High Performance Computer Architectures Practical Course - Exercise 2 -

Tutorium 1

David Jordan (6260776)
Florian Rüffer (7454628)
Michael Samjatin (7485765)

May 4, 2023

1 Neural Networks and SIMD

1

2

The task for this exercise is to add SIMD (Single Instruction, Multiple Data) support to our neural network functions. We will start of with the simd-ized version of the ReLU activation function.

The first noticable detail is the changed input. We input type is "fvec". Fvec is a SIMD-packed set of values, to be exact 4 packed 32-bit floating point numbers. Once again we create a result vector, the same size as the input. Subsequently we loop for 1.) every element of fvec (here this is 4) and 2.) the number of elements of the input. As a result, we loop over every element of the input.

Finally, we check the conditions associated with the ReLU activation function for every of those elements. The defintion of ReLU is provided in the Appendix (5.1)

```
    std::vector<fvec> MLPMath::applyReLU(std::vector<
        fvec>& input) {

    std::vector<fvec> result(input.size());

    for (std::size_t iv = 0; iv < fvecLen; iv++) {
        for (std::size_t i = 0; i < input.size(); i++)
            {
            if (input[i][iv] > 0.0f) {
                result[i][iv] = input[i][iv];
            }
            else {
                result[i][iv] = 0.0f;
            }
        }
    }

    return result;
}
```

File 1: SIMD-ized ReLU

Another essential part is the backPropActivation function. Here we provide just the code snippet, which specifically implements ReLU.

In this case there are not many significant changes. We want to ensure, that this function will execute properly if provided with a fvec vector input. For this to work, line 4 is the most essential one to understand. Here the fvec-type overloads the "¿"-operator. To understand the underlying functionality, please view the following example.

"rawOutput" contains the elements: [1.0, -1.0, 2.0, -0.5]
"zero" contains the elements [0.0, 0.0, 0.0, 0.0]

Overloading will produce a result like this: $[1.0 > 0.0, -1.0 > 0.0, 2.0 > 0.0, -0.5 > 0]$

This checks the condition for every element and can the abbreviated into the following form: [0xFFFFFFFF, 0x00000000, 0xFFFFFFFF, 0x00000000] (1 - True & 0 - False)

```
1       case 1:{
2           fvec zero = fvec(0.0f);
3           for (std::size_t i = 0; i < rDelta.size(); i
                ++) {
4               rDelta[i] = (rawOutput[i] > zero) * rDelta
                    [i];
5           }
6           break;
7       }
```

File 2: backPropActivation

Finally we need to incorporate our SIMD-ized ReLU function into the apply-Activation function (Line 10-14).

```
 1      std::vector<fvec> MLPNet::applyActivation(std::
            vector<fvec>& input) {
 2
 3      std::vector<fvec> output(input.size());
 4      switch (activationType_) {
 5          case 0: {// TanH
 6              output = MLPMath::applyTanH(input);
 7              return output;
 8              break;
 9          }
10          case 1: {// SIMD-ized ReLU
11              output = MLPMath::applyReLU(input);
12              return output;
13              break;
14          }
15          default: {
16              return output;
17              break;
18          }
19      }
20  }
```

File 3: applyActivation

3

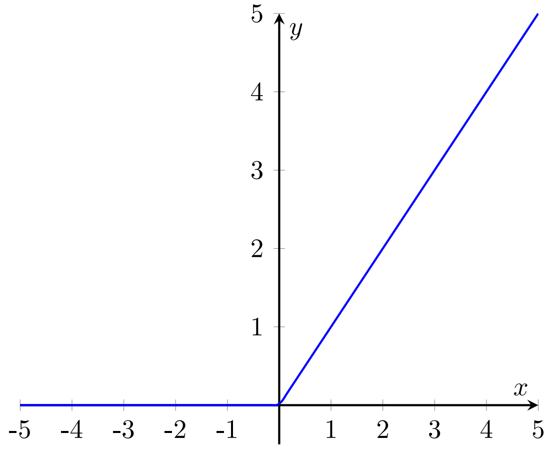
2 Matrix

3 Quadratic Equation

4 CheckSum

5 Appendix

5.1 ReLU



$$\text{ReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Figure 1: Add caption