COM S 673: Advanced Topics in Computational Models of Learning
Assignment #3 (part a)

---

1. You need to submit a report and your code to Canvas. Your hard-copy report should include (1) answers to the non-programming part, and (2) analysis and results of the programming part. Please put all your code files and report into a compressed file named "HW#_FirstName_LastName.zip"
2. Unlimited number of submissions are allowed and the latest one will be timed and graded.
3. Please read and follow submission instructions. No exception will be made to accommodate incorrectly submitted files/reports.
4. All students are required to typeset their reports using latex.
5. Only write your code between the following lines. Do not modify other parts.

   ### YOUR CODE HERE

   ### END YOUR CODE

---

1. (70 points) (Coding Task) Graph Neural Networks.

   In this assignment, you will use PyTorch to implement some graph deep learning layers including graph convolution layer, graph attention layer, graph differentiable pooling layer, and graph pooling layer. You are given a graph classification dataset. The code folder provides the starting code. You must implement the layers using the starting code. In this assignment, you may need a GPU.

   (a) (15 points) Complete the class GConv in "src/utils/layers.py", which implements the graph convolution layer proposed in [1].

   (b) (15 points) Complete the class GAttn in "src/utils/layers.py", which implements the graph attention layer proposed in [2].

   (c) (15 points) Complete the class GDiffPool in "src/utils/layers.py", which implements the graph differentiable pooling layer proposed in [3].

   (d) (15 points) Complete the class GPool in "src/utils/layers.py", which implements the graph pooling layer proposed in [4].

   (e) (10 points) Run the model by "bash run_GNN.sh" and report the mean 10-fold cross-validation performance. After running the model, the 10-fold results are automatically store in the "results/PROTEINS.txt" file. Please try different graph convolution layers (GConv or GAttn) and different graph pooling layers (GDiffPool or GPool) in "configs/PROTEINS" file and provide a short analysis of the results.

   Required Materials:

   [1] Semi-Supervised Classification with Graph Convolutional Networks (`https://arxiv.org/pdf/1609.02907.pdf`)

[2] Graph Attention Networks (https://arxiv.org/pdf/1710.10903.pdf)

[3] Hierarchical Graph Representation Learning with Differentiable Pooling (https://arxiv.org/pdf/1806.08804.pdf)

[4] Graph U-Nets (https://arxiv.org/pdf/1905.05178.pdf)

2. (30 points) As introduced in class, the attention mechanism can be written into:

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T)V.$$

By adding linear transformations on Q, K, and V, it turns into:

$$\text{Attention}(QW^Q, KW^K, VW^V) = \text{softmax}(QW^Q(KW^K)^T)VW^V.$$

Here, we set $Q \in \mathbb{R}^{n\times d}$, $W^Q \in \mathbb{R}^{d\times d}$, $K \in \mathbb{R}^{n\times d}$, $W^K \in \mathbb{R}^{d\times d}$, $V \in \mathbb{R}^{n\times d}$, $W^V \in \mathbb{R}^{d\times d}$.

In practice, the multi-head attention is used, which is defined as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, \ldots, head_h),$$

where

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V),\ i = 1, \ldots, h.$$

Here, Q, K, V are the same as defined above. We set $W_i^Q \in \mathbb{R}^{d\times \frac{d}{h}}$, $W_i^K \in \mathbb{R}^{d\times \frac{d}{h}}$, $W_i^V \in \mathbb{R}^{d\times \frac{d}{h}}$.

(a) (10 points) Compute and compare the number of parameters between the single-head and multi-head attention.

(b) (10 points) Compute and compare the amount of computation between the single-head and multi-head attention, including the softmax step. Use the big-O notation to show your results.

(Hint2: Quoted from the paper (https://arxiv.org/pdf/1706.03762.pdf), "Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.")

(c) (10 points) In a variant of attention layer, the softmax operator is replaced by $1/n$, which can save a lot of computational resources. Show how this change can reduce the computational cost. Please ignore the computational differences between softmax and $1/n$. You can assume $d < n$, which is common in computer vision applications.