HLS for Neural Network Classifier

Group Number: 29
Navdeep Patel - 234101031

Ujjawal Mawar - 234101053

Kuldeep Patel - 234156009

Harshita Mishra -234101016

Nupur Brahamanya - 234101034

1. Description of Model

Task: The model takes an input array, feeds it through multiple dense layers with ReLU activations, and produces a final classification output using a sigmoid activation function.

Number of Layers: There are 4 layers in the model where each layer performs a dense operation, followed by a ReLU activation except for the last layer, which applies a sigmoid activation.

Types of Layers: This model constitutes the layers of an Artificial Neural Network, specifically a feedforward network with fully connected layers and activation functions, designed for classification tasks.

1. InputLayer: Performs the initial transformation of input data (dense_13_input_input_array) using weights(dense_13_kernel_array) and biases (dense_ 13_bias_array), applying the ReLU activation function to enhance nonlinearity.
1. HiddenLayer1: Refines the feature representation obtained from previous layer using ReLU activation.

2. HiddenLayer2: Continues to refine feature representation by applying further nonlinear transformations using ReLU activation.
3. OutputLayer: Produces the final classification output by applying a sigmoid activation function to the learned representations, mapping them to a probability distribution over classes.

Other Details: The layers contain hidden neurons which contribute to learning complex representations of the input data, which are crucial for the model's classification task. The hidden neurons in each layer are as follows:
Layer 1 - 128, Layer 2 – 32, Layer 3 – 16.

2. Changes made to keras2c generated files synthesizable.

- Files produced from keras2c variables named were changed in such manner "str1"+input_input+"str2". We changed them with corresponding correct variable names.
- Multiple definitions of variables were removed.
- Dynamic allocation of k2c_tesnor structure was removed by fixing the size of the array and then manual allocation of values using loops and initialize all the parameters of structure.
- An error of segmentation fault / SIGSEGV was removed by making the arrays of size > 10^6 global.
- Included the path of files provided in include folder of keras2c into the code.
- Function pointers were removed. (In function calling of k2c_dense() function pointer is passed)
- Encountered an error stating "unsupported machine mode '__TC__' typedef _Complex float __cfloat128" which was resolved by changing the c file into cpp and includes all the corresponding C libraries in files
- We changed memset and memcopy to static array memory.

3. Changes made to generate HLS4ML report.

We did not remove any pragma for generating HLS4ML report.

We made some changes in HLS4ML python script these are :

```
config = hls4ml.utils.config_from_keras_model(model, granularity='name')

for Layer in config['LayerName'].keys():
    config['LayerName'][Layer]['Precision'] = 'ap_fixed<8,2>'   #balancing between acuracy and resource usage
    config['LayerName'][Layer]['Pipeline'] = 'None'
    config['LayerName'][Layer]['Strategy'] = 'resource'
    config['LayerName'][Layer]['ReuseFactor'] = 15
#  how many times the resources within a layer can be reused during computation
```

These changes balance between the accuracy and resource usage

Before this optimization HLS4ML runs for long time and run out of system memory which is solved by this line.

After running ls_mode.compile() we have to open build_project.tcl and make comment Config_schedule -enable_dsp_full_reg=false.

5.Optimizations:

```
//float dense_15_output_array[16] = {0};
// k2c_tensor dense_15_output = {&dense_15_output_array[0],1,16,{16, 1, 1, 1, 1}};

for(i=0;i<16;i++)
{
#pragma HLS UNROLL factor = 4
    //dense_15_output.array[i]= dense_15_output_array[i];
    dense_15_output.array[i]= 0.0;
}
dense_15_output.ndim=1;
dense_15_output.numel= 16;
dense_15_output.shape[0]=16;
dense_15_output.shape[1]=1;
dense_15_output.shape[2]=1;
dense_15_output.shape[3]=1;
dense_15_output.shape[4]=1;
```

Fig. 5.1

1. We manually broke down struct initializations and observed that dense_15_output_array[16] is initialized with a for loop. To reduce memory usage, we deleted all unused arrays in the code. (refer Fig 5.1)

```
k2c_dense(&dense_13_output,dense_13_input_input,&dense_13_kernel,
    &dense_13_bias,0,dense_13_fwork);
```

Fig. 5.2

2. These main functions are called in our solution. We identified arrays used only for reading and made them const to prevent mapping to block RAM, ensuring they are mapped to ROM only (refer to fig.5.2)

```
const float dense_14_bias_array[32] = {
```

3. In our solution, many loops are used for struct initialization. We unroll them with factors {4, 8, 16} and perform loop merging

First Report we synthesized:

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	1509
FIFO	-	-	-	-
Instance	0	334	57825	42590
Memory	427	-	128	16
Multiplexer	-	-	-	3253
Register	-	-	1626	-
Total	427	334	59579	47368
Available	730	740	269200	129000
Utilization (%)	58	45	22	36

Fig. 5.3

We observed BRAM are DSP48E are used in significant amount because we are using the fixed size arrays in structure, and we choose the size to be the largest array size

```
struct k2c_tensor
{
    /** Pointer to array of tensor value
    float  array[10000];

    /** Rank of the tensor (number of di
    size_t ndim;

    /** Number of elements in the tensor
    size_t numel;

    /** Array, size of the tensor in eac
    size_t shape[K2C_MAX_NDIM];
};
```

Fig.5.4

4. We removed the structure and initialized single variables and arrays for parameters. (refer to Fig5.4 and Fig.5.5)

```
float dense_16_bias_array[1] = {
-3.89912687e-02f,};
size_t dense_16_bias_dim = 1;
size_t dense_16_bias_numel = 1;
size_t dense_16_bias_shape[5] = {1, 1, 1, 1, 1};
```

Fig.5.5

5. In our solution, the main function used is k2c_dense, which is calls k2c_affine_matmul after tracking the parameters of arrays. We analyzed the array access pattern inside k2c_affine_matmul and partitioned the array while unrolling the loop with a perfect factor of 4.

```
for (size_t i = 0; i < outrows; ++i) {
    const size_t outrowidx = i * outcols;
    const size_t inneridx = i * innerdim;

    for (size_t j = 0; j < outcols; ++j) {
        size_t temp = outrowidx + j;
        float sum = d[j];  // Initialize sum with the bias value
        for (size_t k = 0; k < innerdim; ++k) {
            #pragma HLS PIPELINE II=1
            float A_val = A[inneridx + k];  // Load A value
            float B_val = B[k * outcols + j];  // Load B value
            sum += A_val * B_val;  // Accumulate the result
        }
        C[temp] = sum;  // Assign the accumulated value to C
    }
}
```

Fig.5.6

Our results before optimization:

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	Pass	47200	47200	47200	NA	NA	NA
Verilog	NA	NA	NA	NA	NA	NA	NA

Export the report(.html) using the Export Wizard

After Optimization:

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	Pass	37944	37944	37944	NA	NA	NA
Verilog	NA	NA	NA	NA	NA	NA	NA

6. Same Optimization we applied in matmul but results reversed.
7. We use loop unrolling in k2c_bias_add we analyzed that bias array is max size of 128

```
void k2c_bias_add(float A_array[],size_t A_dim, size_t * A_numel, size_t A_shape[],
                  float b_array[],size_t b_dim, size_t * b_numel, size_t b_shape[]) {

    #pragma HLS array_partition variable=A_array cyclic factor=8 dim=1
    #pragma HLS array_partition variable=b_array cyclic factor=8 dim=1
    for (size_t i=0; i<*A_numel; i+=*b_numel) {
        for (size_t j=0; j<*b_numel; ++j) {
            #pragma HLS unroll factor= 4
            A_array[i+j] += b_array[j];
        }
    }


}
```

8. We further analyzed the code for optimizations by commenting out functions one by one and checking the results. We found that the costliest functions in the solution are k2c_affine_matmul and k2c_matmul. We attempted to use loop tiling in matrix multiplication, which resulted in significant improvements.

   After applying loop tiling and array partition together, along with pipelining, we also observed a reduction in BRAMs.

```
    #pragma HLS array_partition variable=reshapeA cyclic factor=8 dim=1
    #pragma HLS array_partition variable=reshapeB cyclic factor=8 dim=1
    size_t i , k , j , ii , kk , jj;
    for(i=0;i<free_axesA/4;i++)
    {
        for(k=0;k<prod_axesA/4;k++)
        {
            for(j=0;j<free_axesB/4;j++)
            {
#pragma HLS pipeline
                for(ii=0;ii<4;ii++)
                {
                    const size_t outrowidx = ii * free_axesB;
                    const size_t inneridx = ii * prod_axesA;
                    for(kk=0;kk<4;kk++)
                    {

                        size_t y = inneridx+kk;
                        size_t z = free_axesB*kk;
                        for(jj=0;jj<4;jj++)
                        {
                            size_t x = outrowidx+jj;
                            size_t a = z+jj;
                            C_array[x] += reshapeA[y] * reshapeB[a];
                        }
                    }
                }
            }
        }
    }


    }
}
```

Before Optimization:

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	Pass	37944	37944	37944	NA	NA	NA
Verilog	NA	NA	NA	NA	NA	NA	NA

After optimization:

Cosimulation Report for 'vlsi_proj'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	Fail	993	993	993	NA	NA	NA
Verilog	NA	1013	1013	1013	NA	NA	NA

Export the report(.html) using the Export Wizard

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	26
FIFO	-	-	-	-
Instance	9	87	39303	42873
Memory	1	-	1536	88
Multiplexer	-	-	-	1052
Register	-	-	18	-
Total	10	87	40857	44039
Available	730	740	269200	129000
Utilization (%)	1	11	15	34

6. Results.

Board we have used: ARTIX7(virtexuplus)

Target device: xcvu13p-flga2577-2-e

Design	LUT	FF	DSP	BRAM	Latency	Clock period
Unoptimized	41821	52047	238	427	47200(min) 47200(max)	5
Resource Optimized	60785	19800	21	26 (Instance)	37944(min) 37944(max)	5
Latency Optimized	44039	40857	87	10	993(min) 993(max)	5
HLS4ML	94692	13539	0	123	30(min) 31(max)	5

7. FINAL COMPARISION

```
+ Latency (clock cycles):
    * Summary:
    +-----+-----+-----+-----+----------+
    |  Latency  |  Interval | Pipeline |
    | min | max | min | max |   Type   |
    +-----+-----+-----+-----+----------+
    |   30|   31|   19|   20| dataflow |
    +-----+-----+-----+-----+----------+
```

THIS IS HLS ML GENERATED LATENCY

Cosimulation Report for 'vlsi_proj'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	Fail	993	993	993	NA	NA	NA
Verilog	NA	1013	1013	1013	NA	NA	NA

THIS IS THE FINAL OPTIMIZATION WE COULD REACH SO FAR.

These latencies are based on same board and 5ns clock period which is **ARTIX7 Virtex Plus**

We cannot achieve final latency because of data flow optimization. We tried to implement data flow optimization in main functions. We can easily figure out the single producer consumer and feed forward dependency, but we got some errors.

ERROR: dense_13_oupout_array cannot be read and write operation together in k2c_dense instance.

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-------+---------+---------+------+
|       Name      | BRAM_18K| DSP48E|    FF   |   LUT   | URAM |
+-----------------+---------+-------+---------+---------+------+
|DSP              |        -|      -|        -|        -|     -|
|Expression       |        -|      -|        0|        2|     -|
|FIFO             |        0|      -|     1765|     7060|     -|
|Instance         |      123|      0|    11774|    87630|     -|
|Memory           |        -|      -|        -|        -|     -|
|Multiplexer      |        -|      -|        -|        -|     -|
|Register         |        -|      -|        -|        -|     -|
+-----------------+---------+-------+---------+---------+------+
|Total            |      123|      0|    13539|    94692|     0|
+-----------------+---------+-------+---------+---------+------+
|Available        |     5376|  12288|  3456000|  1728000|  1280|
+-----------------+---------+-------+---------+---------+------+
|Utilization (%)  |        2|      0|    ~0   |        5|     0|
+-----------------+---------+-------+---------+---------+------+
```

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	26
FIFO	-	-	-	-
Instance	9	87	39303	42873
Memory	1	-	1536	88
Multiplexer	-	-	-	1052
Register	-	-	18	-
Total	10	87	40857	44039
Available	730	740	269200	129000
Utilization (%)	1	11	15	34

We have optimized the resources BRAM and LUT as required but DSP48E can be optimized because of operations like % and exp is used