Multi-relational Graph Neural Networks

ACMS 80770: Deep Learning with Graphs

Instructor: Navid Shervani-Tabar

Department of Applied and Comp Math and Stats

Multi-relational GNNs

- In the previous lecture, we used **shallow embedding** approach to define a framework for link prediction problem.

Multi-relational GNNs

- In the previous lecture, we used **shallow embedding** approach to define a framework for link prediction problem.
- One can enhance the performance of the model by introducing a **multi-relational encoder**.

Multi-relational GNNs

- In the previous lecture, we used **shallow embedding** approach to define a framework for link prediction problem.
- One can enhance the performance of the model by introducing a **multi-relational encoder**.
- To define the information propagation rule for GNNs, we relied on the notion of **neural message passing** to learn node embeddings on graphs.
- This model can be **generalized** to account for the edge types.

Relational GCN

- The propagation rule for GCN models is defined as

$$\boldsymbol{h}_i^{(k)} = \sigma\left(\mathbf{W}^{(k-1)} \sum_{v_j \in N(v_i) \cup v_i} \frac{\boldsymbol{h}_j^{(k-1)}}{\sqrt{d_i d_j}}\right)$$

  where $\mathbf{W}^{(k-1)} \in \mathbb{R}^{d^{(k)} \times d^{(k-1)}}$ is a trainable **parameter** matrix and σ is a **non-linear** activation function.

- In this propagation rule, the aggregated message from node v_i's neighborhood is **updated** by matrix $\mathbf{W}^{(k-1)}$.

Relational GCN

- The propagation rule for GCN models is defined as

$$\boldsymbol{h}_i^{(k)} = \sigma\left(\mathbf{W}^{(k-1)} \sum_{v_j \in N(v_i) \cup v_i} \frac{\boldsymbol{h}_j^{(k-1)}}{\sqrt{d_i d_j}}\right)$$

  where $\mathbf{W}^{(k-1)} \in \mathbb{R}^{d^{(k)} \times d^{(k-1)}}$ is a trainable **parameter** matrix and σ is a **non-linear** activation function.

- In this propagation rule, the aggregated message from node v_i's neighborhood is **updated** by matrix $\mathbf{W}^{(k-1)}$.

- While this performs well for graphs with the same edge types, this does **not** take into account **different relations** of node v_i with its **neighbors**.

Relational GCN

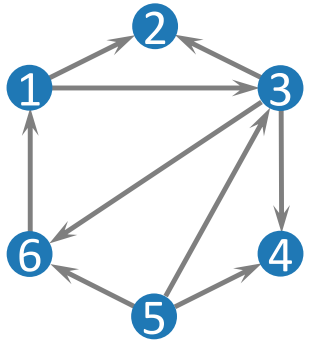
- To alleviate this, one can use **different parameter** matrices to update the aggregated neighborhood information through **different relation** type τ.
- To that end, one can redefine the message aggregation as

$$m_{N_i \to i}^{(k-1)} = \sum_{\tau \in \mathcal{R} \cup \tau_0} \sum_{v_j \in N(v_i) \cup v_i} \frac{1}{f_n\left(d_i, d_j\right)} \mathbf{W}_{\tau}^{(k-1)} \boldsymbol{h}_j^{(n-1)}$$

  where $\mathbf{W}_{\tau}^{(k-1)}$ is an update matrix which is shared by neighborhoods of type τ, f_n is a normalization function
- In the first summation, τ_0 represents the introduced self-loop that integrates previous node embedding to the update.

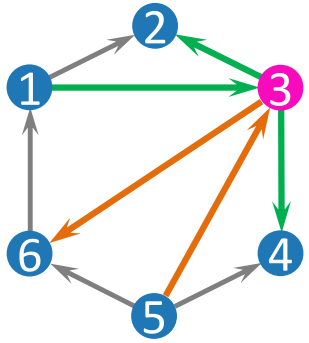
Relational GCN

- We represent the **diagram** of message computation for node v_3 in the relational GCN as following



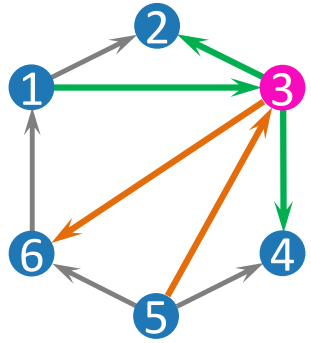
Relational GCN

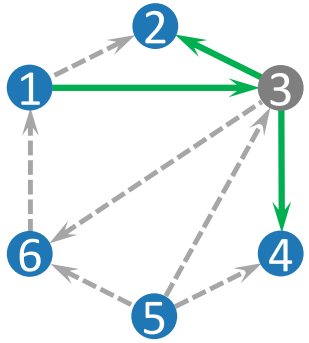
- We represent the **diagram** of message computation for node v_3 in the relational GCN as following

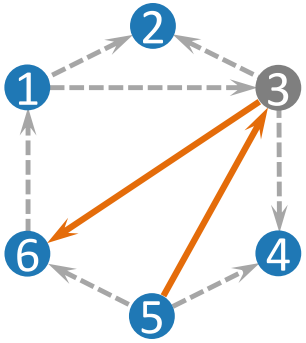


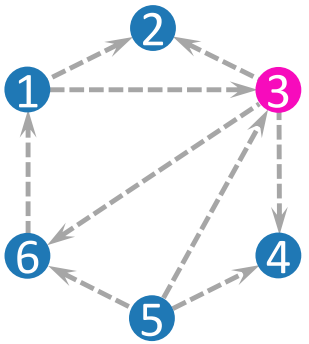
Relational GCN

- We represent the **diagram** of message computation for node v_3 in the relational GCN as following



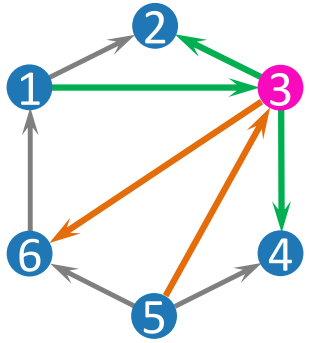


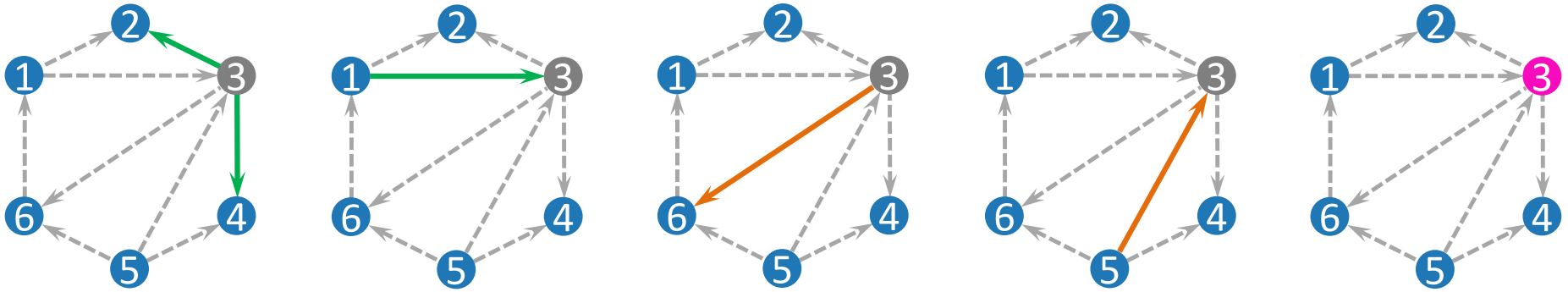




Relational GCN

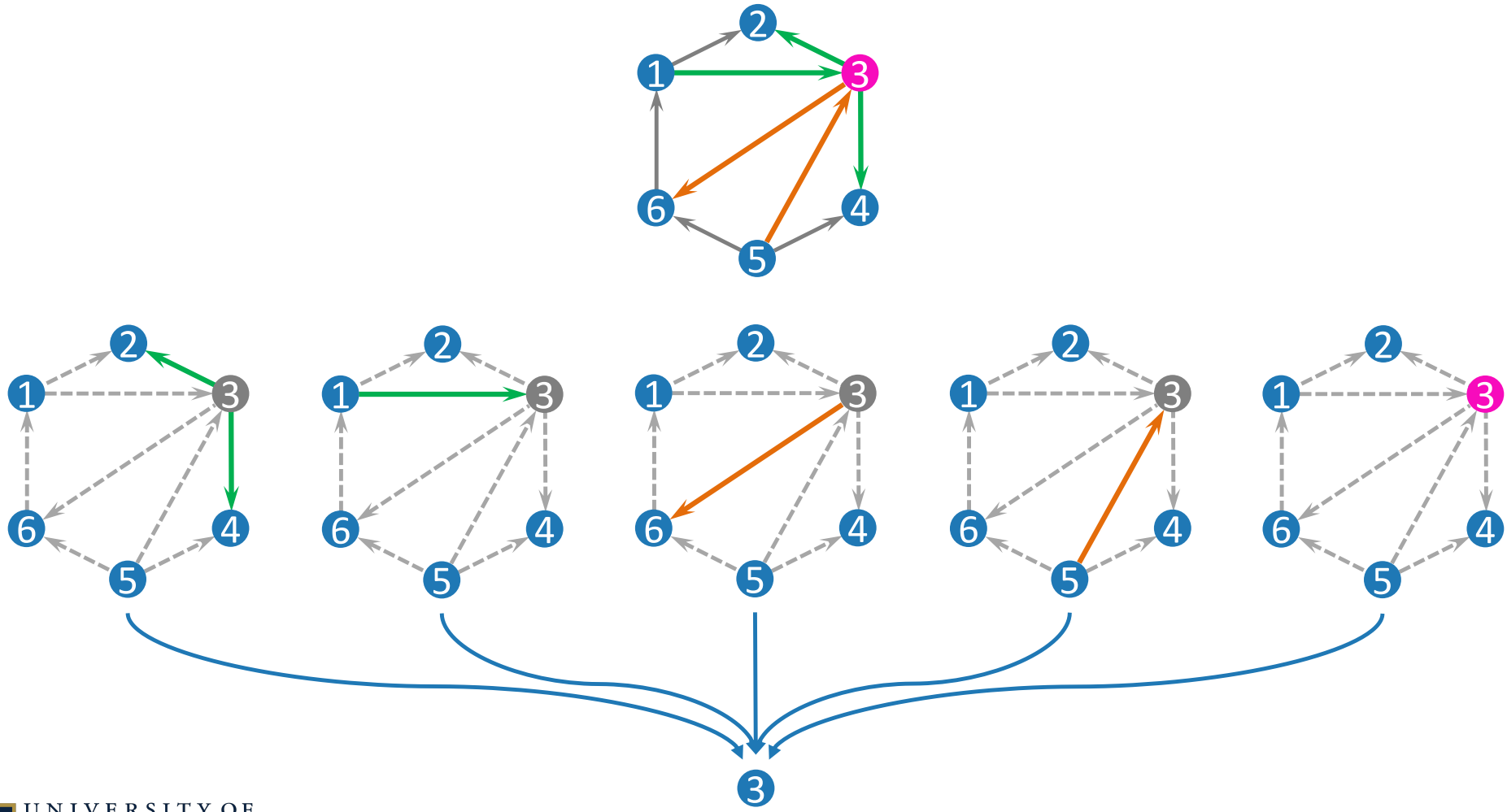
- We represent the **diagram** of message computation for node v_3 in the relational GCN as following





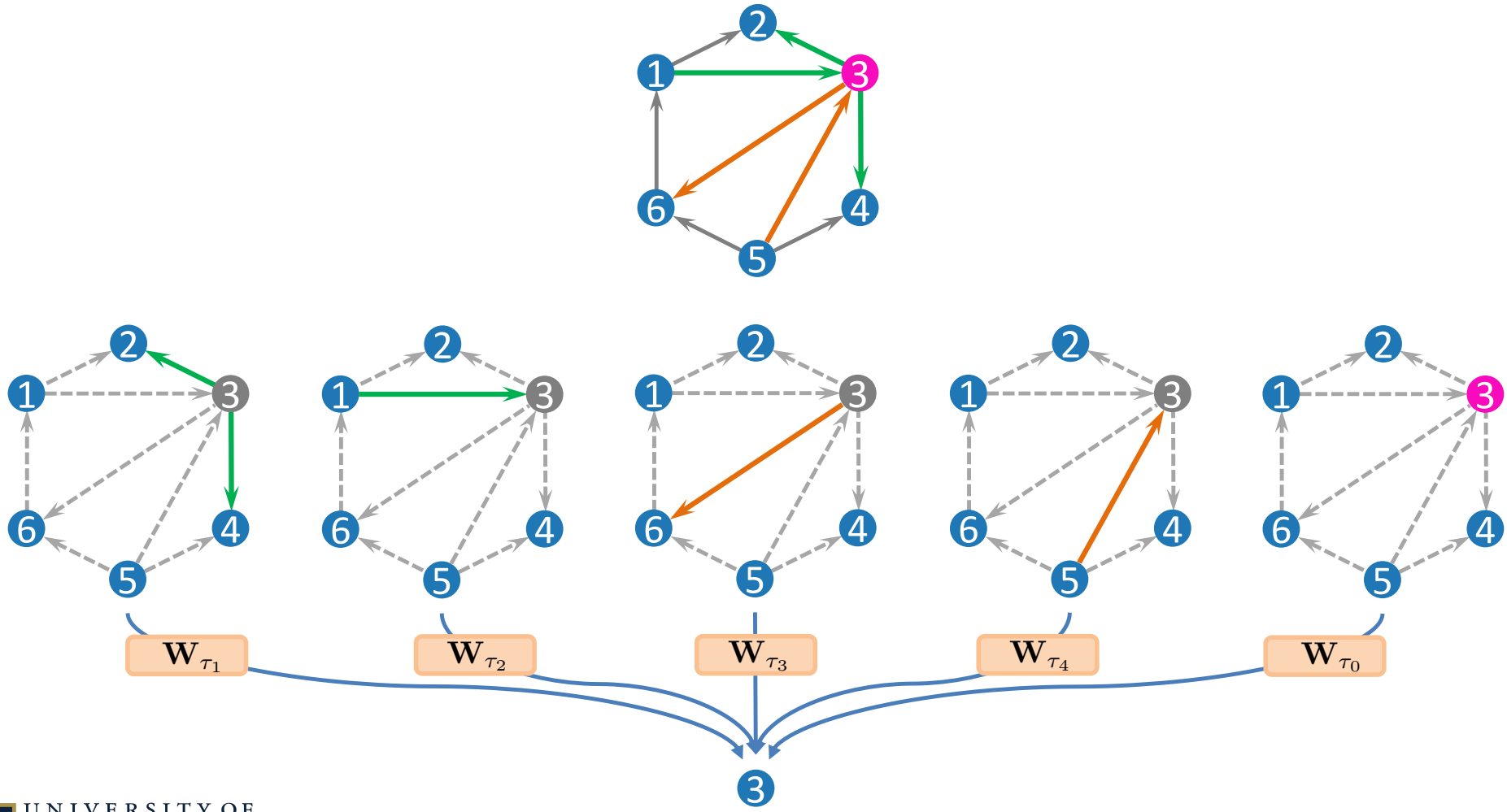
Relational GCN

- We represent the **diagram** of message computation for node v_3 in the relational GCN as following



Relational GCN

- We represent the **diagram** of message computation for node v_3 in the relational GCN as following



Relational GCN

- One problem with such definition arises due to the **number of parameters** required by this model.
- The **number of relation types** in heterogenous graphs, such as knowledge graphs, can be in the order of hundreds of thousands.

Relational GCN

- One problem with such definition arises due to the **number of parameters** required by this model.
- The **number of relation types** in heterogenous graphs, such as knowledge graphs, can be in the order of hundreds of thousands.
- Therefore, introducing **relation-specific** weight matrices $\mathbf{W}_\tau$ can be prohibitively **expensive**.

Relational GCN

- One problem with such definition arises due to the **number of parameters** required by this model.
  - The **number of relation types** in heterogenous graphs, such as knowledge graphs, can be in the order of hundreds of thousands.
- Therefore, introducing **relation-specific** weight matrices $\mathbf{W}_\tau$ can be prohibitively **expensive**.
- To alleviate this, we can **reduce the parameter size** using the following approaches:
  - Basis Decomposition
  - Block-diagonal Decomposition

Basis Decomposition

- In the **basis decomposition** approach, we redefine the weights as expansions in bases

$$\mathbf{W}_{\tau}^{(k)} = \sum_{b=1}^{B} a_{\tau,b}^{(k)} \mathbf{V}_{b}^{(k)}$$

where $\mathbf{V}_{b}^{(k)} \in \mathbb{R}^{d^{(k)} \times d^{(k+1)}}$ are basis matrices and $a_{\tau,b}^{(k)}$ are coefficients.

Basis Decomposition

- In the **basis decomposition** approach, we redefine the weights as expansions in bases

$$\mathbf{W}_{\tau}^{(k)} = \sum_{b=1}^{B} a_{\tau,b}^{(k)} \mathbf{V}_{b}^{(k)}$$

where $\mathbf{V}_{b}^{(k)} \in \mathbb{R}^{d^{(k)} \times d^{(k+1)}}$ are basis matrices and $a_{\tau,b}^{(k)}$ are coefficients.

- **Sharing basis** matrices $\mathbf{V}_{b}^{(k)}$ between all relation-specific weights $\mathbf{W}_{\tau}^{(k)}$ can significantly **reduce** the number of parameters in the model.

- Thus, $a_{\tau,b}^{(k)}$ are the only **relation-specific** learnable parameters.

Block-diagonal Decomposition

- Another approach is to use **sparse matrices** to represent relation specific weights $\mathbf{W}_{\tau}^{(k)}$.

Block-diagonal Decomposition

- Another approach is to use **sparse matrices** to represent relation specific weights $\mathbf{W}_{\tau}^{(k)}$.
- To that end, we use **block-diagonal decomposition** to represent the weights $\mathbf{W}_{\tau}^{(k)}$.
- For two arbitrary matrices $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$, the **directed sum** of the two matrices is defined as

$$\mathbf{V}_1 \oplus \mathbf{V}_2 = \begin{bmatrix} \mathbf{V}_1 & 0 \\ 0 & \mathbf{V}_2 \end{bmatrix}$$

Block-diagonal Decomposition

- Another approach is to use **sparse matrices** to represent relation specific weights $\mathbf{W}_{\tau}^{(k)}$.
- To that end, we use **block-diagonal decomposition** to represent the weights $\mathbf{W}_{\tau}^{(k)}$.
- For two arbitrary matrices $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$, the **directed sum** of the two matrices is defined as

$$\mathbf{V}_1 \oplus \mathbf{V}_2 = \begin{bmatrix} \mathbf{V}_1 & 0 \\ 0 & \mathbf{V}_2 \end{bmatrix}$$

- In block-diagonal decomposition, we construct our weight as **direct sum of dense blocks** on the diagonal and use zero elsewhere to yield a sparse matrix.

Block-diagonal Decomposition

- Here, we construct the relation-specific **weight** matrix $\mathbf{W}_{\tau}^{(k)} \in \mathbb{R}^{d^{(k)} \times d^{(k+1)}}$ as

$$\mathbf{W}_{\tau}^{(k)} = \bigoplus_{b=1}^{B} \mathbf{V}_{b,\tau}^{(k)}$$

where $\mathbf{V}_{b}^{(k)} \in \mathbb{R}^{\frac{d^{(k)}}{B} \times \frac{d^{(k+1)}}{B}}$ are dense low-dimensional blocks of same dimensionality.

Block-diagonal Decomposition

- Here, we construct the relation-specific **weight** matrix $\mathbf{W}_{\tau}^{(k)} \in \mathbb{R}^{d^{(k)} \times d^{(k+1)}}$ as

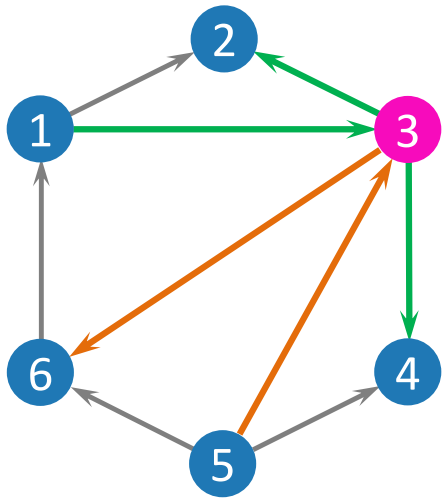
$$\mathbf{W}_{\tau}^{(k)} = \bigoplus_{b=1}^{B} \mathbf{V}_{b,\tau}^{(k)}$$

where $\mathbf{V}_{b}^{(k)} \in \mathbb{R}^{\frac{d^{(k)}}{B} \times \frac{d^{(k+1)}}{B}}$ are dense low-dimensional blocks of same dimensionality.

- One drawback with this approach is that dimensions of embedding **communicate more tightly** with the nearby dimensions that fall within same block.

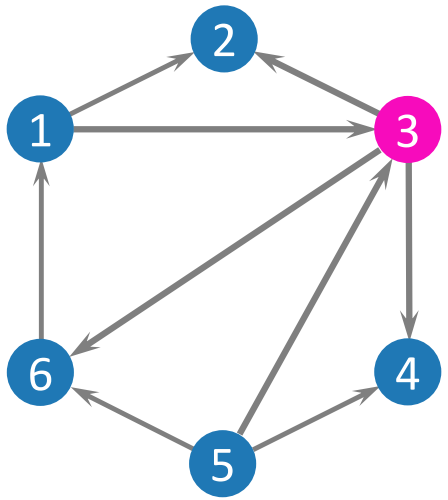
Embedding Concatenation

- So far, we have seen relational GNN models that learn embeddings for graphs with **discrete** relation types $\tau \in \mathcal{R}$.
- These models accommodate this by introducing relation-specific parameters $\mathbf{W}_{\tau}^{(k)}$.



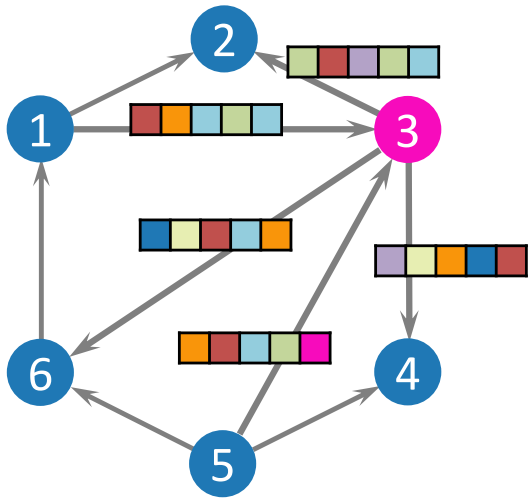
Embedding Concatenation

- So far, we have seen relational GNN models that learn embeddings for graphs with **discrete** relation types $\tau \in \mathcal{R}$.
- These models accommodate this by introducing relation-specific parameters $\mathbf{W}_{\tau}^{(k)}$.
- However, in some cases, the edge attributes can have more general forms, e.g., **real-valued vectors**.



Embedding Concatenation

- So far, we have seen relational GNN models that learn embeddings for graphs with **discrete** relation types $\tau \in \mathcal{R}$.
- These models accommodate this by introducing relation-specific parameters $\mathbf{W}_{\tau}^{(k)}$.
- However, in some cases, the edge attributes can have more general forms, e.g., **real-valued vectors**.



Embedding Concatenation

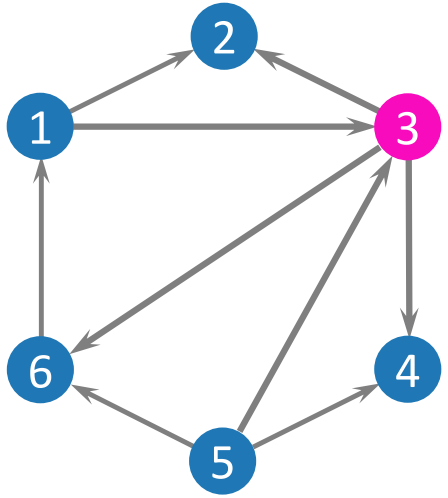
- So far, we have seen relational GNN models that learn embeddings for graphs with **discrete** relation types $\tau \in \mathcal{R}$.
- These models accommodate this by introducing relation-specific parameters $\mathbf{W}_{\tau}^{(k)}$.
- However, in some cases, the edge attributes can have more general forms, e.g., **real-valued vectors**.
- To that end, defining relation-specific weight parameters $\mathbf{W}_{\tau}^{(k)}$ may **not** be feasible.

Embedding Concatenation

- So far, we have seen relational GNN models that learn embeddings for graphs with **discrete** relation types $\tau \in \mathcal{R}$.
- These models accommodate this by introducing relation-specific parameters $\mathbf{W}_{\tau}^{(k)}$.
- However, in some cases, the edge attributes can have more general forms, e.g., **real-valued vectors**.
- To that end, defining relation-specific weight parameters $\mathbf{W}_{\tau}^{(k)}$ may **not** be feasible.
- To tackle this problem, one can use **concatenation** of embeddings in the neighborhood **aggregation** stage.

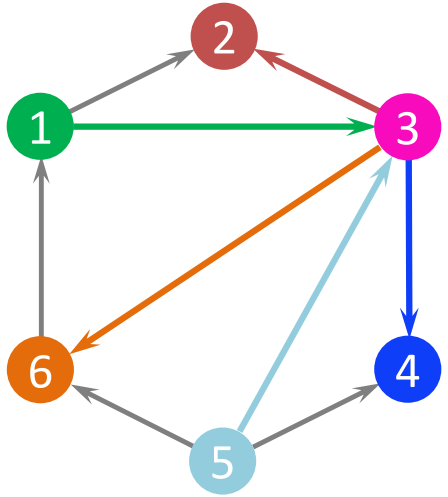
Embedding Concatenation

- In concatenation approach, one can leverage **edge attributes** by concatenating them with **node attributes** during message-passing.
- In other words, messages from each neighboring node is **augmented** with the corresponding edge feature.



Embedding Concatenation

- In concatenation approach, one can leverage **edge attributes** by concatenating them with **node attributes** during message-passing.
- In other words, messages from each neighboring node is **augmented** with the corresponding edge feature.



Embedding Concatenation

- In concatenation approach, one can leverage **edge attributes** by concatenating them with **node attributes** during message-passing.
- In other words, messages from each neighboring node is **augmented** with the corresponding edge feature.
- We modify the message aggregation step as

$$m_{N_i \to i}^{(k-1)} = \sum_{v_j \in N(v_i)} \left[\boldsymbol{h}_j^{(k-1)} \oplus \boldsymbol{h}_{(i,j)}^{(k-1)} \right]$$

where $\boldsymbol{h}_i^{(k)} \in \mathbb{R}^{d^{(k)}}$ is the node embedding for node $v_i \in \mathrm{V}$ and $\boldsymbol{h}_{(i,j)}^{(k)} \in \mathbb{R}^{d'^{(k)}}$ is the edge attributes for edge $(v_i, v_j) \in E$.

Training

- After defining modified neural message passing schemes based on **relation-aware** propagation of information, we now **train** this model.
- To train the model we need to define a **decoder** and a **loss** function.
- There are **three** types of **loss** functions generally used to train relational GNNs:

Training

- After defining modified neural message passing schemes based on **relation-aware** propagation of information, we now **train** this model.
- To train the model we need to define a **decoder** and a **loss** function.
- There are **three** types of **loss** functions generally used to train relational GNNs:
  - Reconstruction loss

Training

- After defining modified neural message passing schemes based on **relation-aware** propagation of information, we now **train** this model.
- To train the model we need to define a **decoder** and a **loss** function.
- There are **three** types of **loss** functions generally used to train relational GNNs:
  - Reconstruction loss
  - Cross-entropy loss

Training

- After defining modified neural message passing schemes based on **relation-aware** propagation of information, we now **train** this model.
- To train the model we need to define a **decoder** and a **loss** function.
- There are **three** types of **loss** functions generally used to train relational GNNs:
  - Reconstruction loss
  - Cross-entropy loss
  - Max-margin loss

Reconstruction Loss

- A basic reconstruction loss compares the **predicted** plausibility score by the decoder with the **multi-relational adjacency** tensor.
- This basic reconstruction loss can be formulated as

$$\mathcal{L} = \sum_{v_h \in V} \sum_{v_t \in V} \sum_{\tau \in \mathcal{R}} \left\| f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_t\right) - [\boldsymbol{A}]_{(h,\tau,t)} \right\|^2$$

where $A \in \mathbb{R}^{|V| \times |\mathcal{R}| \times |V|}$ is the adjacency tensor.

Reconstruction Loss

- A basic reconstruction loss compares the **predicted** plausibility score by the decoder with the **multi-relational adjacency** tensor.
- This basic reconstruction loss can be formulated as

$$\mathcal{L} = \sum_{v_h \in V} \sum_{v_t \in V} \sum_{\tau \in \mathcal{R}} \left\| f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_t\right) - [\boldsymbol{A}]_{(h,\tau,t)} \right\|^2$$

  where $A \in \mathbb{R}^{|V| \times |\mathcal{R}| \times |V|}$ is the adjacency tensor.

- However, there are two **issues** with this definition of the loss function.

Reconstruction Loss

- A basic reconstruction loss compares the **predicted** plausibility score by the decoder with the **multi-relational adjacency** tensor.
- This basic reconstruction loss can be formulated as

$$\mathcal{L} = \sum_{v_h \in V} \sum_{v_t \in V} \sum_{\tau \in \mathcal{R}} \left\| f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_t\right) - [\boldsymbol{A}]_{(h,\tau,t)} \right\|^2$$

  where $A \in \mathbb{R}^{|V| \times |\mathcal{R}| \times |V|}$ is the adjacency tensor.

- However, there are two **issues** with this definition of the loss function.
  - Computational cost

Reconstruction Loss

- A basic reconstruction loss compares the **predicted** plausibility score by the decoder with the **multi-relational adjacency** tensor.
- This basic reconstruction loss can be formulated as

$$\mathcal{L} = \sum_{v_h \in V} \sum_{v_t \in V} \sum_{\tau \in \mathcal{R}} \left\| f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_t\right) - [\boldsymbol{A}]_{(h,\tau,t)} \right\|^2$$

  where $A \in \mathbb{R}^{|V| \times |\mathcal{R}| \times |V|}$ is the adjacency tensor.

- However, there are two **issues** with this definition of the loss function.
  - Computational cost
  - Nature of the task

Reconstruction Loss

- The major drawback of this method is that the reconstruction loss defined above is very **expensive**.
- Due to the three **summations** in the loss function, this approach requires $O(|V|^2|R|)$ operations to compute the loss.

$$\mathcal{L} = \sum_{v_h \in V} \sum_{v_t \in V} \sum_{\tau \in \mathcal{R}} \left\| f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_t\right) - [\boldsymbol{A}]_{(h,\tau,t)} \right\|^2$$

Reconstruction Loss

- The major drawback of this method is that the reconstruction loss defined above is very **expensive**.
- Due to the three **summations** in the loss function, this approach requires $O(|V|^2|R|)$ operations to compute the loss.

$$\mathcal{L} = \sum_{v_h \in V} \sum_{v_t \in V} \sum_{\tau \in \mathcal{R}} \left\| f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_t\right) - [\boldsymbol{A}]_{(h,\tau,t)} \right\|^2$$

- However, multi-relational graphs are typically **sparse**, i.e.,

$$|E| << |V|^2|R|$$

- One can benefit from this sparse nature and define a function that requires $O(|E|)$ operations to compute the loss.

Reconstruction Loss

- Secondly, the multi-relational adjacency tensor is typically composed of **binary** values

$$\boldsymbol{A} \in [0,1]^{|V|\times|\mathcal{R}|\times|V|}$$

Reconstruction Loss

- Secondly, the multi-relational adjacency tensor is typically composed of **binary** values

$$\boldsymbol{A} \in [0,1]^{|V|\times|\mathcal{R}|\times|V|}$$

- Therefore, predicting links is, in essence, a **classification** problem where different classes are represented by different edge types.

$$\tau \in \mathcal{R} = \{\tau_1, \tau_2, \ldots, \tau_{|\mathcal{R}|}\}$$

- While we can use the reconstruction loss to train multi-relational embeddings, MSE loss is better suited for training **regression** models.

Cross-Entropy Loss

- Another type of loss function is based on the **cross-entropy** loss with **negative sampling**.

Cross-Entropy Loss

- Another type of loss function is based on the **cross-entropy** loss with **negative sampling**.
  - Using this loss, we **maximize** the probability of true triplet (v_h, τ, v_t) in the graph.
  - Additionally, we use negative sampling in this approach, which **minimizes** the probability of the triplets (v_h, τ, v_n) that **do not** exist.
- We refer to the triplet (v_h, τ, v_n) as **corrupted edge** or corrupted triplet.

Cross-Entropy Loss

- Another type of loss function is based on the **cross-entropy** loss with **negative sampling**.
  - Using this loss, we **maximize** the probability of true triplet (v_h, τ, v_t) in the graph.
  - Additionally, we use negative sampling in this approach, which **minimizes** the probability of the triplets (v_h, τ, v_n) that **do not** exist.
- We refer to the triplet (v_h, τ, v_n) as **corrupted edge** or corrupted triplet.
- To that end, we formulate the loss as a set of **binary classification** problems.

Cross-Entropy Loss

- During training, we **maximize** the probability

$$p\left(D=1 \mid (v_h, \tau, v_t)\right) \prod_{v_n \in V'} p\left(D=0 \mid (v_h, \tau, v_n)\right)$$

where $V' \subset V$ is a subset of entities that are **not related** to entity v_h through relation τ.

Cross-Entropy Loss

- During training, we **maximize** the probability

$$p\left(D=1 \mid (v_h, \tau, v_t)\right) \prod_{v_n \in V'} p\left(D=0 \mid (v_h, \tau, v_n)\right)$$

where $V' \subset V$ is a subset of entities that are **not related** to entity v_h through relation τ.

- $D=1$ is the event that the pair of entities v_h and v_t are **related** by relation τ, with the binary probability

$$p\left(D=1 \mid (v_h, \tau, v_t)\right)$$

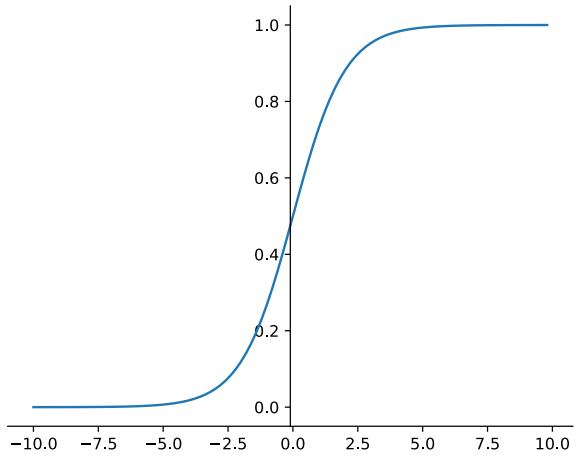
- $D=0$ is the event that the pair of entities v_n and v_t are **not related** by relation τ, with the binary probability

$$p\left(D=0 \mid (v_h, \tau, v_t)\right)$$

Cross-Entropy Loss

- We can represent the binary **probabilities** using a **sigmoid** function

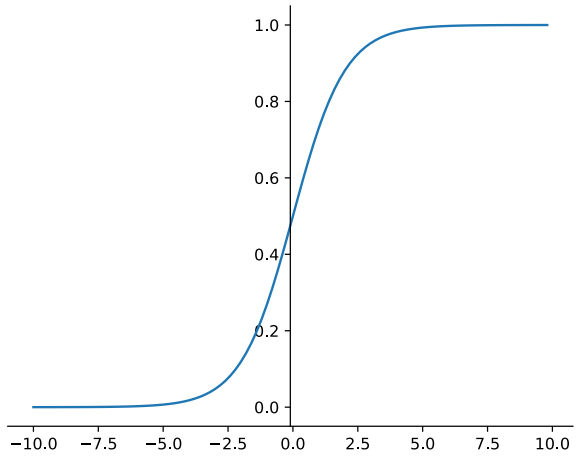
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$



Cross-Entropy Loss

- We can represent the binary **probabilities** using a **sigmoid** function

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$



- Given the score $f_d(\boldsymbol{z}_h, \tau, \boldsymbol{z}_t)$, we compute the probability of event $D = 1$ as

$$p\left(D = 1 \mid (v_h, \tau, v_t)\right) = \sigma(f_d(\boldsymbol{z}_h, \tau, \boldsymbol{z}_t)) = \frac{1}{1 + \exp(-f_d(\boldsymbol{z}_h, \tau, \boldsymbol{z}_t))}$$

Cross-Entropy Loss

- The objective is then to maximize

$$p\left(D=1 \mid (v_h, \tau, v_t)\right) \prod_{v_n \in V'} p\left(D=0 \mid (v_h, \tau, v_n)\right)$$

Cross-Entropy Loss

- The objective is then to maximize

$$p\left(D=1 \mid (v_h, \tau, v_t)\right) \prod_{v_n \in V'} p\left(D=0 \mid (v_h, \tau, v_n)\right)$$

$$p\left(D=1 \mid (v_h, \tau, v_t)\right) \prod_{v_n \in V'} \left[1 - p\left(D=1 \mid (v_h, \tau, v_n)\right)\right]$$

Cross-Entropy Loss

- The objective is then to maximize

$$p\left(D=1 \mid \left(v_h, \tau, v_t\right)\right) \prod_{v_n \in V'} p\left(D=0 \mid \left(v_h, \tau, v_n\right)\right)$$

$$p\left(D=1 \mid \left(v_h, \tau, v_t\right)\right) \prod_{v_n \in V'} \left[1 - p\left(D=1 \mid \left(v_h, \tau, v_n\right)\right)\right]$$

$$\sigma\left(f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_t\right)\right) \prod_{v_n \in V'} \left[1 - \sigma\left(f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_n\right)\right)\right]$$

Cross-Entropy Loss

- The objective is then to maximize

$$p\left(D=1 \mid \left(v_h, \tau, v_t\right)\right) \prod_{v_n \in V'} p\left(D=0 \mid \left(v_h, \tau, v_n\right)\right)$$

$$p\left(D=1 \mid \left(v_h, \tau, v_t\right)\right) \prod_{v_n \in V'} \left[1 - p\left(D=1 \mid \left(v_h, \tau, v_n\right)\right)\right]$$

$$\sigma\left(f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_t\right)\right) \prod_{v_n \in V'} \left[1 - \sigma\left(f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_n\right)\right)\right]$$

- We can rewrite the objective as the minimization of **negative log** probability

$$-\log\left(\sigma(f_d(\boldsymbol{z}_h, \tau, \boldsymbol{z}_t))\right) - \sum_{v_n \in V'} \log\left(1 - \sigma(f_d(\boldsymbol{z}_h, \tau, \boldsymbol{z}_t))\right)$$

Cross-Entropy Loss

- Using the **identity**

$$1 - \sigma(x) = \sigma(-x)$$

We rewrite the objective as minimizing

$$\mathcal{L}_{(h,t)} = -\log\left(\sigma\left(f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_f\right)\right)\right) - \sum_{v_n \in V'} \log\left(\sigma\left(-f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_n\right)\right)\right)$$

Cross-Entropy Loss

- Using the **identity**

$$1 - \sigma(x) = \sigma(-x)$$

We rewrite the objective as minimizing

$$\mathcal{L}_{(h,t)} = -\log\left(\sigma\left(f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_f\right)\right)\right) - \sum_{v_n \in V'} \log\left(\sigma\left(-f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_n\right)\right)\right)$$

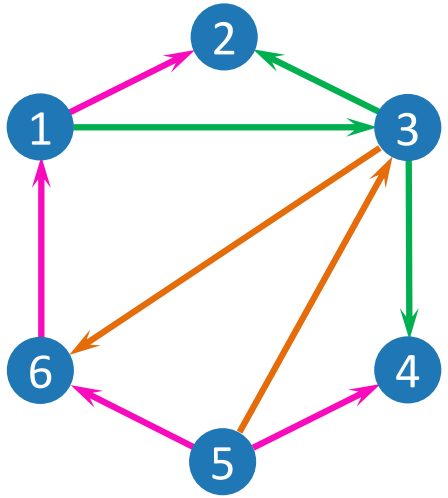
- We formulate the cross-entropy loss with negative sampling as

$$\mathcal{L} = -\sum_{(v_h,\tau,v_t)\in E} \Bigg[\log\left(\sigma\left(f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_f\right)\right)\right) + \sum_{v_n \sim p(V)} \log\left(\sigma\left(-f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_n\right)\right)\right)\Bigg]$$

where $P(V)$ is a distribution over nodes.

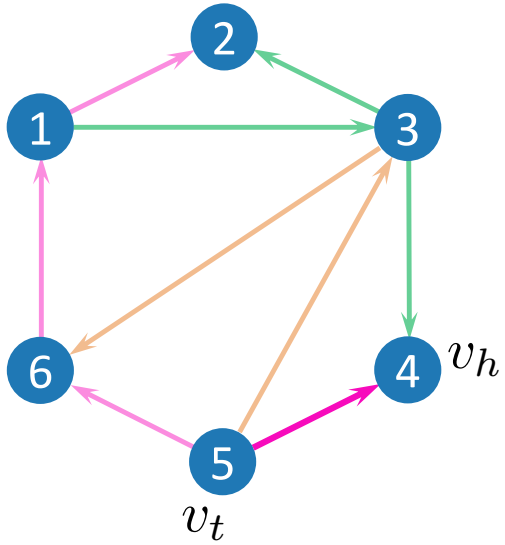
Negative Sampling

- In this method, we sample **negative edges**, i.e. edges that do not exist, by **replacing** the **tail** entity by a randomly selected node v_n.



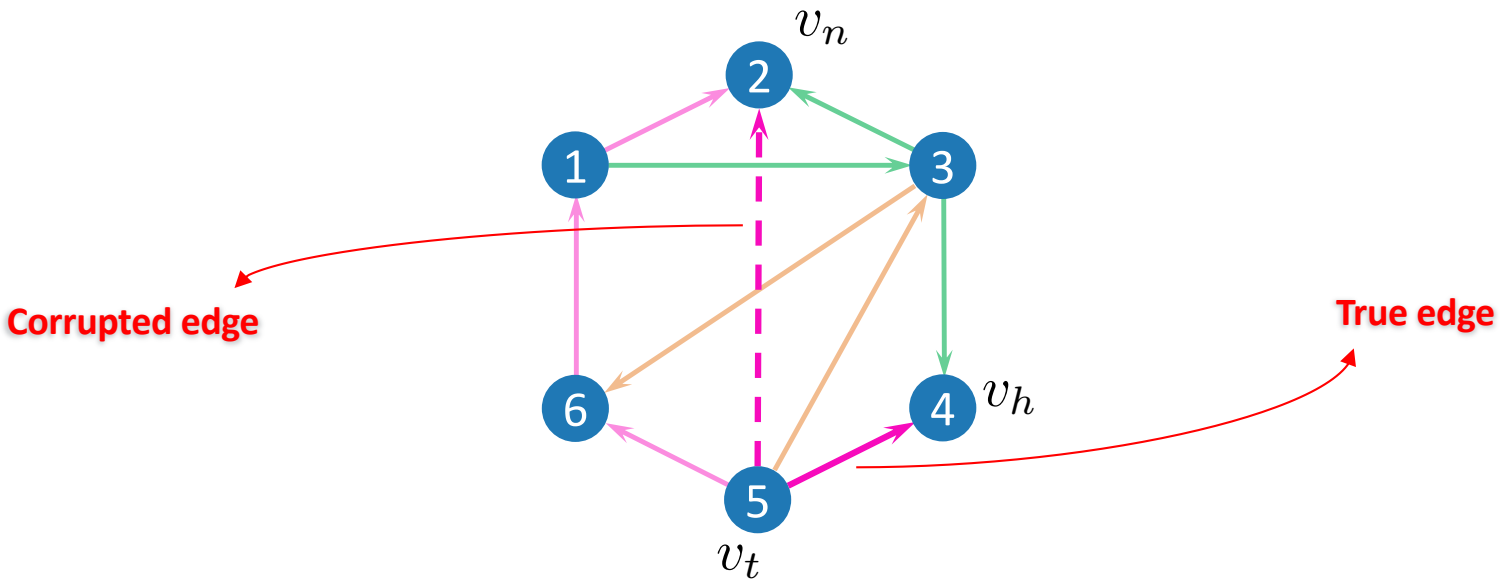
Negative Sampling

- In this method, we sample **negative edges**, i.e. edges that do not exist, by **replacing** the **tail** entity by a randomly selected node v_n.



Negative Sampling

- In this method, we sample **negative edges**, i.e. edges that do not exist, by **replacing** the **tail** entity by a randomly selected node v_n.



Negative Sampling

- In this method, we sample **negative edges**, i.e. edges that do not exist, by **replacing** the **tail** entity by a randomly selected node v_n.
- While we have corrupted the edge by only replacing the tail node, this can lead to a **directional bias** during training.
- Therefore, it would be better to sample negative edges by corrupting either **head** or **tail**, but not both at the same time.

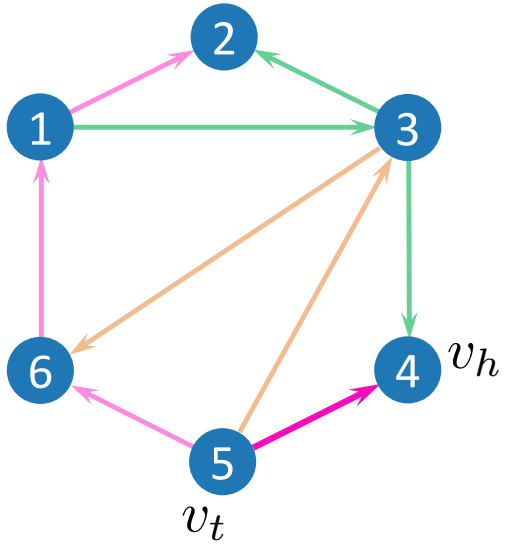
Negative Sampling

- In this method, we sample **negative edges**, i.e. edges that do not exist, by **replacing** the **tail** entity by a randomly selected node v_n.
- While we have corrupted the edge by only replacing the tail node, this can lead to a **directional bias** during training.
- Therefore, it would be better to sample negative edges by corrupting either **head** or **tail**, but not both at the same time.
- In other words, for each triplet (v_h, τ, v_t), we can define the set of negative samples as

$$E'_{(v_h,\tau,v_t)} = \{(v_h, \tau, v_n) | v_n \sim p(V)\} \cup \{(v_n, \tau, v_t) | v_n \sim p(V)\}$$

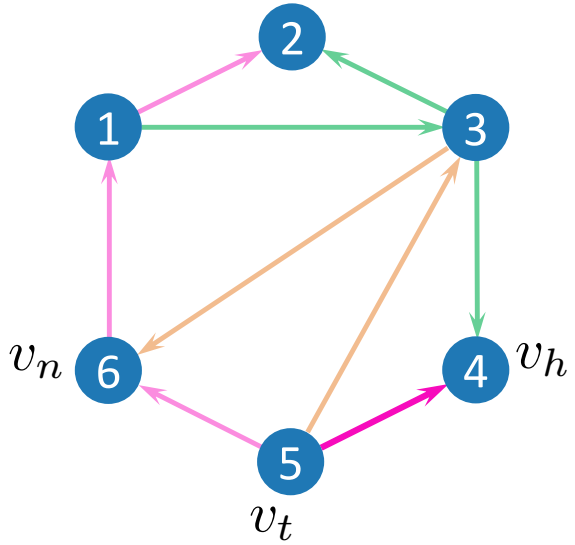
Negative Sampling

- The distribution $P(V)$ can be a **uniform** distribution.



Negative Sampling

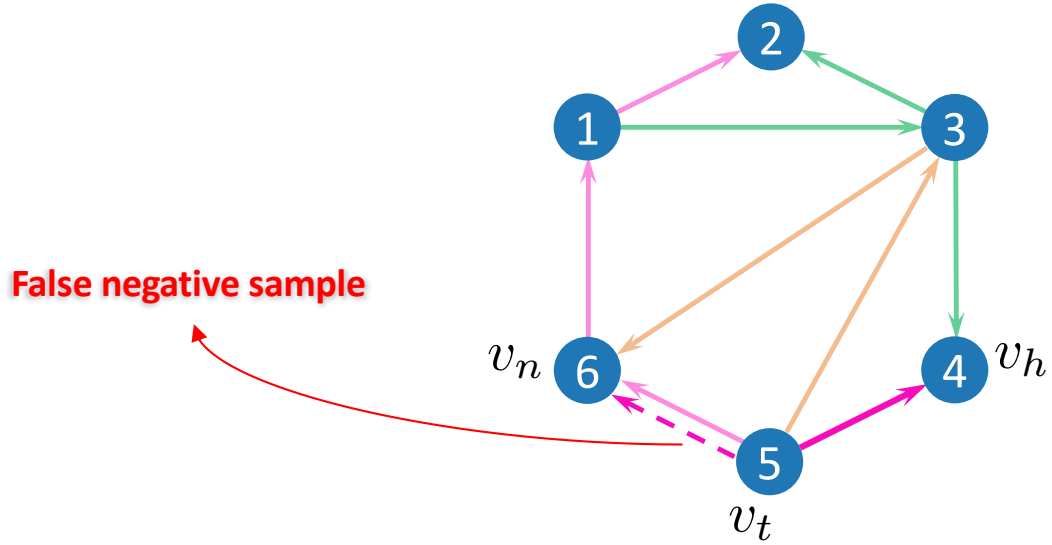
- The distribution $P(V)$ can be a **uniform** distribution.



Negative Sampling

- The distribution $P(V)$ can be a **uniform** distribution.
- One issue with this is the possibility of having **false negatives**, i.e., corrupted edges that actually, exist.

$$(v_h, \tau, v_n) \in E$$



Negative Sampling

- The distribution $P(V)$ can be a **uniform** distribution.
- One issue with this is the possibility of having **false negatives**, i.e., corrupted edges that actually, exist.

$$(v_h, \tau, v_n) \in E$$

- More sophisticated approaches use **filtering** to remove such samples.
- After sampling the corrupted edges, **Monte Carlo** methods are used to compute the loss.

Max-Margin Loss

- The last type of loss function we can use is **max-margin loss**.
- In this method, also referred to as hinge loss, we don't use probabilities, but instead **compare the score** for the edges in the training set with negative samples.

Max-Margin Loss

- The last type of loss function we can use is **max-margin loss**.
- In this method, also referred to as hinge loss, we don't use probabilities, but instead **compare the score** for the edges in the training set with negative samples.
- Max-margin loss is defined as

$$\mathcal{L} = \sum_{(v_h,\tau,v_t)} \sum_{v_n \sim P(V)} \max\left(0, -f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_t\right) + f_d\left(\boldsymbol{z}_h, \tau, \boldsymbol{z}_n\right) + \Delta\right)$$

  where $\Delta > 0$ is the **margin parameter**.

- By using max-margin loss, we dictate that the true triplet score by f_d should be **larger** than the corrupted triplet's score by at least a **margin** size Δ.

Summary

- Multi-relational Graph Neural Networks
  - Relational GCN
    - Basis Decomposition
    - Block-diagonal Decomposition
  - Concatenation-based
- Loss
  - Reconstruction loss
  - Cross entropy loss with negative sampling
  - Max-margin loss