Simple GNN HLS Implementation

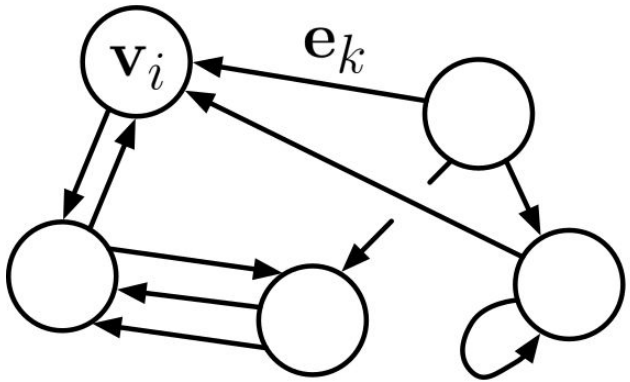
Vesal Razavimaleki, Javier Duarte

IRIS-HEP GNN Meeting

July 30, 2020

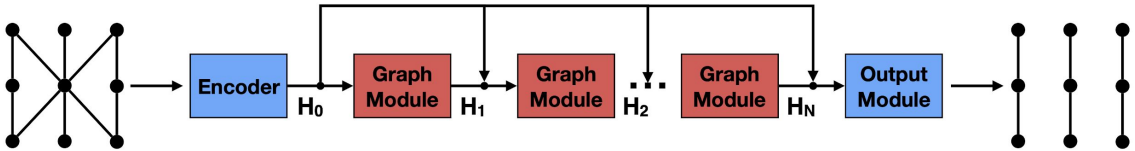
Input Graph Structure

- Graphs represent ⅛ phi-slice of a tracking detector with node attributes, edge attributes, and receivers and senders (adjacency matrices in sparse rep.)
- Nodes: hits in a layer
- Edges: connections between hits on adjacent layers based on geometrical restrictions
- Node features: cylindrical coordinates (r, φ, z)
- Edge features: difference of coordinates (Δη, Δφ)
- Edge labels:
  - 1 if two hits come from same track
  - 0 if not



Segment Classifier Model

- Encoder: two “GraphIndepenent” fully-connected 2-layer networks transform node and edge features to latent representations, independently
- Core: “InteractionNetwork” fully-connected 2-layer networks update edge and node features (with aggregation of “messages” using senders and receivers)
    - Features from first encoding and previous iteration are concatenated as input to the following iteration
- Decoder: two “GraphIndependent” layers transform edge features into single edge classification output



Simple graph_nets Implementation

- 1 input graph: 100 nodes, 130 edges (configurable)
- 3 node features, 4 edge features
- 32 latent dimensions (configurable)
- 1 iteration of the IN (for now)
- Randomized weights (just for testing)

- GNN Python Implementation Script:
  https://github.com/jmduarte/exatrkx-neurips19/blob/master/gnn-tracking/scripts/make_graph_module.py

```
LATENT = 32
encoder = gn.modules.GraphIndependent(
   edge_model_fn=lambda: snt.Sequential([snt.nets.MLP([LATENT, LATENT], activation=tf.nn.relu,
activate_final=True)]),
   node_model_fn=lambda: snt.Sequential([snt.nets.MLP([LATENT, LATENT], activation=tf.nn.relu,
activate_final=True)]),
   global_model_fn=None)
```

```
core = gn.modules.InteractionNetwork(
   edge_model_fn=lambda: snt.Sequential([snt.nets.MLP([LATENT, LATENT], activation=tf.nn.relu,
activate_final=True)]),
   node_model_fn=lambda: snt.Sequential([snt.nets.MLP([LATENT, LATENT], activation=tf.nn.relu,
activate_final=True)]),
   reducer=tf.unsorted_segment_sum)
```

```
decoder = gn.modules.GraphIndependent(
   edge_model_fn=lambda: snt.Sequential([snt.nets.MLP([LATENT, LATENT, LATENT, 1],
activation=tf.nn.relu, activate_final=False),
                                         tf.sigmoid]),
   node_model_fn=None,
   global_model_fn=None)
```

HLS Implementation

https://github.com/vesal-rm/hls4ml/tree/master/example-prjs/graph/gnn_simple

- Inputs
  - Node features: $N_{nodes} \times N_{node\_features}$
  - Edge features: $N_{edges} \times N_{edge\_features}$
  - Receivers: $N_{edges} \times 1$
  - Senders: $N_{edges} \times 1$
- Outputs
  - Edge classifications: $N_{edges} \times 1$
- “GraphIndependent” layers
  - Dense batch matrix multiplication (weights and biases)
  - ReLU activation function
- “InteractionNetwork” layer
  - Retrieve receiver and sender indices
  - Concatenate appropriate edge/node features
  - Dense batch matrix multiplication (weights and biases)
  - ReLU activation function

```
//core edge updates
for(int j = 0; j < N_EDGES; j++){
  index_t r = receivers[j][0];
  index_t s = senders[j][0];
  input_t l_logits[4*latent_dim];
  #pragma HLS ARRAY_PARTITION variable=l_logits complete dim=0
  nnet::merge<input_t, 2*latent_dim, 2*latent_dim>(Ce[j], Cn[r], l_logits);
  input_t l[6*latent_dim];
  #pragma HLS ARRAY_PARTITION variable=l complete dim=0
  nnet::merge<input_t, 4*latent_dim, 2*latent_dim>(l_logits, Cn[s], l);
  input_t L0_logits[latent_dim];
  #pragma HLS ARRAY_PARTITION variable=L0_logits complete dim=0
  nnet::dense_large<input_t, input_t, dense_config5>(l, L0_logits, core_edge_w0, core_edge_b0);
  input_t L0[latent_dim];
  #pragma HLS ARRAY_PARTITION variable=L0 complete dim=0
  nnet::relu<input_t, input_t, relu_config3>(L0_logits, L0);
  input_t L_logits[latent_dim];
  #pragma HLS ARRAY_PARTITION variable=L_logits complete dim=0
  nnet::dense_large<input_t, input_t, dense_config6>(L0, L_logits, core_edge_w1, core_edge_b1);
  nnet::relu<input_t, input_t, relu_config3>(L_logits, L[j]);
}
```

Output Comparison

Python

```
'edges':
array([[0.49879126],
       [0.49851833],
       [0.49911052],
       [0.49695702],
       [0.49892634],
       [0.49947994]])
```

HLS csim

```
e =
0.5
0.5
0.5
0.5
0.5
0.5
```

- Snippets of outputs
- Similar results
- All outputs are ~0.5 due to random weights (will update to realistic weights)

Synthesizability

- Synthesis unable to complete
  - Number of iterations in a loop exceeded the maximum (currently 4096)
  - Options:
    - Increase Reuse Factor
    - *Decrease Input Graph size*

```
ERROR: [XFORM 203-504] Stop unrolling loop 'ReuseLoop' (./nnet_dense_large.h:64) in function 'nnet::dense_large<ap_fixed<16, 6, (ap_q_mode)5, (ap_o_mode)3, 0>, ap_fixed<16, 6, (ap_q_mode)5, (ap_o_mode)3, 0>, dense_config5>' because it may cause large runtime and excessive memory usage due to increase in code size. Please avoid unrolling the loop or form sub-functions for code in the loop body.
ERROR: [HLS 200-70] Pre-synthesis failed.
```