Capsule Graph Neural Network

Zhang et al., ICLR 2019

2020/06/10

Jae-Won Chung

Seoul National University

Outline

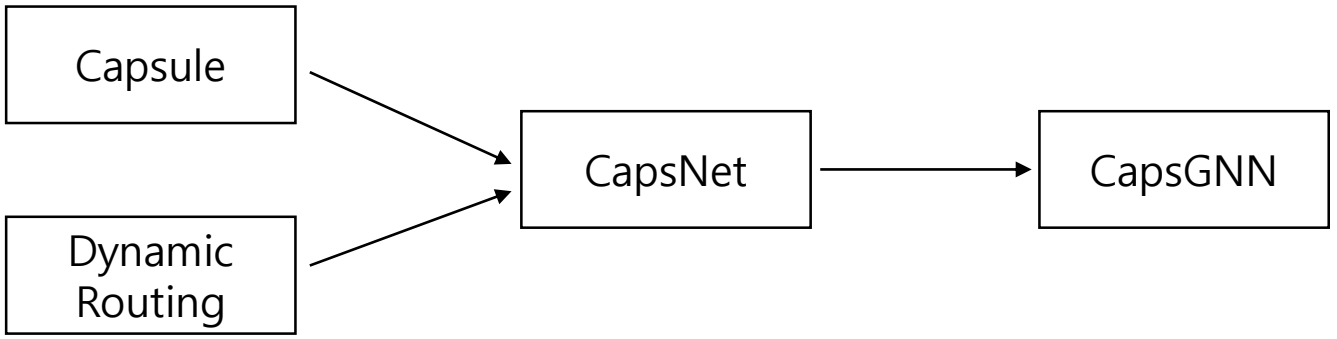
1. Background: Capsule, CapsNet
2. CapsGNN: Motivation, Architecture
3. Evaluation: Graph Classification, Capsule Efficiency
4. Conclusion and Critique

Background: Capsule

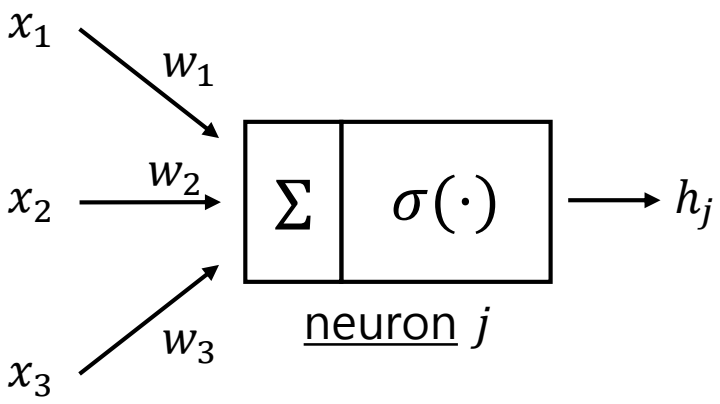
Transforming Auto-encoders, Hinton et al., 2011

Dynamic Routing between Capsules, Sabour et al., 2017

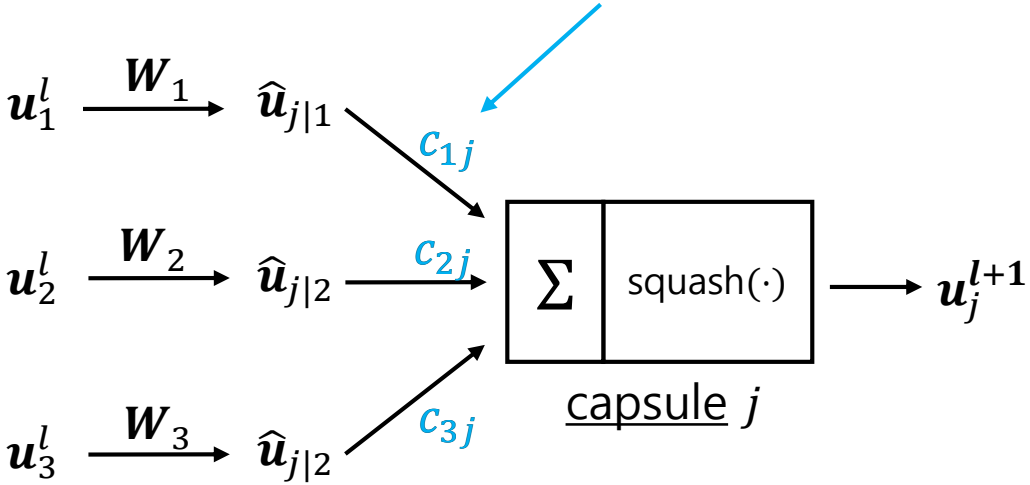
Capsule Graph Neural Network, Zhang et al., 2019



Background: Capsule

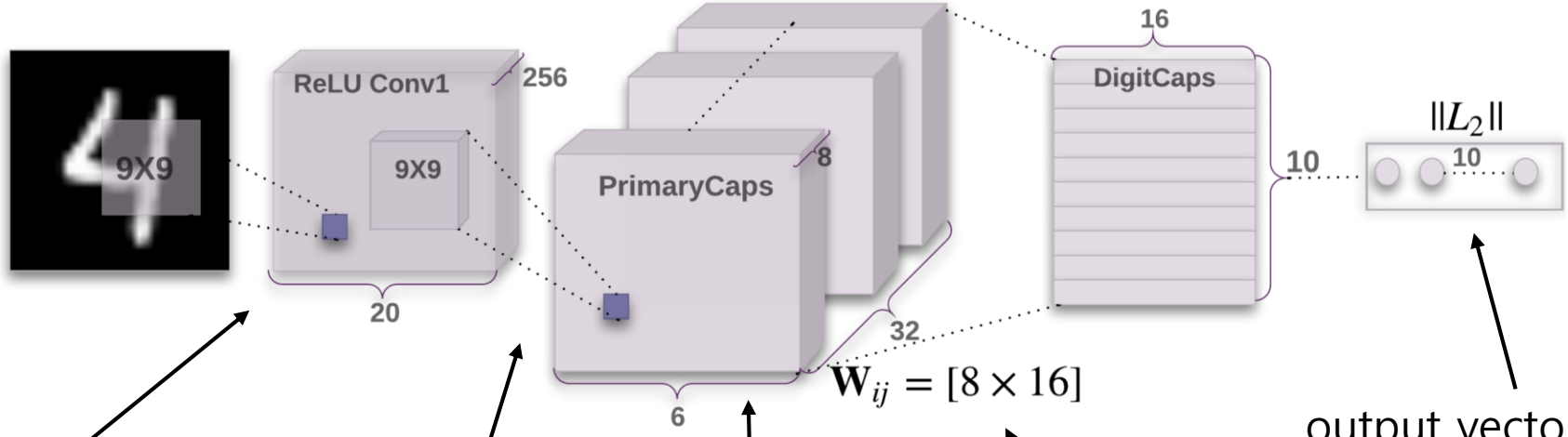


- scalar in scalar out
  - intensity of filter match
- max pool on h_j loses information



- vector in vector out
  - length: the probability of existence
  - orientation: the entity's current state
- no max pool; preserves information

Background: CapsNet



9x9 conv, 256
activated by ReLU

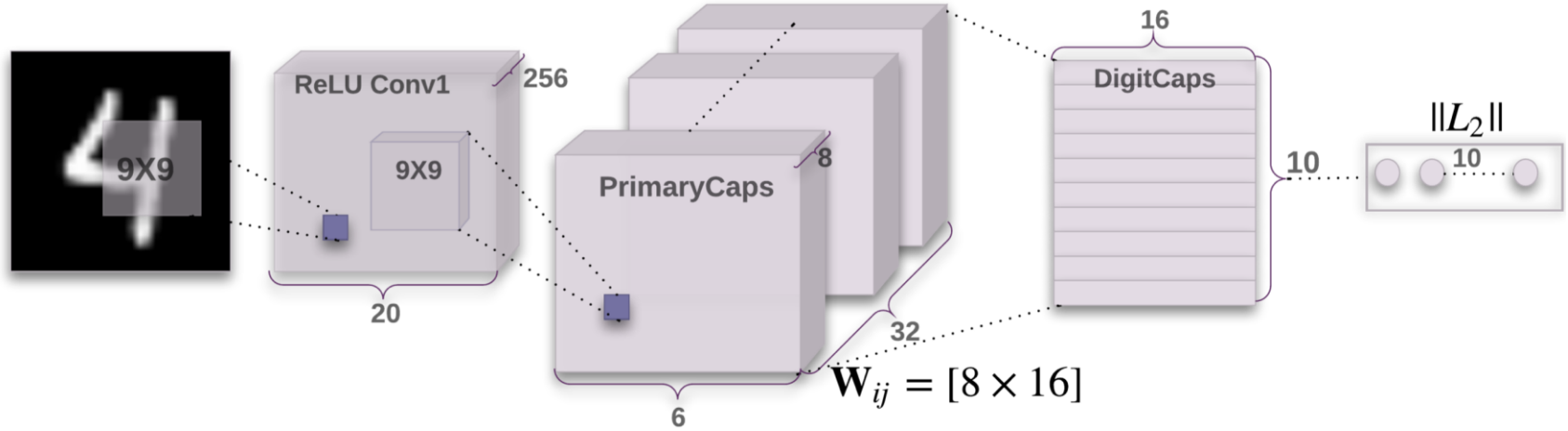
9x9 conv, 4 x 8
activated by squash

viewed as outputs
from 4 capsules
with dimension 8

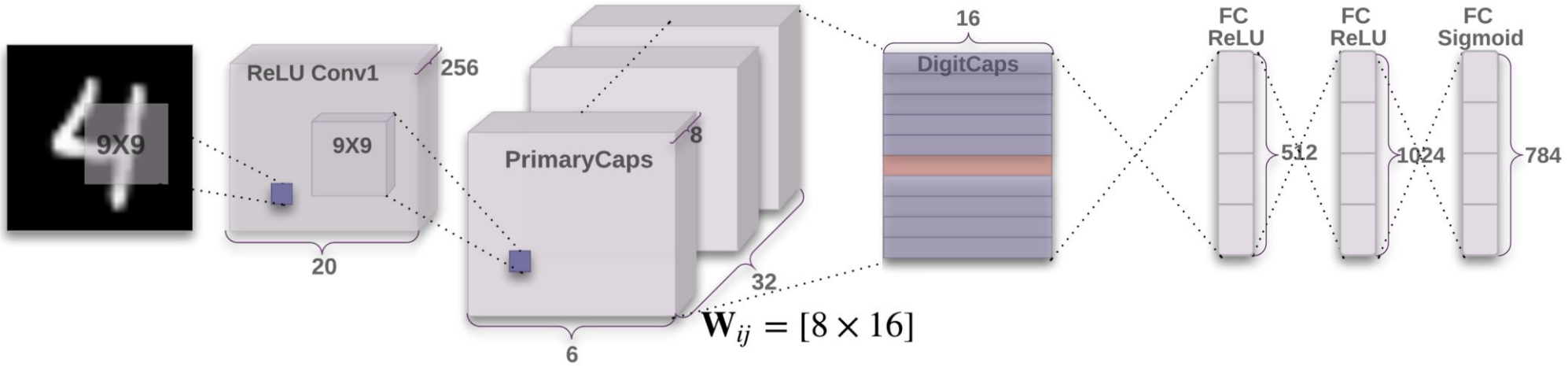
dynamic routing
capsules: 4 → 10
dimension: 8 → 16

output vector length
is probability!

Background: CapsNet



→ Hinge loss



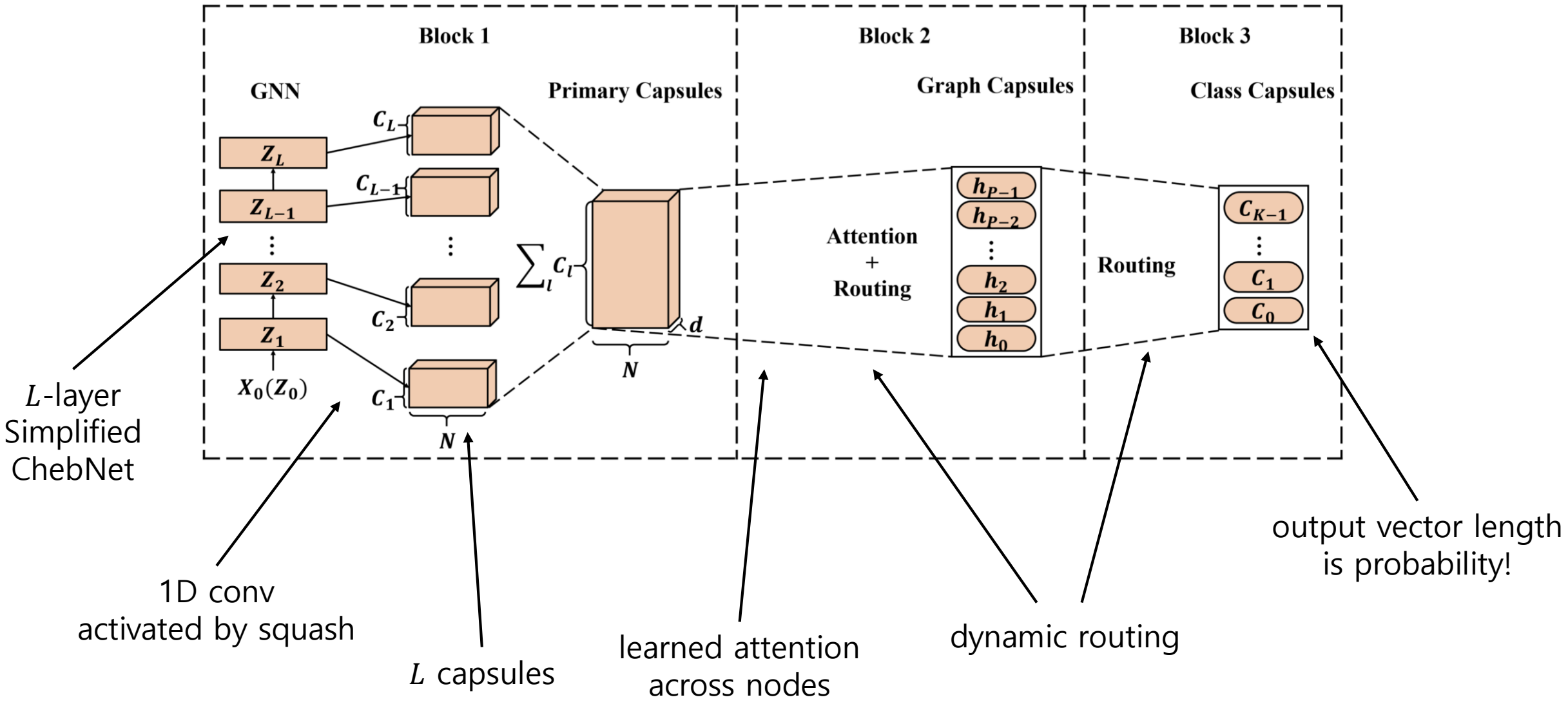
→ Recon. loss

CapsGNN: Motivation

- Simple aggregations in GCNs treat the learned node embeddings as merely an array of scalar features, rather than a meaningful vector in the feature space.
  - Element-wise max-pooling on the node features [Zhang et al., 2018]
  - Taking the element-wise covariance across nodes [Verma & Zhang, 2018]

- Graph embeddings must also capture the position of and structure around each node. Scalar neuron outputs lack representation power.

CapsGNN: Architecture



Capsule Graph Neural Network, Zhang et al, 2019

Evaluation: Graph Classification

biological datasets

Algorithm	MUTAG	NCI1	PROTEINS	D&D	ENZYMES
WL	82.05±0.36	**82.19±0.18**	74.68±0.49	**79.78±0.36**	52.22±1.26
GK	81.58±2.11	62.49±0.27	71.67±0.55	78.45±0.26	32.70±1.20
RW	79.17±2.07	>3days	74.22±0.42	>3days	24.16±1.64
Graph2vec	83.15±9.25	73.22±1.81	73.30±2.05	-	-
AWE	**87.87±9.76**	-	-	71.51±4.02	35.77±5.93
DGK	87.44±2.72	80.31±0.46	75.68±0.54	73.50±1.01	53.43±0.91
PSCN	**88.95±4.37**	76.34±1.68	75.00±2.51	76.27±2.64	-
DGCNN	85.83±1.66	74.44±0.47	75.54±0.94	**79.37±0.94**	51.00±7.29
ECC	76.11	76.82	-	72.54	45.67
GCAPS-CNN	-	**82.72±2.38**	**76.40±4.17**	77.62±4.99	**61.83±5.39**
CapsGNN	86.67±6.88	78.35±1.55	**76.28±3.63**	75.38±4.17	**54.67±5.67**

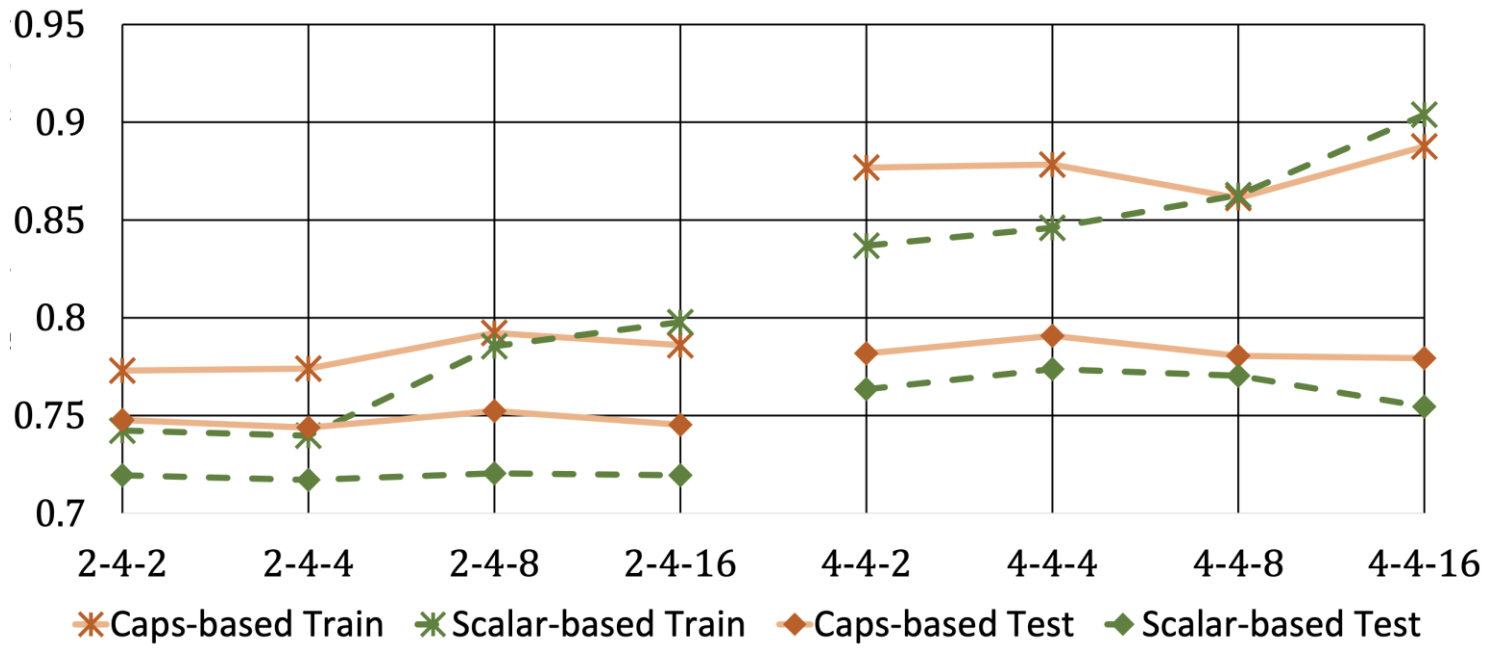
social datasets

Algorithm	COLLAB	IMDB-B	IMDB-M	RE-M5K	RE-M12K
WL	**79.02±1.77**	**73.40±4.63**	49.33±4.75	49.44±2.36	38.18±1.30
GK	72.84±0.28	65.87±0.98	43.89±0.38	41.01±0.17	31.82±0.08
DGK	73.09±0.25	66.96±0.56	44.55±0.52	41.27±0.18	32.22±0.10
AWE	73.93±1.94	**74.45±5.83**	**51.54±3.61**	**50.46±1.91**	39.20±2.09
PSCN	72.60±2.15	71.00±2.29	45.23±2.84	49.10±0.70	**41.32±0.42**
DGCNN	73.76±0.49	70.03±0.86	47.83±0.85	48.70±4.54	-
GCAPS-CNN	77.71±2.51	71.69±3.40	48.50±4.10	50.10±1.72	-
CapsGNN	**79.62±0.91**	73.10±4.83	**50.27±2.65**	**52.88±1.48**	**46.62±1.90**

approaches or surpasses SOTA on 6 of 10 datasets

Evaluation: Capsule Efficiency

- *scalar network*
  - replace capsule layers with fully-connected layers
  - adjust weights s.t. the number of trainable weights is similar



Conclusion and Critique

- Used capsules for graph classification.
- Extended CapsNet with clear reasons and proved its efficacy through SOTA performance.

- No ablation study of the attention module.
- Comparison with GCAPS-CNN[Verma & Zhang, 2018] is incomplete.

Code

Code: Introduction

- The author's implementation is available.
  - https://github.com/XinyiZ001/CapsGNN
  - Capsules based on https://github.com/naturomics/CapsNet-Tensorflow
- This guide will walk you through:
  - graph classification datasets
  - GEXF, a standard format for graph data
  - graph data preprocessing
  - tensorflow implementation of CapsGNN
  - how to run the model + my results
- Emphasis is on providing a kick-start for those new to graph classification.

Code: Graph Classification Datasets

- Bio datasets: composed of N discrete labels, each with C classes
  - MUTAG (binary): chemical → 7 types of mutagenic effects
  - ENZYMES: enzyme → 6 EC (enzyme commission) of 3 top-level classes
  - NCI1 (binary): chemical → anticancer effects on 37 tumor types
  - PROTEINS (binary): protein → 3 protein properties (helix, sheet, turn)
  - D&D (binary): protein → 82 protein classes

Code: Graph Classification Datasets

- Bio datasets: SOTA as of 2020.06.19 (source: paperswithcode)
  - MUTAG (binary): 95.00% (G_Inception)
  - ENZYMES (6 classes): 78.39% (DSGCN-allfeat)
  - NCI1 (binary): 87.2% (WKPI-kmeans)
  - PROTEINS (binary): 84.91% (HGP-SL)
  - D&D (binary): 95.67% (U2GNN-unsupervised)

Code: Graph Classification Datasets

- Social datasets: one label with C classes
  - COLLAB: collaboration graph of a researcher → one of 3 research domains
  - IMDB: actor graph of a movie → which genre is this movie?
    - 2 versions: binary (IMDB-B) and 3 classes (IMDB-M)
  - Reddit: user comment graph of a thread → which sub-reddit is this from?
    - 3 versions: binary (RE-B), 5 classes (RE-M5K), and 11 classes (RE-M11K)

Code: Graph Classification Datasets

- Social datasets: SOTA as of 2020.06.19 (source: paperswithcode)
  - COLLAB (3 classes): 95.62% (U2GNN-unsupervised)
  - IMDB
    - IMDB-B (binary): 93.5% (U2GNN-unsupervised)
    - IMDB-M (3 classes): 74.8% (U2GNN-unsupervised)
  - Reddit
    - RE-B (binary): 92.4% (GIN-0)
    - RE-M5K (5 classes): 57.5% (GIN-0)
    - RE-M11K (11 classes): 49.75% (GFN-light)

Code: Graph Classification Datasets

- GEXF formats of the datasets can be obtained at:
  - https://drive.google.com/drive/folders/1qXx-OZlJtgRYn579aQX13ou2hutqJz41?usp=sharing

- What is GEXF?

Code: Standard Graph Data Formats

- NetworkX: a widely used python package for graph data processing
  - Supported graph formats
    - https://networkx.github.io/documentation/stable/reference/readwrite/index.html#

- GEXF is a standard graph data format supported by NetworkX.
  - Graph Exchange XML Format
  - See https://gephi.org/gexf/format/ for full specification

Code: GEXF

```
<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://www.gexf.net/1.2draft" version="1.2">
    <meta lastmodifieddate="2009-03-20">
        <creator>Gexf.net</creator>
        <description>A hello world! file</description>
    </meta>
    <graph mode="static" defaultedgetype="directed">
        <nodes>
            <node id="0" label="Hello" />
            <node id="1" label="Word" />
        </nodes>
        <edges>
            <edge id="0" source="0" target="1" />
        </edges>
    </graph>
</gexf>
```

list of nodes

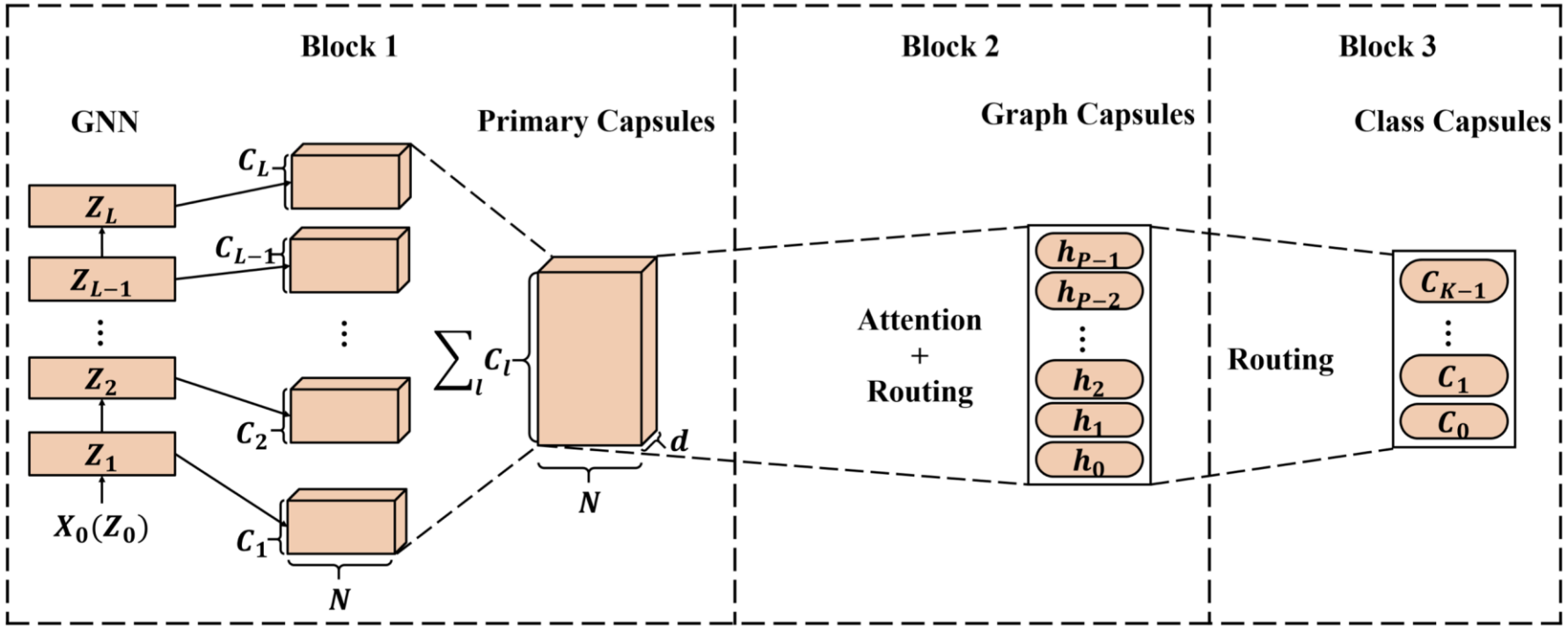
list of edges

Code: Preprocessing for TensorFlow

- See `dataset_utils/preprosessing.py`
- Preprocessing data flow:
  1. Read every `.gexf` file with NetworkX (here).
  2. For each node in graph, parse according to attribute list (here).
  3. Create the graph's adjacency matrix and add self-connections (here).
     - Used the COO (coordinate format) sparse matrix format for compression!
  4. Final preprocessing results for each graph are (here):
     - adjacency matrix
     - node input attributes
     - node reconstruction attributes (for reconstruction loss)
     - graph classification labels
  5. Serialize and save with the python built-in `pickle` module (here).

Code: TensorFlow Model Implementation

- See `capsule_utils/GraphCap_net.py`



node embedding

node attention

graph embedding

class capsules

Code: TensorFlow Model Implementation

- Node embedding
  - The number of channels (`Ci`, `Co`) and GCN layers (`L`) are configured here.
  - The node embedding dimension (`d`) is configured here.
  - The node embedding operations are encapsulated here.
    1. Transform channel size from `Ci*d` to `Co*d` with `tf.layers.conv2d`.
    2. Perform message-passing by multiplying the normalized transition matrix.
    3. Activate output with `tf.nn.tanh`.
    4. Reshape to shape `(Batch, N, Co, d)`, where `N` is the number of nodes.

Code: TensorFlow Model Implementation

- Node attention
  - The node attention operations are encapsulated here.
    1. Input and output tensor shapes are both `(Batch, N, C, d)`.
    2. Input features are transformed with two consecutive `tf.layers.dense`.
    3. The first FC layer reduces the channel by a factor of 16, and the second restores it.
    4. Attention weights are generated after a final node-dimension softmax operation.
    5. The input tensor is attended by multiplying the attention weights and returned directly.

Code: TensorFlow Model Implementation

- Graph embedding
  - Code allows the configuration of the number of graph embedding layers, alth ough the paper used only one layer. Seems like the authors once tried multip le graph embedding layers.
  - The graph embedding operations are encapsulated here.
    1. Two large operations: transforming the input matrix (here), and dynamic routing (here) .
    2. To input features to capsules, they should be evenly split by the number of capsules a nd transformed with the weight matrix to match capsule input dimensions.
    3. A single weight matrix is generated with `tf.contrib.slim.variable`, and repeated `Batch*N` times. The input matrix is transformed with `tf.matmul`.
    4. Code for dynamic routing was borrowed from the CapsNet implementation here.
    5. Capsule outputs are activated with `squash`, an activation function for vectors.

Code: TensorFlow Model Implementation

- Class capsules
  - The class capsule operations are encapsulated here.
    1. The implementation is essentially the same as the graph embedding layer.
    2. Differences come from the name of configuration parameters.
    3. The output of class capsules are used for classification.

Code: TensorFlow Model Implementation

- Loss
  - The lambda value for the margin loss can be configured here.
    - This is important! The authors note that they set this value to 1.0 for binary classification tasks, and 0.5 for multi-class classification tasks.
  - The scaling factor on the reconstruction loss can be configured here.
  - The loss function is encapsulated here.
    - The implementation of margin loss (used as classification loss) is straightforward.
    - The reconstruction loss is the mean L2 error between the actual and reconstructed input .
    - The input is reconstructed by 1) extracting the capsule output of the correct class, 2) pas sing it through two `tf.contrib.layers.fully_connected` layers, and 3) activating wit h `tf.nn.sigmoid`.
    - The final loss is `margin_loss + scale * recon_loss`.

Code: TensorFlow Model Implementation

- Training
  - The training code is standard and straightforward. Implemented here.
  - Used Adam as optimizer and applied 0.9 exponential learning rate decay ev ery 20000 steps.
  - The base learning rate and decay rate can be configured here.

Code: Running the Model

- Environment
  - I wrote a Dockerfile that handles all dependencies. Find it here.
  - You will need to add the NVIDIA container runtime to utilize GPUs. See here.
  - Run the container with:
    - `docker build -t capsgnn .`
    - `docker run -it --gpus all -v <path to capsgnn>:/workspace --name caps capsgnn bash`
    - I find it sensible to *mount* the code directory to the container, since we want to keep the preprocessed data even when the container crashes.

Code: Running the Model

- Preprocessing the ENZYMES dataset
  - ```
    python dataset_utils/preprocessing.py    \
      --dataset_input_dir graph_gexf/ENZYMES \
      --output_data_dir data_plk             \
      --pickle_v 3                           \
      --x_fold 10                            \
      --gen_split_file True
    ```
  - This creates data for 10-fold cross validation, which is what the authors did.
    ```

Code: Running the Model

- Training CapsGNN on ENZYMES
  - ```
    python main.py                      \
      --dataset_dir data_plk/ENZYMES \
      --epochs 3000                     \
      --lambda_val 0.5
    ```
  - This trains CapsGNN for 3000 epochs.
    ```

Code: Results

- I’m a poor undergraduate, without enough computation power. ☹
- I ran training and inference only on the ENZYMES dataset.
- The training script produces a large accuracy log file. I wrote a script to extract the test-set accuracy of the moment the validation-set acc uracy is highest, for each cross-validation run. Find the script here.
- My result
  - 60.00, 61.67, 50.00, 41.67, 61.67, 60.00, 58.33, 51.67, 41.67, 50.00
  - average: 53.67, std: 7.41
- Paper claims
  - average: 54.67, std: 5.67