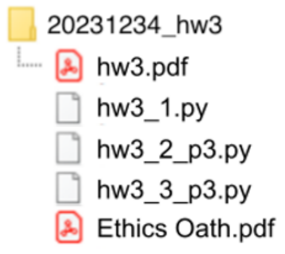
EE412 Foundation of Big Data Analytics, Fall 2023
HW3

Due date: 11/23/2023 (11:59pm)

Submission instructions: Use KAIST KLMS to submit your homework. Your submission should be a single gzipped tar file with the file name `YourStudentID_hw3.tar.gz`. For example, if your student ID is 20231234, the file name should be `20231234_hw3.tar.gz`. You can also use these extensions: tar, gz, zip, tar.zip. Do not use other options besides the ones mentioned above.

For the programming assignment except for problem 1, you are free to use either python 2.x or 3.x. Make sure to add p2 or p3 to the end of the file names. For example, if you used python 3, the file names should be `hw3_2_p3.py`, and `hw3_3_p3.py`

Your zip file should contain a total five files; one PDF file for writeup answers (`hw3.pdf`), three python files and the Ethics Oath pdf file. Before zipping your files, please make a directory named `YourStudentID_hw3` and put your files in the directory. Then, please compress the directory to make a zipped file. Do not use any Korean letters in file names or directory names. [To Do: Later change directory image and specification according to hw3-2]



```
20231234_hw3
  hw3.pdf
  hw3_1.py
  hw3_2_p3.py
  hw3_3_p3.py
  Ethics Oath.pdf
```

If you do not follow this format, we will **deduct 1 point** of the total score per mistake.

Submitting writeup: Write down the solutions to the homework questions in a single PDF file. You can use the following template. If you use Python in the exercises, **make sure you attach your code to your document**(`hw3.pdf`). Please write as succinctly as possible.

Submitting code: Each problem is accompanied by a programming section. Put all the code for each question into a single file. Good coding style (including comments) will be one criterion for grading. Please make sure your code is well structured and has descriptive comments.

We will **deduct** 1 point per error type (i.e., printing extra lines, format error, and etc)

Ethics Oath: For every homework submission, please fill out and submit the **PDF** version of this document that pledges your honor that you did not violate any ethics rules required by this course and KAIST. You can either scan a printed version into a PDF file or make the Word document into a PDF file after filling it out. Please sign on the document and submit it along with your other files.

Discussions with other students are permitted and encouraged. However, for writing down the solutions, such discussions (except with course staff members) are no longer acceptable: you must write down your own solutions independently. If you received any help, you must specify on the top of your written homework any individuals from whom you received help, and the nature of the help that you received. *Do not, under any circumstances, copy another person's solution.* We check all submissions for plagiarism and take any violations seriously.

1 Link Analysis (35 points)

(a) [10 pts] Solve the following problems, which are based on the exercises in the Mining of Massive Datasets 3rd edition (MMDS) textbook.

- Exercise 5.1.2 (5 points)
  Compute the PageRank of each page in Fig 5.7, assuming $\beta = 0.8$.
  You can use programs for simple calculations. If you use any code to solve the problem, please attach your code in `hw3.pdf` with a brief explanation.

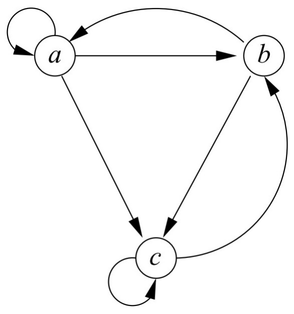


Figure 5.7: An example graph for exercises

- Exercise 5.3.1 (5 points)
  Compute the topic-sensitive PageRank for the graph of Fig 5.15, assuming $\beta = 0.8$ and the teleport set is:

  (a) A only

  (b) A and C

  You can use programs for simple calculations. If you use any code to solve the problem, please attach your code in `hw3.pdf` with a brief explanation.

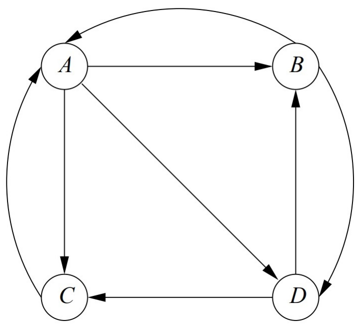


Figure 5.15: Repeat of example Web graph

(b) [25 pts] Implement the PageRank algorithm using Spark.

Implement the PageRank algorithm described in MMDS Chapter 5.2.2 using Spark.

Use the dataset (graph.txt[1]) in HW3 zip file, provided in KLMS. The graph is randomly generated and has 1000 nodes and about 8000 edges with no dead ends. For each row, the left page id represents the source and the right id represents the destination. If there are duplicate edges from one page to another, treat them as the same.

You may set $\beta = 0.9$ and start from the vector $\mathbf{v}$ initialized as all 1's divided by the number of pages. You do not have to break the transition matrix M into stripes. Run your algorithm for 50 iterations to produce the final vector $\mathbf{v}$ and return the ids of the top-10 pages with the highest PageRank scores.

Please use **command-line** arguments to obtain the file path of the dataset. (Do not fix the path in your code.) For example, run:

```
bin/spark-submit hw3_1.py path/to/graph.txt
```

Your code should `print` the top-10 page ids with the highest PageRank scores in $\mathbf{v}$. Print up to **5 decimal digits** for the PageRank scores. The output format is the following:

```
<PAGE_ID_0><TAB><SCORE_0>
<PAGE_ID_1><TAB><SCORE_1>
...
<PAGE_ID_9><TAB><SCORE_9>
```

The output should be 10 lines and sorted in **descending order.**

[1]This dataset is from Stanford University.

2 Neural Nets and Deep Learning (35 points)

(a) [10 pts] Compute the gradients using the chain rule

Figure 2 shows a simple neural network which consists of two input nodes, two hidden nodes and two output nodes. Given input data x_1, x_2, we want the network to predict the target label y_1, y_2 correctly. In other words, the network outputs o_1, o_2 are equal to the target labels y_1, y_2. In order to train the network using gradient descent, we have to compute the gradient of the loss function with respect to the parameters of the network using the backpropagation algorithm.

Suppose we use the sigmoid activation function in the hidden and output layers. Also say we use the mean squared error for the loss function. Express the answer using the parameters in Figure 2.

- Compute the gradient of the loss with respect to w^2_{ij} $(i, j = 1, 2)$
- Compute the gradient of the loss with respect to w^1_{ij} $(i, j = 1, 2)$

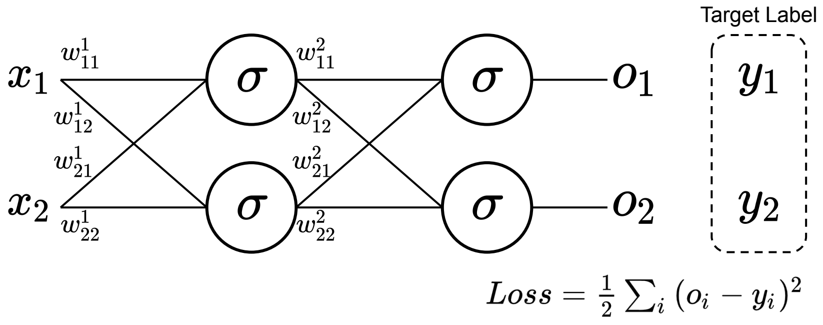


Figure 2: A simple neural network

(b) [25 pts] Implement a fully-connected network to distinguish digits using Python.

Use the dataset (mnist_sample.zip[2]) in HW3 zip file, provided in KLMS. The dataset is sampled from the MNIST dataset, which is one of the most common datasets used for image classification and contains 70,000 grayscale images of handwritten digits (0 to 9). Each image has 28 x 28 resolution and is transformed to a one-dimensional vector of length 784.

- `training.csv`: Used for the model training and contains 100 data points for each digit. Each line contains a vector of length 784 and the corresponding label.
- `testing.csv`: Used for the model testing. Uses the same format as `training.csv`.

Construct a fully-connected network, which has an input layer, one hidden layer, and an output layer. First, convert the label to a vector of size 10 using **one-hot**

[2]THE MNIST DATABASE of handwritten digits.

encoding where the vector element corresponding to the label has a 1 while all the other elements are 0. For example, 0 is encoded as [1, 0, 0, 0, 0, 0, 0, 0, 0, 0] while 5 is encoded as [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]. One-hot encoding is commonly used to deal with categorical values. Then, implement the gradient descent algorithm using forward and backward propagation to train the network. Use the sigmoid activation function and mean squared error loss function as in (a).

We provide a simple skeleton code (`hw3_2_sample.py`) for better understanding, but feel free to modify it as needed. You will have to try different numbers of iterations and learning rates while monitoring the loss and accuracy to find a hyperparameter setting that results in high accuracy. We define accuracy as the fraction of predictions the network got right (range in [0, 1]).

Please **use command-line** arguments to obtain the file path of the dataset. (Do not fix the path in your code.) For example, run:

```
python hw3_2_p3.py path/to/training.csv path/to/testing.csv
```

After training the network, your code should `print` the accuracy on both training and test data as well as the the number of iterations and learning rate η. Typically, the test accuracy is lower than the training accuracy. The output format is the following:

```
<TRAINING ACCURACY>
<TEST ACCURACY>
<NUMBER OF ITERATIONS>
<η>
```

For example,

```
0.941
0.820
5000
0.001
```

We will give full credit as long as the test accuracy is reasonably high (> 0.7).

3 Graph Representation Learning (30 points)

(a) [30 pts] Implement the node2vec algorithm

We learned node2vec, which is an algorithm for learning an embedding vector for each node in a graph. For each node u, it samples a set of neighbors $\Phi(u)$ as the **context** of u following its own sampling algorithm. Then, for each node $v \in \Phi(u)$, the model is learned to maximize the likelihood of v given the embedding of u.

Figure 1 shows the architecture of node2vec for computing the likelihood of each context node, where we provide more details of its component below:

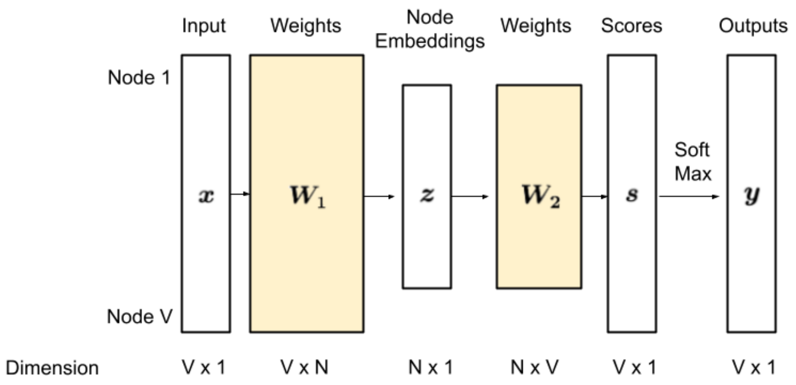


Figure 1: Architecture of node2vec

- Input: The input $\boldsymbol{x}$ is a V-dimensional one-hot vector, representing each node in the given graph, where V is the number of nodes. For example, if we have three nodes, we have $[1, 0, 0]$, $[0, 1, 0]$, and $[0, 0, 1]$.
- Node embedding: With a $V \times N$ weight matrix $\boldsymbol{W}_1$, the input $\boldsymbol{x}$ is converted to a N-dimensional node embedding $\boldsymbol{z}$ such that $\boldsymbol{z} = \boldsymbol{W}_1^\top \boldsymbol{x}$.
- Score: With another $N \times V$ weight matrix $\boldsymbol{W}_2$, node embedding $\boldsymbol{z}$ is mapped back into a V-dimensional vector $\boldsymbol{s}$ such that $\boldsymbol{s} = \boldsymbol{W}_2^\top \boldsymbol{z}$. Each element of $\boldsymbol{s}$ can be considered as the "score" of each node to be a neighbor of $\boldsymbol{x}$.
- Output: To convert the score vector $\boldsymbol{s}$ into a probability vector $\boldsymbol{y}$, the softmax function is applied to $\boldsymbol{s}$.

The goal of training is to maximize the log likelihood $\log p(v|\boldsymbol{W}_1\boldsymbol{x})$, where $v \in \Phi(u)$ is a context node and $\boldsymbol{W}_1\boldsymbol{x}$ is the embedding of the target node u. Suppose that v is the i-th node in the graph. We formulate the objective function to minimize as

$$\mathcal{L}(u, v) = -\log p(v|\boldsymbol{W}_1\boldsymbol{x}) = -\log y_i = -\log\left(\frac{\exp(s_i)}{\sum_{k=1}^{V}\exp(s_k)}\right)$$

Implement the **deterministic version** of node2vec and run it on the graph dataset used in problem 1(b). In this version, neighborhood sampling is done by the depth-first search (DFS) algorithm without randomness, assuming that the search is done in an increasing order of the integer indices of nodes.

For example, in a graph with edges $\{(0,1),(1,2),(2,3)\}$, DFS starting from node 2 is $[2,1,0,1,2,3,2]$. Note that we assume a random walker who follows the DFS trajectory, and thus the same node can be visited multiple times while the walker goes back to the previous node to visit another new node.

Also, your implementation should satisfy the following rules:

- Set node embedding dimension $N = 128$, and window size $W = 2$.
- For neighborhood sampling, run the random walk only once for each node, where the length of walk sequence is set to $L = 5$.
- Don't use negative sampling, and use the softmax function.
- During gradient descent, update the parameters by iterating through the random walk sequences in a fixed order. For example, starting from node 1 (assuming that the first index is 1), we update the parameter matrices $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ for all adjacent pairs of nodes appearing in the sequence, and then we go to the random walk sequence generated for node 2, and so on.
- Use a learning rate of 0.01.
- Repeat this process for $K = 3$ times on all random walk sequences.

Please **use command-line** arguments to obtain the file path of the dataset. (Do not fix the path in your code.) For example, run:

```
python hw3_3_p3.py path/to/graph.txt
```

We provide a skeleton code (`hw3_3_sample.py`), and fill in the blanks following the hints. The output format is the following:

```
<DFS-based embedding of node 5>
<DFS-based embedding of node 10>
```

As the embedding is lengthy (i.e., 128-dim), print the **first element** of each embedding, up to **5 decimal digits**. For example,

```
0.91252
-0.75905
```

Answers to Frequently Asked Questions

§ General FAQ

- For HW3, we will only allow you to to import **pyspark, re, sys, math, csv, time, collections, itertools and os** in your code. Numpy will only be allowed in 2(b) and 3(b).

- An example of 5 decimal digits is 123.12345.

- Please use the print **built-in function** in Python to produce outputs.

- `<TAB>` means `\t` in the Python print function.

§ About Problem 1

- Time limit: Codes of 1(a) (optional), 1(b) should run within **30 minutes**.

- Q: In problem 1(b), I cannot use Numpy. Then can I use pyspark.ml.linalg?
  A: We did not allow Numpy because we wanted to implement matrix multiplication in parallel (i.e. each worker performs a single multiplication). If you can implement parallel multiplication with pyspark.ml.linalg you may use it.

- Q: In problem 1(b), the print condition is print top 10 items and print the score with 5 decimal digits. I wonder considering 5 digits is just for the print statement or should it be considered before choosing the top10 item?
  A: Since the 5 decimal digit specification is only mentioned for the printing format, it should be considered only for the print statement (although it might not make much of a difference if considered before choosing top10 item).

- Q: In problem 1(b), do we have to round up the 5th digit when we reduce the decimals to 5 digits only? or just print the first 5 decimal digits as they are without considering the rounding?
  A: Both approaches will be considered correct.

- Q: In problem 1(b), I am working on haedong machine. I think my algorithm is correct, but I keep receiving "java.lang.OutOfMemoryError: unable to create new native thread". Even though when I use smaller data file, I receive this error when the number of iteration is > 5. (When iteration ≤ 5, the calculated rank is the same as when I use only numpy.)
  A: For out-of-memory issues, you may check the followings in your code: (1) number of processes you are using, (2) whether you are saving too large matrix, (3) and

usage of collect or top functions. We provide more details for each item in below.
(1) We limited the number of threads per student so that everyone can use the Haedong server. Please try to modify the PageRank algorithm or configure the number of threads generated to avoid this issue.
(2) Matrix multiplication should not use more memory per iteration. You might be saving too much in memory. Also, make sure you don't use all of matrix M in each tuple generated by the map function.
(3) RDDs in spark lazily computes the map reduce functions. Once you use collect or top (functions that computes the results) spark allocates workers to the jobs. If you try to compute all 50 iterations in a single map reduce sequence without using collect or top, the computation graph becomes too large for the system to handle. Make sure to generate check points for some number of iterations and reconstruct the rdds rather than computing the entire 50 iterations in a single run.

- Q: In problem 1(b), if we do not implement the matrix multiplication by spark RDD methods, and if the answer is correct, will there still be some partial points?
  A: We will test your code via spark-submit. If you submit your code as a regular python file, you will not get any credit.

§ About Problem 2

- Time limit: There is no time limit; it does not take much time if you have implemented it correctly.

- Q: In problem 2(a), I'm confused about the meaning of "parameters" we can use to express the answer. Since the o_1 and o_2 are the outputs, I guess we should use only w's and x's, but that will make the equation more complicated. Are we allowed to use o's in the answer?
  A: You can use x, w, o, y for your answer.

- In problem 2(b),
  - You can add more hidden layers.
  - You can add a bias vector.
  - You can use variants of gradient descent algorithm (e.g., mini-batch gradient descent, stochastic gradient descent).
  - You can report the number of epochs instead of the number of iterations.

- Q: In problem 2(b), we were asked to use Sigmoid and MSE for the classification of MNIST tasks instead of Cross entropy + Softmax, which seems a bit awkward, though possible. Here, we were told to use the one-hot-vector computation for the labels of y, but shouldn't we not use the one-hot-vector to calculate the error? I thought that if using MSE, it would be reasonable to see the differences of actual value and predicted value to optimize. If so, how do we calculate the accuracy in

this case? Should we get the vector and choose the element that corresponds to the highest probability and then compare them?
A: Our intention of using MSE was to simplify the problem as much as possible, and to maintain consistency with problem 3(a). Therefore, please use MSE instead of Cross entropy + Softmax. To calculate accuracy, you need to convert a model's output to a label (e.g., choose the index with the highest probability).

- Q: In problem 2(b), when we calculate the MSE should we compare the actual values (i.e. predicted:3 by converting label and real value:4 and computing their squared diff) or just comparing two vector differences and summing their squared sum?
A: For training the model, you may compute MSE using two models' outputs, not labels.

- Q: In problem 2(b), there are some random weights to generate Neural Networks. In this case, I have random accuracy for many tries, and some of them could not exceed 0.7. When you give score to this, does this code just tested at once? Also in this case, should we find the stable iteration and learning rate for all cases?
A: Actually, the basic setting in the problem is enough to achieve 0.7 test accuracy. But if the accuracy in your solution variates a lot, one possible solution is to set the random seed to a specific value, and report it in your writeup file. This will guarantee the same output when scoring your code. Also, you can design your own network if you want (i.e, add more layers, use other activation functions, use other ways to define initial weights, etc).

- Q: In problem 2(b), is the test data random or same as given test data when scoring?
A: We will use the same test dataset for scoring, but make sure that you should not use the test data in the model training step.

- Q: In problem 2(b), there is initial weight that is generated randomly in skeleton code. Can I fix the initial weight instead of randomly generating it?
A: Sure, it's just a skeleton code. You can implement your own version.

- Q: In problem 2(b), the accuracy changes during several iteration. Do I have to print the last acc? or the highest acc (both for train and test accuracy)?
A: As you said, you can find out the optimal number of iterations with your own setting, which gives the highest accuracy. Please report both train/test accuracy, by setting the number of iterations to that optimal one (which will result in the last accuracy to be equal with your highest accuracy).

§ About Problem 3

- Time limit: Python code of 3(a) should run within 15 minutes.