A Fast, Streaming SIMD Extensions 2, Logistic Squashing Function

J. J. Milner  
cplusplus@hotmail.co.uk  
A. J. Grandison  
A.J.Grandison@gre.ac.uk  
School of Computing and Mathematical Sciences, University of Greenwich, Greenwich, London SE10 9SL, U.K.

Schraudolph proposed an excellent exponential approximation providing increased performance particularly suited to the logistic squashing function used within many neural networking applications. This note applies Intel’s streaming SIMD Extensions 2 (SSE2), where SIMD is single instruction multiple data, of the Pentium IV class processor to Schraudolph’s technique, further increasing the performance of the logistic squashing function. It was found that the calculation of the new 32-bit SSE2 logistic squashing function described here was up to 38 times faster than the conventional exponential function and up to 16 times faster than a Schraudolph-style 32-bit method on an Intel Pentium D 3.6 GHz CPU.

1 Background

The logistic squashing function is used to give a smooth transition between 0 and 1 and is extensively used for computing activation functions within neural networks:

\[ f(y) = \frac{1}{1+e^{-y}}, \quad f(y) \rightarrow 0, \quad y \rightarrow -\infty \]
\[ f(y) \rightarrow 1, \quad y \rightarrow \infty. \]  

(1.1)

The calculation of the exponential is computationally expensive and can create a speed bottleneck in an application.

2 Algorithm

The algorithm is based on the fast exponential approximation technique of Schraudolph (1999), later refined by Cawley (2000), but manipulates the
Figure 1: Function to implement 32-bit Schraudolph-style approximate exponentiation.

```c
#define EXPAF (8388608/0.6931471806f)
float ExpFloat32(float y)
{
    union
    {
        float f;
        int i;
    }eco;
    eco.i = (int) (EXPAF * (y)) + 1065353216;
    return eco.f;
}
```

Contents of a 32-bit single precision float instead of a 64-bit double precision number:

\[
i = \frac{2^a y}{\ln 2} + 2^a (\text{bias}),
\]

(2.1)

where \( y \) is the input parameter (float), \( i \) is the output (integer), \( a = 23 \), and \( \text{bias} = 127 \). When \( i \) is reinterpreted as a float, the approximate exponentiation is given:

With \( y = 5.5450001 \)

32-bit integer \( i = 1132459904 \)

Reinterpreting the integer bits as a 32-bit float \( = 255.966796875 \)

This can be implemented as shown in Figure 1 and will be referred to as the Schraudolph-style 32-bit exponential:

The 32-bit implementation’s precision is comparable, to five significant features, to the original 64-bit version as the mantissa is represented by 23 bits compared to the 20 bits used in Schraudolph’s original 64-bit method. The addressable range for the input parameter \( y \) is reduced to a range of about \(-87\) to \(87\) compared to Schraudolph’s original method, which had a range of about \(-700\) to \(700\). However, this range is adequate for use with the logistic squashing function. The use of SSE2 instructions (Intel, 2006) allows
the vector (SIMD) processing capabilities (simultaneously processing four elements) of the Intel Pentium IV processor to be utilized. A 32-bit representation was chosen to maximize the parallel potential of the Pentium IV CPU. A 64-bit representation could have been used, but only two elements could have been calculated in parallel.
3 The SSE2 Logistic Squashing Function Source Code

The source code for Microsoft Windows, compiled using Visual Studio 2003, for the SSE2 logistic squashing function, is given in Figure 2 and can be downloaded from Grandison (2006). The squashing function incorporates the exponential calculation as described above.

The alignment of memory on a 64-bit boundary favors the Pentium IV cache line size. If unaligned memory is used, then the code can use slightly slower operations to perform the necessary loading and storing of variables. The function can also be bounded to ensure there is no overflow of the exponential calculation. Generally it should be safe to use the fast version of the SSE2 logistic squashing function if care is taken in implementing the source code (e.g., aligned memory is used to store the neuron inputs).

4 Performance

Comparative function timings for calculating the logistic squashing function using the floating point unit (FPU) exponential function, the Schraudolph-style 32-bit exponential (see Figure 1), and the SSE2 squashing function (see Figure 2) for increasing input array sizes are presented in Figure 3 and Table 1 for an Intel Pentium D 3.6 GHz CPU running Windows XP Professional. The timings are given in CPU cycles and were averaged from 10 experiments.
Table 1: Comparative Timings for Calculating the Logistic Squashing Function.

<table>
<thead>
<tr>
<th>Array Size</th>
<th>FPU Exponential Function</th>
<th>Schraudolph-Style 32-Bit Exponential Function</th>
<th>SSE2 Squashing Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5010</td>
<td>1515</td>
<td>3200</td>
</tr>
<tr>
<td>40</td>
<td>23,572</td>
<td>7884</td>
<td>5607</td>
</tr>
<tr>
<td>400</td>
<td>125,028</td>
<td>54,533</td>
<td>7524</td>
</tr>
<tr>
<td>4000</td>
<td>1,227,309</td>
<td>532,766</td>
<td>29,106</td>
</tr>
<tr>
<td>40,000</td>
<td>12,257,082</td>
<td>5,283,083</td>
<td>309,360</td>
</tr>
<tr>
<td>400,000</td>
<td>121,358,934</td>
<td>51,010,335</td>
<td>3,125,288</td>
</tr>
<tr>
<td>4,000,000</td>
<td>1,215,877,343</td>
<td>516,893,465</td>
<td>31,993,422</td>
</tr>
</tbody>
</table>

Neural networks typically contain a high number of neurons, and this is where the best gains will be observed; however, speed-up can be observed for input arrays as small as 40 elements. By examination of the larger array sizes (greater than 4000), it can be seen that the SSE2 squashing function is 38 times faster than using the exponential calculation of the FPU and 16 times faster than using the Schraudolph-style 32-bit exponential float method. The original Schraudolph 64-bit method is about 5% slower than the 32-bit Schraudolph-style method tabulated in Table 1. The “safe” version of the SSE2 squashing function (i.e., using bounded input and unaligned memory) was found to be about 60% slower than the “fast” version of the SSE2 squashing function. The “safe” SSE2 squashing function is still considerably faster than the alternative methods. Similar timing behavior was also exhibited on a range of Pentium IV computers with the SSE2 logistic squashing function exhibiting significant speed-ups compared to the alternative methods.

5 Conclusion

It has been shown in this note that significant performance gains can be made by using SSE2 instructions for calculating the logistic squashing function, which is 16 times faster than Schraudolph-style 32-bit fast exponential approximation on a Pentium D 3.6 GHz CPU.

Neural networks containing layers nondivisible by 4 will require modulo operations, calling the Schraudolph approximation for the remaining neurons. Testing has shown the best performance was achieved by extending the number with sentinel values and using SIMD exclusively. For example, a 25-element calculation processed as a 28-element array with trailing zeros is faster than calling another implementation for the remaining values.

This code was compiled for Microsoft Windows using Visual Studio 2003 for the Intel Pentium IV processor. It should be possible to port this code to
a Linux environment, although this has not been attempted here. This code is specific to the Pentium IV class of processor, but equivalent instructions may exist for other CPU types.

References


____________________

Received October 17, 2006; accepted November 10, 2006.