PyTorch Geometric to HLS

An hls4ml add-on for conversion of Graph Neural Networks

https://github.com/abdelabd/hls4ml/tree/pyg_to_hls_rebase
Graph Neural Networks

- \( G = (V, E, \text{edge\_index}) \);
  - \( V[i,j] \) = jth attribute of ith vertex
  - \( E[i,j] \) = jth attribute of ith edge
  - \( \text{edge\_index}[0,i] \) = the index of the sending node for the ith edge
  - \( \text{edge\_index}[1,i] \) = the index of the receiving node for the ith edge

- We can represent arbitrary-dimensional objects as ‘nodes’
- We can represent arbitrary-dimensional relationships as ‘edges’
- Benefits: efficient encoding of context, generalizability, permutation invariance
Learning to Simulate Complex Physics with Graph Networks, DeepMind 2020

Permutation Invariant

[Diagram showing a neural network for feature learning and classification]
See [https://github.com/jmduarte/pytorch_dev_hls4ml/tree/commit_hls](https://github.com/jmduarte/pytorch_dev_hls4ml/tree/commit_hls)
Graph Neural Networks: non-feedforward data flow

← Interaction Network

- https://github.com/GageDeZoort/interaction_network_paper
torch_geometric.nn.conv.MessagePassing

- Message → Aggregate → Update

```python
def forward(self, data):  # x, edge_index, edge_attr:
    x = data.x
    edge_index = data.edge_index
    edge_attr = data.edge_attr

    # Message
    x_i, x_j = x[edge_index[1]], x[edge_index[0]]
    msg_out = self.message(x_i, x_j, edge_attr)  # self.message(edge_index, edge_attr)

    # Aggregate
    index = edge_index[1,:]
    ptr = None
    dim_size = x.shape[0]
    aggr_out = self.aggregate(msg_out, index, ptr, dim_size)

    # Update
    update_out = self.update(aggr_out, x)
    x_tilde = update_out

    # return update_out
    m2 = torch.cat([x_tilde[edge_index[1]],
                    x_tilde[edge_index[0]],
                    self.E], dim=1)
    return torch.sigmoid(self.R2(m2))
```
1. class GraphBlock(hls4ml.model.hls_layers.Layer): packages a torch *module* into a form that will be accepted as an HLS *layer*
2. class EdgeBlock(GraphBlock):
   i. Inputs
      1. Node attributes
      2. Edge attributes
      3. Edge Index
   ii. Function:
      1. For each edge:
         a. \(<\text{edge attributes, receiver-node attributes, sender-node attributes}>\) \rightarrow \text{Neural Network} \rightarrow \text{Edge predictions}
         b. \text{Edge predictions} \rightarrow \text{permutation-invariant aggregation} \rightarrow \text{Aggregate edge predictions}
   iii. Outputs
      1. Edge predictions
      2. Aggregate edge predictions
3. class NodeBlock(GraphBlock):
   i. Inputs
      1. Node attributes
      2. Aggregate edge attributes (or aggregate messages)
   ii. Function:
      1. For each node:
         a. \(<\text{node attributes, aggregate attributes}>\) \rightarrow \text{Neural Network} \rightarrow \text{Node predictions}
   iii. Outputs
      1. Node predictions
4. class Aggregate(hls4ml.model.hls_layers.Layer):
   a. \text{Edge attributes} \rightarrow \text{permutation-invariant aggregation} \rightarrow \text{Aggregate edge attributes}
Interaction Network Benchmarks
User-inputs

```python
def pyg_to_hls(model, forward_dict, graph_dims,
               activate_final = None,
               fixed_precision_bits=16,
               fixed_precision_int_bits=6,
               int_precision_bits=16,
               int_precision_signed=False,
               output_dir = None):
```
Graph padding and truncation

Hardware-implementation requires: n_nodes_max, n_edges_max

1. Truncation: removes true nodes, true edges, or both
   a. Always bad
      i. Fixes: Look for disconnected nodes, remove the least connected nodes first
         1. Compute

2. Padding: adds dummy nodes, dummy edges, or both
   a. Sometimes bad
      i. n_nodes >= n_nodes_max, n_edges < n_edges_max
         1. Dummy edges must connect true nodes
   b. Sometimes alright
      i. n_nodes < n_nodes_max, n_edges <= n_edges_max
         1. Dummy nodes disconnected, or dummy nodes connected with dummy edges
Support yet to come

- Non-linear layers
- Arbitrary # of layers per block
  - Currently: 1-->4 layers
- Higher-degree neighbors for “Message” and “Aggregation”
  - Currently only first-degree neighbors
- Different “Message” schemes
  - Currently: NN_input = concat(receiver attributes, sender attributes, edge attributes)
About the author

- Contact: aelabd@sas.upenn.edu, abdelabd777@gmail.com
- B.A.: Physics, Economics @ U.Penn.
  - Graduation: August 2021
- Research Interests:
  - HEP, Astronomy, Applied ML for experimental Physics
  - Information theory, entropy and thermodynamics
  - Signal processing and hardware design
  - Graph Neural Networks and Graphical Causality
  - Reinforcement learning, Game Theoretic ML
- Hobbies:
  - Mountain biking, hiking, fishing
  - Basketball
  - Electric Guitar (noob)
- Applying to Physics and CS PhD programs!