# HLS FOR IMDB SENTIMENT ANALYSIS

GROUP : 24

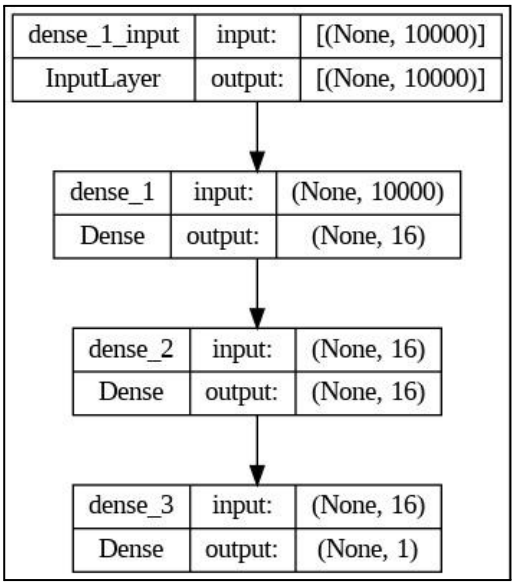
| | |
|---|---|
| Abhrajyoti Kundu | 234101003 |
| Gorachand Mondal | 234101015 |
| Nihalbhai Shaikh | 234101032 |
| Pratik Sisodiya | 234101040 |
| Sayan Pal | 234101048 |


## 1. DESCRIPTION OF THE MODEL:

The sentiment analysis model utilized in this project is based on artificial neural networks (ANN) and is designed to analyze movie reviews from the Internet Movie Database (IMDB) using neural network approaches. The neural network model achieves a higher accuracy compared to the base model, indicating its improved performance in sentiment analysis. The model has been converted to a hardware description language format for hardware synthesis and optimization using Vivado HLS.

## ARCHITECTURE:

| Layer | Description |
|---|---|
| 1 | **INPUT LAYER:** This is the first layer of the network that receives the input. It is defined to have 10,000 input nodes, which means it can take vectors of size 10,000 as input. This layer does not perform any computation; it merely serves as the entry point for data. |
| 2 | **dense_1 (Dense):** This is the first hidden layer in the network. It is a fully connected (Dense) layer with 16 neurons. Each neuron in this layer is connected to all 10,000 nodes of the input layer. This layer will take the input, apply a weighted sum across the inputs, add a bias, and then possibly follow it with a non-linear activation function. |
| 3 | **dense_2 (Dense):** This is the second hidden layer, also a fully connected (Dense) layer with 16 neurons. It takes the output from the first dense layer (which would be a 16-element vector) as its input and processes it in the same manner, again applying a weighted sum and an activation function. |
| 4 | **dense_3 (Dense):** This is the output layer of the network with a single neuron. It takes the output from the second dense layer and outputs a single value. The single neuron suggests that this network is likely designed for a regression task or a binary classification task. |




The trained Keras model has been converted into a C file using the Keras2c tool. This C file is then synthesized and optimized using the Vivado HLS 2018.2 tool for hardware implementation. To validate the efficiency and effectiveness of our hardware implementation, we compare the performance metrics and resource utilization with another implementation generated by the HLS4ML tool.

## 2. Changes made to make Keras2c generated files synthesizable and a brief description of the changes made.

1. **Memory Allocation:**
   - **Static Allocation Replacement:**
     Dynamic memory allocation, typically unsupported in Vivado HLS due to its hardware synthesis limitations, was replaced with static allocation. This change ensures that memory usage is static and known at compile time, facilitating hardware synthesis.

```
// size_t k2c_sub2idx(const size_t * sub, const size_t * shape, const size_t ndim)
size_t k2c_sub2idx(const size_t sub[], const size_t shape[], const size_t ndim)
```

2. **Specific Changes Made to the Functions:**
   - **Replaced memset and memcpy: k2c_matmul & k2c_affine_matmul**
     **a.** Replaced the memset function with manual loop initialization to set the output matrices to zero. This change ensures compatibility with Vivado HLS.

```
// memset(C, 0, outrows*outcols*sizeof(C[0]));
for (i = 0; i < outrows * outcols; ++i)
{
    C[i] = 0;
}
```

**b.** Removed the use of memcpy in k2c_flatten1, replacing it with a manual loop to copy elements from the input to the output array. This adjustment simplifies memory management and improves efficiency.

```
// memcpy(output->array, input->array, input->numel*sizeof(input->array[0]));
for (i = 0; i < input_numel; ++i)
{
    output_array[i] = input_array[i];
}
```

- **Introduced Flags for Activation Functions: k2c_dense1:**

Introduced flags (flag) to control activation functions, providing flexibility in choosing between ReLU and sigmoid activations. This modification enhances the function's versatility.

```
if (flag)
    k2c_relu_func(output_array, outsize);
else
{
    k2c_sigmoid_func(output_array, outsize);
}
```

- **Changed Dynamic Arrays to Static Arrays: k2c_matmul & k2c_affine_matmul:**

Both functions changed dynamic arrays to static arrays. This adjustment simplifies memory management and improves performance in certain contexts.

```
// void k2c_matmul(float * C, const float * A, const float * B, const size_t outrows,
//                 const size_t outcols, const size_t innerdim)
void k2c_matmul(float C[], const float A[], const float B[], const size_t outrows,
                const size_t outcols, const size_t innerdim)
```

- **Optimized Memory Accesses and Introduced Temporary Storage Arrays: k2c_dot:**

Reordered memory accesses and optimized calculation of inner and outer dimensions to improve performance. Additionally, temporary storage arrays (reshapeA and reshapeB) were introduced to reshape input arrays, enhancing memory access patterns and simplifying code logic. These optimizations improve overall efficiency.

- **Global Variable Declaration:**

Due to multiple redeclarations of loop index variables (i, j, k) across different functions, they were consolidated into one global variable with the type size_t. This adjustment simplifies code maintenance and reduces the likelihood of naming conflicts.

```
#include <math.h>
#include <string.h>

size_t i, j, k;
```

- **Converted and broken down K2c Tensor Struct into separate variables:**

```
// k2c_tensor dense_1_bias = {&dense_1_bias_array[0],1,16,{16, 1, 1, 1, 1}};
// k2c_tensor dense_1_bias;
size_t dense_1_bias_ndim = 1;
size_t dense_1_bias_numel = 16;
size_t dense_1_bias_shape[] = {16, 1, 1, 1, 1};
```

```
// k2c_tensor dense_2_output = {&dense_2_output_array[0],1,16,{16, 1, 1, 1, 1}};

// k2c_tensor dense_2_output;
size_t dense_2_output_ndim = 1;
size_t dense_2_output_numel = 16;
size_t dense_2_output_shape[] = {16, 1, 1, 1, 1};
```

## 3. Changes made to generate HLS4ML report

### Downgraded Tensorflow version from 2.16.0 to 2.14.0

HLS4ML is not updated to incorporate Tensorflow 2.16.0 at the backend , hence we had to downgrade our Tensorflow version to 2.14.0 to ensure HLS4ML works fine.

### Changed configuration details for the model.

```
# Configure HLS4ML (e.g., specify target FPGA platform, optimizations, constraints)
config['Model']['Precision'] = 'ap_fixed<6,2>'
config['Model']['Strategy'] = 'resource'
config['Model']['ReuseFactor'] = 250
```

- The model utilizes a precision of 'ap_fixed<6,2>', indicating fixed-point arithmetic with 6 total bits and 2 bits allocated for the fractional part, enabling representation of numbers within a range from -2 to just under 2.
- The optimization strategy employed during synthesis is 'resource', emphasizing the minimization of hardware resources such as LUTs, FFs, and DSPs. This was done to minimise the hardware resource.
- Additionally, a reuse factor of 250 is specified, indicating extensive reuse of hardware resources to optimize performance while reducing consumption.
- config_schedule -enable_dsp_full_reg=false : DSP blocks won't be able to utilize full register files, due to resource constraints or optimization considerations.

```
config_compile -name_max_length 80
set_part $part
#config_schedule -enable_dsp_full_reg=false
create_clock -period $clock_period -name default
set_clock_uncertainty $clock_uncertainty default
```

**Removed Pragmas**

1. **#pragma HLS ARRAY_RESHAPE :-**

```
//#pragma HLS ARRAY_RESHAPE   variable=weights block factor=block_factor
```

Our precision was set to <6,2>, resulting in a bit width of 60000. However, to facilitate vertical mapping, we require double the bit width, which exceeds the maximum limit of 65536. Therefore, it was removed.

2. **#pragma HLS ARRAY_PARTITION**

```
// #pragma HLS ARRAY_PARTITION variable=biases complete

 typename CONFIG_T::accum_t acc[CONFIG_T::n_out];
 //#pragma HLS ARRAY_PARTITION variable=acc complete
```

We decided to remove HLS array partitioning due to resource constraints. Our system could not support the increased demand for resources that array partitioning requires, including multiple small memories or registers instead of one large memory. This decision was made to ensure that our design remained within resource limits and could be effectively synthesized.

3. **#pragma HLS UNROLL:** Removed from InitAccum, MultLoop, Result, AccumLoop2, ResetMult

```
ResetMult:
    for (int imult = 0; imult < multiplier_limit; imult++) {
      //  #pragma HLS UNROLL
        mult[imult] = 0;
    }
```

The Array Unroll pragma, which allows loops to be fully or partially unrolled to increase data access and throughput, was removed due to resource constraints. Unrolling loops creates multiple copies of the loop body in the RTL design, potentially exceeding available resources. Removing Array Unroll ensured our design remained within resource constraints for effective synthesis.

## 4. In a markdown cell of jupyter notebook, mention all the issues that are faced(dependencies and versions) and solutions to resolve.

**Ans:** Present here

**5. Optimizations: For each optimization applied (pragma), justify why it has been used.**

**Ans:**

1. **Loop Unroll:**
   Loop unrolling with factor 2 is used to improve performance by duplicating loop bodies, effectively reducing loop overhead and allowing for better optimization. Here, the factor is being used as complete unroll was generating error for long runtime.

```
for (i=0; i < max_size; ++i) {
    #pragma HLS unroll factor=2
    if(i<size){
        if (x[i] <= 0.0f) {
            x[i] = 0.0f;
        }
    }

}
```

2. **Removed variable loop bound:**
   Loop unrolling requires fixed bound and in code, there were many variable bounds for loop. So we took maximum 512 size of fixed bound for loop and ran loop for variable size using conditional statements.

```
for (i = 0; i < max_size; ++i) {
    #pragma HLS unroll factor=2
    if(i<input_numel){output_array[i] = input_array[i];}
}
```

3. **Loop Pipelining:**
   Loop Pipelining is used to improve performance by parallel execution and better resource utilization. Loop pipelining allows multiple iterations of a loop to execute concurrently, reducing the latency between iterations. This leads to a higher throughput, enabling more data to be processed in a given time frame.

```
#pragma HLS pipeline
for(i2=0;i2<4;i2++)
{
    const size_t outrowidx = i*outcols;
    const size_t inneridx = i*innerdim;

    for(k2=0;k2<4;k2++)
    {
        size_t y = inneridx+k2;
        size_t z = outcols*k2;

        for(j2=0;j2<4;j2++)
        {
            size_t x = outrowidx+j2;
            size_t a = z+j2;
            C[x] += A[y]*B[a];
        }
    }
}
```

4. **Loop tiling:**
   Loop tiling involves partitioning or dividing the iterations of a loop into smaller, contiguous blocks called tiles. Instead of processing the entire loop body at once, the loop is divided into smaller sections and loop tiling can also facilitate parallelism and pipelining in hardware designs. By breaking down the loop into smaller, independent tiles, it becomes easier to exploit parallel execution and pipeline stages efficiently.

```
size_t i1,j1,k1,i2,j2,k2;
for (i1 = 0 ; i1 < outrows/4; ++i1) {
    for (k1 = 0; k1 < innerdim/4; ++k1) {
        for (j1 = 0;  j1 < max_size; ++j1)
        {
            #pragma HLS unroll factor=2
            if(j < outcols)
            {
            #pragma HLS pipeline
            for(i2=0;i2<4;i2++)
            {
                const size_t outrowidx = i*outcols;
                const size_t inneridx = i*innerdim;
                for(k2=0;k2<4;k2++)
                {
                    size_t y = inneridx+k2;
                    size_t z = outcols*k2;
```

```
                    for(j2=0;j2<4;j2++)
                    {
                        size_t x = outrowidx+j2;
                        size_t a = z+j2;
                        C[x] += A[y]*B[a];
                    }
                }
            }
            }
        }
    }
}
```

**6. Results:**

• **Latency and area overhead table for Baseline (Unoptimized).**

# Result

| | | Latency | | | Interval | | |
|---|---|---|---|---|---|---|---|
| RTL | Status | min | avg | max | min | avg | max |
| VHDL | NA | NA | NA | NA | NA | NA | NA |
| Verilog | Pass | 1451814 | 1451814 | 1451814 | NA | NA | NA |

| Name | BRAM_18K | DSP48E | FF | LUT |
|---|---|---|---|---|
| DSP | - | - | - | - |
| Expression | - | - | 0 | 222 |
| FIFO | - | - | - | - |
| Instance | 1 | 43 | 9522 | 10752 |
| Memory | 1536 | - | 128 | 16 |
| Multiplexer | - | - | - | 510 |
| Register | - | - | 284 | - |
| Total | 1537 | 43 | 9934 | 11500 |
| Available | 730 | 740 | 269200 | 129000 |
| Utilization (%) | 210 | 5 | 3 | 8 |

**Latency and area overhead table for Optimized (if there are multiple versions like various tradeoffs, give all of them).**

## Performance Estimates

### Timing (ns)

#### Summary

| Clock | Target | Estimated | Uncertainty |
|---|---|---|---|
| ap_clk | 5.00 | 5.496 | 0.62 |

### Latency (clock cycles)

#### Summary

| Latency | | Interval | | |
|---|---|---|---|---|
| min | max | min | max | Type |
| ? | ? | ? | ? | none |

## Utilization Estimates

### Summary

| Name | BRAM_18K | DSP48E | FF | LUT |
|---|---|---|---|---|
| DSP | - | - | - | - |
| Expression | - | - | 0 | 287 |
| FIFO | - | - | - | - |
| Instance | 0 | 13 | 4913 | 23081 |
| Memory | 20 | - | 64 | 8 |
| Multiplexer | - | - | - | 883 |
| Register | - | - | 470 | - |
| Total | 20 | 13 | 5447 | 24259 |
| Available | 730 | 740 | 269200 | 129000 |
| Utilization (%) | 2 | 1 | 2 | 18 |

# Result

| | | Latency | | | Interval | | |
|---|---|---|---|---|---|---|---|
| RTL | Status | min | avg | max | min | avg | max |
| VHDL | Pass | 162506 | 162506 | 162507 | 162506 | 162506 | 162507 |
| Verilog | NA | 162506 | 162506 | 162506 | NA | NA | NA |

**HLS4ML generated Latency and area overhead table.**

+ Latency (clock cycles):
    * Summary:

| Latency | | Interval | | Pipeline |
|---|---|---|---|---|
| min | max | min | max | Type |
| 15901 | 15902 | 15302 | 15302 | dataflow |

* Summary:

| Name | BRAM_18K | DSP48E | FF | LUT | URAM |
|---|---|---|---|---|---|
| DSP | - | - | - | - | - |
| Expression | - | - | 0 | 274 | - |
| FIFO | 0 | - | 325 | 1170 | - |
| Instance | 184 | 0 | 3667511 | 1509923 | - |
| Memory | - | - | - | - | - |
| Multiplexer | - | - | - | 576 | - |
| Register | - | - | 64 | - | - |
| Total | 184 | 0 | 3667900 | 1511943 | 0 |
| Available | 5376 | 12288 | 3456000 | 1728000 | 1280 |
| Utilization (%) | 3 | 0 | 106 | 87 | 0 |

**Finally, a comparison report of both Optimized and HLS4ML generated reports.**

| Design | Normal | HLS4ML | Manual (Vivado) |
|---|---|---|---|
| | | | |
| LUT | 11500 | 1511943 | 24259 |
| FF | 9934 | 3667900 | 5447 |
| DSP | 43 | 0 | 13 |
| BRAM | 1537 | 184 | 20 |
| Latency (min / max) | 1451814 / same | 15901 | 162508 / same |
| Clock period | 4.6 / 5 | 4.3 / 5 | 5.49 / 5 |