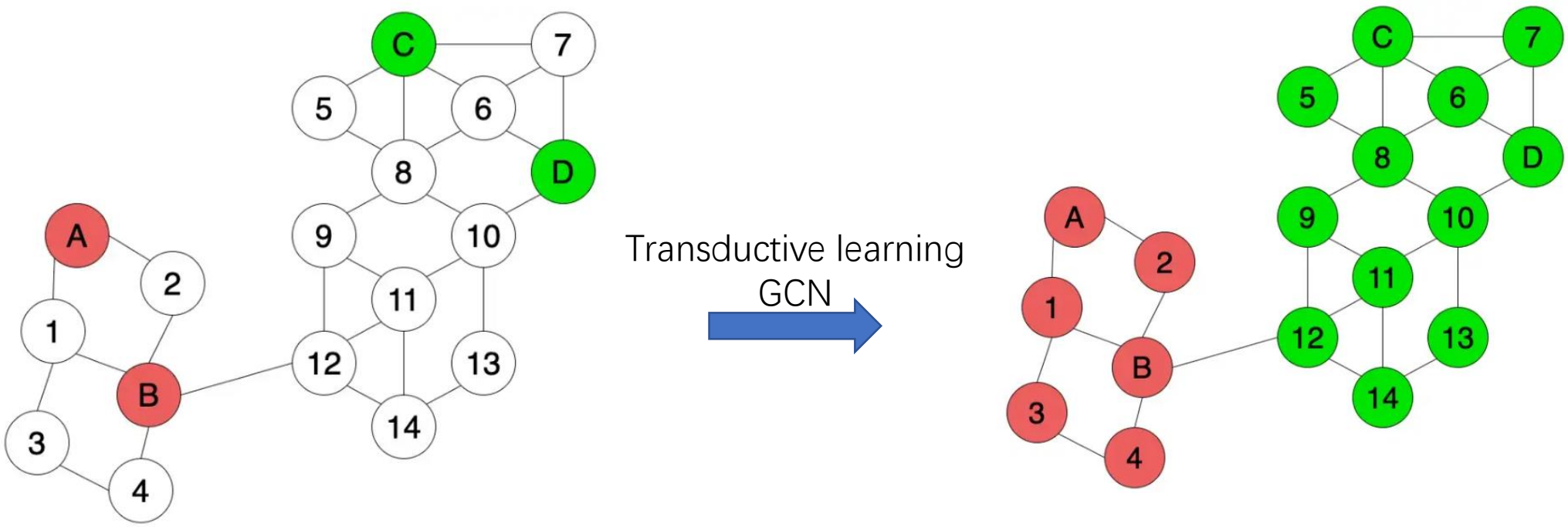
Transductive learning vs Inductive learning

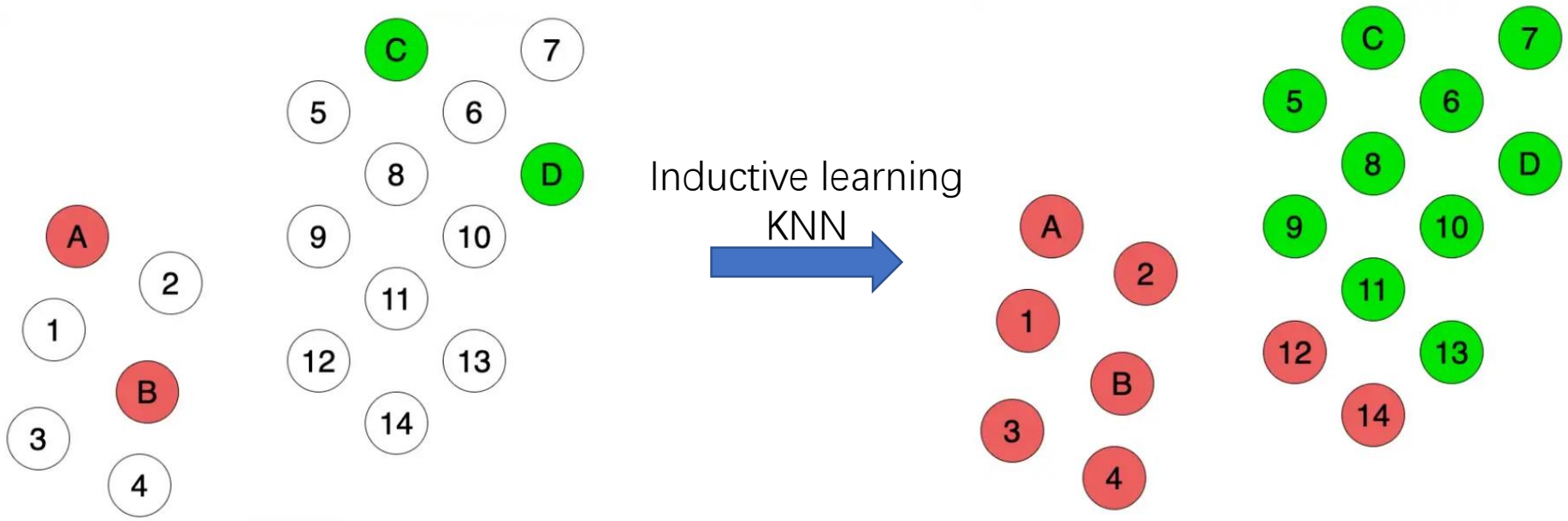
Inductive learning	Transductive learning
Train the model and label unlabelled points which we have never encountered.	Train the model and label unlabelled points which we have already encountered.
Builds a predictive model. If new unlabelled points are encountered, we can use the initially built model.	Does not build a predictive model. If new unlabelled points are encountered, we will have to re-run the algorithm.
Can predict any point in the space of points beyond the unlabelled points.	Can predict only the points in the encountered testing dataset based on the observed training dataset.
Less computational cost.	Can become more computationally costly.

Transductive learning vs Inductive learning



If a new point 15 appears?
Re-train a GCN model.

Transductive learning vs Inductive learning

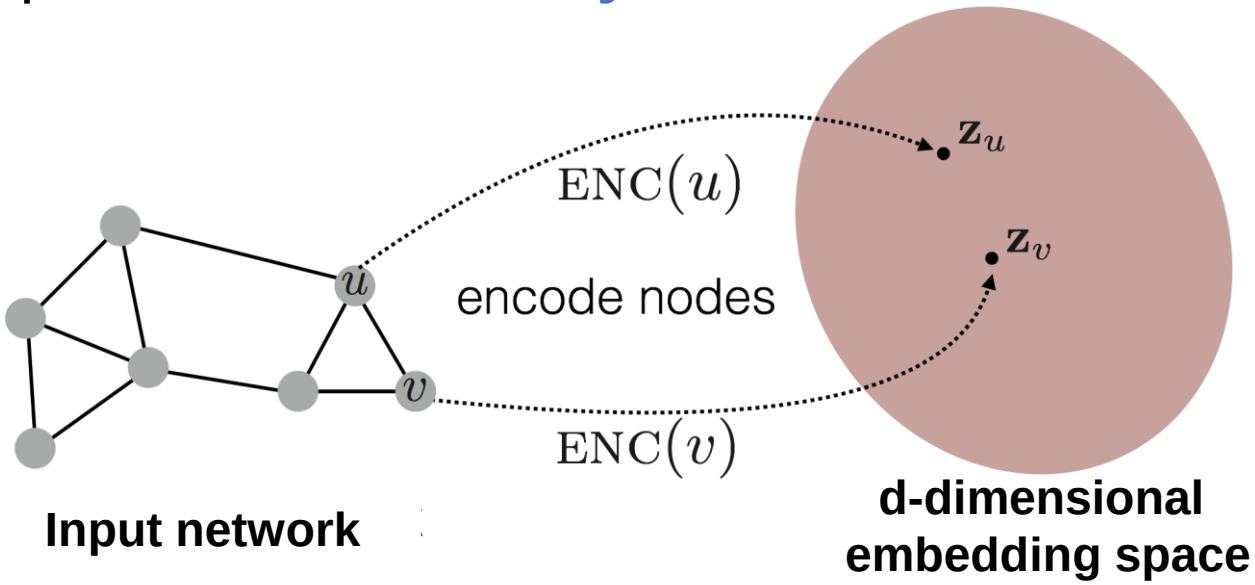


If a new point 15 appears?

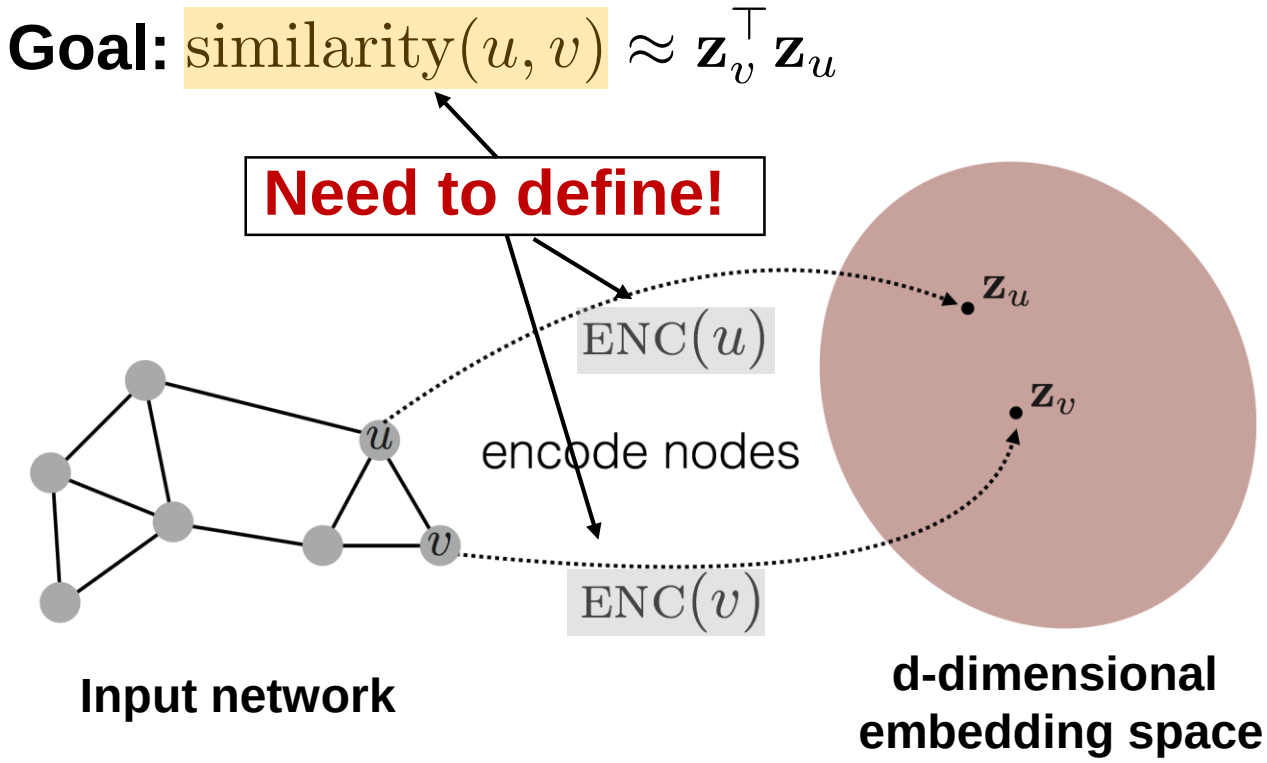
Inductive learning can easily be applied to new data points. Dynamic Graphs

Embedding Nodes

Goal: Map nodes so that **similarity in the embedding space** (e.g., dot product) approximates **similarity in the network**



Embedding Nodes



Embedding Nodes

- **Encoder:** Map a node to a low-dimensional vector:

$$\text{ENC}(v) = \mathbf{z}_v$$

d-dimensional embedding

node in the input graph

- **Similarity function** defines how relationships in the input network map to relationships in the embedding space:

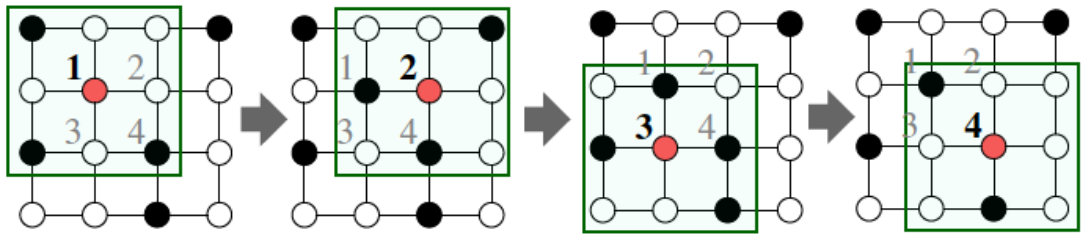
$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

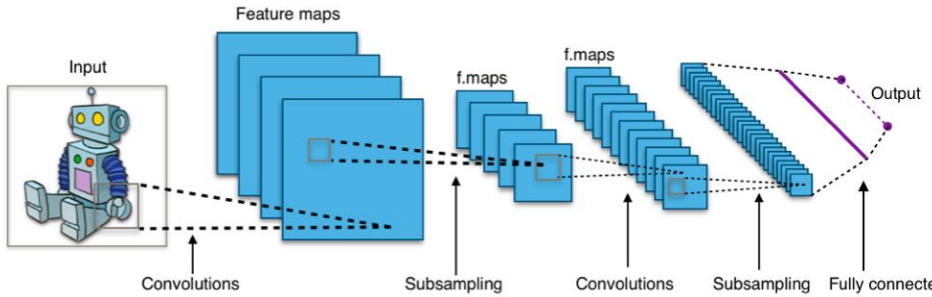
Similarity of u and v in the network

dot product between node embeddings

CNN

CNN on an image:





Goal is to generalize convolutions beyond simple lattices

Leverage node features/attributes (e.g., text, images)

卷积层

- 卷积层通过卷积操作来捕获filters关注的信息。

1	0	0	0	0	1
0	1	0	0	1	0
1	0	0	0	0	1
1	0	1	0	1	1
0	0	0	0	1	1
0	0	1	0	1	1

6X6图像

1	0	0
0	1	0
1	0	0

0	0	1
0	0	1
0	0	1

⋮ ⋮

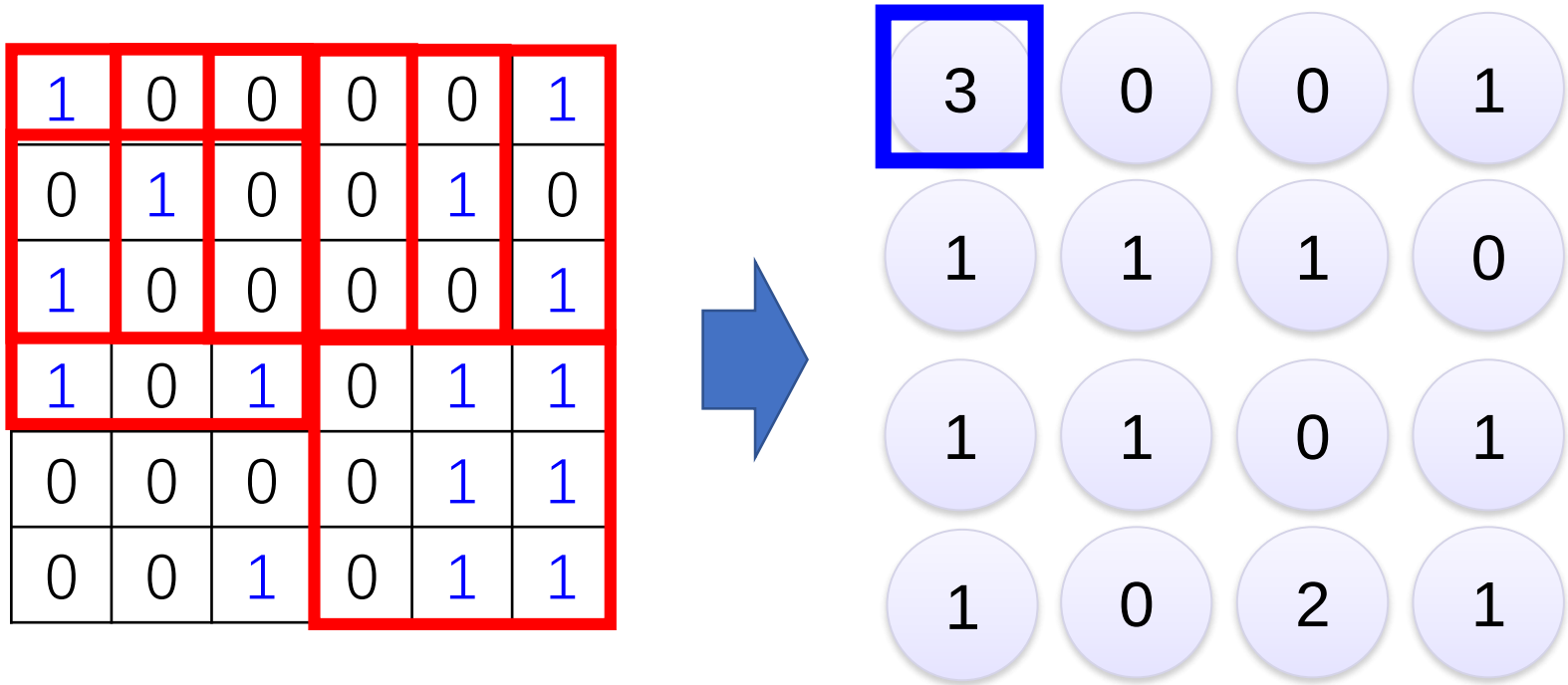
每个filter检测3X3的pattern。

卷积层

- 卷积层通过卷积操作来捕获filters关注的信息。

Filter1

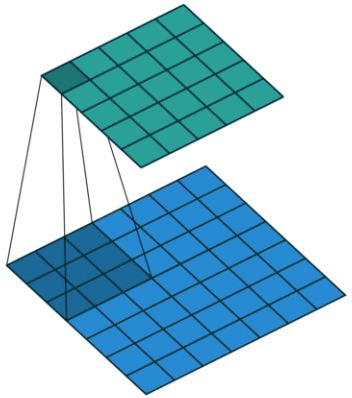
1	0	0
0	1	0
1	0	0



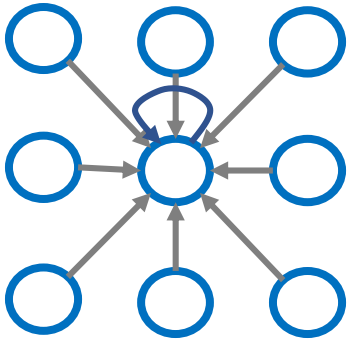
6X6图像

CNN

Single CNN layer with 3x3 filter:



Image

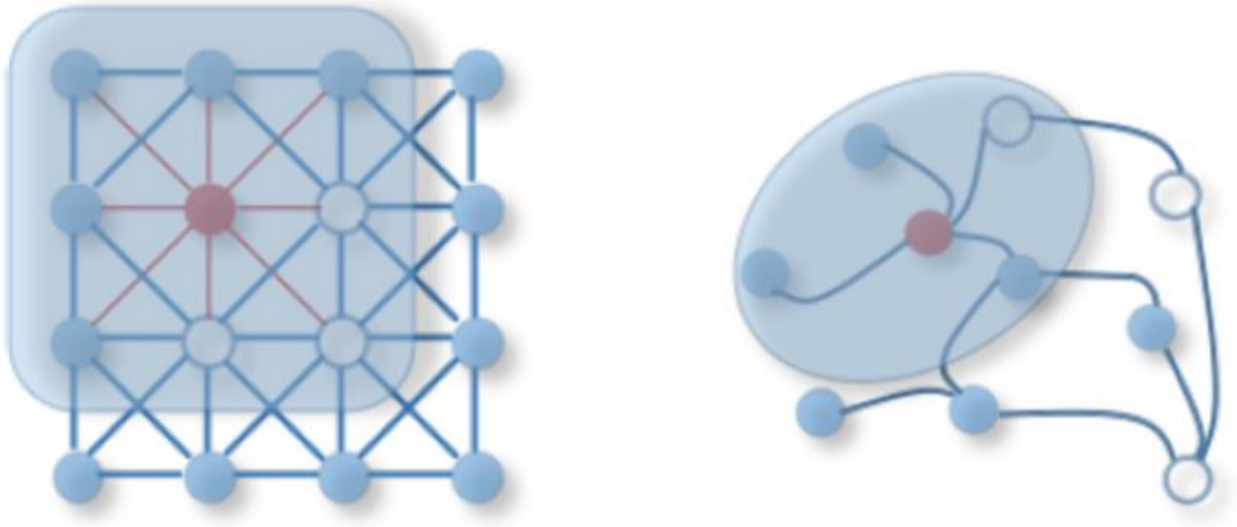


Graph

Transform information at the neighbors and combine it

- Transform “messages” h_i from neighbors: $W_i\ h_i$
- Add them up: $\sum_i W_i\ h_i$

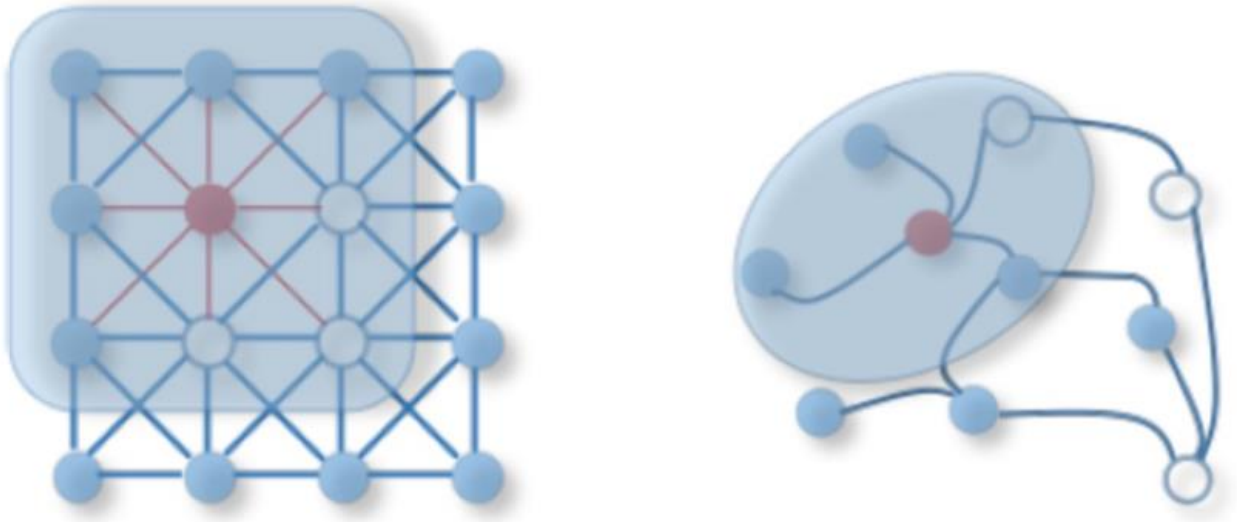
Domain: Euclidean → non-Euclidean



A regular Convolutional Neural Network used popularly for Image Recognition, captures **the surrounding information of each pixel of an image.**

The convolution framework aims to **capture neighbourhood information for non-euclidean spaces** like graph nodes.

GCNs



Spectral GCNs: Spectral-based approaches define graph convolutions by introducing **filters** from the perspective of graph signal processing based on graph spectral theory.
Spatial GCNs: Spatial-based approaches formulate graph convolutions as **aggregating** feature information from neighbours.

Feature learning

- CNN：As the number of network layers increases, features become more and more abstract and are then used in the final task.

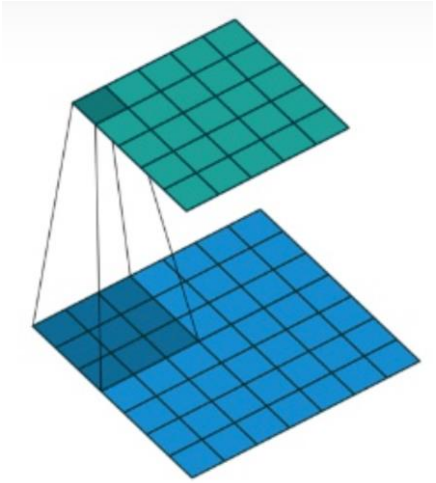
- GCN：Learning from the most initial feature of the graph X to a more abstract feature, incorporates the features of other nodes in the graph according to the graph structure.

$$H^{(k+1)} = f(H^{(k)}, A)$$

where

$$H^{(0)} = X \in R^{N \times F}, H^{(k)} \in R^{N \times F^{(k)}}$$

Feature learning

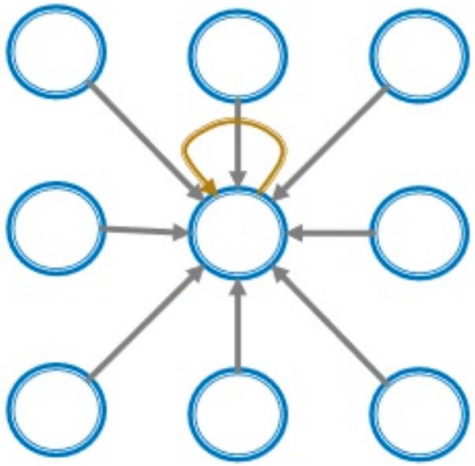


Step1:

$$w_i x_i$$

Step2:

$$\sum_i w_i x_i$$



$$H^{(k+1)} = f\left(H^{(k)}, A\right) = \sigma(AH^{(k)}W^{(k)})$$

where

$$H^{(0)} = X$$

$$H^{(1)} = f\left(H^{(0)}, A\right) = \sigma\left(AH^{(0)}W^{(0)}\right) = \sigma\left(AXW^{(0)}\right)$$

Feature learning

$$H^{(k+1)} = f\left(H^{(k)}, A\right) = \sigma\left(AH^{(k)}W^{(k)}\right)$$

- The aggregated representation of a node is only a function of its neighbours and does not include its own features.

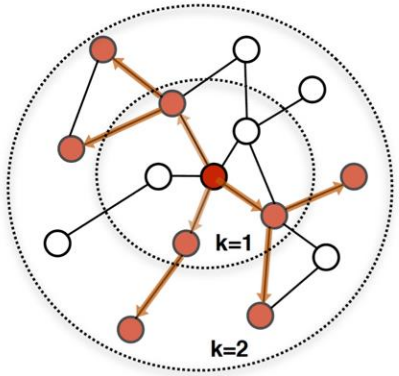
$$\tilde{A} = A + I_N$$

- Feature is influenced by degree.

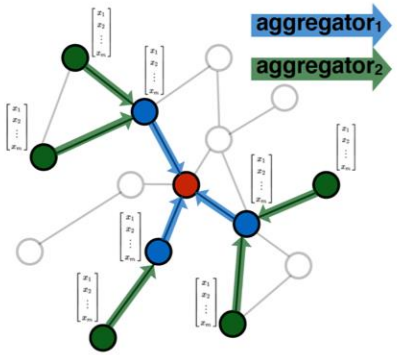
$$\tilde{D} = \sum_j A_{ij}$$

$$H^{(k+1)} = f\left(H^{(k)}, A\right) = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(k)}W^{(k)})$$

From a general perspective



Determine node
computation graph



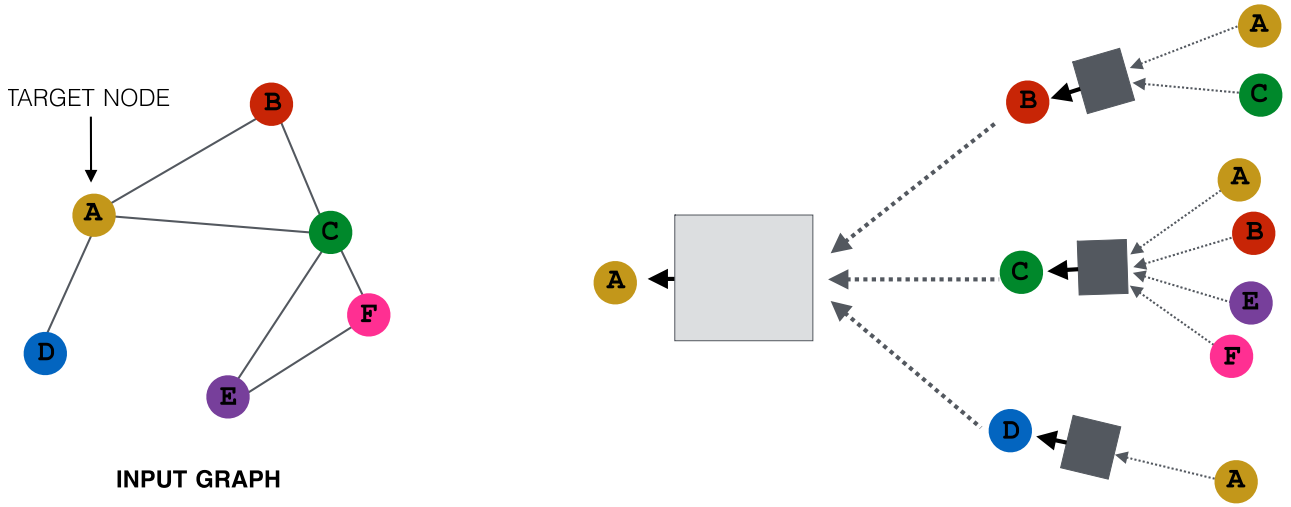
Propagate and
transform information

Aggregate
Neighbors

Learn how to propagate information across the graph to compute node features

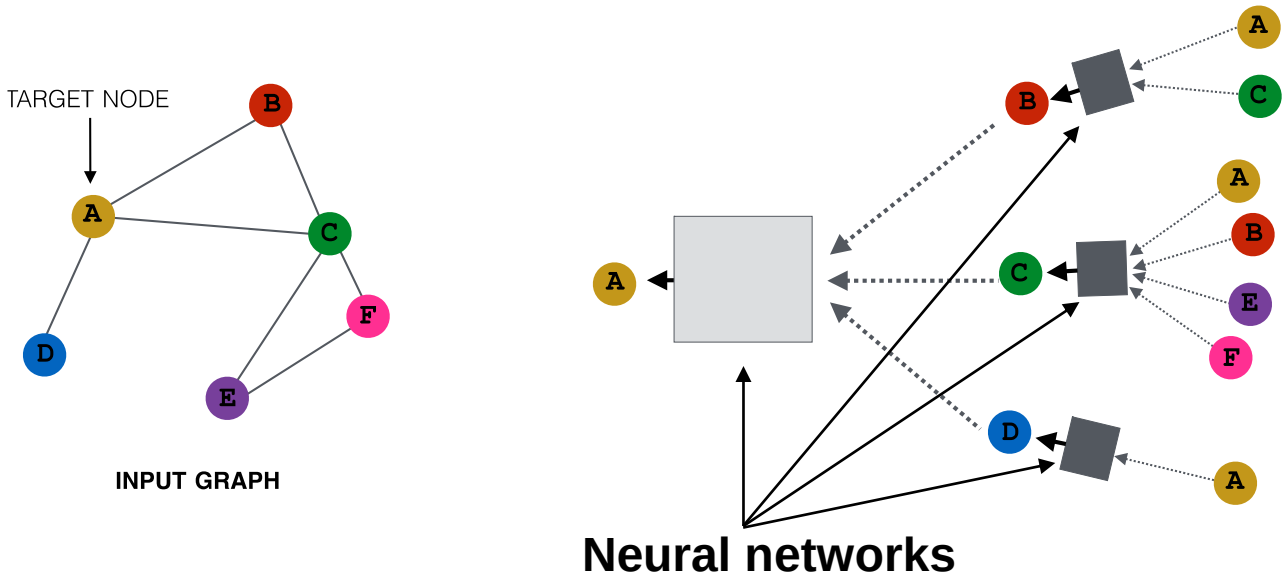
Aggregate Neighbors

Key idea: Generate node embeddings based on **local network neighborhoods**



Aggregate Neighbors

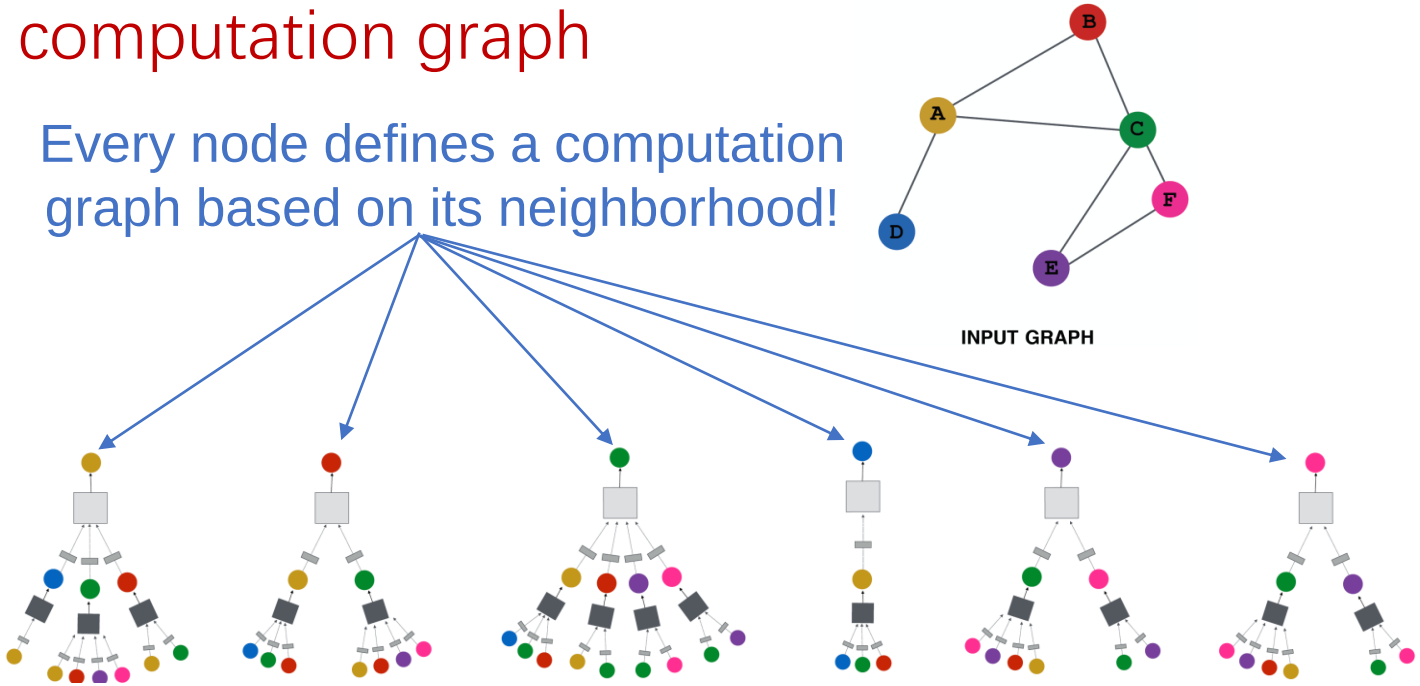
Intuition: Nodes aggregate information from their neighbors using neural networks



Deep Model: Many Layers

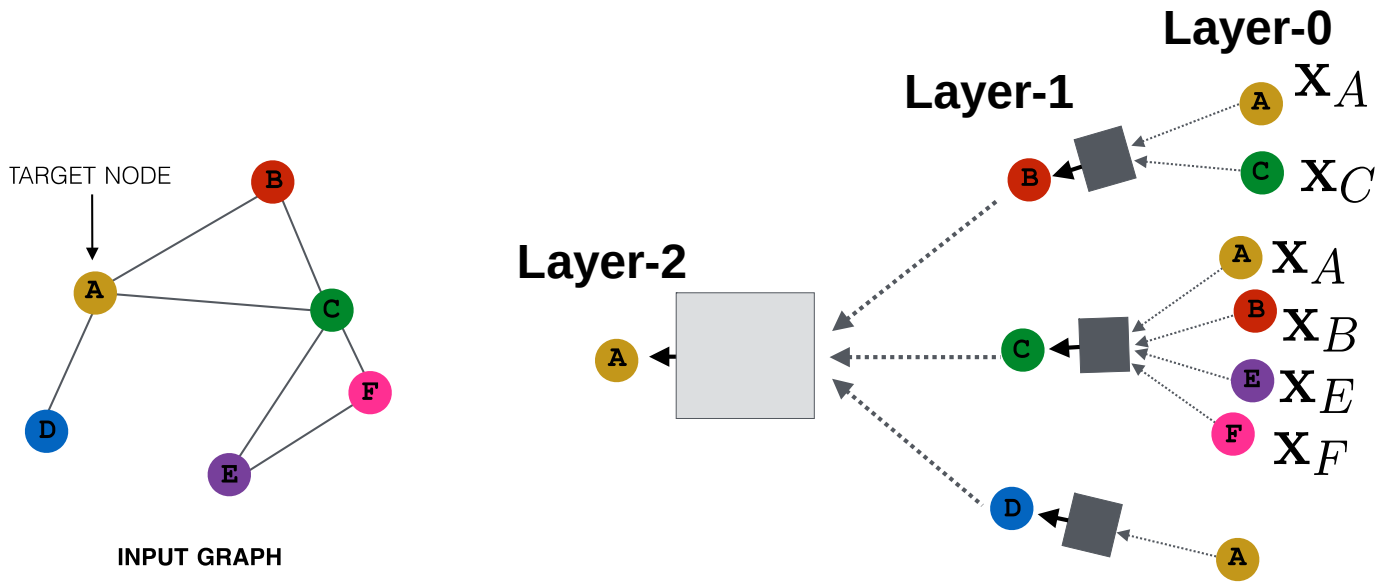
Intuition: Network neighborhood defines a computation graph

Every node defines a computation graph based on its neighborhood!



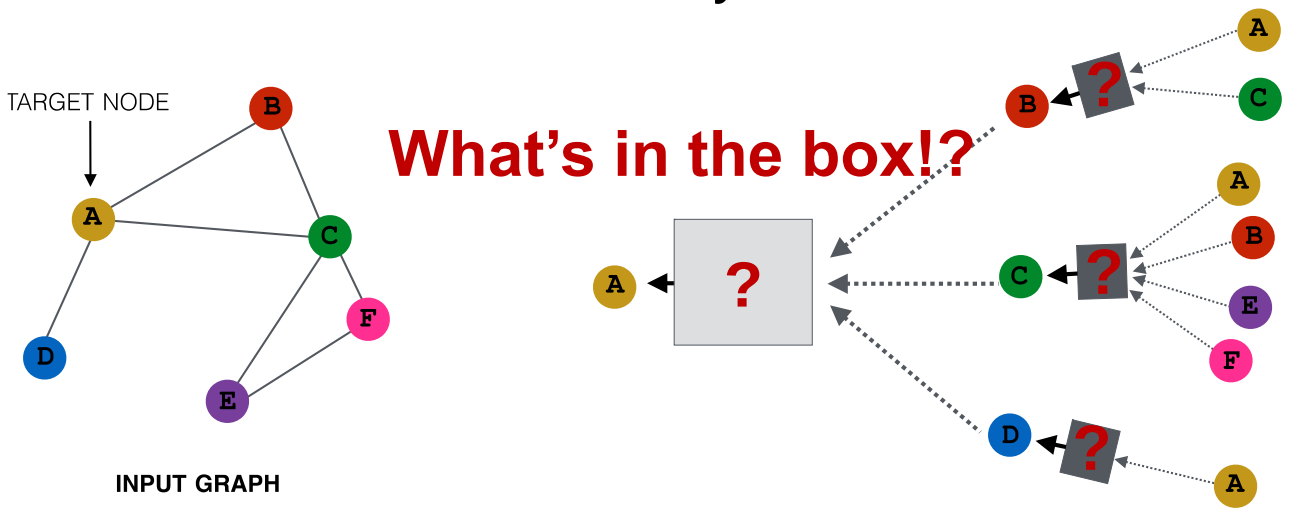
Deep Model: Many Layers

- Model can be of arbitrary depth:
  - Nodes have embeddings at each layer
  - Layer-0 embedding of node $\mathbf{u}$ is its input feature, i.e. $\mathbf{x}_u$.



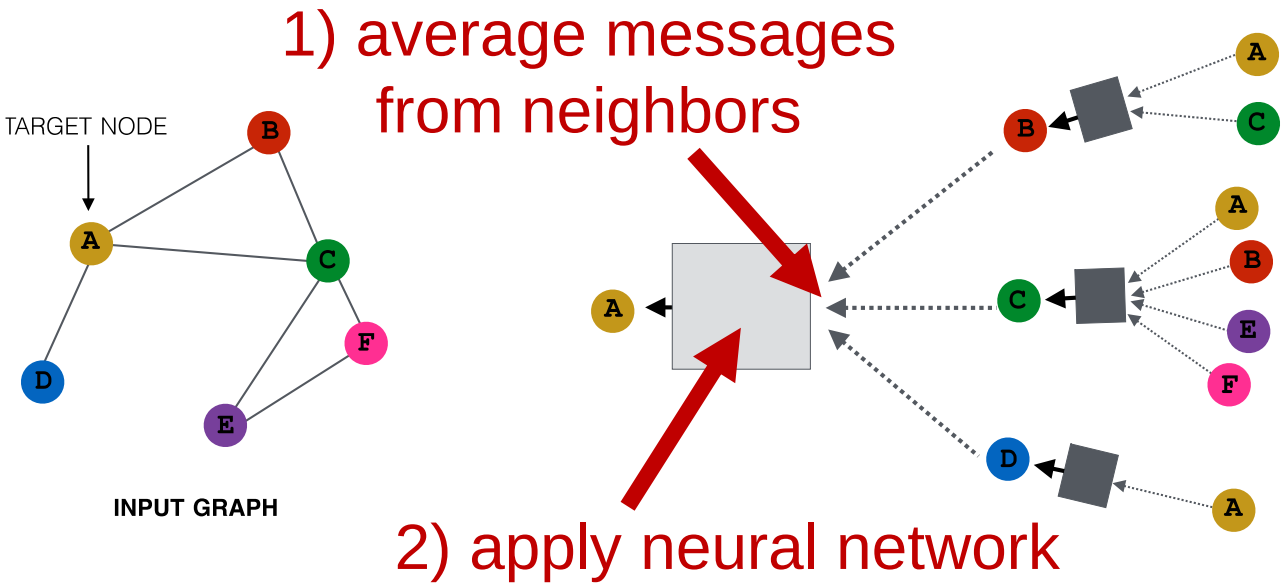
Aggregation Strategies

- Neighborhood aggregation: Key distinctions are in how different approaches aggregate information across the layers



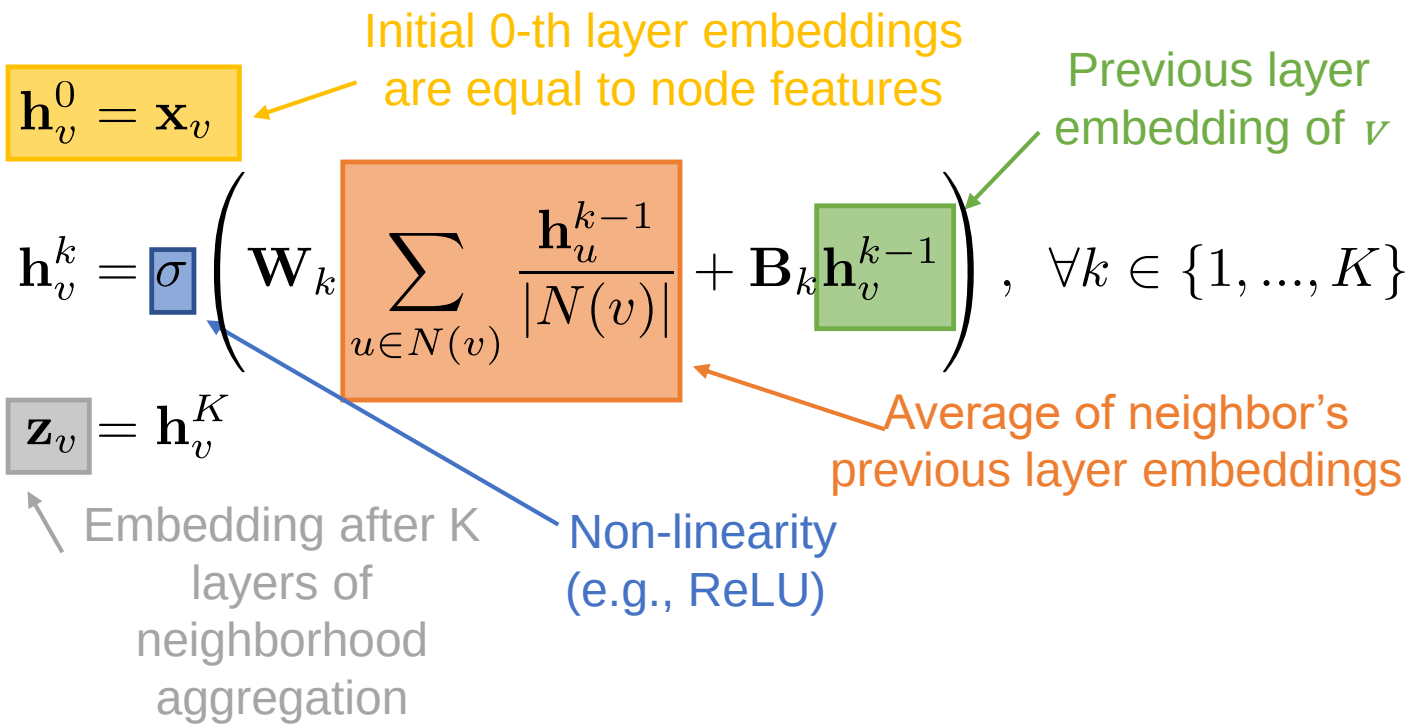
Average neighbor messages

- **Basic approach:** Average information from neighbors and apply a neural network



Average neighbor messages

- **Basic approach:** Average neighbor messages and apply a neural network



Training the Model

trainable weight matrices
(i.e., what we learn)

$$\mathbf{h}_v^0 = \mathbf{x}_v$$

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right), \quad \forall k \in \{1, ..., K\}$$

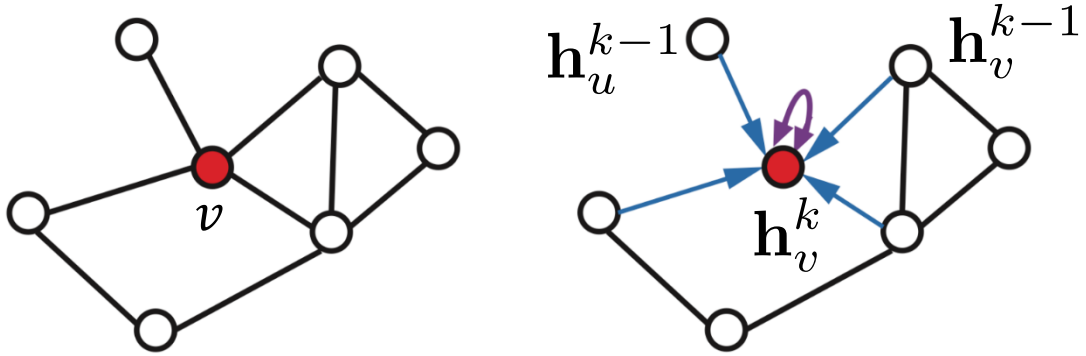
$$\mathbf{z}_v = \mathbf{h}_v^K$$

- Train in an unsupervised manner:
  - Use only the graph structure
  - **"Similar" nodes have similar embeddings**
- Unsupervised loss function can be anything from the last section, e.g., a loss based on
  - Random walks (node2vec, DeepWalk, struc2vec)
  - Graph factorization
  - Node proximity in the graph

GraphSAGE

Key idea: Generate node embeddings based on local neighborhoods

- Nodes aggregate "messages" from their neighbors using neural networks



Concatenate self embedding and neighbor embedding

$$\mathbf{h}_v^k = \sigma\left(\left[\mathbf{W}_k \cdot \text{AGG}\left(\left\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\right\}\right), \mathbf{B}_k \mathbf{h}_v^{k-1}\right]\right)$$

generalized aggregation

GraphSAGE

- Simple neighborhood aggregation:

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right)$$

- GraphSAGE:

Concatenate self embedding and neighbor embedding

$$\mathbf{h}_v^k = \sigma\left(\left[\mathbf{W}_k \cdot \mathrm{AGG}\left(\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right), \mathbf{B}_k \mathbf{h}_v^{k-1}\right]\right)$$

generalized aggregation

GraphSAGE

- **Mean:** Take a weighted average of neighbors

$$\mathrm{AGG} = \sum_{u \in N(v)} \frac{\mathrm{h}_u^{(l)}}{|N(v)|}$$

- **Pool:** Transform neighbor vectors and apply symmetric vector function

Element-wise mean/max

$$\mathrm{AGG} = \gamma(\{\mathrm{MLP}(\mathrm{h}_u^{(l)}), \forall u \in N(v)\})$$

- **LSTM:** Apply LSTM to reshuffled of neighbors

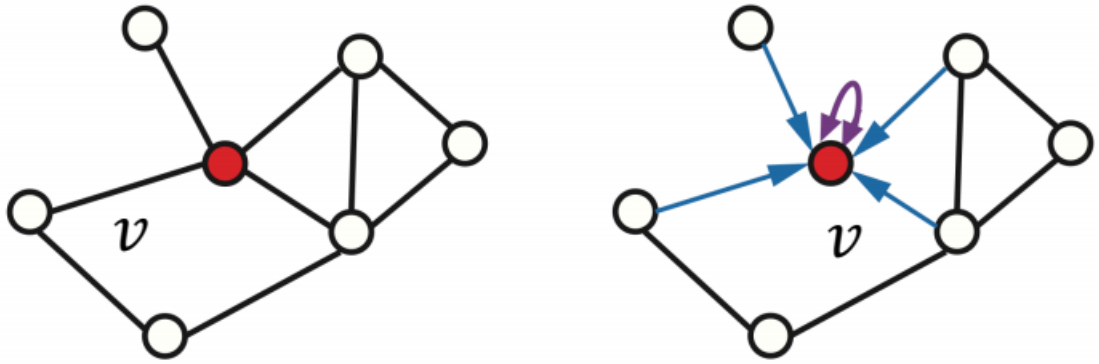
$$\mathrm{AGG} = \mathrm{LSTM}([\mathrm{h}_u^{(l)}, \forall u \in \pi(N(v))])$$

GCN and GraphSAGE

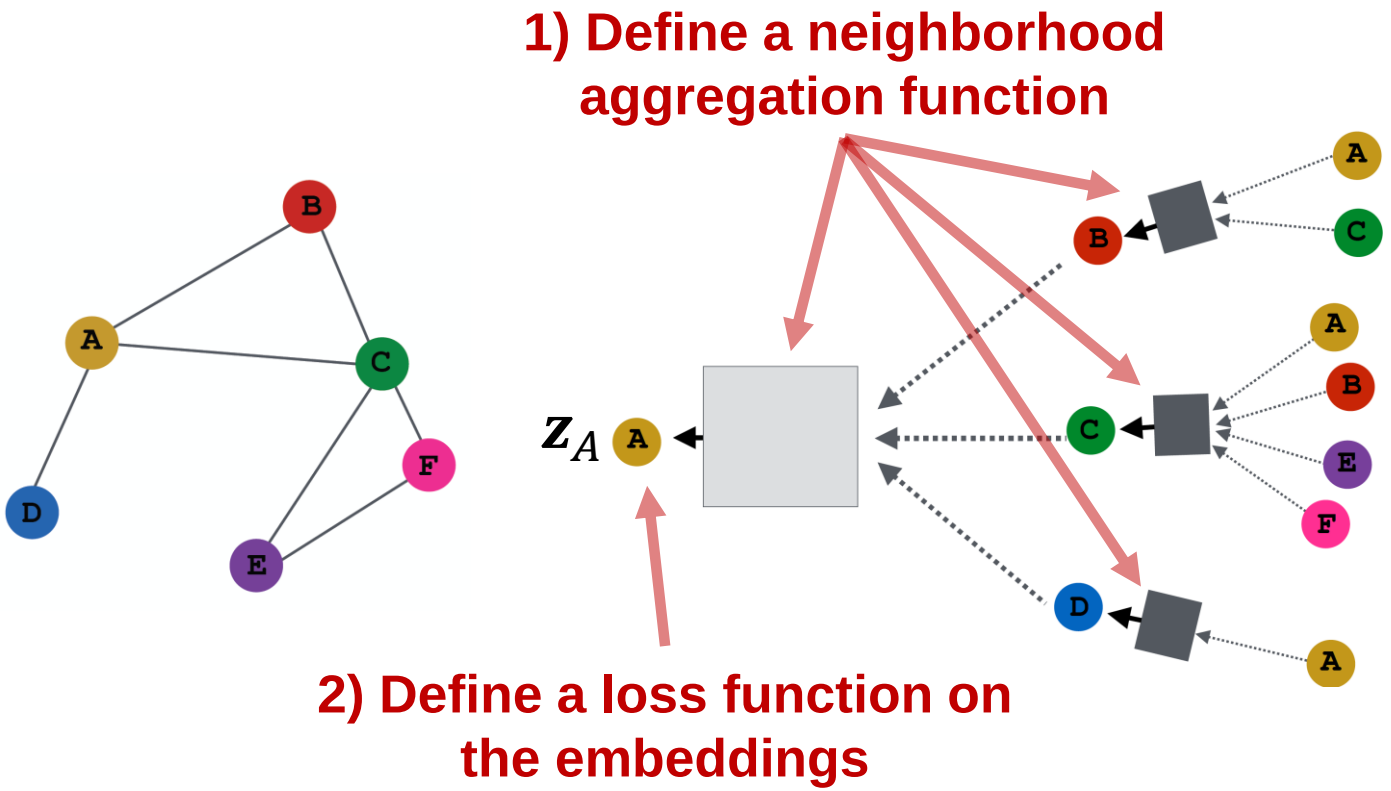
Key idea: Generate node embeddings based on local neighborhoods

- Nodes aggregate "messages" from their neighbors using neural networks

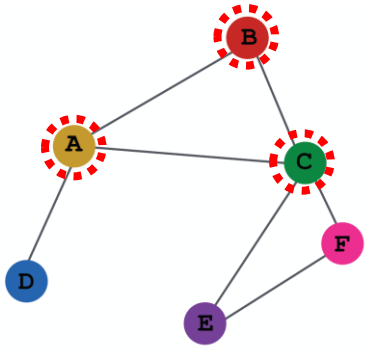
- **Graph convolutional networks:**
  - **Basic variant:** Average neighborhood information and stack neural networks
- **GraphSAGE:**
  - Generalized neighborhood aggregation



Model Design: Overview

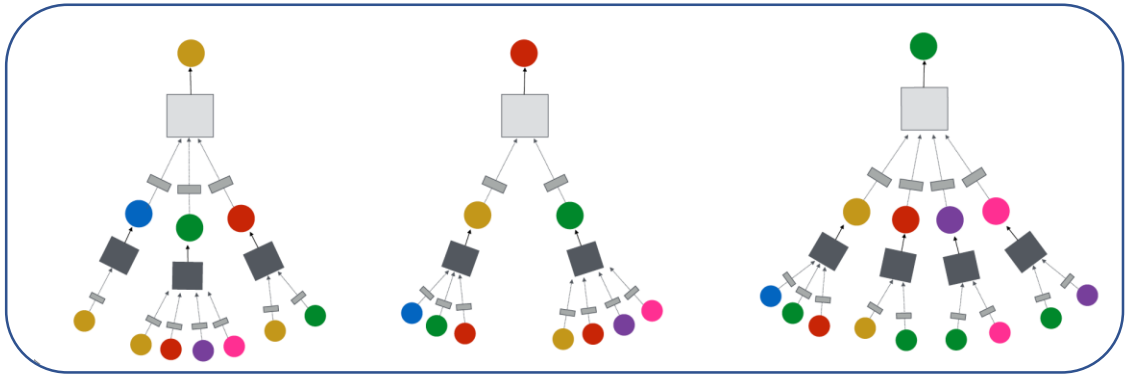


Model Design: Overview

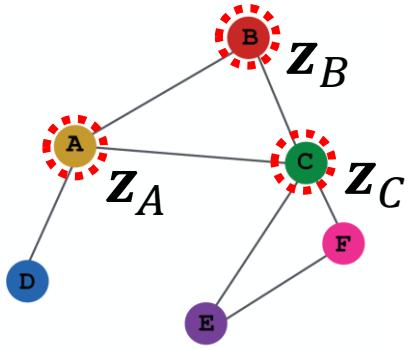


INPUT GRAPH

3) Train on a set of nodes, i.e., a batch of compute graphs

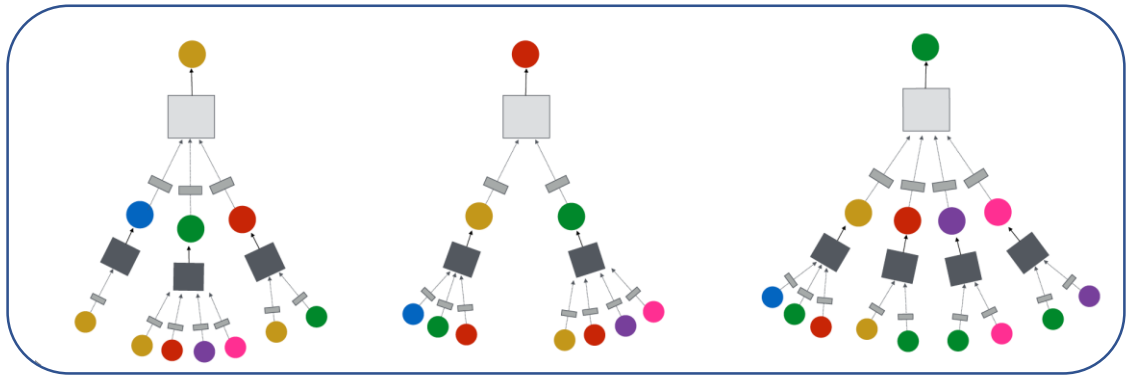


Model Design: Overview



4) Generate embeddings for nodes

INPUT GRAPH

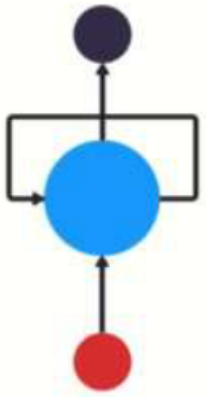


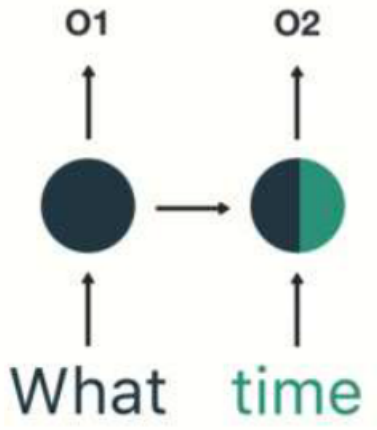
附录：NLP相关模块

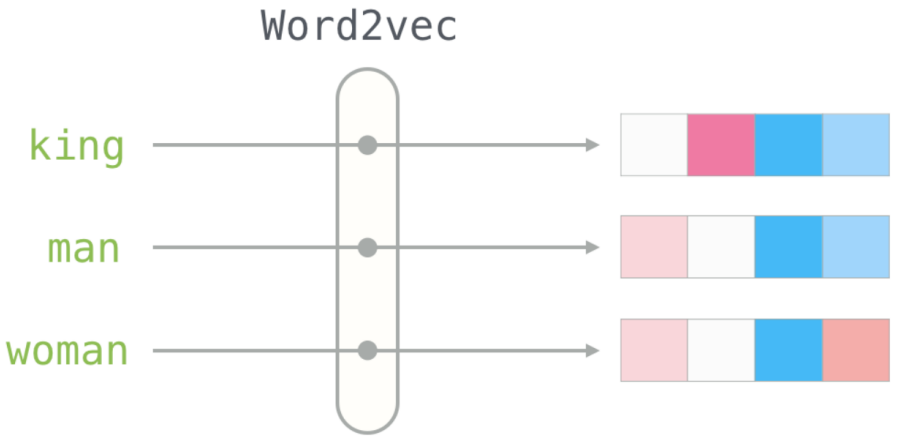
循环神经网络

- 循环神经网络(Recurrent Neural Network，RNN)是一类用于处理序列数据的神经网络，它广泛的用于自然语言处理中的语音识别、机器翻译、故障诊断等领域。

- RNN可以将之前的输入“记忆”在神经网络中，从而输出也与之前输入有关，如下图的例子：

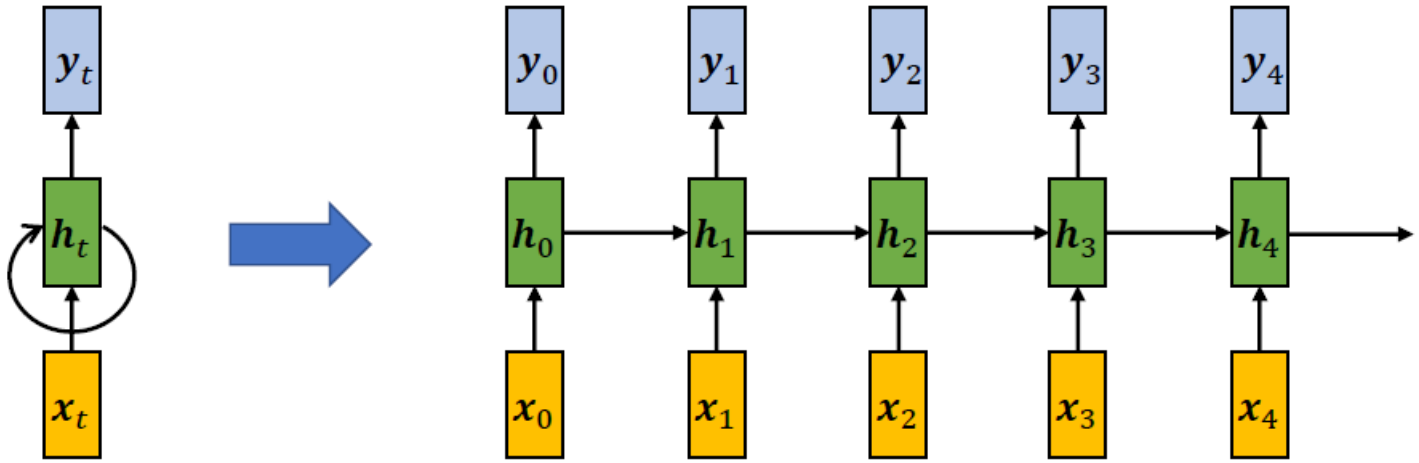






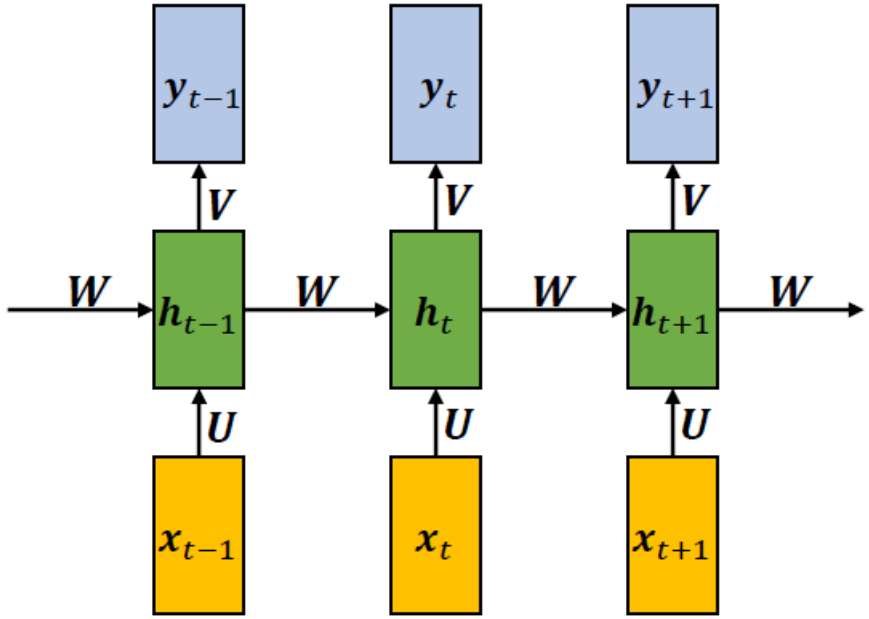
循环神经网络

- RNN是包含有循环的网络，可以使用展开计算图的方式表示。
  - $\boldsymbol{x}_t$：输入，$\boldsymbol{h}_t$：隐藏层，$\boldsymbol{y}_t$：输出



循环神经网络

- RNN每一层的计算需要考虑到上一层的影响，假设使用双曲正切激活函数(tanh)的前提下，用RNN来预测词或者字符，其推导过程如下所示。



$$\boldsymbol{h}_t = \tanh(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t)$$

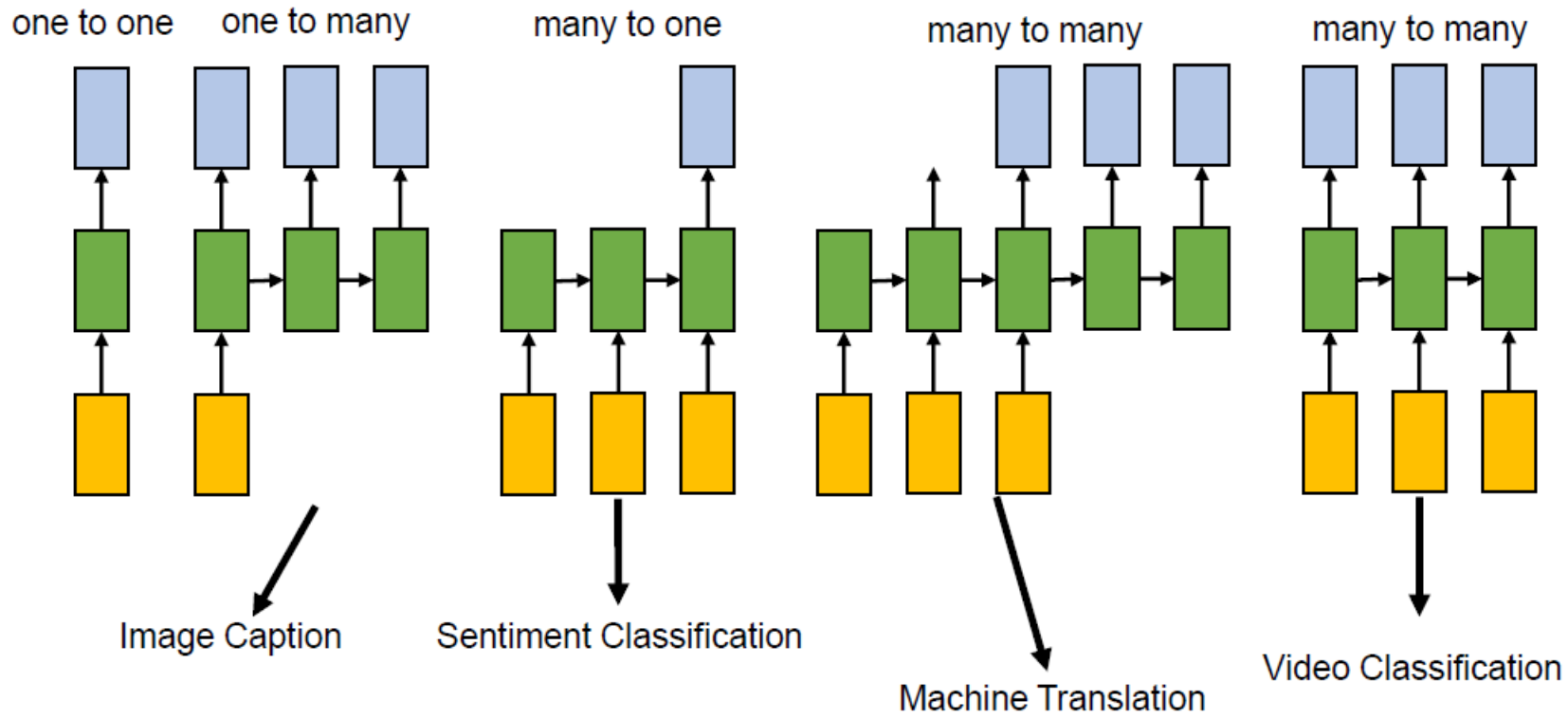
$$\boldsymbol{y}_t = \mathrm{softmax}(\boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}_t)$$

$\boldsymbol{b}, \boldsymbol{c}$为偏置向量

其它RNN模型：LSTM，GRU等。

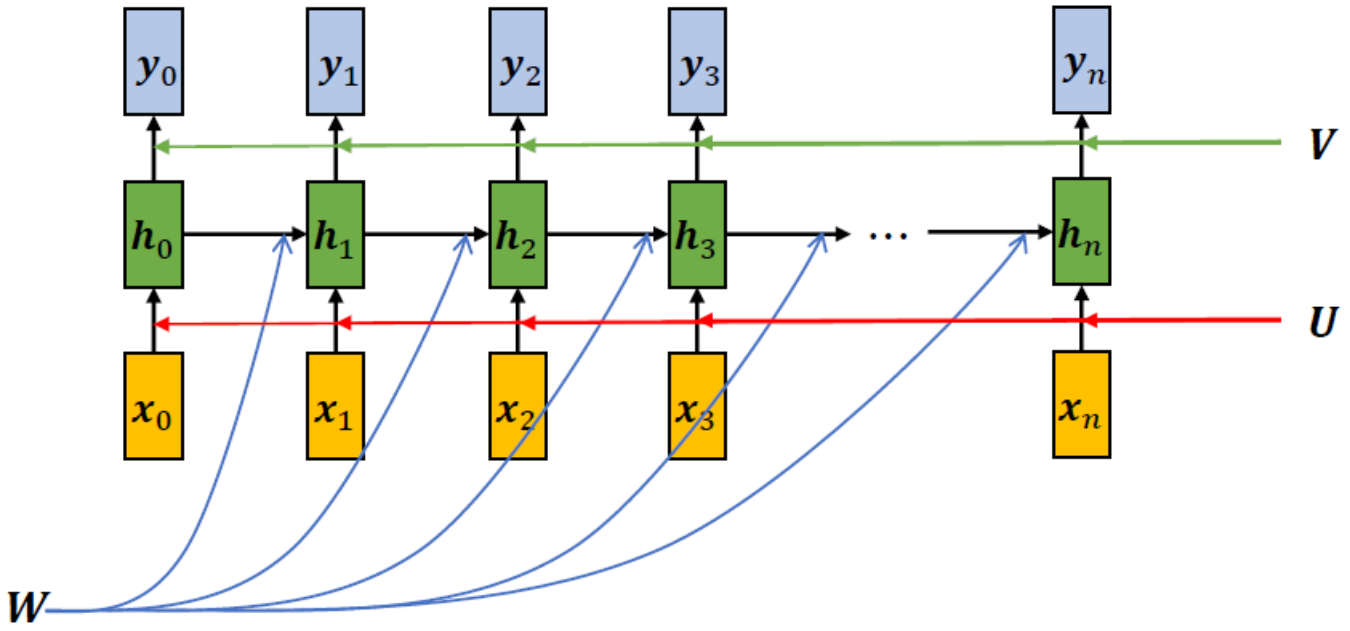
循环神经网络

- RNN按照输入输出可以分为以下四种。



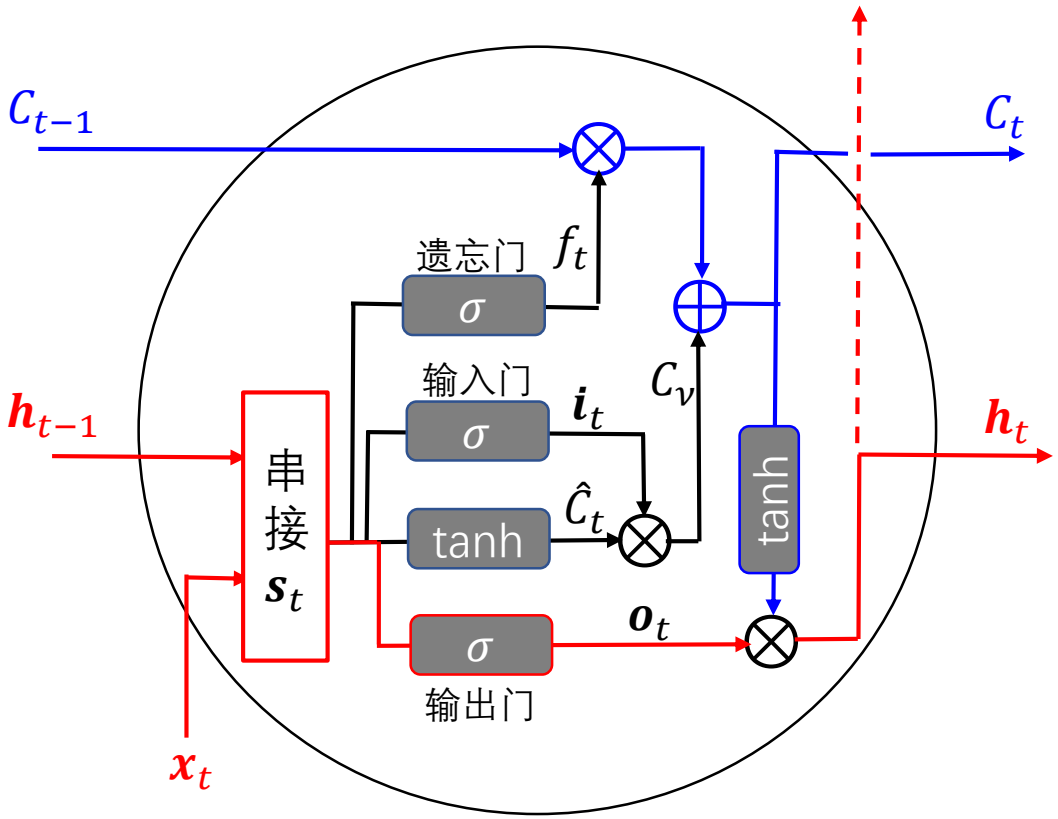
循环神经网络：参数共享

- RNN的一个特点是所有的隐层共享参数($\boldsymbol{W}$，$\boldsymbol{U}$，$\boldsymbol{V}$)，这样极大地缩小了参数空间。



Long short-term memory (LSTM)

更好地处理长程记忆

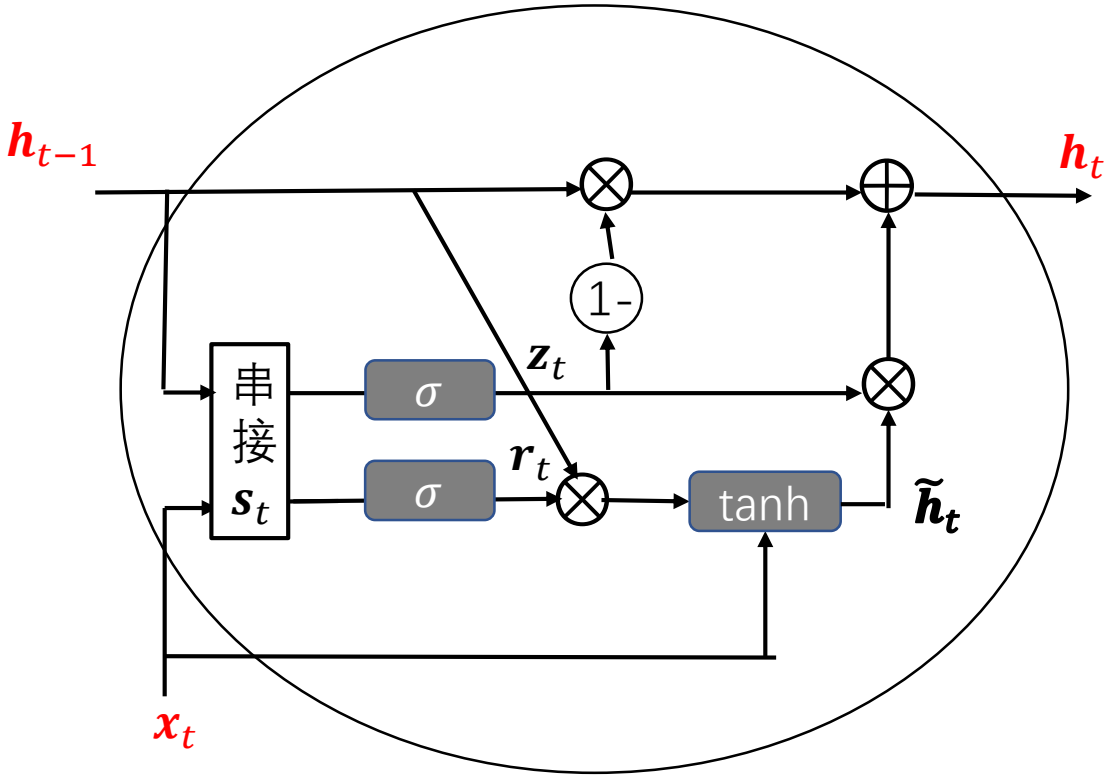


⊗ 表示按位相乘

⊕ 表示按位相加

Gated Recurrent Unit (GRU)

LSTM的简化变体



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Junyoung Chung, et al., 2014, https://arxiv.org/pdf/1412.3555.pdf