Graph Neural Network

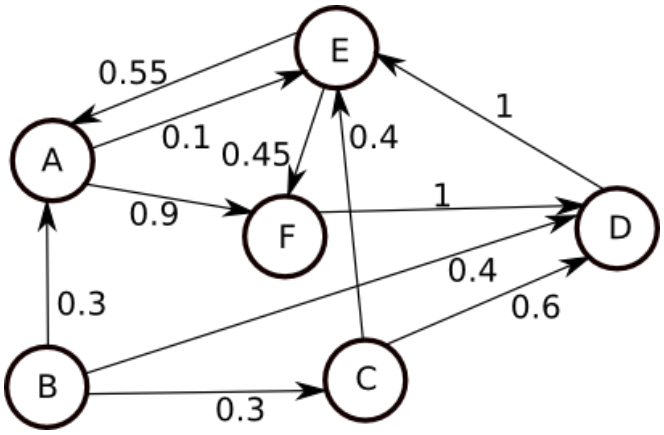
Lin Guosheng
School of Computer Science and Engineering
Nanyang Technological University

Outline

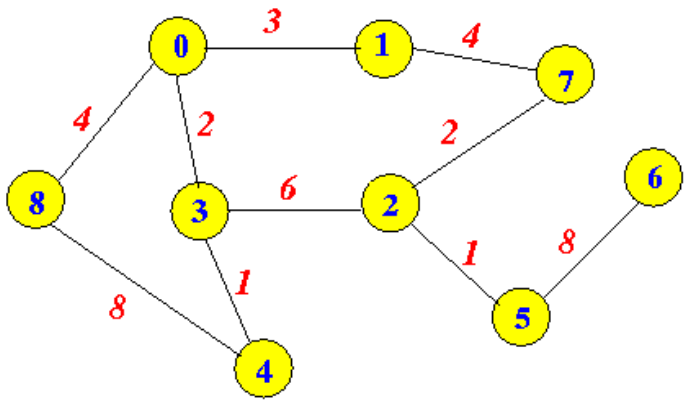
- Introduction to graphs
- Graph convolutional network (GCN)
  - Network prediction (forward pass)
  - GCN applications/tasks
  - Network training
    - Loss functions
    - Backward pass

Introduction to graphs

- A graph
  - Nodes and links (edges). May have weights on links.
    - Weights indicate the strength of links
  - Links can be directed or undirected.
    - One undirected link can be treated as two directed links (forms bidirectional connections)



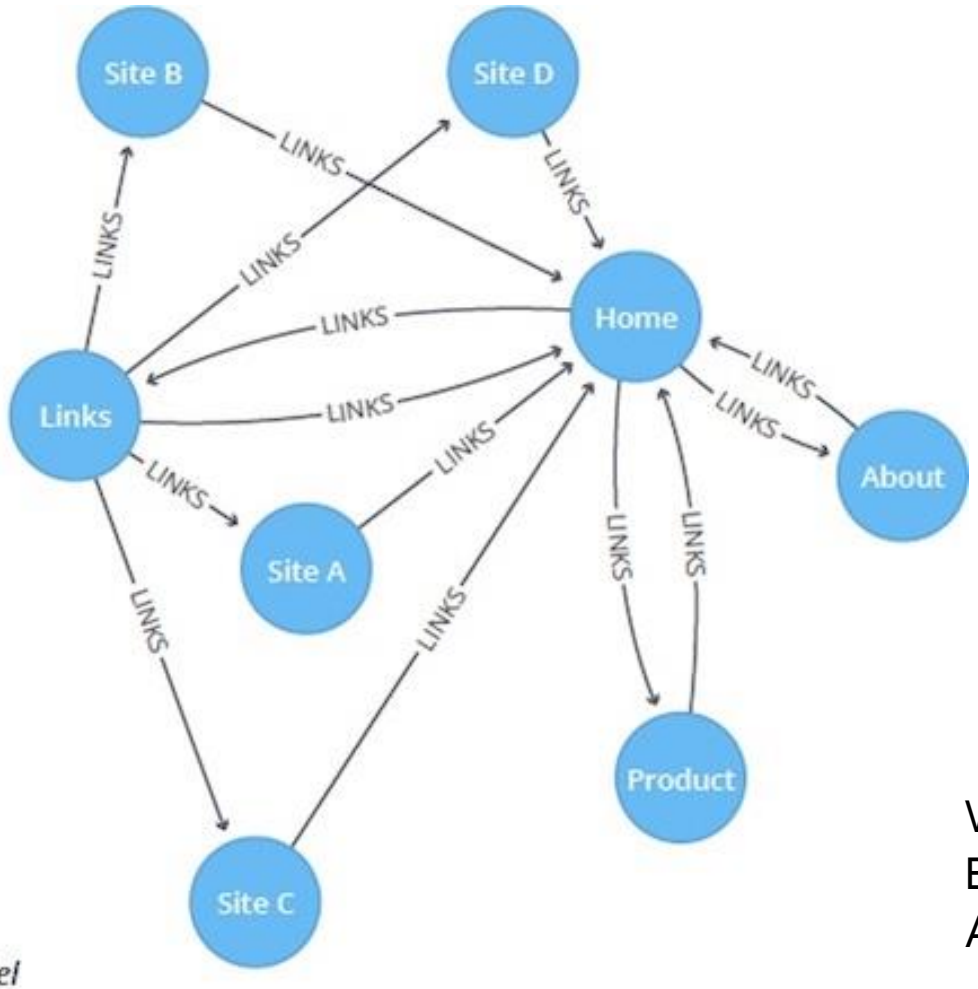
Directed graph



Un-directed graph

http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/11-Graph/weighted.html
https://iwillgetthatjobatgoogle.tumblr.com/post/12156105945/weighted-graph-implementation

Graph examples



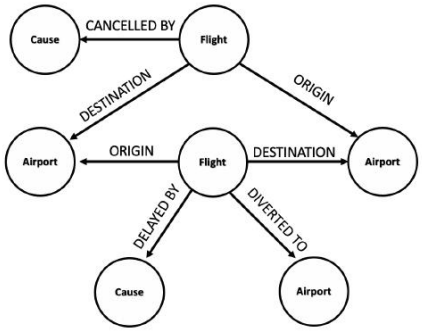
Graph Model

Web pages:
Each web page is a node
A hyperlink is a link

Web pages as graph (PageRank)

Graph examples

Many datasets can be formulated as graphs



Event Graphs

Image credit: SalientNetworks

Computer Networks

Image credit: Pinterest

Particle Networks

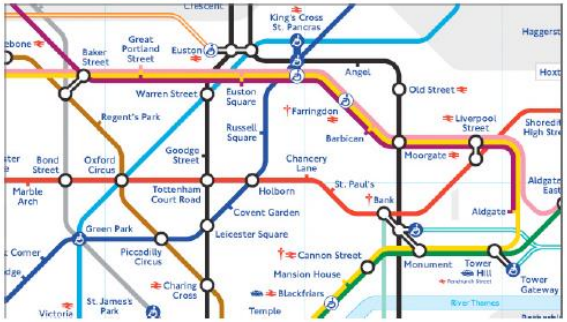


Image credit: visitlondon.com

Underground Networks

http://snap.stanford.edu/class/cs224w-2020/slides/01-intro.pdf

Graph examples

Image credit: Medium

Social Networks

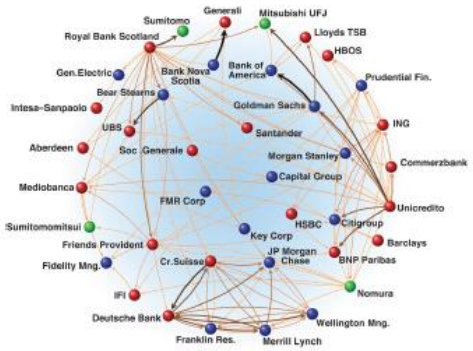


Image credit: Science

Economic Networks

Image credit: Lumen Learning

Communication Networks

Citation Networks

Image credit: Missoula Current News

Internet

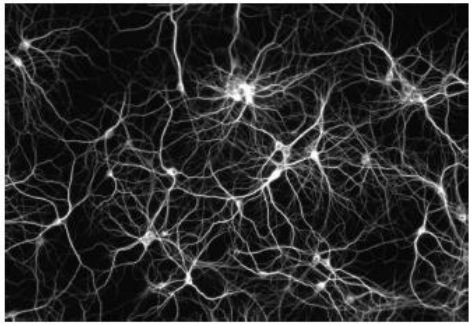


Image credit: The Conversation

Networks of Neurons

Many datasets can be formulated as graphs

Graph examples

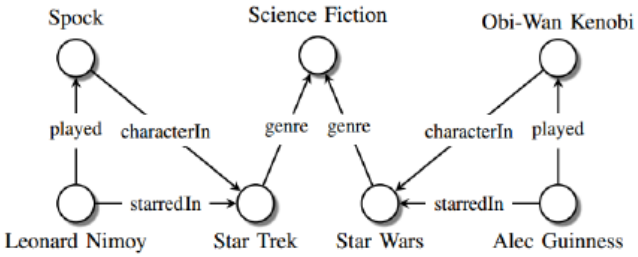


Image credit: Maximilian Nickel et al

Knowledge Graphs

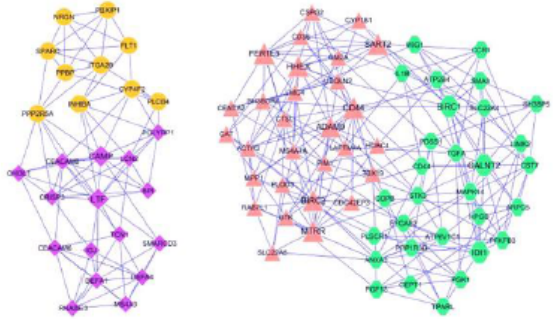


Image credit: ese.wustl.edu

Regulatory Networks

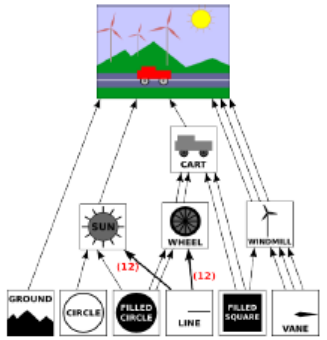


Image credit: math.hws.edu

Scene Graphs

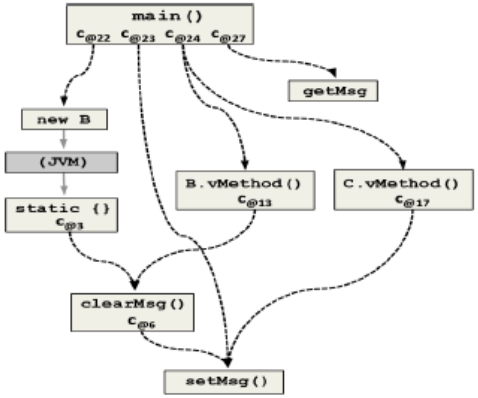


Image credit: ResearchGate

Code Graphs

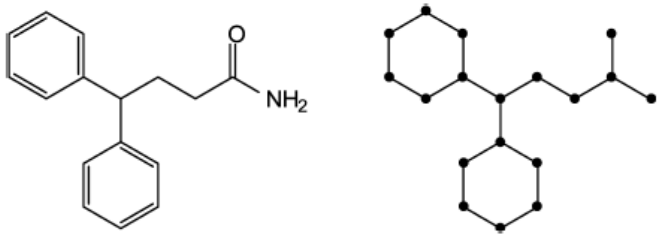


Image credit: MDPI

Molecules

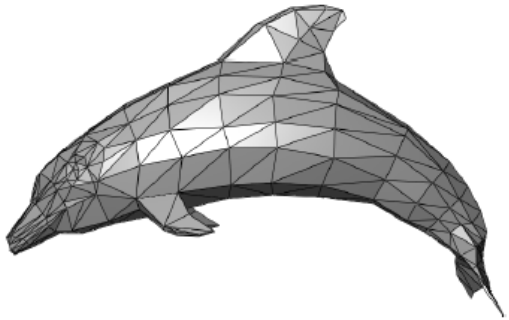
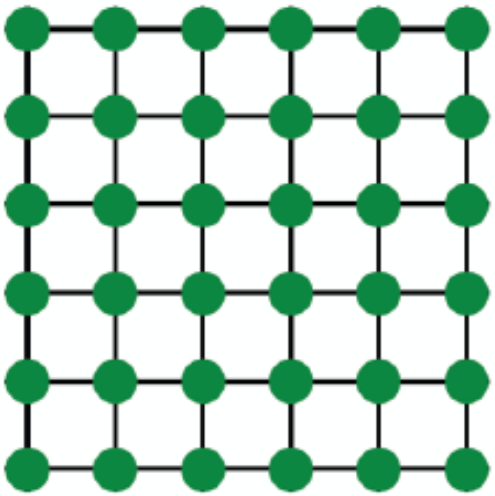


Image credit: Wikipedia

3D Shapes

Many datasets can be formulated as graphs

Graph examples



Images

3D point clouds

Many datasets can be formulated as graphs

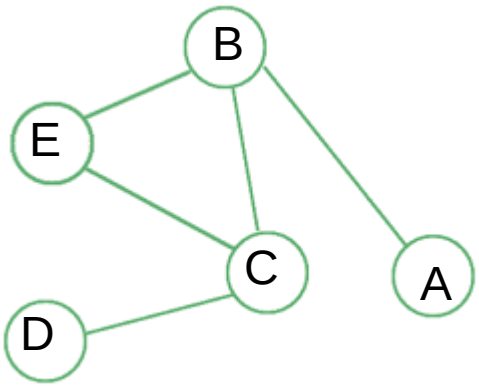
- More examples for formulating graphs
  - Social networks (Facebook, YouTube):
    - Each user is a node
    - A friendship relation is a link, or a contact record forms a link
  - Amazon product data:
    - Each product is a node
    - A co-purchase relationship is a link
  - DBLP publication data
    - Each author is a node
    - A co-author relationship is a link
  - More examples:
    - https://snap.stanford.edu/data/index.html

Graph convolutional network

- Graph neural network (GNN)
  - Neural networks for graph data
- Graph convolutional network (GCN)
  - GCN is a type of GNN
  - Main idea:
    Aggregate the information from neighbourhoods to generate node features.

Node classification task

- GCN output for node classification
  - Predict a class label for each node
    - In practice, we predict **a class score vector** for each node. Each score indicates the confident for one class.
  - In this class, we focus on the node classification task using Graph convolutional network (GCN).



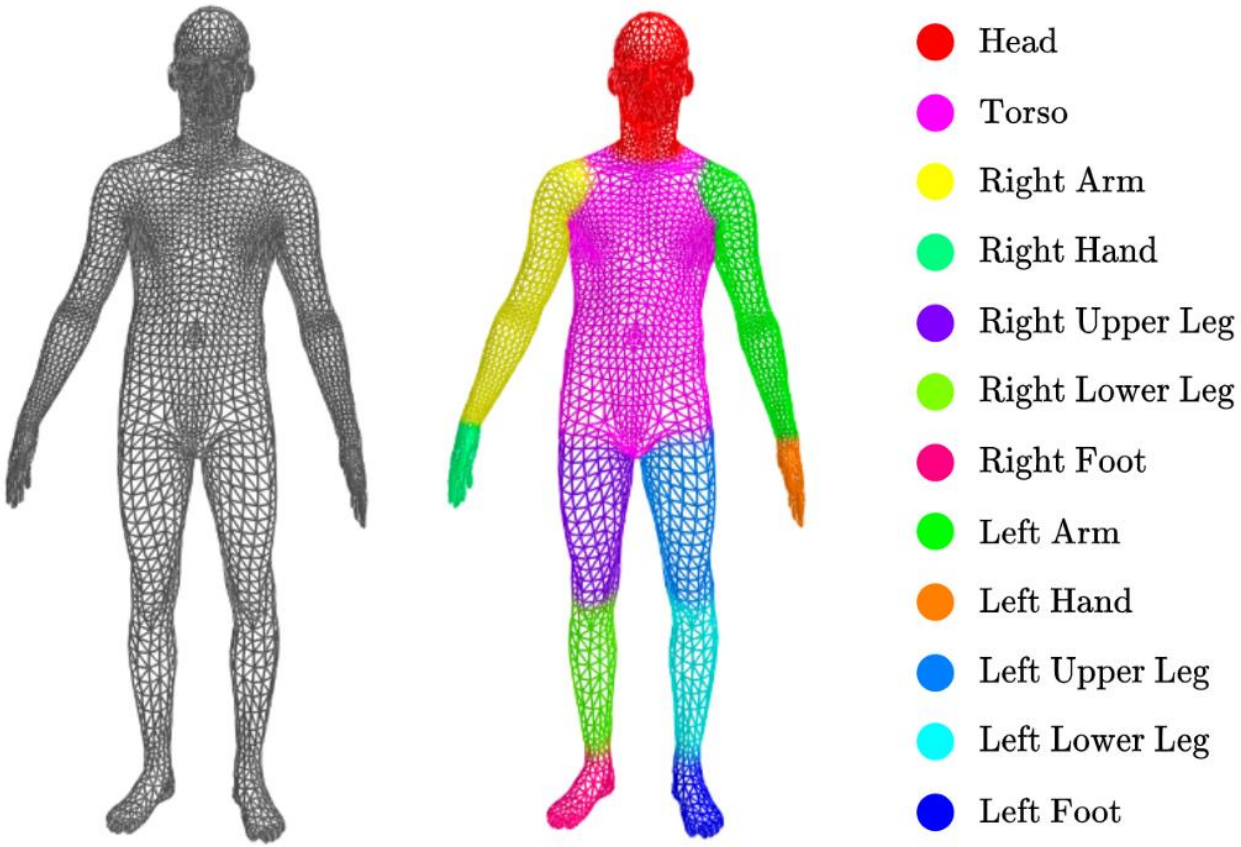
Example: Assume we have 4 classes in total. We aim to predict a 4-dimensional score vector for each node. One dimension corresponds to one class

For node A, suppose the GCN output is the following score vector: [0.1, 0.1, 0.6, 0.2]

As the 3rd element has the largest value, we predict the class label as 3 for node A.

An application example

GCN for node classification on 3D meshes:



Graph segmentation task: predict a body-part class label for each vertice.
Node features: (x,y,z) coordinates of each vertice

https://medium.com/stanford-cs224w/deep-learning-on-3d-meshes-9608a5b33c98

GCN learning tasks

- GCN for other learning tasks
  - Example: graph-level classification
    - Classification for the whole graph rather than each node.
  - Example: node regression problem
    - Predict a real number for each node.

GCN

- Graph convolutional network (GCN)
  - Input: node features and graphs (affinity matrix)
  - Forward pass for prediction
  - Backward pass for training

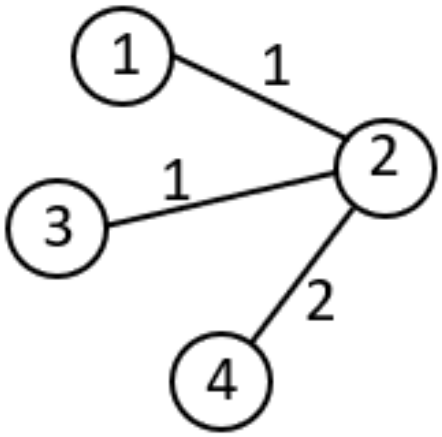
GCN input

- Input of GCN:
  - 1. The input (initial) feature vector for each node.
    - We assume the feature vector for each node is available.
      - The feature vector encode the information for each individual node
    - The generation of the input feature vectors for each node depends on the applications.
      - E.g., in the 3D point segmentation example, the feature vector for each node is the (x, y, z) coordinates of the vertice
  - 2. The graph affinity matrix A.
    - describing the connections in a graph

GCN input

- The graph affinity matrix A.
  - In practice, we use row normalized affinity matrix A'.

$a_{i,j}$: The element (i,j) in the affinity matrix A.
It is the edge weight of the edge between node i and node j. If there is no link between (i, j), $a_{i,j} = 0;$



Input graph

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}$$

affinity matrix

- Normalized affinity matrix A'.

Row normalized affinity matrix A': the sum of each row in A' is 1.
The (i,j) element in A' is :

$$a'_{ij} = \frac{a_{ij}}{\sum_{k=1}^{N} a_{ik}}$$

$\sum_{k=1}^{N} a_{ik}$: is the sum of the i-th row in the original Affinity matrix A for node i

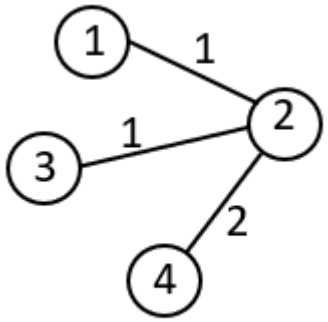
$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \Rightarrow A' = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.25 & 0 & 0.25 & 0.5 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

affinity matrix

row-normalized affinity matrix

After normalization, the matrix is not symmetric

1. example on weighted graph:



Input graph

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}$$

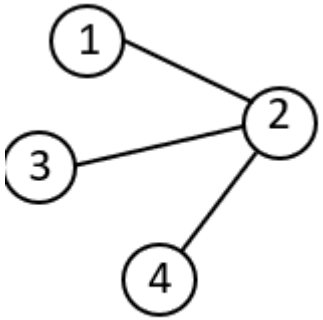
affinity matrix

$$A' = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.25 & 0 & 0.25 & 0.5 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

row-normalized affinity matrix

2. example on unweighted graph:

(treat the edge weight as 1 for all edges)



Input graph

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

affinity matrix

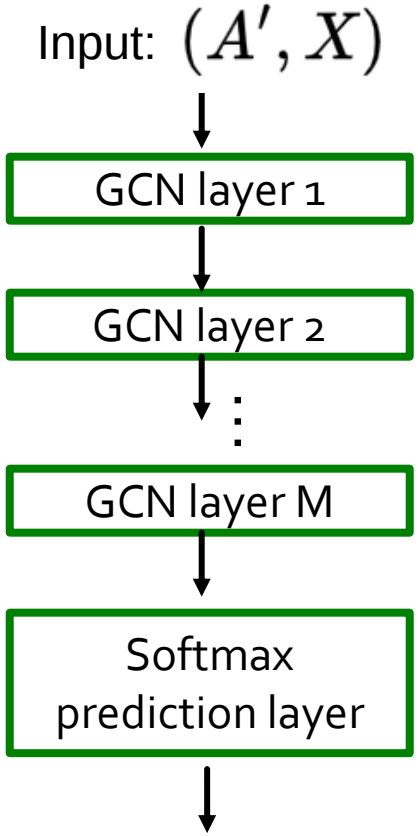
$$A' = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

row-normalized affinity matrix

GCN forward pass

- GCN is a stack of multiple GCN layers

An example for node classification tasks.



A' :Row-normalized affinity matrix (the sum of each row is 1)

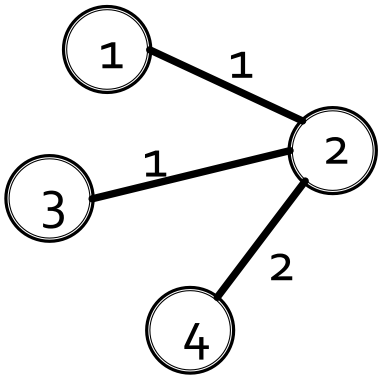
X : A matrix of all input node features. Each column indicate one node feature vector

p_i : The class score vector for node i
Each element indicates the prediction score for one class.
Assume there are C classes in total, the length of this score vector is C

GCN can be seen as a complicated mapping function:

$$P = f_{GCN}(A', X)$$

Example for the input (A', X)



Input graph

Calculate affinity matrix and row-normalized affinity matrix

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \qquad A' = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.25 & 0 & 0.25 & 0.5 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

X : A matrix of all input node features.

One column indicates the input feature vector for one node, denoted by x_i or $h_i^{(0)}$

The initial node features are given below as:

$$x_1 = [1, -1]^{\mathrm{T}},$$
$$x_2 = [0, 1]^{\mathrm{T}},$$
$$x_3 = [-1, 0]^{\mathrm{T}},$$
$$x_4 = [1, 0]^{\mathrm{T}}.$$

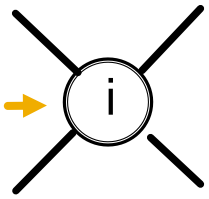
$$h_i^{(0)} = x_i$$

$$\begin{bmatrix} 1 & 0 & -1 & 1 \\ -1 & 1 & 0 & 0 \end{bmatrix}$$

$h_1^{(0)}$ $h_2^{(0)}$ $h_3^{(0)}$ $h_4^{(0)}$

GCN is applied to every node

The node i represents one node in the graph



The operation for one node:

Each GCN layer will update the feature vector for each node (non-linear transformation)

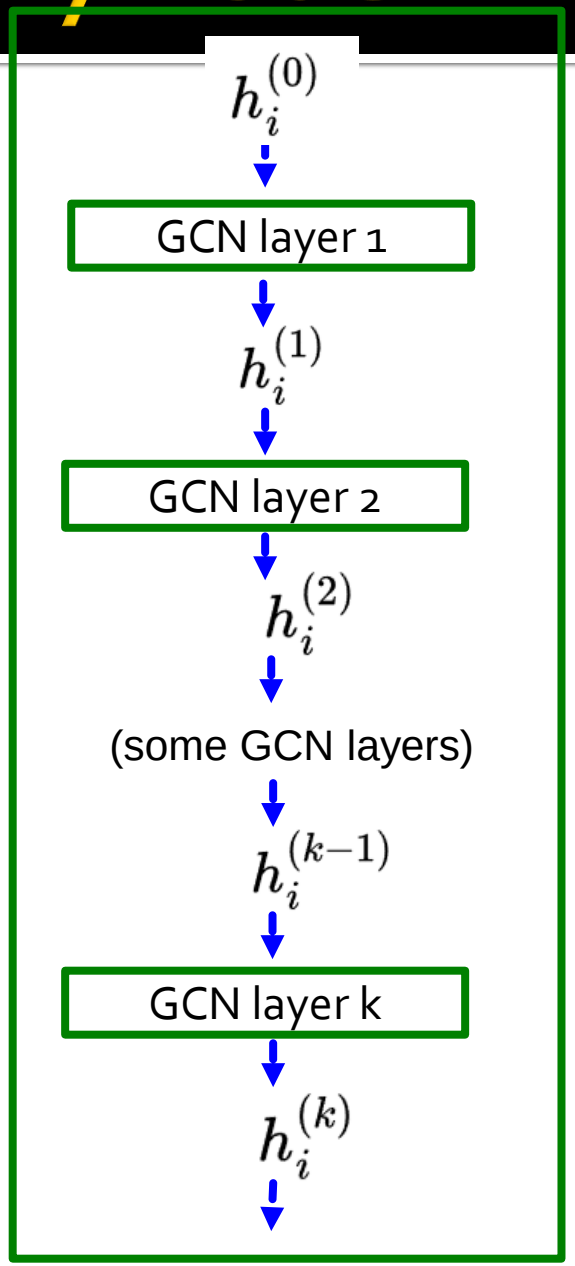
$h_i^{(0)}$: The input feature vector for node i (or called the initial feature vector)

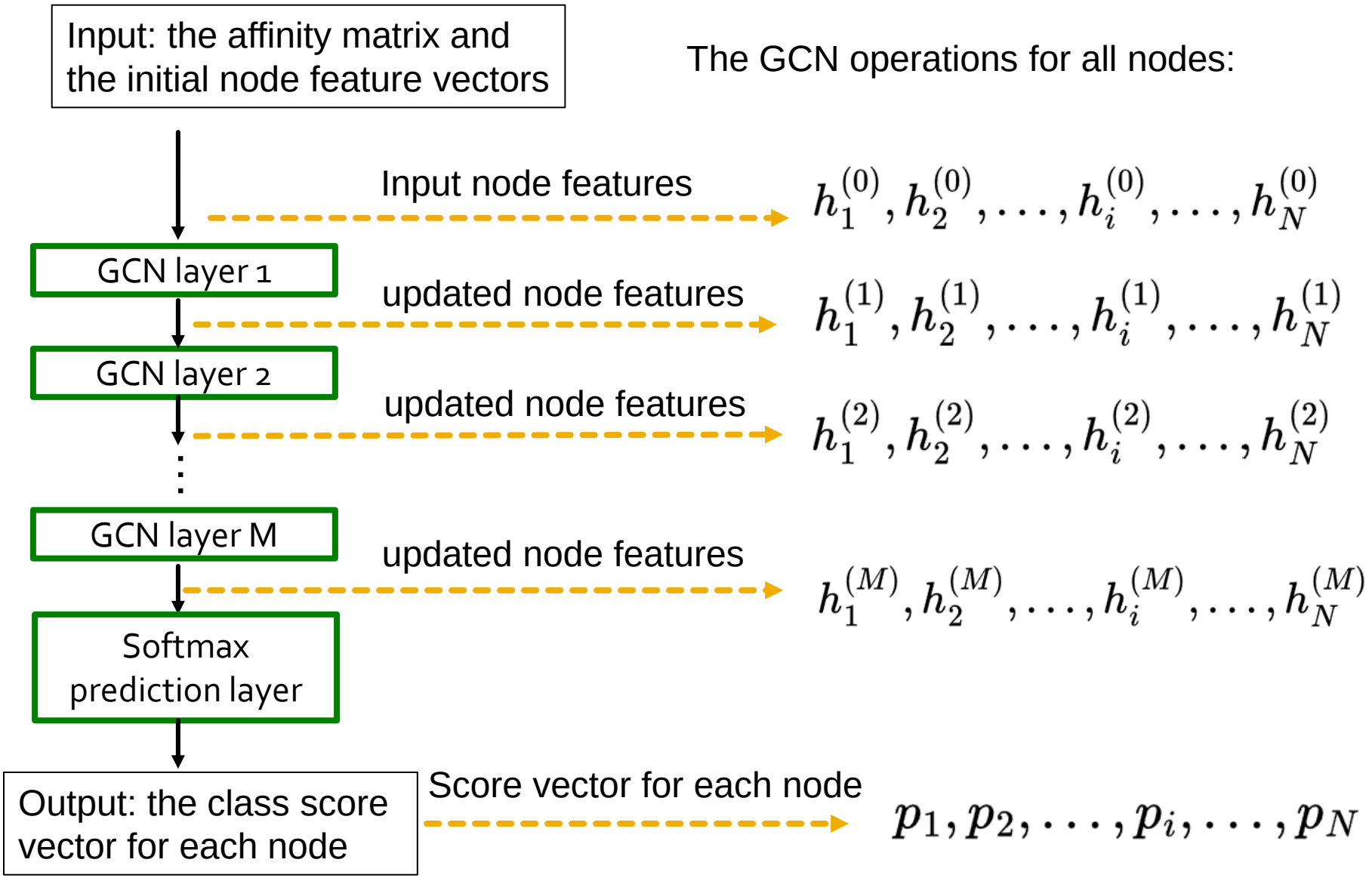
We have: $\boxed{h_i^{(0)} = x_i}$

x_i is the input feature vector for node i

$h_i^{(k-1)}$: the feature vector output by layer k-1

$h_i^{(k)}$: the feature vector output by layer k



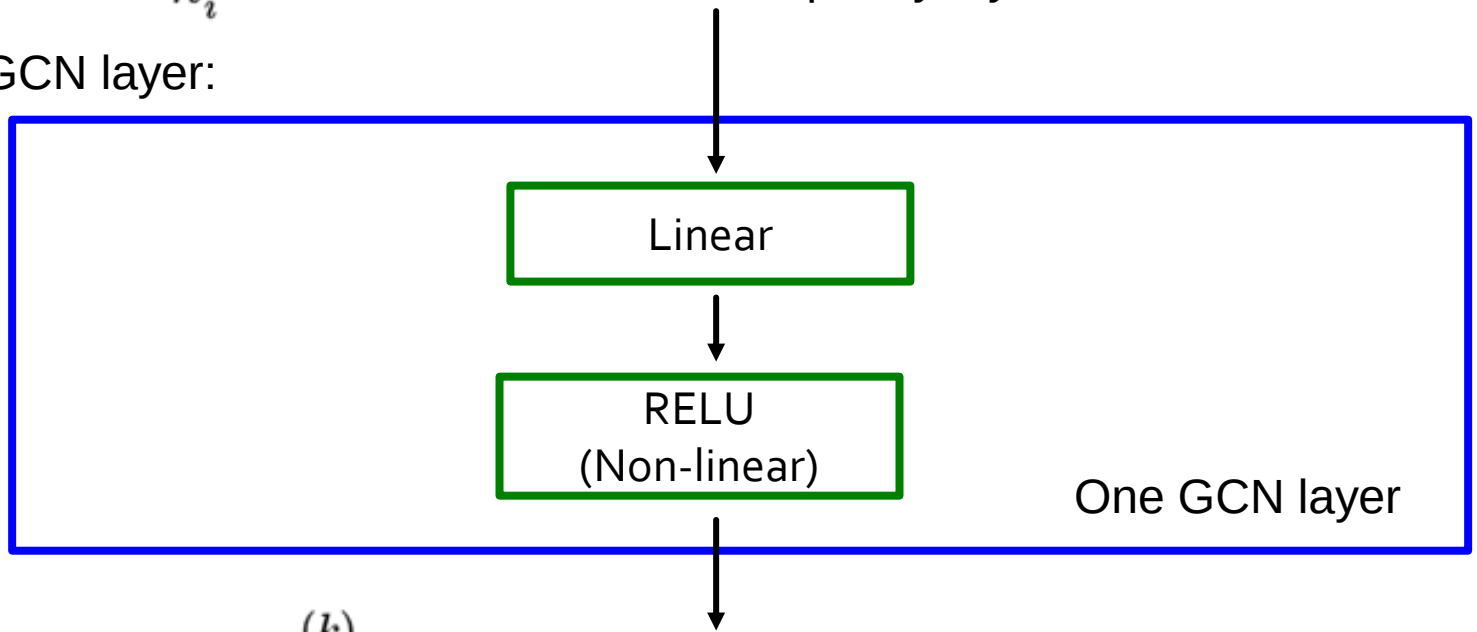

Input: the affinity matrix and the initial node feature vectors
The GCN operations for all nodes:
Input node features
$h_1^{(0)}, h_2^{(0)}, \ldots, h_i^{(0)}, \ldots, h_N^{(0)}$
GCN layer 1
updated node features
$h_1^{(1)}, h_2^{(1)}, \ldots, h_i^{(1)}, \ldots, h_N^{(1)}$
GCN layer 2
updated node features
$h_1^{(2)}, h_2^{(2)}, \ldots, h_i^{(2)}, \ldots, h_N^{(2)}$
GCN layer M
updated node features
$h_1^{(M)}, h_2^{(M)}, \ldots, h_i^{(M)}, \ldots, h_N^{(M)}$
Softmax prediction layer
Output: the class score vector for each node
Score vector for each node
$p_1, p_2, \ldots, p_i, \ldots, p_N$

One GCN layer

The calculation for one GCN layer:
Each GCN layer consist of two blocks (operations):
1. linear block; 2: Non-linear block

$h_i^{(k-1)}$: the feature vector output by layer k-1 for node i

The k-th GCN layer:



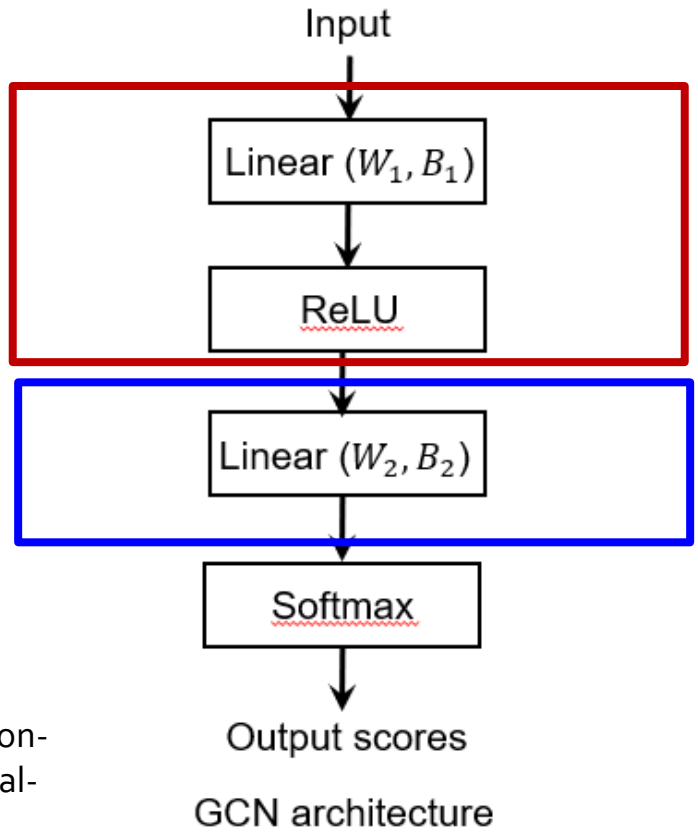
$h_i^{(k)}$: the feature vector output by layer k for node i

The output node feature vector is also called:
node representation or **node embedding**.

- GCN layer exception:
  In the last GCN layer, usually there is no non-linearity block.

- Example:

The last GCN layer,
There is no RELU operation
before Softmax



GCN architecture

https://stats.stackexchange.com/questions/163695/non-linearity-before-final-softmax-layer-in-a-convolutional-neural-network

One GCN layer

Linear transform:

$$\hat{h}_i^{(k)} = W_k \sum_{j \in \mathcal{N}_i} a'_{ij}\, h_j^{(k-1)} + B_k\, h_i^{(k-1)}$$

RELU (Non-linear transform):

$$h_i^{(k)} = \max\{0,\ \hat{h}_i^{(k)}\}$$

a'_{ij} : the element (i,j) in the row-normalized affinity matrix A';

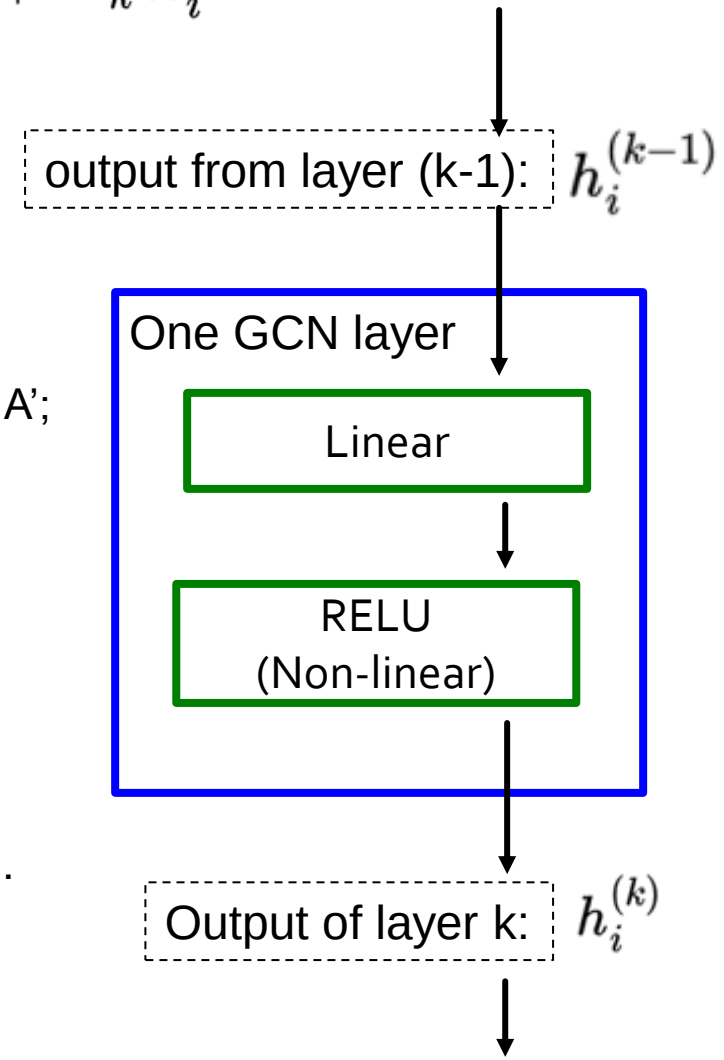
$\mathcal{N}_i$: the set of neighbours of node i (connected nodes)

$h_i^{(k-1)}$: input feature vector of node i for layer k

h_i^k : output feature vector of node i for layer k

W_k :Weight matrix for **neighbourhood aggregation**.

B_k :Weight matrix for **self transformation**



Linear transform:

$$\hat{h}_i^{(k)} = W_k \sum_{j \in \mathcal{N}_i} a'_{ij}\, h_j^{(k-1)} + B_k\, h_i^{(k-1)}$$

neighbourhood aggregation

self transformation

a'_{ij} : indicates the edge weight in the graph.
it is the element (i,j) in the row-normalized affinity matrix A';

W_k, B_k : are the GCN layer parameters. They are two matrixes. They are learnable parameters (or called layer weights).

In the exam or tutorial questions,
the values of W_k, B_k will be given in the questions.

Network training: the training of GCN is to solve an optimization problem to obtain the values of these layer parameters

Linear transform:

$$\hat{h}_i^{(k)} = W_k \sum_{j \in \mathcal{N}_i} a'_{ij}\, h_j^{(k-1)} + B_k\, h_i^{(k-1)}$$

neighbourhood aggregation

self transformation

1. Neighbourhood aggregation :

Aggregate the messages from the neighbours.

Neighbours here means directly connected nodes.

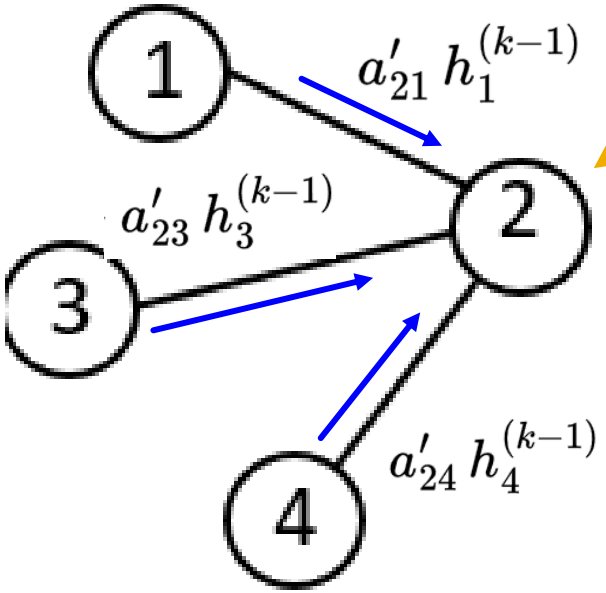
Linear transform:

$$\hat{h}_i^{(k)} = W_k \sum_{j \in \mathcal{N}_i} a'_{ij}\, h_j^{(k-1)} + B_k\, h_i^{(k-1)}$$

neighbourhood aggregation

self transformation

Aggregation calculation example:



Input graph

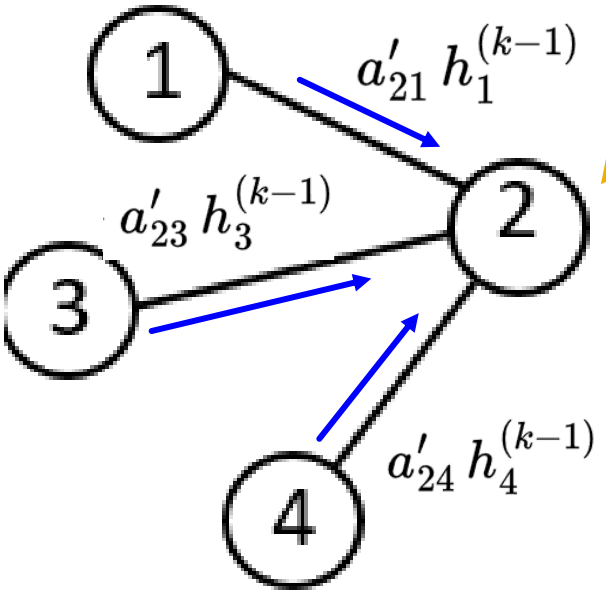
Target node

For node 2, the messages from the neighbors: node 1, node 3 and node 4 are aggregated:

$$\sum_{j \in \mathcal{N}_2} a'_{2j}\, h_j^{(k-1)} = a'_{21}\, h_1^{(k-1)} + a'_{23}\, h_3^{(k-1)} + a'_{24}\, h_4^{(k-1)}$$

Example: the message from node 1 to node 2

Aggregation example



Input graph

Target node

For node 2, the messages from the neighbors: node 1, node 3 and node 4 are aggregated:

$$\sum_{j \in \mathcal{N}_2} a'_{2j}\, h_j^{(k-1)} = a'_{21}\, h_1^{(k-1)} + a'_{23}\, h_3^{(k-1)} + a'_{24}\, h_4^{(k-1)}$$

a'_{ij} is the weight on the edge, which can be obtained from the row-normalized affinity matrix

$$A' = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

row-normalized affinity matrix

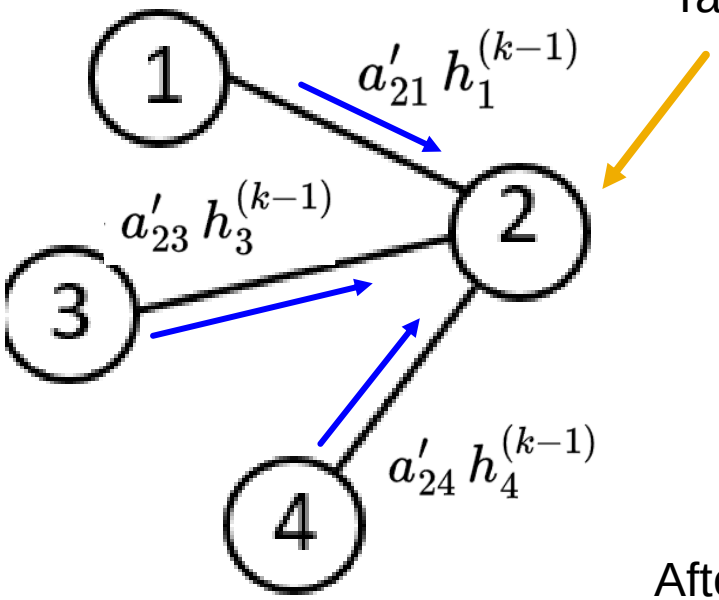
Aggregation example

Linear transform:

$$\hat{h}_i^{(k)} = W_k \sum_{j \in \mathcal{N}_i} a'_{ij}\, h_j^{(k-1)} + B_k\, h_i^{(k-1)}$$

neighbourhood aggregation

self transformation



Input graph

Target node

For node 2, the messages from the neighbors: node 1, node 3 and node 4 are aggregated

$$\sum_{j \in \mathcal{N}_2} a'_{2j}\, h_j^{(k-1)} = a'_{21}\, h_1^{(k-1)} + a'_{23}\, h_3^{(k-1)} + a'_{24}\, h_4^{(k-1)}$$

After aggregation, we perform transformation using W_k:

$$W_k \sum_{j \in \mathcal{N}_2} a'_{2j}\, h_j^{(k-1)}$$

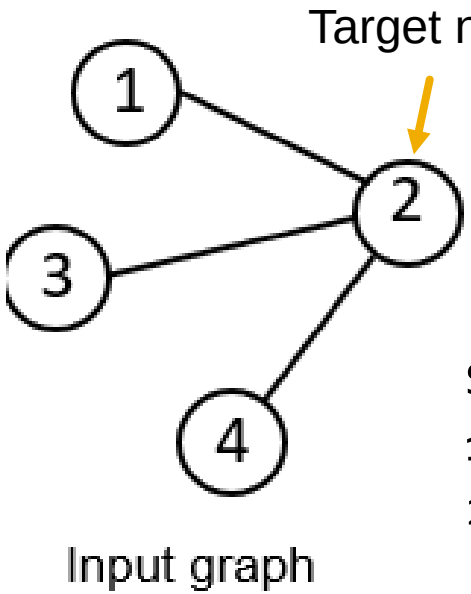
Self transformation example

Linear transform:

$$\hat{h}_i^{(k)} = W_k \sum_{j \in \mathcal{N}_i} a'_{ij}\, h_j^{(k-1)} + B_k\, h_i^{(k-1)}$$

neighbourhood aggregation

self transformation

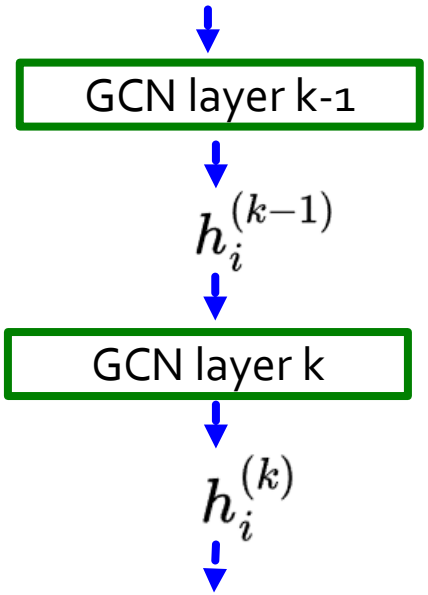


Input graph

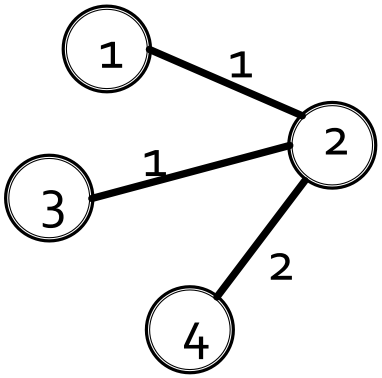
self transformation for node 2:

1. do not need to look at the neighbours
2. just use the feature vector of node 2 output by the previous layer (k-1)
3. Use the B_k matrix for transformation:

$$B_k h_2^{(k-1)}$$

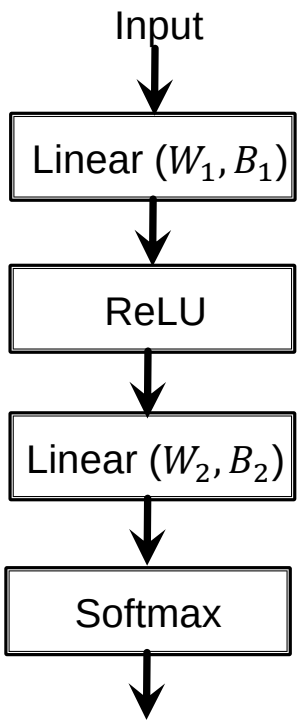


- Examples of W and B matrix:



Input graph

$$x_1 = [1, -1]^{\mathrm{T}},$$
$$x_2 = [0, 1]^{\mathrm{T}},$$
$$x_3 = [-1, 0]^{\mathrm{T}},$$
$$x_4 = [1, 0]^{\mathrm{T}}.$$



GCN architecture

$$W_1 = \begin{bmatrix} -0.2 & 0.1 \\ 0.0 & -0.1 \\ 0.1 & -0.1 \end{bmatrix}$$

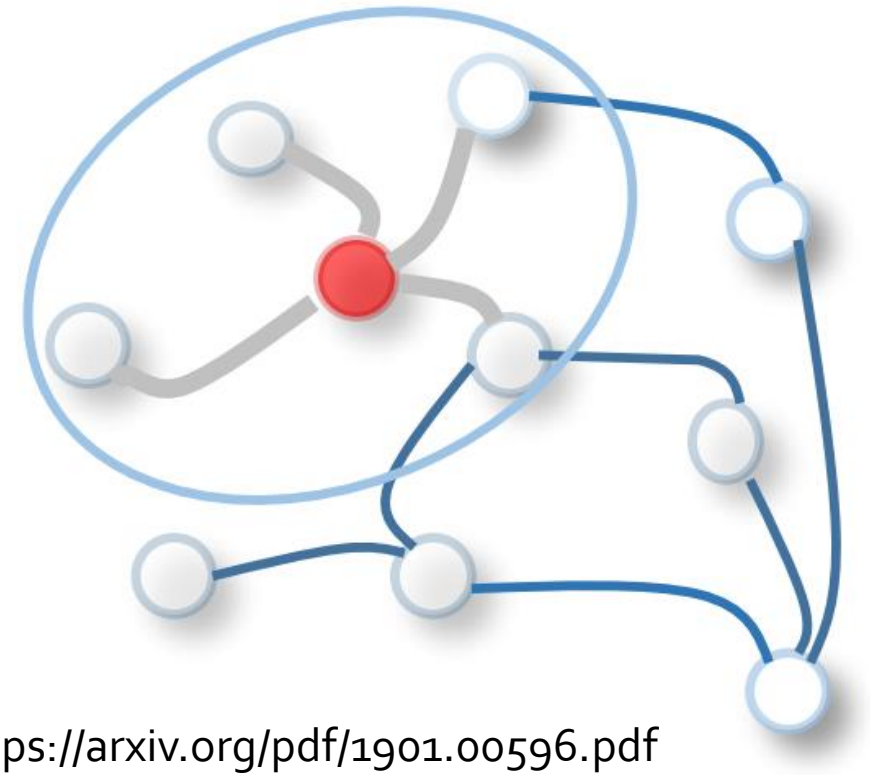
$$B_1 = \begin{bmatrix} -0.1 & -0.1 \\ 0.1 & 0.0 \\ 0.0 & 0.1 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} 0.1 & 0.0 & -0.1 \\ 0.1 & 0.2 & -0.1 \end{bmatrix}$$

$$B_2 = \begin{bmatrix} 0.0 & 0.1 & 0.1 \\ 0.1 & -0.1 & -0.1 \end{bmatrix}$$

Different layers have different W, B matrix

- Key idea of GCN:
  - Aggregate the information (messages) from the neighbours to generate the features for one node.



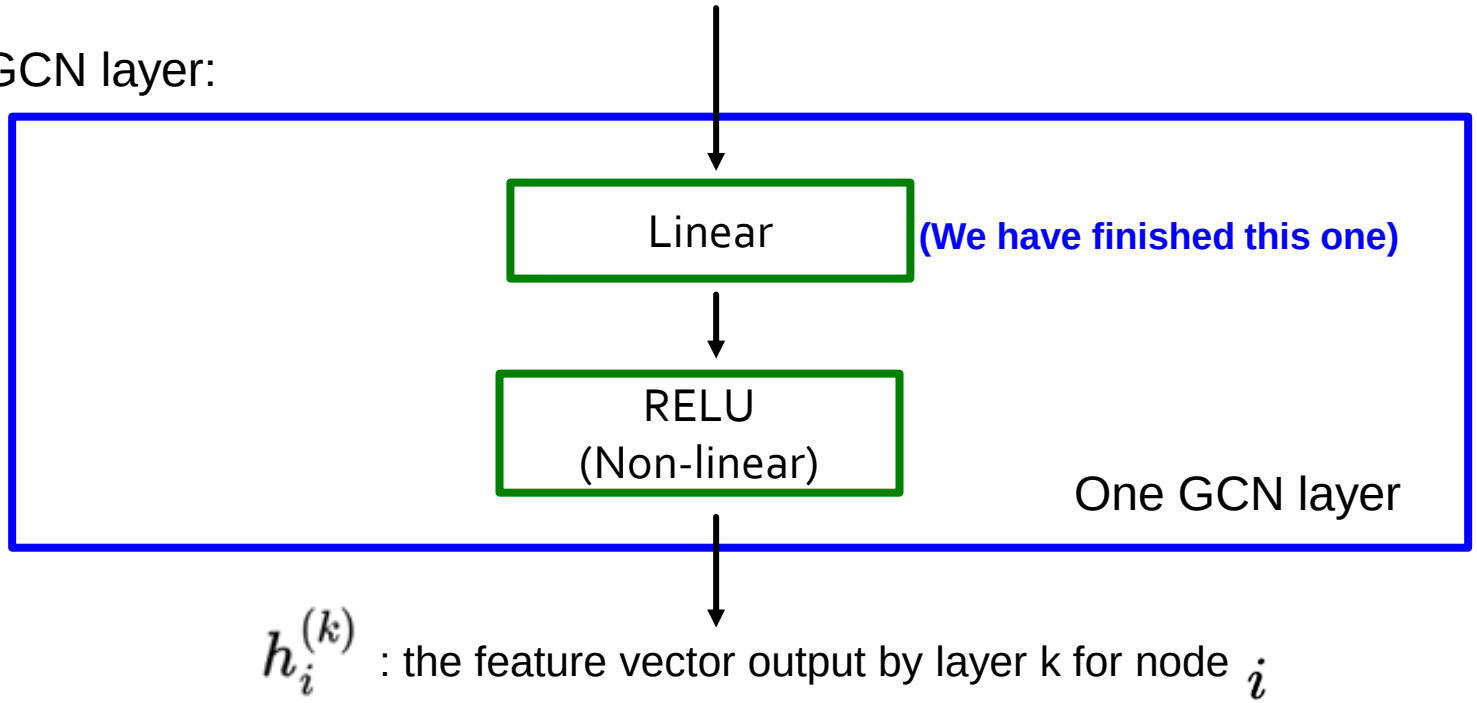
https://arxiv.org/pdf/1901.00596.pdf

One GCN layer

The calculation for one GCN layer:
Each GCN layer consist of two blocks (operations):
1. linear block; 2: Non-linear block

$h_i^{(k-1)}$: the feature vector output by layer k-1 for node i

The k-th GCN layer:



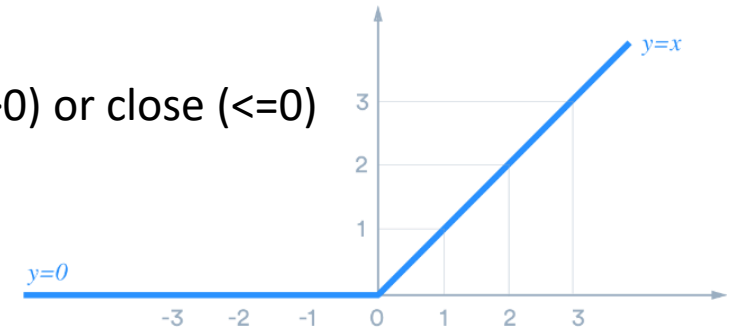
$h_i^{(k)}$: the feature vector output by layer k for node i

The output node feature vector is also called:
node representation or **node embedding**.

- Non-linearity functions
  - Also called activation functions
  - Like a “gate” or a tiny classifier: open (>0) or close (<=0)

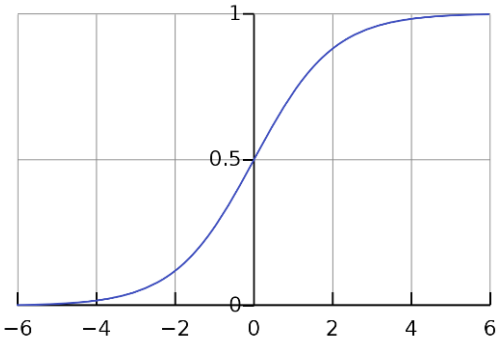
1. Rectified linear unit (RELU)

$$ReLU(x) = \max(x, 0)$$



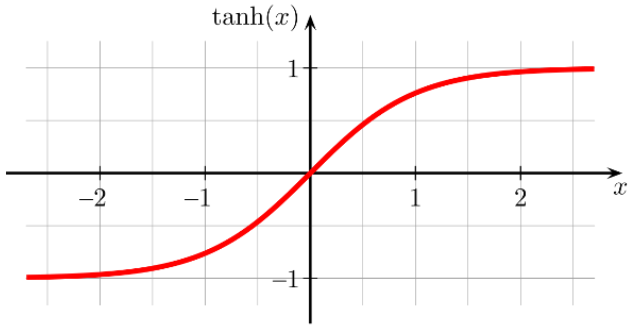
2. Sigmoid

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^{x}}{e^{x} + 1}$$



3. Hyperbolic tangent

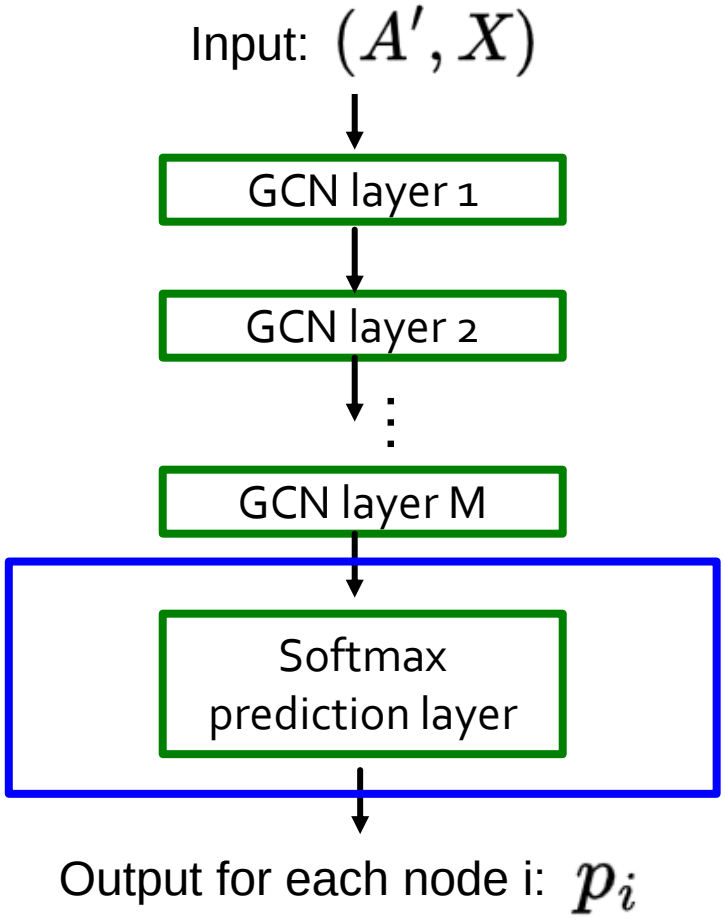
$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$



GCN forward pass

- GCN is a stack of multiple GCN layers

An example for node classification tasks.



p_i : The class score vector for node i
Each element indicates the prediction score for one class.
Assume there are C classes in total, the length of this score vector is C

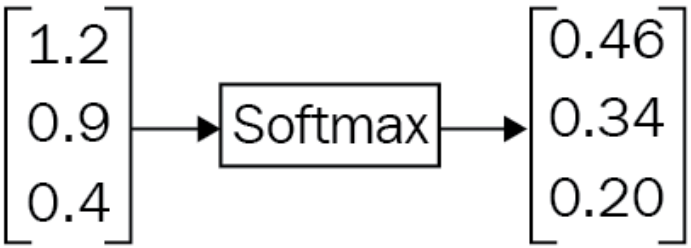
Example for 3-class classification:
p1= [0.1 0.1 0.8],
p2= [0.2 0.3 0.5],
….

(We now come to this one)

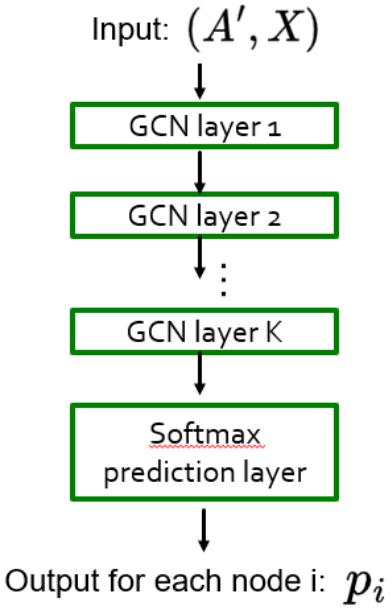
- Softmax layer (prediction layer)
  - The output values lie in [0 1]
    - Can be used as a normalization layer
  - Turn the input vector into a probability output vector
    - Sum of all elements is 1. Each element >=0.
    - The output is class probability scores

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}$$

Example (3 classes):
The output is a 3-dimensional score vector for each node

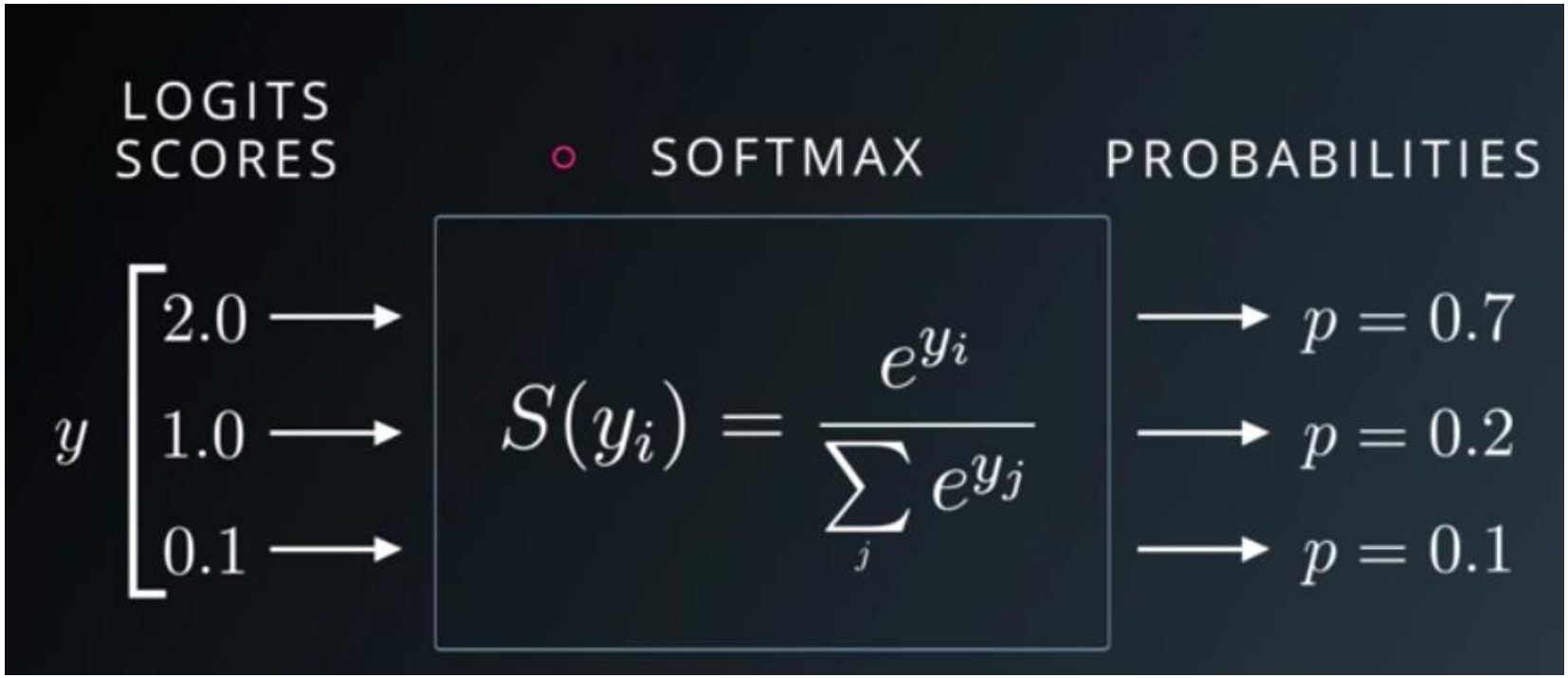


Recall: GCN for classification:



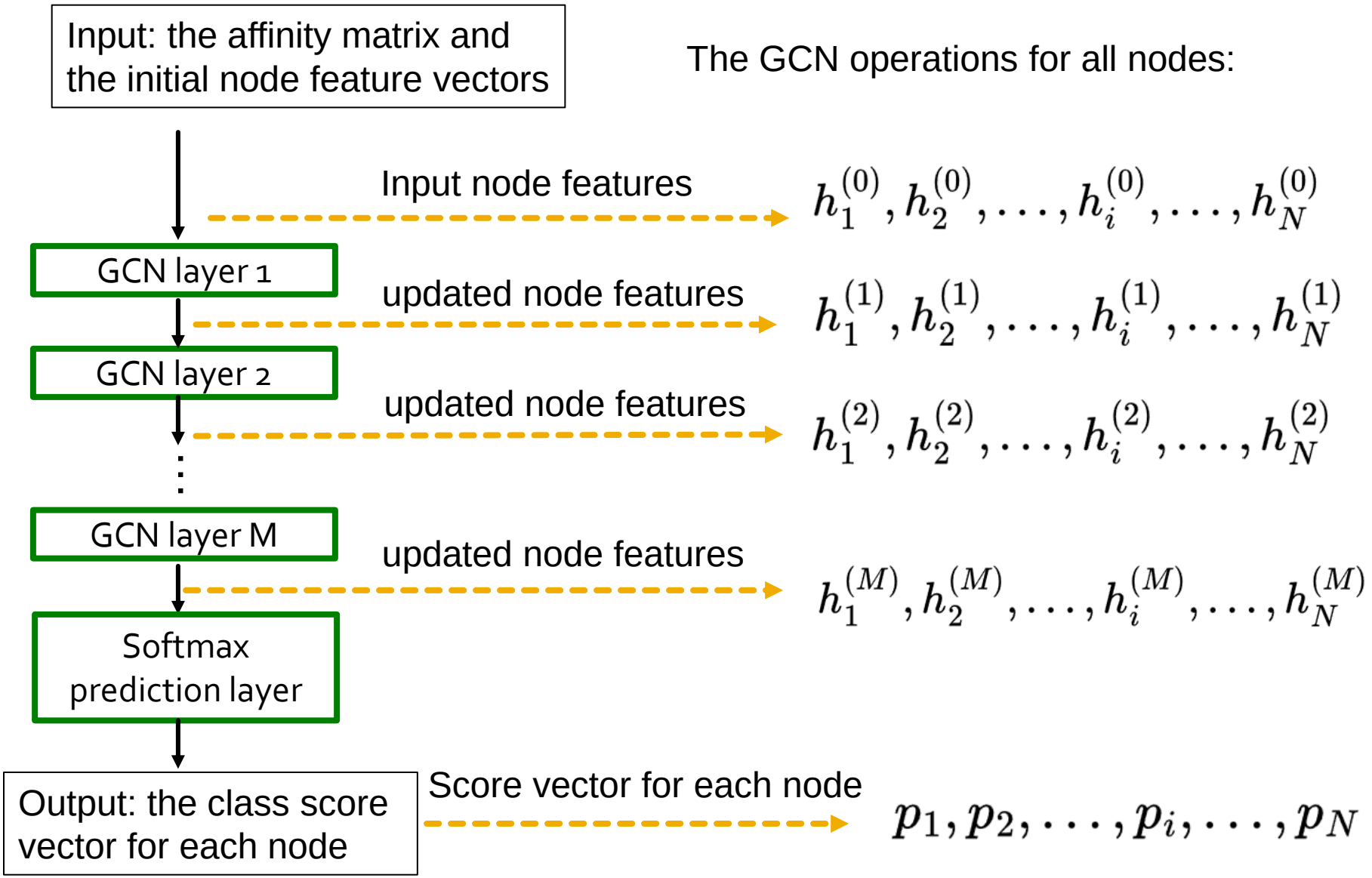
Another example:

The input vector before using the Softmax function for class prediction is called logits.



https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d

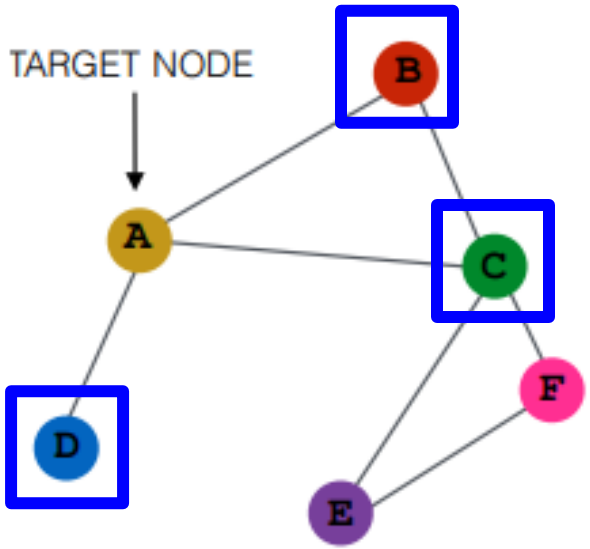
Recap of GCN for node classification



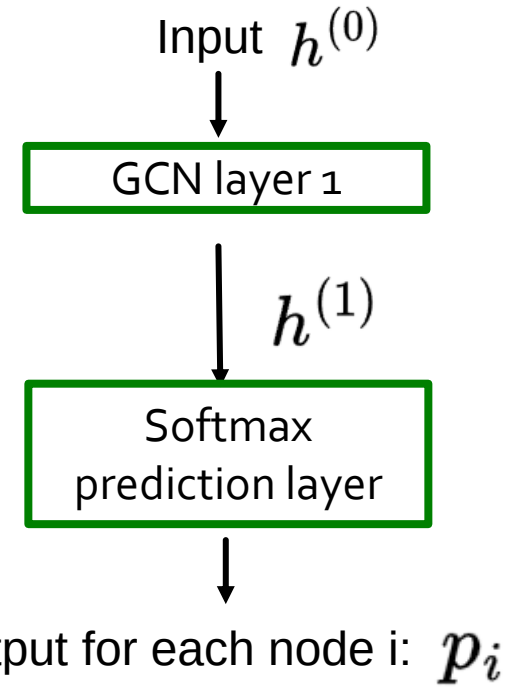
Discussion

If we only have 1 layer of GCN,
we only perform local message passing

$$\hat{h}_i^{(k)} = W_k \sum_{j \in \mathcal{N}_i} a'_{ij}\, h_j^{(k-1)} + B_k\, h_i^{(k-1)}$$



INPUT GRAPH

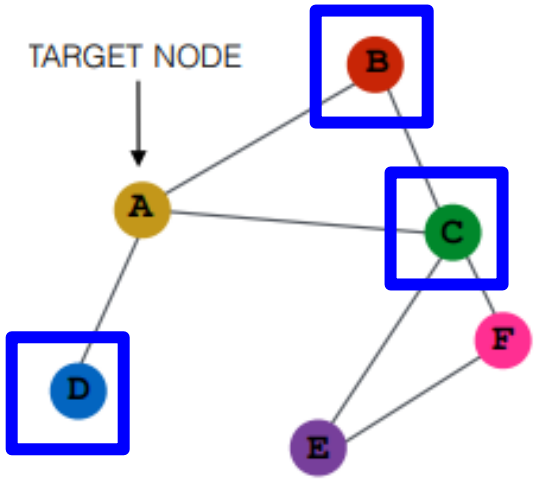


http://web.stanford.edu/class/cs224w/slides/06-GNN1.pdf

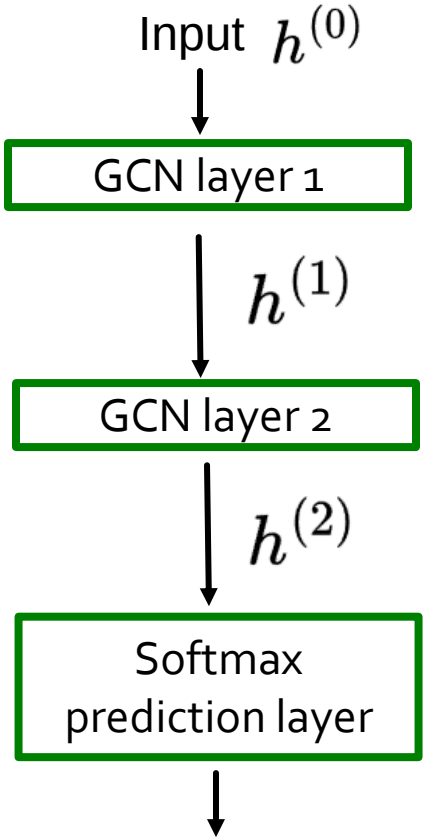
Discussion

How about 2 GCN layers?

$$\hat{h}_i^{(k)} = W_k \sum_{j \in \mathcal{N}_i} a'_{ij}\, h_j^{(k-1)} + B_k\, h_i^{(k-1)}$$



INPUT GRAPH



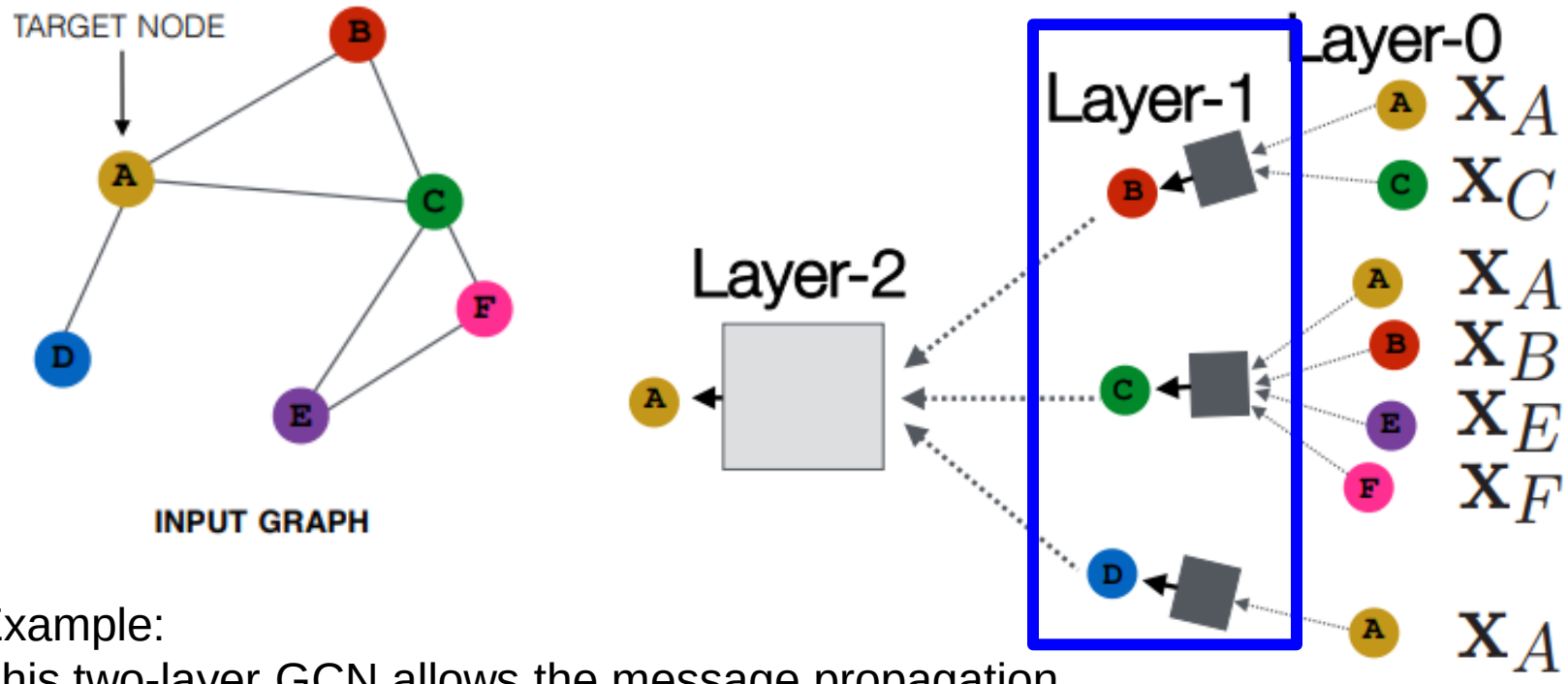
Output for each node i: p_i

Messages will be propagated from distant nodes to the target node by multiple-layer neighbourhood propagation.

$h^{(2)}$ Will encode information from distant nodes

Discussion

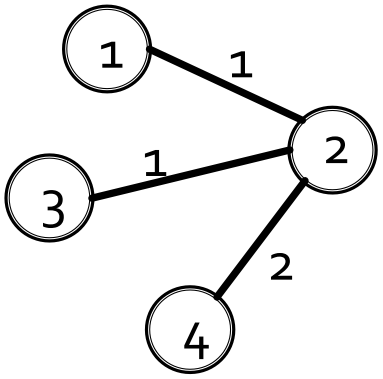
Message passing is not local in a multi-layer GCN.
Messages will be propagated from distant nodes to the target node by multiple-layer neighbourhood propagation.



Example:
This two-layer GCN allows the message propagation from any other nodes in the graph to the target node A

http://web.stanford.edu/class/cs224w/slides/06-GNN1.pdf

GCN Forward Pass example



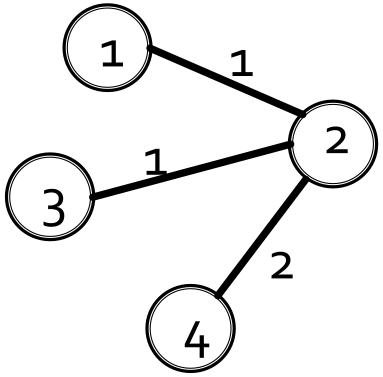
Input graph

Question:

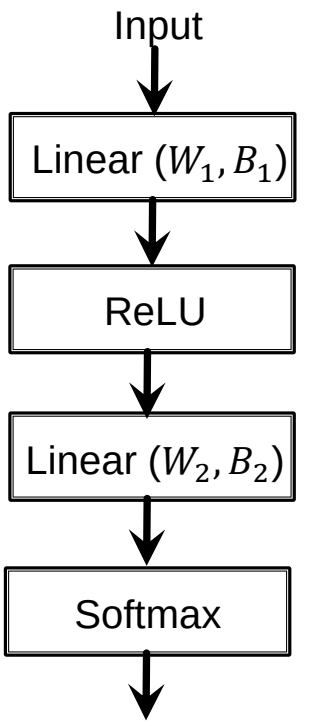
Given a graph below, the task is to do node-wise classification using a 2-layer graph convolutional network (GCN) and a cross-entropy loss, with network architecture shown. The initial node features are given below as:

$$x_1 = [1, -1]^T, x_2 = [0,1]^T,$$
$$x_3 = [-1,0]^T, x_4 = [1,0]^T.$$

Question: (continued from last page)



Input graph



GCN architecture

With the initial GCN weight parameters given below (W_k and B_k are the weight matrices for neighborhood aggregation and self transformation, respectively, for the k-th layer). Calculate the prediction of **node 2** by performing one forward pass.

$$W_1 = \begin{bmatrix} -0.2 & 0.1 \\ 0.0 & -0.1 \\ 0.1 & -0.1 \end{bmatrix} \quad B_1 = \begin{bmatrix} -0.1 & -0.1 \\ 0.1 & 0.0 \\ 0.0 & 0.1 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} 0.1 & 0.0 & -0.1 \\ 0.1 & 0.2 & -0.1 \end{bmatrix}$$

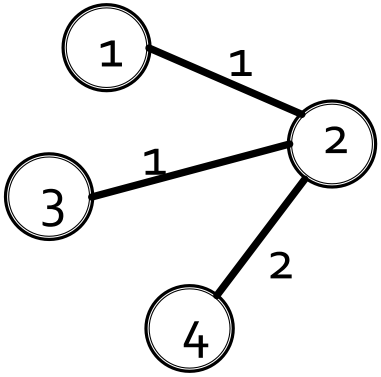
$$B_2 = \begin{bmatrix} 0.0 & 0.1 & 0.1 \\ 0.1 & -0.1 & -0.1 \end{bmatrix}$$

1st GCN layer

GCN Forward Pass Example

Solution:

Calculate affinity matrix and row-normalized affinity matrix



Input graph

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \qquad A' = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.25 & 0 & 0.25 & 0.5 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

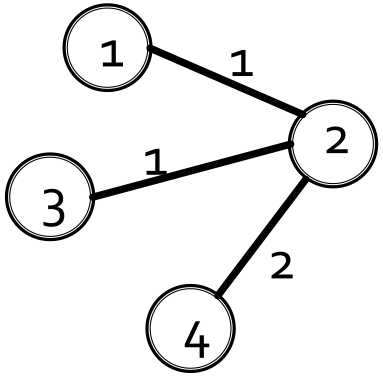
Linear transform:

$$\hat{h}_i^{(k)} = W_k \sum_{j \in \mathcal{N}_i} a'_{ij} \, h_j^{(k-1)} + B_k \, h_i^{(k-1)}$$

Non-linear transform (ReLU):

$$h_i^{(k)} = max\{0, \, \hat{h}_i^{(k)}\}$$

Forward pass for node 1



Input graph

We have $h_i^{(0)} = x_i$

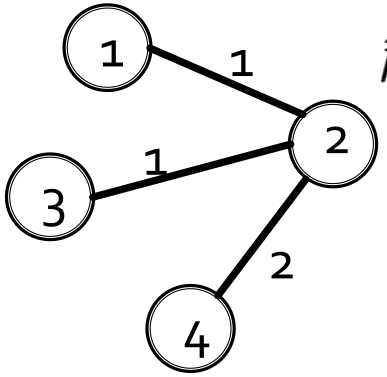
Linear transform:

$$\sum_{j \in \mathcal{N}_1} a'_{1j}\, h_j^{(0)} = a'_{12}\, h_2^{(0)} = 1 \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$W_1 \sum_{j \in \mathcal{N}_1} a'_{1j}\, h_j^{(0)} = \begin{bmatrix} -0.2 & 0.1 \\ 0.0 & -0.1 \\ 0.1 & -0.1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.1 \\ -0.1 \\ -0.1 \end{bmatrix}$$

$$B_1\, h_1^{(0)} = \begin{bmatrix} -0.1 & -0.1 \\ 0.1 & 0.0 \\ 0.0 & 0.1 \end{bmatrix} \times \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.1 \\ -0.1 \end{bmatrix}$$

Forward pass for node 1



Input graph

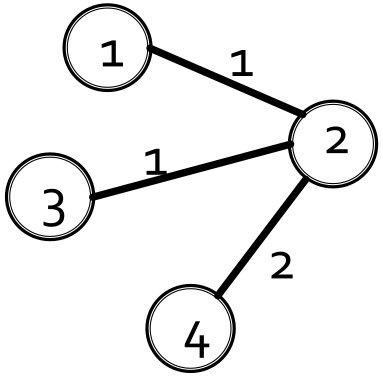
Linear transform:

$$\hat{h}_1^{(1)} = W_1 \sum_{j \in \mathcal{N}_1} a'_{1j}\, h_j^{(0)} + B_1\, h_1^{(0)} = \begin{bmatrix} 0.1 \\ -0.1 \\ -0.1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.1 \\ -0.1 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.0 \\ -0.2 \end{bmatrix}$$

Non-linear transform (ReLU):

$$h_1^{(1)} = max\{0,\ \hat{h}_1^{(1)}\} = \begin{bmatrix} 0.1 \\ 0.0 \\ 0.0 \end{bmatrix}$$

Forward pass for node 2



Input graph

We have $h_i^{(0)} = x_i$

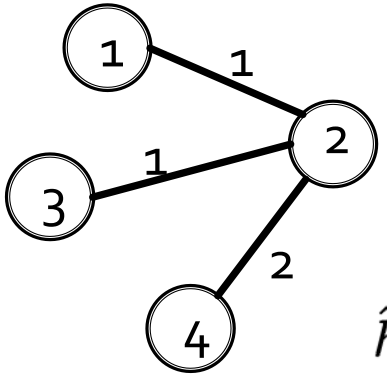
$$\sum_{j \in \mathcal{N}_2} a'_{2j}\, h_j^{(0)} = a'_{21}\, h_1^{(0)} + a'_{23}\, h_3^{(0)} + a'_{24}\, h_4^{(0)}$$

$$= 0.25 \times \begin{bmatrix} 1 \\ -1 \end{bmatrix} + 0.25 \times \begin{bmatrix} -1 \\ 0 \end{bmatrix} + 0.5 \times \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 \\ -0.25 \end{bmatrix}$$

$$W_1 \sum_{j \in \mathcal{N}_2} a'_{2j}\, h_j^{(0)} = \begin{bmatrix} -0.2 & 0.1 \\ 0.0 & -0.1 \\ 0.1 & -0.1 \end{bmatrix} \times \begin{bmatrix} 0.5 \\ -0.25 \end{bmatrix} = \begin{bmatrix} -0.125 \\ 0.025 \\ 0.075 \end{bmatrix}$$

Forward pass for node 2



Input graph

$$B_1\, h_2^{(0)} \quad = \quad \begin{bmatrix} -0.1 & -0.1 \\ 0.1 & 0.0 \\ 0.0 & 0.1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad = \quad \begin{bmatrix} -0.1 \\ 0.0 \\ 0.1 \end{bmatrix}$$

Linear transform:

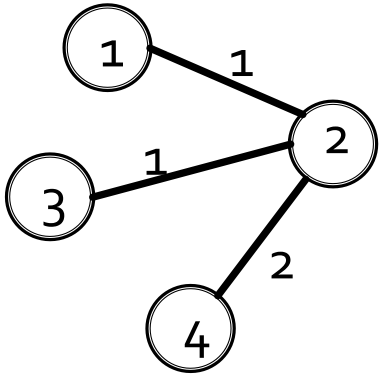
$$\hat{h}_2^{(1)} = W_1 \sum_{j \in \mathcal{N}_2} a'_{2j}\, h_j^{(0)} + B_1\, h_2^{(0)} = \begin{bmatrix} -0.125 \\ 0.025 \\ 0.075 \end{bmatrix} + \begin{bmatrix} -0.1 \\ 0.0 \\ 0.1 \end{bmatrix} = \begin{bmatrix} -0.225 \\ 0.025 \\ 0.175 \end{bmatrix}$$

Non-linear transform (ReLU):

$$h_2^{(1)} \quad = \quad max\{0,\ \hat{h}_2^{(1)}\} \quad = \quad \begin{bmatrix} 0.0 \\ 0.025 \\ 0.175 \end{bmatrix}$$

Forward pass for node 3

We have $h_i^{(0)} = x_i$



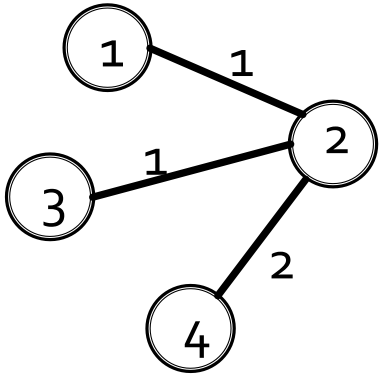
Input graph

$$\sum_{j \in \mathcal{N}_3} a'_{3j}\, h_j^{(0)} = a'_{32}\, h_2^{(0)} = 1 \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$W_1 \sum_{j \in \mathcal{N}_3} a'_{3j}\, h_j^{(0)} = \begin{bmatrix} -0.2 & 0.1 \\ 0.0 & -0.1 \\ 0.1 & -0.1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.1 \\ -0.1 \\ -0.1 \end{bmatrix}$$

$$B_1\, h_3^{(0)} = \begin{bmatrix} -0.1 & -0.1 \\ 0.1 & 0.0 \\ 0.0 & 0.1 \end{bmatrix} \times \begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.1 \\ -0.1 \\ 0.0 \end{bmatrix}$$

Forward pass for node 3



Input graph

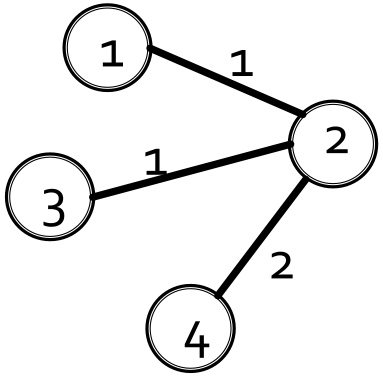
Linear transform:

$$\hat{h}_3^{(1)} = W_1 \sum_{j \in \mathcal{N}_3} a'_{3j}\, h_j^{(0)} + B_1\, h_3^{(0)} = \begin{bmatrix} 0.1 \\ -0.1 \\ -0.1 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.1 \\ 0.0 \end{bmatrix} = \begin{bmatrix} 0.2 \\ -0.2 \\ -0.1 \end{bmatrix}$$

Non-linear transform (ReLU):

$$h_3^{(1)} = max\{0,\ \hat{h}_3^{(1)}\} = \begin{bmatrix} 0.2 \\ 0.0 \\ 0.0 \end{bmatrix}$$

Forward pass for node 4



Input graph

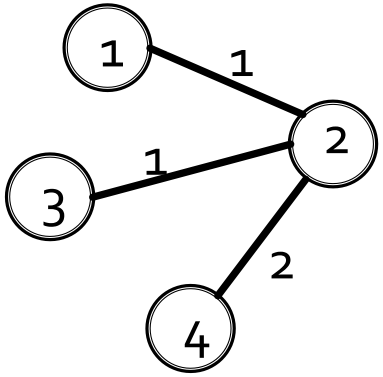
We have $h_i^{(0)} = x_i$

$$\sum_{j \in \mathcal{N}_4} a'_{4j}\, h_j^{(0)} = a'_{42}\, h_2^{(0)} = 1 \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$W_1 \sum_{j \in \mathcal{N}_4} a'_{4j}\, h_j^{(0)} = \begin{bmatrix} -0.2 & 0.1 \\ 0.0 & -0.1 \\ 0.1 & -0.1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.1 \\ -0.1 \\ -0.1 \end{bmatrix}$$

$$B_1\, h_4^{(0)} = \begin{bmatrix} -0.1 & -0.1 \\ 0.1 & 0.0 \\ 0.0 & 0.1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.1 \\ 0.1 \\ 0.0 \end{bmatrix}$$

Forward pass for node 4



Input graph

Linear transform:

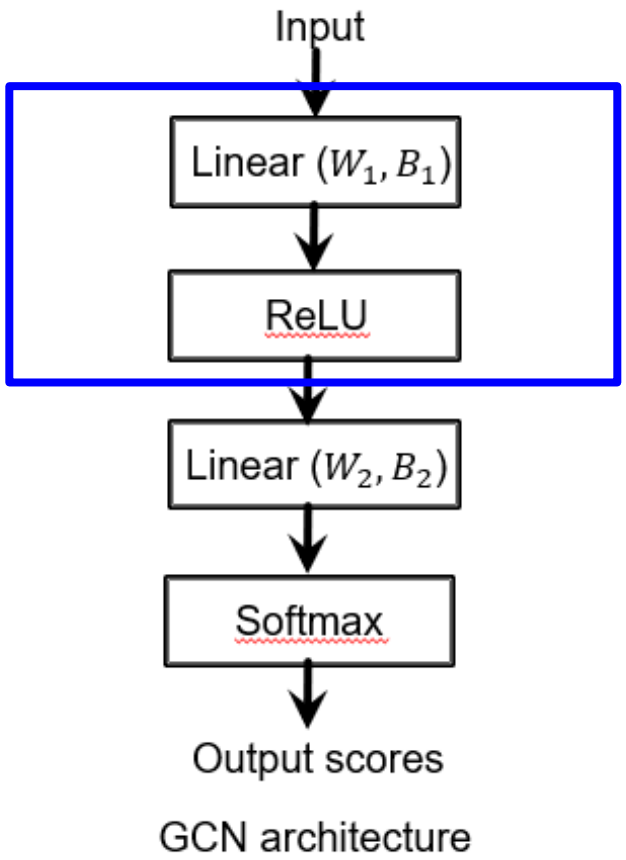
$$\hat{h}_4^{(1)} = W_1 \sum_{j \in \mathcal{N}_4} a'_{4j}\, h_j^{(0)} + B_1\, h_4^{(0)} = \begin{bmatrix} 0.1 \\ -0.1 \\ -0.1 \end{bmatrix} + \begin{bmatrix} -0.1 \\ 0.1 \\ 0.0 \end{bmatrix} = \begin{bmatrix} 0.0 \\ 0.0 \\ -0.1 \end{bmatrix}$$

Non-linear transform (ReLU):

$$h_4^{(1)} = max\{0,\ \hat{h}_4^{(1)}\} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

Summary (1st GCN layer)

We have finished the 1st GCN layer



GCN architecture

Node features output by the 1st GCN layer:

$$h_1^{(1)} = max\{0,\ \hat{h}_1^{(1)}\} = \begin{bmatrix} 0.1 \\ 0.0 \\ 0.0 \end{bmatrix}$$

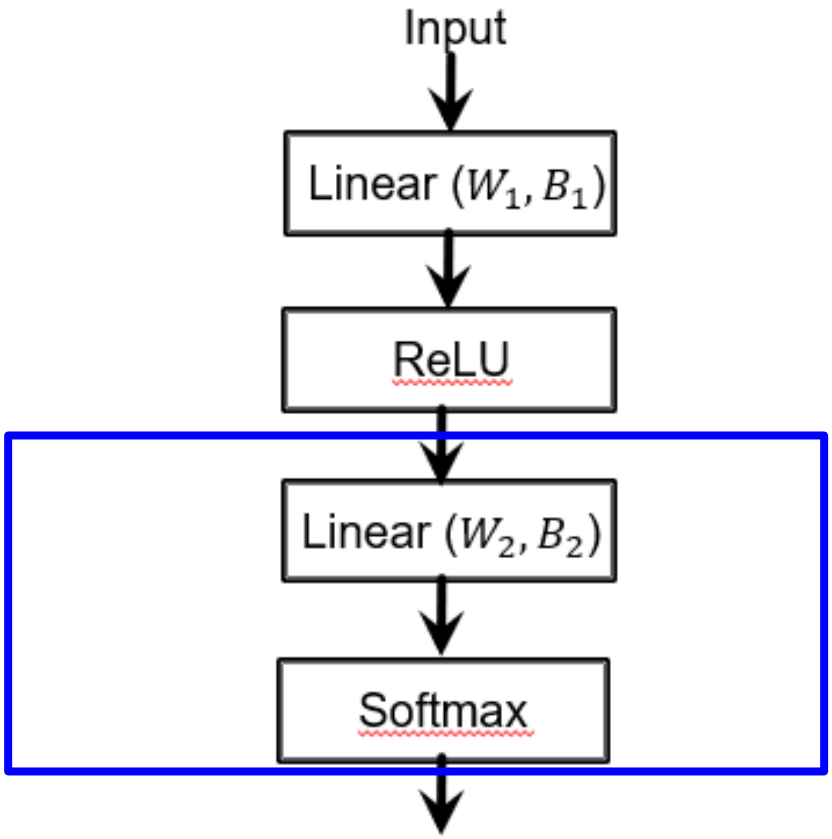
$$h_2^{(1)} = max\{0,\ \hat{h}_2^{(1)}\} = \begin{bmatrix} 0.0 \\ 0.025 \\ 0.175 \end{bmatrix}$$

$$h_3^{(1)} = max\{0,\ \hat{h}_3^{(1)}\} = \begin{bmatrix} 0.2 \\ 0.0 \\ 0.0 \end{bmatrix}$$

$$h_4^{(1)} = max\{0,\ \hat{h}_4^{(1)}\} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

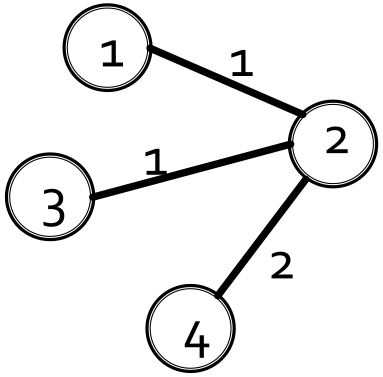
2nd GCN layer

Next step: the 2nd GCN layer and Softmax prediction



GCN architecture

Forward pass (2nd GCN layer) for node 2



Input graph

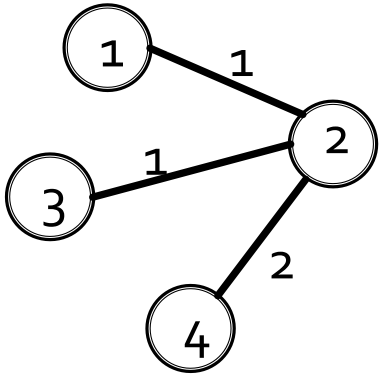
$$\sum_{j \in \mathcal{N}_2} a'_{2j}\, h_j^{(1)} = a'_{21}\, h_1^{(1)} + a'_{23}\, h_3^{(1)} + a'_{24}\, h_4^{(1)}$$

$$= 0.25 \times \begin{bmatrix} 0.1 \\ 0.0 \\ 0.0 \end{bmatrix} + 0.25 \times \begin{bmatrix} 0.2 \\ 0.0 \\ 0.0 \end{bmatrix} + 0.5 \times \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.075 \\ 0.0 \\ 0.0 \end{bmatrix}$$

$$W_2 \sum_{j \in \mathcal{N}_2} a'_{2j}\, h_j^{(1)} = \begin{bmatrix} 0.1 & 0.0 & -0.1 \\ 0.1 & 0.2 & -0.1 \end{bmatrix} \times \begin{bmatrix} 0.075 \\ 0.0 \\ 0.0 \end{bmatrix} = \begin{bmatrix} 0.0075 \\ 0.0075 \end{bmatrix}$$

Forward pass (2nd GCN Layer) for node 2



Input graph

$$B_2\, h_2^{(1)} = \begin{bmatrix} 0.0 & 0.1 & 0.1 \\ 0.1 & -0.1 & -0.1 \end{bmatrix} \times \begin{bmatrix} 0.0 \\ 0.025 \\ 0.175 \end{bmatrix} = \begin{bmatrix} 0.02 \\ -0.02 \end{bmatrix}$$

Linear transform:

$$\hat{h}_2^{(2)} = W_2 \sum_{j \in \mathcal{N}_2} a'_{2j}\, h_j^{(1)} + B_2\, h_2^{(1)} = \begin{bmatrix} 0.0075 \\ 0.0075 \end{bmatrix} + \begin{bmatrix} 0.02 \\ -0.02 \end{bmatrix} = \begin{bmatrix} 0.0275 \\ -0.0125 \end{bmatrix}$$

Softmax:

$$h_2^{(2)} = Softmax(\hat{h}_2^{(2)}) = Softmax(\begin{bmatrix} 0.0275 \\ -0.0125 \end{bmatrix}) = \begin{bmatrix} 0.51 \\ 0.49 \end{bmatrix}$$

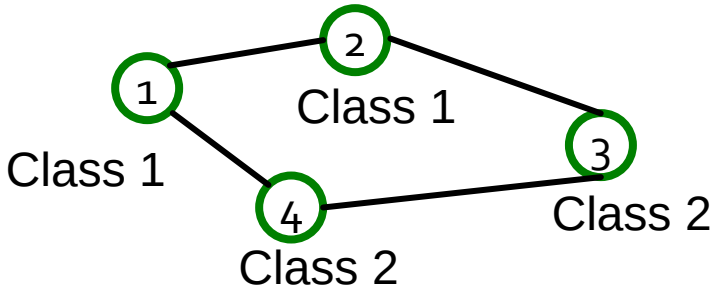
$$e^{z1} + e^{z2} = e^{0.0275} + e^{-0.0125} = 1.0279 + 0.9876 = 2.0155$$

GCN discussions

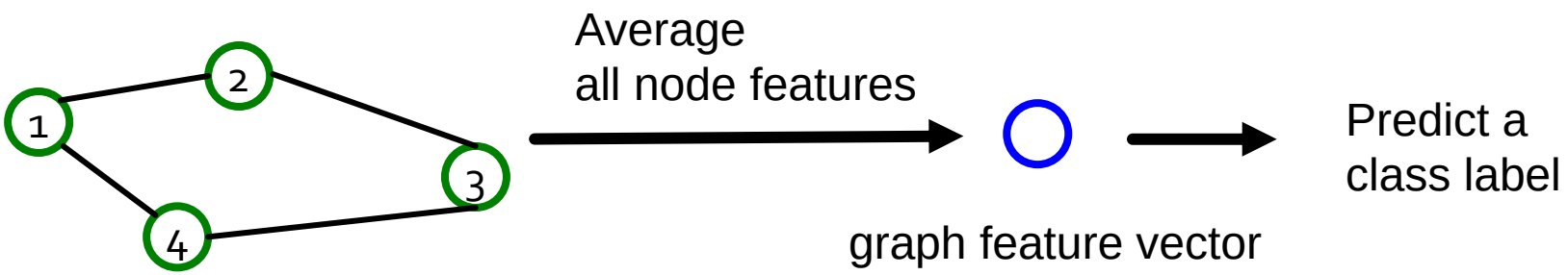
- Input feature vectors for GCN:
  - The input node feature vector depends on applications:
    - Web: web page descriptions (extra text features);
    - Social networks: User profile (text features) + User images (image features).
    - 3D recognition: 3D coordinate (x, y, z) for each 3D point
    - General case:
      Using one-hot feature vectors (or called indicator vectors).
      - E.g., given 4 nodes in total, their on-hot feature vectors can be:
        $x1=[1, 0, 0, 0]^T$;
        $x2=[0, 1, 0, 0]^T$;
        $x3=[0, 0, 1, 0]^T$;
        $x4=[0, 0, 0, 1]^T$;

A one-hot vector:
encode the identity of one input node

- Node classification
  - Predict a class label for each node in a graph

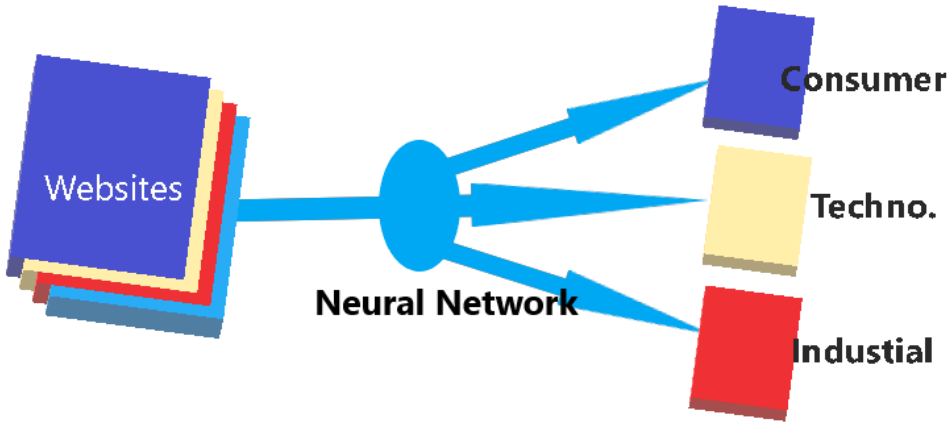


- Graph classification
  - Predict a class label for the whole graph



GCN applications

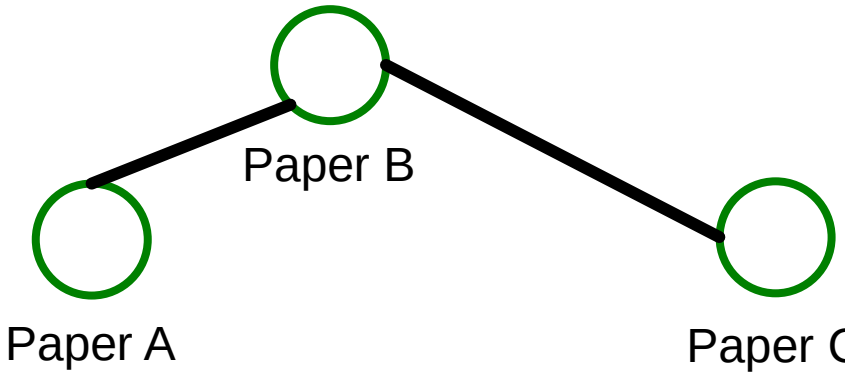
- Web page topic prediction
  - Input: Graph of web pages;
    - Node: Web page; Edge: Hyper-link between web pages;
    - Input node features: extra text features from the webpage
  - Task: Predict the topic of the webpage
    - Classify each node



https://towardsdatascience.com/industrial-classification-of-websites-by-machine-learning-with-hands-on-python-3761b1b530f1

GCN applications

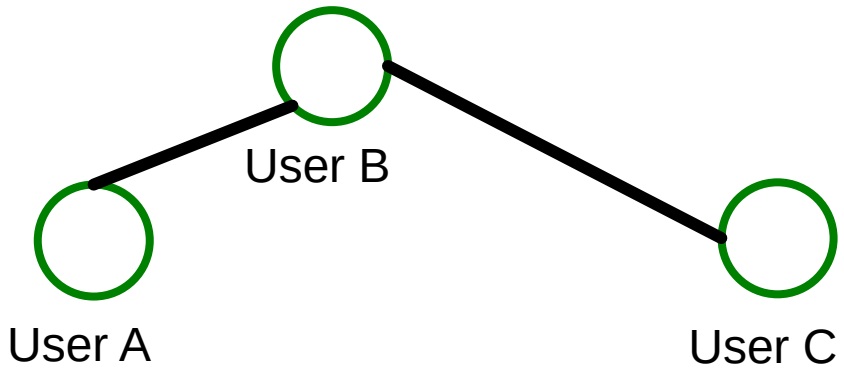
- Paper research topic classification
  - Datasets: publication datasets
  - Node: each paper; Edge: citation relation;



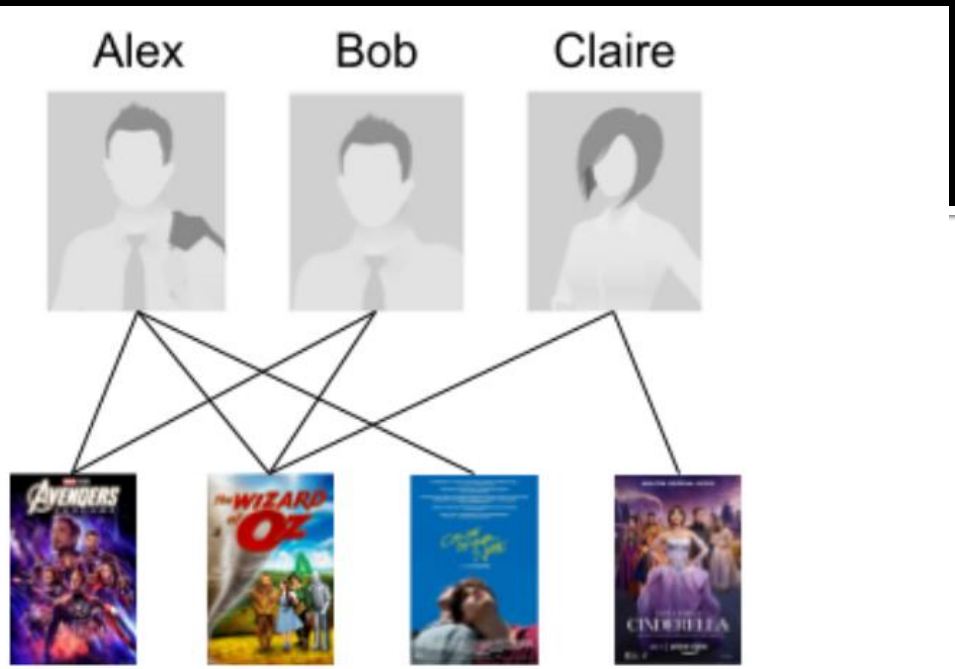
Predict the topic of the paper as a classification problem
Topics: {action recognition, semantic segmentation, instance segmentation ….}

GCN applications

- User attribute (tag) prediction
  - Attributes can be occupations, professionals, interests, …
  - Datasets: social network datasets
  - Node: each user;
    Edge: friendship relation; contact record; subscription, following relationship.



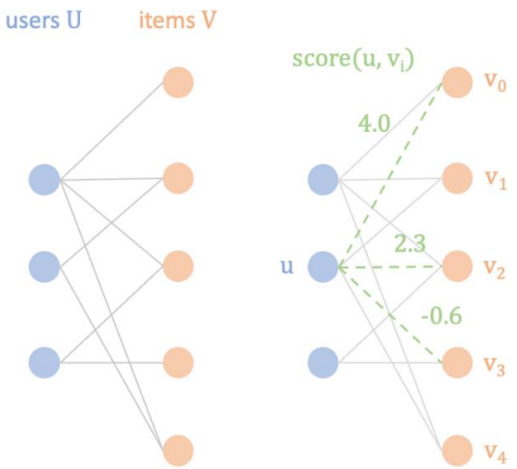
Predict the attributes of a user as a binary classification problem for each attribute:
Interests: {dancing, reading, traveling, cycling, swimming, PC gaming, ….}



User-Item interactions can be modeled as a bipartite undirected graph.

GCN for Recommender Systems

Classify positive and negative pairs
Positive (u,v): connected pairs
Negative (u,v): not connected pairs
Use dot-product to calculate the similarity of a user-item pair



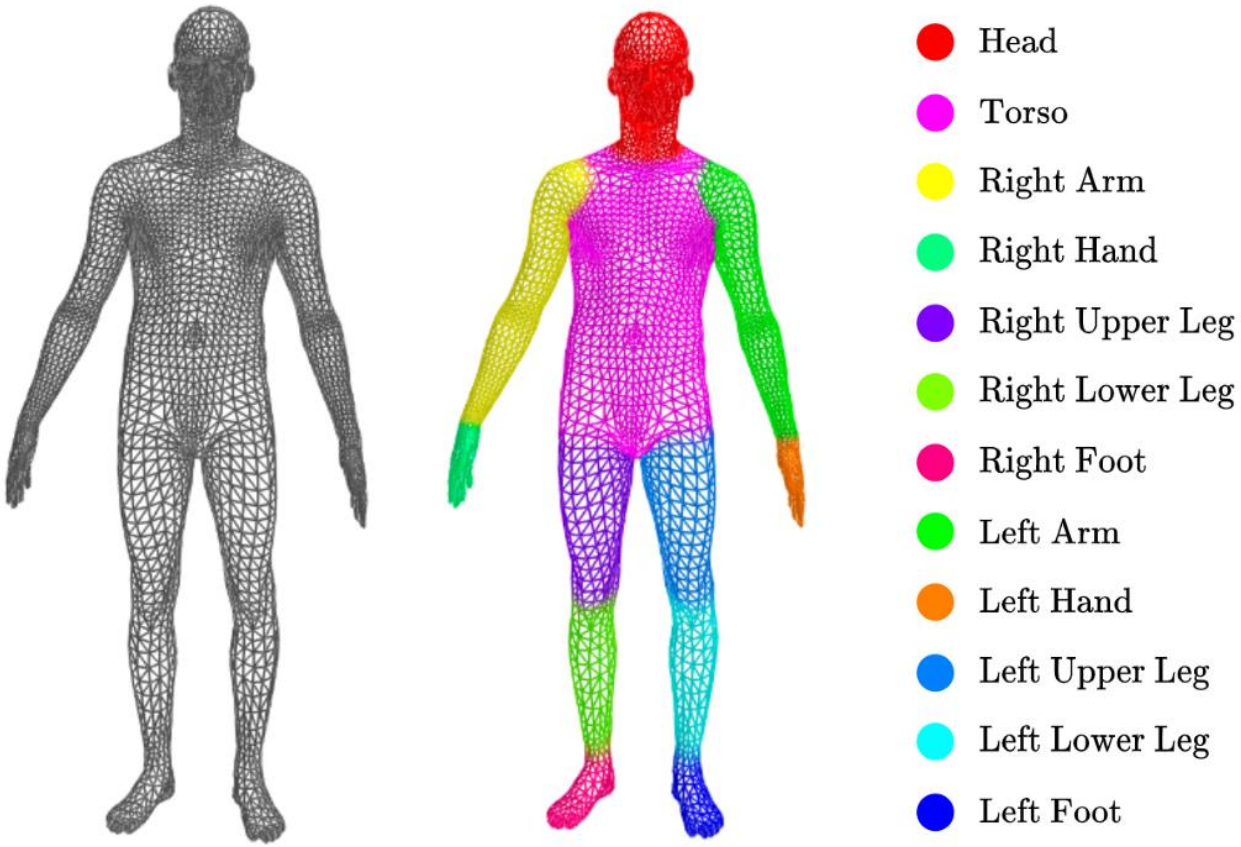
top-2 recommendations for user u

$\{v_0, v_2\}$

https://medium.com/stanford-cs224w/recommender-systems-with-gnns-in-pyg-d8301178e377

GCN applications

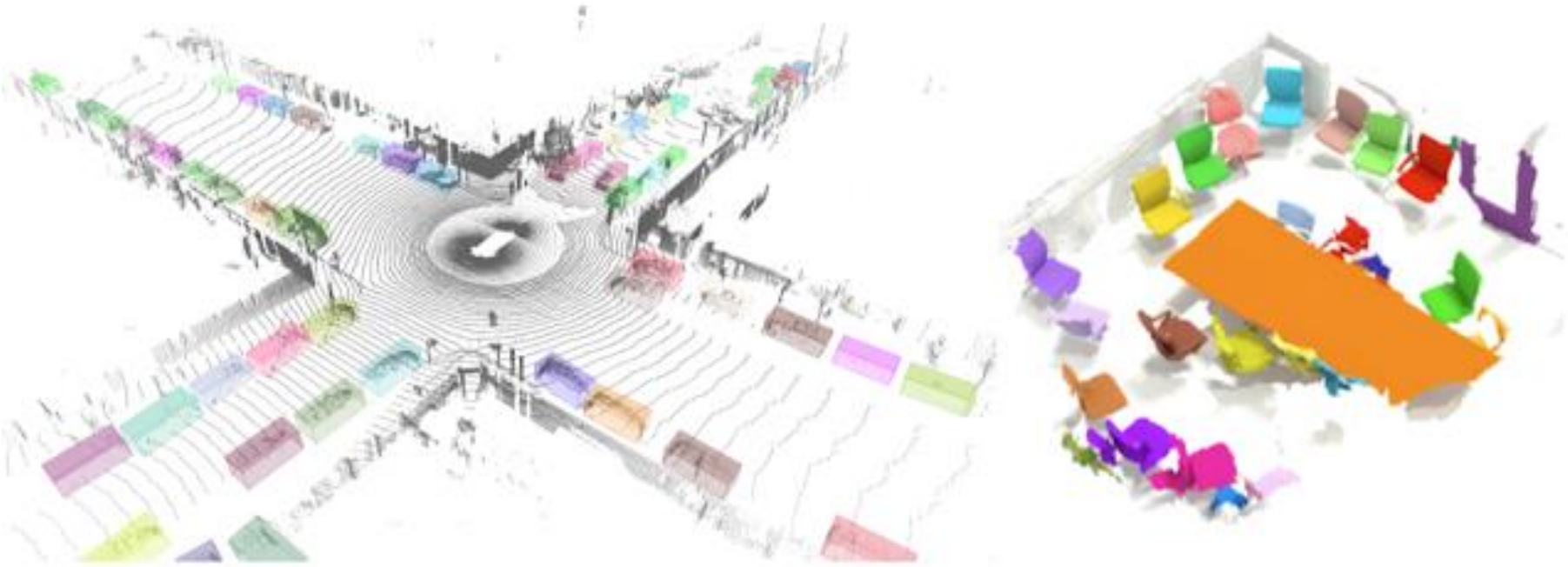
GCN for node classification on 3D meshes:



Graph segmentation task: each vertex in the mesh is assigned to one of twelve body-parts.

Node features: (x,y,z) coordinates of each vertice

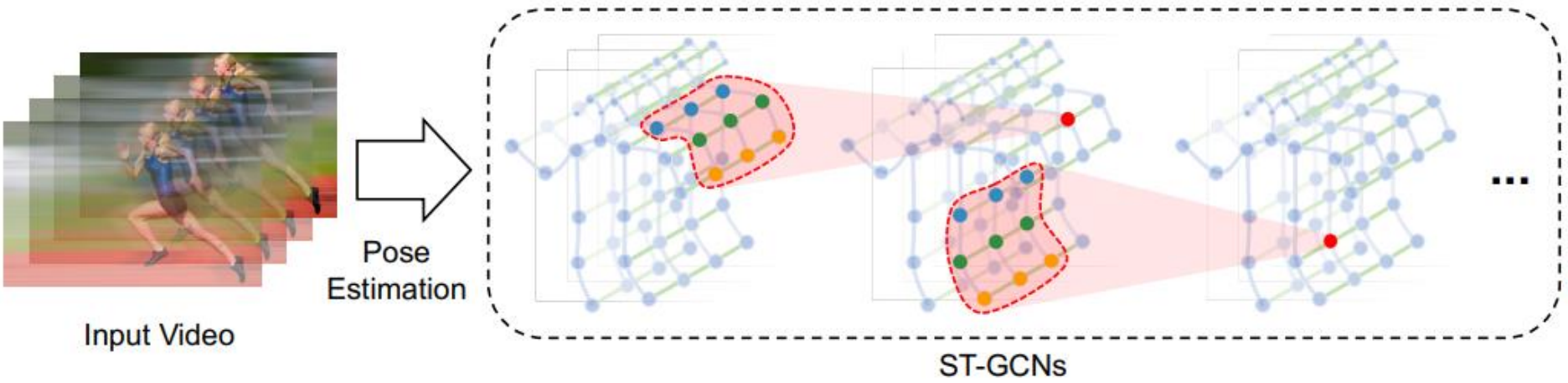
https://medium.com/stanford-cs224w/deep-learning-on-3d-meshes-9608a5b33c98



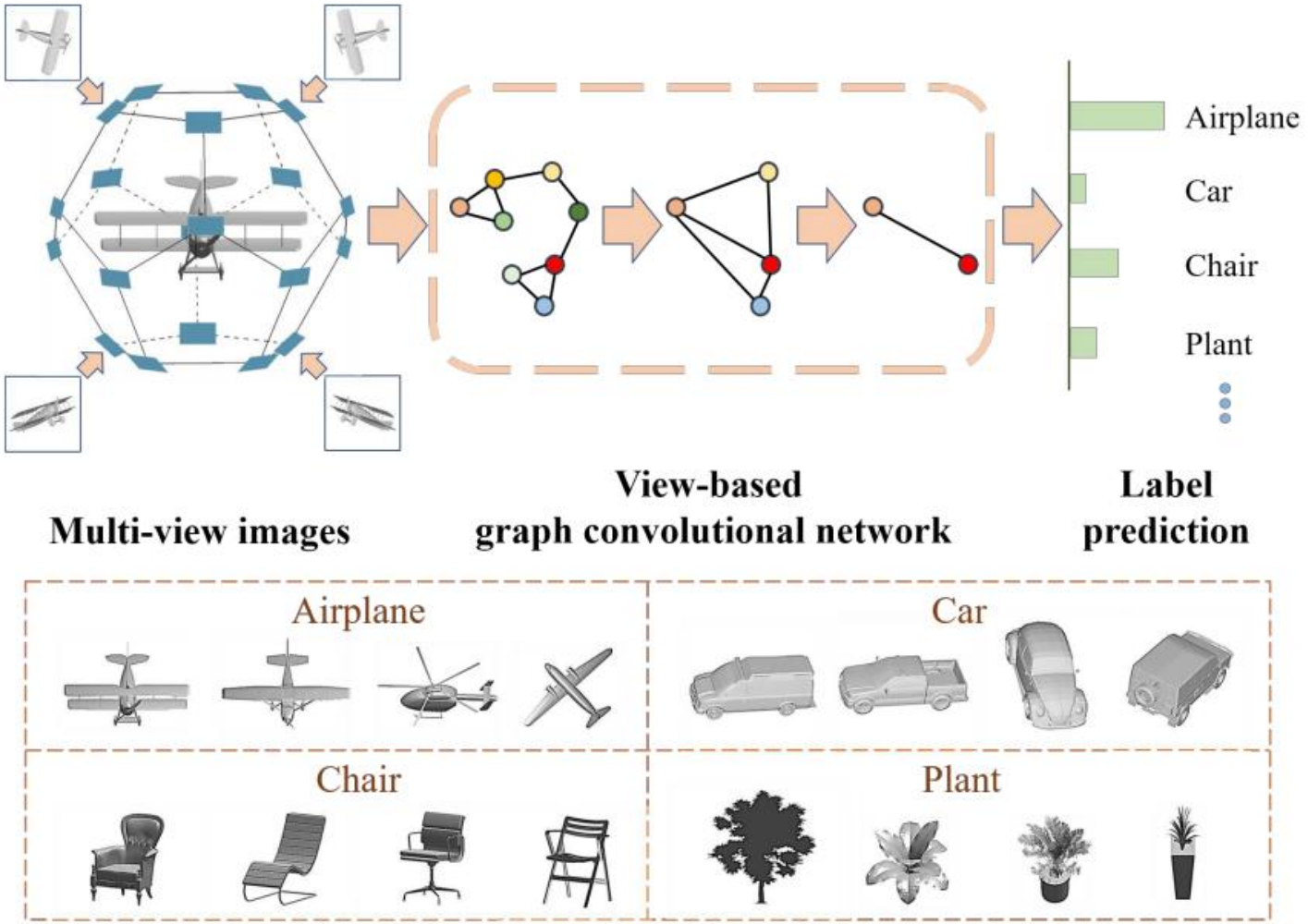
3D point cloud segmentation

https://ai.googleblog.com/2021/02/3d-scene-understanding-with-tensorflow.html

- GCN for skeleton based action recognition



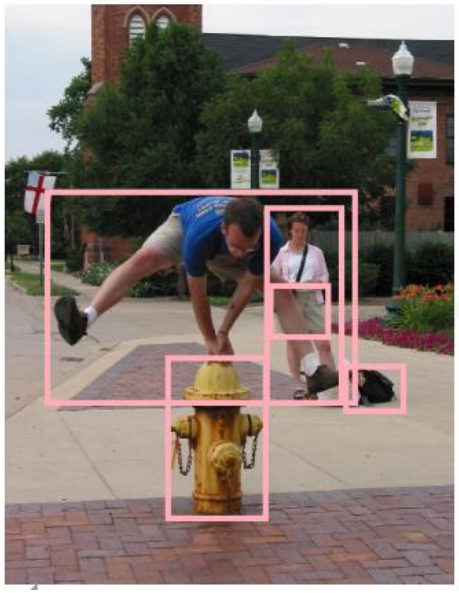
Paper: Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition
https://arxiv.org/pdf/1801.07455.pdf



- GCN for 3D object classification

Paper: View-GCN: View-based Graph Convolutional Network for 3D Shape Analysis

- GCN for scene graph generation and captioning



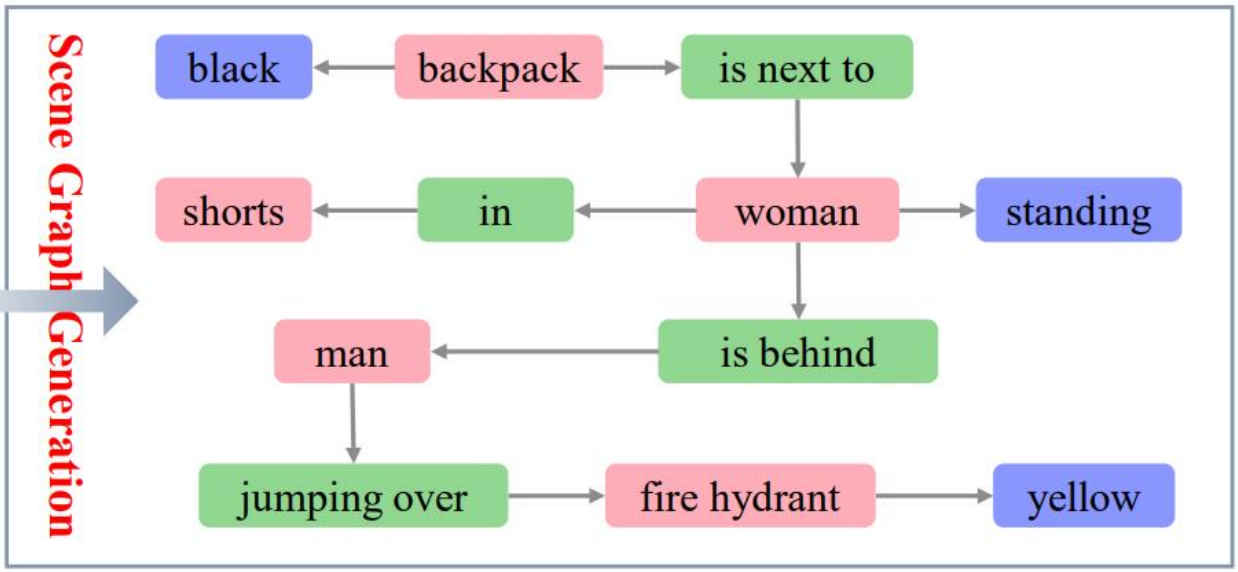


Image captioning

The woman in shorts is standing behind the man who is jumping over fire hydrant.

https://arxiv.org/pdf/2201.00443.pdf

More applications

- A collection of graph Learning tutorials/applications
  - https://medium.com/stanford-cs224w
  - https://github.com/pyg-team/pytorch_geometric#implemented-gnn-models
- other examples:
  - https://paperswithcode.com/task/node-classification

GCN discussions

- Other deep learning methods for graphs:
  - Transformer
    - Another popular method that can be used on graphs
      - http://papers.neurips.cc/paper/9367-graph-transformer-networks.pdf
      - http://web.stanford.edu/class/cs224w/

GCN training

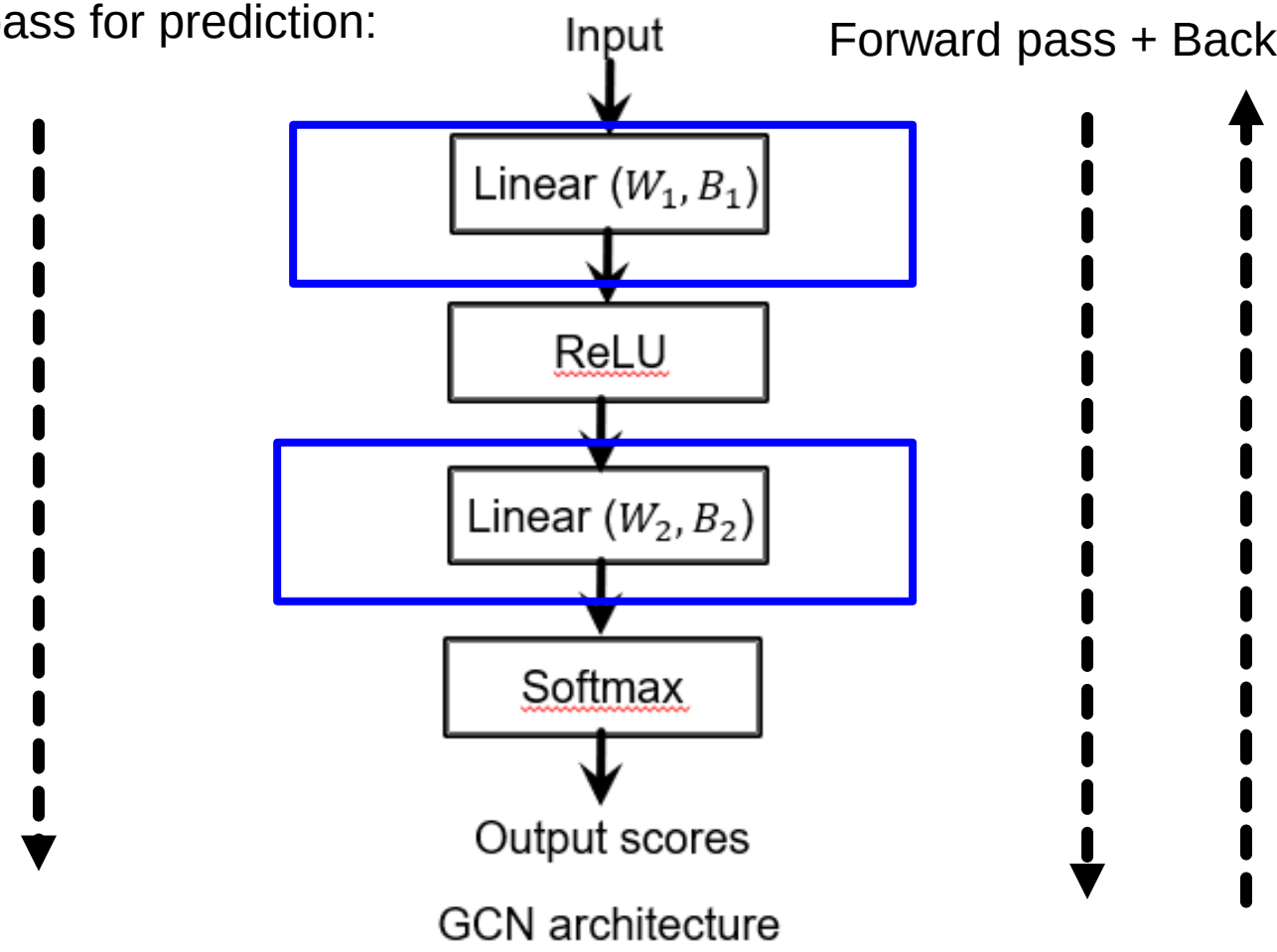
- GCN training
  - Loss functions
    - Cross-entropy loss
    - L2 loss
  - Backward pass
    - gradient descent methods

GCN training

Use gradient descent for training

Forward pass for prediction:

Forward pass + Backward pass



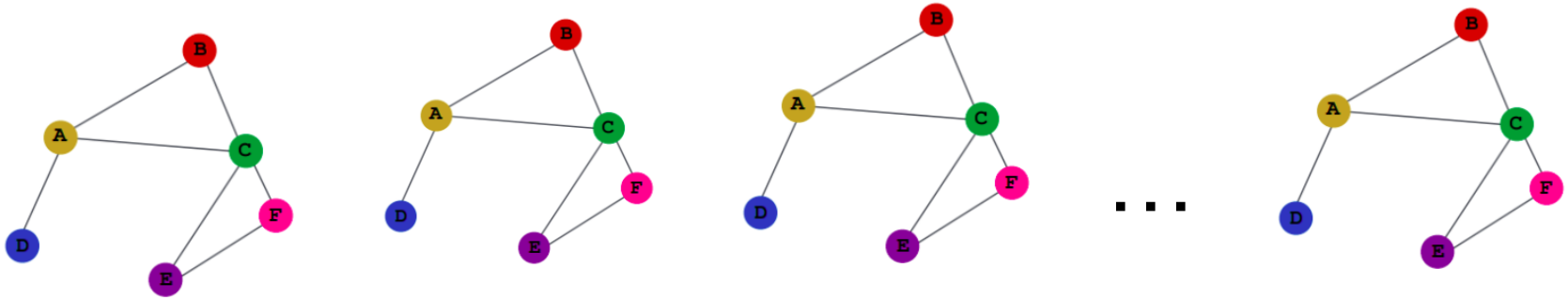
GCN architecture

Training: find solutions to the network parameters in each layer!

GCN training

- Training data

Training set:

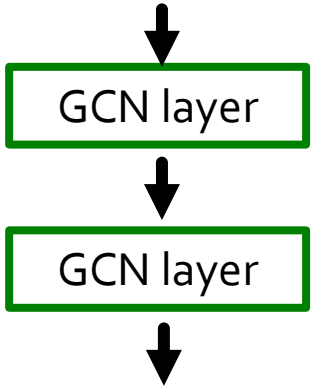


+

Class label for each node

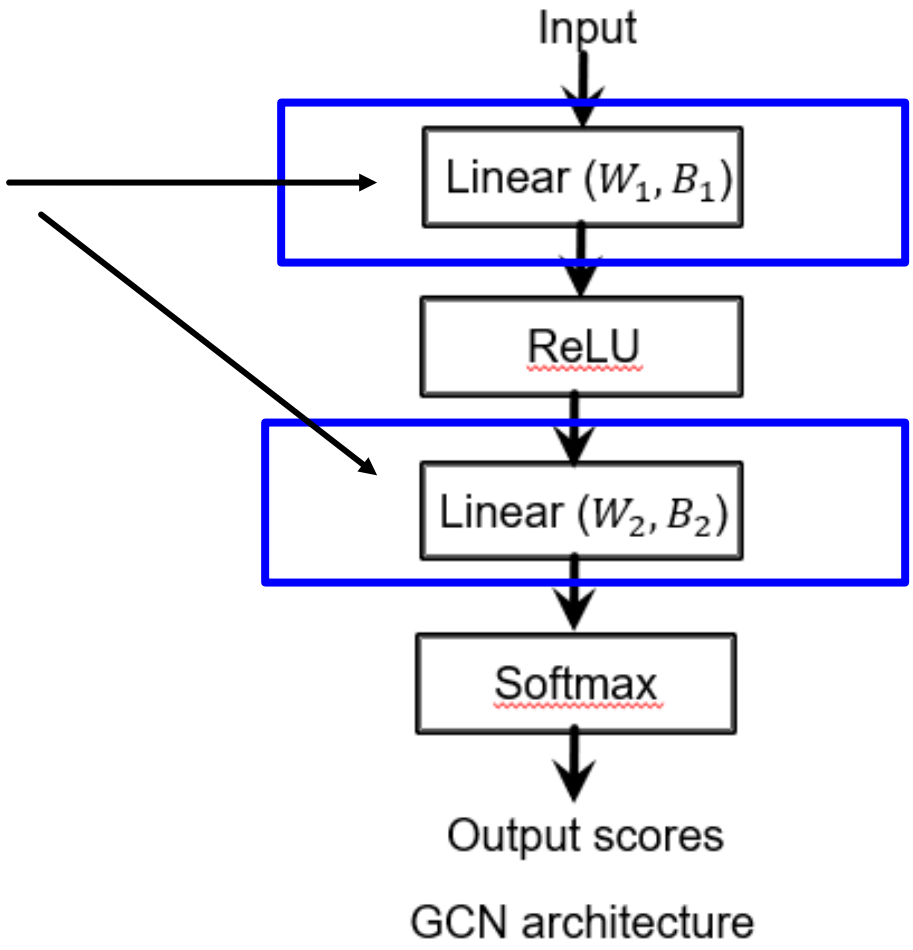
(ground-truth labels)

Train a GCN model



Example:

Learnable parameters



GCN architecture

- GCN loss functions

1. Cross-entropy loss for classification:

$$L(f(x)) = -\sum_i \mathbf{y}_i \cdot \log \mathbf{z}_i(\theta)$$

$$z_{ik}(\theta) = \text{Softmax}(h_{ik}) = \frac{e^{h_{ik}(\theta)}}{\sum_{j=1}^{C} e^{h_{ij}(\theta)}}$$

2. For node regression problems, you may use L2 loss:
E.g., predict real numbers for each node

$$L(f(\mathbf{x})) = \sum_{\mathbf{i}} \|\mathbf{y_i} - \mathbf{h_i}(\mathbf{x})\|^2$$

GCN Loss functions

- Loss function for node classification
  - Cross-entropy loss (vector expression, also called softmax loss)

$$L(f(x)) = -\sum_i \mathbf{y}_i \cdot \log \mathbf{z}_i(\theta)$$

$$z_{ik}(\theta) = \mathrm{Softmax}(h_{ik}) = \frac{e^{h_{ik}(\theta)}}{\sum_{j=1}^{C} e^{h_{ij}(\theta)}}$$

$\mathbf{z}_i$: The score vector output by Softmax for the node i

z_{ik} : The k-th element the score vector z_i

h_{ik} : The k-th element of the logit vector output by the network, corresponding to the k-th class.

$\mathbf{y}_i$:The one-hot vector indicating the ground-truth class label

Cross-entropy measures the distance between the ground-truth vector and the prediction vector.

One-hot vector example:

One-hot vector is the ground-truth class score vector

Groudtruth class label: Represented by one-hot vector:

Class 1 → $[\ 1, 0, 0, 0\ ]^T$

Class 2 → $[\ 0, 1, 0, 0\ ]^T$

Class 3 → $[\ 0, 0, 1, 0\ ]^T$

Class 4 → $[\ 0, 0, 0, 1\ ]^T$

For the c-th class, the c-th element in the one-hot vector is set to 1

Cross-entropy loss:

$$\min_{\theta} L(Y, Z) = -\sum_{i} y_i \cdot \log z_i(\theta)$$

If the score vector z_i is similar to the groudtruth vector y_i, the loss is small.

θ is the the learnable parameters of all GCN layers:

$$\theta = \{W_1, B_1; W_2, B_2; \ldots; W_K, B_K;\}$$

GCN training:
We need to solve the optimization problem to obtain the solution of the learnable parameters

$$L(f(x)) = -\sum_i \mathbf{y}_i \cdot \log \mathbf{z}_i(\theta)$$

- Cross-entropy loss can be also written as:

$$L(f(x)) = -\sum_i \sum_{k=1}^{C} y_{ik} \log z_{ik}(\theta)$$

$$z_{ik}(\theta) = \text{Softmax}(h_{ik}) = \frac{e^{h_{ik}(\theta)}}{\sum_{j=1}^{C} e^{h_{ij}(\theta)}}$$

k indicates the k-th class. C is the total number of classes

y_{ik} : is the k-th element in the groundtruth one-hot vector $\mathbf{y}_i$

Cross-entropy between two distributions: measure the distance (similarity) between two distributions:

Example: we have 3 classes in total.

y: ground-truth distribution for one sample

z: predicted distribution for one sample

Ground-truth

$$\mathbf{y_1} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Prediction 1

$$\mathbf{z_1} = \begin{bmatrix} 0.2 \\ 0.5 \\ 0.3 \end{bmatrix}$$

Prediction 2

$$\mathbf{z'_1} = \begin{bmatrix} 0.1 \\ 0.9 \\ 0.0 \end{bmatrix}$$

Cross-entropy for one sample:

$$L_i(\mathbf{y_i}, \mathbf{z_i}) = -\sum_{k=1}^{C} y_{ik} \log z_{ik}$$

$$L(\mathbf{y_1}, \mathbf{z_1}) = -\log(0.5) = 0.3010$$ Large distance

$$L(\mathbf{y_1}, \mathbf{z'_1}) = -\log(0.9) = 0.0458$$ Small distance

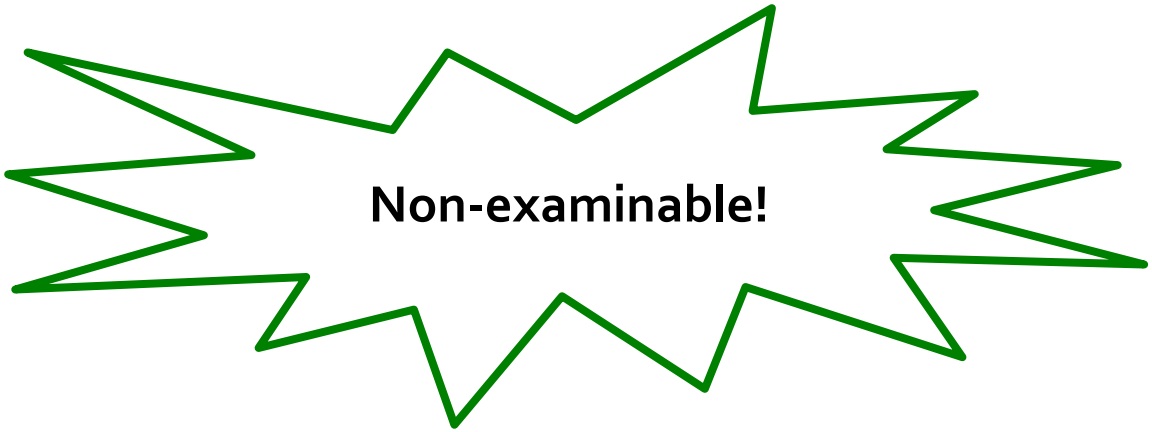
We need to solve this optimization problem to obtain the network parameters (weights)

Cross-entropy loss:

$$\min_{\theta} L(Y, Z) = -\sum_{i} y_i \cdot \log z_i(\theta)$$

GCN training:
We need to solve the optimization problem to obtain the solution of the learnable parameters

- GCN training
  - Backward pass **(non-examinable)**
    - gradient descent methods



- Use gradient decent to solve the optimization

Iterative algorithm: repeatedly update weights in the (opposite) direction of gradients until convergence

$$\theta(t+1) = \theta(t) + \Delta\theta(t);$$
$$\Delta\theta(t) = -\eta\nabla_\theta L;$$

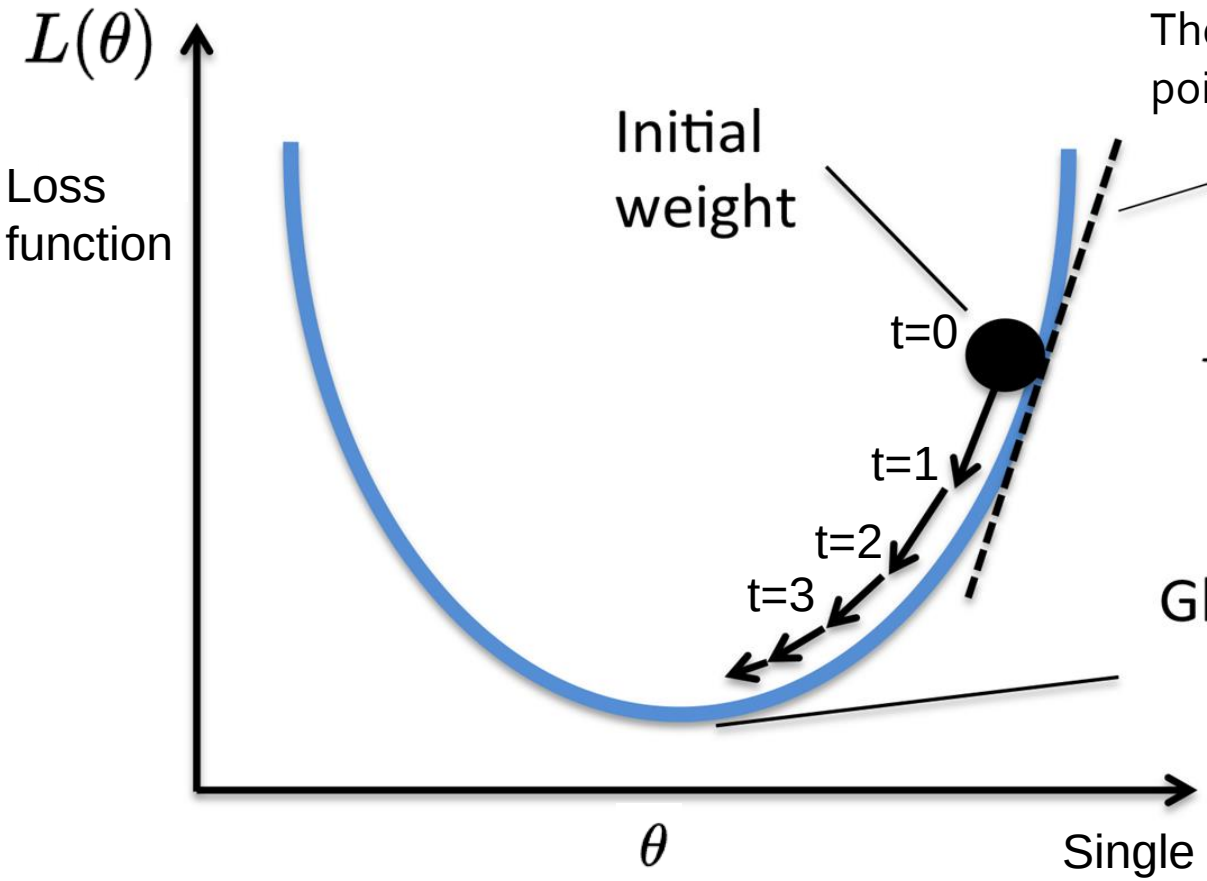
$\Delta\theta(t)$: The update vector, a small change to the solution.

η :Learning rate (LR), the step size for gradient update:
Hyperparameter that controls the size of gradient step
Can vary over the course of training (LR scheduling)

Gradient vector: a vector of partial derivatives of all variables.

Two key factors for gradient descent:
1 update direction (gradients),
2 update step size (learning rate)

$$\theta(t+1) = \theta(t) + \Delta\theta(t);$$
$$\Delta\theta(t) = -\eta\nabla_\theta L;$$



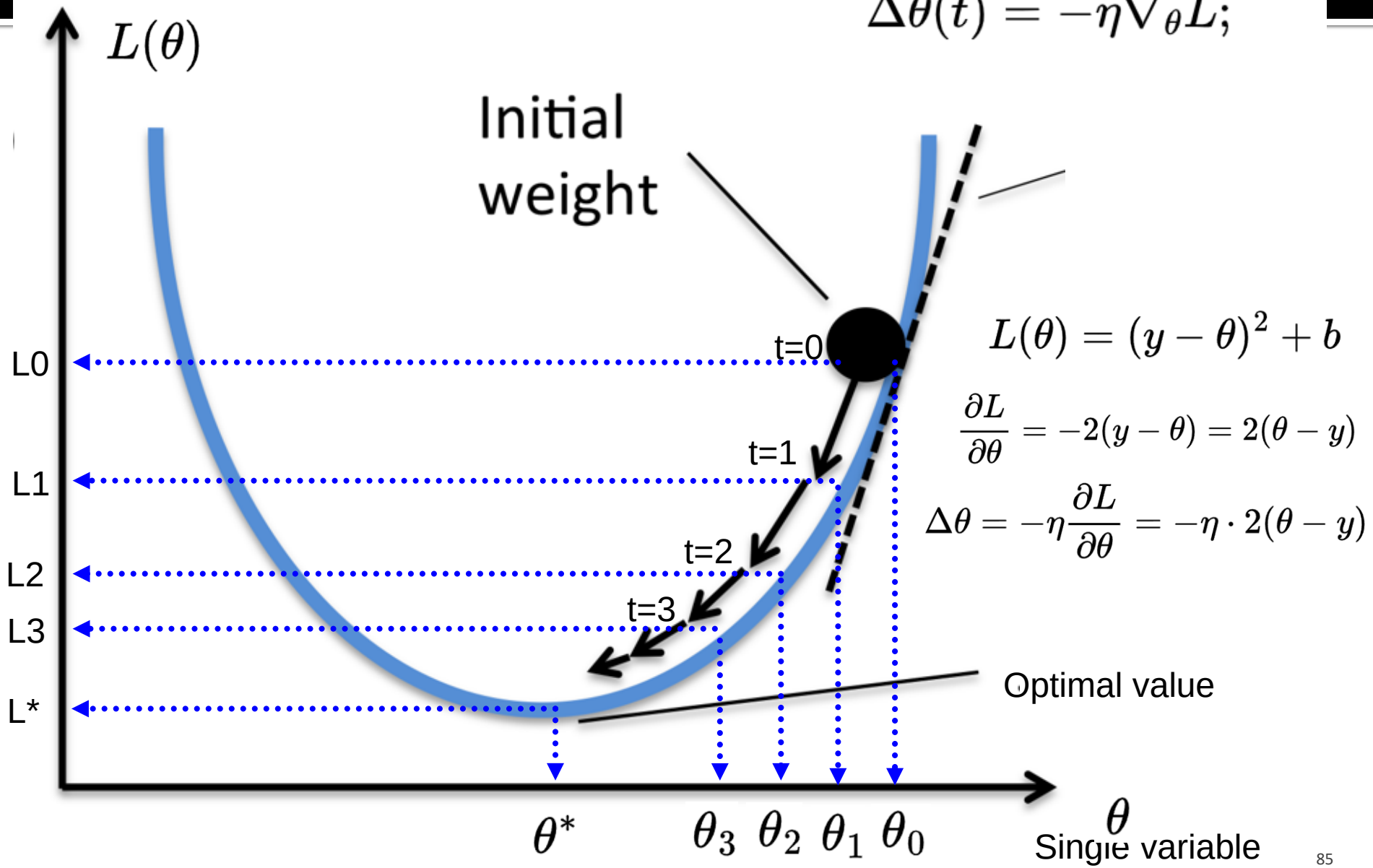
The slope of a tangent line at a point is its derivative at that point.

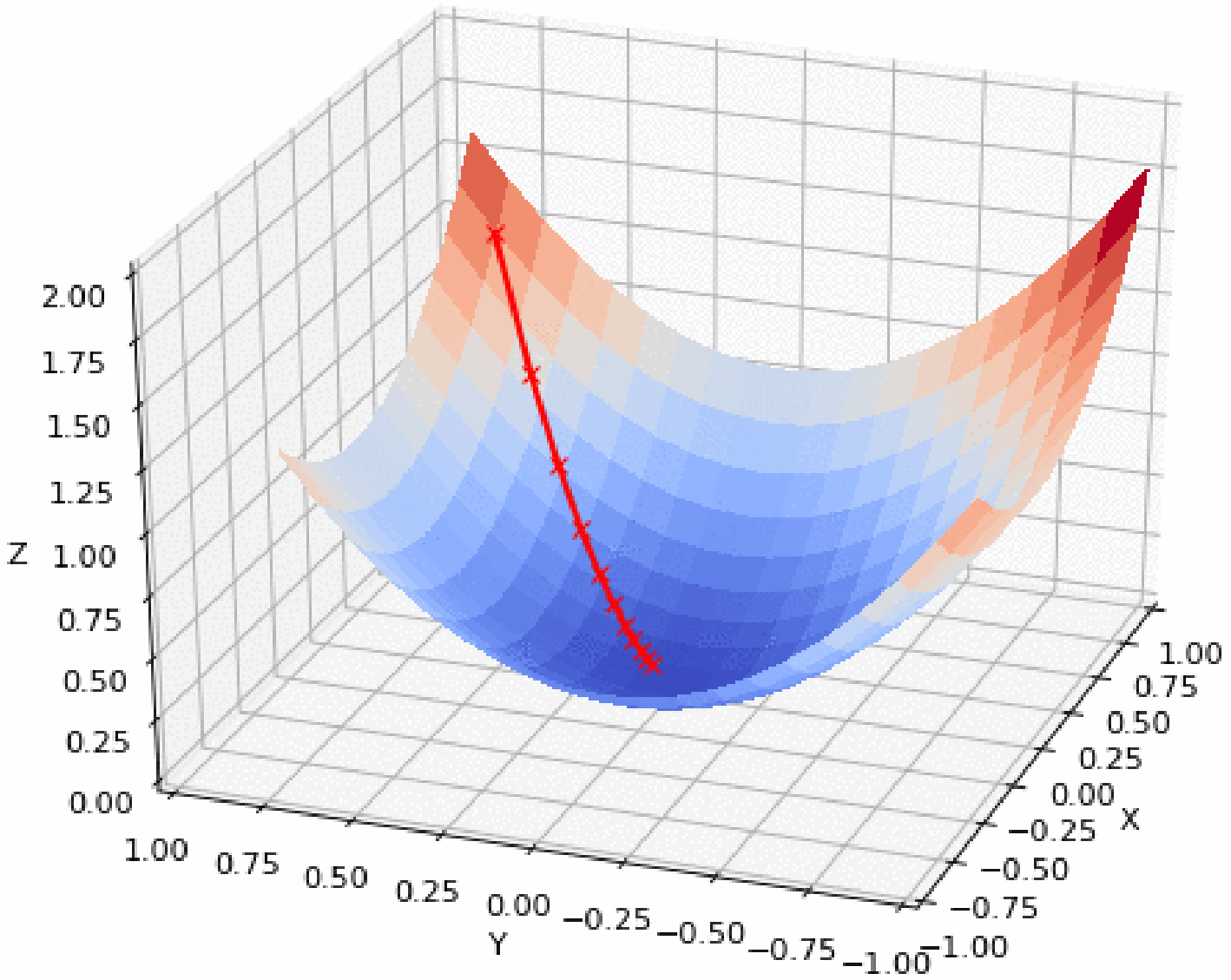
Gradient

$$L(\theta) = (y-\theta)^2 + b$$

Global cost minimum

$$\theta(t+1) = \theta(t) + \Delta\theta(t);$$
$$\Delta\theta(t) = -\eta\nabla_\theta L;$$





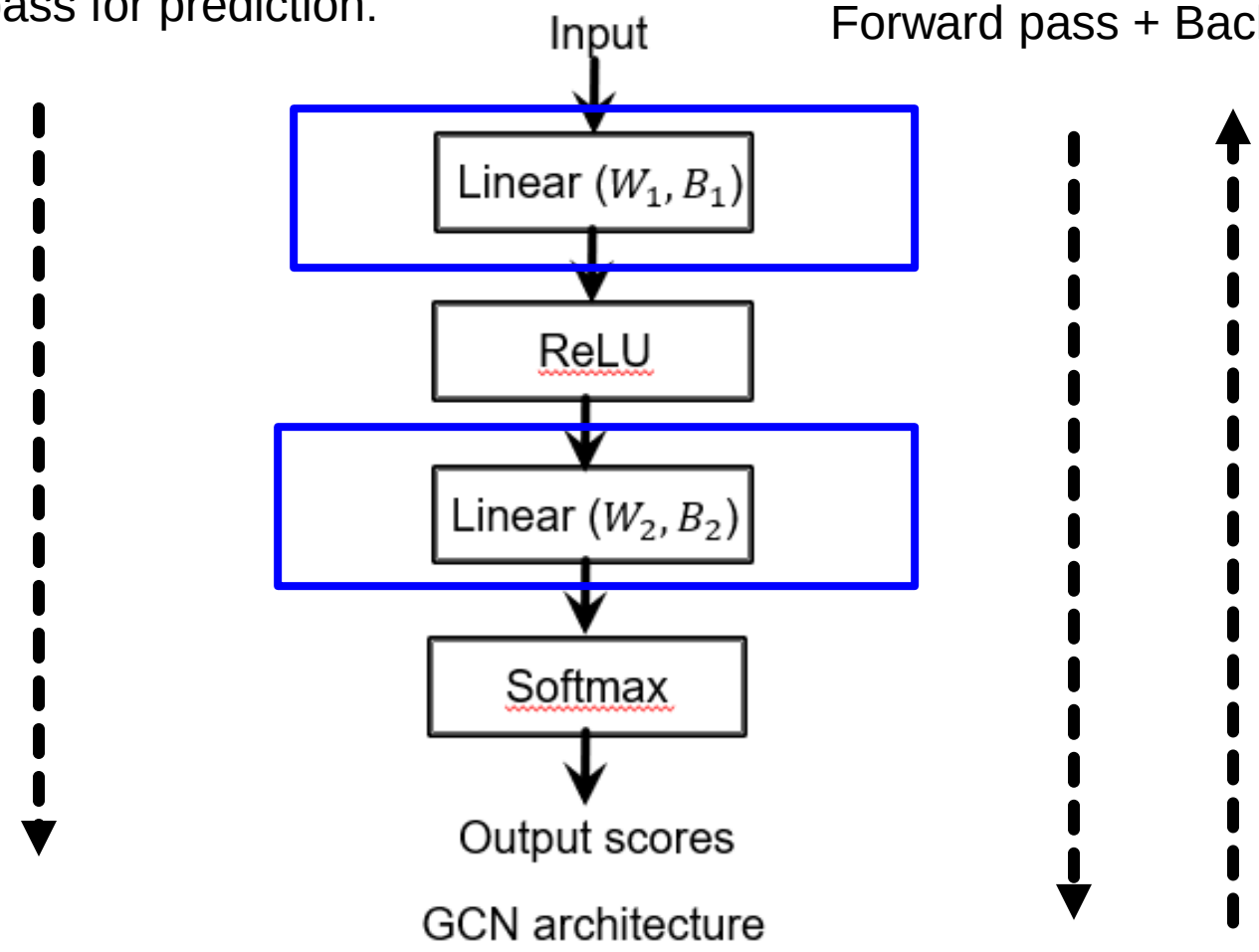
two variables

https://thinkingandcomputing.com/posts/genetic-algorithms-neural-networks.html

GCN training

Forward pass for prediction:

Use gradient descent for training
Each training iteration:
Forward pass + Backward pass



GCN architecture

- Backward pass:
  - Apply back-propagation to calculate gradients for all layers and update the parameters.
  - Back-propagation: Use of chain rule to calculate gradients for all layers

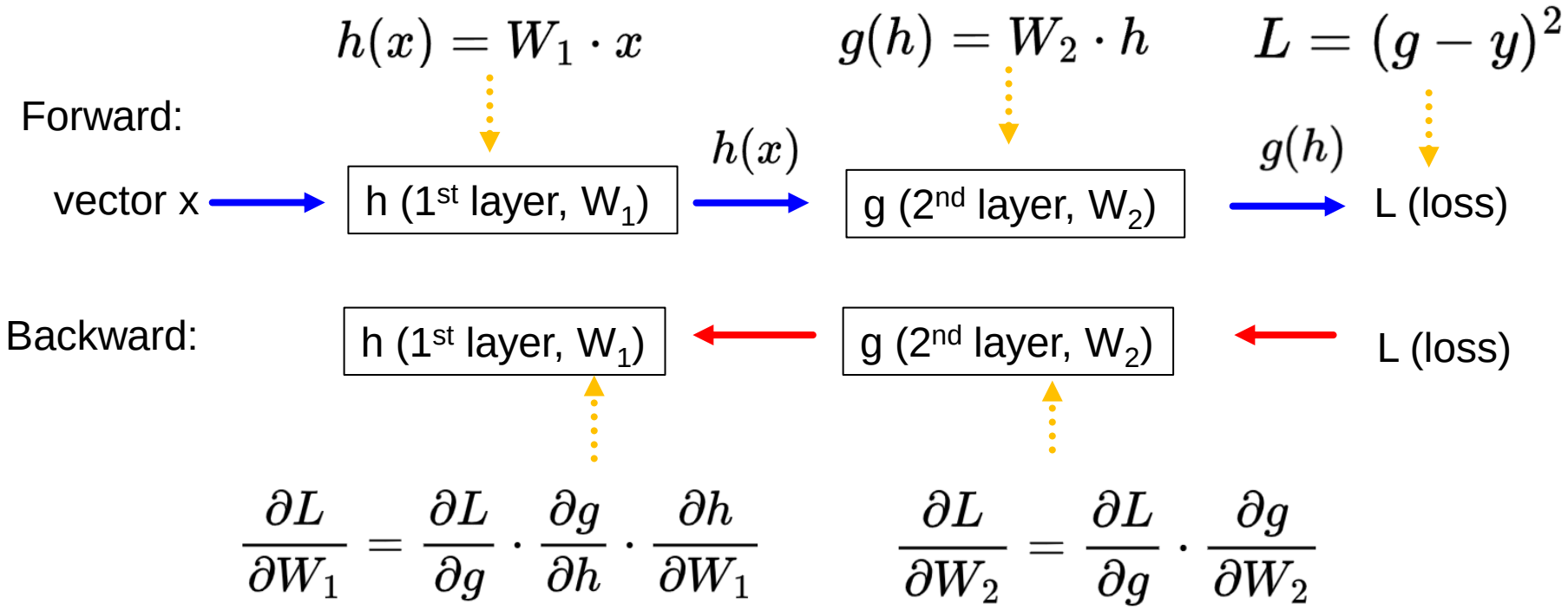
chain rule for gradient calculation for composition function:

$$\frac{d}{dt} f(g(t)) = f'(g(t))g'(t) = \frac{df}{dg} \cdot \frac{dg}{dt}$$

https://pl.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/differentiating-vector-valued-functions/a/multivariable-chain-rule-simple-version

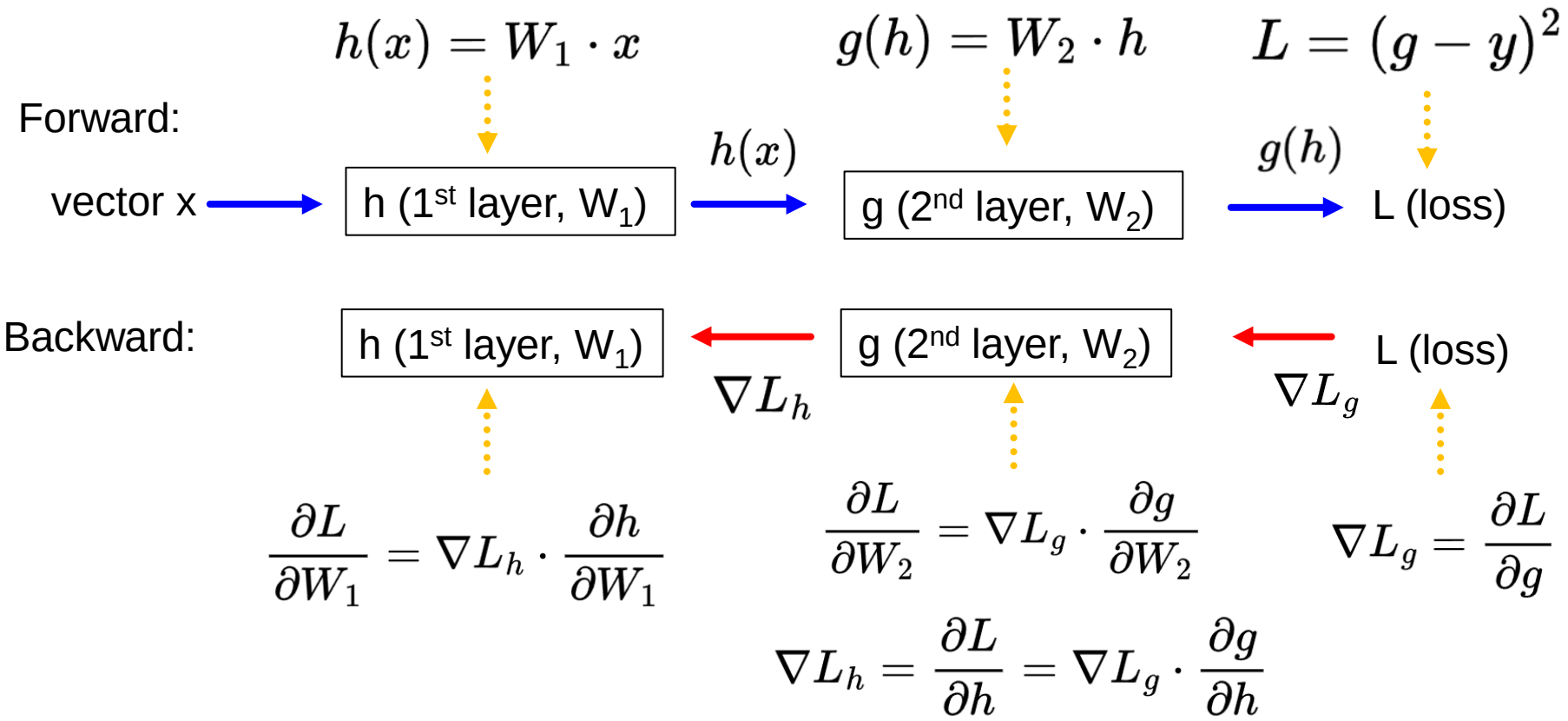
- Backward pass example on a simplified network (2 layers, no RELU)

L is a loss function defined on f(x), For example, L is an L2 loss for regression

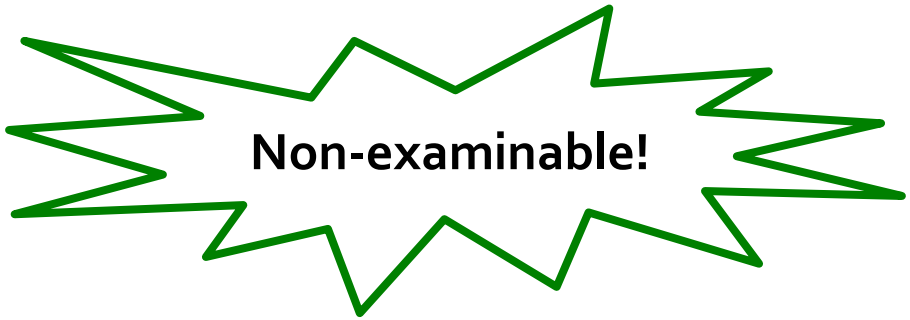


Using chain rule for gradient calculation

- Backward pass example on a simplified network (2 layers, no RELU)

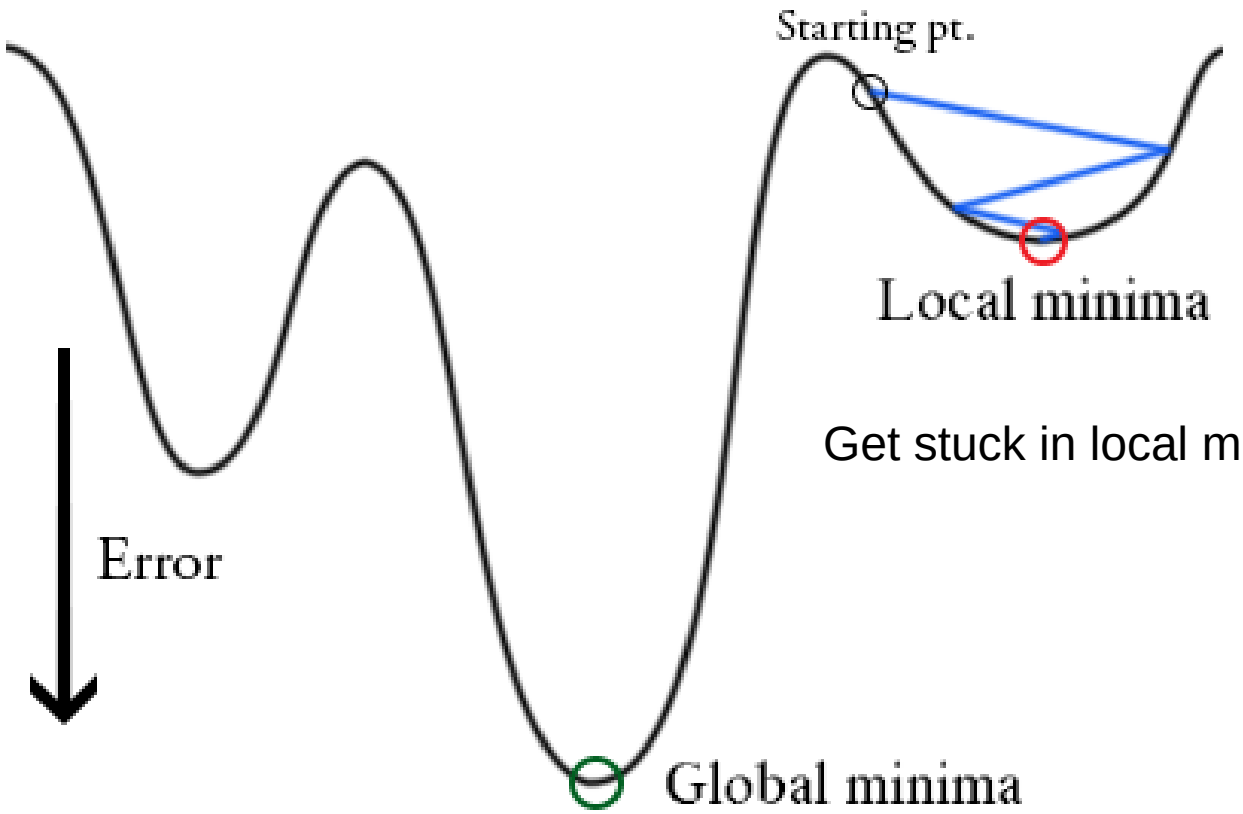


Backward pass for gradient calculation



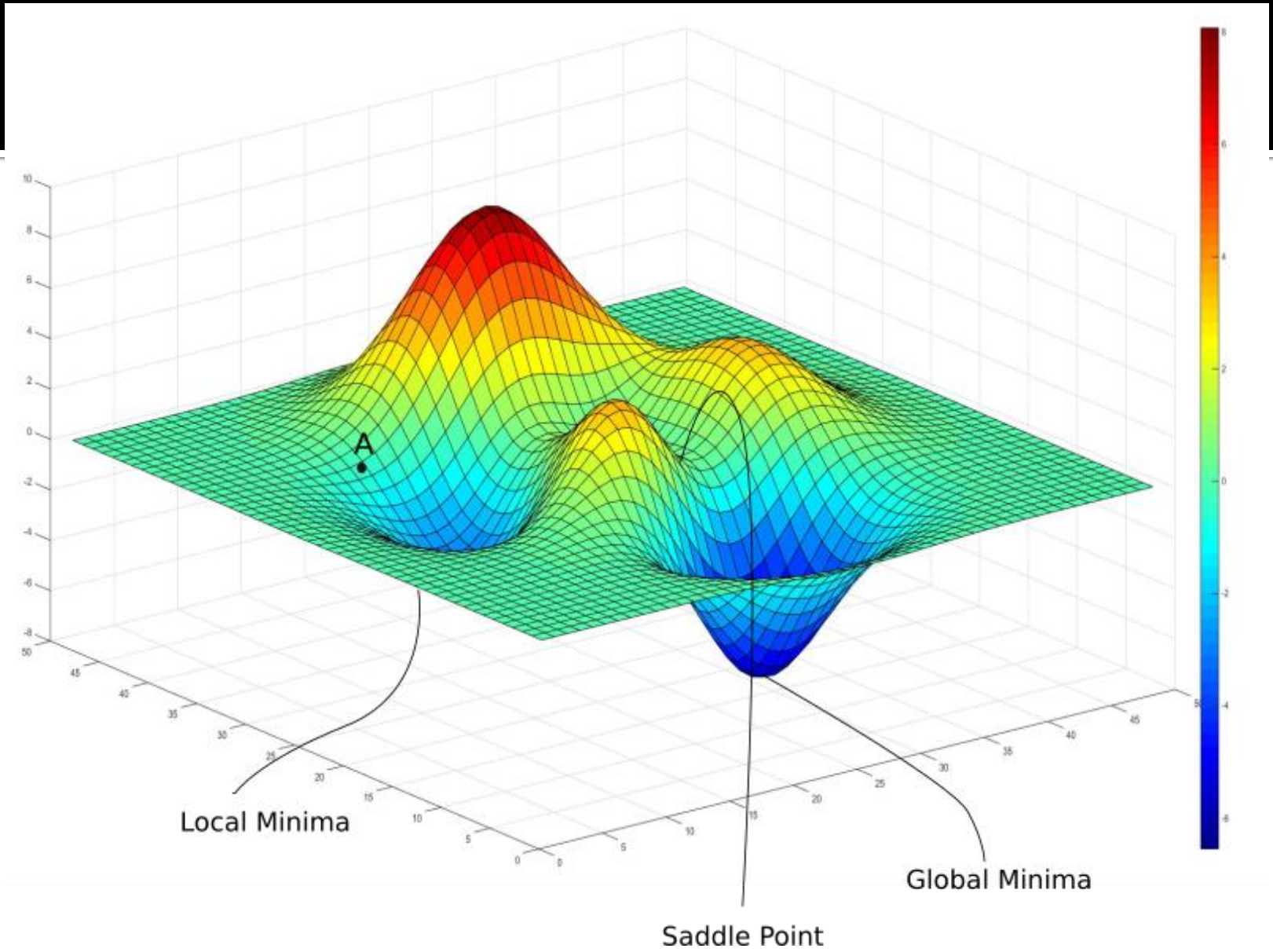
- Further discussion
  - Limitation of gradient descent
  - Minibatch SGD
  - How to set learning rate?
  - Comparisons of activation functions
  - Weight decay (L2 regularization)
  - Momentum SGD

Limitations of Gradient decent



Get stuck in local minimum

Gradient decent for non-convex function (one variable)



Non-convex loss function (two variables)

https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/

GCN training

- Minibatch Stochastic Gradient Decent (SGD)
  - Minibatch SGD
  - For each training iteration, randomly select a small set of data points (e.g., nodes) to perform gradient decent update.
  - Advantages:
    - Can be applied for large scale problems
      - Cannot calculate the gradients for all data points
    - The random sampling Introduces noises into the optimization, help to regularize the training and jump out of local minimum.

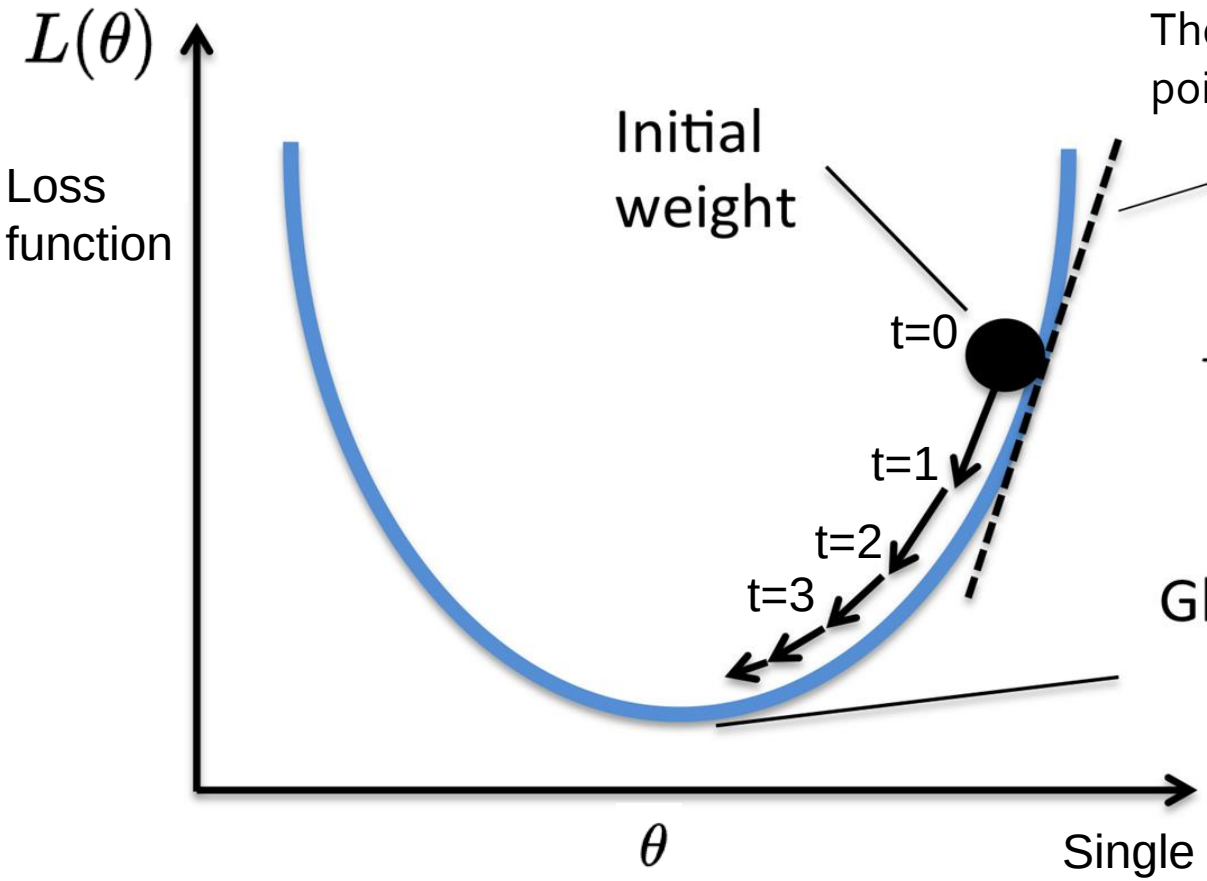
- Minibatch Stochastic Gradient Decent (SGD)

  Some concepts for minibatch SGD:

  - Batch size: the number of data points in a minibatch e.g., number of nodes for node classification task
  - Iteration: 1 step of SGD on a minibatch
  - Epoch: one full pass over the dataset (# iterations is equal to ratio of dataset size and batch size)

Two key factors for gradient descent:
1 update direction (gradients),
2 update step size (learning rate)

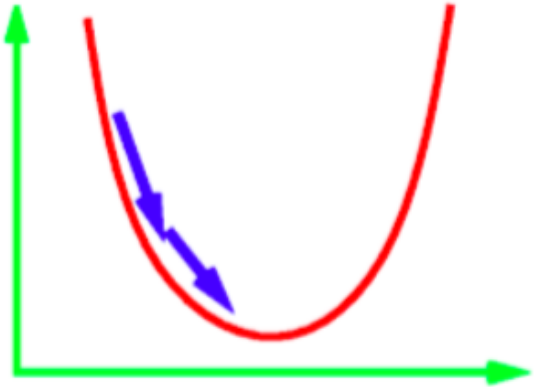
$$\theta(t+1) = \theta(t) + \Delta\theta(t);$$
$$\Delta\theta(t) = -\eta\nabla_\theta L;$$



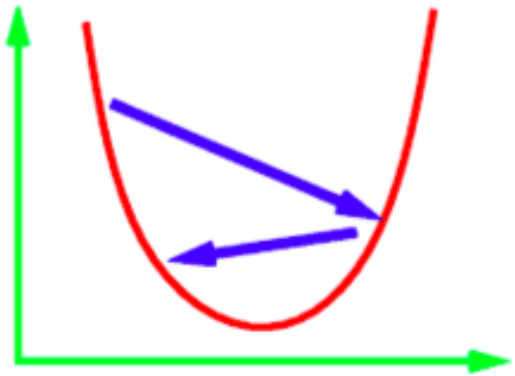
The slope of a tangent line at a point is its derivative at that point.

$$L(\theta) = (y - \theta)^2 + b$$

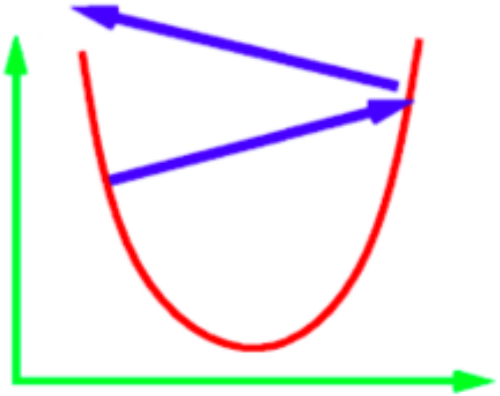
Global cost minimum



Small learning rate



Large learning rate

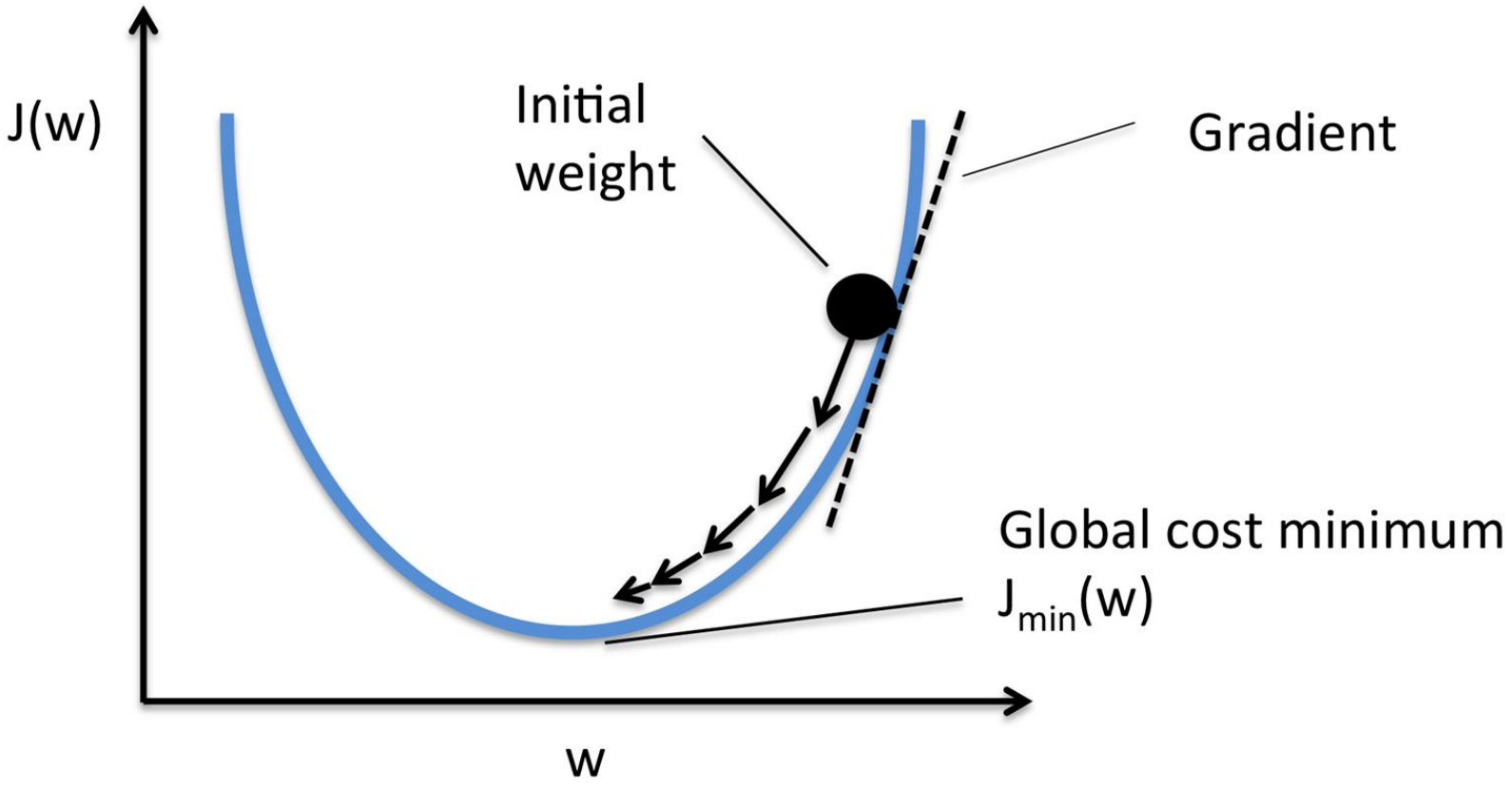


Large learning rate
(diverged)

We need a good setting of learning rate

Learning rate needs to be gradually decreased

Learning rate (LR): the step size for gradient update



learning rate decay strategies:
how to gradually decrease the learning rate

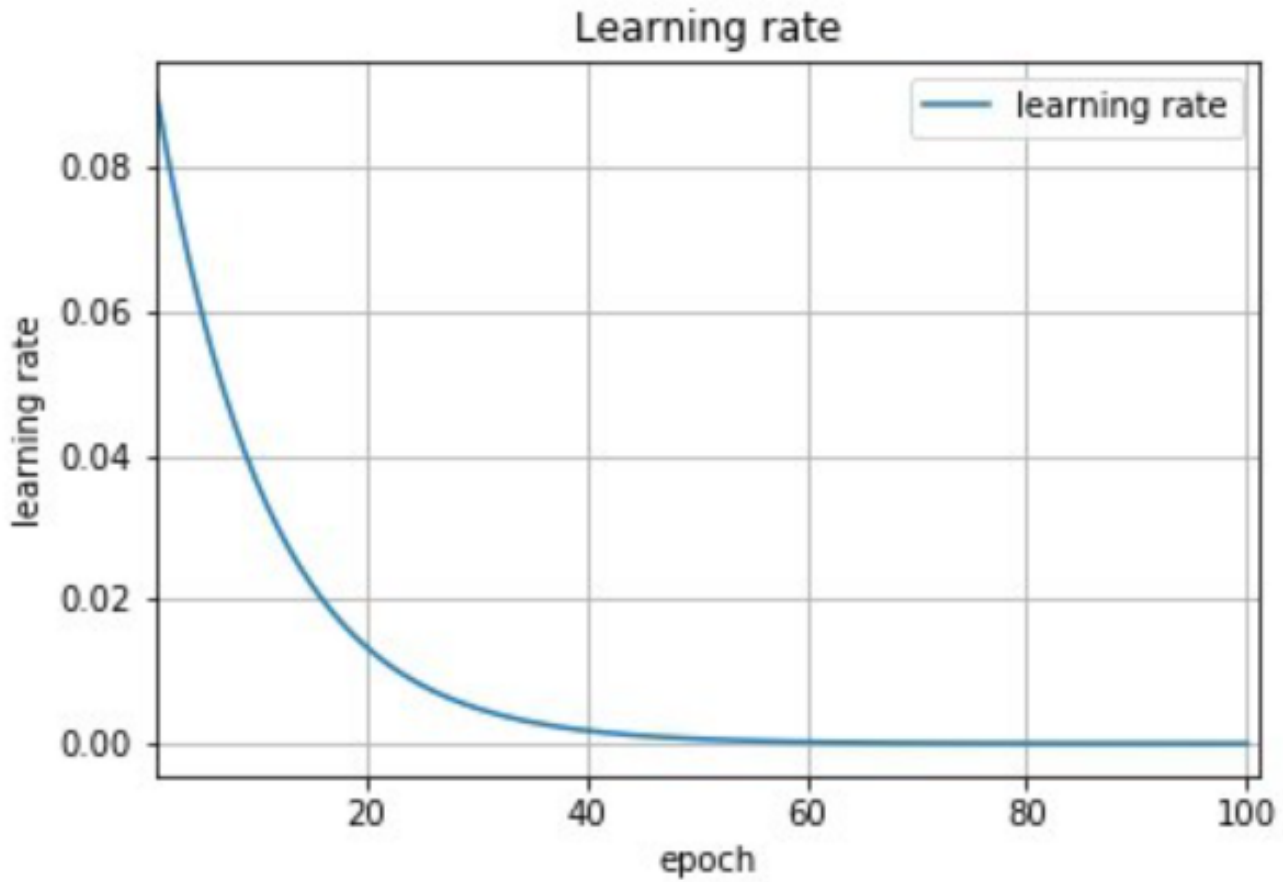


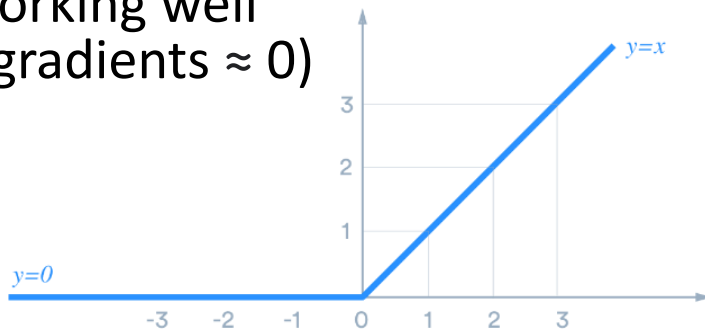
Fig 4b : Exponential Decay Schedule

Learning rate decay scheduler: gradually reduce the learning rate

- Revisit activation function
  - Sigmoid and Hyperbolic tangent are not working well with gradient descent (saturated regions: gradients ≈ 0)

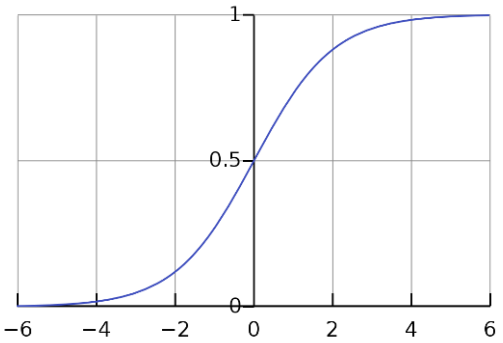
1. Rectified linear unit (RELU)

$$ReLU(x) = \max(x, 0)$$



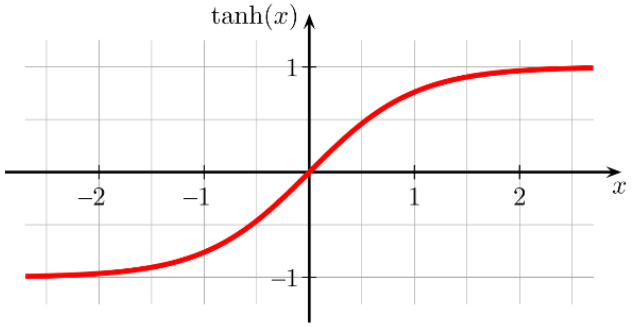
2. Sigmoid

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^{x}}{e^{x} + 1}$$



3. Hyperbolic tangent

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$



- Other details

1. In practice, we optimize this loss function:
Cross-entropy loss + L2 regularization (weight decay)

$$\min_{\theta} L(Y, Z) = -\sum_{i} y_i \cdot \log z_i(\theta) + \frac{1}{2}\lambda \|\theta\|^2$$

L2 regularization: encourage small values for the solution

2. Use momentum mechanism for gradient update (Momentum SGD)

SGD for solution update:

$$\theta(t+1) = \theta(t) + \Delta\theta(t)$$

Standard SGD: $\Delta\theta(t) = -\eta\Delta_\theta L$

Momentum SGD: $\Delta\theta(t) = -\eta\Delta_\theta L + \alpha\Delta\theta(t-1)$

Momentum

Momentum:

1. Encourage the update directions of two consecutive iterations to be similar (more consistent)

2. Noise Smoothing effect

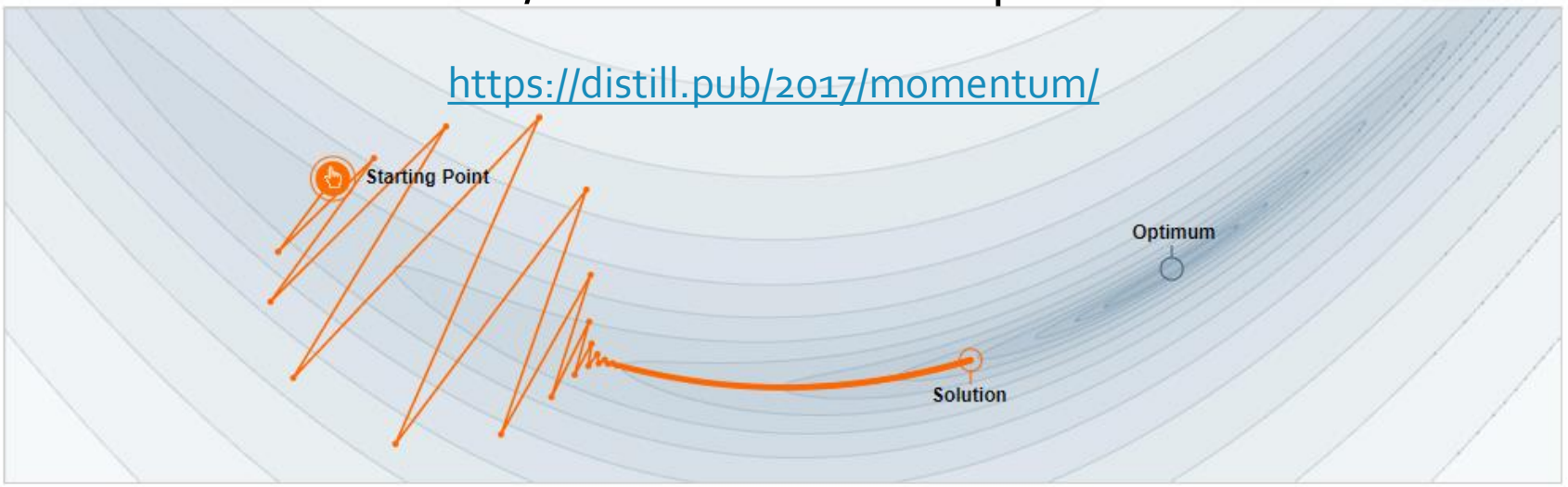
Momentum averages gradients of recent iterations.

It is a running average of gradients that we use for each update step.

Momentum SGD: $\Delta\theta(t) = -\eta\Delta_\theta L + \alpha\Delta\theta(t-1)$

Momentum

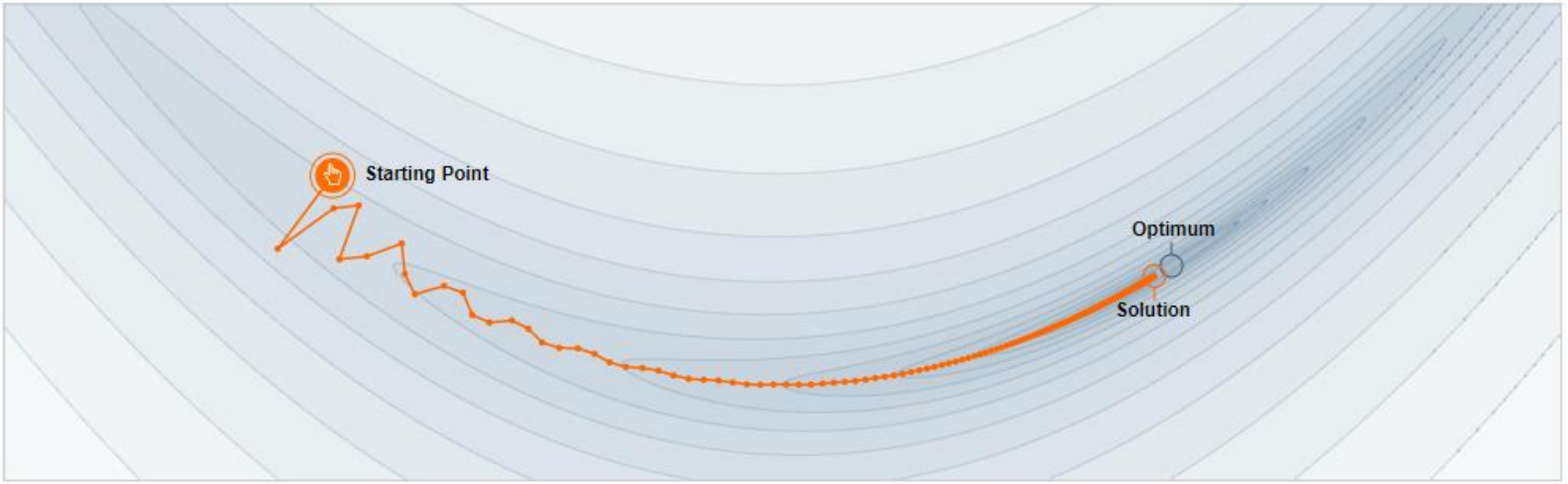
With momentum, two consecutive updates are smooth.



Step-size α = 0.0041

Momentum β = 0.0

We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be



Step-size α = 0.0041

0 0.003 0.006

Momentum β = 0.73

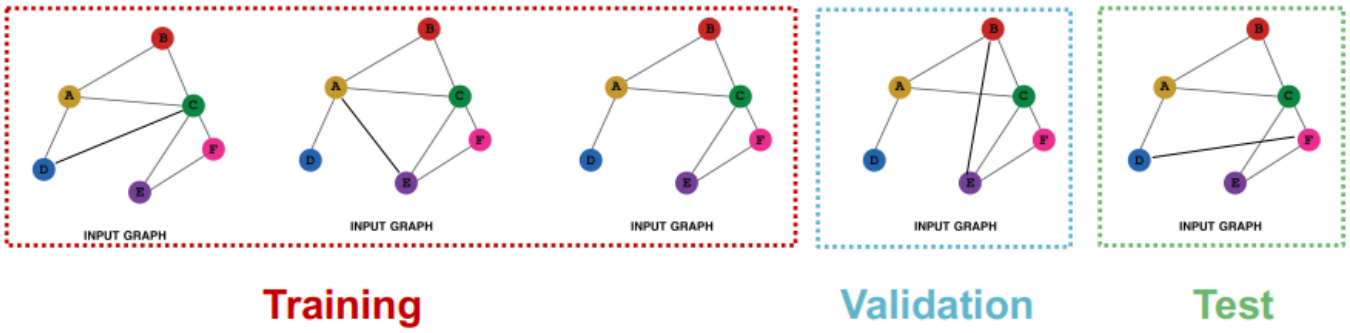
0.00 0.500 0.990

We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

- Other details
  - More advanced gradient methods (adaptive methods):
    - ADAM
      - https://arxiv.org/abs/1412.6980
    - LARS
      - https://arxiv.org/abs/1708.03888

GCN training

- Determine hyper-parameters:
  - Hyper-parameters:
    - network architectures (e.g, #layers),
    - network training iterations,
    - learning rate, etc.
  - Use a validation set to determine hyper-parameters.
    - Test the model with different hyper-parameters on the validation set, choose the setting with the best performance



- Further readings:
  - Deep learning basics
    - ConvNet, Batch normalization, residual/skip connections, optimizer (ADAM, momentum SGD), up/down-sampling, pooling, …
    - http://cs231n.stanford.edu/
    - https://atcold.github.io/pytorch-Deep-Learning/
  - Transformers
    - Another popular network architecture that can be used on graphs
    - http://web.stanford.edu/class/cs224w/
  - Other advanced topics on GNN
    - https://github.com/AntonioLonga/PytorchGeometricTutorial
    - http://web.stanford.edu/class/cs224w/
    - https://medium.com/stanford-cs224w
    - http://web.stanford.edu/class/cs224w/
      - GraphSAGE, Graph attention network

- Online implementation
  - PyTorch Geometric
    - https://www.pyg.org/
    - PyG is a library built upon PyTorch to easily write and train Graph Neural Networks for a wide range of applications
    - https://pytorch-geometric.readthedocs.io/en/latest/notes/resources.html
    - https://pytorch-geometric.readthedocs.io/en/latest/notes/colabs.html
  - Deep graph library
    - https://github.com/dmlc/dgl

- Graph classification datasets:
  - https://pytorch-geometric.readthedocs.io/en/latest/notes/colabs.html
  - https://chrsmrrs.github.io/datasets/docs/datasets/
  - https://paperswithcode.com/task/node-classification