Graph Neural Networks

Introduction

Iulia Duta

Andrei Nicolicioiu

Bitdefender

July 2021

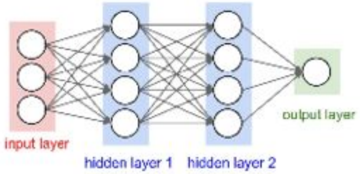
Human Pose Recovery and Behavior Analysis Group
University of Barcelona

Choose your model

Bitdefender

UNSTRUCTURED

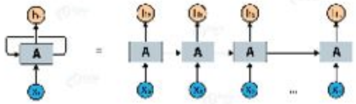
Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	[illegible]	Iris setosa
4.5	3.0	1.4	0.2	Iris setosa
7.0	3.2	4.7	1.4	Iris versicolor
6.4	3.2	4.5	1.5	Iris versicolor
6.3	3.3	6.0	2.5	Iris virginica
5.8	3.3	6.0	2.5	Iris virginica



MLP

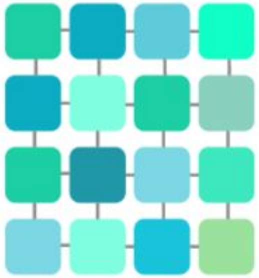
SEQUENTIAL

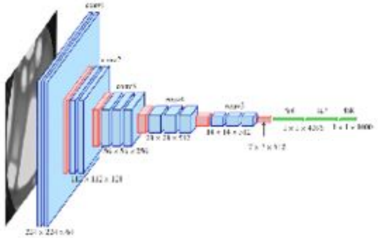
Have a nice day! :)



RNN

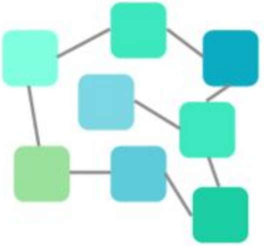
GRID

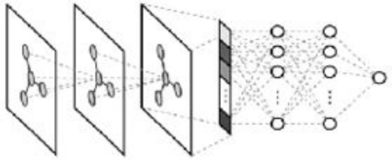




CNN

RELATIONAL STRUCTURE





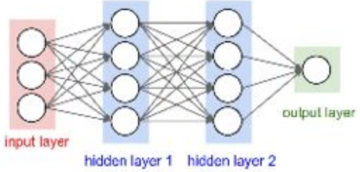
GNN

Choose your model

Bitdefender

UNSTRUCTURED

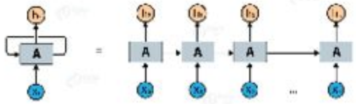
Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	Iris setosa
4.9	3.0	1.4	0.2	Iris setosa
7.0	3.2	4.7	1.4	Iris versicolor
6.4	3.2	4.5	1.5	Iris versicolor
6.3	3.3	6.0	2.5	Iris virginica
5.8	3.3	6.0	2.5	Iris virginica



MLP

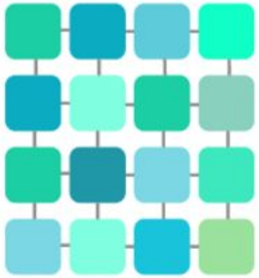
SEQUENTIAL

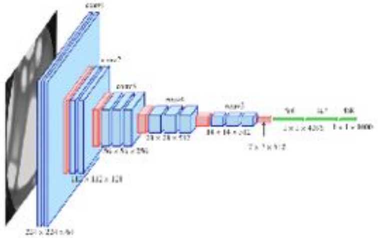
Have a nice day! :)



RNN

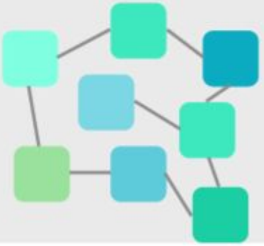
GRID

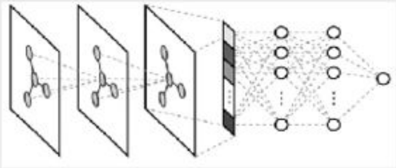




CNN

RELATIONAL STRUCTURE

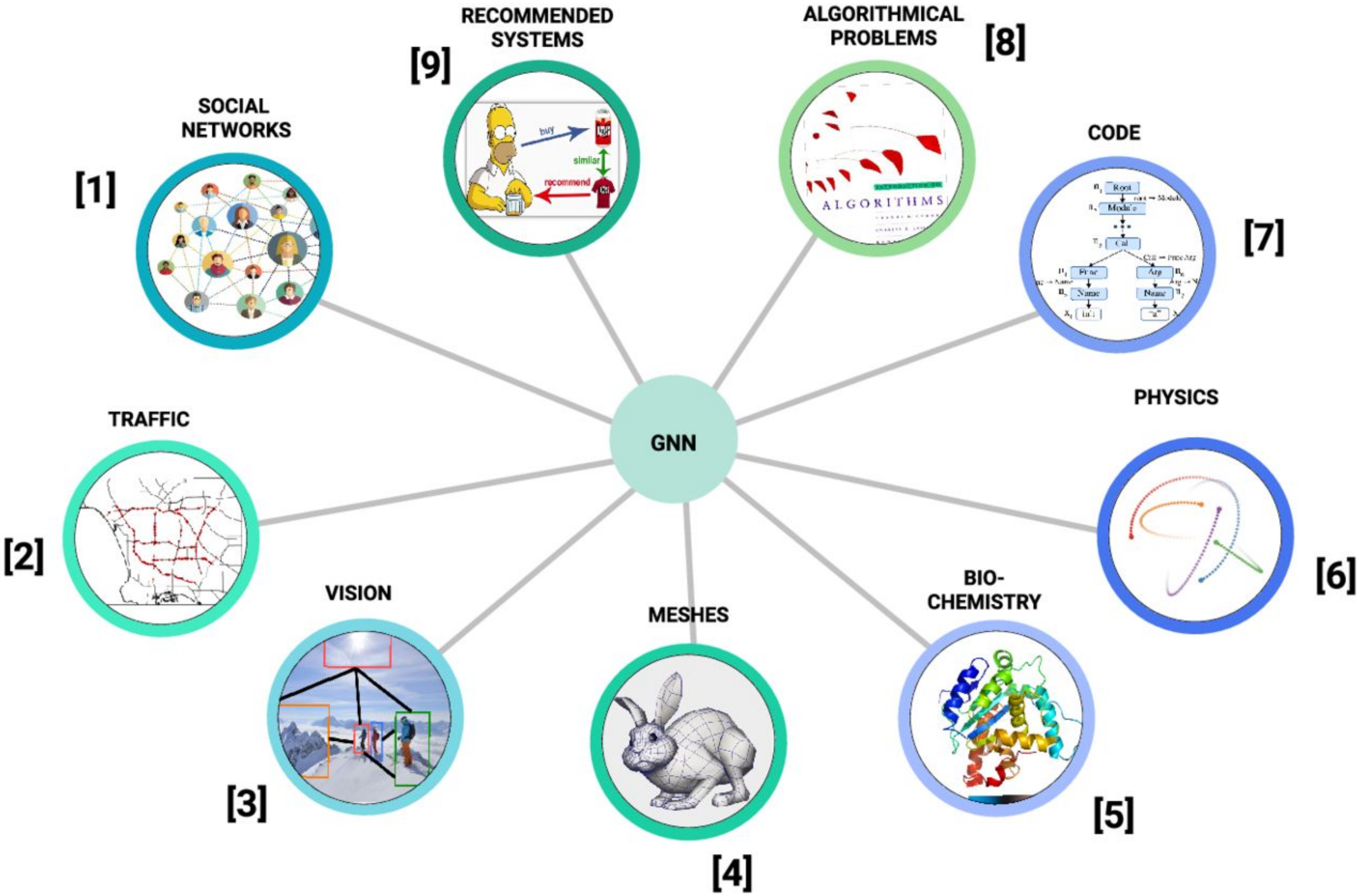




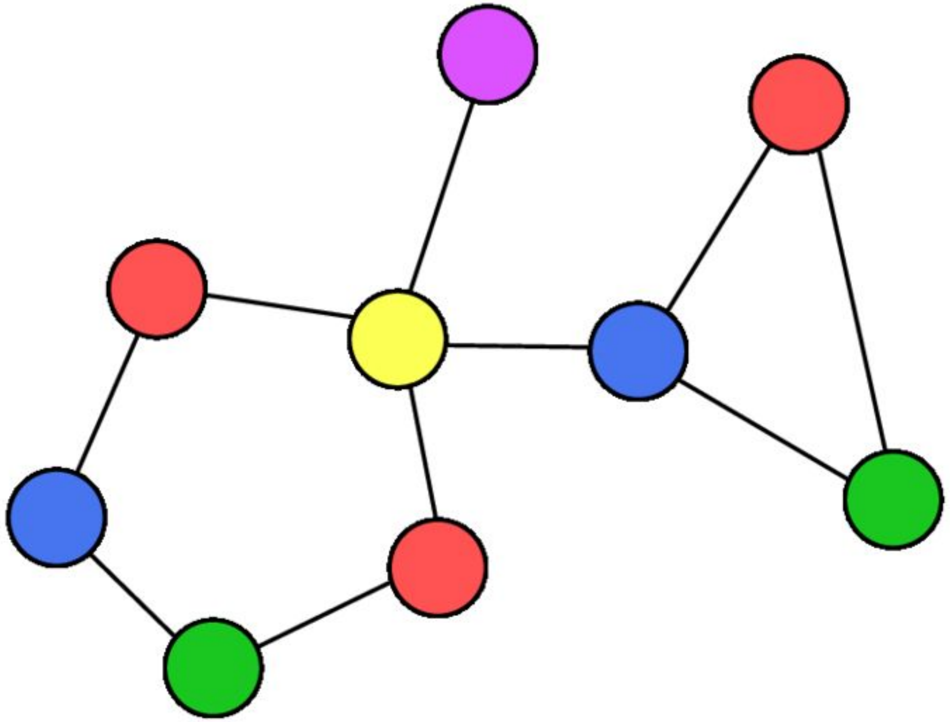
GNN

Tasks

Bitdefender



Data: Graph Structure



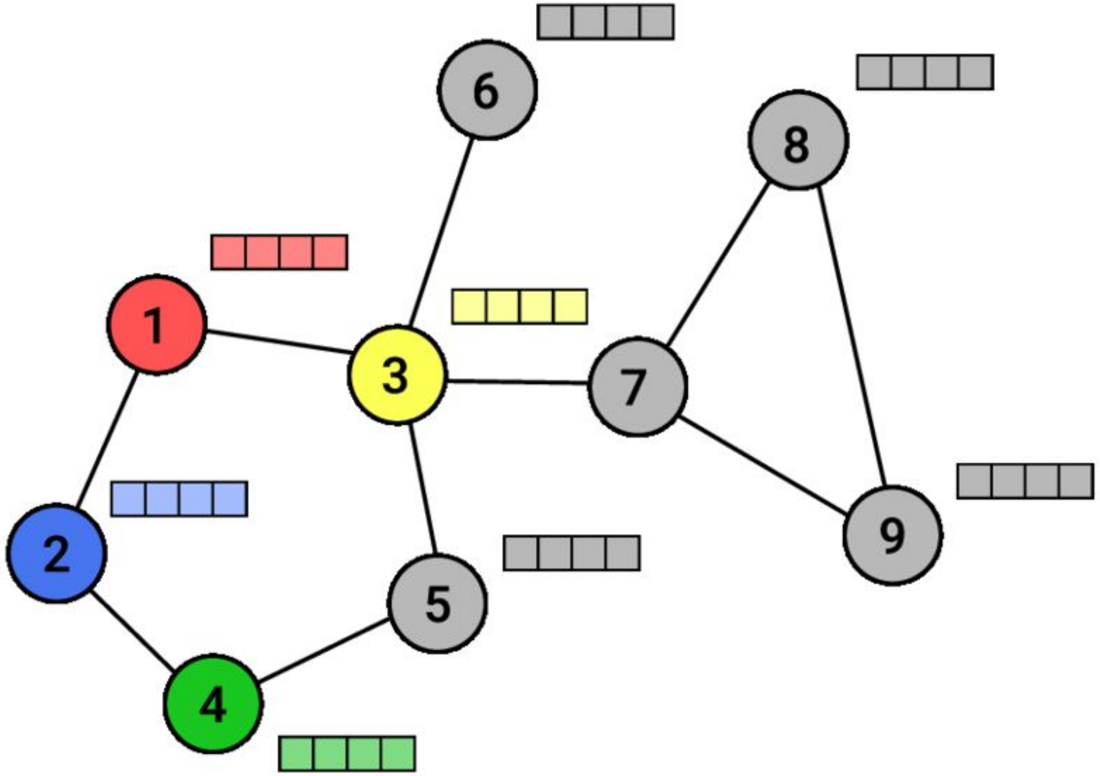
Tasks where we have access or we can create a graph structure.

A graph G is characterized by:

- a set of **nodes**
  $X = \{x_i | i \in 1..N\}$
- connected by **edges**
  $\mathcal{E} = \{e_{ij} | i, j \in 1..N\}$

Data: Graph Structure

Bitdefender



Tasks where we have access or we can create a graph structure.

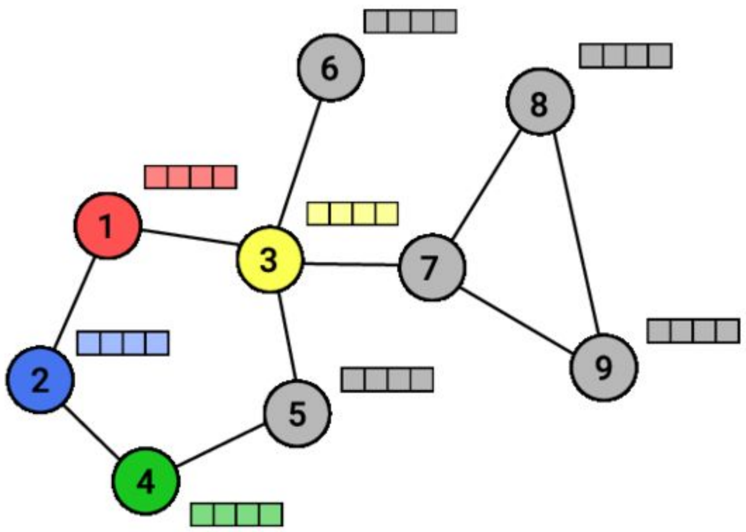
A graph G is characterized by:

- a set of **nodes** $X = \{x_i | i \in 1..N\}$
- connected by **edges** $\mathcal{E} = \{e_{ij} | i, j \in 1..N\}$

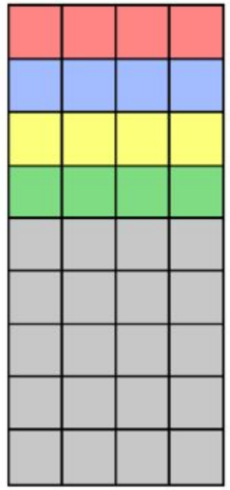
Each node i is characterized by a set of features $x_i \in \mathbb{R}^D$

Data: Graph Structure - Nodes

Bitdefender

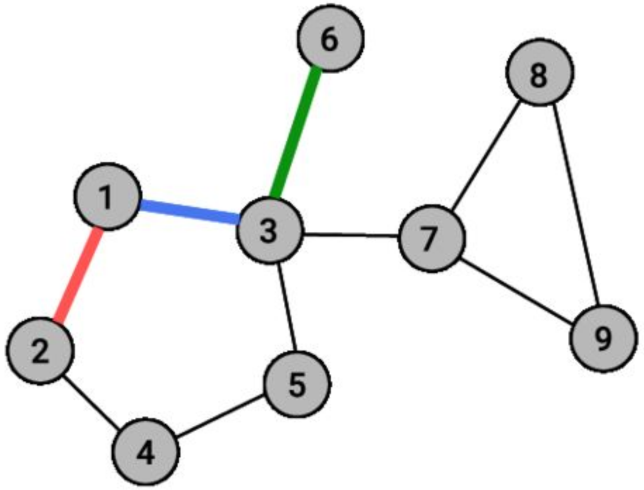


$X \in \mathbb{R}^{N \times D}$



- all the nodes $x_i \in \mathbb{R}^D$ are stacked into a matrix $X \in \mathbb{R}^{N \times D}$
- each row corresponds to a node $x_i \in \mathbb{R}^D$

Data: Graph Structure - Edges



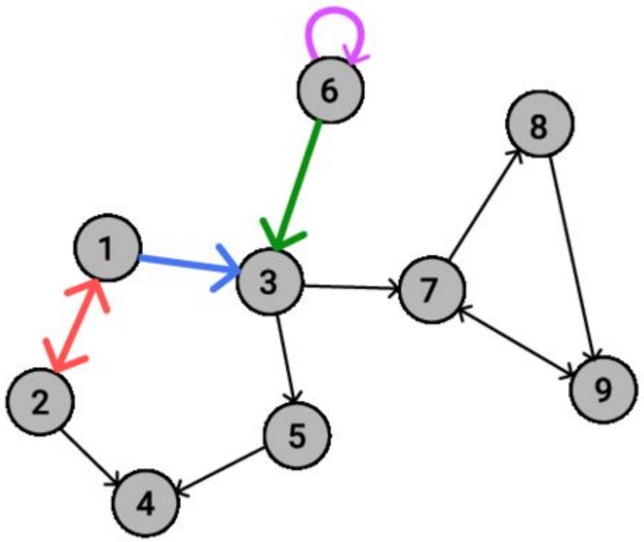
$$A \in \mathbb{R}^{N \times N}$$

	1	2	3	4	5	6	7	8	9
1	0	1	1	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0
3	1	0	0	0	1	1	1	0	0
4	0	1	0	0	1	0	0	0	0
5	0	0	1	1	0	0	0	0	0
6	0	0	1	0	0	0	0	0	0
7	0	0	1	0	0	0	0	1	1
8	0	0	0	0	0	0	1	0	1
9	0	0	0	0	0	0	1	1	0

- the edges $\mathcal{E}$ could be represented by an adjacency matrix $A \in \mathbb{R}^{N \times N}$
- $a_{ij} \neq 0$ if there is an edge between node i and node j

Data: Graph Structure - Edges

Bitdefender



$$A \in \mathbb{R}^{N \times N}$$

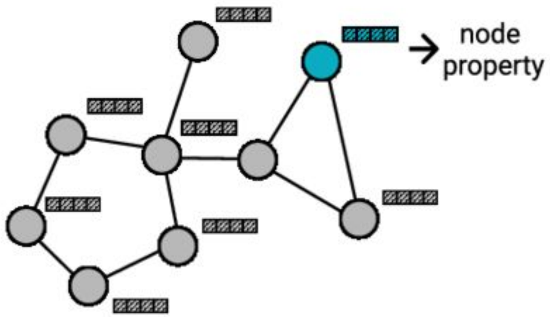
	1	2	3	4	5	6	7	8	9
1	0	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0
3	1	0	0	0	0	1	0	0	0
4	0	1	0	0	1	0	0	0	0
5	0	0	1	1	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	0	1
8	0	0	0	0	0	0	1	0	0
9	0	0	0	0	0	0	1	1	0

- un-directed graph: adjacency matrix is symmetric
- directed graph: adjacency matrix is **not** symmetric
- $a_{ij} \neq 0$ if there is an edge **from j to i**
- a graph could contain *self-loops*

GNNs Goal

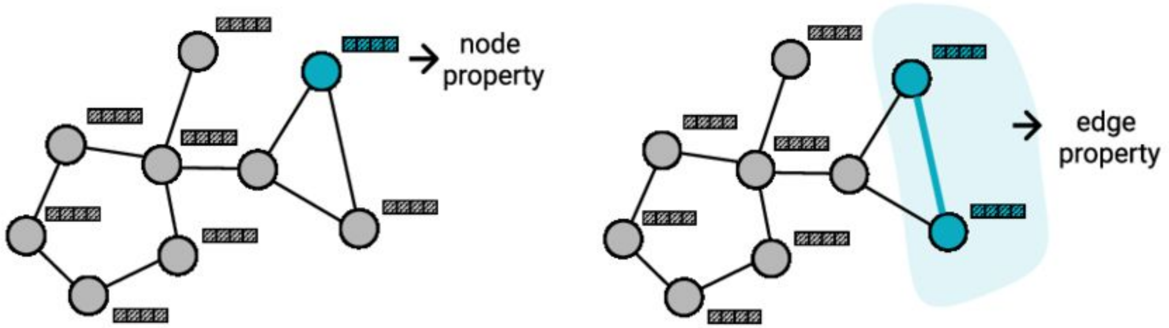
Bitdefender

- Based on the node features (X) and the graph structure (A), we want to learn a representation of the graph.
- Depending on the task, the representation could be:
    1. node level: $Y \in \mathbb{R}^{N \times K}$



GNNs Goal

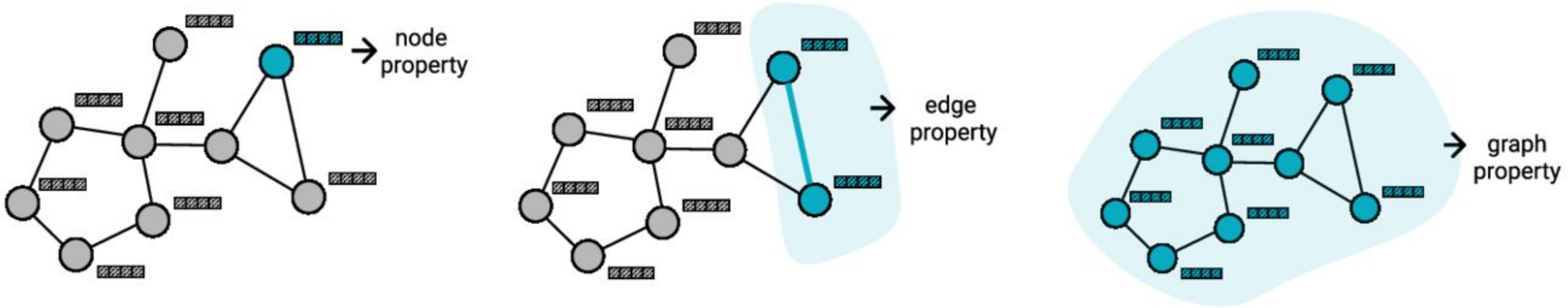
- Based on the node features (X) and the graph structure (A), we want to learn a representation of the graph.
- Depending on the task, the representation could be:
  1. node level: $Y \in \mathbb{R}^{N \times K}$
  2. edge level: $Y \in \mathbb{R}^{M \times K}$



GNNs Goal

Bitdefender

- Based on the node features (X) and the graph structure (A), we want to learn a representation of the graph.
- Depending on the task, the representation could be:
  1. node level: $Y \in \mathbb{R}^{N \times K}$
  2. edge level: $Y \in \mathbb{R}^{M \times K}$
  3. graph level: $Y \in \mathbb{R}^{K}$



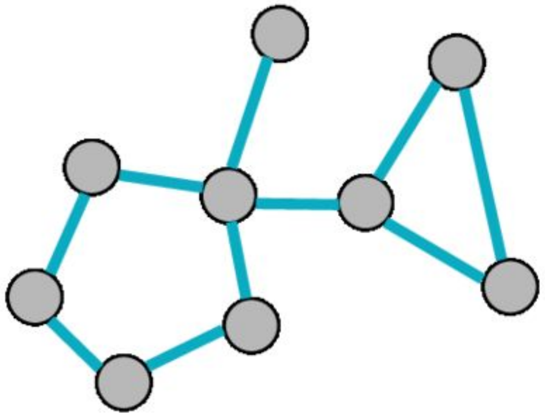
Properties: structure

Bitdefender

Structure - dependent
the processing should take into account the structure of the graphs

1. the processing should take into account how nodes are connected

CONNECTIVTY

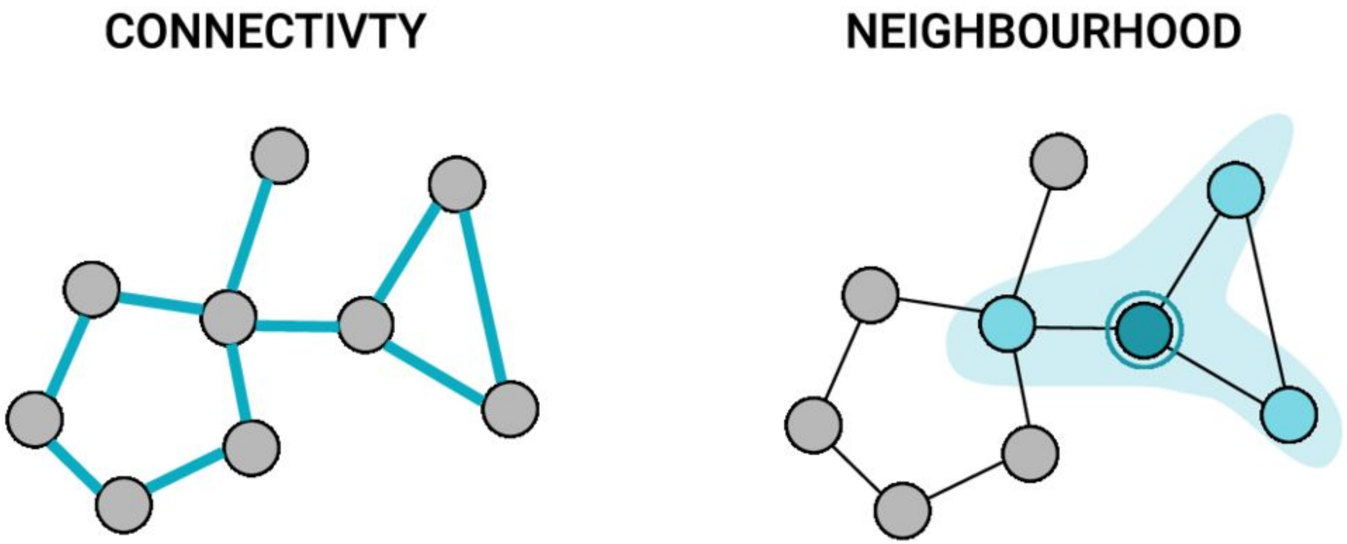


Properties: structure

Bitdefender

Structure - dependent
the processing should take into account the structure of the graphs

1. the processing should take into account how nodes are connected
2. a node should be influenced more by its neighbours



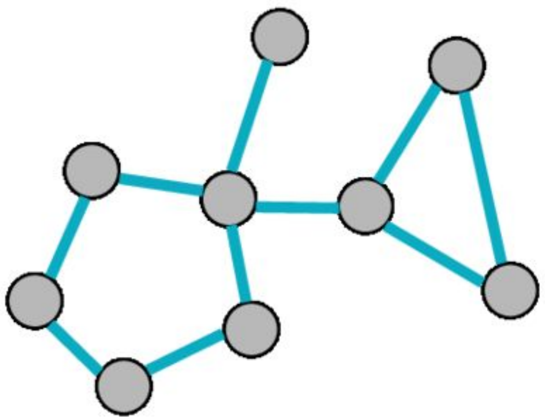
Properties: structure

Bitdefender

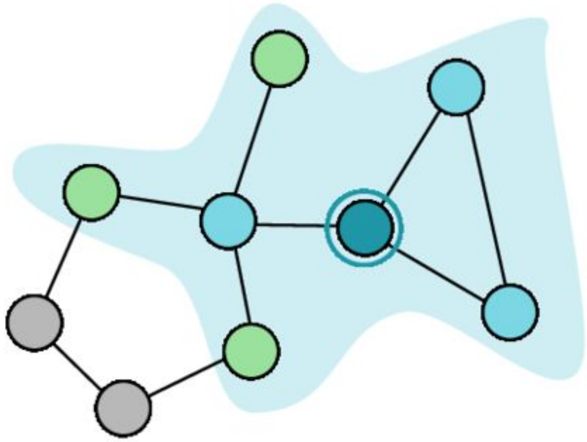
Structure - dependent
the processing should take into account the structure of the graphs

1. the processing should take into account how nodes are connected
2. a node should be influenced more by its neighbours

CONNECTIVTY



NEIGHBOURHOOD



Properties: permutation invariance and equivariance

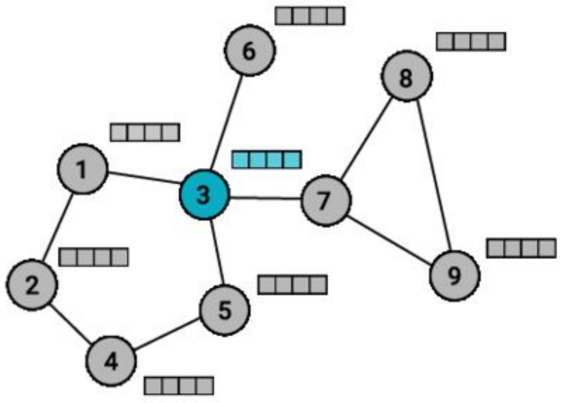
Bitdefender

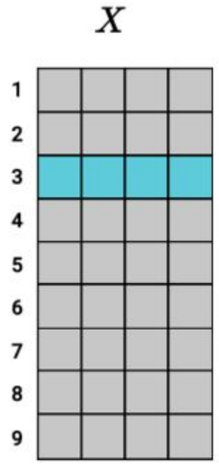
There is no canonical order for the nodes of the graph.

Permutation invariance

The global output of the graph processing should be invariant to the order of the nodes.

$$f(PX, PAP') = f(X, A)$$





A

	1	2	3	4	5	6	7	8	9
1	0	1	1	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0
3	1	0	0	0	1	1	1	0	0
4	0	1	0	0	1	0	0	0	0
5	0	0	1	1	0	0	0	0	0
6	0	0	1	0	0	0	0	0	0
7	0	0	1	0	0	0	0	1	1
8	0	0	0	0	0	0	1	0	1
9	0	0	0	0	0	0	1	1	0

Y

0.5	0.3	0.2

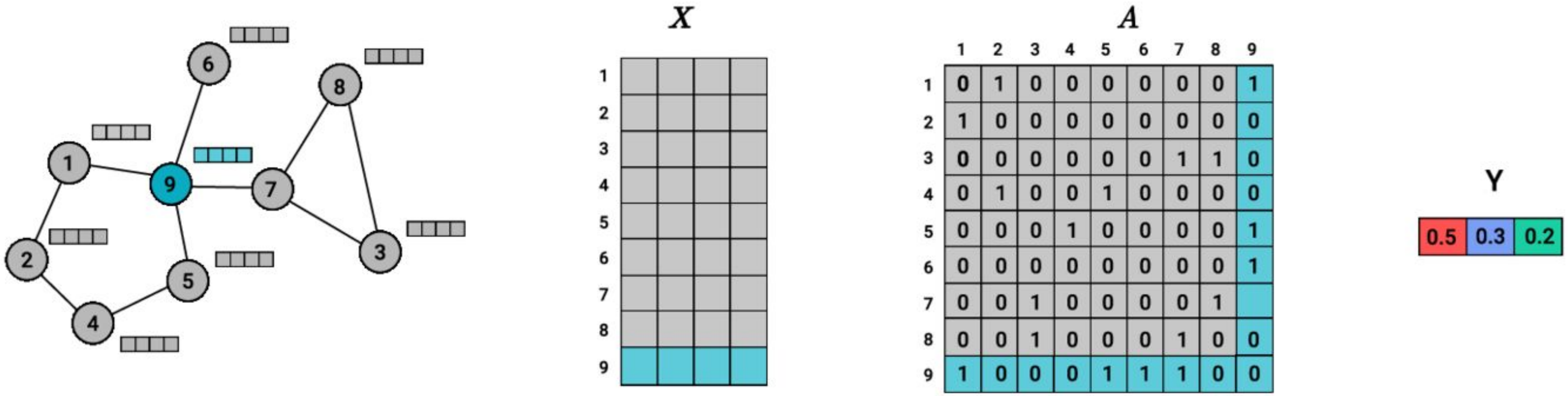
Properties: permutation invariance and equivariance

There is no canonical order for the nodes of the graph.

Permutation invariance

The global output of the graph processing should be invariant to the order of the nodes.

$$f(PX, PAP') = f(X, A)$$



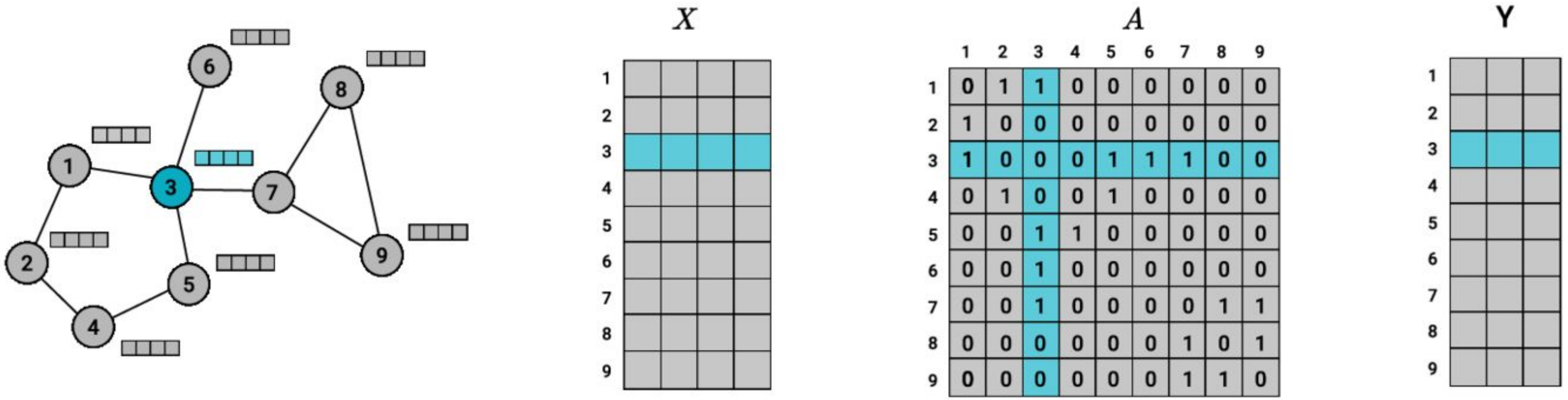
Properties: permutation invariance and equivariance

There is no canonical order for the nodes of the graph.

Permutation equivariance

If we permute the input nodes of the graph, the nodes' output should be permuted in the same way.

$$f(PX, PAP') = Pf(X, A)$$



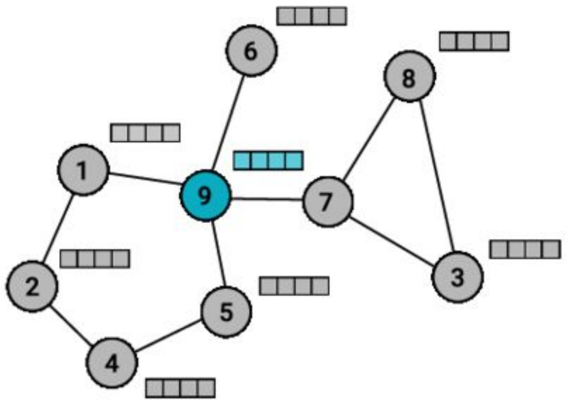
Properties: permutation invariance and equivariance

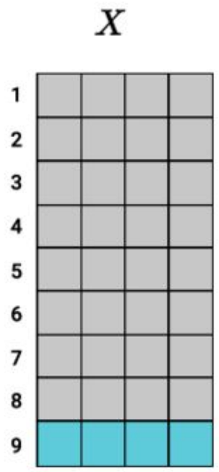
There is no canonical order for the nodes of the graph.

Permutation equivariance

If we permute the input nodes of the graph, the nodes' output should be permuted in the same way.

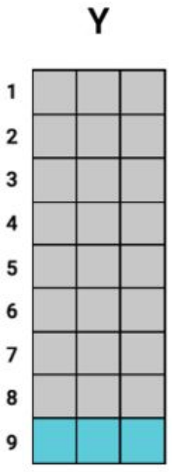
$$f(PX, PAP') = Pf(X, A)$$





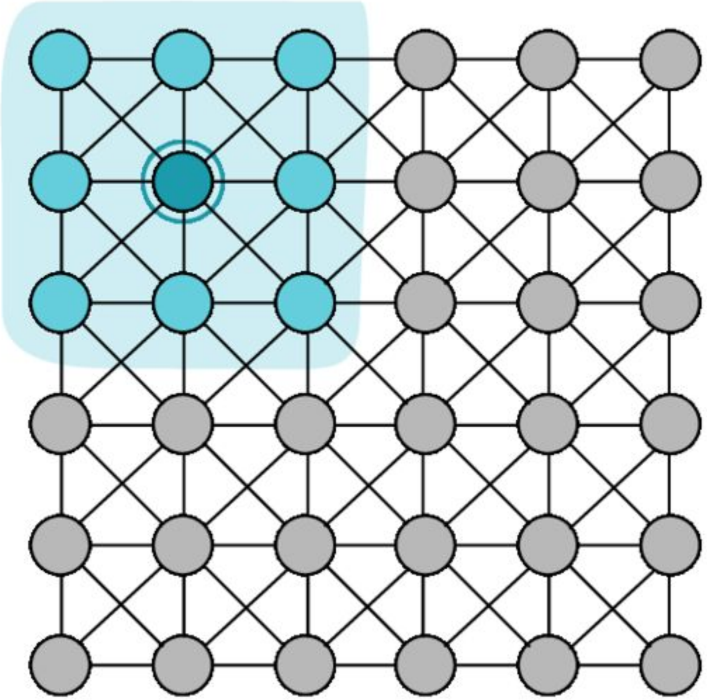
A

	1	2	3	4	5	6	7	8	9
1	0	1	0	0	0	0	0	0	1
2	1	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	0	0
5	0	0	0	1	0	0	0	0	1
6	0	0	0	0	0	0	0	0	1
7	0	0	1	0	0	0	0	1	1
8	0	0	1	0	0	0	1	0	0
9	1	0	0	0	1	1	1	0	0



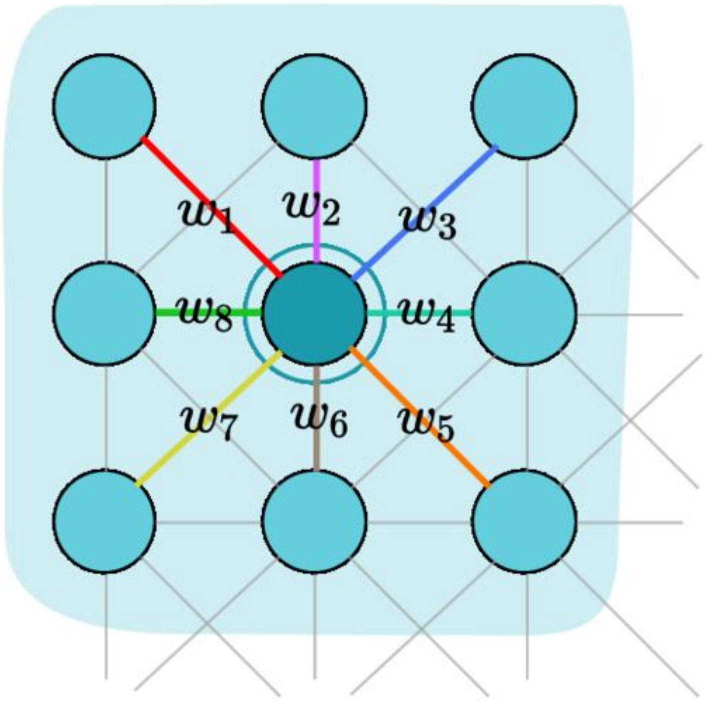
Convolutional Network

Bitdefender



- takes into account a **neighbourhood**
- the **structure is fixed**: a grid for 2D Conv or a sequence for 1D Conv
- the model is invariant to translations

Convolutional Network



$$y_i = \sum_{j \in \mathcal{N}_i} w_j x_j$$

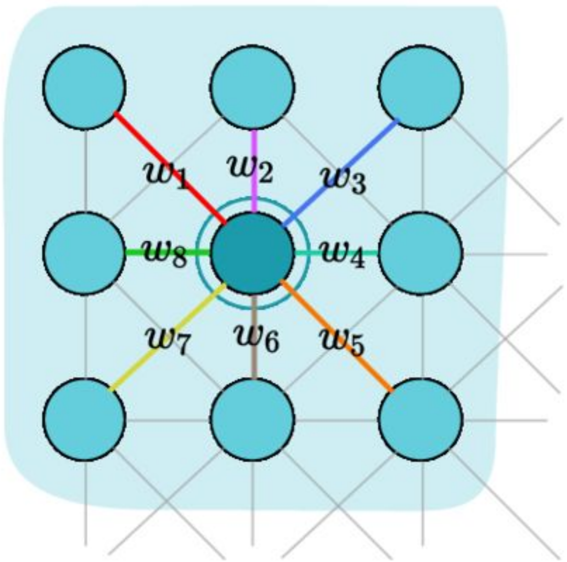
For a convolutional network the neighbourhood is

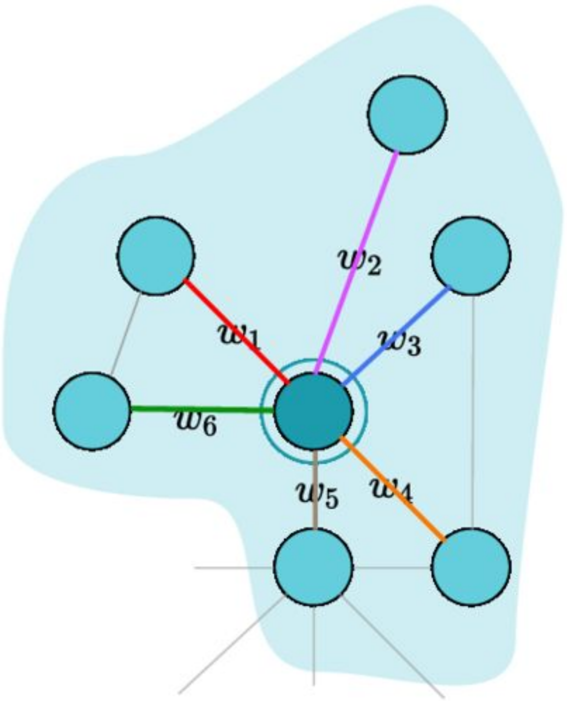
- **fixed**: for a $K \times K$ convolutional filter we combine exactly K^2 neighbours
- **ordered**: we can impose a canonical order among neighbours (left, right, up, down)

Convolutional Network

$$y_i = \sum_{j \in \mathcal{N}_i} w_j x_j$$

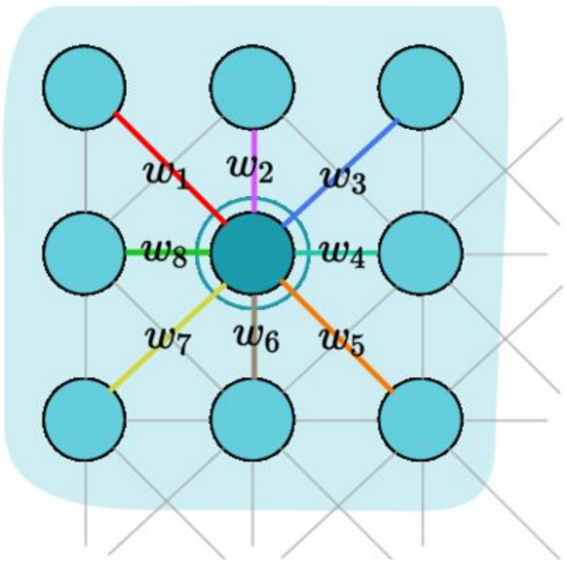
Can we do the same for graphs?



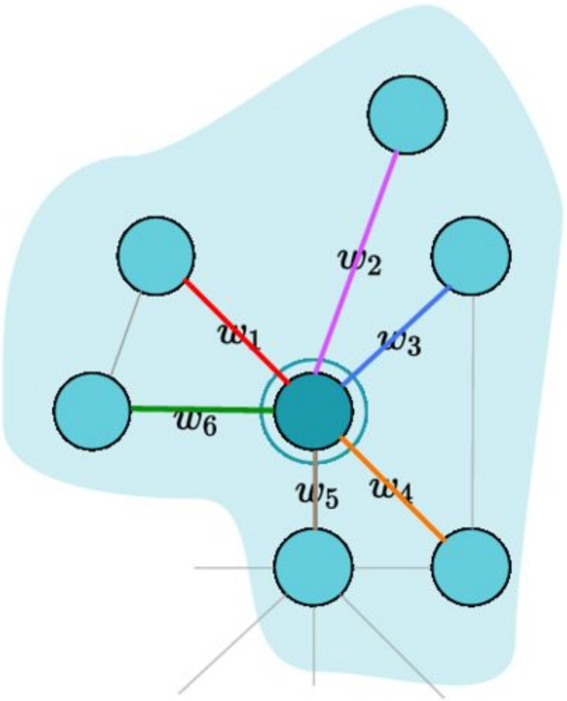


Convolutional Network

$$y_i = \sum_{j \in \mathcal{N}_i} w_j x_j$$



$$y_i = \sum_{j \in \mathcal{N}_i} w_j x_j$$



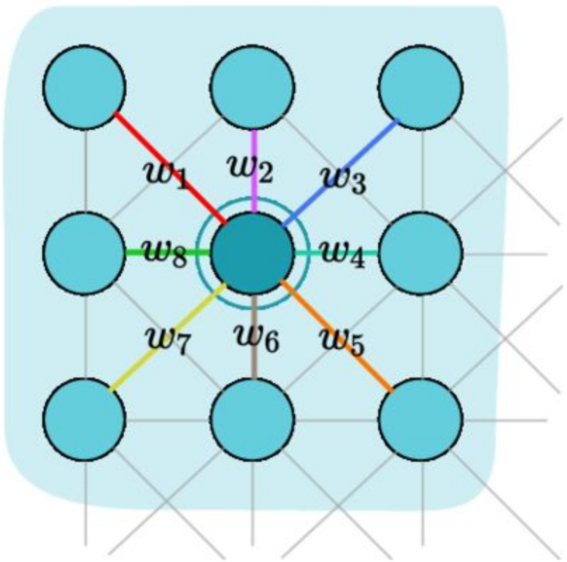
- can't have variable number of weights
- have to establish an order

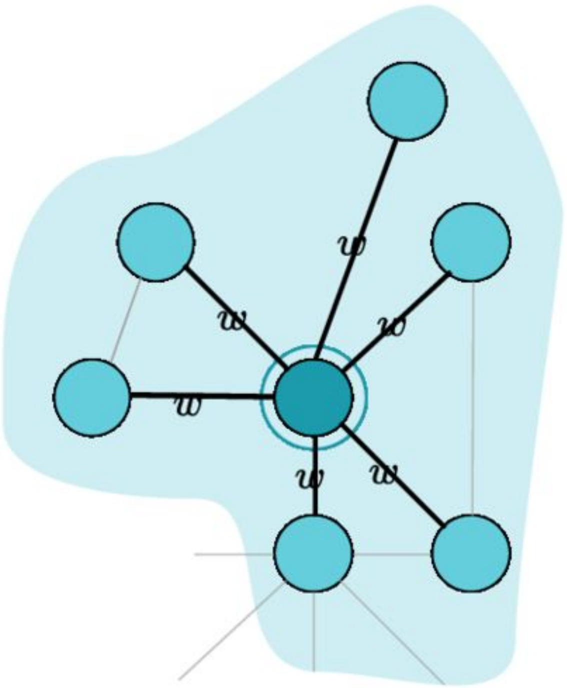
Convolutional Network

$$y_i = \sum_{j \in \mathcal{N}_i} w_j x_j$$

$$y_i = \sum_{j \in \mathcal{N}_i} w \, x_j$$

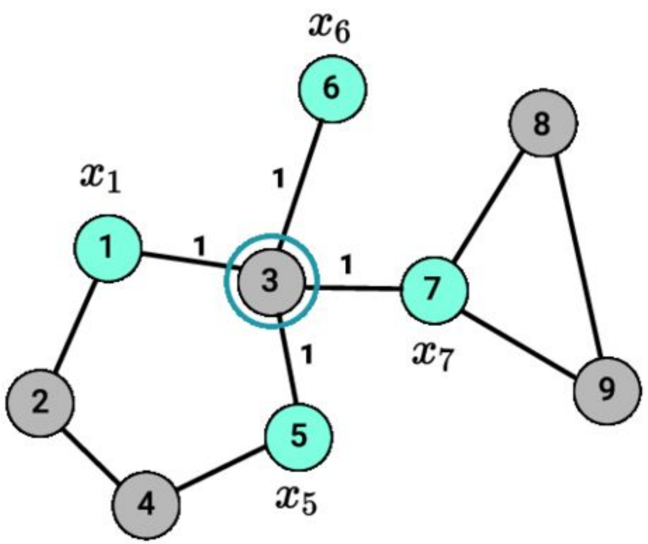
- Solution: same w for all nodes





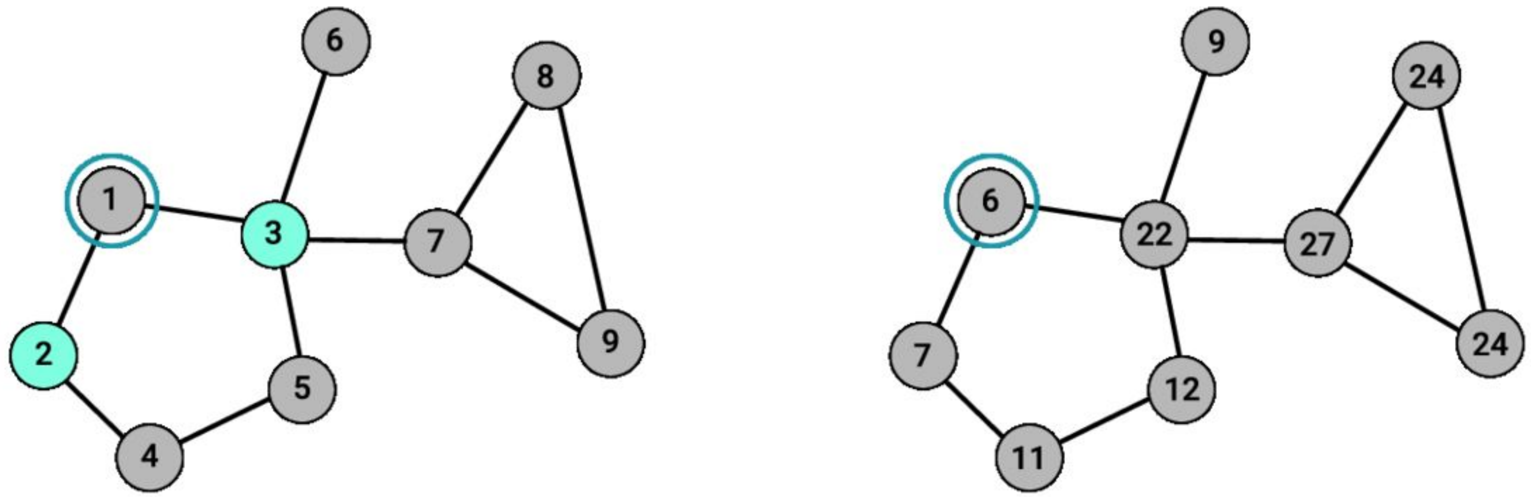
Graph Propagation

Simple graph representation (set $w = 1$): $y_i = x_i + \sum_{j \in \mathcal{N}_i} x_j$



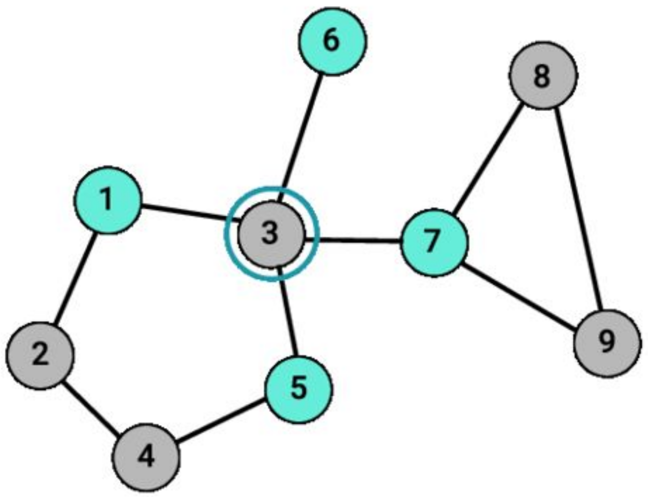
Graph Propagation

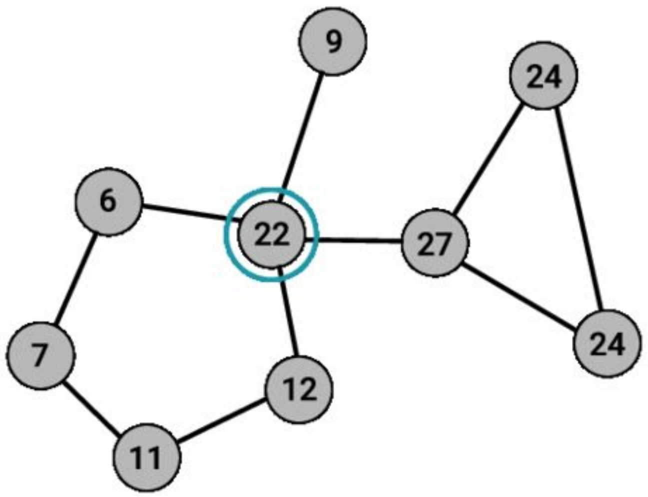
Simple graph representation (set $w = 1$): $y_i = x_i + \sum_{j \in \mathcal{N}_i} x_j$



Graph Propagation

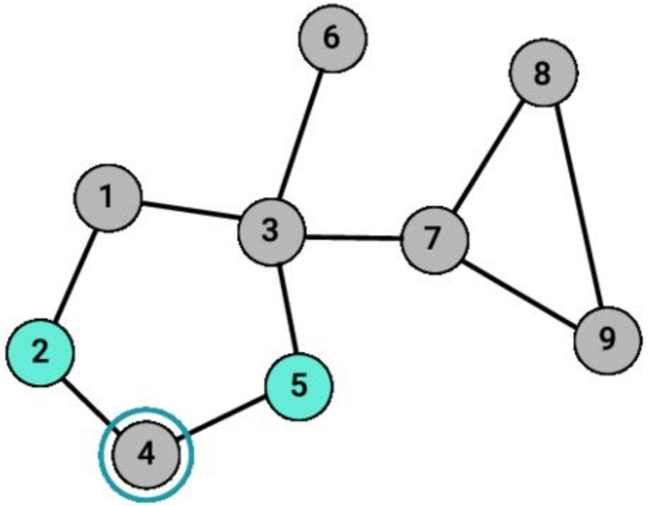
Simple graph representation (set $w = 1$): $$y_i = x_i + \sum_{j \in \mathcal{N}_i} x_j$$

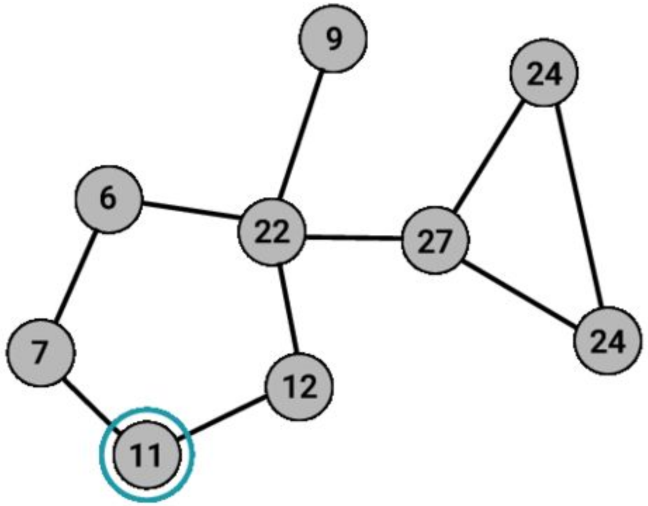




Graph Propagation

Simple graph representation (set $w = 1$): $y_i = x_i + \sum_{j \in \mathcal{N}_i} x_j$

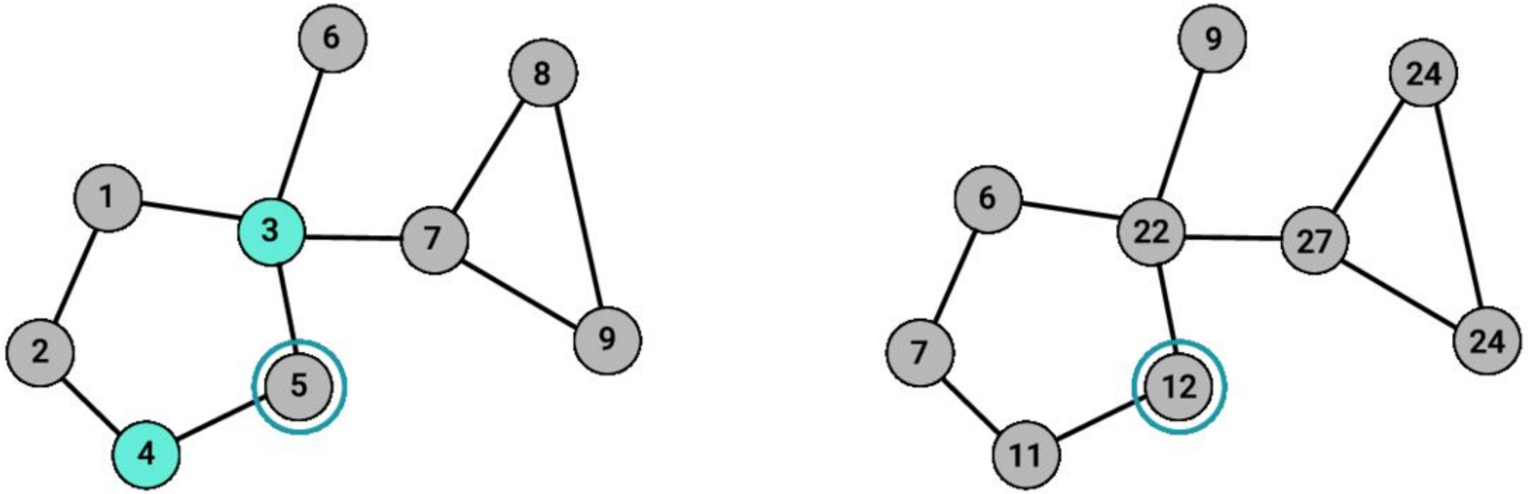




Graph Propagation

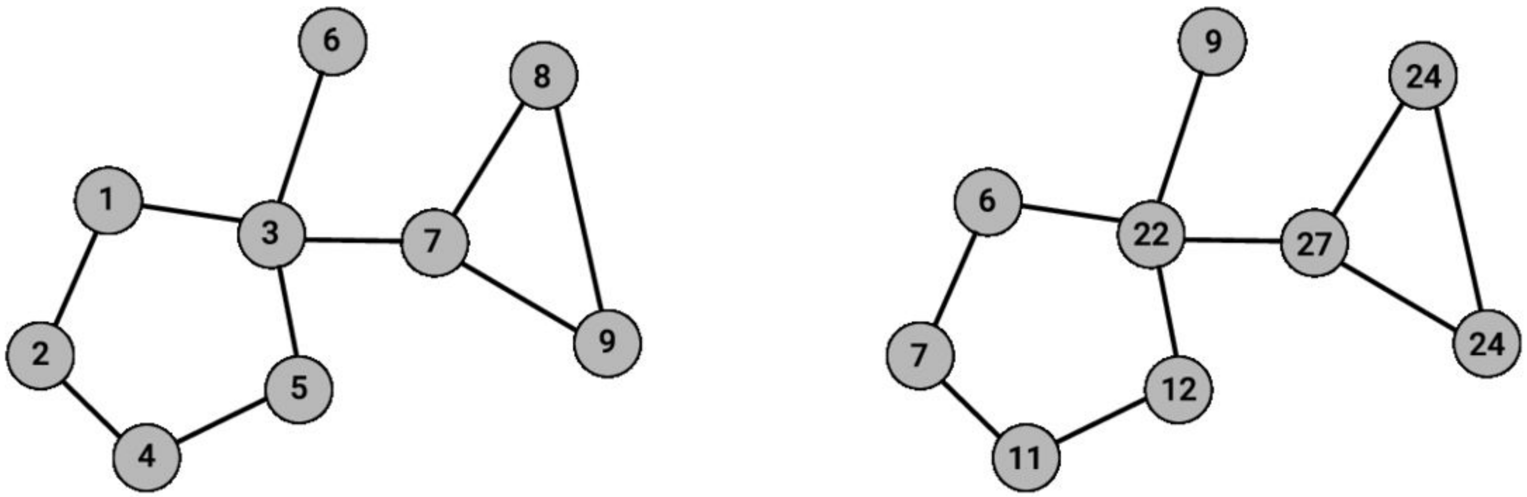
Bitdefender

Simple graph representation (set $w = 1$): $y_i = x_i + \sum_{j \in \mathcal{N}_i} x_j$



Graph Propagation

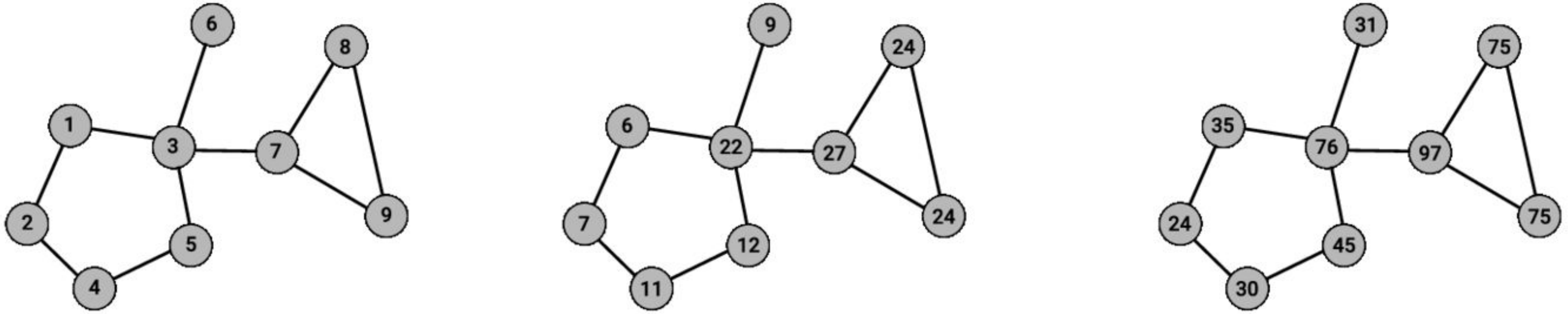
Simple graph representation (set $w = 1$): $y_i = x_i + \sum_{j \in \mathcal{N}_i} x_j$



Graph Propagation

Bitdefender

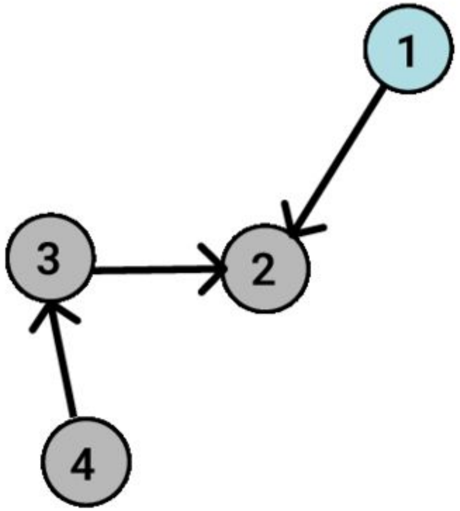
Simple graph propagation (set $w = 1$): $y_i = x_i + \sum_{j \in \mathcal{N}_i} x_j$



- if applied iteratively, it takes into account the structure

Simplest Graph Propagation

$y_i = \sum_{j \in \mathcal{N}_i} x_j$ can be rewritten in a compact, matrix form as $Y = AX$



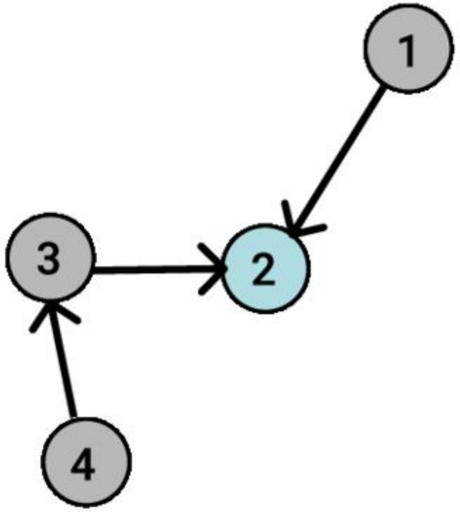
$\boldsymbol{A} \in \mathbb{R}^{\boldsymbol{N} \times \boldsymbol{N}}$ $\quad \boldsymbol{X} \in \mathbb{R}^{\boldsymbol{N}}$ $\quad \boldsymbol{Y} \in \mathbb{R}^{\boldsymbol{N}}$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ \ \\ \ \\ \ \end{bmatrix}$$

Simplest Graph Propagation

Bitdefender

$y_i = \sum_{j \in \mathcal{N}_i} x_j$ can be rewritten in a compact, matrix form as $Y = AX$

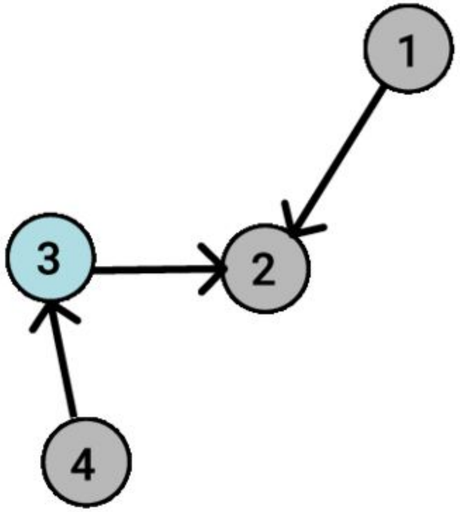


$$\boldsymbol{A} \in \mathbb{R}^{N \times N} \quad \boldsymbol{X} \in \mathbb{R}^{N} \quad \boldsymbol{Y} \in \mathbb{R}^{N}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1+3 \\ \\ \\ \end{bmatrix}$$

Simplest Graph Propagation

$y_i = \sum_{j \in \mathcal{N}_i} x_j$ can be rewritten in a compact, matrix form as $Y = AX$

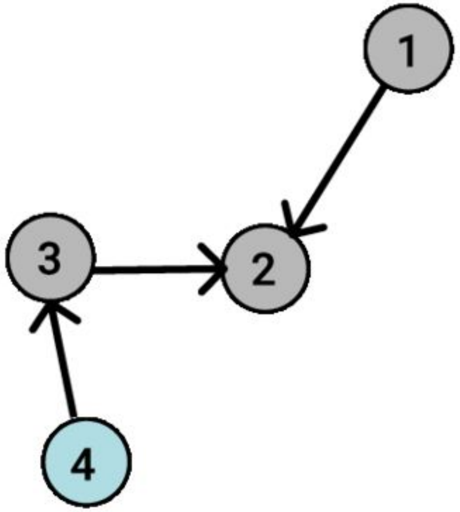


$$\boldsymbol{A} \in \mathbb{R}^{\boldsymbol{N} \times \boldsymbol{N}} \quad \boldsymbol{X} \in \mathbb{R}^{\boldsymbol{N}} \quad \boldsymbol{Y} \in \mathbb{R}^{\boldsymbol{N}}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1+3 \\ 4 \\ \ \end{bmatrix}$$

Simplest Graph Propagation

$y_i = \sum_{j \in \mathcal{N}_i} x_j$ can be rewritten in a compact, matrix form as $Y = AX$

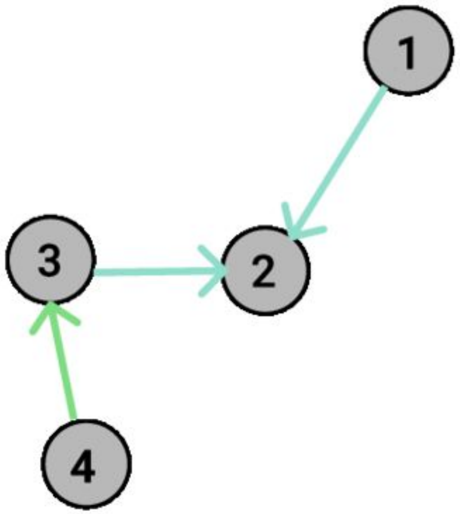


$$\boldsymbol{A} \in \mathbb{R}^{N \times N} \quad \boldsymbol{X} \in \mathbb{R}^{N} \quad \boldsymbol{Y} \in \mathbb{R}^{N}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1+3 \\ 4 \\ 0 \end{bmatrix}$$

Simplest Graph Propagation

$y_i = \sum_{j \in \mathcal{N}_i} x_j$ Nodes could have high-dimensional representation $X \in R^{N \times D}$

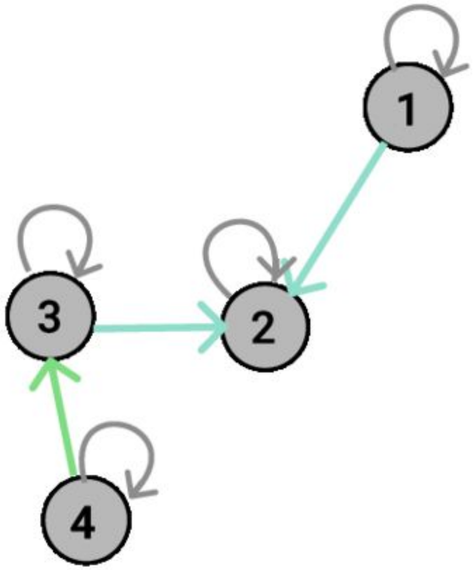


$$\boldsymbol{A} \in \mathbb{R}^{N \times N} \quad \boldsymbol{X} \in \mathbb{R}^{N \times D} \quad \boldsymbol{Y} \in \mathbb{R}^{N \times D}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ x_1 + x_3 \\ x_4 \\ 0 \end{bmatrix}$$

Simplest Graph Propagation

$y_i = x_i + \sum_{j \in \mathcal{N}_i} x_j$ We should take into account also the current node - self-loops.



$A \in \mathbb{R}^{N \times N}$ $X \in \mathbb{R}^{N \times D}$ $Y \in \mathbb{R}^{N \times D}$

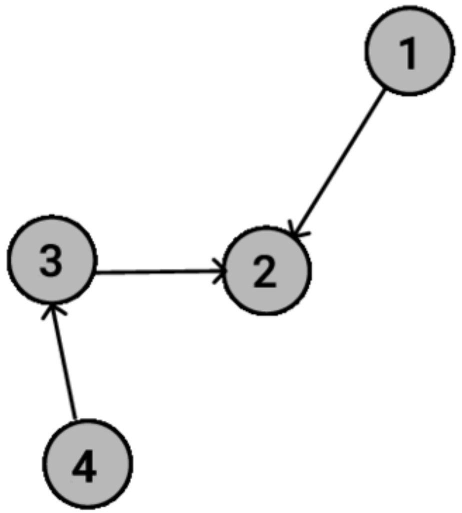
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_1 + x_2 + x_3 \\ x_3 + x_4 \\ x_4 \end{bmatrix}$$

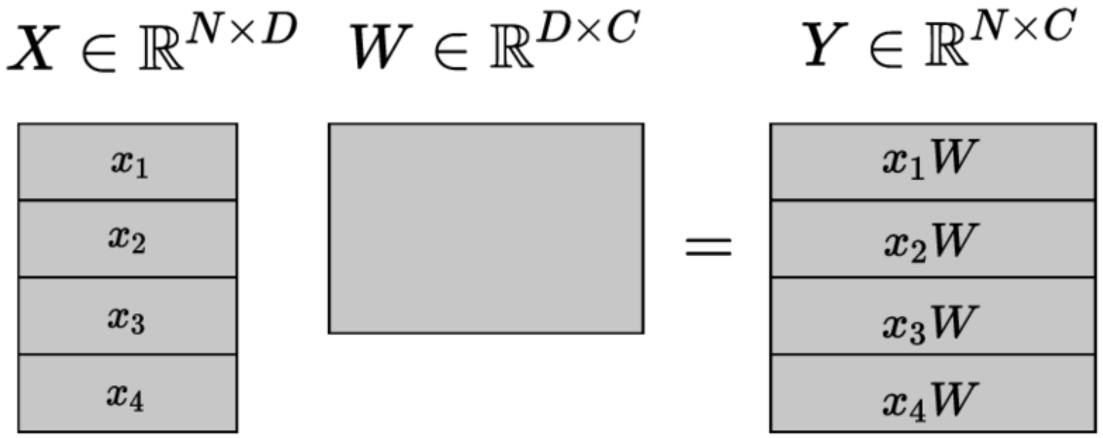
Bitdefender

Simplest Graph Propagation

To combine more complex representations:

$y_i = x_i + \sum_{j \in \mathcal{N}_i} x_j \quad \rightarrow \quad y_i = x_i W + \sum_{j \in \mathcal{N}_i} x_j W$



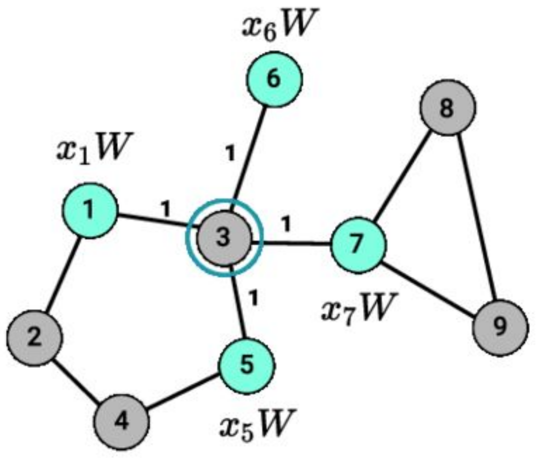


Simplest Graph Propagation

Bitdefender

To combine more complex representations:

$$y_i = x_i + \sum_{j \in \mathcal{N}_i} x_j \qquad \rightarrow \qquad y_i = x_i W + \sum_{j \in \mathcal{N}_i} x_j W$$



The operations performed in the graph could be rewritten as:

$$Y = AXW$$

Iteratively, for more layers:

$$Y = A\sigma(AXW_1)W_2)$$

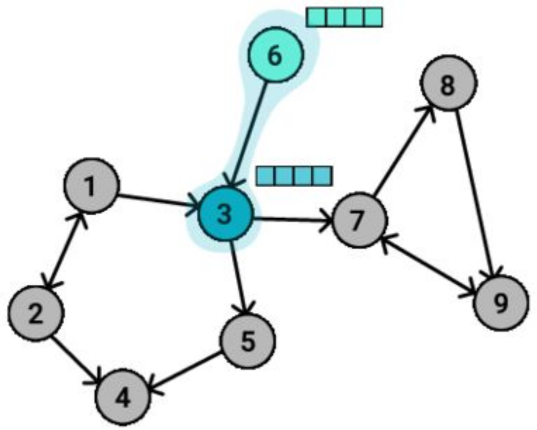
$$Y = A\sigma...A\sigma(AXW_1)W_2)..W_n$$

GNNs: Message Passing Framework - Send

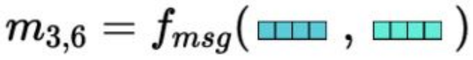
Bitdefender

Send Function

- for each pair of 2 connected nodes, create a **message**



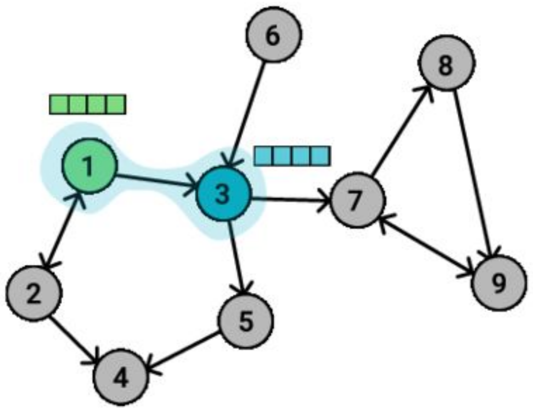
$$m_{ij} = f_{msg}(x_i, x_j) \in \mathbb{R}^C \quad \forall (i,j) \in \mathcal{E}$$



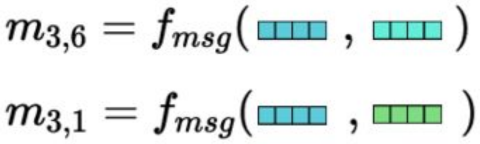
GNNs: Message Passing Framework - Send

Bitdefender

Send Function
- for each pair of 2 connected nodes, create a **message**



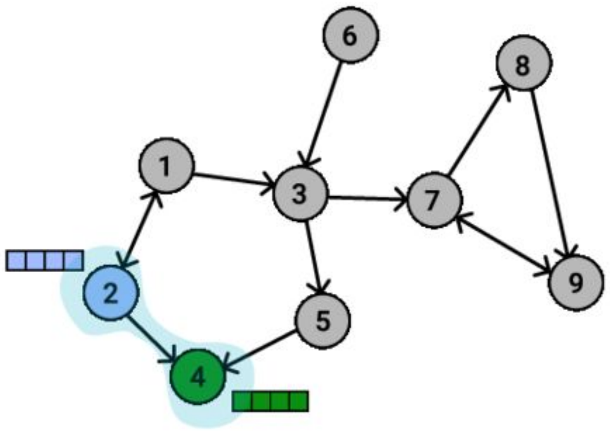
$$m_{ij} = f_{msg}(x_i, x_j) \in \mathbb{R}^C \quad \forall (i,j) \in \mathcal{E}$$



GNNs: Message Passing Framework - Send

Send Function

- for each pair of 2 connected nodes, create a **message**



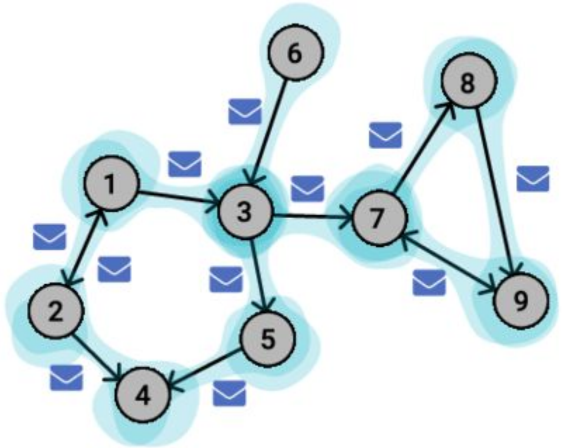
$$m_{ij} = f_{msg}(x_i, x_j) \in \mathbb{R}^C \quad \forall (i,j) \in \mathcal{E}$$

$$m_{3,6} = f_{msg}(\square , \square)$$

$$m_{3,1} = f_{msg}(\square , \square)$$

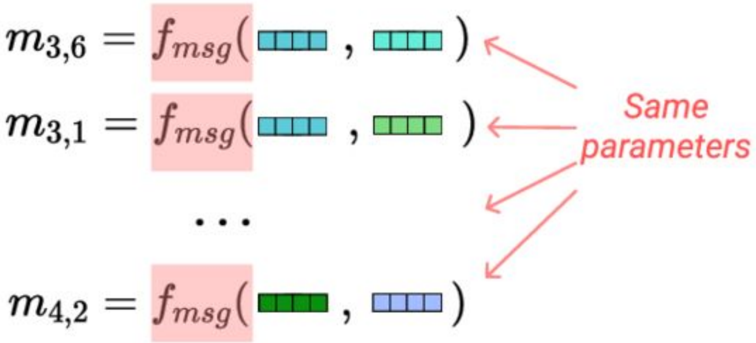
$$m_{4,2} = f_{msg}(\square , \square)$$

GNNs: Message Passing Framework - Send



- f_{msg} is a learnable function (e.g. an MLP)
- its parameters are shared between each pair of nodes

$$m_{ij} = \overbrace{f_{msg}(x_i, x_j)}^{\text{Learnable funcion}} \in \mathbb{R}^C \quad \forall (i,j) \in \mathcal{E}$$

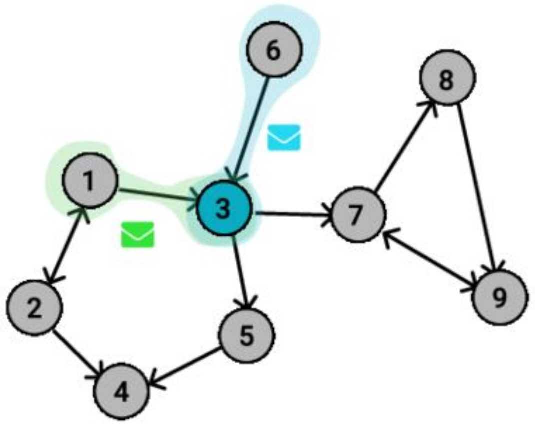


GNNs: Message Passing Framework - Aggregation

Bitdefender

Aggregation Function

For each node i, **aggregate** the incoming messages from all its neighbours.



$$h_i = f_{agg}(\{m_{ij} | \forall j \in \mathcal{N}_i\})$$

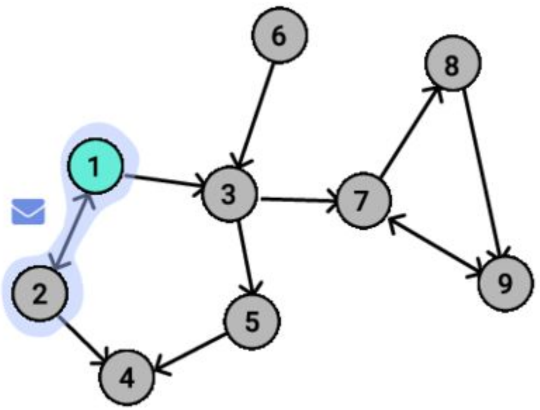
$$h_3 = f_{agg}(\{\text{✉}, \text{✉}\})$$

GNNs: Message Passing Framework - Aggregation

Bitdefender

Aggregation Function

For each node i, **aggregate** the incoming messages from all its neighbours.



$$h_i = f_{agg}(\{m_{ij} | \forall j \in \mathcal{N}_i\})$$

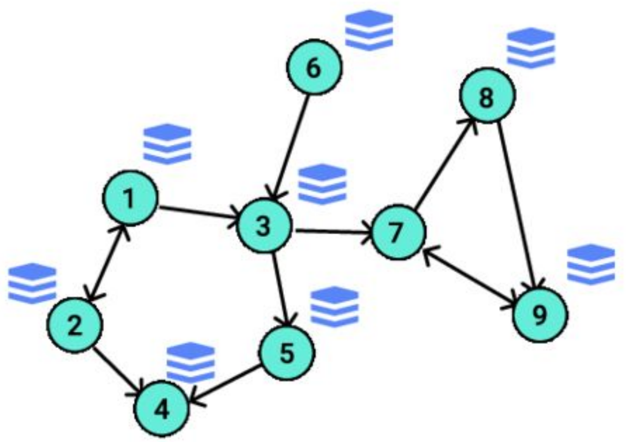
$h_3 = f_{agg}(\{✉, ✉\})$

$h_1 = f_{agg}(\{✉\})$

GNNs: Message Passing Framework - Aggregation

Bitdefender

- aggregate incoming messages with the function f_{agg}: eg. sum, mean, max, min
- it should be **invariant to the order** of the nodes and should **allow a variable number** of messages



$$h_i = \overbrace{f_{agg}}^{\text{operator}} (\{m_{ij} | \forall j \in \mathcal{N}_i\}) \in \mathbb{R}^C$$

$$h_3 = f_{agg}(\{\ ,\ \})$$

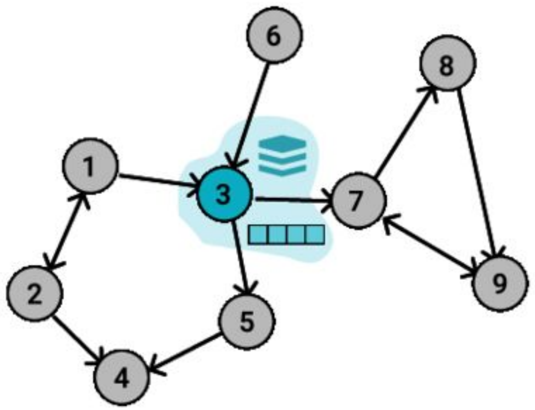
$$\dots$$

$$h_1 = f_{agg}(\{\ \})$$

GNNs: Message Passing Framework - Update

Update Function

For each node i, **update** its representation using the aggregated message.



$$\tilde{x}_i = f_{upd}(x_i, h_i)$$

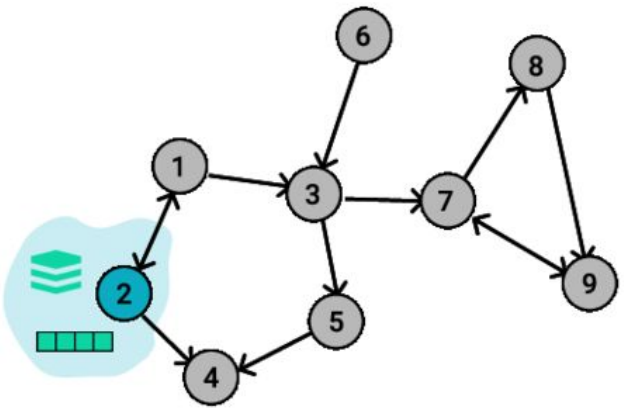
$\tilde{x}_3 = f_{upd}($  ,  $)$

GNNs: Message Passing Framework - Update

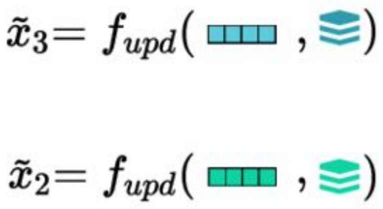
Bitdefender

Update Function

For each node i, **update** its representation using the aggregated message.

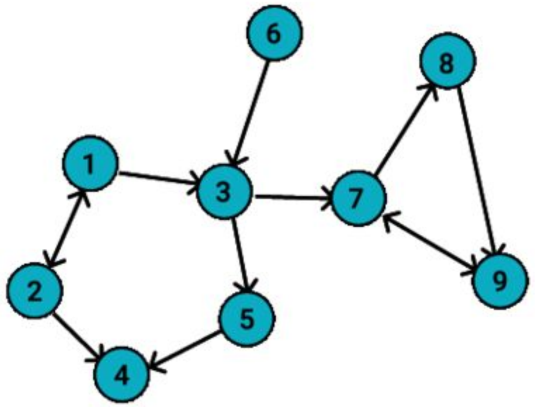


$$\tilde{x}_i = f_{upd}(x_i, h_i)$$



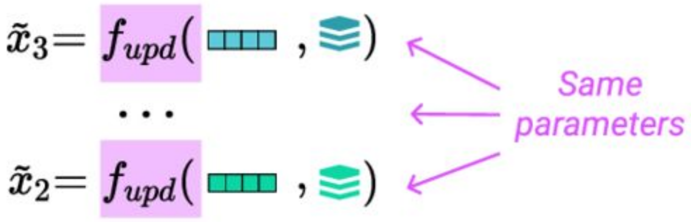
GNNs: Message Passing Framework - Update

- f_{upd} is a learnable function (e.g. an MLP)
- its parameters are shared between all the nodes



Learnable function

$$\tilde{x}_i = \overbrace{f_{upd}(x_i, h_i)}^{\text{Learnable function}} \in \mathbb{R}^C$$

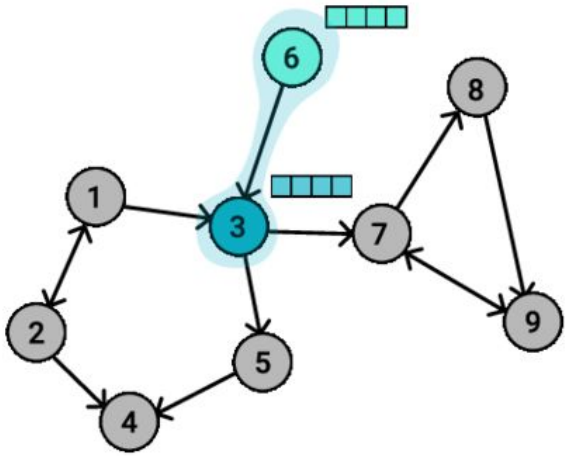


GNNs - Overview

Bitdefender

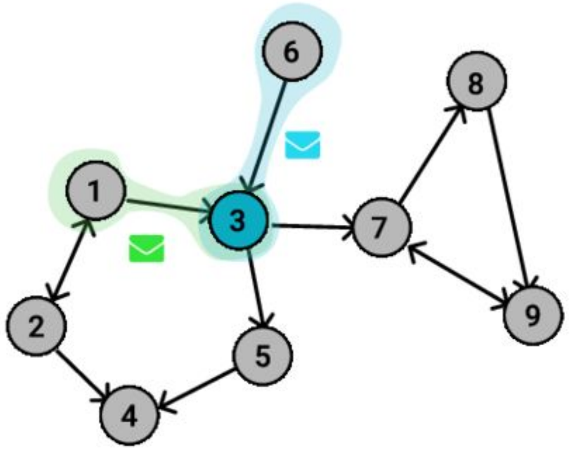
1. Send

$$m_{ij} = f_{msg}(x_i, x_j)$$



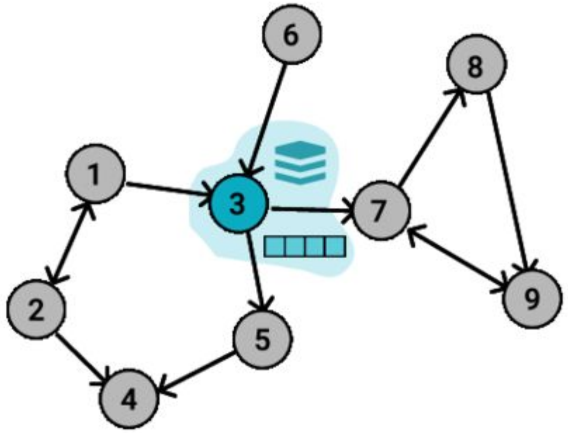
2. Aggregate

$$H_i = f_{agg}(\{m_{ij} | \forall j \in \mathcal{N}_i\})$$



3. Update

$$\tilde{x}_i = f_{upd}(x_i, H_i)$$



General GNN framework

Bitdefender

$$f_{upd}\{x_i,\ f_{agg}\{\ f_{msg}(x_i, x_j)\ |\forall j \in \mathcal{N}_i\}\ \}$$

Depending on how the 3 functions are instantiated, different architectures could be obtained:

Convolutional GNNs

$$f_{upd}(x_i, \bigoplus_{\forall j \in \mathcal{N}_i} \{\phi(x_j)\})$$

Attention GNNs

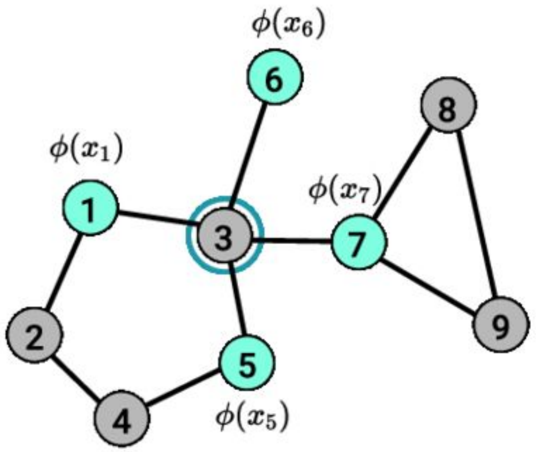
$$f_{upd}(x_i, \bigoplus_{\forall j \in \mathcal{N}_i} \{\alpha(x_i, x_j)\phi(x_j))$$

Message Passing

$$f_{upd}(x_i, \bigoplus_{\forall j \in \mathcal{N}_i} \{\phi(x_i, x_j)\})$$

Graph Convolutional Network

Bitdefender

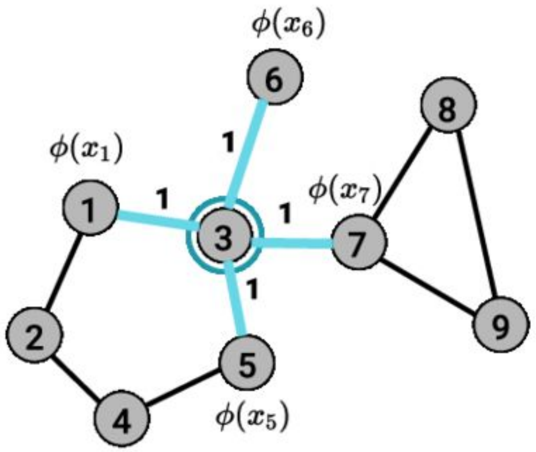


$$y_i = f_{upd}(x_i, \bigoplus_{\forall j \in \mathcal{N}_i} \{ \phi(x_j) \})$$

- messages depend only on the source nodes

[10] *Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. ICLR 2017*

Graph Convolutional Network



$$y_i = f_{upd}(x_i, \bigoplus_{\forall j \in \mathcal{N}_i} \{\phi(x_j)\})$$

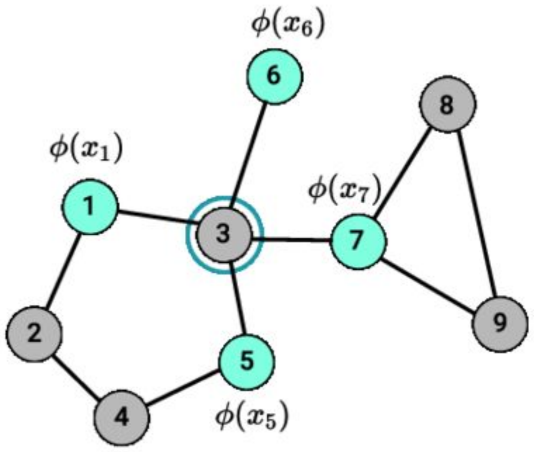
- messages depend only on the source nodes
- aggregation function is implemented as a sum/mean operation
- aggregation could be normalized according to the nodes' degree: $\frac{1}{\sqrt{deg(i)deg(j)}}$

Matrix form: $Y = \sigma(\tilde{A}XW)$

[10] *Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. ICLR 2017*

Graph Attention Network

Bitdefender



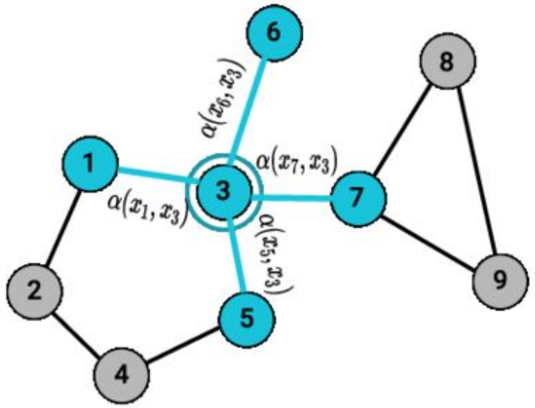
$$y_i = f_{upd}(x_i, \bigoplus_{\forall j \in \mathcal{N}_i} \{\alpha(x_i, x_j)\{\phi(x_j)\}\})$$

- messages depend only on the source nodes

[11] *Vaswani et. al. Attention is all you need. NeurIPS 2017*
[12] *Veličković et. al Graph attention networks. ICLR 2018*

Graph Attention Network

Bitdefender



$$y_i = f_{upd}(x_i, \{ \bigoplus_{\forall j \in \mathcal{N}_i} \{\alpha(x_i, x_j)\ \phi(x_j)\}\})$$

- messages depend only on the source nodes
- aggregation function is based on attention mechanism

GAT: $\alpha(x_i, x_j) \propto \text{ReLU}(a[x_i W_1, x_j W_2]^T) \in \mathbb{R}$

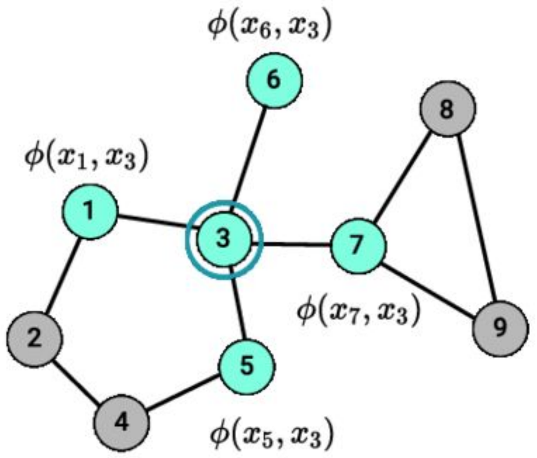
Self-Attention: $\alpha(x_i, x_j) \propto x_i W_1 (x_j W_2)^T \in \mathbb{R}$

- the model is able to learn the desired structure

[11] *Vaswani et. al. Attention is all you need. NeurIPS 2017*
[12] *Veličković et. al Graph attention networks. ICLR 2018*

Message Passing Neural Network

Bitdefender



$$y_i = f_{upd}(x_i, \bigoplus_{\forall j \in \mathcal{N}_i} \{ \phi(x_i, x_j) \})$$

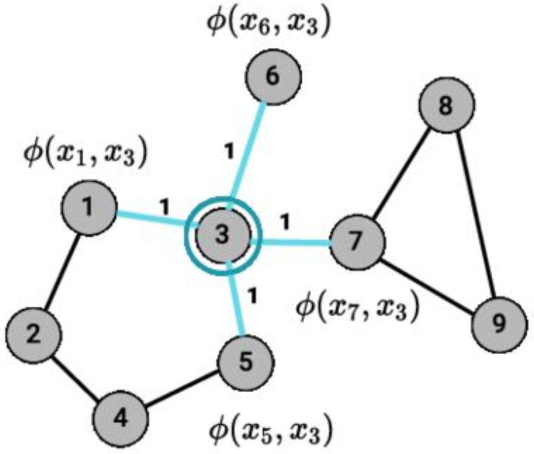
- messages depend on both source and destination
- if edge features are available, the message could also take them into account

---

[13] *Battaglia et. al. Interaction networks. NeurIPS 2016*
[14] *Gilmer et. al. Neural message passing for quantum chemistry. ICML 2017*

Message Passing Neural Network

Bitdefender



$$y_i = f_{upd}(x_i, \bigoplus_{\forall j \in \mathcal{N}_i} \{\phi(x_i, x_j)\})$$

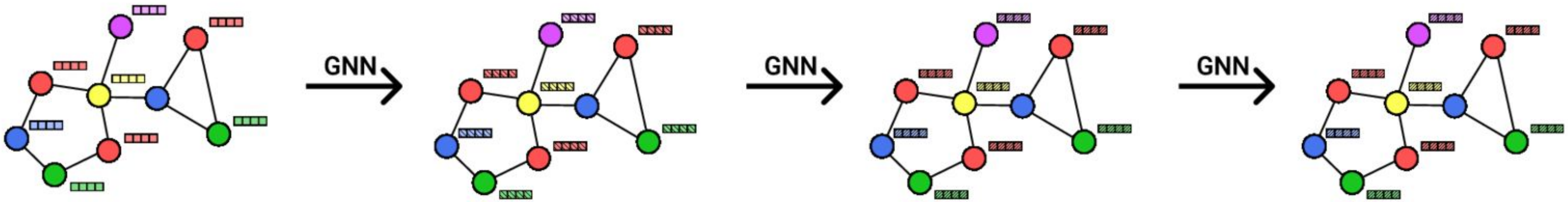
- messages depend on both source and destination
- if edge features are available, the message could also take them into account
- aggregation function is implemented as a sum/mean operation

[13] *Battaglia et. al. Interaction networks. NeurIPS 2016*
[14] *Gilmer et. al. Neural message passing for quantum chemistry. ICML 2017*

Multiple Layers

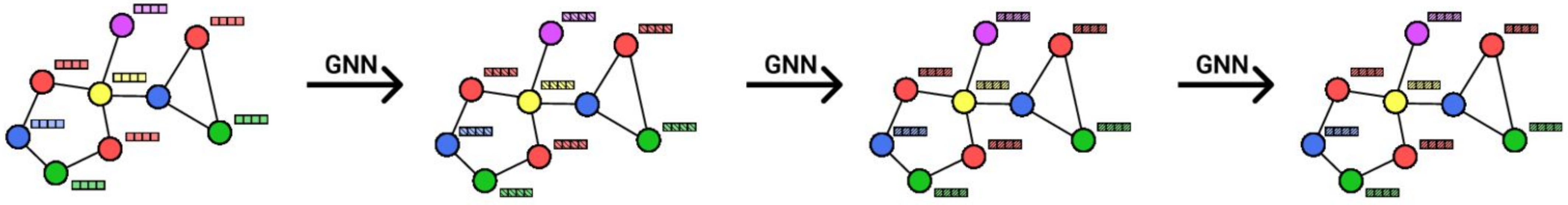
Bitdefender

- for a more powerful representation, we can stack multiple layers



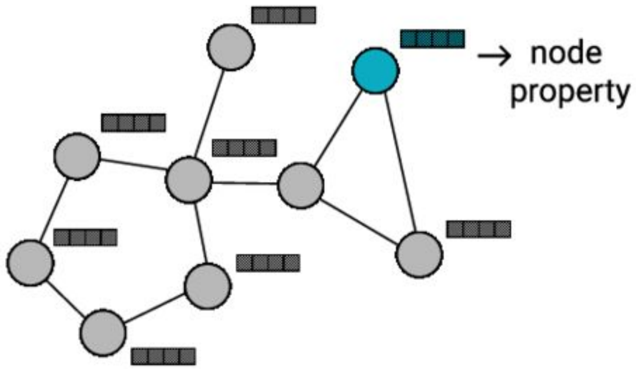
Multiple Layers

- for a more powerful representation, we can stack multiple layers
- each layer increases the receptive field of each node



RECEPTIVE FIELD:

Graph Output - Node Level



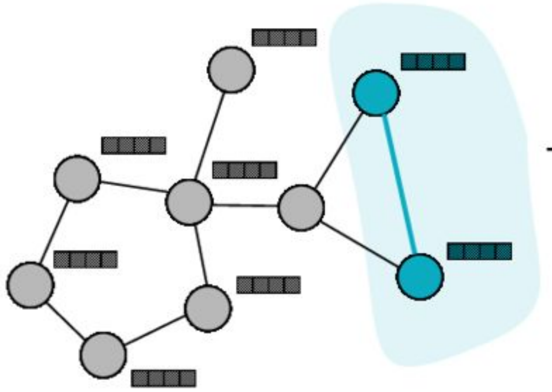
- predict an output y_i from each node

$$y_i = f_{output}(\tilde{x_i}) \quad \in \mathbb{R}^K$$

- the loss function is applied for each node in the graph

$$\mathcal{L} = \sum_{i \in \mathcal{V}} \mathcal{L}_i(y_i, l_i)$$

Graph Output - Edge Level



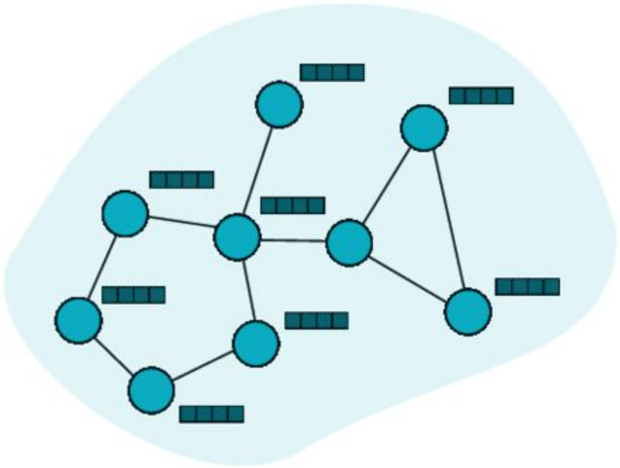
- predict an output y_{ij} from each pair of nodes

$$y_{ij} = f_{output}(\tilde{x_i}, \tilde{x_j}) \quad \in \mathbb{R}^K$$

- the loss function is applied for each edge in the graph

$$\mathcal{L} = \sum_{(i,j) \in \mathcal{E}} \mathcal{L}_i(y_{ij}, l_{ij})$$

Graph Output - Graph Level



- predict a single output y for the whole graph

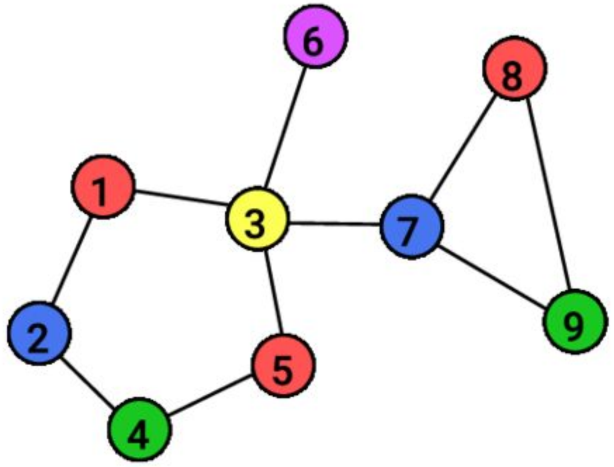
$$y = f_{readout}(\{\tilde{x_i} | \forall i \in \mathcal{V}\}) \quad \in \mathbb{R}^K$$

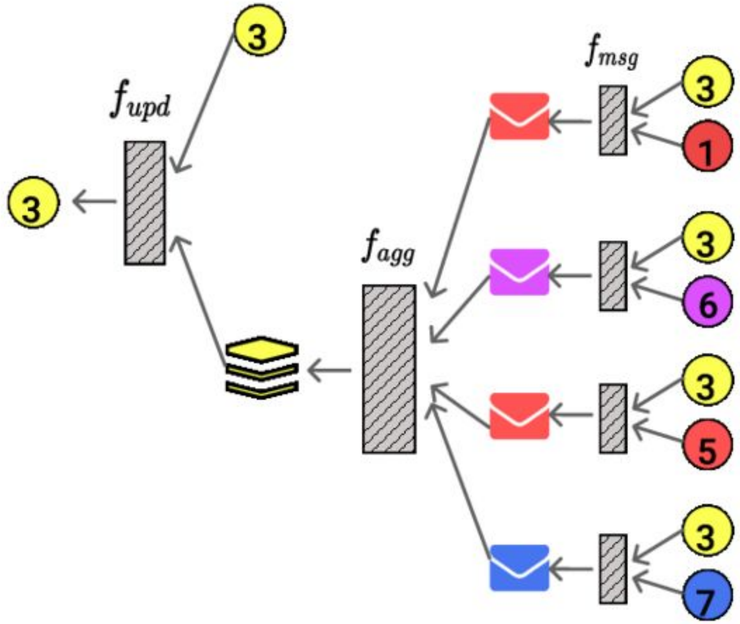
- $f_{readout}$ could be a simple order-invariant aggregator (e.g. sum, mean), or more complex graph pooling mechanisms
- the loss function is applied for each graph in the dataset

$$\mathcal{L} = \mathcal{L}_i(y, l)$$

Learning

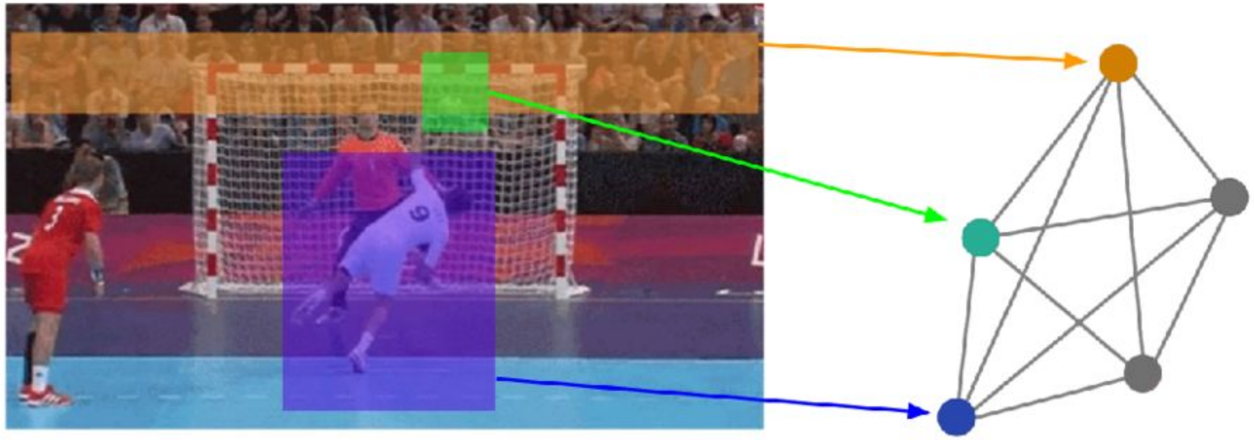
- the output of a GNN for a node i is obtained by applying a **sequence of operations** on the initial nodes
- all the operations along the sequence should be **differentiable**





GNNs applied in Vision

Bitdefender



General Framework:

- Create Nodes
- Create Relations
- GNN Processing

GNNs applied in Vision

Bitdefender

What could be a node in an image?

- fixed points / patches
- object detectors
- predicted region

GNN Application - Object detectors Approach

Pros:

- most interactions in a scene involve objects
- offers some degree of interpretability

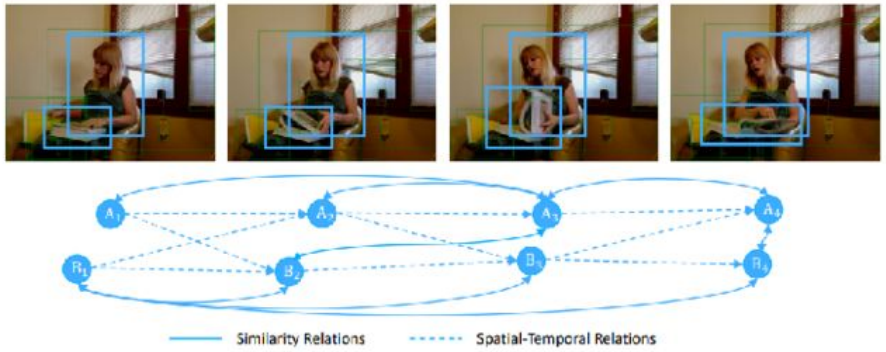
Cons:

- rigid regarding what types of interaction you can model
- expensive in terms of annotations

GNN Application - Object detectors Approach

Bitdefender

[3] Wang and Gupta. Videos as space-time region graphs. ECCV2018

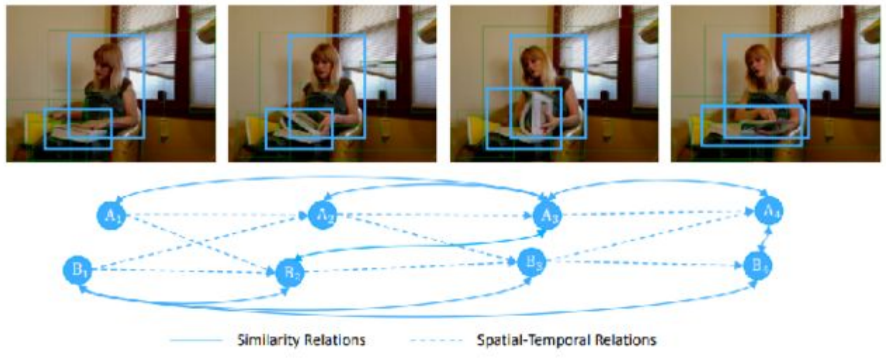


Graph structure:

- nodes are extracted using a pre-trained object detector
- two types of graphs could be built:
  1. *similarity graph:* edges between all the nodes, regardless of the time step
  2. *spatial graph*: for two time steps $(t, t+1)$ draw an edge if $IoU > threshold$. Similar for $(t, t-1)$ pairs.

GNN Application - Object detectors Approach

Bitdefender



Graph model:

- a GCN (AXW) is applied for each type of graph structure and the results are fused.
- to obtain a representation at the graph level (for the whole video) we aggregate all the nodes in the graph.

GNN Application - Patches Approach

Pros:

- simple way to encode a locality bias
- easy to use, no need for external modules

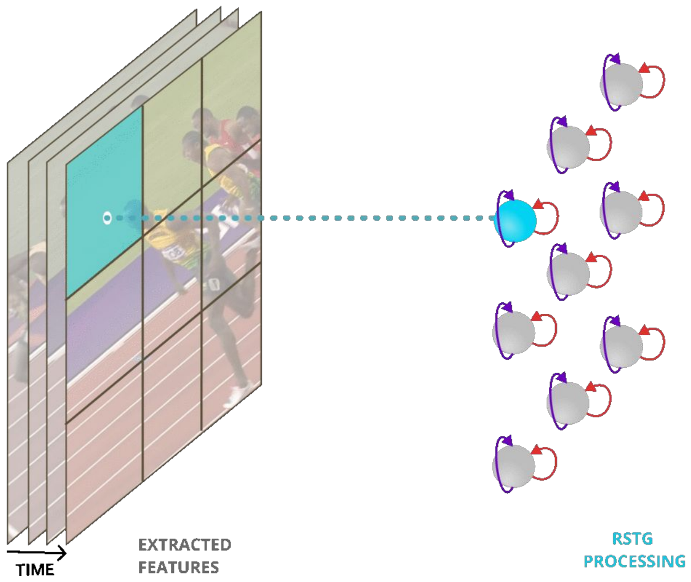
Cons:

- trade-off computation efficiency vs fine grained relations
- the captured interactions are not as interpretable

GNN Application - Patches Approach

Bitdefender

[15] Nicolicioiu, Duta, Leordeanu. Recurrent space-time graph neural networks. NeurIPS 2019



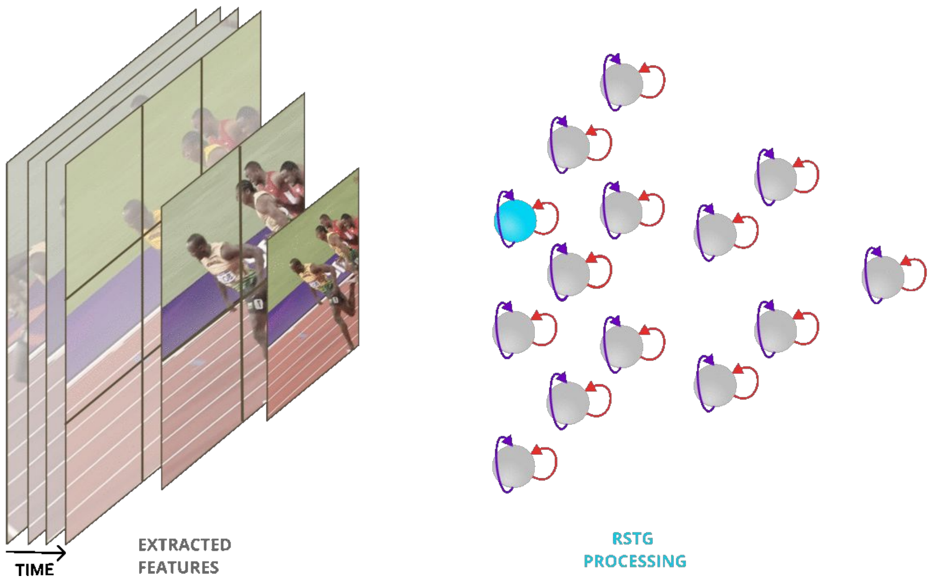
Graph structure:

- form graph nodes from fixed regions at different scales
- connect the neighbouring nodes -> sparse graph

GNN Application - Patches Approach

Bitdefender

[15] Nicolicioiu, Duta, Leordeanu. Recurrent space-time graph neural networks. NeurIPS 2019



Graph structure:

- form graph nodes from fixed regions at different scales
- connect the neighbouring nodes -> sparse graph

GNN Application - Patches Approach

Bitdefender

Graph model:

- Send: messages represent pairwise spatial interactions

$$\mathbf{f}_{\mathbf{send}}(\mathbf{v}_j, \mathbf{v}_i) = \mathsf{MLP}_s([\mathbf{v}_j | \mathbf{v}_i])$$

- Gather: aggregate received messages by an attention mechanism

$$\mathbf{f}_{\mathbf{gather}}(\mathbf{v}_i) = \sum_{j \in \mathcal{N}(i)} \alpha(\mathbf{v}_j, \mathbf{v}_i) \mathbf{f}_{\mathbf{send}}(\mathbf{v}_j, \mathbf{v}_i)$$

- Update: incorporate global context into each local information

$$\mathbf{f}_{\mathbf{space}}(\mathbf{v}_i) = \mathsf{MLP}_u([\mathbf{v}_i | \mathbf{f}_{\mathbf{gather}}(\mathbf{v}_i)])$$

GNN Application - Patches Approach

Graph model:

- across time, each node updates its spatial information using a recurrent function

$$\underbrace{\mathbf{h}_i^{t,k}}_{time} = \mathbf{f}_{\mathbf{time}}(\underbrace{\mathbf{v}_i^{k}}_{space}, \underbrace{\mathbf{h}_i^{t-1,k}}_{time})$$

GNN Application - Predicted Nodes Approach

Pros:

- adapt the type of entities to the current task/scene
- don't need object-level supervision/external modules

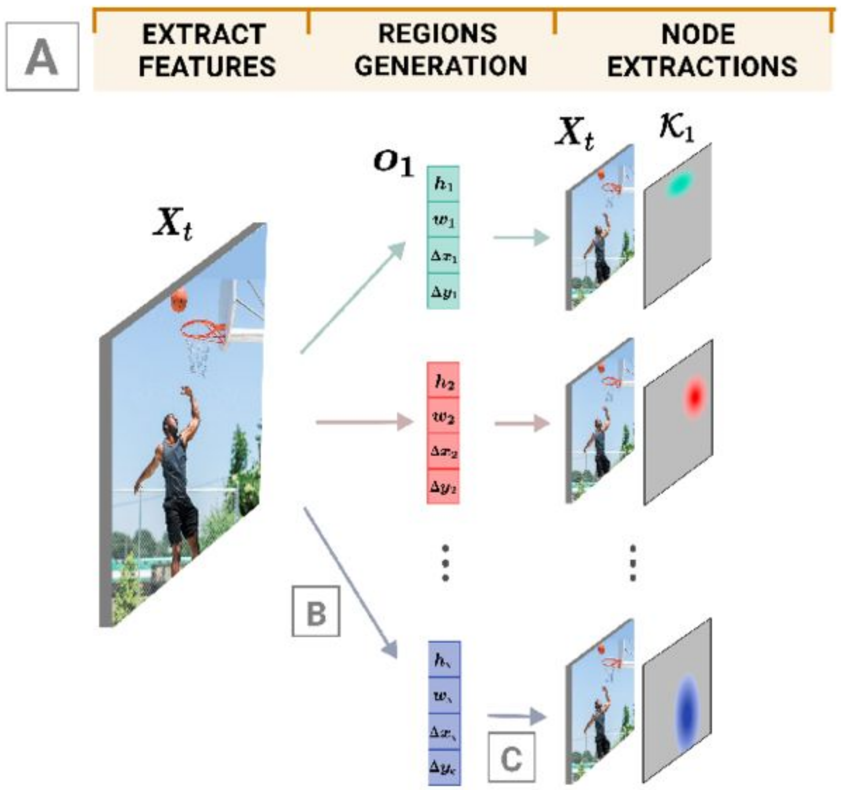
Cons:

- add complexity to the model

GNN Application - Predicted Nodes Approach

Bitdefender

[16] Duta, Nicolicioiu, Leordeanu. Discovering Dynamic Salient Regions with Spatio-Temporal Graph Neural Networks. NeurIPS 2020 - ORLR workshop



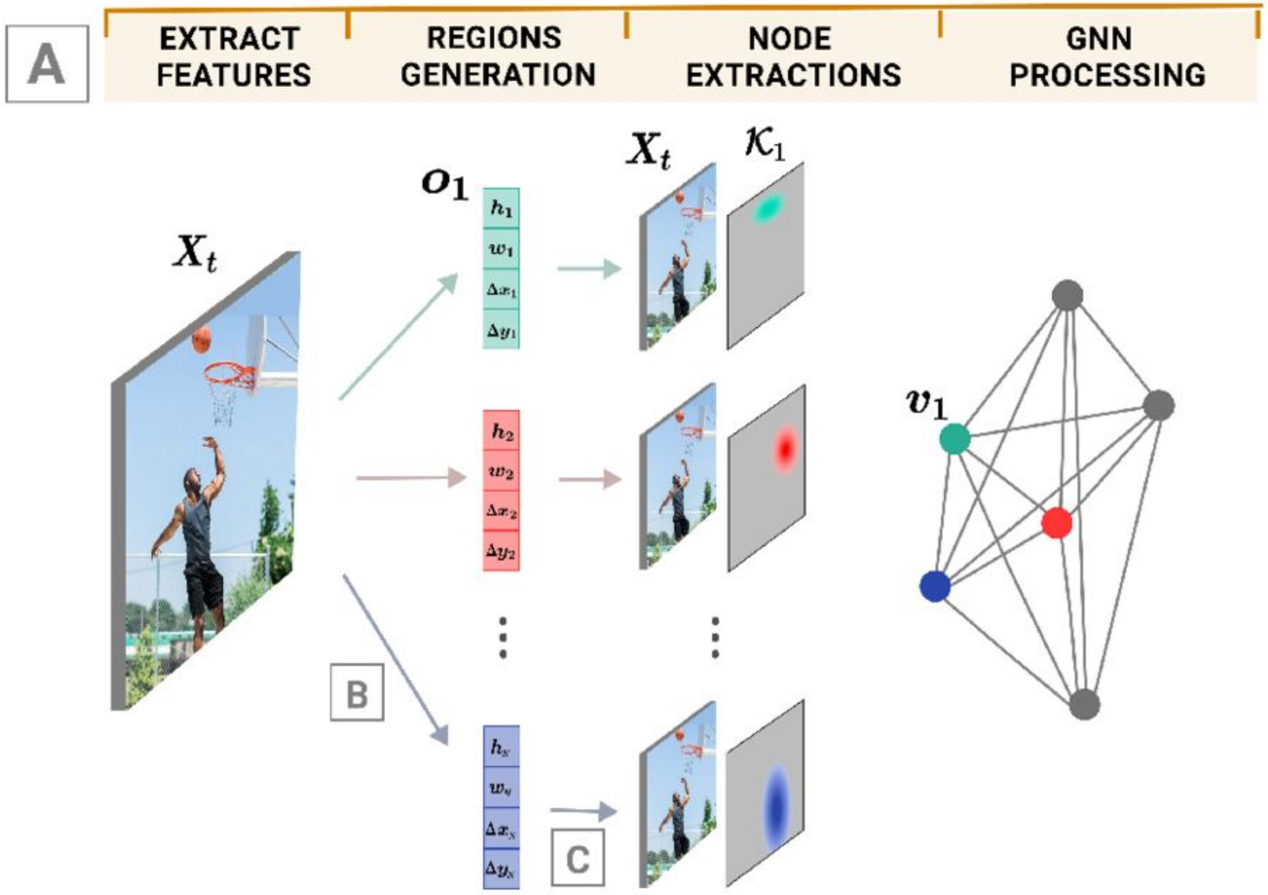
Graph structure:

- dynamically produce N = 9 regions defined by their location $(\Delta x, \Delta y)$ and size (w, h)
- extract features from each region using a differentiable pooling

$$k(\Delta x_i, w_i, p_x) = \max(0, w_i - |c_{i,x} + \Delta x_i - p_x|)$$

- train whole model from the video classification loss

GNN Application - Predicted Nodes Approach

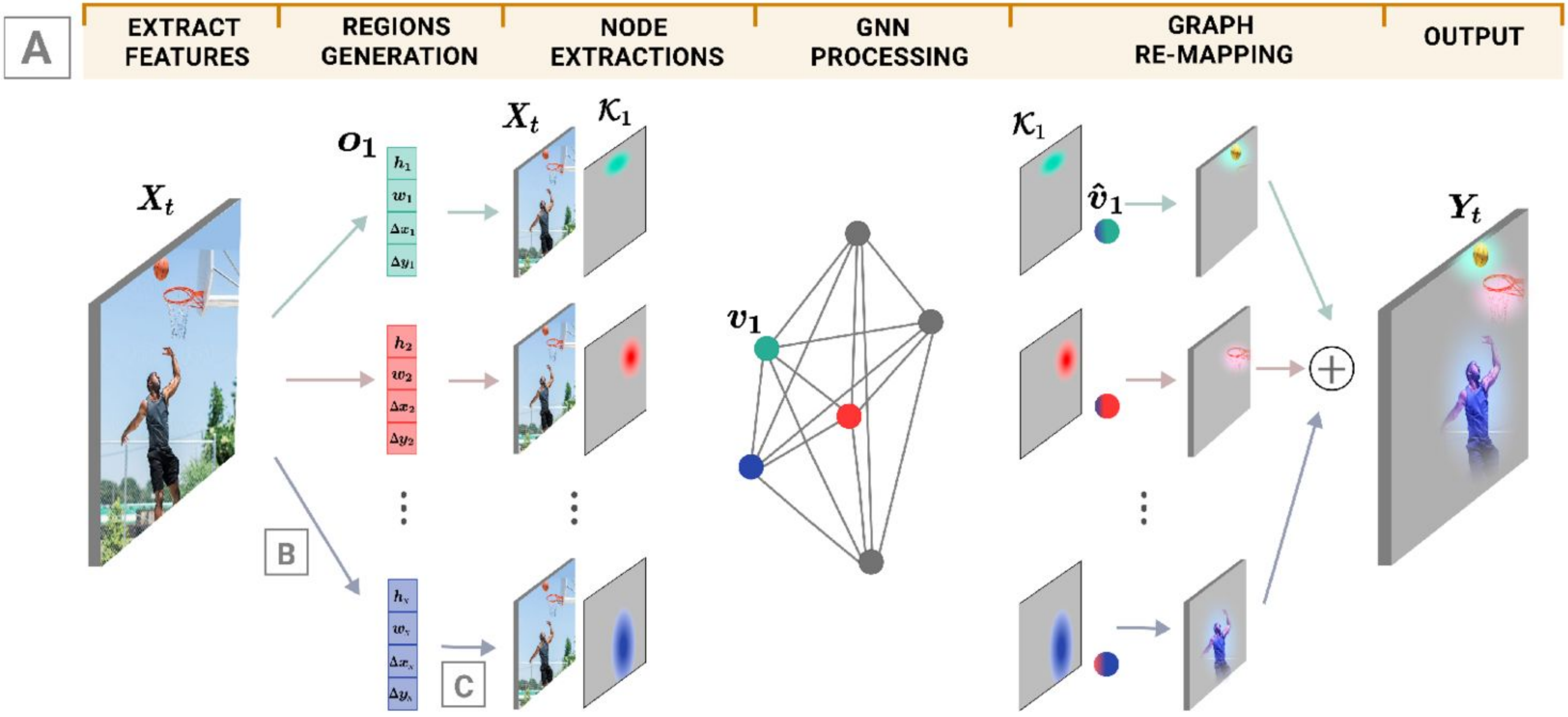


Graph model:

- spatio-temporal graph processing similar to RSTG

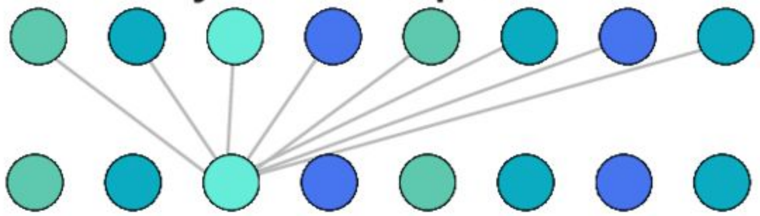
GNN Application - Predicted Nodes Approach

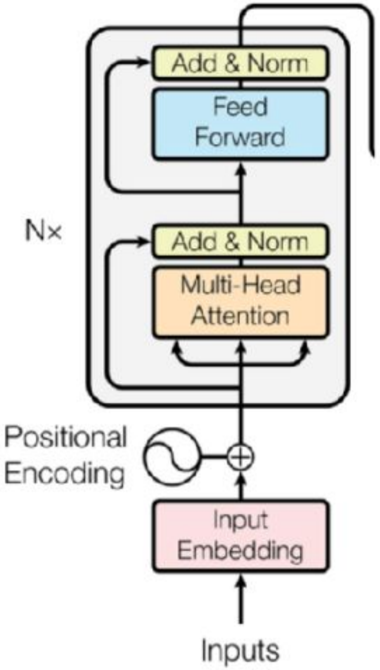
Bitdefender



Transformer

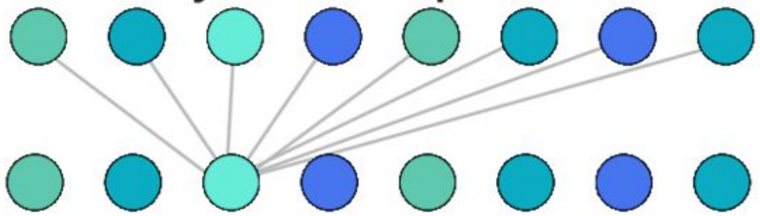
Task: analyse a sequence of words. $X = x_1, x_2, ..., x_N$.



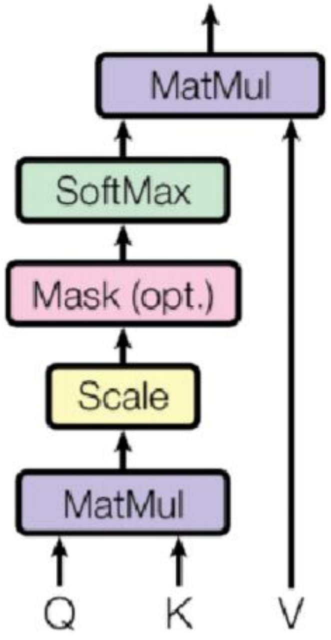


Transformer

Task: analyse a sequence of words. $X = x_1, x_2, ..., x_N$.



Scaled Dot-Product Attention



Self - Attention

- Process a sequence in multiple layers
- Each element attends to all other elements in the previous layer

$$Y = \text{softmax}(\frac{QK^T}{\sqrt{d}})V$$

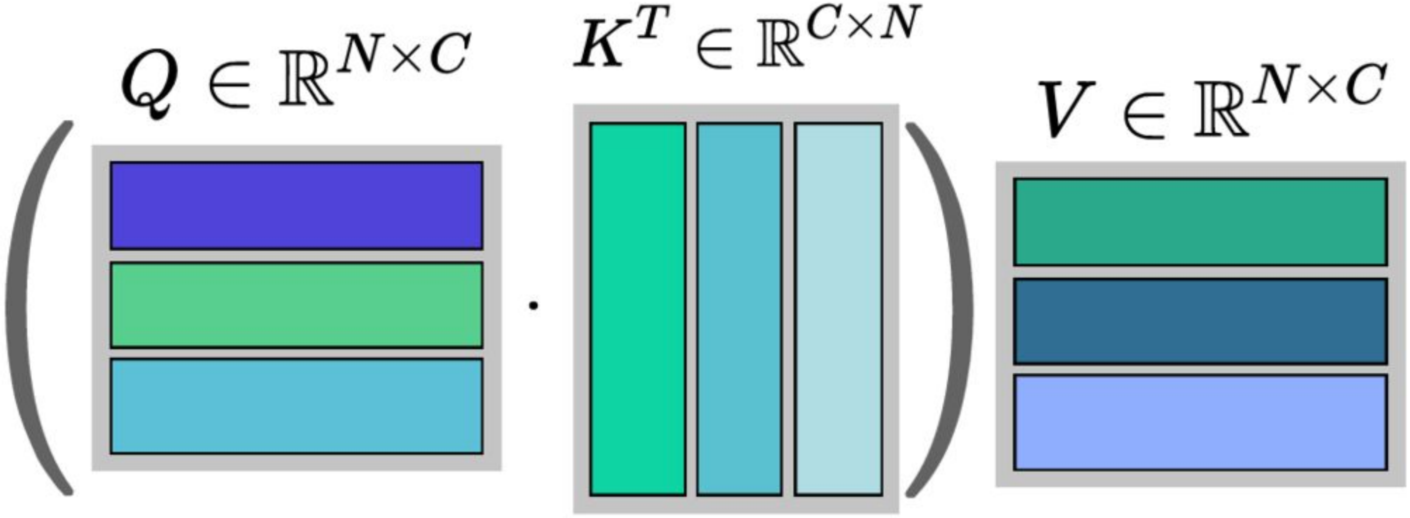
- where $Q = XW_q, K = XW_k, V = XW_v$

Transformer

Self-attention

$$Y = \text{softmax}(\frac{QK^T}{\sqrt{d}})V$$

where $Q = XW_q, K = XW_k, V = XW_v$



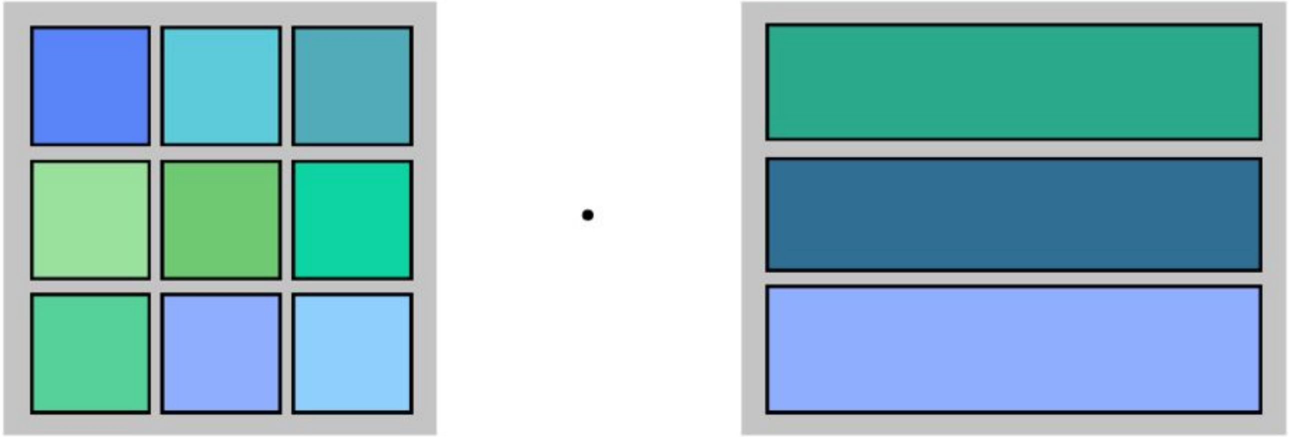
Transformer

Self-attention

$$Y = \text{softmax}(\frac{QK^T}{\sqrt{d}})V$$

where $Q = XW_q, K = XW_k, V = XW_v$

$\boldsymbol{A} \in \mathbb{R}^{\boldsymbol{N}\times\boldsymbol{N}}$ $\boldsymbol{V} \in \mathbb{R}^{\boldsymbol{N}\times\boldsymbol{C}}$



Transformer

Self-attention

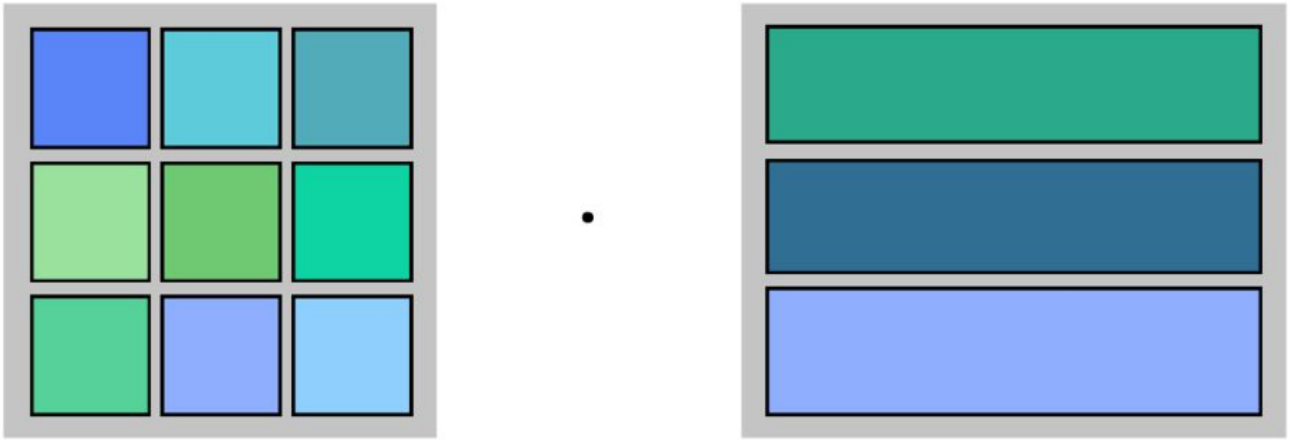
$$Y = \underbrace{\text{softmax}(\frac{QK^T}{\sqrt{d}})}_{A} V$$

GCN

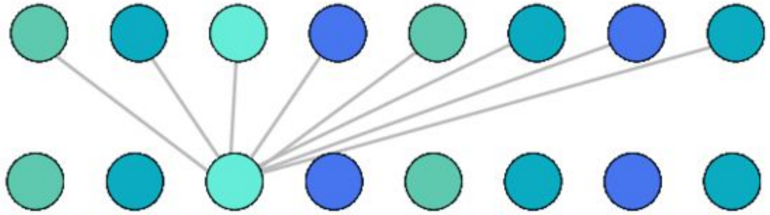
$$Y = \sigma(A \; XW)$$

where $Q = XW_q$, $K = XW_k$, $V = XW_v$

$$\boldsymbol{A} \in \mathbb{R}^{\boldsymbol{N} \times \boldsymbol{N}} \qquad \boldsymbol{V} \in \mathbb{R}^{\boldsymbol{N} \times \boldsymbol{C}}$$

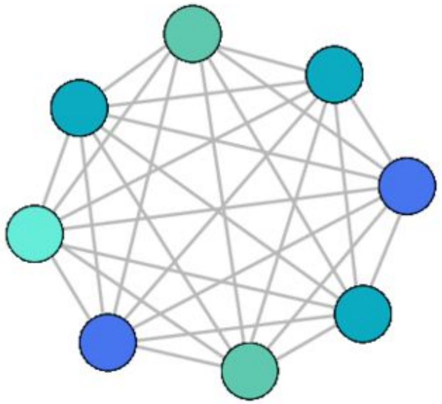


Transformer



$$Y = \frac{QK^T}{\sqrt{d}} V$$

$$y_i = \sum_{\forall j} \underbrace{\frac{1}{\sqrt{d}} \overbrace{(x_i W_q)}^{\text{Query}} \overbrace{(x_j W_k)^T}^{\text{Key}}}_{\alpha(x_i, x_j)} \overbrace{(x_j W_v)}^{\text{Value}}$$



$$y_i = f_{upd}(x_i, \sum_{\forall j \in \mathcal{N}_i} \{\alpha(x_i, x_j)\phi(x_j)\})$$

$$\alpha(x_i, x_j) = \frac{1}{\sqrt{d}}(x_i W_q)^T (x_j W_k)$$

$$\phi(x_j) = x_j W_j$$

Transformer

Bitdefender

Transformers vs GNNs

Transformer is a special case of Graph Neural Networks where

- all the nodes are connected
- pairwise messages are weighted by dot product attention

Transformer - Vision

Transformers are becoming popular in CV.

ViT [17]

- ViT [17]: model composed entirely on self-attention modules. Scale well on extremely large datasets.

- DeiT [18]: stronger augmentations + distillation => strong transformers models trained only on ImageNet

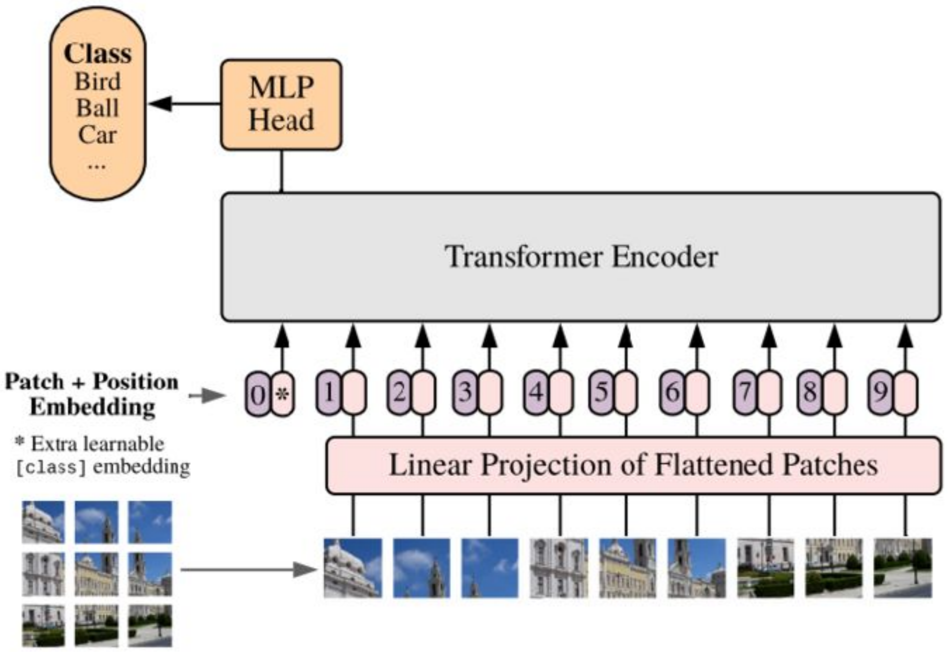
[17] Dosovitskiy at al. An image is worth 16x16 words: Transformers for image recogition. ICLR, 2021
[18] Touvron at al. Training data-efficient image transformers distillation through attention. PMLR 2021.

Transformer - Vision

Bitdefender

Transformers are becoming popular in CV.

ViT [17]

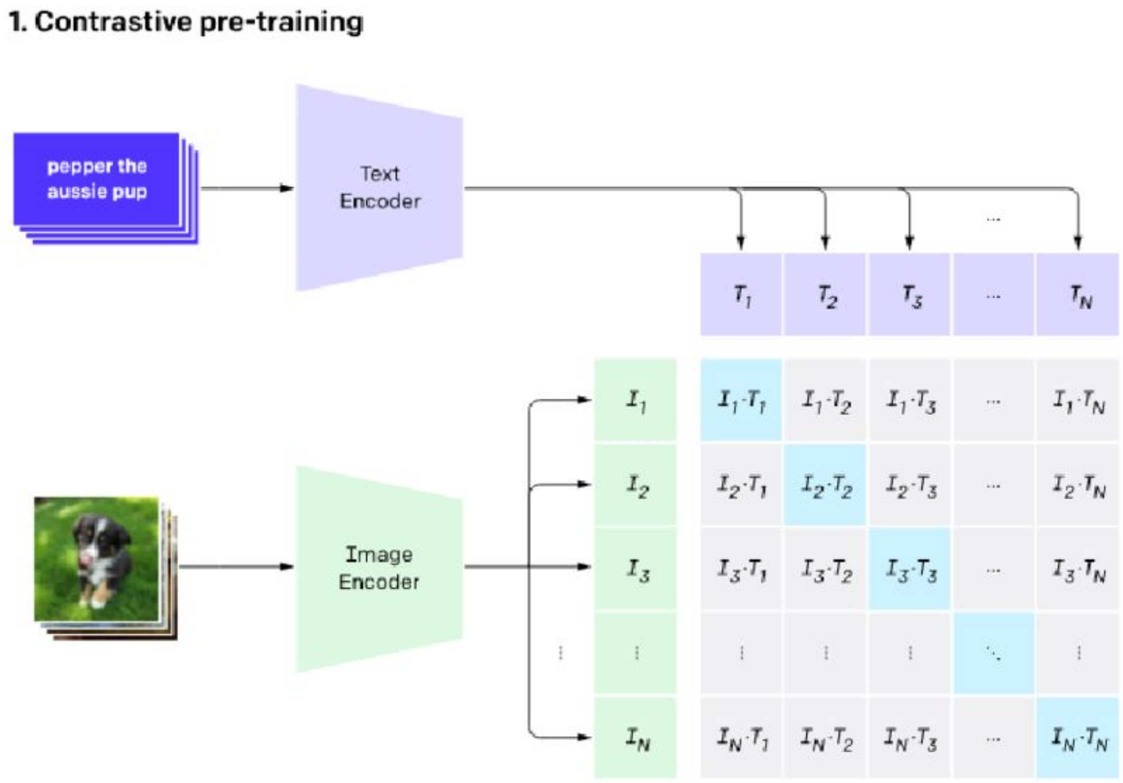


- ViT [17]: model composed entirely on self-attention modules. Scale well on extremely large datasets.
- DeiT [18]: stronger augmentations + distillation => strong transformers models trained only on ImageNet

[17] Dosovitskiy at al. An image is worth 16x16 words: Transformers for image recogition. ICLR, 2021
[18] Touvron at al. Training data-efficient image transformers distillation through attention. PMLR 2021.

Supervision from Language

CLIP (Contrastive Language–Image Pre-training) [19]

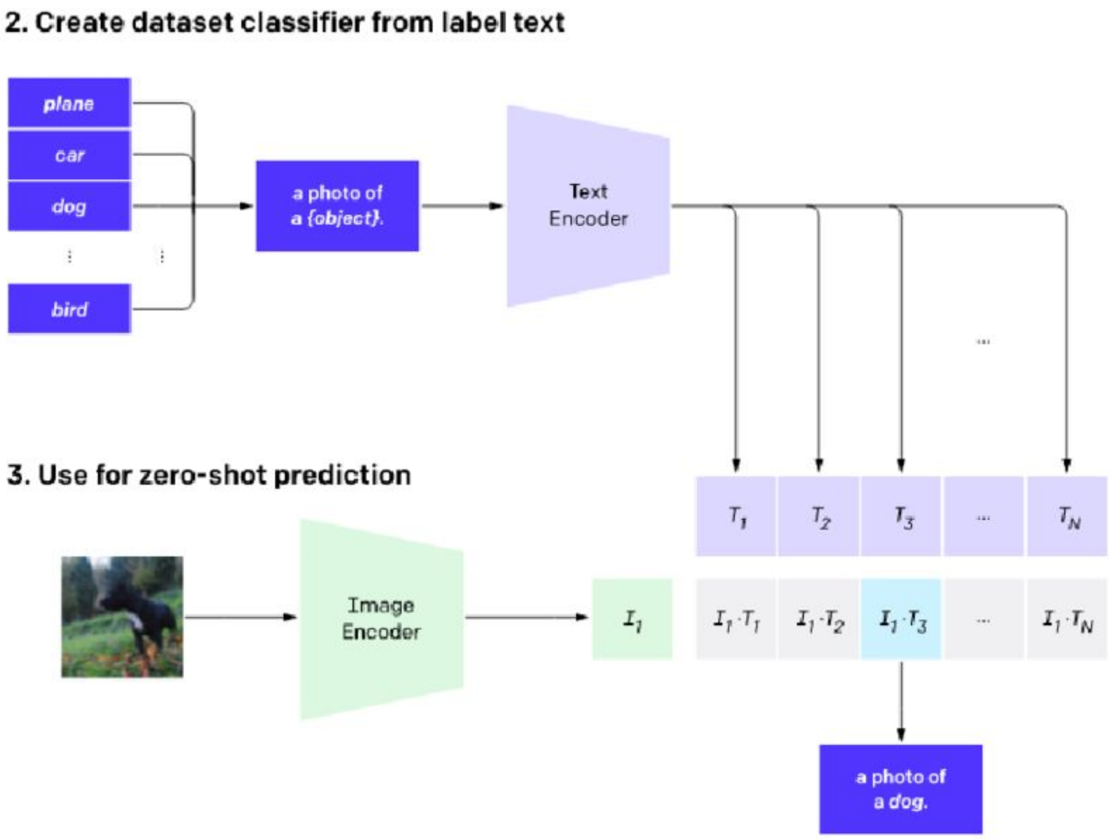


- learn from large collection of image-sentence pairs
- best models use Transformer models both for text (GPT2) and images (ViT)
- zero shot transfer

Supervision from Language

Bitdefender

CLIP (Contrastive Language–Image Pre-training) [19]

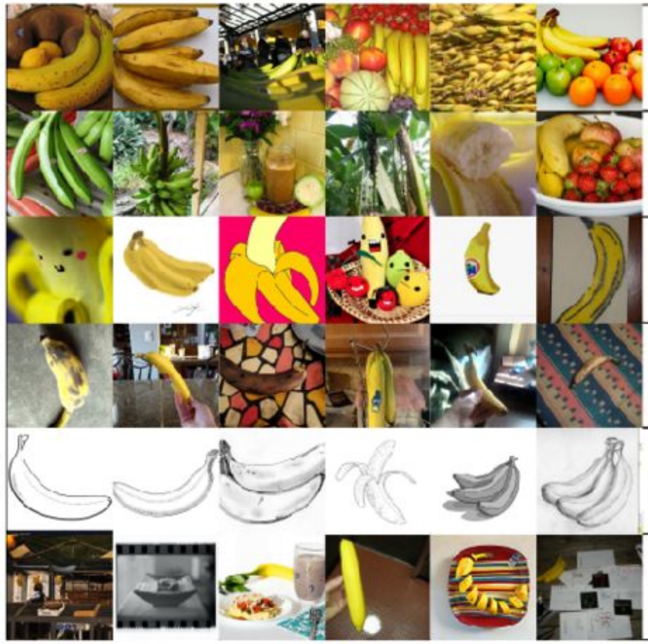


- learn from large collection of image-sentence pairs
- best models use Transformer models both for text (GPT2) and images (ViT)
- zero shot transfer

Supervision from Language

CLIP (Contrastive Language–Image Pre-training) [19]

Dataset Examples



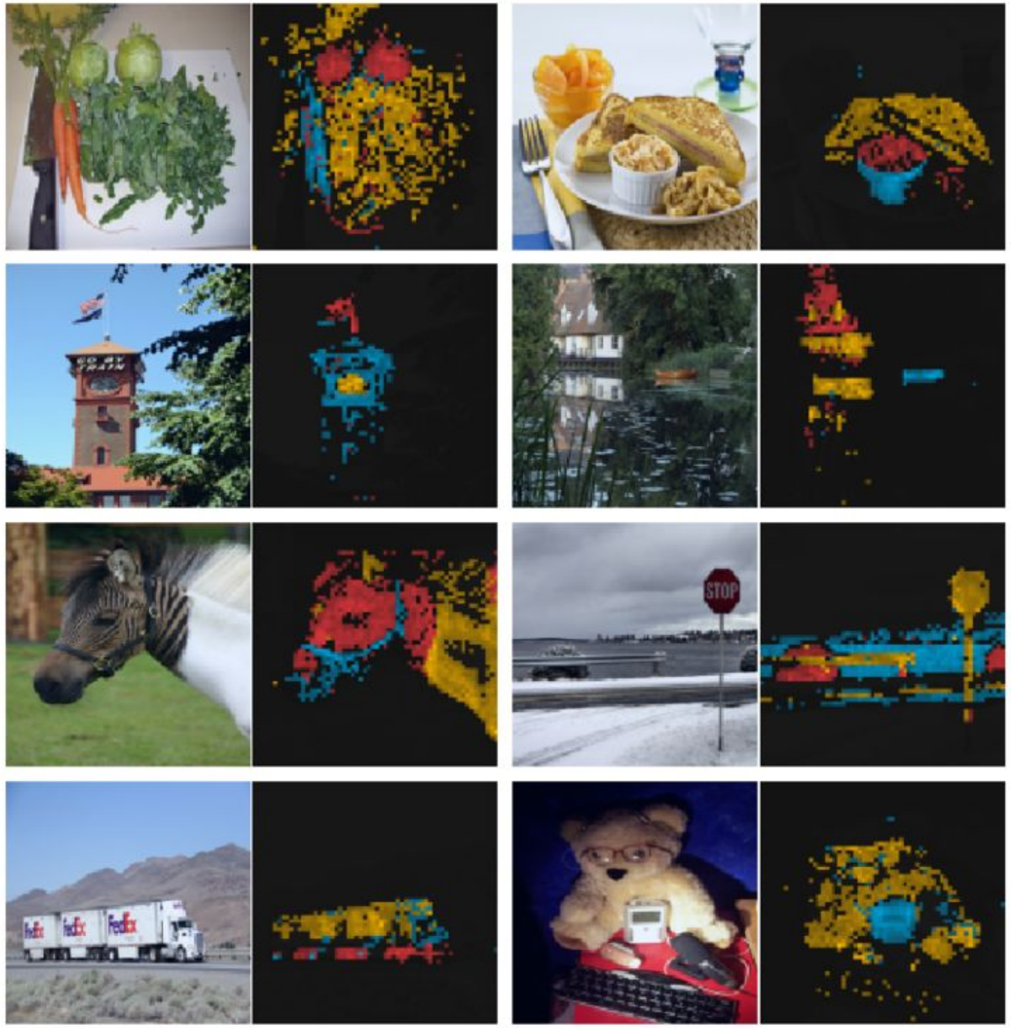
Dataset	ImageNet ResNet101	Zero-Shot CLIP	Δ Score
ImageNet	76.2	76.2	0%
ImageNetV2	64.3	70.1	+5.8%
ImageNet-R	37.7	88.9	+51.2%
ObjectNet	32.6	72.3	+39.7%
ImageNet Sketch	25.2	60.2	+35.0%
ImageNet-A	2.7	77.1	+74.4%

- CLIP is more robust than standard supervised models

Self Supervision - Transformers

Bitdefender

DINO [20]

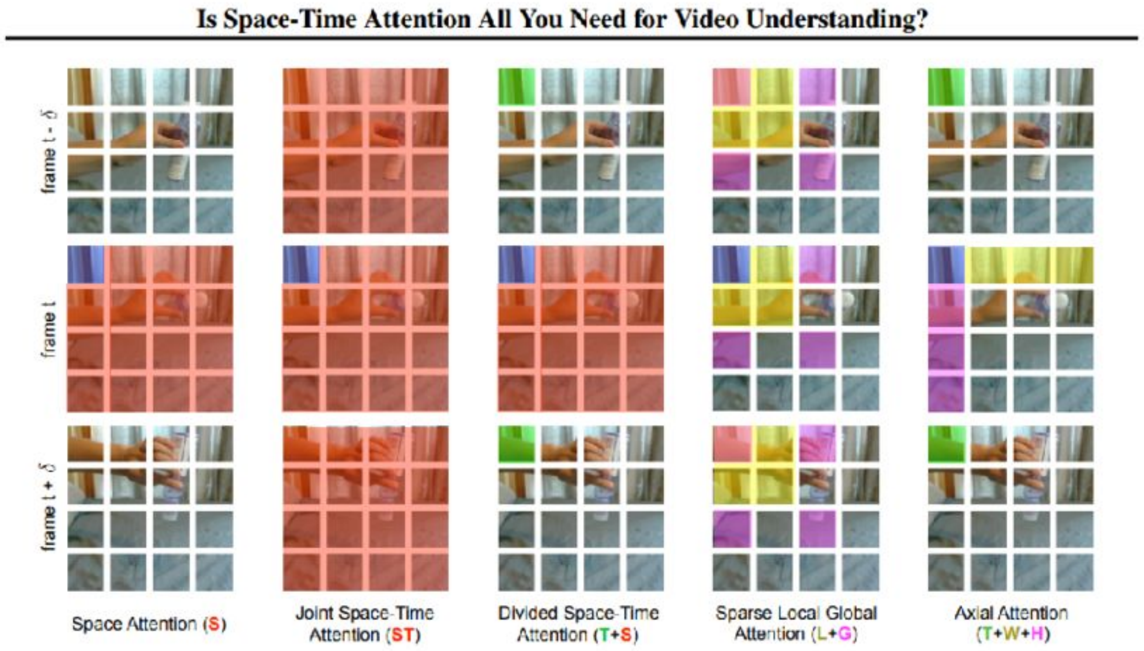


- model self-supervised on ImageNet
- self-supervised models obtain better attention maps compared to supervised ones

Transformer - Vision

Bitdefender

TimeSformer [21]



- evaluate different connectivity patterns in the attention mechanism
- Divided Space-Time Attention (T+S) has best accuracy, while being computational efficient

[21] Is space-time attention all you need for video understanding? Arxiv. 2021

Graph Neural Networks - Resources

Bitdefender

This lecture was influenced by several great resources about Graph Neural Networks. For a more in depth understanding of Graph Neural Networks and other related areas, please take a look:

- Michael Bronstein, *Geometric deep learning, from Euclid to drug design* Link
- Petar Veličković, *Theoretical Foundations of Graph Neural Networks* Link
- Jure Leskovec, *CS224W: Machine Learning with Graphs* Link
- William L. Hamilton, *Graph Representation Learning Book* Link
- Razvan Pascanu, *GraphNets - Lecture at TMLSS (Transylvanian Machine Learning Summer School)*
- Xavier Bresson, *Convolutional Neural Networks on Graphs* Link
- Michael Bronstein, *Graph Deep Learning Blog* Link

Thank You!

Iulia Duta
iduta@bitdefender.com
@DutaIulia

Andrei Nicolicioiu
anicolicioiu@bitdefender.com
@anNicolicioiu

Bitdefender®

July 2021

Human Pose Recovery and Behavior Analysis Group
University of Barcelona

References I

Bitdefender

[1] Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M Bronstein.
Fake news detection on social media using geometric deep learning.
arXiv preprint arXiv:1902.06673, 2019.

[2] Bing Yu, Haoteng Yin, and Zhanxing Zhu.
Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting.
pages 3634–3640. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[3] Xiaolong Wang and Abhinav Gupta.
Videos as space-time region graphs.
In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 399–417, 2018.

[4] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein.
Geometric deep learning on graphs and manifolds using mixture model cnns.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5115–5124, 2017.

References II

Bitdefender

[5] Kexin Huang, Cao Xiao, Lucas M Glass, Marinka Zitnik, and Jimeng Sun.
Skipgnn: predicting molecular interactions with skip-graph networks.
Scientific reports, 10(1):1–16, 2020.

[6] Miles Cranmer, Alvaro Sanchez Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho.
Discovering symbolic models from deep learning with inductive biases.
In *Advances in Neural Information Processing Systems*, 2020.

[7] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli.
Graph matching networks for learning the similarity of graph structured objects.
In *Proceedings of the 36th International Conference on Machine Learning*, Proceedings of Machine Learning Research, pages 3835–3845. PMLR, 09–15 Jun 2019.

[8] Petar Veličković, Lars Buesing, Matthew Overlan, Razvan Pascanu, Oriol Vinyals, and Charles Blundell.
Pointer graph networks.
In *Advances in Neural Information Processing Systems*, volume 33, pages 2232–2244. Curran Associates, Inc., 2020.

References III

[9] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec.
Graph convolutional neural networks for web-scale recommender systems.
In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[10] Thomas N. Kipf and Max Welling.
Semi-supervised classification with graph convolutional networks.
In *International Conference on Learning Representations (ICLR)*, 2017.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin.
Attention is all you need.
In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[12] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio.
Graph attention networks.
In *International Conference on Learning Representations*, 2018.

References IV

Bitdefender

[13] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al.
Interaction networks for learning about objects, relations and physics.
In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[14] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl.
Neural message passing for quantum chemistry.
In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272, 2017.

[15] Andrei Nicolicioiu, Iulia Duta, and Marius Leordeanu.
Recurrent space-time graph neural networks.
In *Advances in Neural Information Processing Systems 32*, pages 12838–12850. Curran Associates, Inc., 2019.

[16] Iulia Duta, Andrei Nicolicioiu, and Marius Leordeanu.
Discovering dynamic salient regions with spatio-temporal graph neural networks.
NeurIPS 2020 Workshop on Object Representations for Learning and Reasoning, 2020.

References V

Bitdefender

[17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby.
An image is worth 16x16 words: Transformers for image recognition at scale.
In *International Conference on Learning Representations*, 2021.

[18] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou.
Training data-efficient image transformers & distillation through attention.
In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[19] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al.
Learning transferable visual models from natural language supervision.
arXiv preprint arXiv:2103.00020, 2021.

[20] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin.
Emerging properties in self-supervised vision transformers.
arXiv preprint arXiv:2104.14294, 2021.

References VI

Bitdefender

[21] Gedas Bertasius, Heng Wang, and Lorenzo Torresani.
Is space-time attention all you need for video understanding?
arXiv preprint arXiv:2102.05095, 2021.