Introduction to GraphNN

Yongbin Feng (Fermilab)

AI Lab-Wide Meetings

November 18th, 2022

Disclaimer

- I'm NOT a GraphNN expert, so I hope the messages I delivered are mostly correct, but it could be wrong/misunderstanding…

- Most of them maybe sound straightforward; but in practice, training these neural networks could be much more tricky; need some experiences/tunings/magic…

Disclaimer

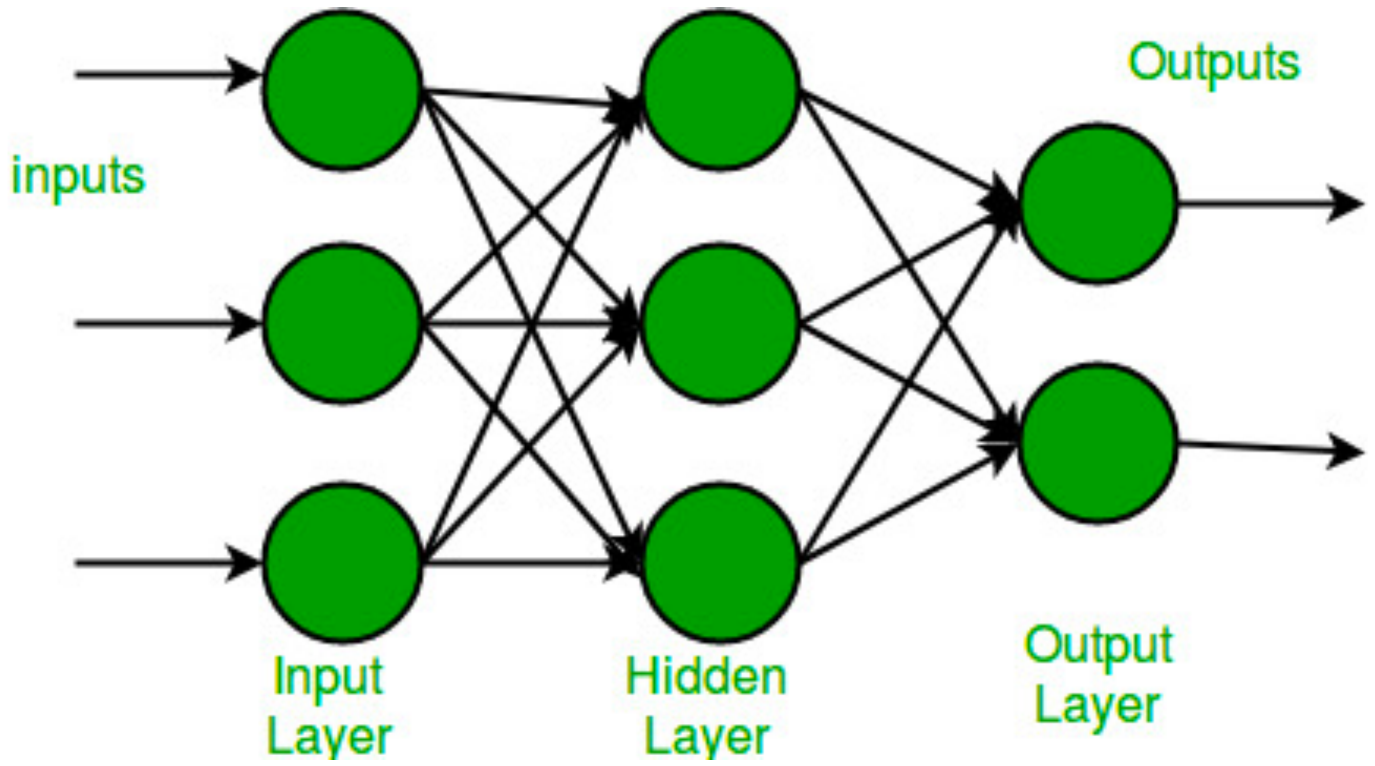
- I'm NOT a GraphNN expert, so I hope the messages I delivered are mostly correct, but it could be wrong/misunderstanding…
- Most of them maybe sound straightforward; but in practice, training these neural networks could be much more tricky; need some experiences/tunings/magic…

Overview

(Fully-Connected) Deep-Neural-Network
(Multi Layer Perceptron)

Convolutional Neural Network

Graph Neural Network

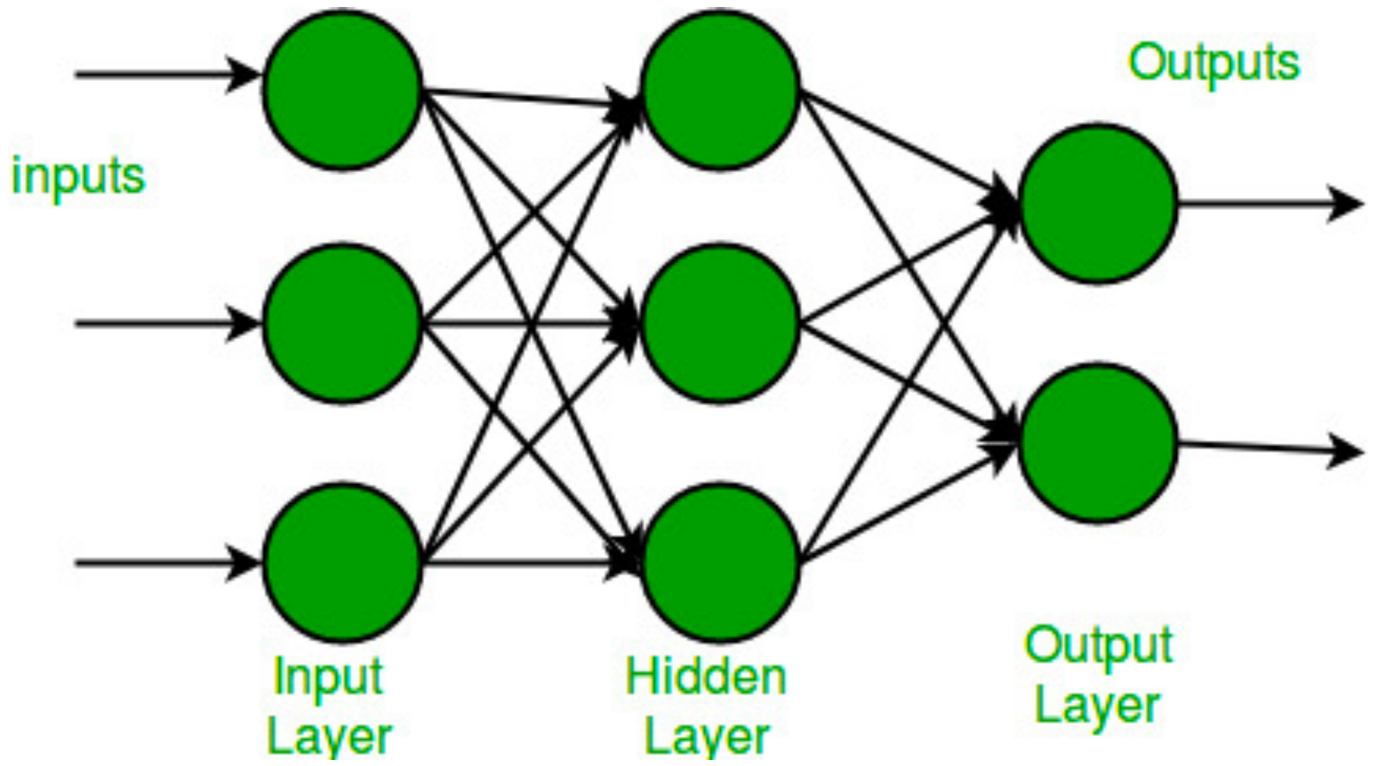


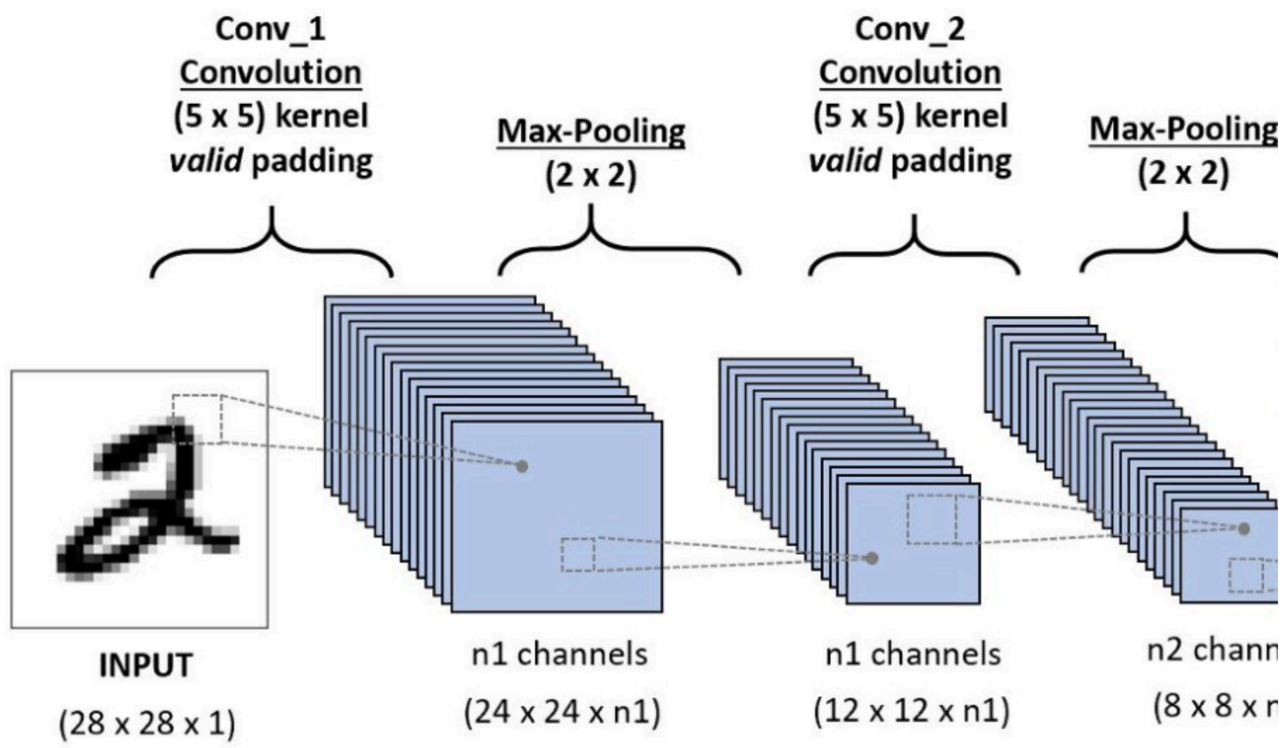
Overview

(Fully-Connected) Deep-Neural-Network (Multi Layer Perceptron)

Convolutional Neural Network

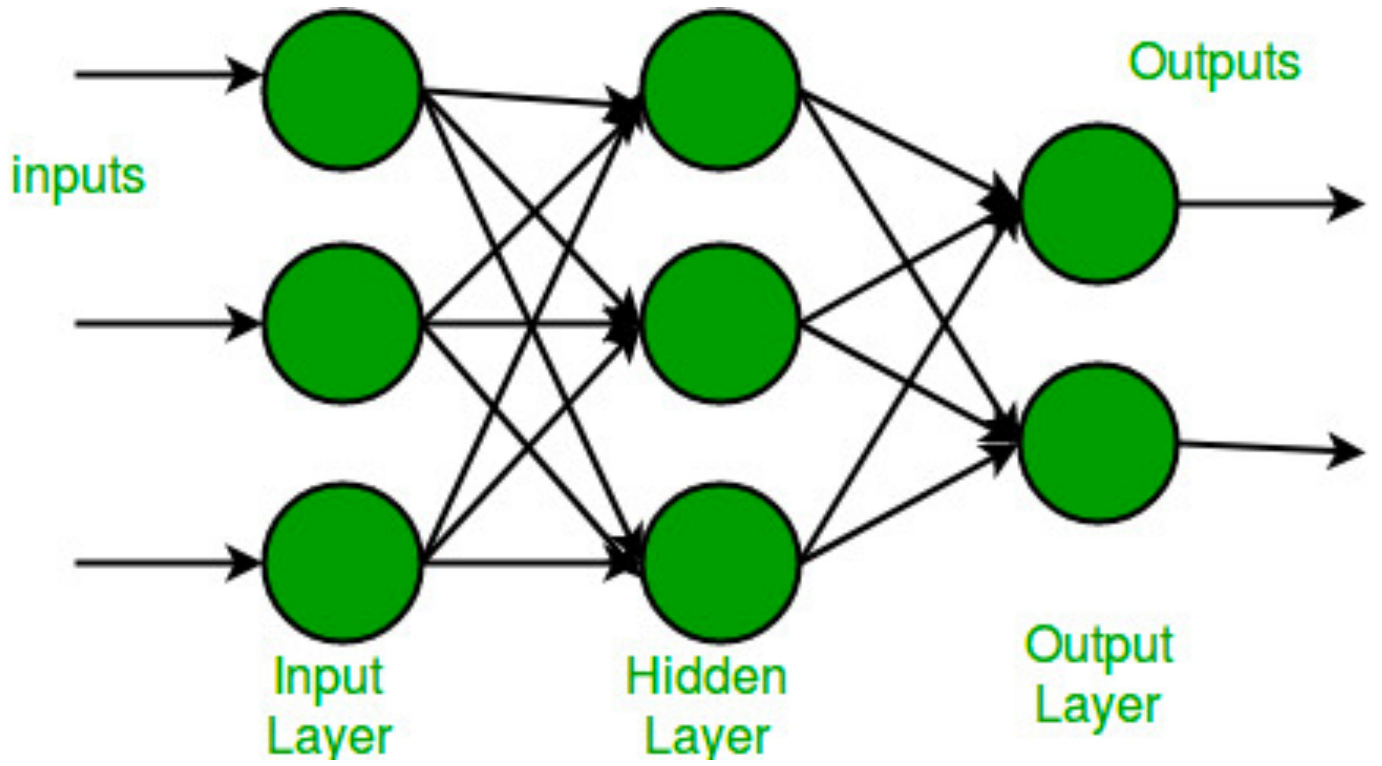
Graph Neural Network

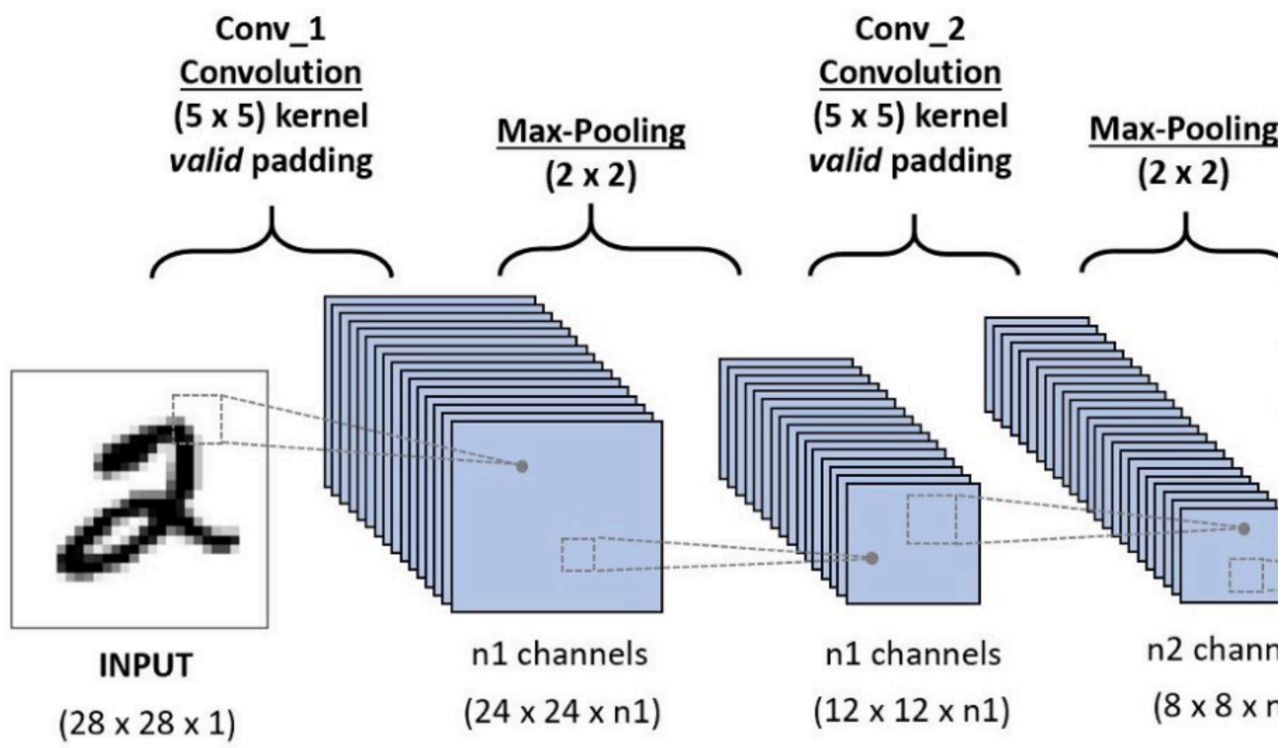


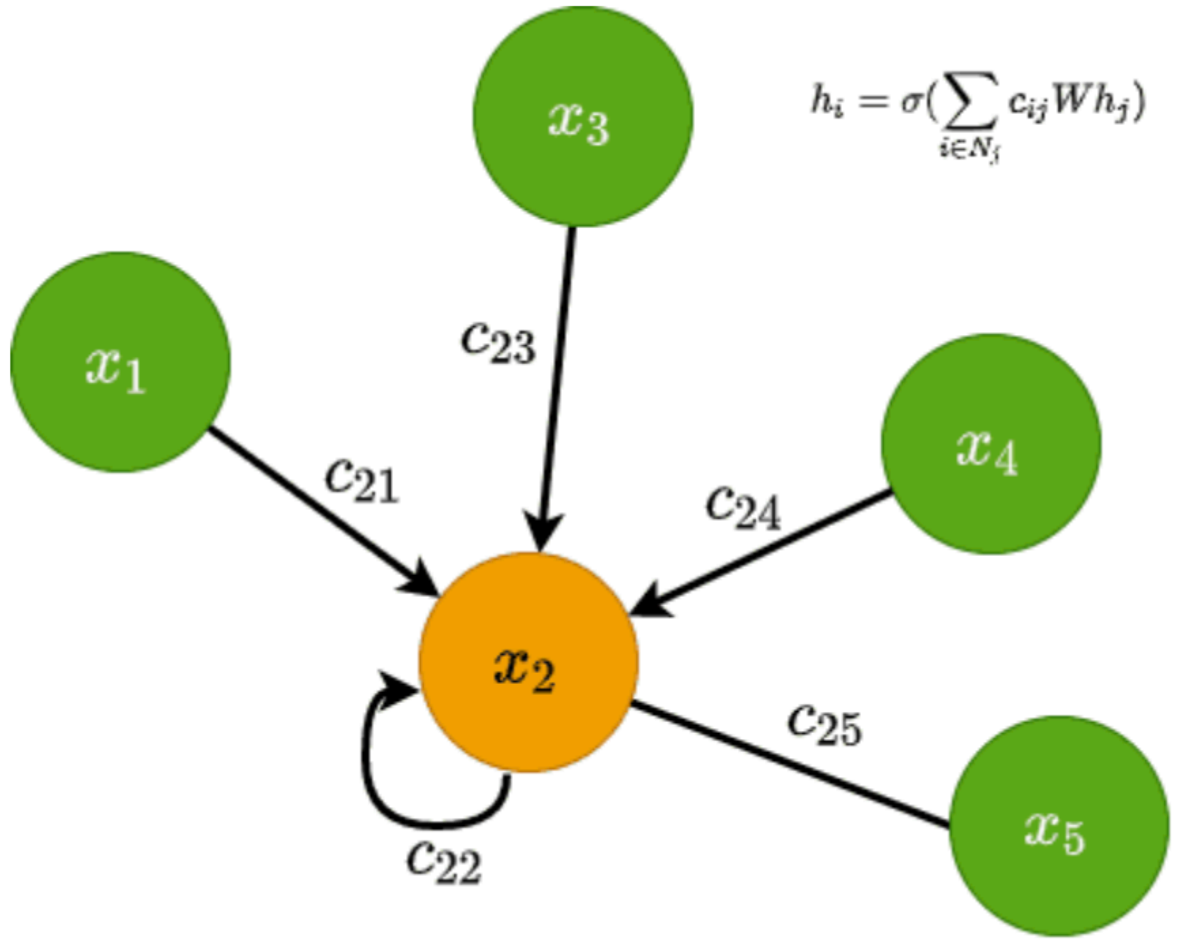


Overview

(Fully-Connected) Deep-Neural-Network (Multi Layer Perceptron) → Convolutional Neural Network → Graph Neural Network

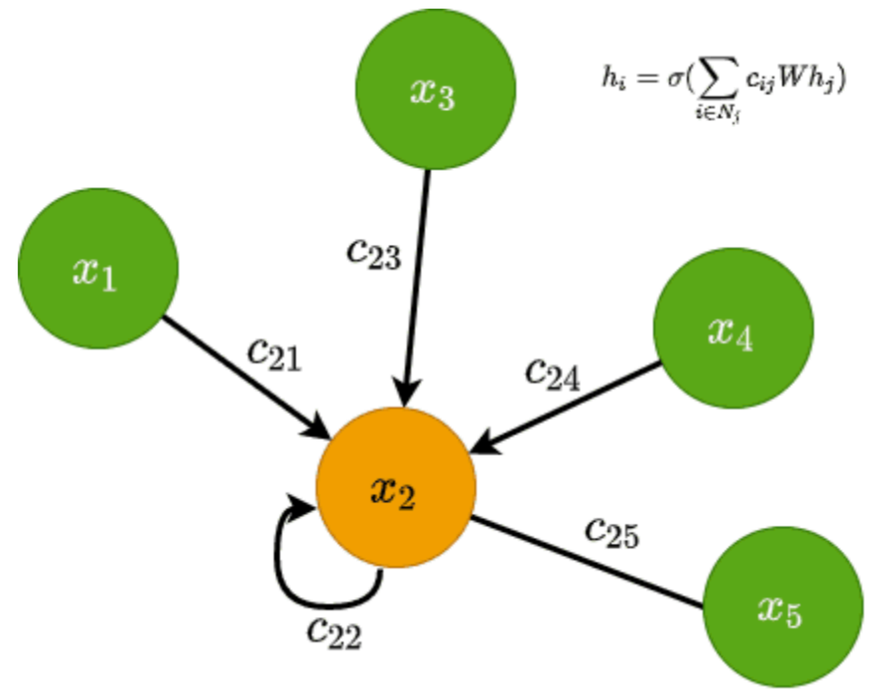






Overview

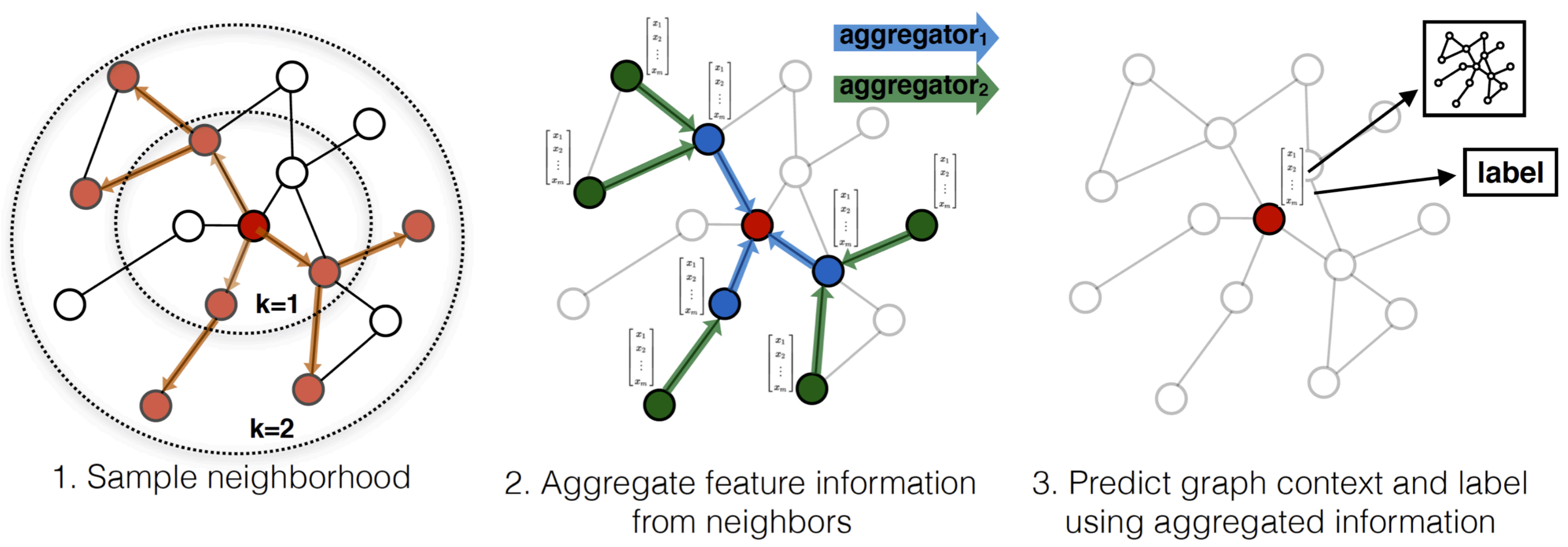
Graph Neural Network
(Message Passing NN)

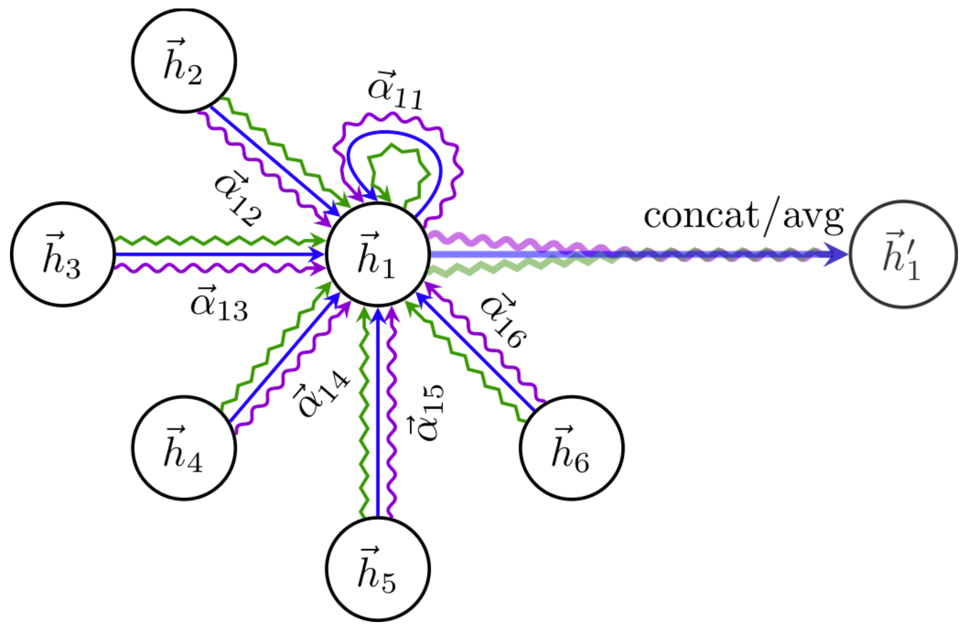


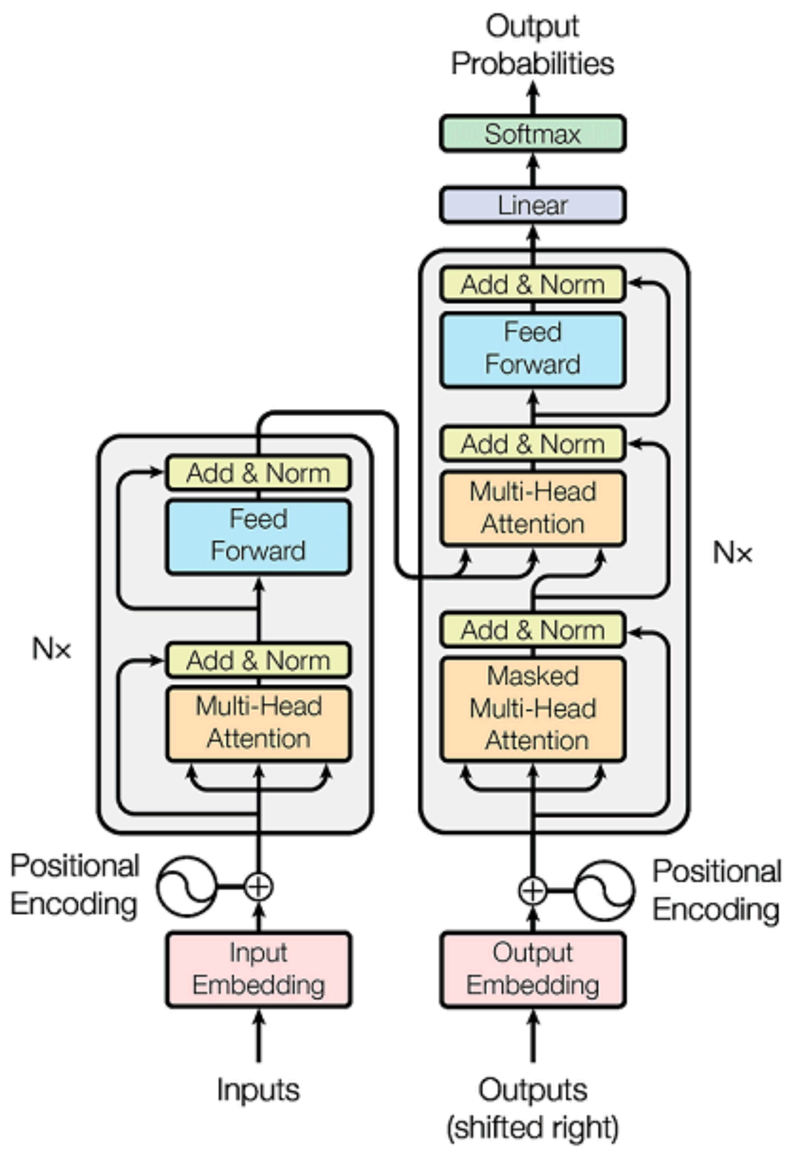
GraphSage

Graph Attention Network

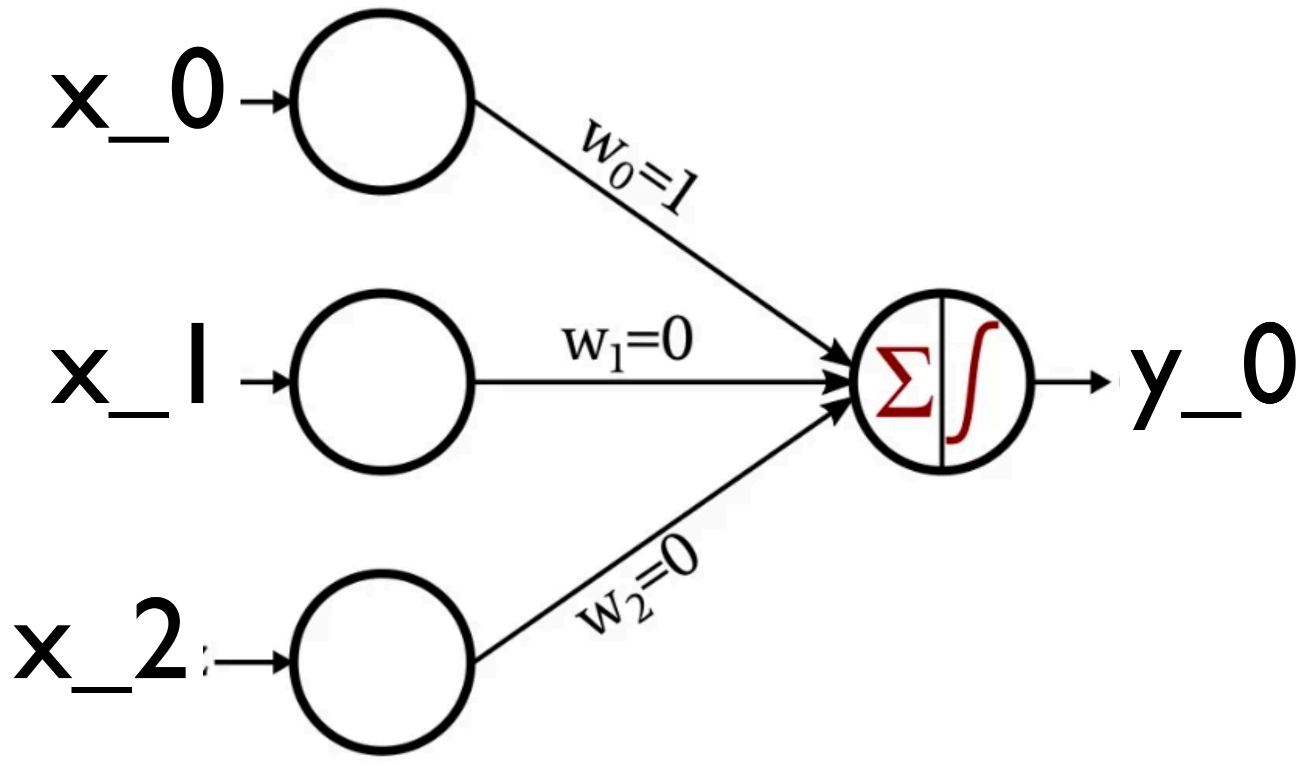
(Graph) Transformers

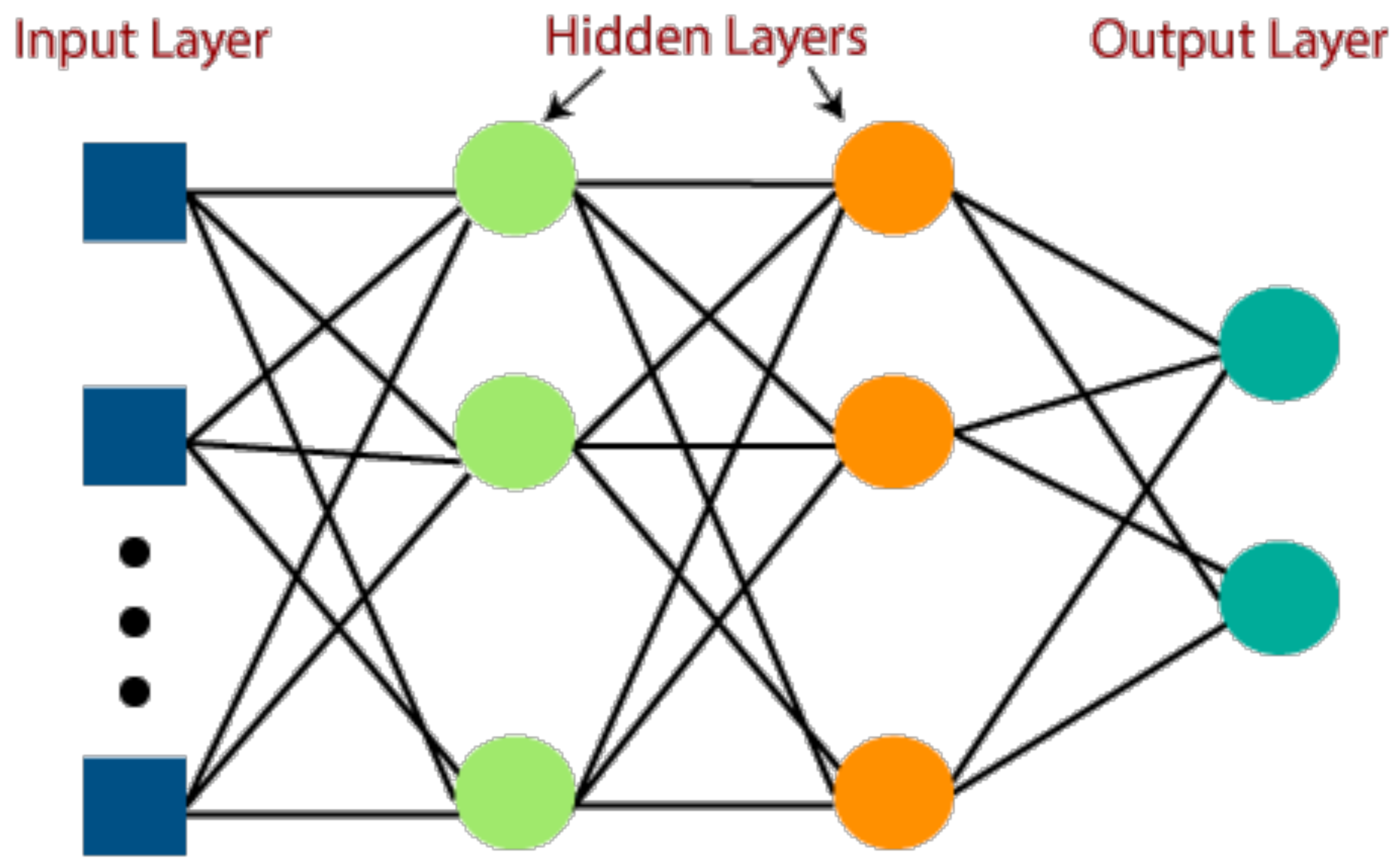






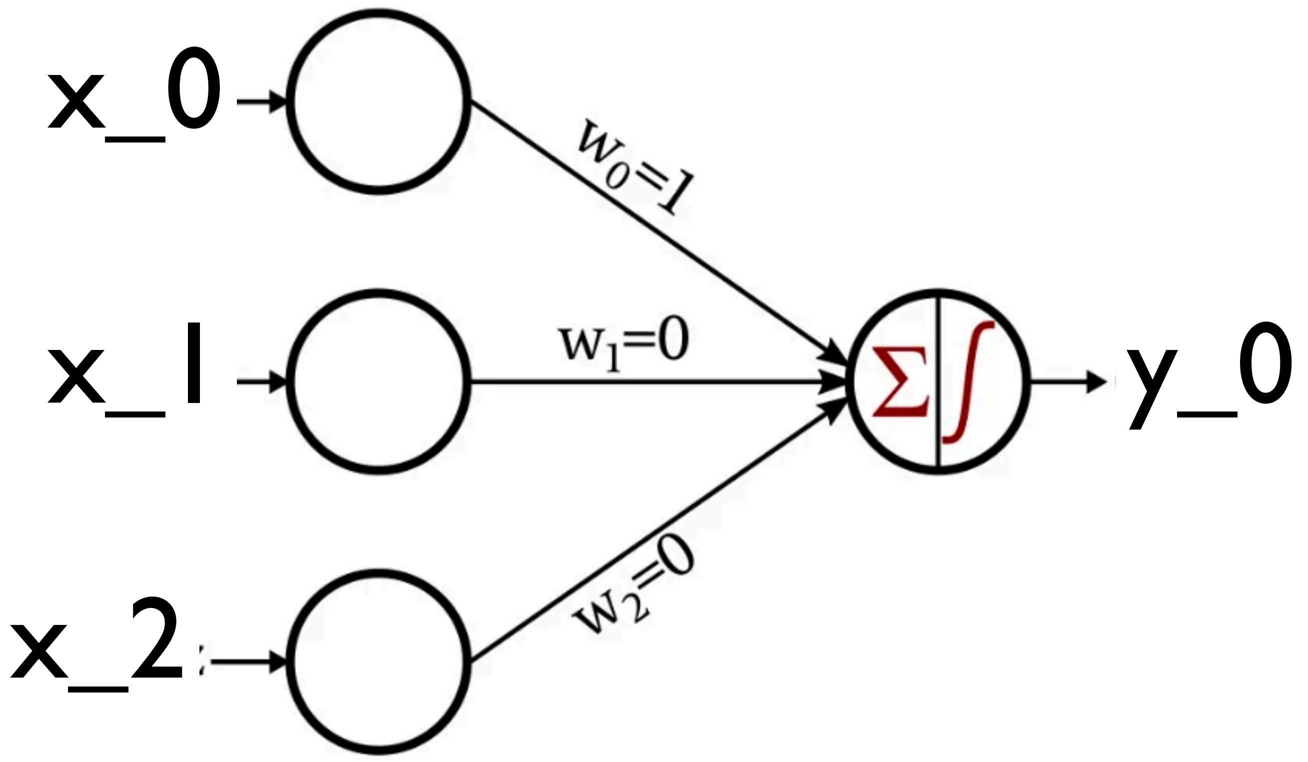
Multi Layer Perceptron

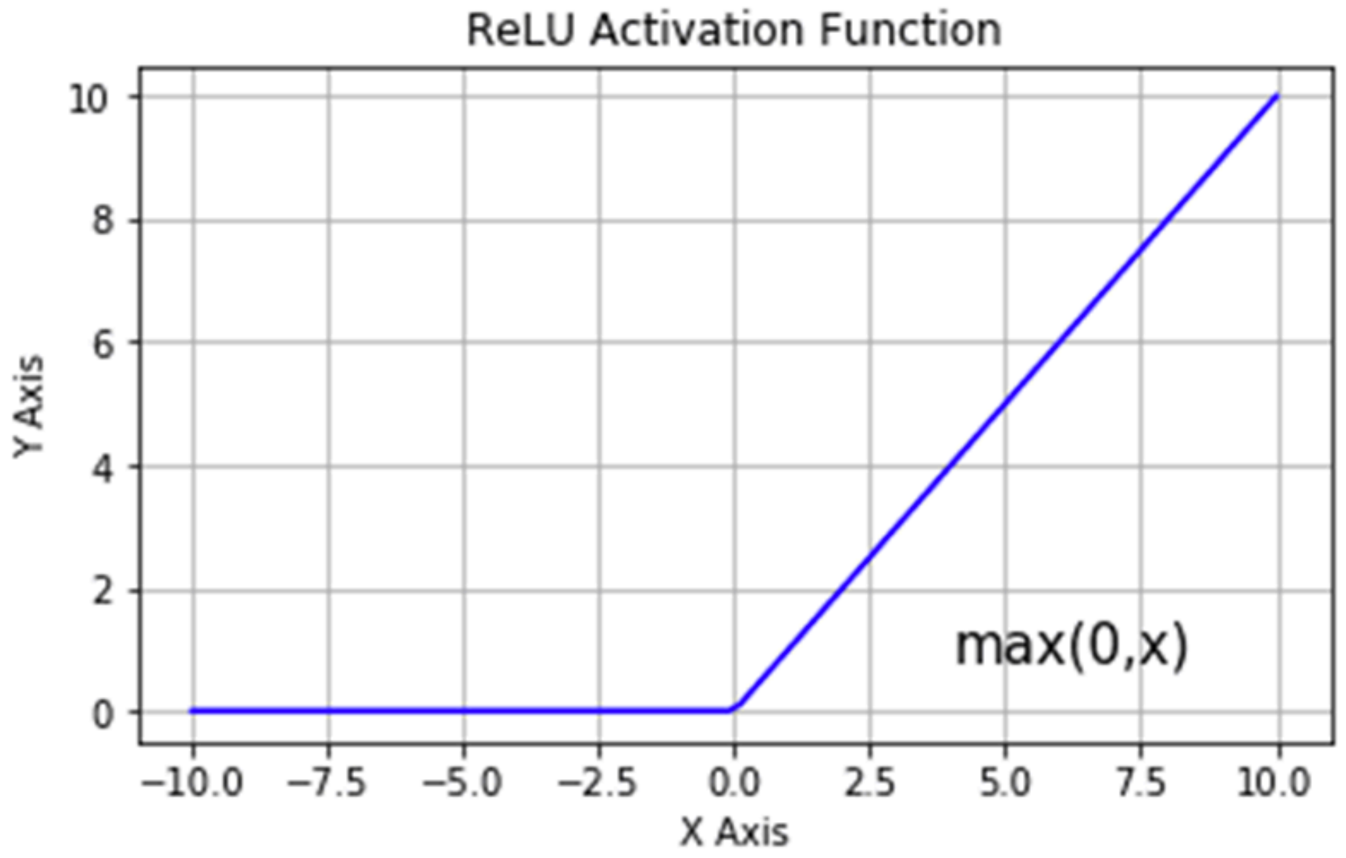


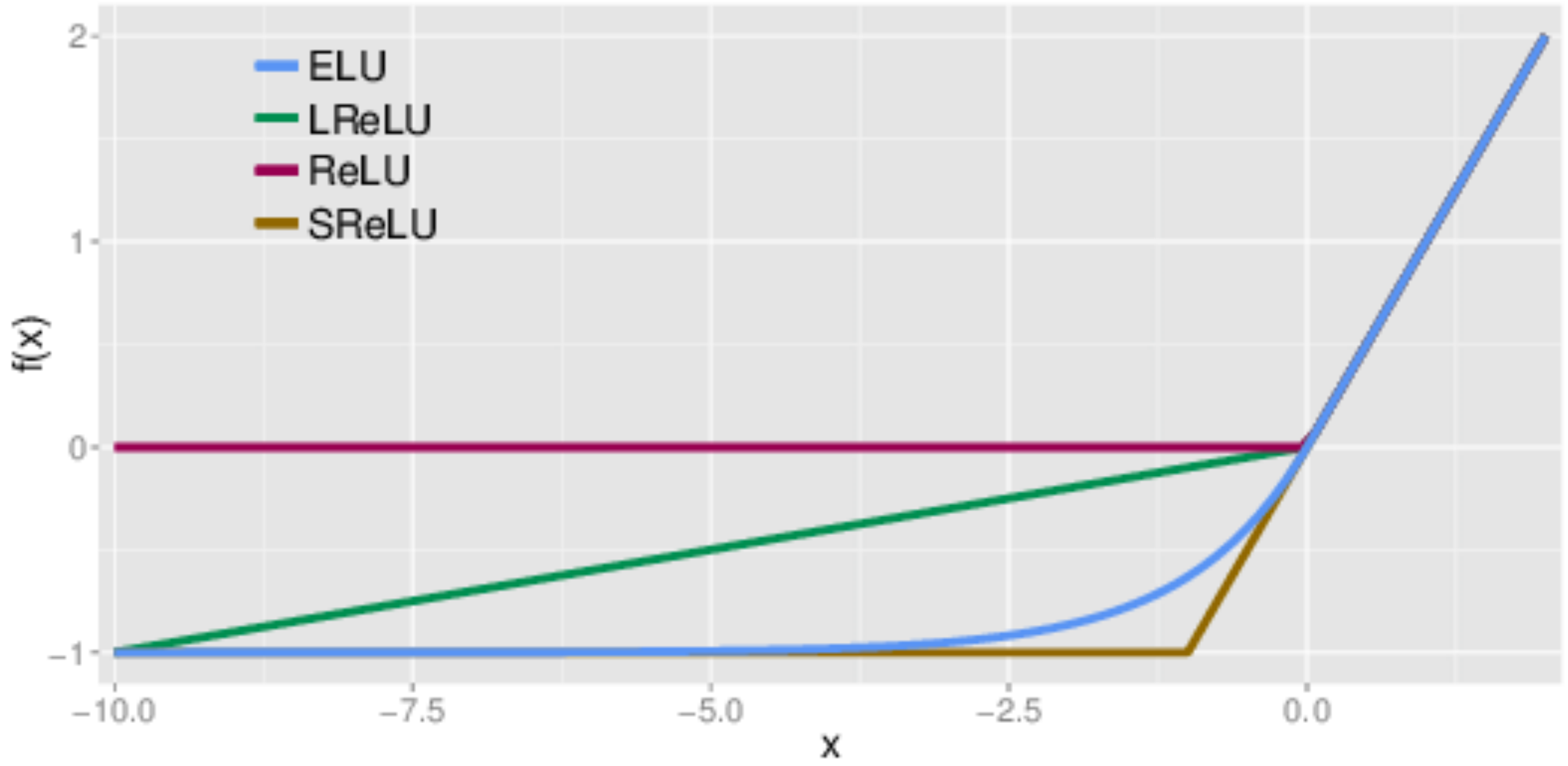


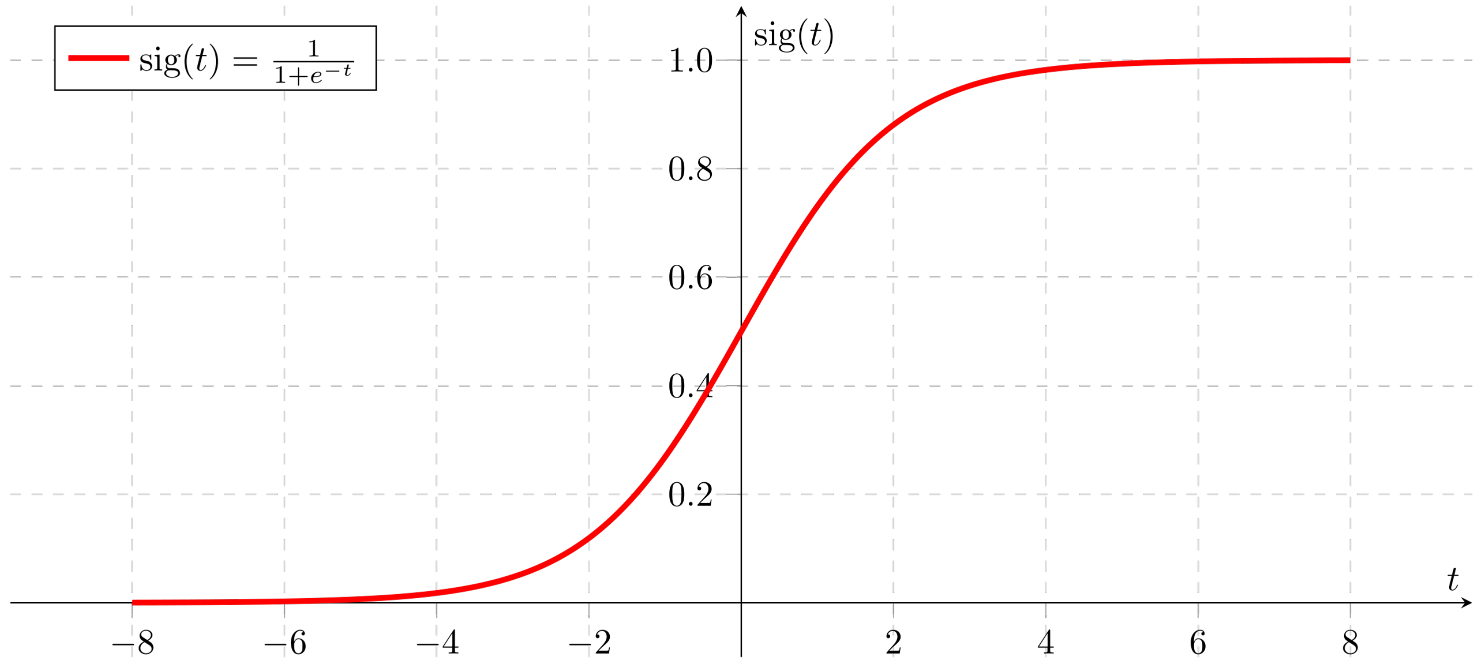
- "Artificial" neuron / "Perceptron"
- Inputs: x (flatten), outputs: y,
- Outputs $y = \sigma(W \times x + b)$
- W and b are the trainable weights and biases
- σ is the activation function, to bring non-linearity to the NN

Multi Layer Perceptron

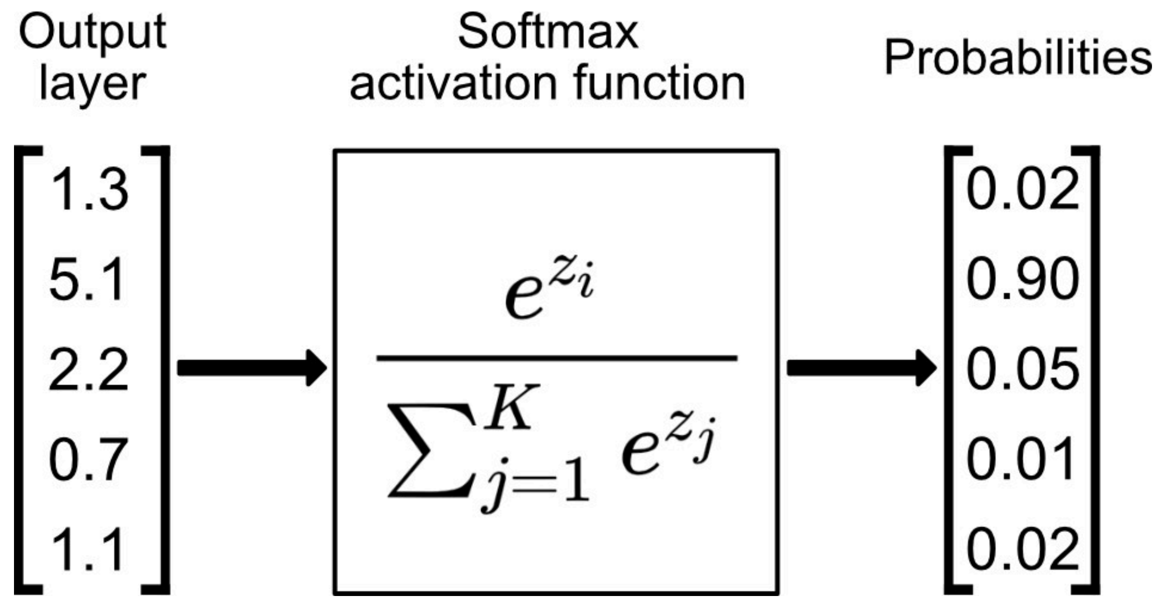




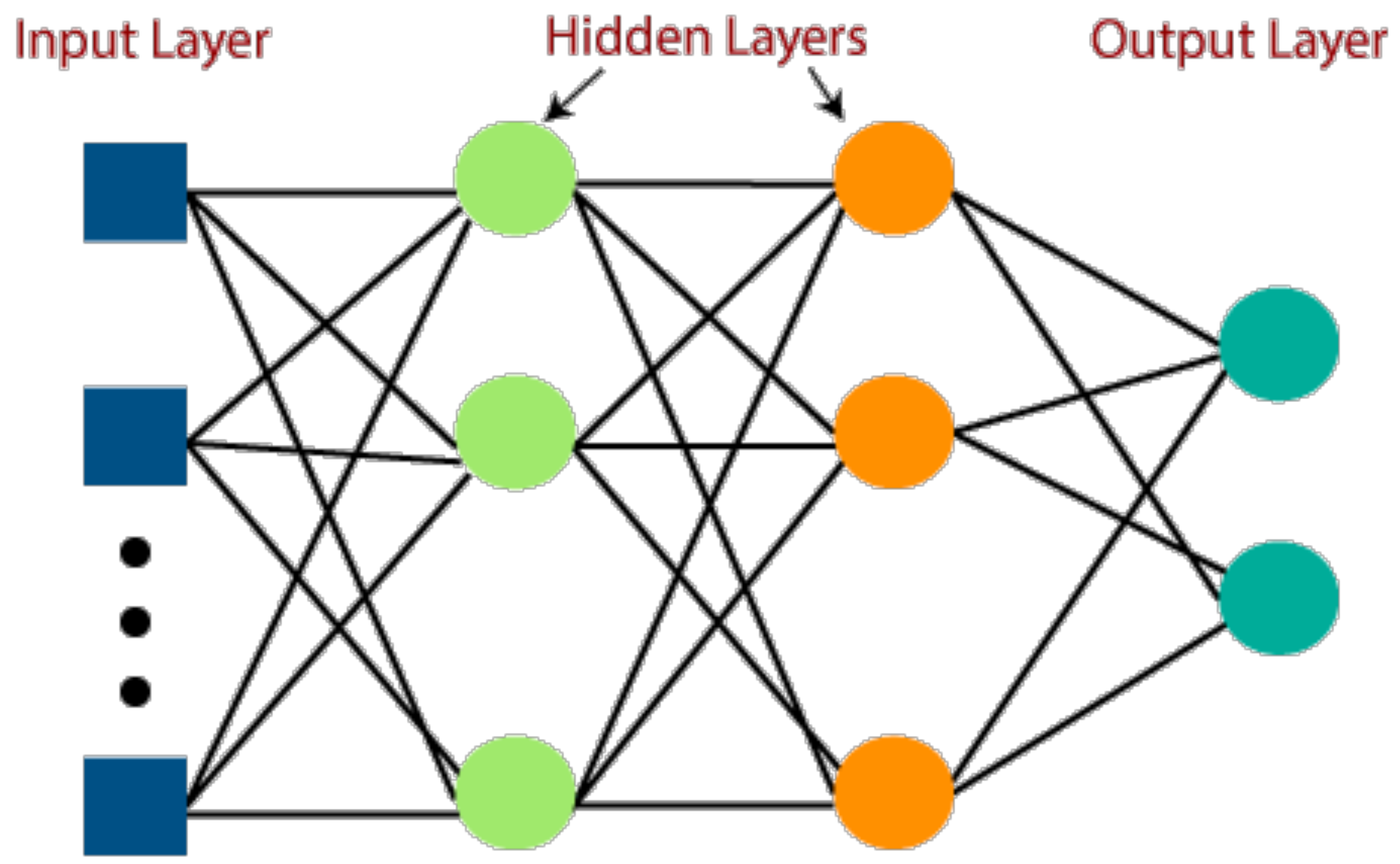


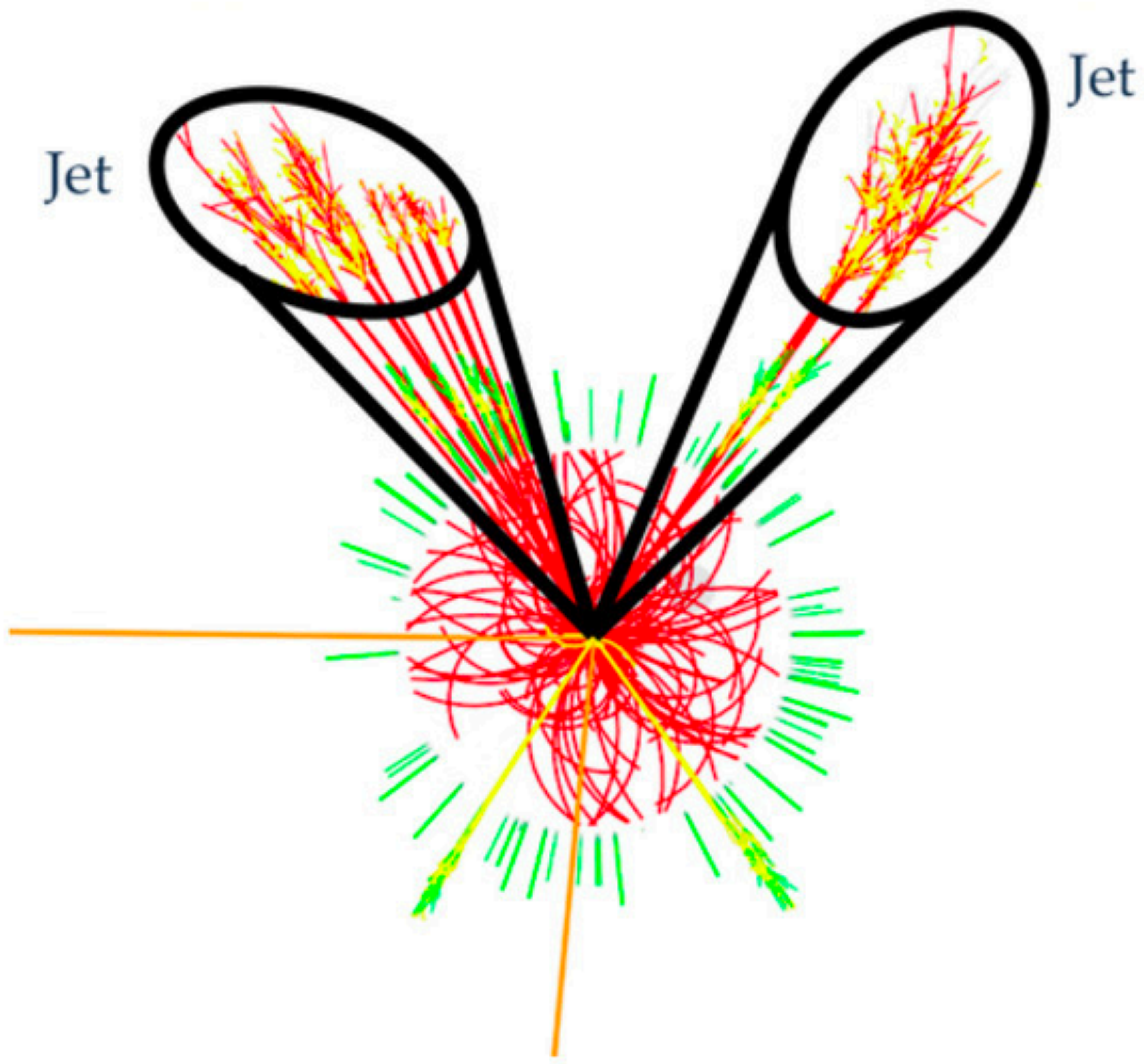


- σ is the activation function, to bring non-linearity to the NN:
  - ReLU for regression problems
  - Sigmoid for classification problems; Softmax for multi-classification problems



Multi Layer Perceptron





- Relatively easy to train and deploy;
- but everything has to be “flat” -> “Geometric”/localized information are lost

Convolutional Neural Network

- E.g: To identity a specific kind of galaxy:

Convolutional Neural Network

- E.g: To identity a specific kind of galaxy:
  - Need “local” information: Conv (with Kernel)

Convolutional Neural Network

- E.g: To identity a specific kind of galaxy:
  - Need "local" information: Conv (with Kernel)

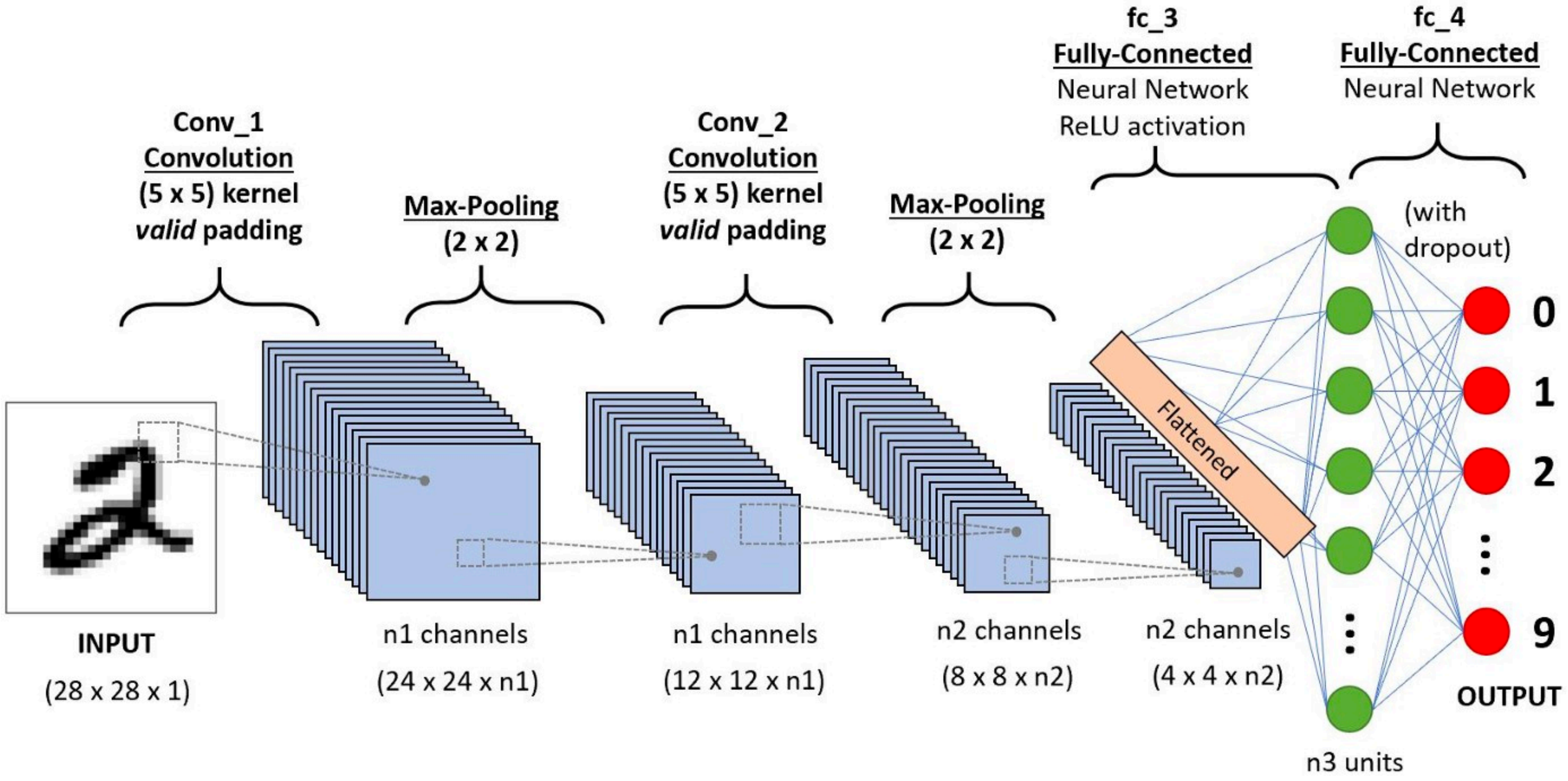



Convolutional Neural Network

- E.g: To identity a specific kind of galaxy:
  - Need "local" information: Conv (with Kernel)
  - Need to "combine" all local information together: Pooling: Max, Mean, Sum, etc

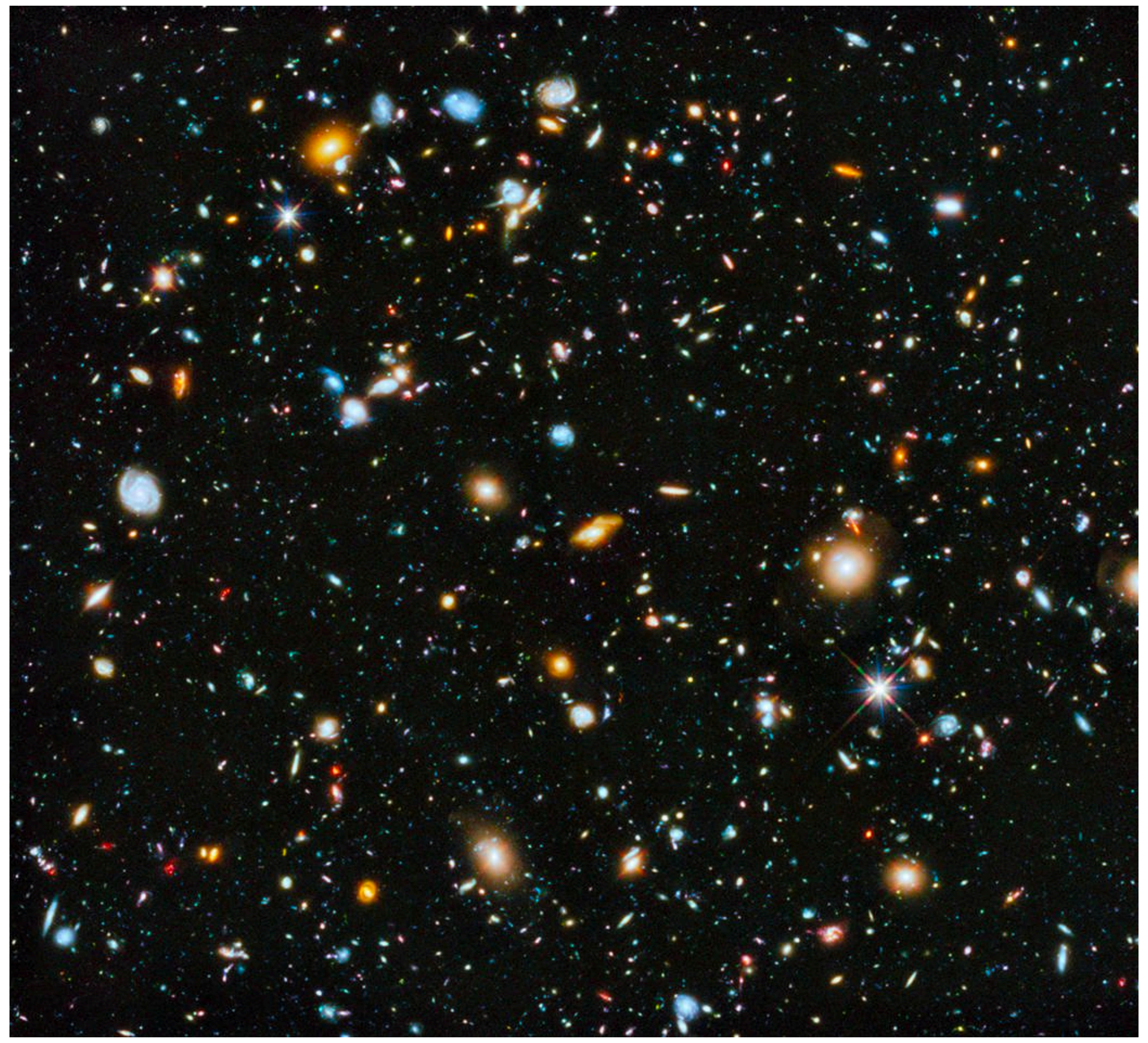
Convolutional Neural Network

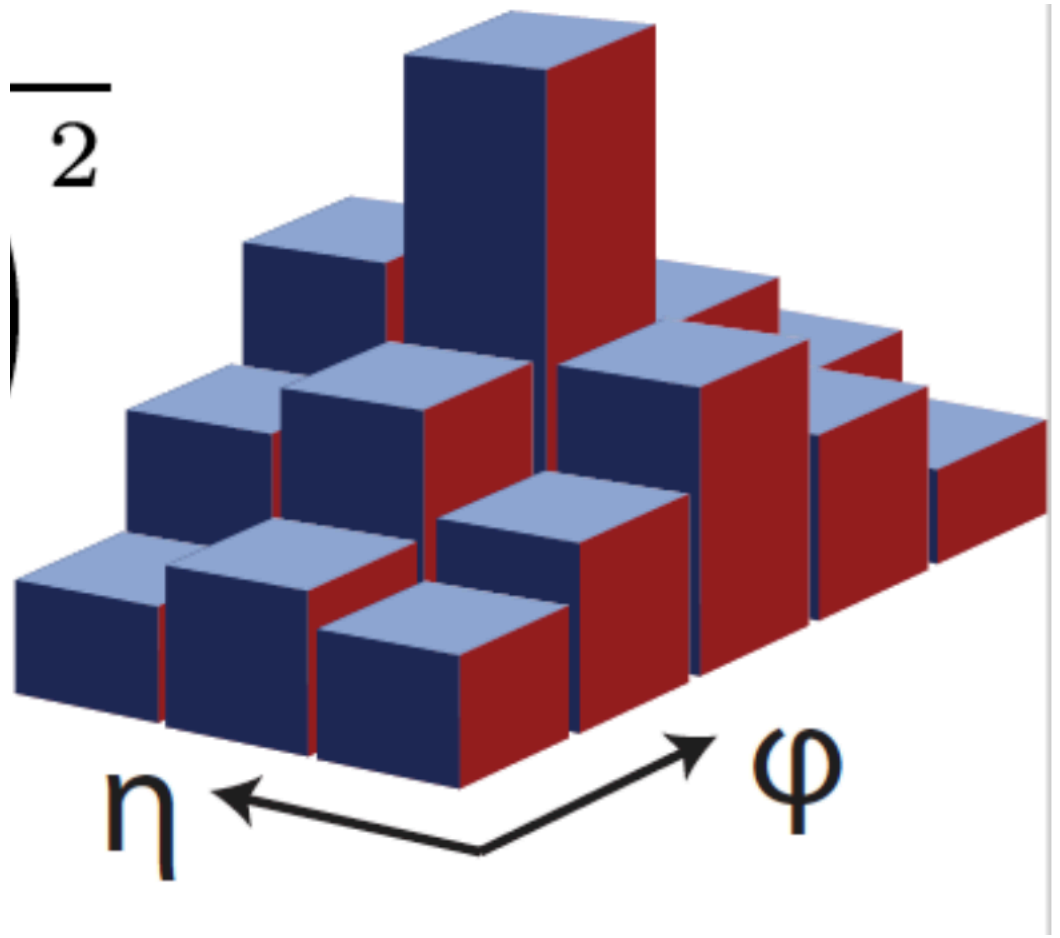
- Convolutional Neural Network
  - Need “local” information: Conv (with Kernel)
  - Need to “combine” all local information together: Pooling: Max, Mean, Sum, etc

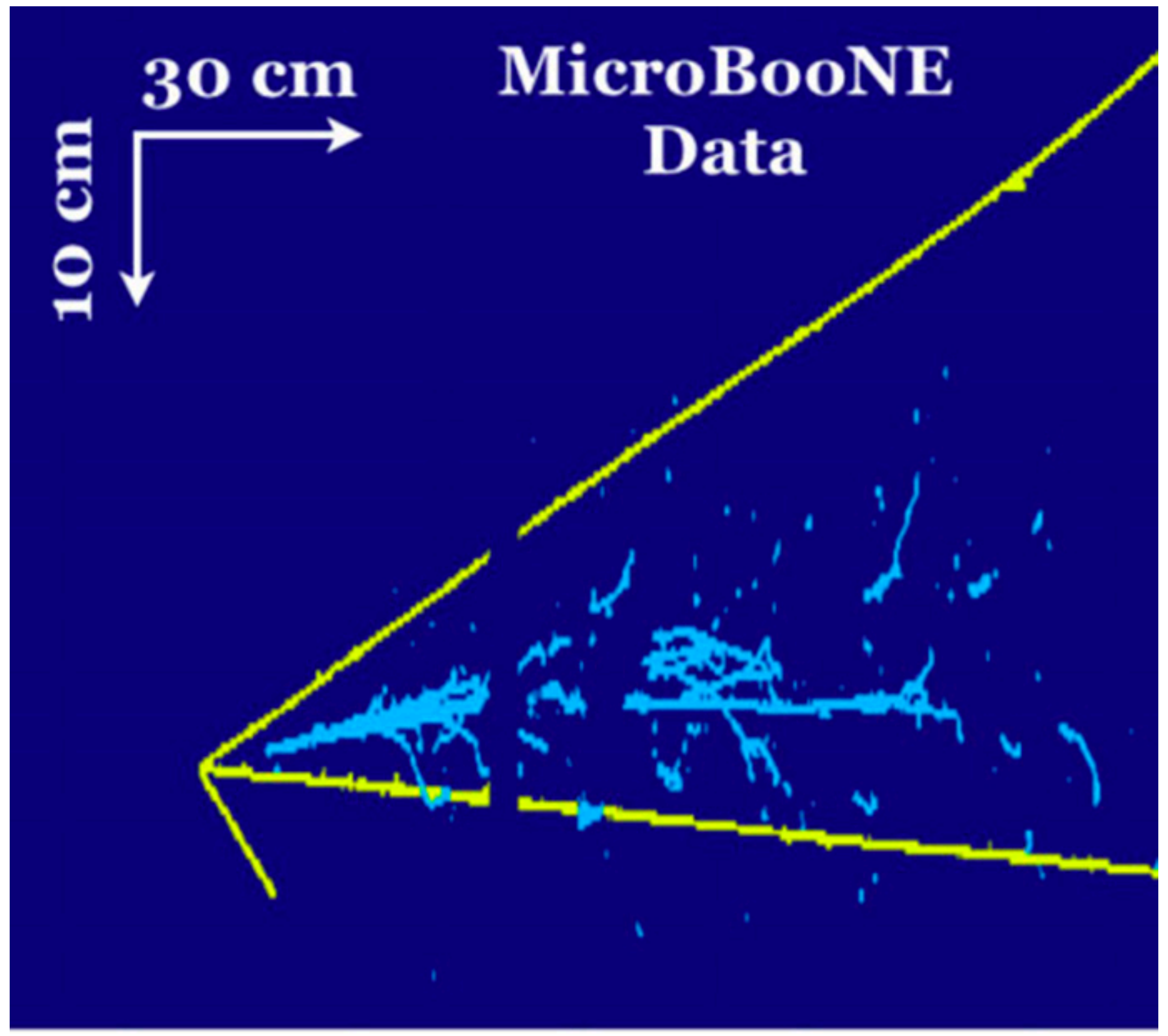


Convolutional Neural Network

- Convolutional Neural Network
  - Works well on image-like data (Euclidean space); computing-wise efficient and fast: same kernel applied everywhere





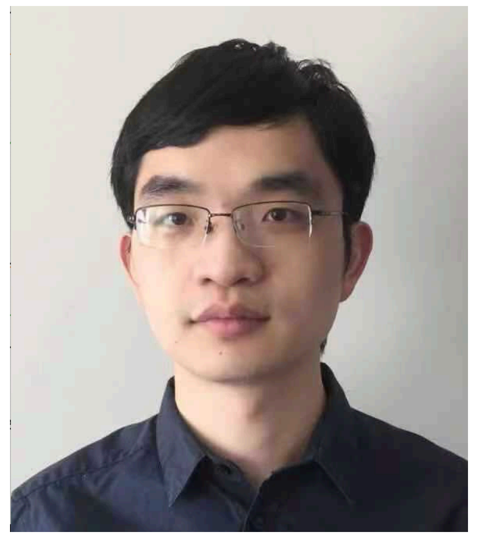


Convolutional Neural Network

- Convolutional Neural Network
  - Works well on image-like data (Euclidean space); computing-wise efficient and fast: same kernel applied everywhere
  - But not all data are image-like;

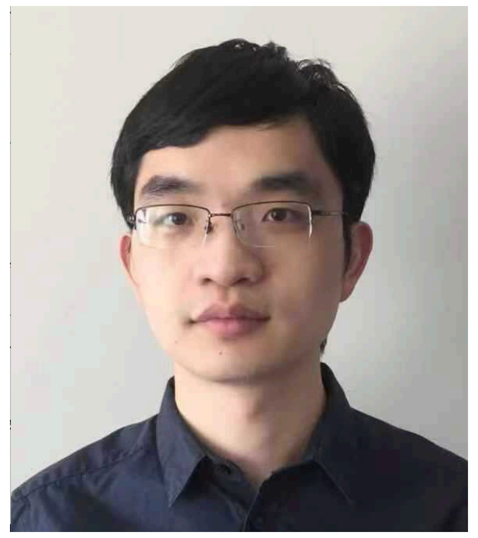
Convolutional Neural Network

- Convolutional Neural Network
  - Works well on image-like data (Euclidean space); computing-wise efficient and fast: same kernel applied everywhere
  - But not all data are image-like: social relationship



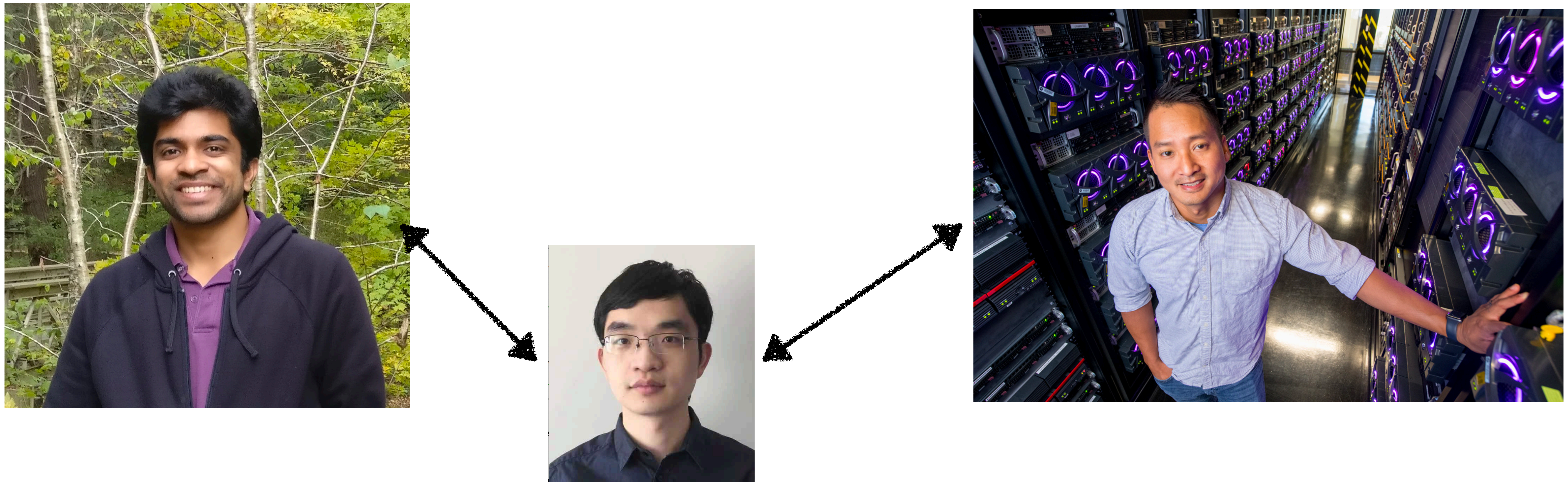
Convolutional Neural Network

- Convolutional Neural Network
  - Works well on image-like data (Euclidean space); computing-wise efficient and fast: same kernel applied everywhere
  - But not all data are image-like: social relationship



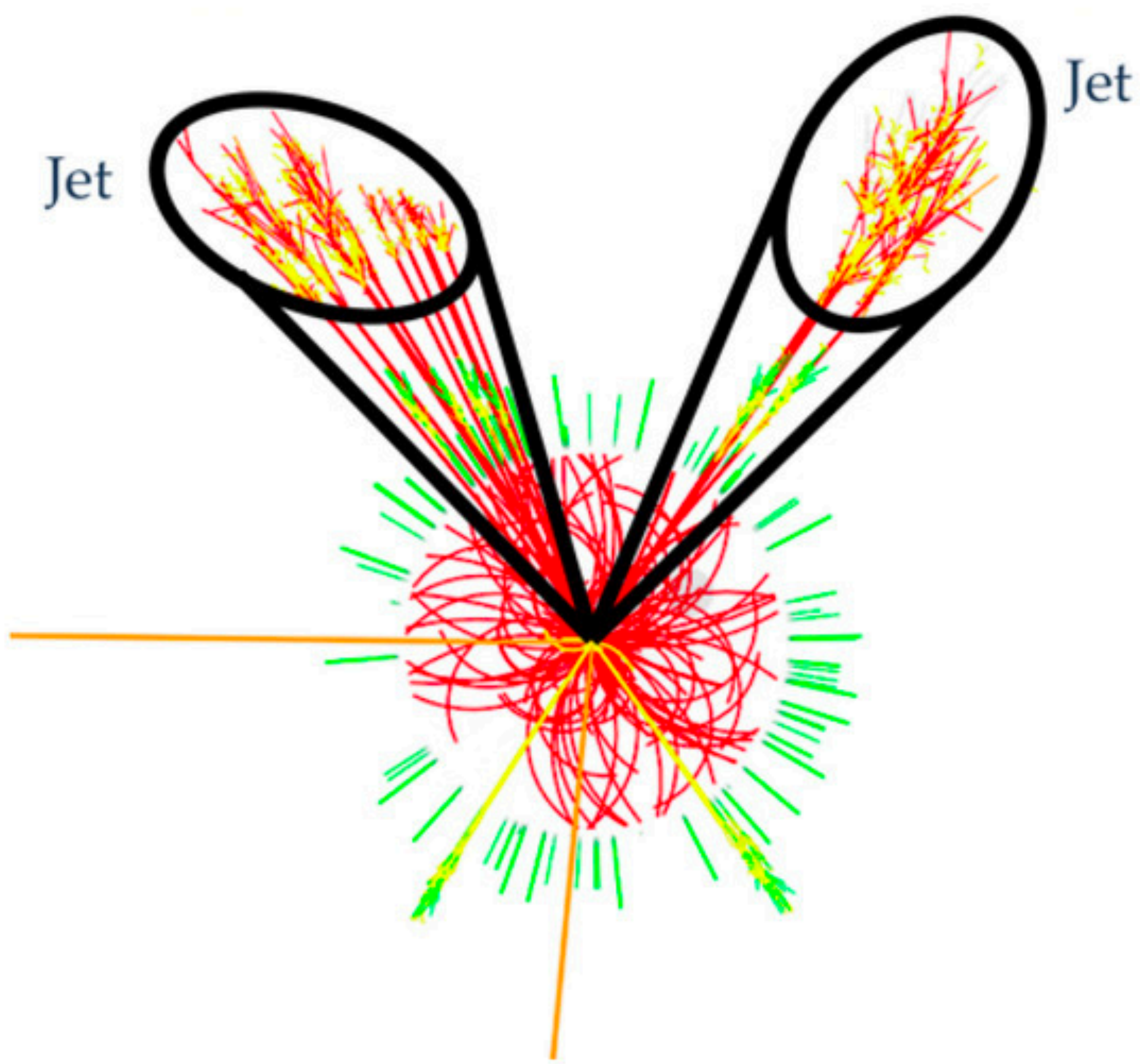
Convolutional Neural Network

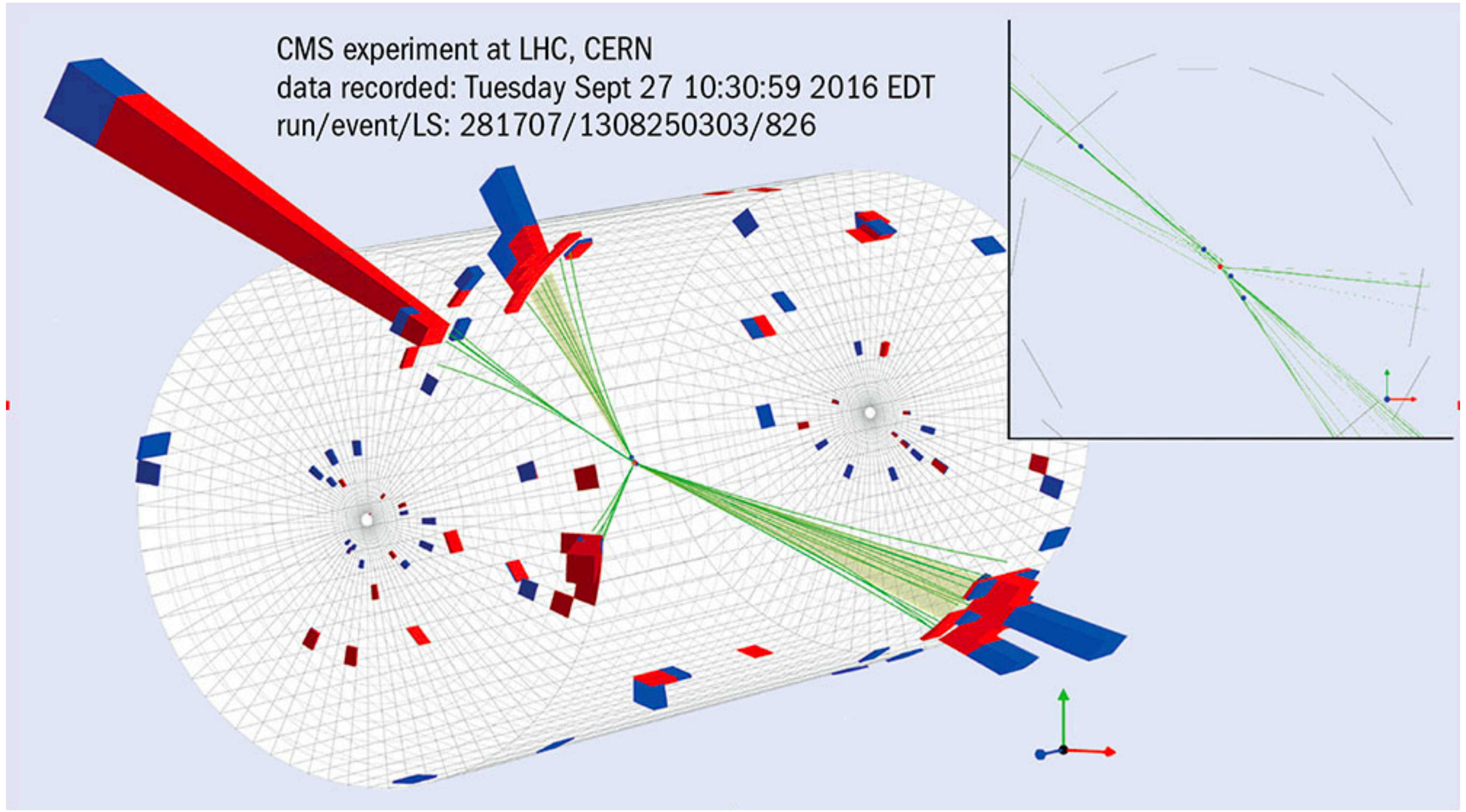
- Convolutional Neural Network
  - Works well on image-like data (Euclidean space); computing-wise efficient and fast: same kernel applied everywhere
  - But not all data are image-like: social relationship



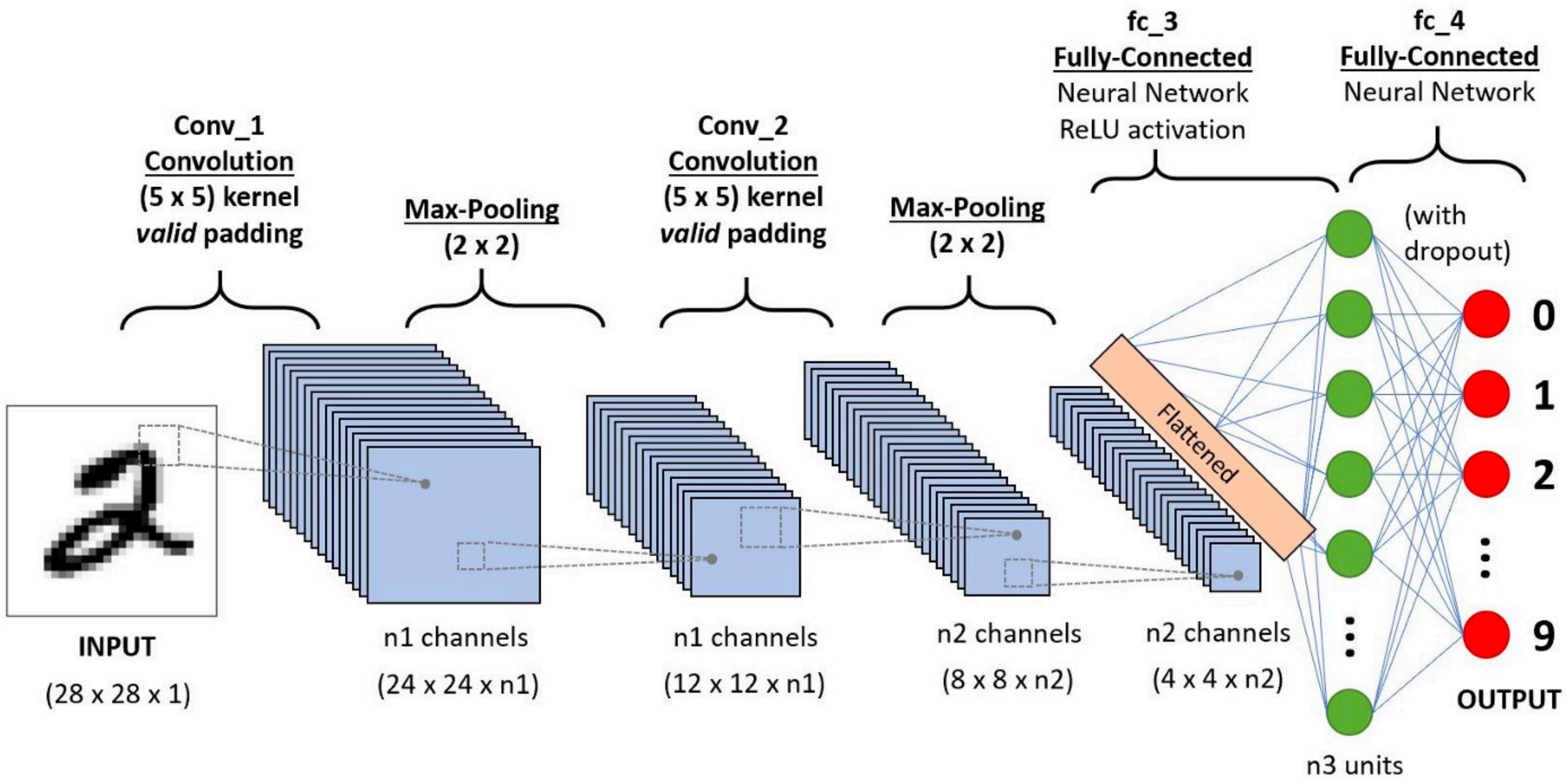
Convolutional Neural Network

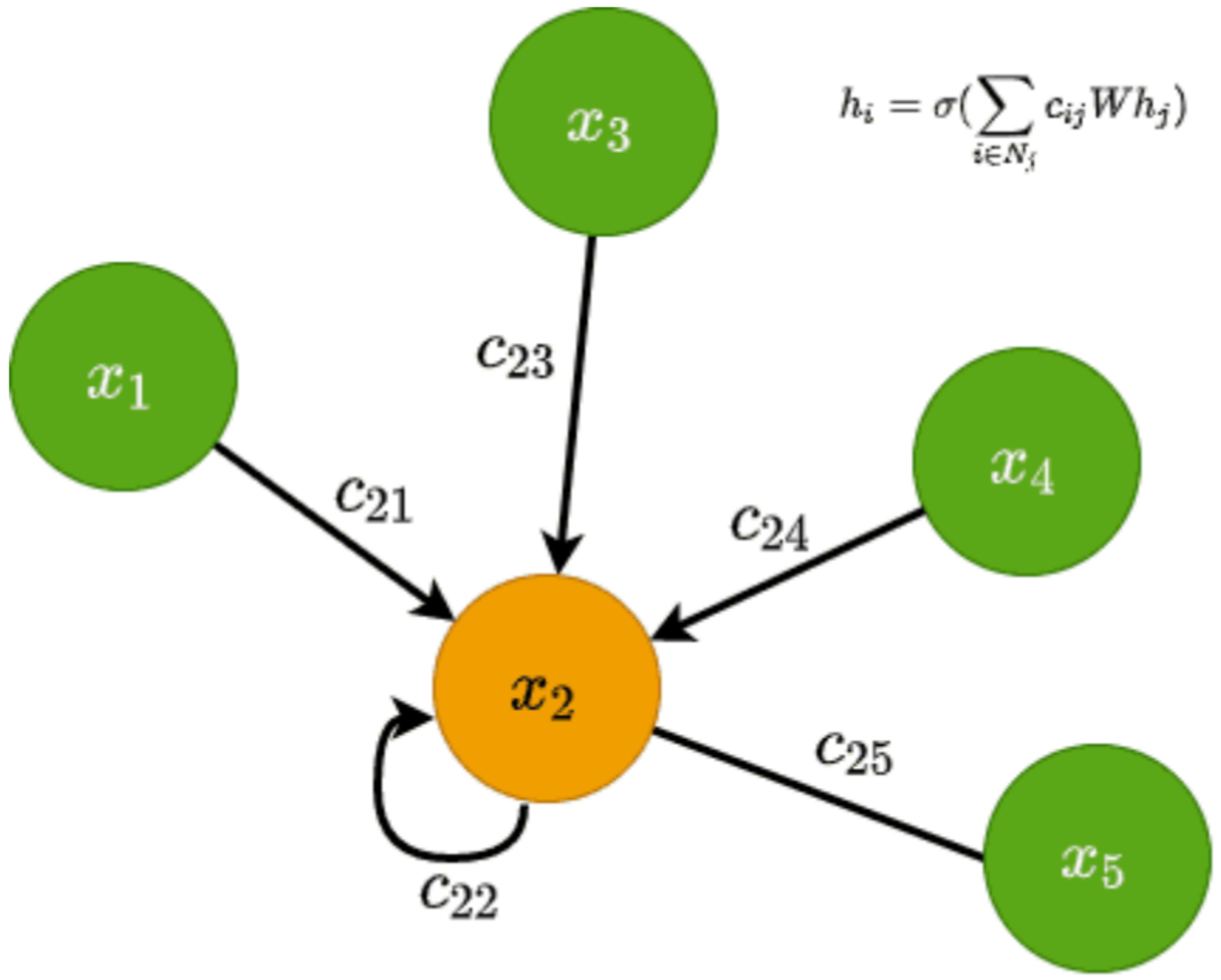
- Convolutional Neural Network
  - Works well on image-like data (Euclidean space); computing-wise efficient and fast: same kernel applied everywhere
  - But not all data are image-like: social network; particles in jets, etc
  - And the resolutions, etc can be different in different phase-space regions





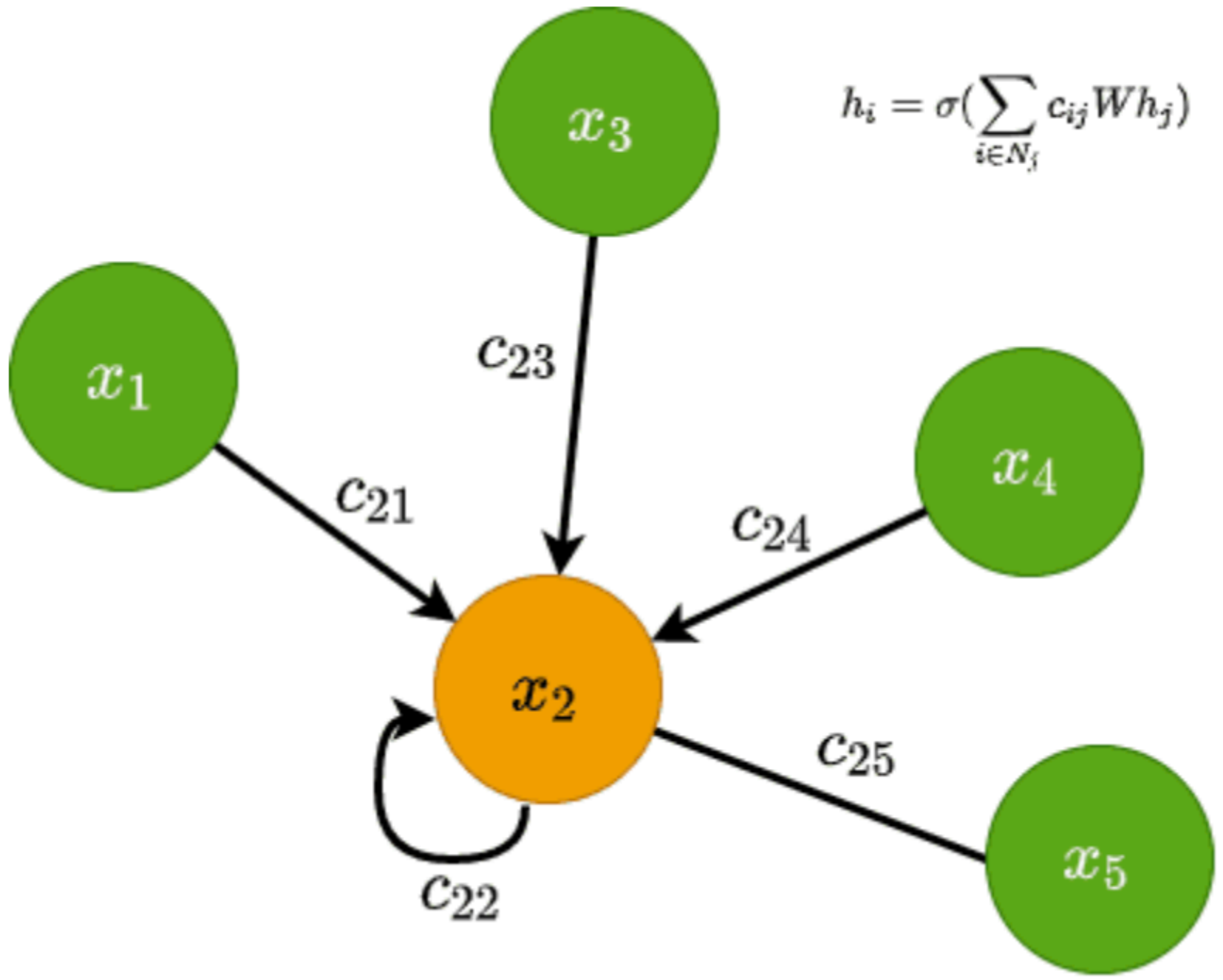
CNN -> GNN





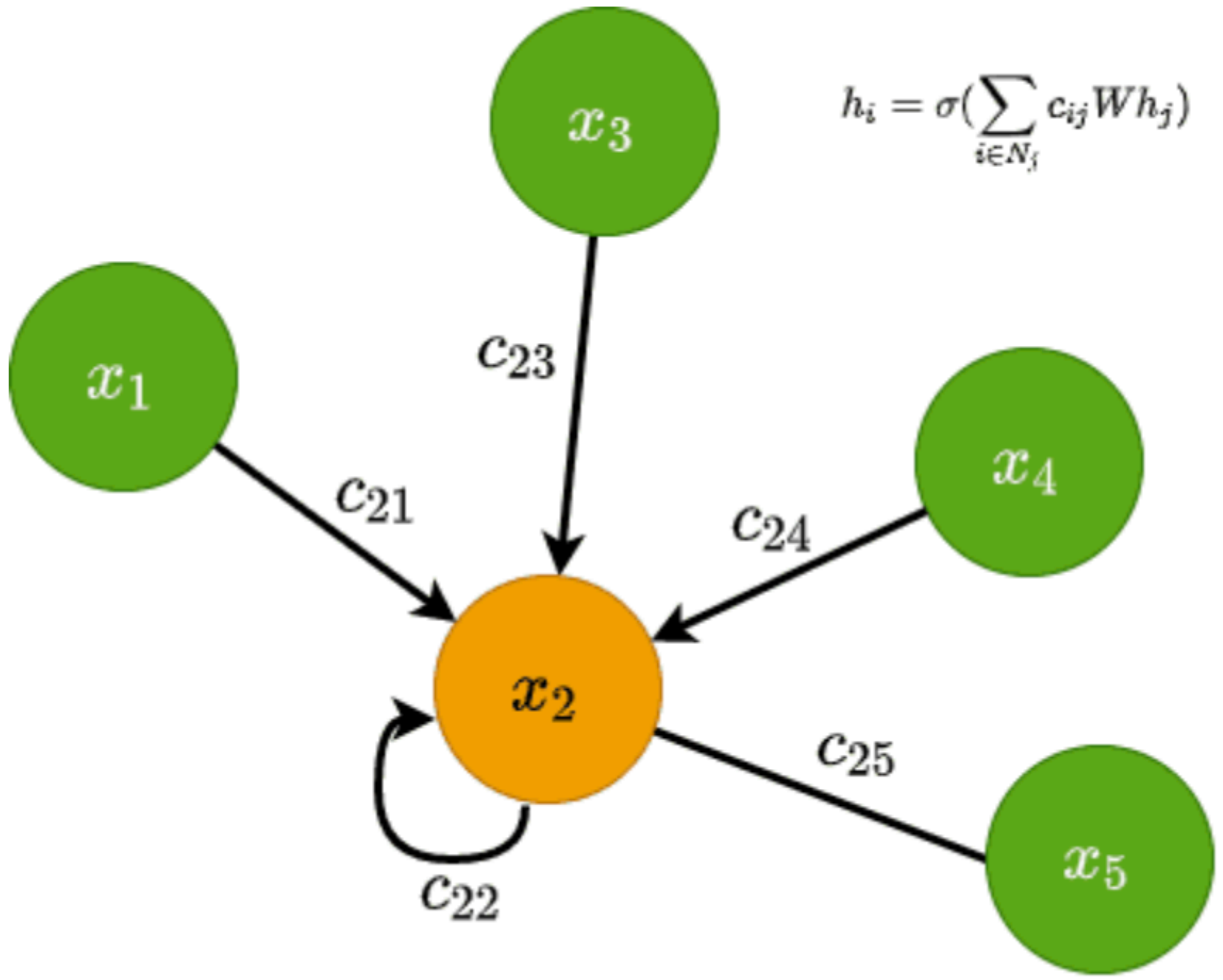
- Convolutional Neural Networks work on Euclidean space and can aggregate information from the “real” neighbors adjacent to each target.
- Moving to Non-Euclidean space; do the similar type of “convolutions” to extract and aggregate information from neighboring particles -> Graph Neural Network (More general and more powerful)

Graph Neural Networks



- One Graph (G) has nodes (V) and edges (E): G = (V,E)
- A set of nodes $\{h_i\}$ and their connections (edges): $\{e_{ij}\}$
- Collect information among the nodes and edges

Message Passing Neural Network



- "message passing": for target node i, "message" passed from neighboring nodes to the target node is:

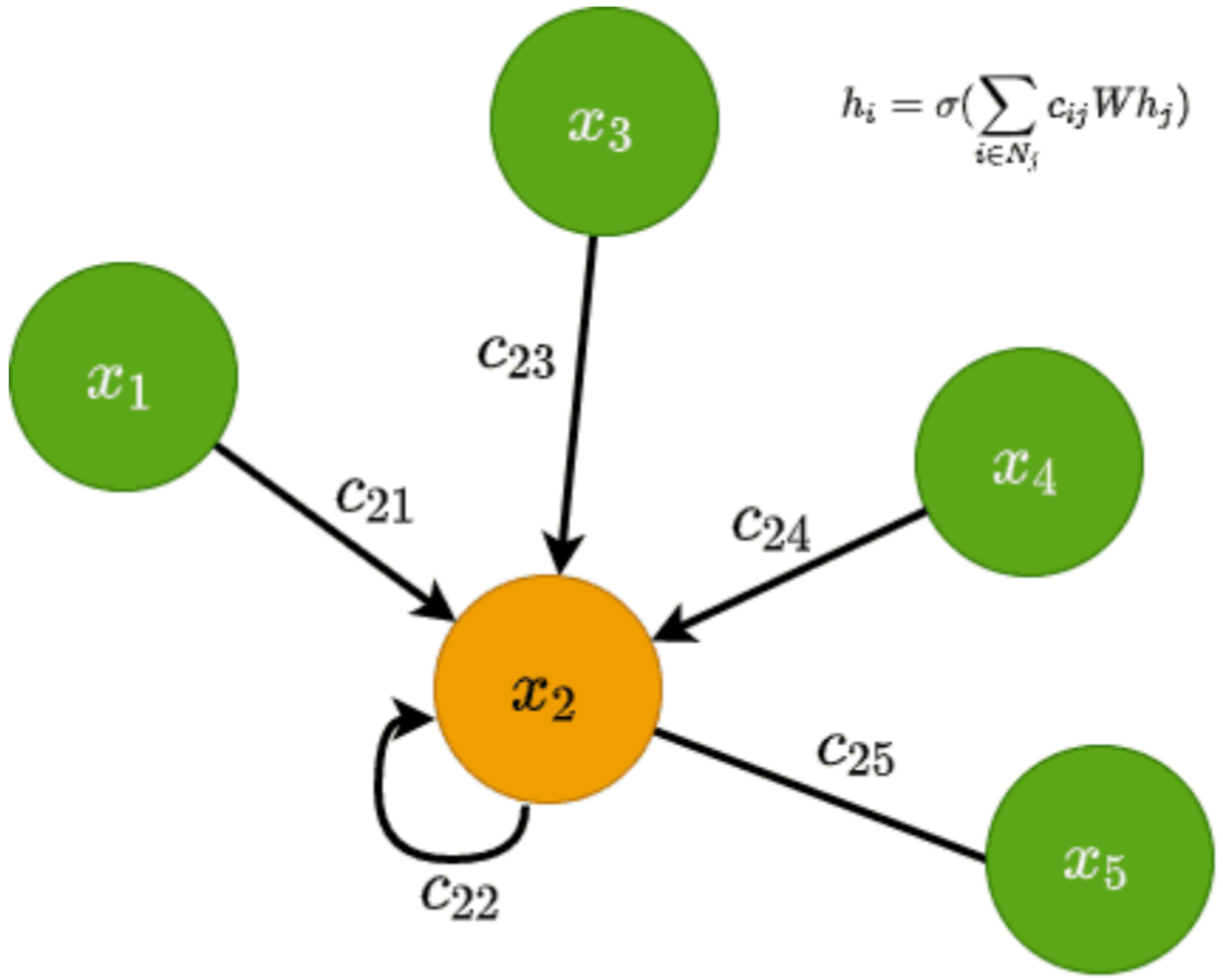
$$m_i^{(k)} = \sum_j M(h_i^{(k)}, h_j^{(k)}, e_{ij})$$

- Node feature update for the target node is:

$$h_i^{(k+1)} = U(h_i^{(k)}, m_i^{(k)})$$

- M and U are message functions and node update functions, respectively.

Message Passing Neural Network



- "message passing": for target node i, "message" passed from neighboring nodes to the target node is:

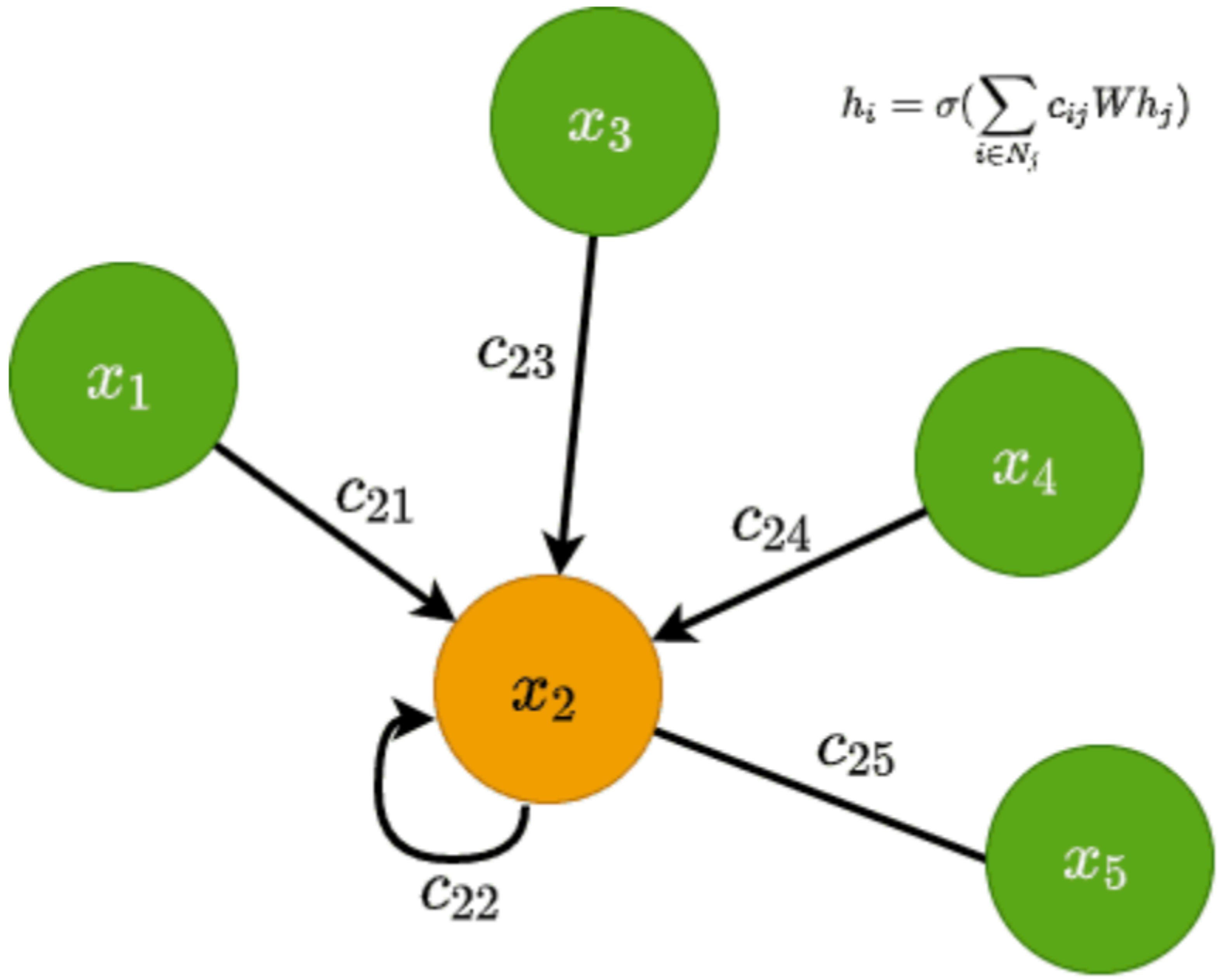
$$m_i^{(k)} = \sum_j M(h_i^{(k)}, h_j^{(k)}, e_{ij})$$

- Node feature update for the target node is:

$$h_i^{(k+1)} = U(h_i^{(k)}, m_i^{(k)})$$

- Finally: with $\{h_i^{(p)}\}$ and $\{e_{ij}^{(P)}\}$, one can do:
    - Node classification: with $f(h_i^{(P)})$
    - Edge classification: with $f(e_{ij}^{P})$ or $f(h_i^{(P)}, h_j^{(P)})$
    - Graph prediction: with pooling of a graph

Message Passing Neural Network



- "message passing": for target node i, "message" passed from neighboring nodes to the target node is:

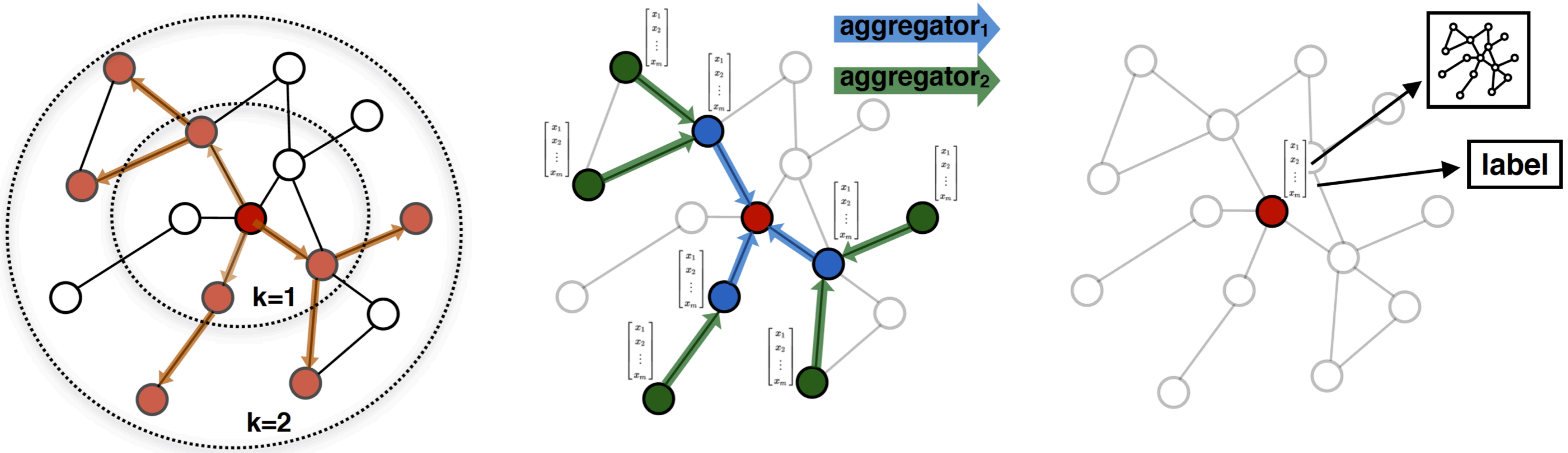
$$m_i^{(k)} = \sum_j M(h_i^{(k)}, h_j^{(k)}, e_{ij})$$

- Node feature update for the target node is:

$$h_i^{(k+1)} = U(h_i^{(k)}, m_i^{(k)})$$

- Can aggregate information from both target node, neighboring node, and the edges;
- can incorporate different kinds of symmetries and assumptions when designing these functions -> very general and powerful

Example: GraphSage



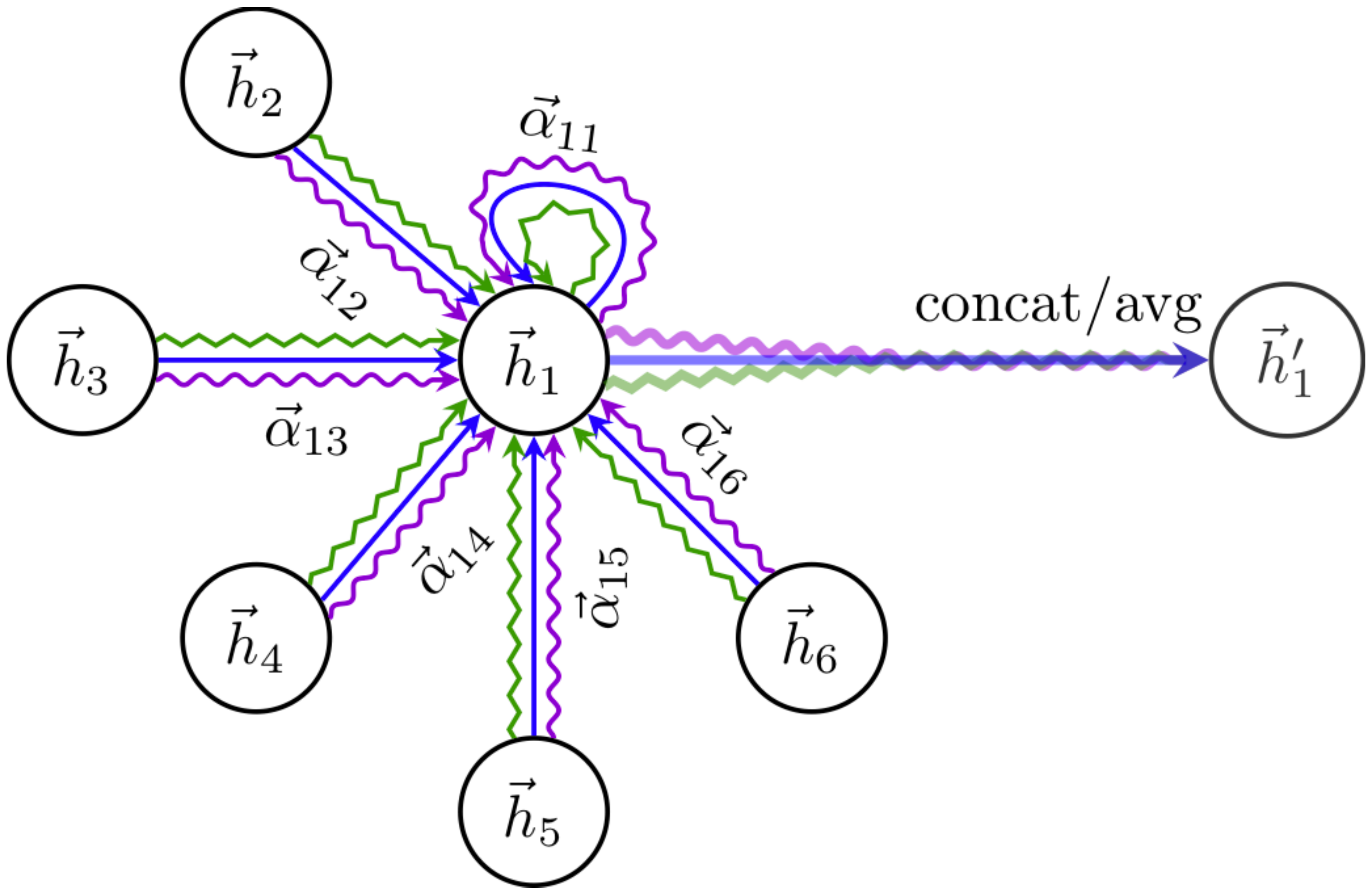
1. Sample neighborhood
2. Aggregate feature information from neighbors
3. Predict graph context and label using aggregated information

- Message: $m_i^{(k)} = \sum_j h_j^{(k)} V$
- Node feature update: $h_i^{(k+1)} = \sigma(h_i^{(k)} W + m_i^{(k)}) = \sigma(h_i^{(k)} W + \sum_j h_j^{(k)} V)$
- Here $\sum_j$ is the pooling operation, can be max, mean, sum, etc;

Example: Dynamic Graph CNN

- After the node feature update: $h_i^{(k+1)} = \sigma(h_i^{(k)}W + m_i^{(k)}) = \sigma(h_i^{(k)}W + \sum_j h_j^{(k)}V)$
- Rebuild the graph in the new $\{h_i^{(k+1)}\}$ latent space, with e.g., k-nearest neighbors
- The graph is dynamic now - the edges can change after one layer

Example: Graph Attention Network



- Graphsage treats all the edges the same; different edges can have different weights when aggregating information
- I.e. the message becomes:

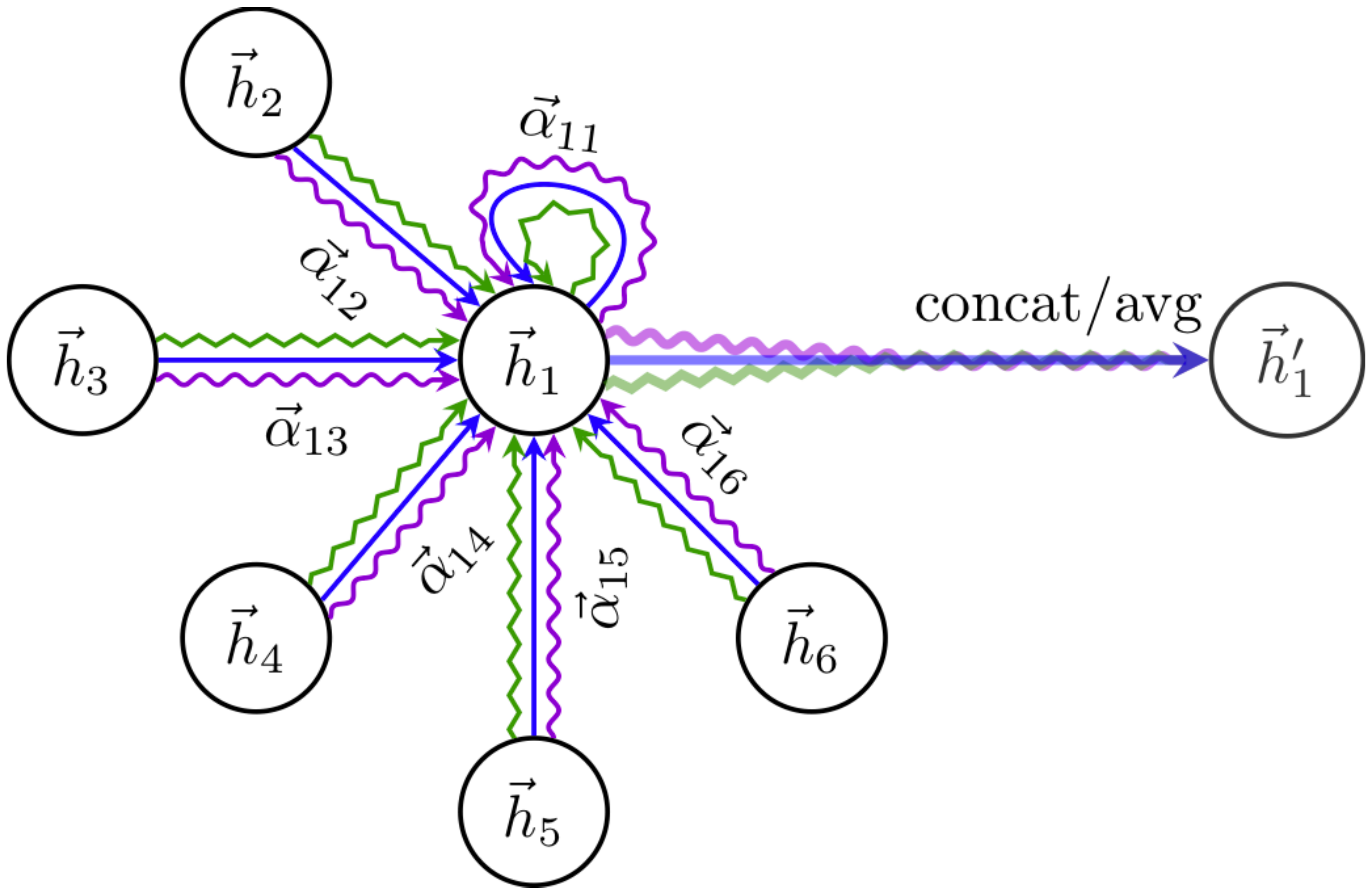
$$m_i^{(k)} = \sum_j h_j^{(k)} V a_{ij}^{(k)}$$

where $a_{ij}^{(k)}$ is "attention" and calculated as:

$$a_{ij}^{(k)} = \mathrm{softmax}(Q^{(k)} h_i^{(k)} \cdot K^{(k)} h_j^{(k)})$$

Q, K, V are often referred to as Query, Key, and Value

Example: Graph Attention Network



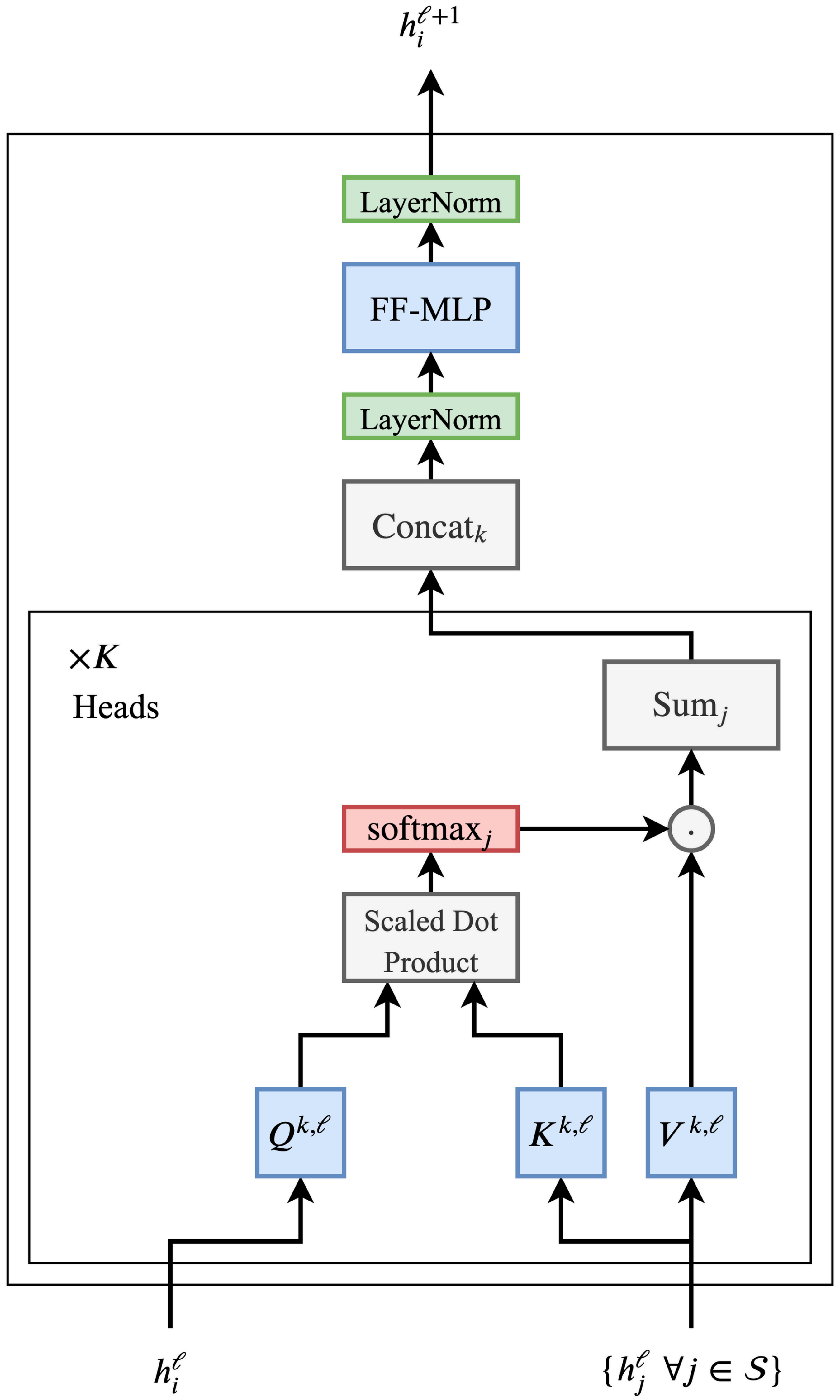
- In partice, one "attention" usually focus on one or a few edges/features
- Need more "attentions" -> multi-head attention
- I.e. the message becomes:

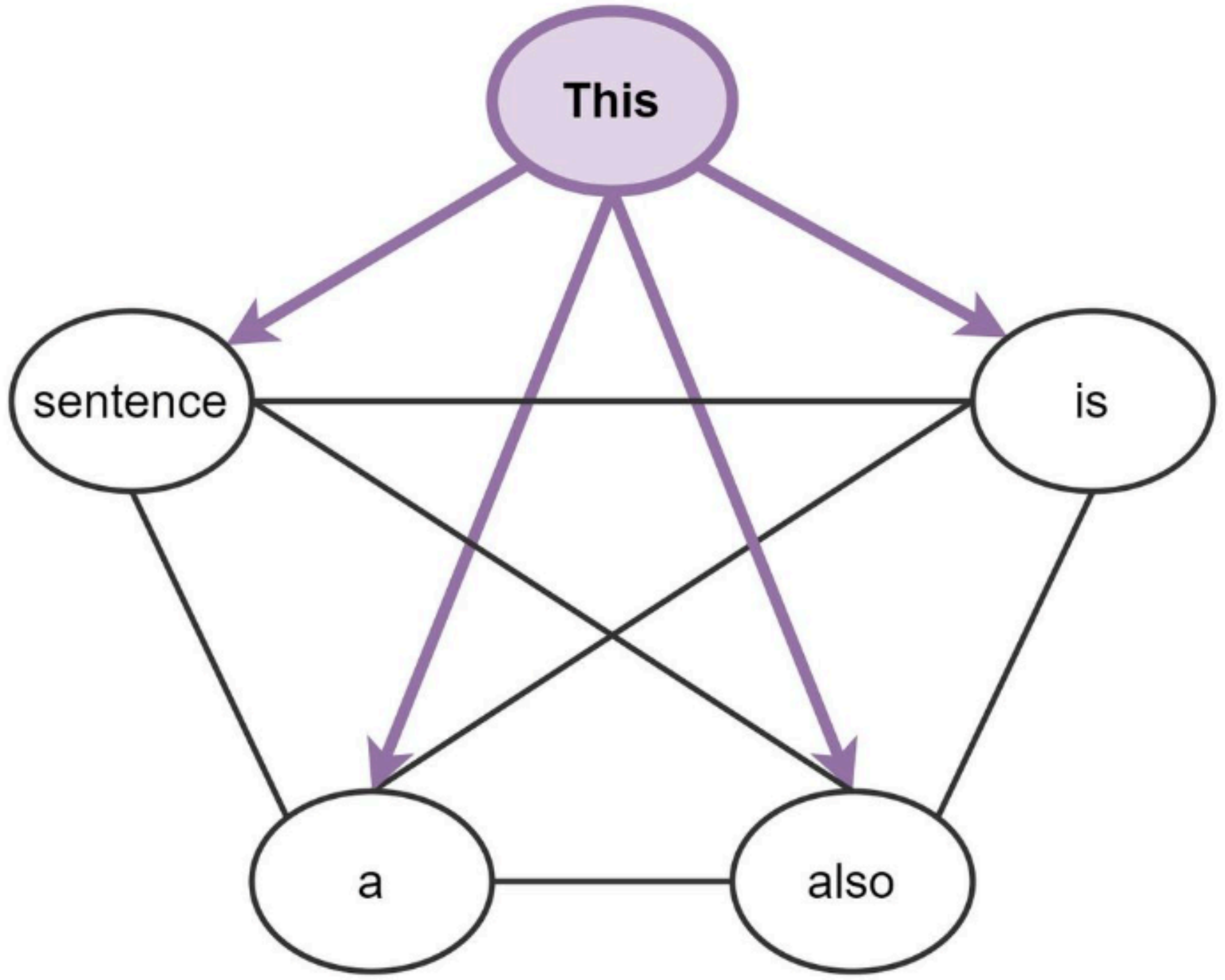
$$m_i^{(k)} = \mathrm{Concat}\Big(\sum_j h_j^{(k)} V^{(l)} a_{ij}^{(k,l)}\Big)$$

where $a_{ij}^{(k,l)}$ is l-th "attention" in the k-th layer:

$$a_{ij}^{(k,l)} = \mathrm{softmax}(Q^{(k,l)} h_i^{(k)} \cdot K^{(k,l)} h_j^{(k)})$$

Graph Attention Network -> Transformer





- Transformer are fully-connected word graph, with multi-head attention, layer-norms, and feed-forward MLP

Goods and Bads

- Goods and bads come at the same time. E.g.:
- Lower and lower level of information, with more advanced architectures, can bring huge boosts to performance increases
- Industry, and open-source community, have provided us lots of tools to play with these, easy to get hands on these
- How much we trust such low-level information, is questionable; calibrations and evaluations of systematic uncertainties can be very hard;
- computing-wise can also take lots of resources

Back Up