Why Attention Graphs Are All We Need: Pioneering Hierarchical Classification of Hematologic Cell Populations with LeukoGraph

Supplementary Material

Table 1 below describes each patient's data, which is represented as a tabular matrix with approximately hundreds of thousands of rows and twelve features.

Patient	Total cell population	T lymphocytes (%)	B lymphocytes (%)	Monocytes (%)	Mast Cells (%)	HSPC (%)	Myeloid HSPC (%)	Lymphoid HSPC (%)
1	94664	66439 (70.18%)	11773 (12.44%)	12768 (13.49%)	66 (0.07%)	3618 (3.82%)	2928 (3.09%)	487 (0.51%)
2	21915	7839 (35.77%)	2982 (13.61%)	6073 (27.71%)	100 (0.46%)	4921 (22.45%)	2694 (12.29%)	1943 (8.87%)
3	85217	49738 (58.37%)	13307 (15.62%)	8938 (10.49%)	86 (0.10%)	13148 (15.43%)	7886 (9.25%)	4582 (5.38%)
4	76704	44914 (58.55%)	7521 (9.81%)	16964 (22.12%)	138 (0.18%)	7167 (9.34%)	5937 (7.74%)	529 (0.69%)
5	107637	52221 (48.52%)	9787 (9.09%)	27350 (25.41%)	160 (0.15%)	18119 (16.83%)	14755 (13.71%)	2836 (2.63%)
6	76125	26920 (35.36%)	4768 (6.26%)	30124 (39.57%)	158 (0.21%)	14155 (18.59%)	9010 (11.84%)	4665 (6.13%)
7	81182	61417 (75.65%)	7484 (9.22%)	4753 (5.85%)	18 (0.02%)	7510 (9.25%)	5202 (6.41%)	1989 (2.45%)
8	48143	30190 (62.71%)	10799 (22.43%)	3064 (6.36%)	68 (0.14%)	4022 (8.35%)	2486 (5.16%)	1282 (2.66%)
9	27137	13667 (50.36%)	136 (0.50%)	4431 (16.33%)	30 (0.11%)	8873 (32.70%)	8278 (30.50%)	9 (0.03%)
10	120654	68396 (56.69%)	47653 (39.50%)	3788 (3.14%)	136 (0.11%)	681 (0.56%)	167(0.14%)	468(0.39%)
11	83704	28560 (34.12%)	5313 (6.35%)	33976 (40.59%)	31 (0.04%)	15824 (18.90%)	12615(15.07%)	2608(3.12%)
12	27091	19633 (72.47%)	1582 (5.84%)	5116 (18.88%)	3 (0.01%)	757 (2.79%)	464 (1.71%)	267 (0.99%)
13	37797	22788 (60.29%)	3768 (9.97%)	8522 (22.55%)	232 (0.61%)	2487 (6.58%)	1580 (4.18%)	703 (1.86%)
14	24626	16627 (67.52%)	4307 (17.49%)	2156 (8.75%)	42 (0.17%)	1494 (6.07%)	1445 (5.87%)	3 (0.01%)
15	114043	86021 (75.43%)	12369 (10.85%)	11357 (9.96%)	51 (0.04%)	4245 (3.72%)	2660 (2.33%)	1107 (0.97%)
16	63750	43842 (68.77%)	12719 (19.95%)	4764 (7.47%)	11 (0.02%)	2414 (3.79%)	1449 (2.27%)	844 (1.32%)
17	109781	37995 (34.61%)	39854 (36.30%)	20879 (19.02%)	278 (0.25%)	10775 (9.81%)	6423 (5.85%)	3785 (3.45%)
18	88001	61466 (69.85%)	6487 (7.37%)	12614 (14.33%)	26 (0.03%)	7408 (8.42%)	5156 (5.86%)	1913 (2.17%)
19	20212	12920 (63.92%)	3656 (18.09%)	2620 (12.96%)	27 (0.13%)	989 (4.89%)	709 (3.51%)	240 (1.18%)
20	49863	38406 (77.02%)	3111 (6.24%)	4340 (8.70%)	40 (0.08%)	3966 (7.95%)	2082 (4.18%)	1680 (3.37%)
21	86991	55871 (64.23%)	14300 (16.44%)	8307 (9.55%)	4 (0.00%)	8509 (9.78%)	4671 (5.37%)	2913 (3.35%)
22	74477	46971 (63.07%)	15616 (20.97%)	5681 (7.63%)	38 (0.05%)	6171 (8.29%)	2675 (3.59%)	3201 (4.30%)
23	49774	31063 (62.41%)	8375 (16.83%)	4699 (9.44%)	815 (1.64%)	4822 (9.69%)	3362 (6.75%)	1277 (2.57%)
24	133820	111218 (83.11%)	5749 (4.30%)	10868 (8.12%)	51 (0.04%)	5934 (4.43%)	3544 (2.65%)	1956 (1.46%)
25	53192	37326 (70.17%)	576 (1.08%)	11163 (20.99%)	30 (0.06%)	4097 (7.70%)	1672 (3.14%)	2161 (4.06%)
26	66972	42925 (64.09%)	974 (1.45%)	16500 (24.64%)	399 (0.60%)	6174 (9.22%)	5614 (8.38%)	211 (0.32%)
27	22253	8100 (36.40%)	11386 (51.17%)	2027 (9.11%)	4 (0.02%)	736 (3.31%)	564 (2.53%)	127 (0.57%)
28	42758	24857 (58.13%)	51 (0.12%)	10836 (25.34%)	117 (0.27%)	6897 (16.13%)	6769 (15.83%)	1 (0.00%)
29	64569	50091 (77.58%)	3452 (5.35%)	5826 (9.02%)	271 (0.42%)	4929 (7.63%)	4650 (7.20%)	143 (0.22%)
30	121287	91772 (75.67%)	867 (0.71%)	5809 (4.79%)	149 (0.12%)	22690 (18.71%)	5506 (4.54%)	15846 (13.06%)

Table 1: Graph statistics for every patient of the dataset.

1 Hierarchical Code Implementation

This section provides additional details on how the hierarchical constraint was implemented in the code for the LeukoGraph model, here are the key steps:

- It first defines the hierarchical structure by creating a directed graph g from the class hierarchy string *ATTRIBUTE_class*, as shown in Figure 1 below. This maps out the parent-child relationships between classes.
- It converts this graph to a matrix R that encodes the ancestor-descendant relationships, where $R[i, j] = 1$ if class i is descendant of class j, 0 otherwise.
- Then the model includes GAT layers, where each of them does message passing and computes attention coefficients between neighboring nodes.
- During training, the raw GAT predictions are returned.
- During inference, the predictions are passed through a constraint function *get_constr_out* that takes the max over descendants to enforce the hierarchy, as depicted in Figure 2 below.
- The loss function is a modified cross-entropy loss MCLoss that also lets subclasses inform about their parent classes predictions.

- The model takes a graph and node features as input, does message passing and attention in the GAT layers, applies the hierarchical constraint, and outputs a prediction vector for each node.

In summary, it implements a hierarchical classifier using a GAT augmented with constraints and loss terms to incorporate the class hierarchy information. The hierarchy structure is predefined and encoded in the constraint matrix R.

1.1 Constraint Layer

The hierarchy consistency is imposed through a constraint layer added on top of the base classifier network. This constraint layer, referred to as the Max Constraint Module (MCM), takes the output predictions from the base network and ensures the hierarchical constraint is satisfied. The MCM output for any class A is computed as:

$$\mathrm{MCM}_A = \max(\mathcal{H}_B \text{ where } B \text{ is subclass of } A), \tag{1}$$

where $\mathcal{H}_B$ is the prediction value from the base network for class B. This takes the maximum over all subclasses of A, guaranteeing that the prediction for class A will be greater than or equal to the prediction for any of its subclasses. By constructing the output this way, the hierarchy constraint is inherently satisfied regardless of the threshold used downstream.

```python
# List all the classes
ATTRIBUTE_class = "1,2,3,4,5,5_1,5_2,5_3"

# Store nodes and direct connections
g = nx.DiGraph()
for branch in ATTRIBUTE_class.split(','):
    term = branch.split('_')
    if len(term)==1:
        g.add_edge(term[0], 'root')
    else:
        for i in range(2, len(term) + 1):
            g.add_edge('.'.join(term[:i]), '.'.join(term[:i-1]))

nodes = sorted(g.nodes(), key=lambda x: (len(x.split('.')),x))

AA = np.array(nx.to_numpy_array(g, nodelist=nodes))

# Compute matrix of ancestors R
# Given C classes, R is an (C x C) matrix where R_ij = 1 if class i is descendant of class j
R = np.zeros(AA.shape)
np.fill_diagonal(R, 1)
gg = nx.DiGraph(AA) # AA is the matrix where the direct connections are stored
for i in range(len(AA)):
    ancestors = list(nx.descendants(gg, i)) # Need to use the function nx.descendants()
    #because in the directed graph the edges have source
    #from the descendant and point towards the ancestor
    if ancestors:
        R[i, ancestors] = 1
R = torch.tensor(R)
#Transpose to get the descendants for each node
R = R.transpose(1, 0)
R = R.unsqueeze(0).to(device)
```

Figure 1: Python code snippet generating a matrix of ancestors R based on the class relationships in the directed graph. Each class is represented as a node, and edges define parent-child connections.

1.2 Constrained Loss Function

The loss function used during training is the max constraint loss (MCLoss), defined as:

$$\mathrm{MCLoss}_A = -y_A \log\left(\max_{B \text{ in subclasses\_of\_A}} (y_B \cdot \mathcal{H}_B)\right) - (1 - y_A)\log(1 - \mathrm{MCM}_A), \tag{2}$$

where y_A is the ground truth label. This loss allows the model to leverage the hierarchical information by letting predictions from subclasses inform the training for parent classes.

```python
def get_constr_out(x, R):
    ''' Given the output of the graph neural network x returns the output of
    MCM given the hierarchy constraint expressed in the matrix R '''
    c_out = x.double()
    c_out = c_out.unsqueeze(1)
    c_out = c_out.expand(len(x),R.shape[1], R.shape[1])
    R_batch = R.expand(len(x),R.shape[1], R.shape[1])
    final_out, _ = torch.max(R_batch*c_out.double(), dim = 2)
    return final_out

# Define LeukoGraph model
class ConstrainedLeukoGraph(nn.Module):
    ''' During training it returns the not-constrained output that is then passed to MCLoss'''
    def __init__(self,R):
        super(ConstrainedLeukoGraph, self).__init__()
        self.R = R
        self.nb_layers = 2
        self.num_heads = 8
        self.out_head = 8
        self.hidden_dim = 64
        self.input_dim = 12
        self.output_dim= len(set(ATTRIBUTE_class.split(',')))+1  # We do not evaluate
        #the performance of the model on the 'root' node
        gat_layers = []
        for i in range(self.nb_layers):

            if i == 0:
                gat_layers.append(GATLayer(self.input_dim, self.hidden_dim,
                self.num_heads,True,0.4))
            elif i == self.nb_layers - 1:
                gat_layers.append(GATLayer(self.hidden_dim*self.num_heads,
                self.output_dim, self.out_head, False,0.4 ))
            else:
                gat_layers.append(GATLayer(self.hidden_dim*self.num_heads,
                self.hidden_dim, self.num_heads,True,0.4))
        self.gat_layers = nn.ModuleList(gat_layers)

        self.sigmoid = nn.Sigmoid()
        self.f = nn.ReLU()
        self.reset_parameters()  # Initialize the weights

    def reset_parameters(self):
        for gat_layer in self.gat_layers:
            gat_layer.reset_parameters()

    def forward(self, data):
        x, edge_index= data.x, data.edge_index

        for i in range(self.nb_layers):
            x = self.gat_layers[i](x, edge_index)
            if i != self.nb_layers - 1:
                x = self.f(x)
            else:
                x= self.sigmoid(x)
        if self.training:
            constrained_out = x
        else:
            constrained_out = get_constr_out(x, self.R )
        return constrained_out
```

Figure 2: LeukoGraph's model definition with hierarchical constraints. The model incorporates the hierarchy matrix R to enforce class relationships during training. After performing message passing and attention computations, the model's output is constrained during inference using the *get_constr_out* function.

1.3 Time Complexity

The time complexity of the constraint layer is $O(N)$ where N is the number of classes, since it just takes a maximum over the subclasses for each class. So it scales linearly with the size of the hierarchy and does not depend on the depth. The overall model maintains the same asymptotic time complexity as the base classification network used, with just an additional linear factor for the constraint layer. Hence, the hierarchical structure does not add significant overhead during training or inference.