```
import torch
import torch.nn as nn

'''

Ensemble model, currently composed of a GNN model and a Transformer model.
Combines embeddings from both with a small MLP,
which projects back to the originial embedding dimension.

'''

class EnsembleEmbedder(nn.Module):
    def __init__(self,
        gnn_model,
        transformer_model,
        d_model,
        global_pool='max',
        dropout=0):

        super(EnsembleEmbedder, self).__init__()

        self.gnn_model = gnn_model
        self.transformer_model = transformer_model

        self.reduce_proj = nn.Sequential(nn.Dropout(dropout),
                                         nn.Linear(d_model * 2, d_model),
                                         nn.ReLU())

        self.global_pool = global_pool

    def forward(self, data):
        # assume data is passed as a tuple, (graph, sequence)

        out = torch.cat([self.gnn_model(data[0]),
            self.transformer_model(data[1])], dim=1)

        out = self.reduce_proj(out)

        return out

```

Listing 1: Adding new model specification to models module

```python
...
from models.ensemble.ensemble import EnsembleEmbedder

def get_model(model_config):
    if model_config.model_type == 'transformer':
    ...
    elif model_config.model_type == 'gcn':
    ...

    ############### Additional code ###############


    elif model_config.model_type == 'ensemble':

        model_0 = FormulaNetEdges(
              input_shape=model_config.model_attributes['vocab_size'],
              embedding_dim=model_config.model_attributes['embedding_dim'],
              num_iterations=model_config.model_attributes['gnn_layers'],
              batch_norm=model_config.model_attributes['batch_norm']
                  if 'batch_norm' in model_config.model_attributes else True)

        model_1 = TransformerWrapper(
            ntoken=model_config.model_attributes['vocab_size'],
             d_model=model_config.model_attributes['embedding_dim'],
             nhead=model_config.model_attributes['num_heads'],
             nlayers=model_config.model_attributes['num_layers'],
             dropout=model_config.model_attributes['dropout'],
             d_hid=model_config.model_attributes['dim_feedforward'],
             small_inner=model_config.model_attributes['small_inner']
                if 'small_inner' in model_config.model_attributes else False,
             max_len=model_config.model_attributes['max_len']
                if 'max_len' in model_config.model_attributes else 512)

        return EnsembleEmbedder(
            d_model=model_config.model_attributes['embedding_dim'],
            gnn_model=model_0,
            transformer_model=model_1,
            dropout=model_config.model_attributes['dropout'])

    ################################################

```

Listing 2: Adding new model to get_model interface

```python
...

############### Additional code ###############

# return list of tuples,
# with graph and sequence data in first/second positions
model_config
def to_ensemble_batch(data_list, attributes):
    data_list_0 = [a[0] for a in data_list]
    data_list_1 = [a[1] for a in data_list]
    data_list_0 = to_graph_batch(data_list_0, attributes)
    data_list_1 = to_sequence_batch(data_list_1, attributes['max_len'])
    data_list_1 = (data_list_1[0], data_list_1[1])
    return (data_list_0, data_list_1)

###############################################

...

def transform_batch(batch, config):
    if config.type == 'graph':
        return to_graph_batch(batch, config.attributes)
    elif config.type == 'sequence':
        return to_sequence_batch(batch, config.attributes['max_len'])
    elif config.type == 'relation':
        return list_to_relation(batch, config.attributes['max_len'])
    elif config.type == 'fixed':
        return batch

############### Additional code ###############

    elif config.type == 'ensemble':
        return to_ensemble_batch(batch, config.attributes)

###############################################

```

Listing 3: Adding necessary transforms for new data type

```

data_config:
  type: 'ensemble'
  data_options:
    filter: ['tokens', 'edge_attr', 'edge_index', 'sequence']
  attributes:
    max_len: 1024

```

Listing 4: Configuration specifying new data type and fields it needs from database

```
# @package _global_
defaults:
  - holist_supervised
  - /data_type/ensemble@_here_

exp_config:
  name: ensemble_holist_supervised

model_config:
  model_type: ensemble
  model_attributes:
    embedding_dim: 128
    gnn_layers: 12
    batch_norm: False
    dropout: 0.2
    num_layers: 4
    num_heads: 4
    dim_feedforward: 256
```

Listing 5: Configuration using the model in an example experiment