TensorFlow

Research and Future Directions

Chun-Ta Lu & Philip Pham

Proprietary + Confidential

Image Semantic Embedding

Spectrum of semantic similarity

Category-level (coarse-grained) → Fine-grained level → Instance level (ultra fine-grained)

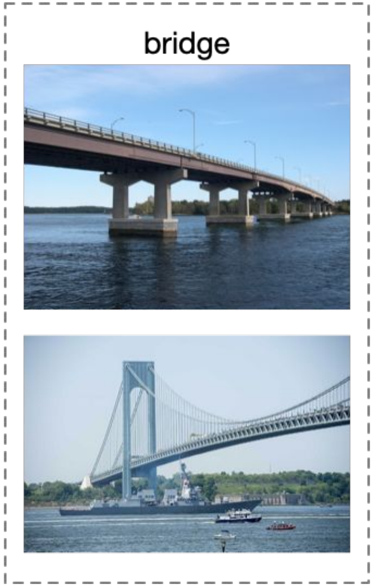


Image embedding:

A dense representation capturing semantics

[Source: Juan, et al., WSDM'20]

Image Semantic Embedding

Spectrum of semantic similarity

Category-level (coarse-grained) | Fine-grained level | Instance level (ultra fine-grained)

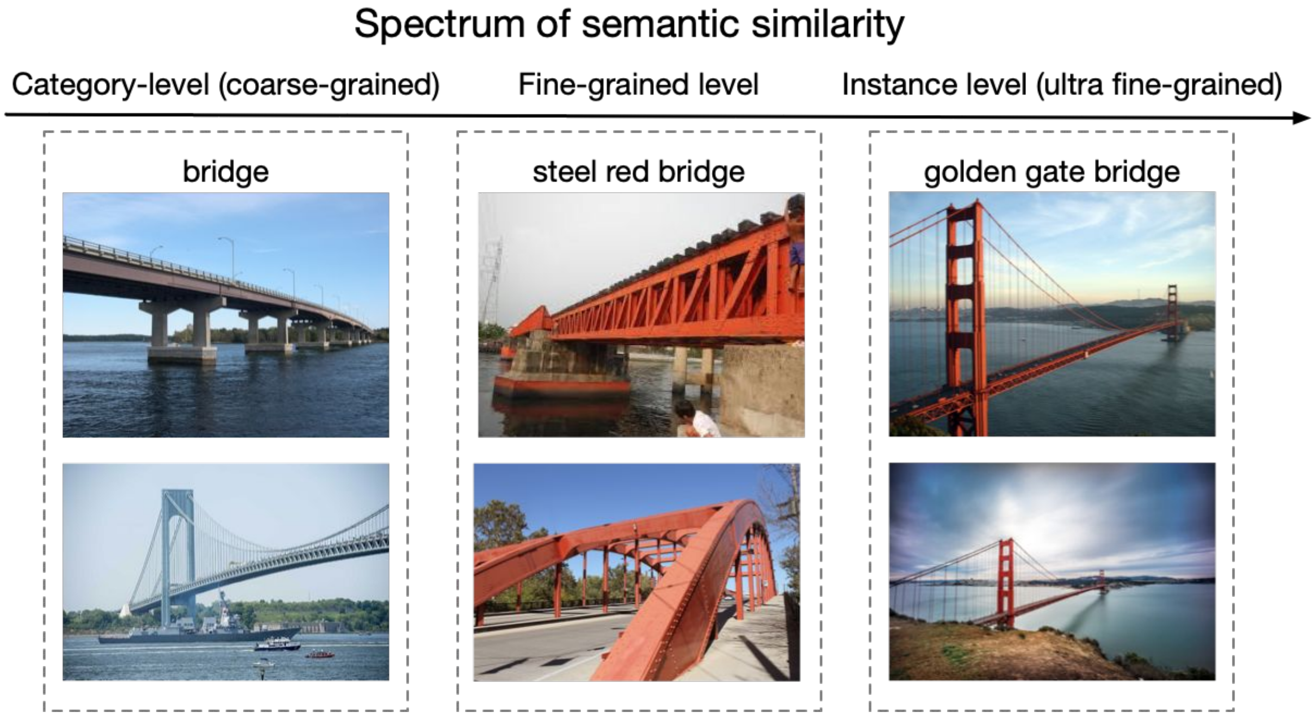
bridge

steel red bridge

[Source: Juan, et al., WSDM'20]

Proprietary + Confidential

Image Semantic Embedding

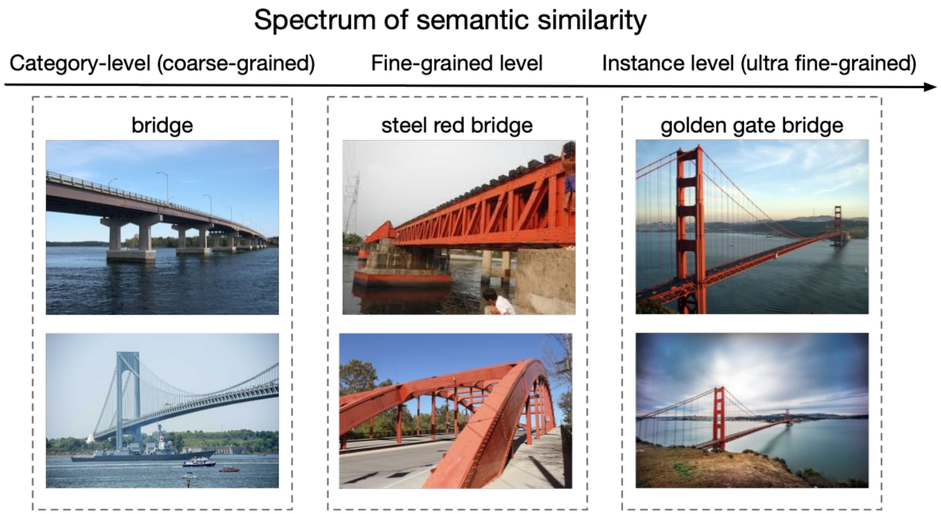


[Source: Juan, et al., WSDM'20]

Learning Image Semantic Embedding

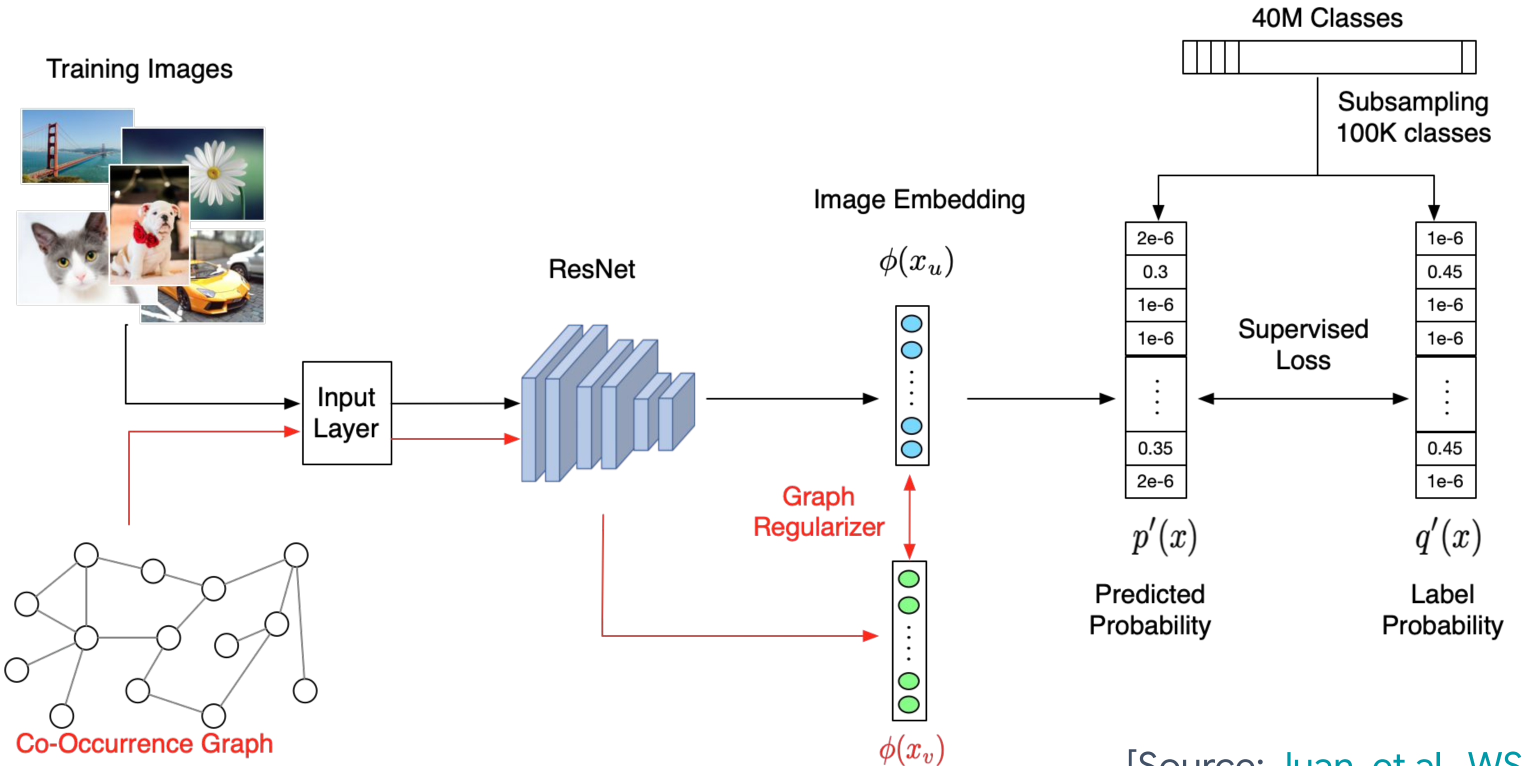
- Embedding to capture semantics in images

- Core of image search
  - By textural queries
  - By image queries



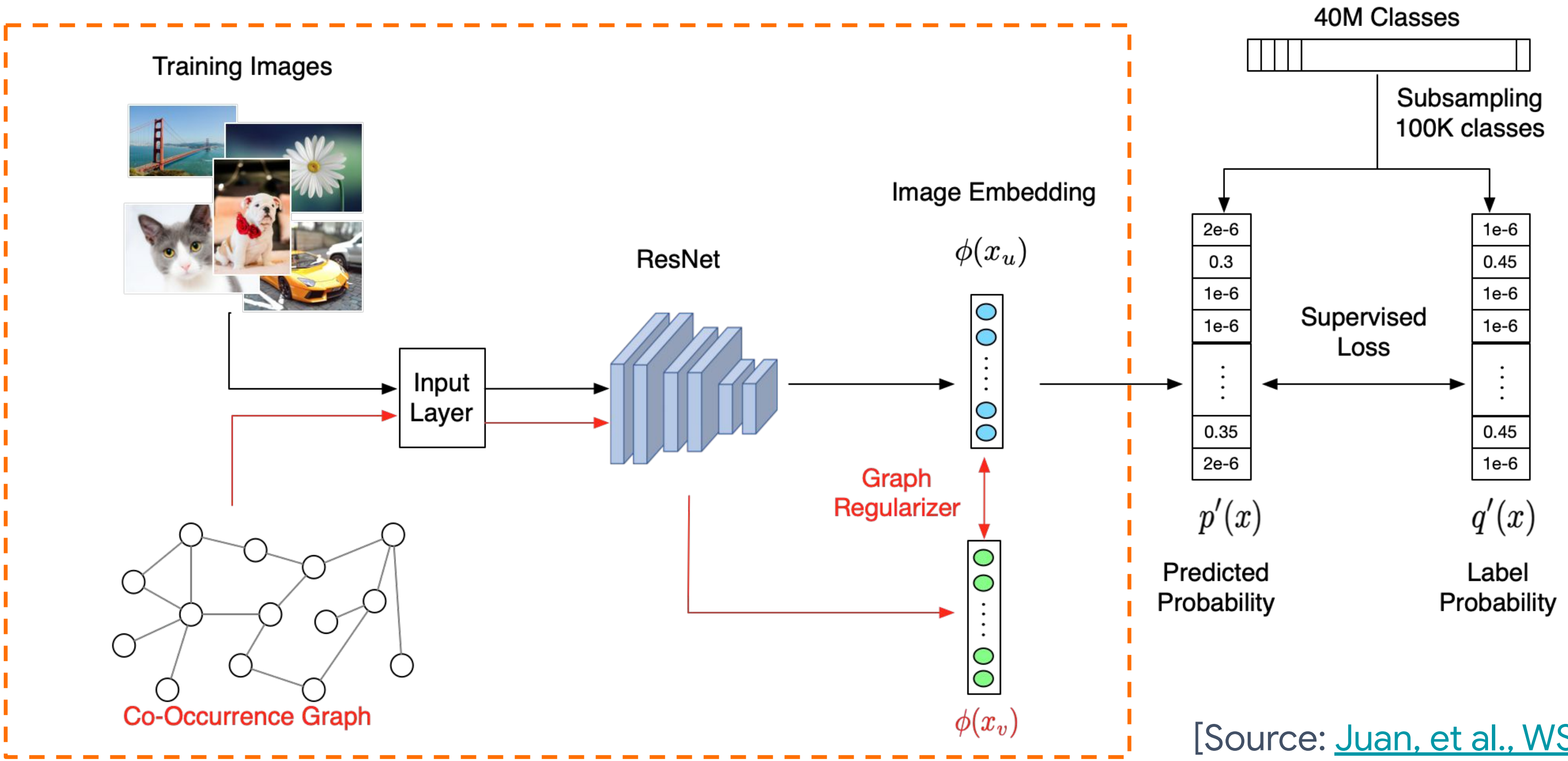
[Source: Juan, et al., WSDM'20]

Neural Architecture



[Source: Juan, et al., WSDM'20]

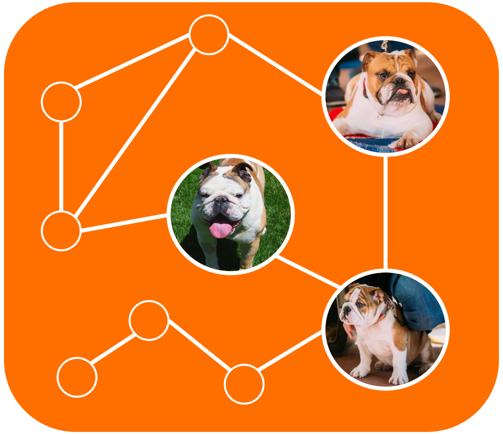
Neural Architecture

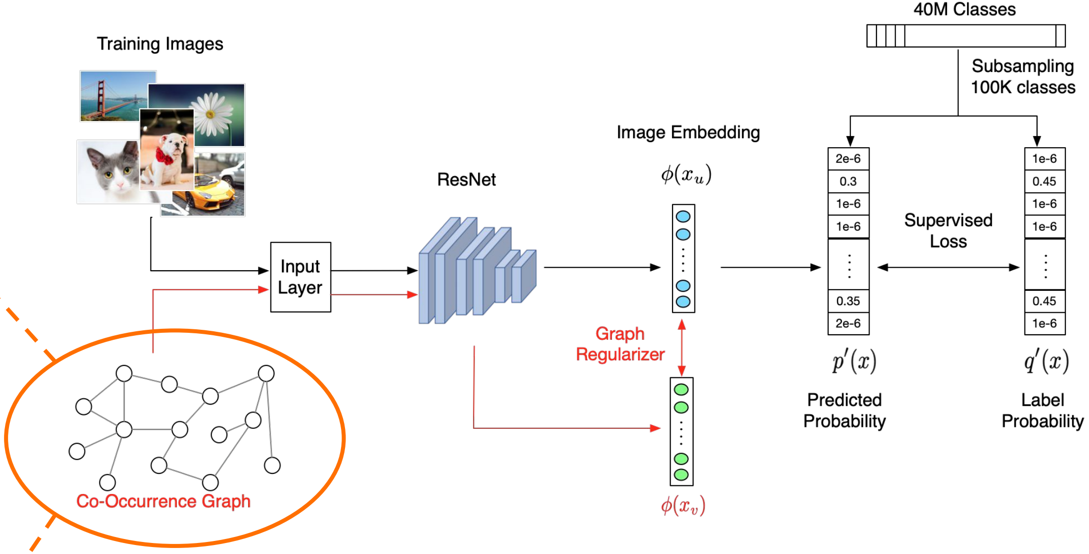


[Source: Juan, et al., WSDM'20]

Neural Architecture

Co-Occurrence Graph





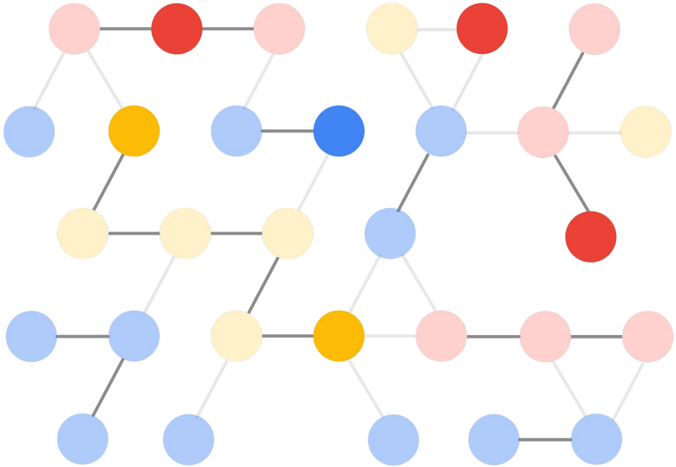
images co-occurring within a search session

[Source: Juan, et al., WSDM'20]

Qualitative Results

[Source: Juan, et al., WSDM'20]

Graph Agreement Models



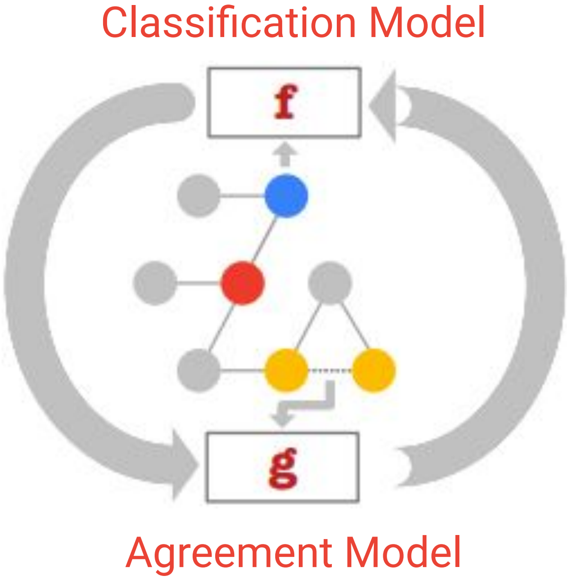
Challenges

- Too few labeled samples
  - overfitting to training data
- Graphs can be noisy
  - edges not relevant to classification task
  - embeddings can be noisy

[Source: Otilia, et al., NeurIPS'19]

Graph Agreement Models

Adds confident predictions to training dataset



Provides regularization for training the classification model

[Source: Otilia, et al., NeurIPS'19]

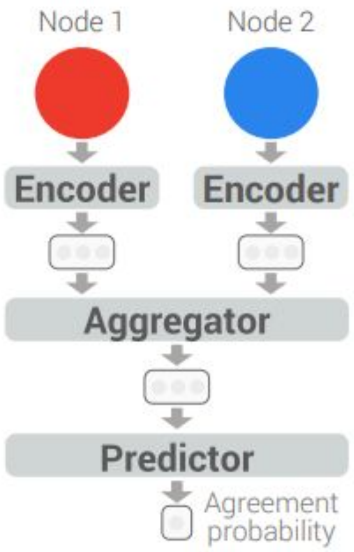
Learn neighbor agreement

Loss function:

$$\mathcal{L}_g = \sum_{i \in L, j \in L, ij \in E} \ell(g(x_i, x_j, w_{ij}), \mathbb{1}_{y_i = y_j})$$

$L =$ labeled nodes set
$E =$ edges set
$x_i =$ features for node i
$f(x_i) =$ predicted label distribution for node i
$\ell =$ loss function (e.g. cross entropy)

Agreement Model



[Source: Otilia, et al., NeurIPS'19]

Classification: use neighbor agreement

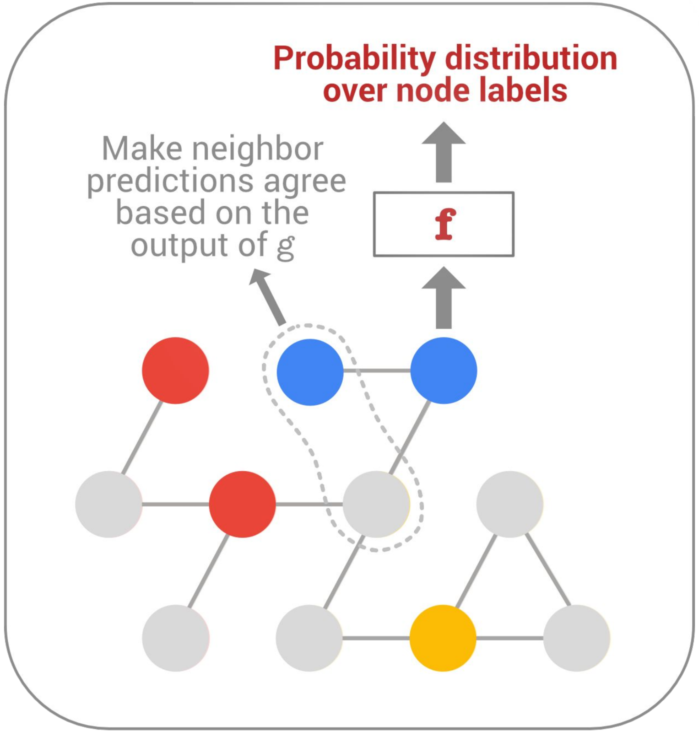
Loss function:

agreement weight

$$\mathcal{L}_f = \sum_{i \in L} \ell(f(x_i), y_i) + \lambda \sum_{\substack{(i,j) \in E \\ i \in L \\ j \in U}} g(x_i, x_j) \ell(f(x_i), f(x_j))$$

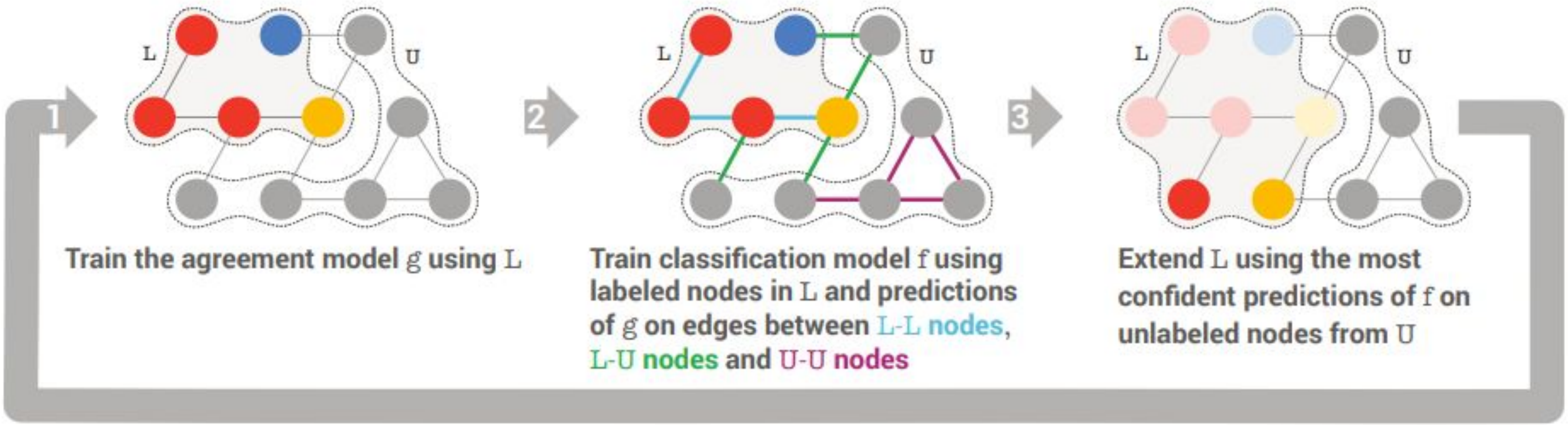
L = labeled nodes set
U = unlabeled nodes set
E = edges set
x_i = features for node i
y_i = true label distribution for node i
$f(x_i)$ = predicted label distribution for node i
$g(x_i, x_j)$ = predicted probability that nodes i and j have similar labels
$\ell(p_1, p_2)$ = distance between label distributions p_1 and p_2
λ = regularization parameter

Classification Model



[Source: Otilia, et al., NeurIPS'19]

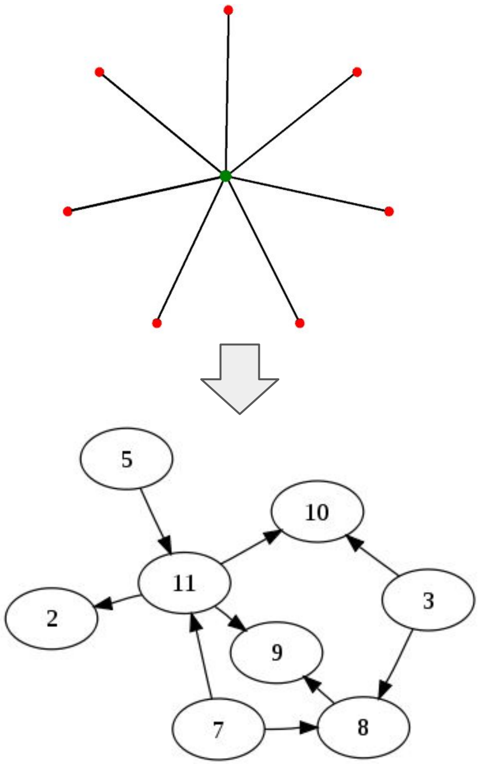
Graph Agreement Models



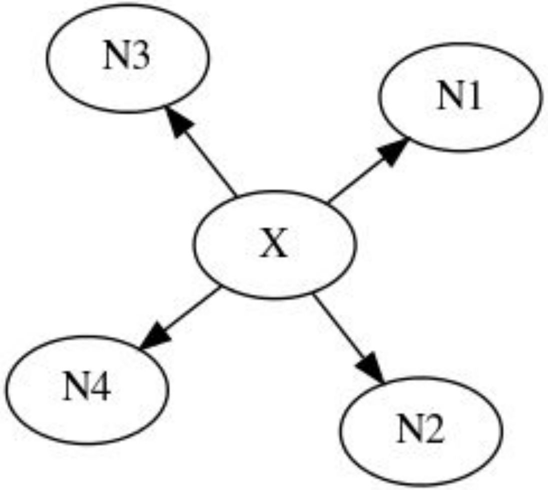
[Source: Otilia, et al., NeurIPS'19]

Beyond Graph Regularization: GNNs

- Graph regularization only incorporates information about a node's neighbors through a distance function.
- There may be more information in other nodes and relationships among neighbors.



Graph Regularization with Message Passing



Distance function

Aggregate distance of neighbors

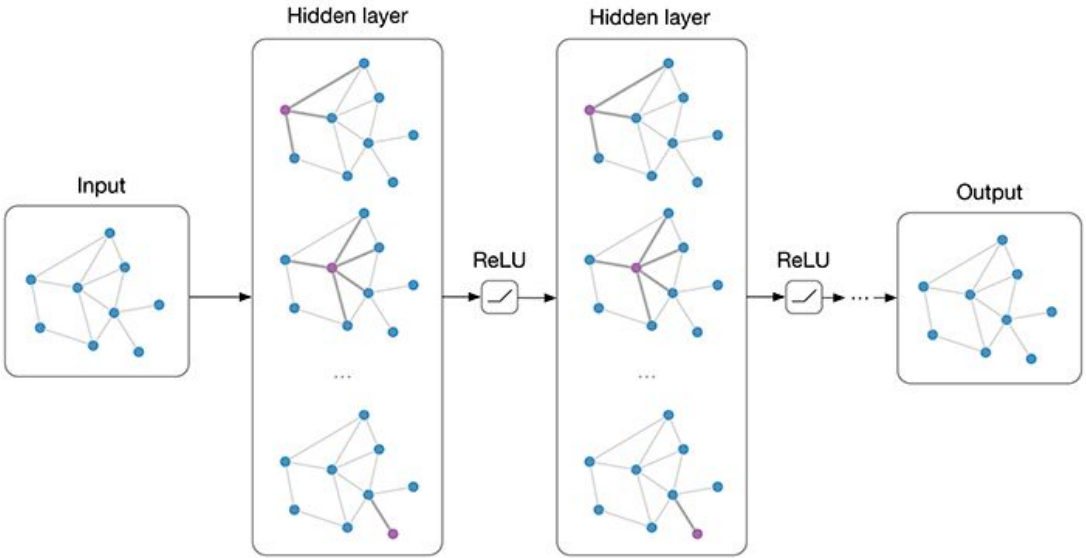
$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

Readout phase normalizes graph loss by weighted degree

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

Gilmer, Justin, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals and George E. Dahl. "Neural Message Passing for Quantum Chemistry." *ArXiv* abs/1704.01212 (2017): n. pag.

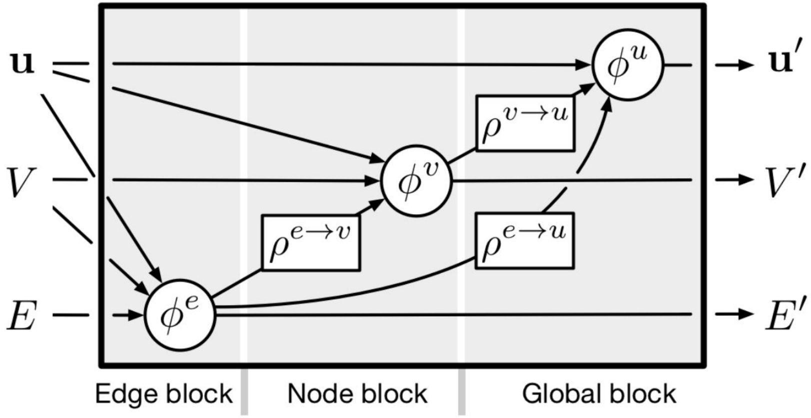
Graph Regularization is a GNN



GNNs use graph relationships to embed nodes, edges, and the graph itself. This framework lets us do computation over arbitrary graphs.

GNNs with GraphNets

- We leverage [Graph Nets] to generalize graph regularization to Graph Neural Networks (GNNs)
- We're able to express these higher-level relationships between neighbors and more distant nodes.




```
model = gnn.GraphRegularizationModel(
        config=graph_reg_config,
        node_model_fn=NodeClassifier)
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
model.fit(train_dataset, epochs=30, validation_data=eval_dataset)
```

Graph Neural Network: GCNs

- With Graph Nets it's easy to implement a Graph Convolutional Network (GCN), which can be a drop-in replacement for GraphRegularizationModel.
- node_model and edge_models are Keras layers.

```
class GraphConvolutionalNodeClassifier(NodeGraphModel):
  """Classifies nodes with a simple Graph Convolutional Network."""

  def __init__(self, seq_length, num_classes, **kwargs):
    # ...

  def graph_call(self, graph, **kwargs):
    # Encode features.
    graph = graph_nets.modules.GraphIndependent(
        node_model_fn=lambda: self._dense_features)(graph)
    # Graph convolutions.
    graph = graph_nets.modules.CommNet(
        edge_model_fn=lambda: self._edge_model1,
        node_encoder_model_fn=lambda: self._node_encoder_model1,
        node_model_fn=lambda: self._node_model1)(graph)
    return graph_nets.modules.CommNet(
        edge_model_fn=lambda: self._edge_model2,
        node_encoder_model_fn=lambda: self._node_encoder_model2,
        node_model_fn=lambda: self._node_model2)(graph)
```