Application title: GraphSAGE

(DRAFT)

#### Abstract to application - use your own words and explain in 150-300 words the application.

The GraphSAGE application performs inductive representation learning on graphs. Similar to the objective of graph2vec, GraphSAGE performs low-dimensional vector embedding of vertices. The input to the application is an attributed graph (each vertex of the graph is associated with ~1000 attributes). The output is the vector embedding, where each vertex of the input graph is converted to a low-dimensional vector. Different from graph2vec, GraphSAGE does not learn the embedding vectors directly from the input graph. Instead, it assumes that embedding vectors of any vertex can be derived by applying a multi-layer neural network on the n-hop neighbors. GraphSAGE thus learns such neural network for the entire graph, and is able to generate embedding vectors for vertices unseen in the training.

There are several highlights of this application. First of all, the learning of embedding vectors is inductive, instead of transductive. In other words, if new vertices are added to the graph, GraphSAGE can generate their embedding directly by the learnt multi-layer neural network, without the need of re-training on the new graph. Secondly, GraphSAGE can process well both attributed and un-attributed graphs. The generated embedding vectors capture the information of the graph topology as well the node attributes.

#### Pseudo code (snapshot of pseudo code is acceptable):

The computation in training includes two phases: forward propagation and backward propagation. The computation in embedding generation only involves forward propagation.

Pseudo code for forward propagation:



Backward propagation is calculated similarly. (The Gitlab implementation uses tensorflow to implement both forward and backward propagation. Once you build the forward chain of computation by the above pseudo code, the backward propagation is handled internally by tensorflow. )

#### What programming model is being used for the application: vertex centric, edge centric, GAS, BLAS?

Vertex centric

#### Can this application be implemented in Hornet & HornetNests with the current set of primitives? Yes/No (highlight one)

Yes. But only for the embedding generation part, not for the training part.

##### If yes, can you show at highlevel how you would do this?

When learning the neural network, the graph is static. After the neural network is learnt, dynamicity comes into picture. New nodes/edges keep getting added, and we apply the neural network on the n-hop neighbors of the newly added nodes to get their embedding. Thus, there is a need for maintaining the dynamic graph CSR.

#### Key primitives needed. Highlight primitives that are missing in current framework. Recall the three level of primitives that we have: Analyst (highest level), algorithmic, architecture (low level):

Dense matrix-matrix multiplication (for learning neural networks) (algorithmic)

(Min-cut) graph partitioning (algorithmic)

Neighbor sampling (algorithmic)

#### What extra utilities do we need to supply? For each utility needed explain why it is needed. Such utilities can be sorting algorithms, parallel prefix summations...

#### Is this application a good fit for PCPM? Yes/No (highlight one)

Partially yes.

##### Explain your answer if possible:

We can use the partitioning method in PCPM for constructing mini-batches.

#### Communication Vs. Computation ratio - collect some information on average bytes per operation, messages across the network…

#### Suppose each node samples s neighbors. The feature vector length for each node is f.

Approximately, we will need to collect s x f data to perform f x f computation.

So the bytes per op is s/f. Say s=10 and f=1000, so s/f=0.01.

#### Implementation details of the application given to us by DARPA:

1. Language: python
2. External dependencies if these exists: TensorFlow, networkx
3. Input (and input format)

#### Profiling results (key insights):

1. Application is computation bound.
2. The most time-consuming computation is performed by the tensorflow matrix-matrix multiplication operation.

#### Additional references. For each reference added, add a bullet point explaining why it is important. Other applications in the same space are good as well.: