egnn n-body system train

Train script:

```
python -u main_nbody.py --exp_name exp_1_egnn_vel --model egnn_vel --
max_training_samples 3000 --lr 5e-4
```

Model

```
EGNN_vel(
  (embedding): Linear(in_features=1, out_features=64, bias=True)
  (gcl_0): E_GCL_vel(
    (edge_mlp): Sequential(
      (0): Linear(in_features=131, out_features=64, bias=True)
      (1): SiLU()
      (2): Linear(in_features=64, out_features=64, bias=True)
      (3): SiLU()
    )
    (node_mlp): Sequential(
      (0): Linear(in_features=128, out_features=64, bias=True)
      (1): SiLU()
      (2): Linear(in_features=64, out_features=64, bias=True)
    )
    (coord_mlp): Sequential(
      (0): Linear(in_features=64, out_features=64, bias=True)
      (1): SiLU()
      (2): Linear(in_features=64, out_features=1, bias=False)
    )
    (coord_mlp_vel): Sequential(
      (0): Linear(in_features=64, out_features=64, bias=True)
      (1): SiLU()
      (2): Linear(in_features=64, out_features=1, bias=True)
    )
  )
  (gcl_1): E_GCL_vel(
    (edge_mlp): Sequential(
      (0): Linear(in_features=131, out_features=64, bias=True)
      (1): SiLU()
      (2): Linear(in_features=64, out_features=64, bias=True)
      (3): SiLU()
    )
    (node_mlp): Sequential(
      (0): Linear(in_features=128, out_features=64, bias=True)
      (1): SiLU()
      (2): Linear(in_features=64, out_features=64, bias=True)
    )
    (coord_mlp): Sequential(
      (0): Linear(in_features=64, out_features=64, bias=True)
      (1): SiLU()
```

```
        (2): Linear(in_features=64, out_features=1, bias=False)
      )
      (coord_mlp_vel): Sequential(
        (0): Linear(in_features=64, out_features=64, bias=True)
        (1): SiLU()
        (2): Linear(in_features=64, out_features=1, bias=True)
      )
    )
    (gcl_2): E_GCL_vel(
      (edge_mlp): Sequential(
        (0): Linear(in_features=131, out_features=64, bias=True)
        (1): SiLU()
        (2): Linear(in_features=64, out_features=64, bias=True)
        (3): SiLU()
      )
      (node_mlp): Sequential(
        (0): Linear(in_features=128, out_features=64, bias=True)
        (1): SiLU()
        (2): Linear(in_features=64, out_features=64, bias=True)
      )
      (coord_mlp): Sequential(
        (0): Linear(in_features=64, out_features=64, bias=True)
        (1): SiLU()
        (2): Linear(in_features=64, out_features=1, bias=False)
      )
      (coord_mlp_vel): Sequential(
        (0): Linear(in_features=64, out_features=64, bias=True)
        (1): SiLU()
        (2): Linear(in_features=64, out_features=1, bias=True)
      )
    )
    (gcl_3): E_GCL_vel(
      (edge_mlp): Sequential(
        (0): Linear(in_features=131, out_features=64, bias=True)
        (1): SiLU()
        (2): Linear(in_features=64, out_features=64, bias=True)
        (3): SiLU()
      )
      (node_mlp): Sequential(
        (0): Linear(in_features=128, out_features=64, bias=True)
        (1): SiLU()
        (2): Linear(in_features=64, out_features=64, bias=True)
      )
      (coord_mlp): Sequential(
        (0): Linear(in_features=64, out_features=64, bias=True)
        (1): SiLU()
        (2): Linear(in_features=64, out_features=1, bias=False)
      )
      (coord_mlp_vel): Sequential(
        (0): Linear(in_features=64, out_features=64, bias=True)
        (1): SiLU()
        (2): Linear(in_features=64, out_features=1, bias=True)
      )
```

```
    )
)
```

Result

Best Val Loss: 0.00728 Best Test Loss: 0.00772 Best epoch 255

Human Label Training

human_label positions preprocessing

finished using layout.ipynb

for muti-label on a single graph, we choose one label:

```
import pandas as pd

data = pd.read_csv("human_label/human_feedback.csv")
result = data.loc[data.groupby("graph_id")["user_id"].idxmin()]
result = result.sort_values("graph_id")
# print(result)
result.to_csv("filtered_file.csv", index=False)
```

we generate the final layout file `aligned_positions.npy` for training.

results

test_loss for about 60 epoch: 270 test_loss for about 140 epoch: 247

Egnn Code Learning

One Single Equivariant Convolutional Layer:

Layer init:

- input_nf: the dim of node embedding input
- output_nf: the dim of node embedding output
- hidden_nf: the dim of hiddle layer
- edges_in_d: the dim of edge feature
- act_fn: action function
- residual: res net work
- attention: introduce attention masks
- normalize: normalization of coords
- coords_agg: coords aggrefation function: sum/mean
- tanh: stub

```
class E_GCL(nn.Module):
  def __init__(self, input_nf, output_nf, hidden_nf, edges_in_d=0,
act_fn=nn.SiLU(), residual=True, attention=False, normalize=False,
coords_agg='mean', tanh=False):
```

Forward:

- h: [num_nodes, input_nf]shape, node eature embedding as input.
- edge_index: [2, num_edges]shape, can be seperated as [row, col]
- coord: [num_nodes, input_nf] shape, positions values, or layouts.
- edge_attr: [num_edges, edges_in_d]shape, edge features
- node_attr: stub

```
def forward(self, h, edge_index, coord, edge_attr=None, node_attr=None):
    row, col = edge_index
    radial, coord_diff = self.coord2radial(edge_index, coord)

    edge_feat = self.edge_model(h[row], h[col], radial, edge_attr)
    coord = self.coord_model(coord, edge_index, coord_diff, edge_feat)
    h, agg = self.node_model(h, edge_index, edge_feat, node_attr)

    return h, coord, edge_attr
```

Main Components

coord2radial

computer distance between node positions.

```
def coord2radial(self, edge_index, coord):
    row, col = edge_index
    coord_diff = coord[row] - coord[col]
    radial = torch.sum(coord_diff**2, 1).unsqueeze(1)

    if self.normalize:
        norm = torch.sqrt(radial).detach() + self.epsilon
        coord_diff = coord_diff / norm

    return radial, coord_diff
```

edge_model

equals the message aggregation in egnn paper: $$ \mathbf{m}_{ij} = \phi_e \left(\mathbf{h}_i^l, \mathbf{h}_j^l, | \mathbf{x}_i^l - \mathbf{x}*j^l /^2, a*{ij} \right) $$

```
def edge_model(self, source, target, radial, edge_attr):
    if edge_attr is None:  # Unused.
        out = torch.cat([source, target, radial], dim=1)
    else:
        out = torch.cat([source, target, radial, edge_attr], dim=1)
    out = self.edge_mlp(out)
    if self.attention:
        att_val = self.att_mlp(out)
        out = out * att_val
    return out
```

coords_model

equals the updating coords in egnn paper: $$ \mathbf{x}_i^{l+1} = \mathbf{x}*i^l + C |sum{j \neq i} \left(\mathbf{x}_i^l - \mathbf{x}*j^l |right) |phi_x |left(|mathbf{m}*{ij} \right) \tag{4} $$

```
def coord_model(self, coord, edge_index, coord_diff, edge_feat):
    row, col = edge_index
    trans = coord_diff * self.coord_mlp(edge_feat)
    if self.coords_agg == 'sum':
        agg = unsorted_segment_sum(trans, row, num_segments=coord.size(0))
    elif self.coords_agg == 'mean':
        agg = unsorted_segment_mean(trans, row,
num_segments=coord.size(0))
    else:
        raise Exception('Wrong coords_agg parameter' % self.coords_agg)
    coord = coord + agg
    return coord

def unsorted_segment_sum(data, segment_ids, num_segments):
    result_shape = (num_segments, data.size(1))
    result = data.new_full(result_shape, 0)  # Init empty result tensor.
    segment_ids = segment_ids.unsqueeze(-1).expand(-1, data.size(1))
    result.scatter_add_(0, segment_ids, data)
    return result
```

node_model

equals the following in egnn paper: $$ \mathbf{m}*i = |sum{j \neq i} \mathbf{m}_{ij} \tag{5} $$

$$ \mathbf{h}_i^{l+1} = \phi_h \left(\mathbf{h}_i^l, \mathbf{m}_i \right) $$

```
def node_model(self, x, edge_index, edge_attr, node_attr):
    row, col = edge_index
    agg = unsorted_segment_sum(edge_attr, row, num_segments=x.size(0))
    if node_attr is not None:
        agg = torch.cat([x, agg, node_attr], dim=1)
```

```
        else:
            agg = torch.cat([x, agg], dim=1)
        out = self.node_mlp(agg)
        if self.residual:
            out = x + out
        return out, agg
```

Applying

Do we need to use graph instead of looping over all i != j?

I suppose no. in Our smartGD code, we compute all shorted path between two nodes, which is very similar to edge model.