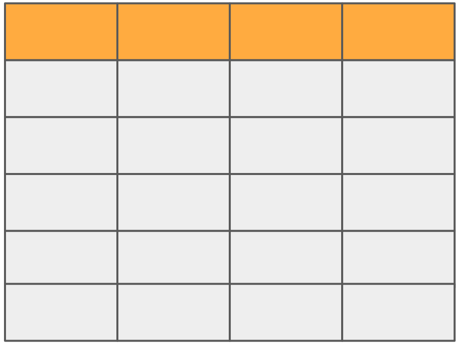
Week 5

ReLU



- Vectorize over rows
- Load vector and compare with zero
- Use pytorch as golden model
    - Compute the result with pytorch
    - Store the correct result in memory before running simulation
    - Compare with the simulation result
- First verify the program with Spike, then run the hardware simulation

```c
void relu(float *mat, int row, int col) {
  size_t vlmax = vsetvlmax_e32m1();

  for (int i = 0; i < row; i++) {
    for (int j = 0; j < col;) {
      int vl = vlmax;
      if (j + vlmax > col) vl = col - j;

      vfloat32m1_t vec = vle32_v_f32m1(&mat[i * col + j], vl);
      vec = vfmax_vf_f32m1(vec, 0, vl);
      vse32_v_f32m1(&mat[i * col + j], vec, vl);

      j += vl;
    }
  }
}
```

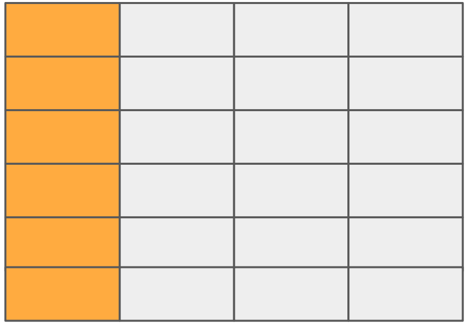
```python
def relu(mat):
    act = nn.ReLU()
    return act(torch.from_numpy(mat)).numpy().astype(np.float32)

row = 256
col = 1024

# Generate inputs
mat = rand_matrix(row, col)
o = np.zeros((row, col)).astype(np.float32)
o_gold = relu(mat)
```

LayerNorm

- Vectorize over columns
  - vec(i): load one element from each row
- Three steps:
  - Mean calculation
  - Variance calculation
  - Apply to the elements
- In debugging



```
    // ==================================================
    // mean calculation
    // ==================================================
    partial_sum = vfmv_v_f_f32m1(0, vl);

    for (int j = 0; j < col; j++) {
+----  4 lines: stride load: in[i * col + j] ~ in[i+vl][j]·········
    }

    partial_sum = vfdiv_vf_f32m1(partial_sum, col, vl);
    vse32_v_f32m1(&mean[i], partial_sum, vl);

    // ==================================================
    // variance calculation
    // ==================================================
    partial_sum = vfmv_v_f_f32m1(0.00001, vl);

    for (int j = 0; j < col; j++) {
+---- 10 lines: vfloat32m1_t vec_x = vlse32_v_f32m1(&mat[i * col +
    }
    partial_sum = vfdiv_vf_f32m1(partial_sum, col, vl);
    // 1/sqrt(var)
    partial_sum = vfrsqrt7_v_f32m1(partial_sum, vl);
    vse32_v_f32m1(&var[i], partial_sum, vl);

    // ==================================================
    // apply to the elements
    // ==================================================
    for (int j = 0; j < col; j++) {
+---- 10 lines: vfloat32m1_t vec_var = vle32_v_f32m1(&var[i], vl);
    }

    i += vl;
  }
}
```

Taylor Series

- Taylor expansion around a point a:

$$f(x) = f(a) + f'(x-a) + f''\frac{(x-a)^2}{2!} + f^{(3)}\frac{(x-a)^3}{3!} + \ldots$$

- For exponential equation: (q = x - a, 6-stage for float)

$$e^x = e^a[1 + q + \frac{q^2}{2} + \frac{q^3}{6} + \frac{q^4}{24} + \frac{q^5}{120} + \frac{q^6}{720}]$$

$$e^x = e^a[1 + q(1 + q(\frac{1}{2} + q(\frac{1}{2} + q(\frac{1}{6} + q(\frac{1}{24} + q(\frac{1}{120} + q\frac{1}{720})))))))]$$

- choose a = 0 (power series), for k-stage series: (k+1) MAC OPs
- Disadvantage: more stages are needed for values much larger than 0 (a).

High throughput floating point exponential function implemented in FPGA, Peter Mal´ık

Table-Lookup Algorithms for exp(x)

- f: function, I: domain of interest, cj: break points j = 1, 2, …, N, tabulates f(cj)
- Reduction:
  - choose a breakpoint ck close to x
  - r = R(x, ck) = (x - ck)
- Approximation:
  - e^r
  - p(r) ~= f(r), p is usually a polynomial (power series)
- Reconstruction:
  - f(r) = e^x = e^(ck) * e^(x - ck)

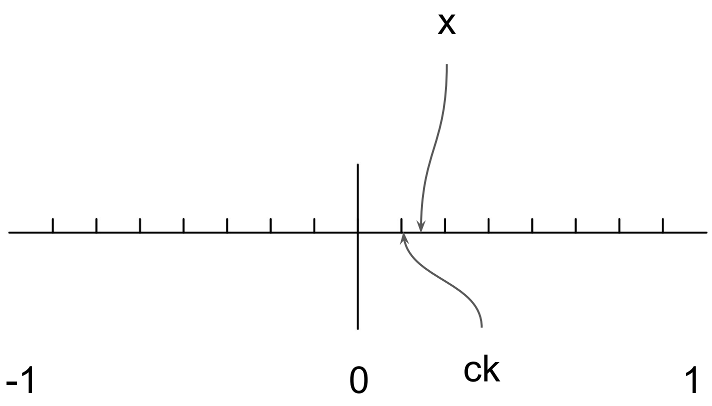


Table-Lookup Algorithms for Elez entary Functions and their Error Analysis, Ping Tak Peter Tang

Other Methods

- Parabolic Synthesis $f_{org}(x) \approx s_1(x) \times s_2(x) \times ..... \times s_n(x)$
  - Normalized input output (0, 1)
  - Different sub-functions can be run in parallel
- CORDIC Algorithm
  - Replace multiplication with add, sub, shift
  - Use theta that tan(theta) = 2^(-i)
  - Example: 60 deg
    - 0 + 45 + 26.525 - 14.036 + 7.125 -3.576 = 61.078
  - exp(x) = sinh(x) + cosh(x)

Sub-function	Corresponding coefficient
$s_1(x) = x + (c_1 \times (x - x^2))$	$c_1 = \lim_{x \to 0} \frac{f_{org}(x)}{x} - 1$
$s_2(x) = 1 + (c_2 \times (x - x^2))$	$c_2 = 4 \times (f_1(0.5) - 1)$
$s_{30}(x) = 1 + (c_{30} \times (x_3 - x_3^2))$ $s_{31}(x) = 1 + (c_{31} \times (x_3 - x_3^2))$ $x_3 = 2 \times x - \lfloor 2 \times x \rfloor$	$c_{30} = 4 \times (f_2(0.25) - 1)$ $c_{31} = 4 \times (f_2(0.75) - 1)$
$s_{40}(x) = 1 + (c_{40} \times (x_4 - x_4^2))$ $s_{41}(x) = 1 + (c_{41} \times (x_4 - x_4^2))$ $s_{42}(x) = 1 + (c_{42} \times (x_4 - x_4^2))$ $s_{43}(x) = 1 + (c_{43} \times (x_4 - x_4^2))$ $x_4 = 4 \times x - \lfloor 4 \times x \rfloor$	$c_{40} = 4 \times (f_{30}(0.125) - 1)$ $c_{41} = 4 \times (f_{31}(0.375) - 1)$ $c_{42} = 4 \times (f_{32}(0.625) - 1)$ $c_{43} = 4 \times (f_{33}(0.875) - 1)$

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \qquad R_i(\theta) = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \end{bmatrix}$$

$$R_i = \frac{1}{\sqrt{1 + \tan^2(\theta_i)}} \begin{bmatrix} 1 & -\tan(\theta_i) \\ \tan(\theta_i) & 1 \end{bmatrix} \qquad v_i = K_i \begin{bmatrix} 1 & -2^{-i} \\ 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix}$$

$$K(n) = \prod_{i=0}^{n-1} K_i = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1 + 2^{-2i}}} \qquad K = \lim_{n \to \infty} K(n) \approx 0.6072529350088812561694$$

A Methodology for Parabolic Synthesis, Erik Hertz and Peter Nilsson
A VLSI Implementation of Logarithmic and Exponential Functions Using a Novel Parabolic Synthesis Methodology Compared to the CORDIC Algorithm, Peyman Pouyan, Erik Hertz, and Peter Nilsson
The CORDIC Trigonometric Computing Technique*, JACK E. VOLDER
Digital Arithmetic, Milos D. Ercegovac, Tomas Lang
The CORDIC method of calculating the exponential function, Leonid Moroz, Volodymyr Samotyy