Chạy file  
python graphsage.unsupervised\_train.py --train\_prefix C:/nam/work/temp/data/example\_data/ppi --model graphsage\_mean --max\_total\_steps 1000 --validate\_iter 10  
  
  
### Overview  
  
This directory contains code necessary to run the GraphSage algorithm.  
GraphSage can be viewed as a stochastic generalization of graph convolutions, and it is especially useful for massive, dynamic graphs that contain rich feature information.  
See our [paper](https://arxiv.org/pdf/1706.02216.pdf) for details on the algorithm.  
  
\*Note:\* GraphSage now also has better support for training on smaller, static graphs and graphs that don't have node features.  
The original algorithm and paper are focused on the task of inductive generalization (i.e., generating embeddings for nodes that were not present during training),  
but many benchmarks/tasks use simple static graphs that do not necessarily have features.  
To support this use case, GraphSage now includes optional "identity features" that can be used with or without other node attributes.  
Including identity features will increase the runtime, but also potentially increase performance (at the usual risk of overfitting).  
See the section on "Running the code" below.  
  
The example\_data subdirectory contains a small example of the protein-protein interaction data,  
which includes 3 training graphs + one validation graph and one test graph.  
The full Reddit and PPI datasets (described in the paper) are available on the [project website](http://snap.stanford.edu/graphsage/).  
  
If you make use of this code or the GraphSage algorithm in your work, please cite the following paper:  
  
 @inproceedings{hamilton2017inductive,  
 author = {Hamilton, William L. and Ying, Rex and Leskovec, Jure},  
 title = {Inductive Representation Learning on Large Graphs},  
 booktitle = {NIPS},  
 year = {2017}  
 }  
  
### Requirements  
  
Recent versions of TensorFlow, numpy, scipy, and networkx are required (but networkx must be <=1.11). To guarantee that you have the right package versions, you can use [docker](https://docs.docker.com/) to easily set up a virtual environment. See the Docker subsection below for more info.   
  
#### Docker  
  
If you do not have [docker](https://docs.docker.com/) installed, you will need to do so. (Just click on the preceding link, the installation is pretty painless).   
  
You can run GraphSage inside a [docker](https://docs.docker.com/) image. After cloning the project, build and run the image as following:  
  
 $ docker build -t graphsage .  
 $ docker run -it graphsage bash  
  
or start a Jupyter Notebook instead of bash:  
  
 $ docker run -it -p 8888:8888 graphsage  
  
You can also run the GPU image using [nvidia-docker](https://github.com/NVIDIA/nvidia-docker):  
  
 $ docker build -t graphsage:gpu -f Dockerfile.gpu .  
 $ nvidia-docker run -it graphsage:gpu bash   
  
### Running the code  
  
The example\_unsupervised.sh and example\_supervised.sh files contain example usages of the code, which use the unsupervised and supervised variants of GraphSage, respectively.  
  
If your benchmark/task does not require generalizing to unseen data, we recommend you try setting the "--identity\_dim" flag to a value in the range [64,256].  
This flag will make the model embed unique node ids as attributes, which will increase the runtime and number of parameters but also potentially increase the performance.  
Note that you should set this flag and \*not\* try to pass dense one-hot vectors as features (due to sparsity).  
The "dimension" of identity features specifies how many parameters there are per node in the sparse identity-feature lookup table.  
  
Note that example\_unsupervised.sh sets a very small max iteration number, which can be increased to improve performance.  
We generally found that performance continued to improve even after the loss was very near convergence (i.e., even when the loss was decreasing at a very slow rate).  
  
\*Note:\* For the PPI data, and any other multi-ouput dataset that allows individual nodes to belong to multiple classes, it is necessary to set the `--sigmoid` flag during supervised training. By default the model assumes that the dataset is in the "one-hot" categorical setting.  
  
  
#### Input format  
As input, at minimum the code requires that a --train\_prefix option is specified which specifies the following data files:  
  
\* <train\_prefix>-G.json -- A networkx-specified json file describing the input graph. Nodes have 'val' and 'test' attributes specifying if they are a part of the validation and test sets, respectively.  
\* <train\_prefix>-id\_map.json -- A json-stored dictionary mapping the graph node ids to consecutive integers.  
\* <train\_prefix>-class\_map.json -- A json-stored dictionary mapping the graph node ids to classes.  
\* <train\_prefix>-feats.npy [optional] --- A numpy-stored array of node features; ordering given by id\_map.json. Can be omitted and only identity features will be used.  
\* <train\_prefix>-walks.txt [optional] --- A text file specifying random walk co-occurrences (one pair per line) (\*only for unsupervised version of graphsage)  
  
To run the model on a new dataset, you need to make data files in the format described above.  
To run random walks for the unsupervised model and to generate the <prefix>-walks.txt file)  
you can use the `run\_walks` function in `graphsage.utils`.  
  
#### Model variants  
The user must also specify a --model, the variants of which are described in detail in the paper:  
\* graphsage\_mean -- GraphSage with mean-based aggregator  
\* graphsage\_seq -- GraphSage with LSTM-based aggregator  
\* graphsage\_maxpool -- GraphSage with max-pooling aggregator (as described in the NIPS 2017 paper)  
\* graphsage\_meanpool -- GraphSage with mean-pooling aggregator (a variant of the pooling aggregator, where the element-wie mean replaces the element-wise max).  
\* gcn -- GraphSage with GCN-based aggregator  
\* n2v -- an implementation of [DeepWalk](https://arxiv.org/abs/1403.6652) (called n2v for short in the code.)  
  
#### Logging directory  
Finally, a --base\_log\_dir should be specified (it defaults to the current directory).  
The output of the model and log files will be stored in a subdirectory of the base\_log\_dir.  
The path to the logged data will be of the form `<sup/unsup>-<data\_prefix>/graphsage-<model\_description>/`.  
The supervised model will output F1 scores, while the unsupervised model will train embeddings and store them.  
The unsupervised embeddings will be stored in a numpy formated file named val.npy with val.txt specifying the order of embeddings as a per-line list of node ids.  
Note that the full log outputs and stored embeddings can be 5-10Gb in size (on the full data when running with the unsupervised variant).  
  
#### Using the output of the unsupervised models  
  
The unsupervised variants of GraphSage will output embeddings to the logging directory as described above.  
These embeddings can then be used in downstream machine learning applications.  
The `eval\_scripts` directory contains examples of feeding the embeddings into simple logistic classifiers.  
  
#### Acknowledgements  
  
The original version of this code base was originally forked from https://github.com/tkipf/gcn/, and we owe many thanks to Thomas Kipf for making his code available.  
We also thank Yuanfang Li and Xin Li who contributed to a course project that was based on this work.  
Please see the [paper](https://arxiv.org/pdf/1706.02216.pdf) for funding details and additional (non-code related) acknowledgements.