CS69000-DPL Spring 2020 Homework	— Homework 4—	Last update: March 7, 2020
https://www.cs.purdue.edu/homes/ribeirob/courses/Spring2020/index.html		Due: **6:00am**, Sunday, March 15th

Instructions and Policy: Each student should write up their own solutions independently, no copying of any form is allowed. You MUST indicate the names of the people you discussed a problem with; ideally you should discuss with no more than two other people.
YOU MUST INCLUDE YOUR NAME IN THE HOMEWORK
You need to submit your answer in PDF. LaTeX is typesetting is encouraged but not required. Please write clearly and concisely - clarity and brevity will be rewarded. Refer to known facts as necessary.

Q0 (0pts correct answer, -1,000pts incorrect answer: (0,-1,000) pts): A correct answer to the following questions is worth 0pts. An incorrect answer is worth -1,000pts, which carries over to other homeworks and exams, and can result in an F grade in the course.

(1) Student interaction with other students / individuals:

   (a) I have copied part of my homework from another student or another person (plagiarism).

   (b) Yes, I discussed the homework with another person but came up with my own answers. Their name(s) is (are) _____

   (c) No, I did not discuss the homework with anyone

(2) On using online resources:

   (a) I have copied one of my answers directly from a website (plagiarism).

   (b) I have used online resources to help me answer this question, but I came up with my own answers (you are allowed to use online resources as long as the answer is your own). Here is a list of the websites I have used in this homework:
   _____

   (c) I have not used any online resources except the ones provided in the course website.

1 Homework 4: Inductive Biases for Graph and Sequence Data

Learning Objectives: Let students understand basic concepts that are used to add inductive biases in deep learning models: for images, sequences, and graphs.

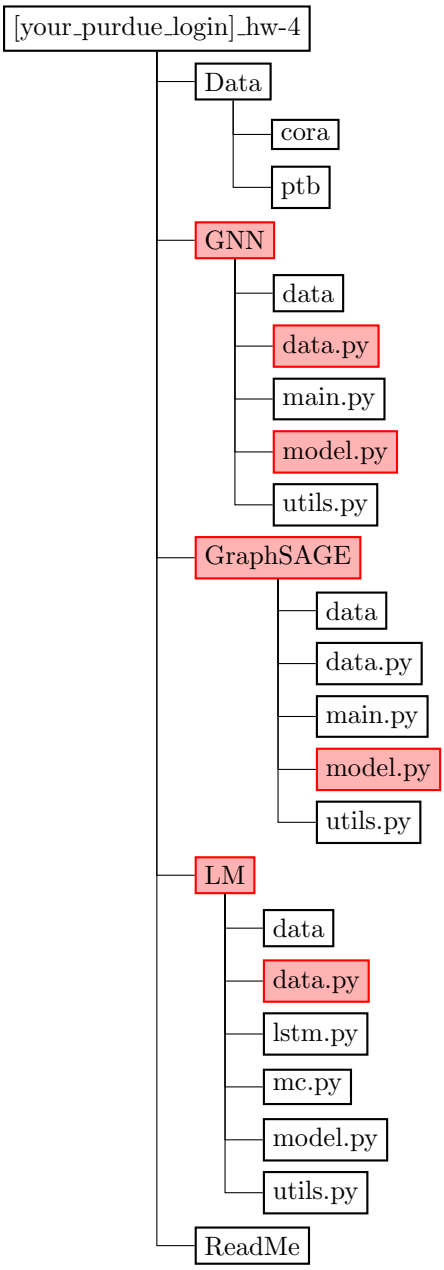
Learning Outcomes: After finishing this homework, students will be able to create new inductive biases to solve new tasks.

Here, we are going to implement (1) Graph Neural Network with mean aggregator, (2) Graph Neural Network with LSTM set aggregator, (3) Markov Chain language model and (4) LSTM language model. This HW is a combination of two distinct tasks.

The necessary files (code and data) can be obtained from Piazza.

1.1 HW Overview

You are going to fill in missing part of several files under each project folder. We plot the folder structure that you should use:



- **[your_purdue_login]_hw-4**: The top-level folder that contains all the files required in this homework. You should replace the file name with your name and follow the naming convention.
- **ReadMe**: Your ReadMe should begin with a couple of **example commands**, e.g., "python hw4.py data", used to generate the outputs you report. (Even if you follow exactly the same command line provided in each question.) TA must be able to replicate your results with the commands provided here. More detailed options, usages and designs of your program can be followed. You can also list any concerns that you think TA should know while running your program. Put the information that

you think it's more important at the top. Moreover, the file should be written in pure text format that can be displayed with Linux "less" command.

- **Data**: The root folder of all related data files in this HW. Each data folder under project folders is linked to this folder.
- **GNN**: Project folder for you to implement Graph Neural Networks with just mean aggregator. One module that you are going to develop:
  - **model.py**

  The detail will be provided in the task descriptions. All the other files are supporting files that you should not change.
- **GraphSAGE**: Project folder for you to implement GraphSAGE with both mean aggregator and LSTM aggregator. Pay attention that the design of this project is totally different from previous one. Implementation of Janossy pooling is also required in this project for LSTM aggregator. One module that you are going to develop:
  - **model.py**

  The detail will be provided in the task descriptions. All the other files are supporting files that you should not change.
- **LM**: Project folder for you to implement language models of Markov chain and LSTM. Four modules that you are going to develop:
  - **data.py**
  - **lstm.py**
  - **mc.py**
  - **model.py**

  The detail will be provided in the task descriptions. All the other files are supporting files that you should not change.

2 Graph Neural Networks (GNN)

Task 1a: GNN with Mean Aggregator

This is a warm-up task. You are going implement mean aggregator based on provided formula and supporting files under GNN folder.

In the following GNN related coding assignment, you will work with the Cora dataset. The Cora dataset consists of 2708 scientific publications classified into one of seven classes. The citation network consists of 5429 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words.

You can regard it as a labeled graph $G = (V, E, X, Y)$ where $V = \{1, 2, \cdots, 2708\}$ is the set of $n = 2708$ nodes, E is the set of 5429 undirected edges, X and Y are the feature and label matrices assigned to the graph. Although the raw dataset is a directed graph, we will regard it as an undirected graph for convenience. For each node $v \in V$, we have a feature vector $\boldsymbol{x}_v$ and a label y_v. Thus, you will have

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_n \end{bmatrix} \quad \boldsymbol{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \tag{1}$$

where each $\boldsymbol{x}_u$ is a 1433 dimensional vector and $\boldsymbol{X}$ is a 2708×1433 matrix.

In GNN of this project, features of a node u and its neighbors N_u will be aggregated together from layer k to next layer $k+1$:

$$\boldsymbol{h}_u^{(k+1)} = \begin{cases} \sigma\left(\boldsymbol{W}^{(k+1)}\left[\boldsymbol{h}_u^{(k)}, \overline{\overline{f}}(\{\boldsymbol{h}_v^{(k)}\}_{v \in N_u})\right] + \boldsymbol{b}^{(k+1)}\right) & k+1 > 0 \\ \boldsymbol{x}_u & k+1 = 0 \end{cases} \tag{2}$$

where $\boldsymbol{h}_u^{(k)}$ is the embedding features of node u at layer k, $\boldsymbol{W}^{(k)}$ and $\boldsymbol{b}^{(k)}$ are weight and bias matrices of layer k, σ is a specific activation and $\overline{\overline{f}}$ is a specific permutation invariant function. In this project, $\overline{\overline{f}}$ will be the *mean* function. Thus, you will have a graph embedding matrix $\boldsymbol{H}^{(k)}$ for each layer k. The final embedding matrix will then be used to classify labels Y for graph G.

$$\boldsymbol{H}^{(k)} = \begin{bmatrix} \boldsymbol{h}_1^{(k)} \\ \boldsymbol{h}_2^{(k)} \\ \vdots \\ \boldsymbol{h}_n^{(k)} \end{bmatrix} \tag{3}$$

Related Modules:

- GNN/data.py
- GNN/model.py

Action Items:

1. The formula above is designed to work on each node separately which is slow for deep GNNs in practice. For the mean aggregator of this project, there will be a matrix version of the formula which allows us to exploit the efficiency of matrix computations. To be specific, you are given an embedding matrix $\boldsymbol{H}^{(k)}$ of layer k, adjacency matrix $\boldsymbol{A}$ where

$$A_{uv} = \begin{cases} 0 & u \text{ and } v \text{ are disconnected} \\ 1 & u \text{ and } v \text{ are connected} \end{cases} \tag{4}$$

   Rewrite the above formula from Equation 2, for mean as $\overline{f}$, so that you can get $\boldsymbol{H}^{(k+1)}$ from $\boldsymbol{H}^{(k)}$ and A. You can only define constant matrices in this question.

   Hint: You can compute the mean operation through a matrix multiplication.

2. Go through all the related modules. Specifically, you should understand **data.CoraDataset** and **model.GraphConvolution** well to use provided data properly.

   (a) Modify `GNN/data.py` to compute the necessary adjacency matrix.

   (b) Implement the GCN formula you derived above in in `GNN/model.py`.

   In the report, provide the min, max and average degrees of training, validation and test nodes of the Cora dataset.

	Min	Max	Average
Training			
Validation			
Test			

3. After understanding the matrix version of the formula and all the related modules, you should implement the formula into the **data.py** and **model.py** to make GNN run correctly. Run the **main.py** with default arguments: `python main.py`. Save the logging outputs from your console. **In the report, you should include:**

   (a) Learning curves of loss function of training and validation nodes.

   (b) Learning curves of accuracy of training and validation nodes.

   (c) Loss functions and accuracies of test nodes for the saved models.

4. Run `python main.py --num_layers 2`, and `python main.py --num_layers 20`. **In your report, include:**

   (a) Describe what happens during the training process

   (b) For the model from `python main.py --num_layers 2`, get $\boldsymbol{H}^{(1)}$; Report the maximum MSE of all pairs of embedding vectors in $\boldsymbol{H}^{(1)}$.

   (c) For the model from `python main.py --num_layers 20`, get $\boldsymbol{H}^{(19)}$; Report the maximum MSE of all pairs of embedding vectors in $\boldsymbol{H}^{(19)}$.

Task 1b: GNN with Mean and LSTM Aggregator

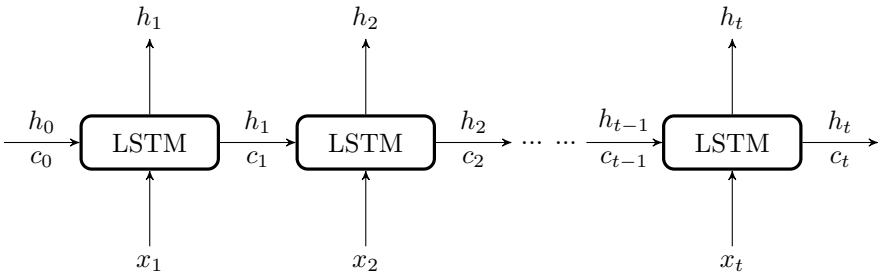
This is an advanced version of previous task. You are going implement a more general GNN supporting both mean and LSTM aggregator. This model is similar to what is known as the GraphSAGE architecture. **Pay attention that the design of this project will have time efficiency issues, so we highly recommend you to do your experiments on GPU resources, e.g. the scholar cluster. If it is not available, you should start as soon as possible. Running on CPU will take hours to finish.**

In the GNN of this project, features of a node u and its neighbors N_u will be aggregated together from layer k to next layer $k+1$:

$$\boldsymbol{h}_u^{(k+1)} = \begin{cases} \sigma\left(\boldsymbol{W}^{(k+1)}\left[\boldsymbol{h}_u^{(k)}, \overline{\overline{f}}(\{\boldsymbol{h}_v^{(k)}\}_{v\in N_u})\right] + \boldsymbol{b}^{(k+1)}\right) & k+1>0 \\ x_u & k+1=0 \end{cases} \tag{5}$$

where $\boldsymbol{h}_u^{(k)}$ is the embedding features of node u at layer k, $\boldsymbol{W}^{(k)}$ and $\boldsymbol{b}^{(k)}$ are weight and bias matrices of layer k, σ is a specific activation and $\overline{\overline{f}}$ is a specific permutation invariant function. But this time, $\overline{\overline{f}}$ will be either the mean function or LSTM architectures with Janossy pooling.

Consider a general LSTM architecture, as we have seen in class:



To use the LSTM as a neighbor aggregator, we will have a sequence of neighbor embeddings $\boldsymbol{h}_{v_1}, \boldsymbol{h}_{v_2}, \cdots, \boldsymbol{h}_{v_t}$ (where $t = |N_u|$ is the number of neighbors to aggregate), which we'll give as the input of the LSTM. To avoid confusion, we'll call the short term outputs from the LSTM by $\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_t$, so the final output of the LSTM is going to be $\boldsymbol{c}_t, \boldsymbol{m}_t = \vec{f}(\boldsymbol{h}_{v_1}, \boldsymbol{h}_{v_2}, \cdots, \boldsymbol{h}_{v_t})$.

Based on the definition of LSTM design, we know that state memory c_t can be regarded as an embedding of sequence $\boldsymbol{h}_{v_1}, \boldsymbol{h}_{v_2}, \cdots, \boldsymbol{h}_{v_t}$. The problem is that the LSTM function $\vec{f}$ is a permutation sensitive function while we are seeking for permutation invariant function $\overline{\overline{f}}$.

From the class, we know that Janossy pooling can be a method to get a permutation invariant function $\overline{\overline{f}}$ from permutation sensitive function $\vec{f}$ by

$$\overline{\overline{f}}(\boldsymbol{h}_{v_1}, \boldsymbol{h}_{v_2}, \cdots, \boldsymbol{h}_{v_t}) = \frac{1}{t!}\sum_{\pi\in\Pi(t)} \vec{f}(\boldsymbol{h}_{v_{\pi(1)}}, \boldsymbol{h}_{v_{\pi(2)}}, \cdots, \boldsymbol{h}_{v_{\pi(t)}}) \tag{6}$$

where $\Pi(t)$ will be all possible permutations to length t. In practice, enumerate $t!$ permutations is not tractable. In this project, we will use π-SGD by sampling a permutation rather than traversing all permutations.

$$\overrightarrow{\overrightarrow{f}}(\boldsymbol{h}_{v_1}, \boldsymbol{h}_{v_2}, \cdots, \boldsymbol{h}_{v_t}) \approx \vec{f}(\boldsymbol{h}_{v_{\hat{\pi}(1)}}, \boldsymbol{h}_{v_{\hat{\pi}(2)}}, \cdots, \boldsymbol{h}_{v_{\hat{\pi}(t)}}), \quad \text{where } \hat{\pi} \sim \text{Uniform}(\Pi(t)) \tag{7}$$

The above procedure is expensive if the node have large degrees. In what follows we consider 5-ary dependencies to ease the computational burden.

5-ary Janossy pooling. Another issue of LSTM is gradient vanishing or explosion when backpropagating through too many time steps. In this project, a node may have more than 100 neighbors which will require LSTM to backpropagate through more than 100 time steps. Rather, we will consider 5-ary Janossy pooling, trained with π-SGD. The key idea is only perform a forward and backward pass over the first 5 neighbors of $\hat{\pi}$ (ignoring the remaining neighbors). At inference time, where we will sample 20 permutations $\hat{\pi}$ (described later in the homework) we also need to only consider the first 5 neighbors from $\hat{\pi}$ in the forward pass.

Related Modules:

- GraphSAGE/model.py

Action Items:

1. You should address the difference between this task and previous task. **Report do you expect would happen if we also do the 20-layer experiment in this project.**
2. Fill in missing parts of `GraphSAGE/model.py`.

   Hint: If necessary, consult PyTorch's documentation of the LSTM: `https://pytorch.org/docs/stable/nn.html?highlight=lstm#torch.nn.LSTM`
3. Run `python main.py --agg mean` and `python main.py --agg LSTM`. Save the logging outputs from your console. **In the report, you should include:**
   (a) Learning curves of loss function of training and validation nodes.
   (b) Learning curves of accuracy of training and validation nodes.
   (c) Loss function and accuracy of test nodes for the saved model.
4. In the previous question with LSTM aggregator, you will also randomly sample a permutation in test stage. Modify the code, and use the average prediction of 20 permutations (average of 20 times of test stage) as final output. **What is the test accuracy over those 20 permutations? Explain why the results improve.**

3 Language Model (LM)

Task 2a: Markov Chain Modeling

In the following LM related tasks, you will work with the Penn Treebank (PTB) dataset. The PTB dataset is a collection of words and sentences. In the provided loader, an additional word "eos" will be added to each sentences to force language model to learn "end-of-sentence" embeddings. Given the first several words of a sentence and its context, your task is to predict next word.

In k-order Markov chain language modeling, we are computing the statistics over training data. Suppose for the t-th word w_t, we have its context $c_t = [w_{t-k}, w_{t-k+1}, \cdots, w_{t-1}]$. If the words of context go out of the context, we will just use token -1 (or "unk") for padding.

$$P(w_t|c_t) = \frac{\#(c_t, w_t)}{\sum_w \#(c_t, w)} \tag{8}$$

where $\#(c_t, w)$ is the number of appearance of sequence $[w_{t-k}, w_{t-k+1}, \cdots, w_{t-1}, w]$ in the whole training subset of the whole dataset.

Equation (8) could lead to zero probabilities. To solve this problem, we will use a simple smoothing schema.

$$P(w_t|c_t) = \begin{cases} \frac{\#(c_t,w)}{\left(\sum_w \#(c_t,w)\right)+1} & \#(c_t, w) > 0 \\ \frac{1/\#_{\text{unk}}(c_t,w)}{\left(\sum_w \#(c_t,w)\right)+1} & \#(c_t, w) = 0 \end{cases} \tag{9}$$

where $\#_{\text{unk}}(c_t, w) = \mathbb{1}(w|\#(c_t, w) = 0)$.

Related Modules:

- LM/data.py

Action Items:

1. Finish the missing part of each file. Feel free to modify the files as long as you can use Markov chain to predict cross entropy loss and accuracy on test data correctly.
2. Run `python LM/mc.py --order 1`, `python LM/mc.py --order 2`, and `python LM/mc.py --order 10`. **Report their perplexity on test subset.** Perplexity is a common metric used to evaluate the performance of language models, and it is indeed the exponent of cross entropy loss.

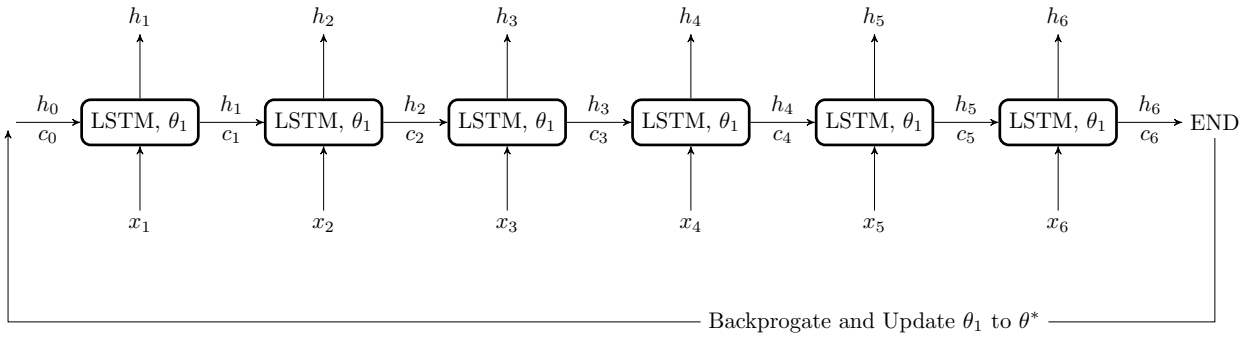
$$\exp\left(\frac{1}{N}\sum_{i=1}^{N} -\log P(w_i|w_{i-1}, \cdots, w_1)\right) \tag{10}$$

Task 2b: LSTM Modeling

In LSTM language modeling, we are computing the statistics over training data.

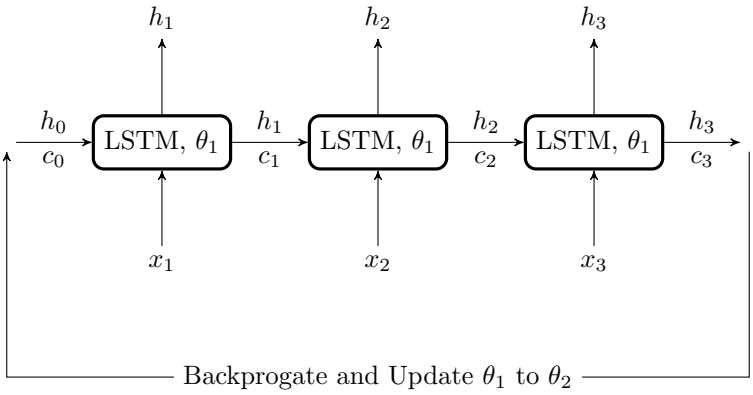
$$P(w_t|w_{t-1}, \cdots, w_1) \tag{11}$$

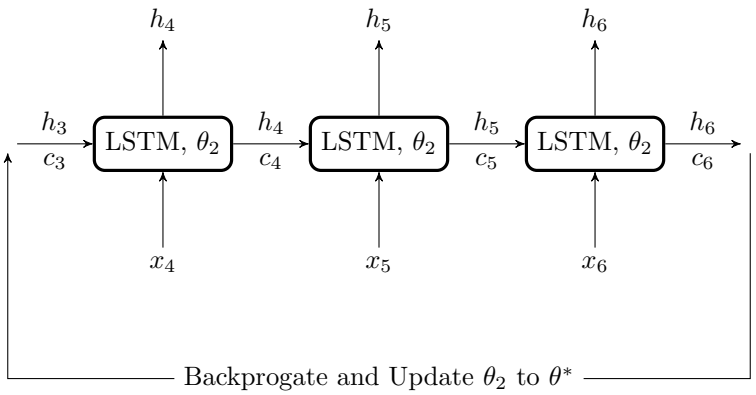
As mentioned in the class, one of the problem of LSTM is gradient vanishing or explosion over long sequences. To solve this problem, we often do truncated backpropagation through time rather than backprogration though the whole sequence. To be specific, for a sequence $x_1, x_2, x_3, x_4, x_5, x_6$ with common backpropagation, a full backprogation will work in the following way:



We will feed forward all 6 time steps with parameter θ_1, compute the loss over output all 6 time steps, backpropagate gradients through all 6 time steps and get new parameter θ^*.

For a truncated backprogragation through time with length 3, a full backprogation will work in the following way:





We will treat the first 3 time steps as a full sequence, and feed forward and backpropagate just like the previous one with parameter θ_1. After the parameter are updated to θ_2, we take the last hidden states h_3, c_3 as the initial hidden states of next process. Take the last 3 time steps as a full sequence (because there is not enough time step to generate a 3-time-step sequence), and feed forward and backpropagate just like the previous one but with parameter θ_2 and initial hidden states h_3, c_3. After parameters are updated, we get new parameter θ^*

For using batches, we just equally divide the whole dataset sequence into several independent segments. For example, for sequence x_1 to x_6, if we want to use a batch size of 2, we just split it into 2 sequences x_1, x_2, x_3 and x_4, x_5, x_6, and training them together.

Related Modules:

- LM/data.py
- LM/model.py

Action Items:

1. Understand the truncated backprogatation process, and finish the missing part of each file to support a truncated backprogation through time process. You should correctly split training sequence and forward the correct segment to earn full points for this question.
2. Run `python LM/lstm.py --bptt 5`, `python LM/lstm.py --bptt 35`, and `python LM/lstm.py --bptt 80`. **In the report, you should include, for each of the three runs:**
   (a) Learning curves of loss function of training and validation nodes.
   (b) Learning curves of accuracy of training and validation nodes.
   (c) Loss functions and accuracies of test nodes for the saved models.

Submission Instructions

Please read the instructions carefully. Failing to follow the instructions might lead to loss of points.

Naming convention: [your_purdue_login]_hw-4

All your submission files, including a ReadMe, and codes, should be included in one folder. The folder should be named with the above naming convention. For example, if my purdue account is "jsmith123", then for Homework 4 I should name my folder as "jsmith123_hw-4".

Remove any unnecessary files in your folder, such as training datasets (Data folder). Make sure your folder is structured as the tree shown in Overview section.

Submit: **TURNIN INSTRUCTIONS**

Please submit your homework files on **data.cs.purdue.edu** using the turnin command, e.g.:

`turnin -c cs690-dpl -p hw-4 jsmith123_hw-4`.

Please make sure you didn't use any library/source explicitly forbidden to use. If any such library/-source code is used, you will get 0 pt for the coding part of the assignment. If your code doesn't run on scholar.rcac.purdue.edu, then even if it compiles in another computer, your code will still be considered not-running and the respective part of the assignment will receive 0 pt.