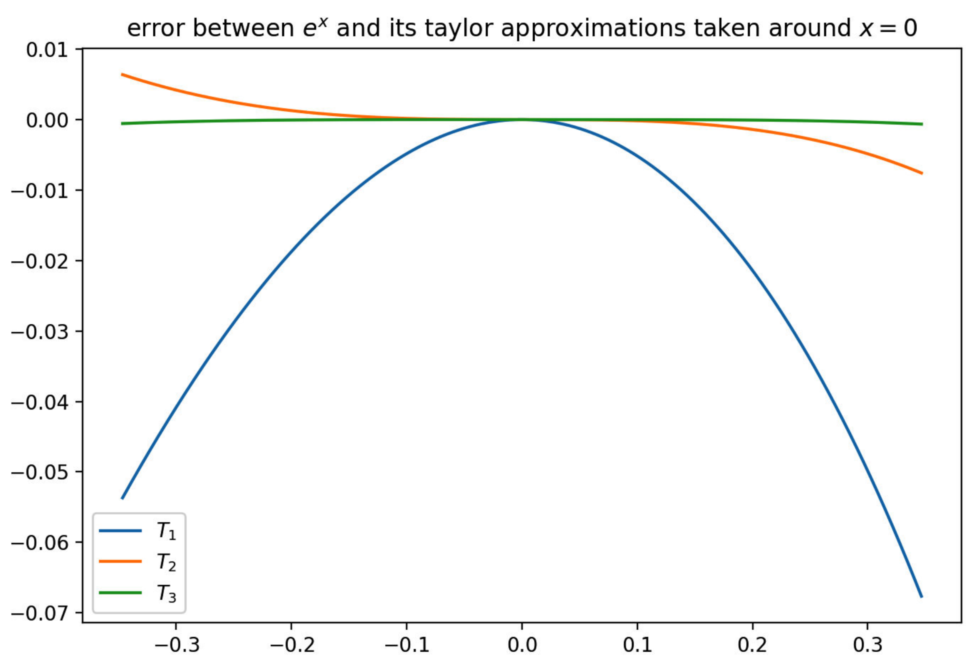

Figure 3: The error $T_s(f(x)) - f(x)$ between Taylor approximations of order s and $f(x) = e^x$.

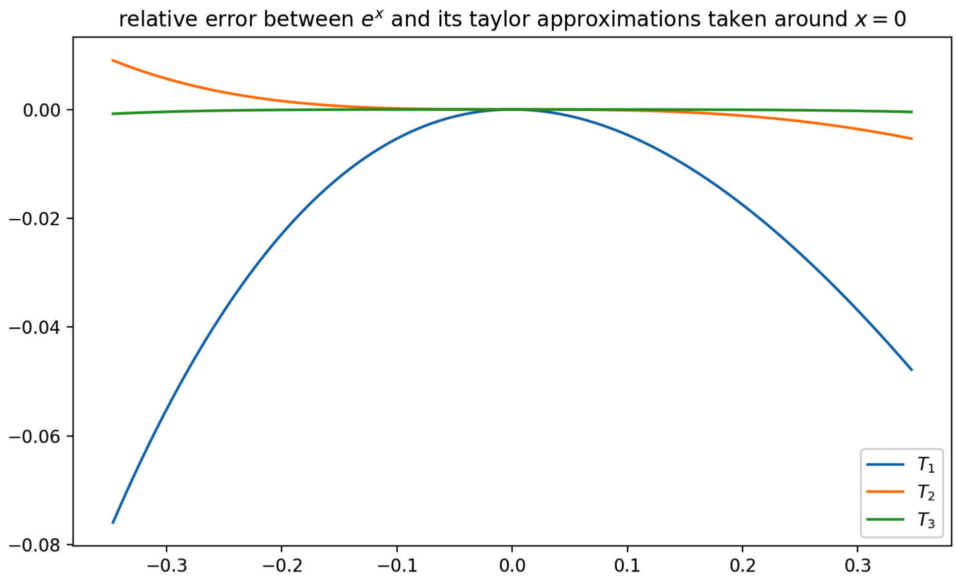

Figure 4: The relative error $\frac{T_S(f(x))-f(x)}{f(x)}$ between Taylor approximations of order s and $f(x) = e^x$.