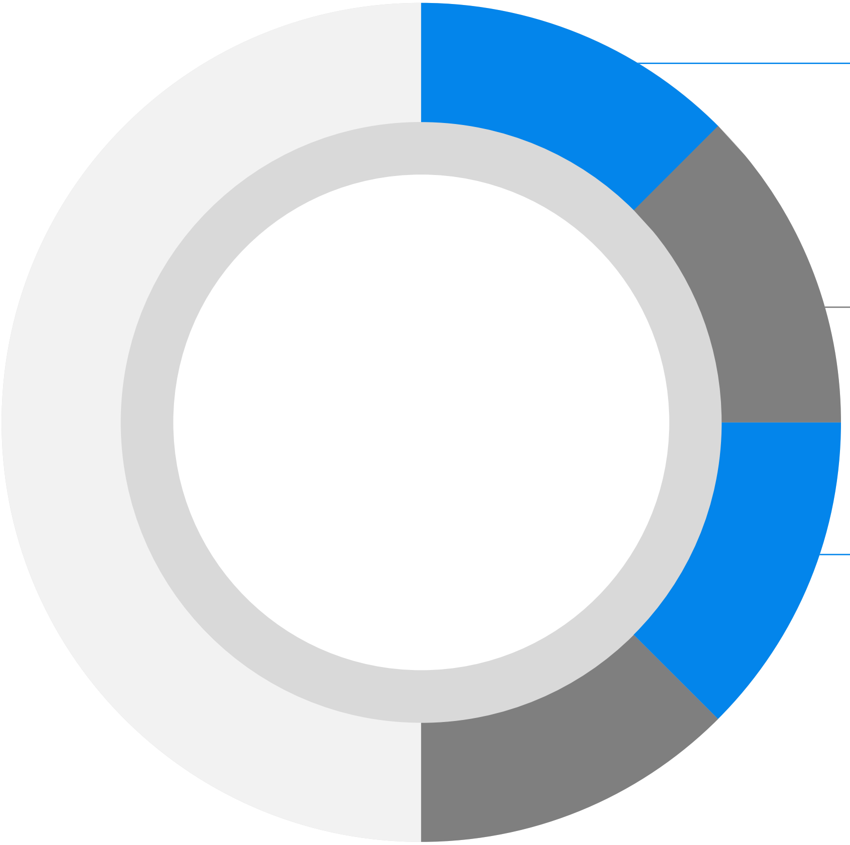
《有监督图学习》

主讲： CV 彭老师

https://www.julyedu.com/

七月在线
JULYEDU.COM



图采样聚合网络

GraphSAGE

注意力机制

Attention Mechanism

图注意力网络

Graph Attention Network

/01 GraphSAGE

图采样聚合网络

Inductive vs. Transductive Learning

Induction is reasoning from observed training cases to general rules, which are then applied to the test cases.

Transduction is reasoning from observed, specific (training) cases to specific (test) cases.

Inductive Learning

Inductive learning is the same as what we commonly know as traditional supervised learning.

We build and train a machine learning model based on a labelled training dataset we already have. Then we use this trained model to predict the labels of a testing dataset which we have never encountered before.

Transductive Learning

Transductive learning techniques have observed all the data beforehand, both the training and testing datasets.

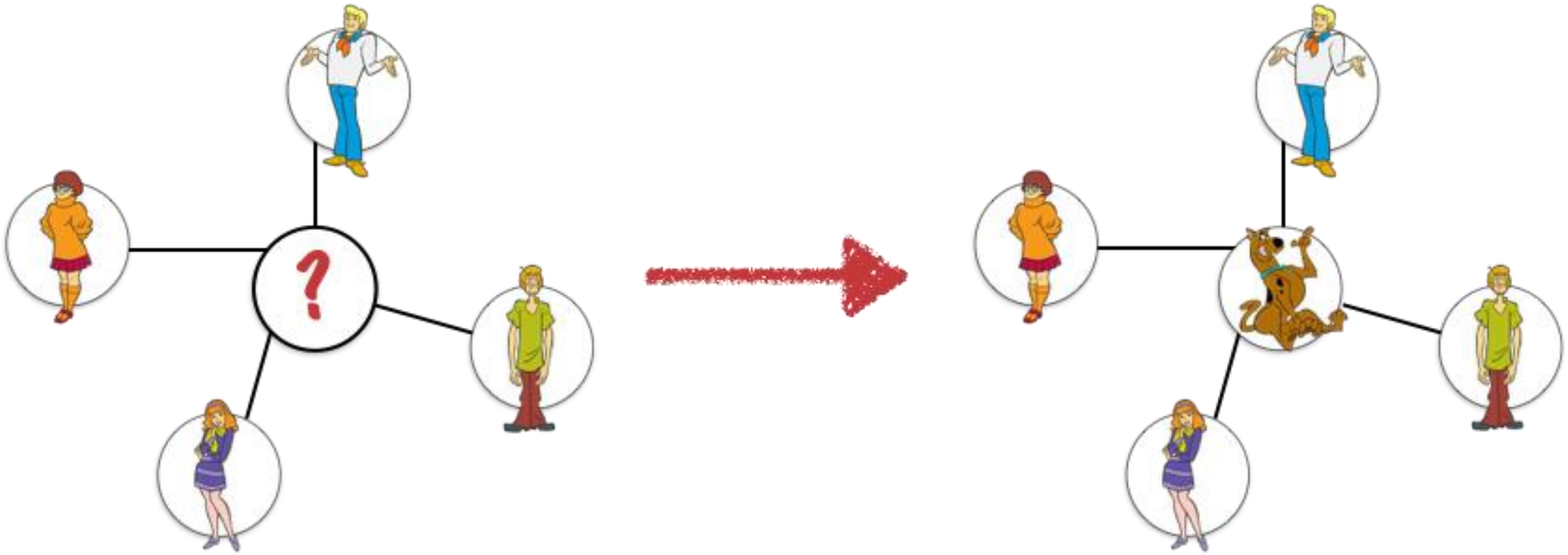
We learn from the already observed training dataset and then predict the labels of the testing dataset.

Even though we do not know the labels of the testing datasets, we can make use of the patterns and additional information present in this data during the learning process.

GraphSAGE

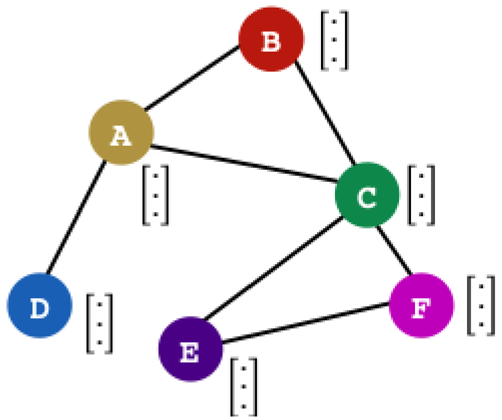
GraphSAGE is an iterative algorithm that learns graph embeddings for every node in a certain graph.

The novelty of GraphSAGE is that it was the *first* work to create inductive node embeddings in an unsupervised manner!



GraphSAGE

The **goal** of GraphSAGE is to learn a **representation** for every node based on some combination of its *neighboring* nodes.



$X_v =$ Node features for a random node v

$h_v^0 =$ Initial node embedding representation for a random node v

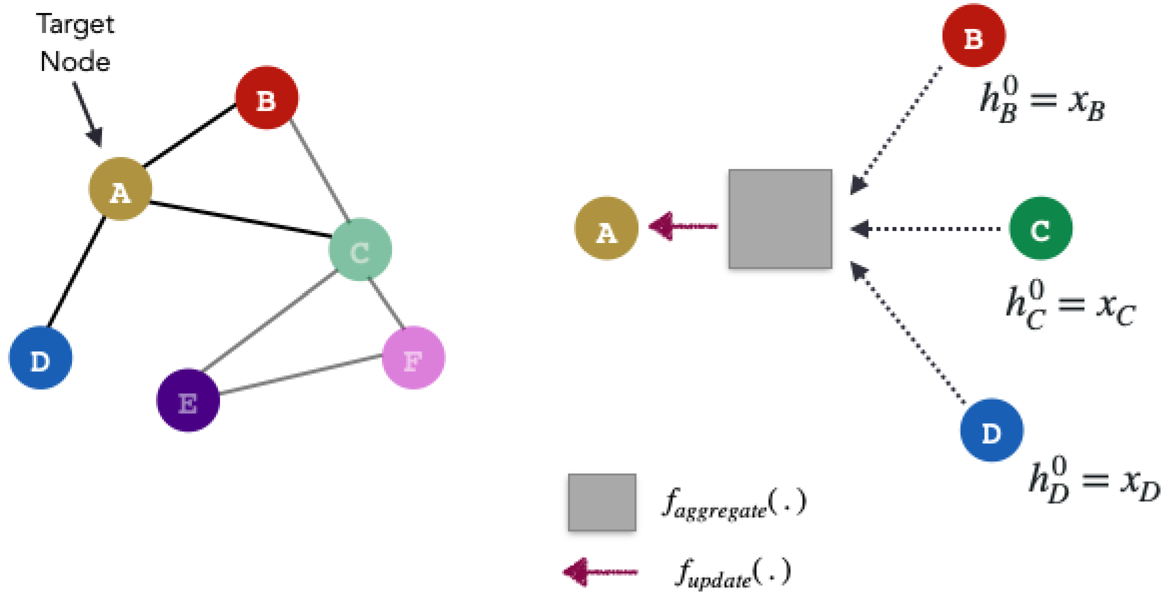
$h_v^k =$ Node embedding representation for a random node v at the k-th iteration

$z_v =$ Final node embedding representation for a random node v after a round of GraphSAGE

GraphSAGE

Since every node can be defined by their neighbours, the embedding for node A can be represented by some combination of its neighbouring node embedding vectors.

Through one round of the GraphSAGE algorithm, we will obtain a new representation for node A.



GraphSAGE

Two steps:

- ✓ Aggregate:

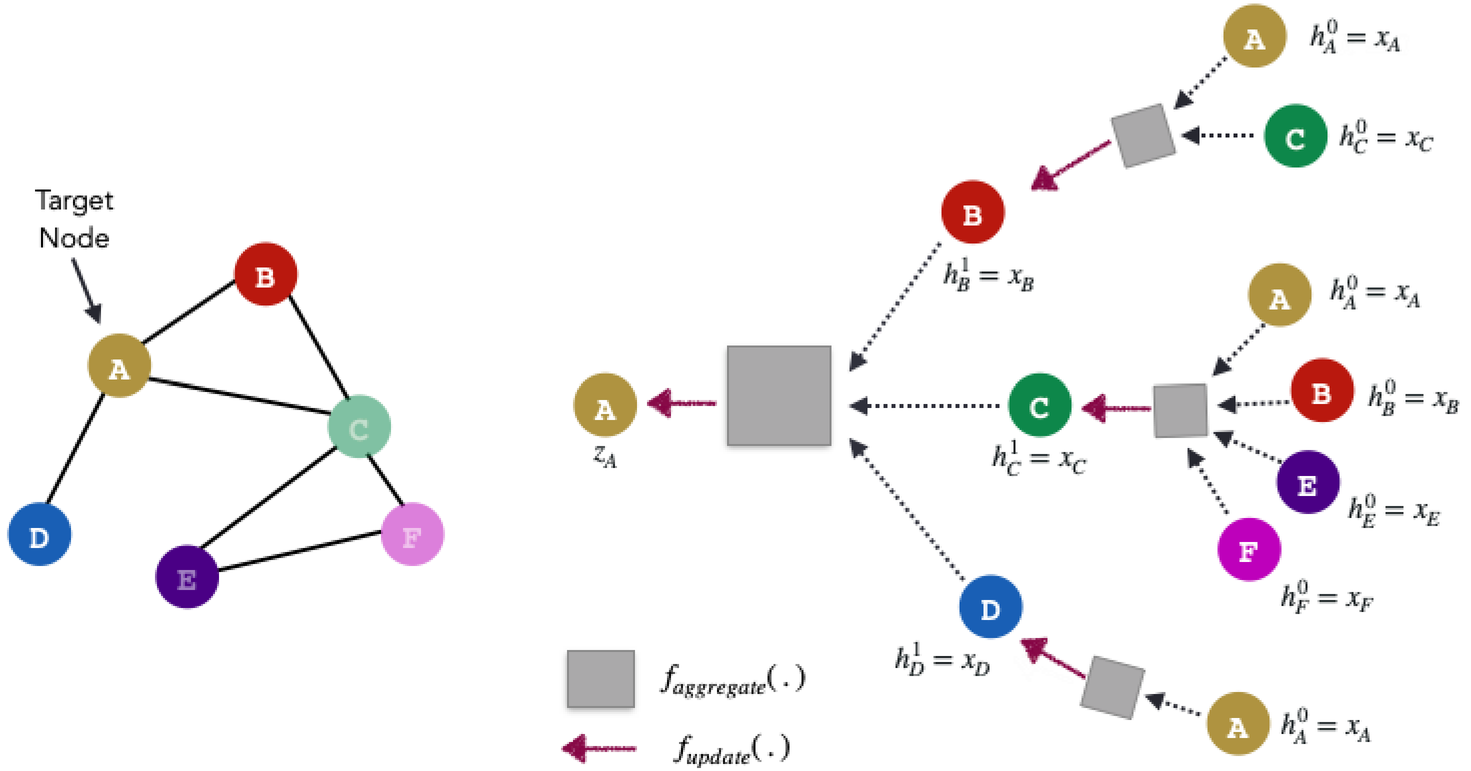
$$a_v = f_{aggregate}(\{h_u \mid u \in N(v)\})$$

- ✓ Update:

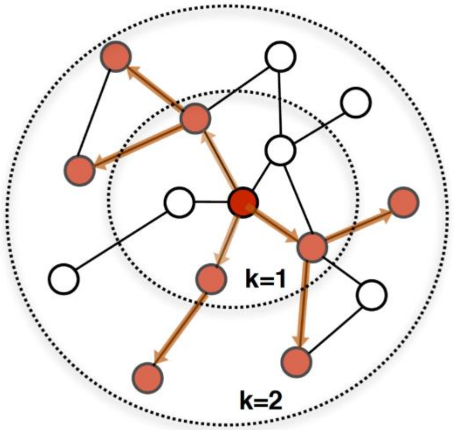
$$h_v^k = f_{update}(a_v, h_v^{k-1})$$

K-Hop

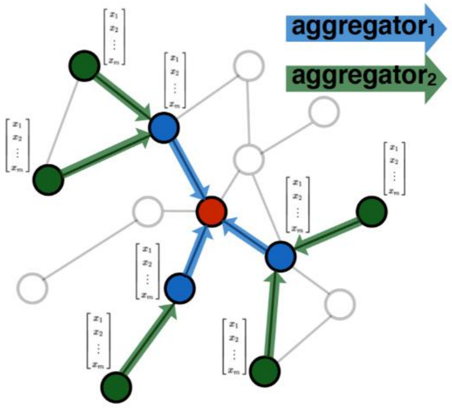
The k-parameter tells the algorithms *how many* neighborhoods or *how many* hops to use to <u>compute</u> the representation for node v.



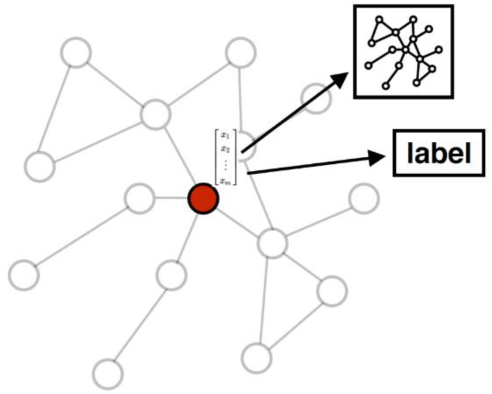
GraphSAGE



1. Sample neighborhood



2. Aggregate feature information from neighbors



3. Predict graph context and label using aggregated information

GraphSAGE

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \to 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$; Set the initial node embedding to be the node's feature vector

for $k = 1...K$ **do** Iterate over all the k-hop neighbourhoods

  for $v \in \mathcal{V}$ **do** Iterate through all the nodes in the graph

    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;

    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$

    Run the aggregate-update cycle for every node in this k-th iteration

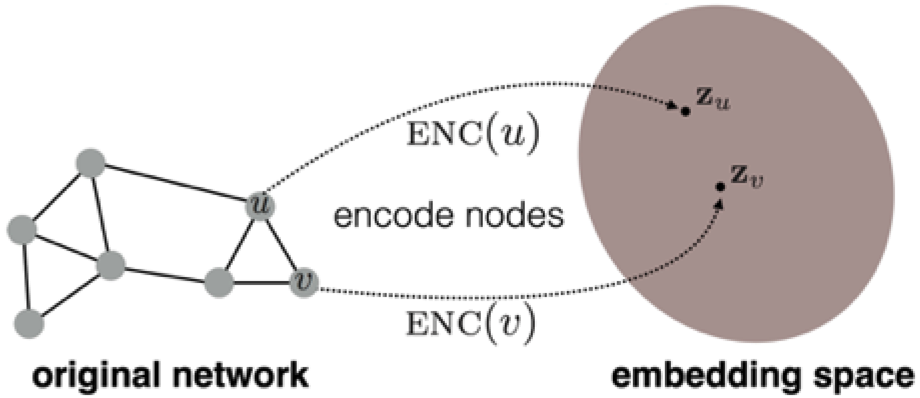
  end

  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$ Normalize the node embedding

end

$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Loss Function



$$\ell(z_v) = -\log(\sigma(z_u^T z_v) + \epsilon) - Q * \mathbb{E}_{v_n \sim P_n(v)} \log(\sigma(-z_u^T z_v) + \epsilon)$$

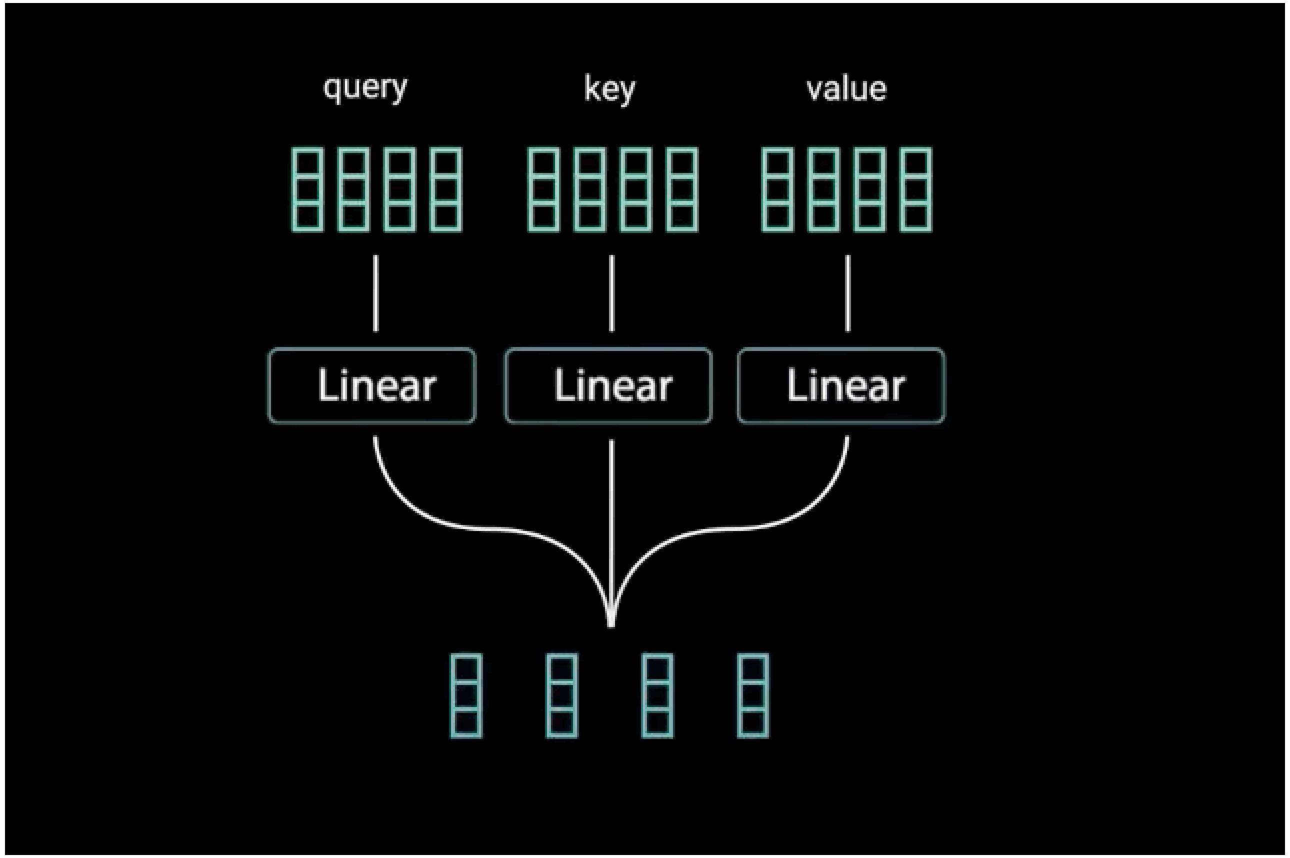
/02 Attention Mechanism

注意力机制

Visual Attention

Text Attention

Query, Key, & Value



Visual Attention

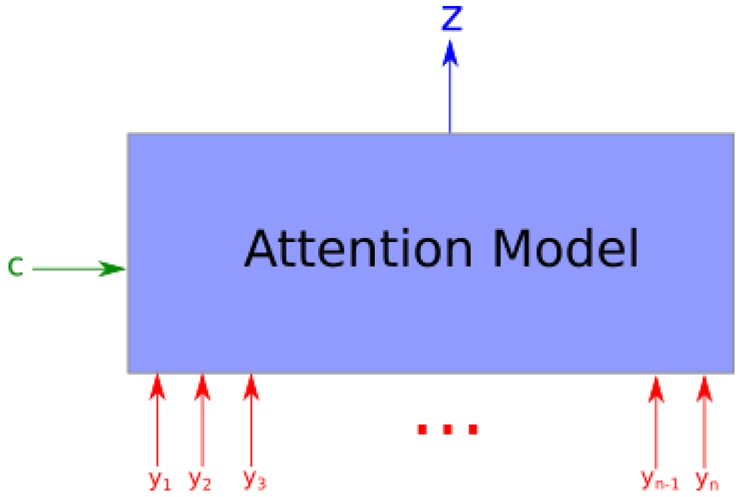
Visual Attention: We focus on **specific parts** of their visual inputs to compute adequate responses.

This principle has a large impact on neural computation as we need to select the most pertinent piece of information, rather than using all available information, a large part of it being irrelevant to compute the neural response.

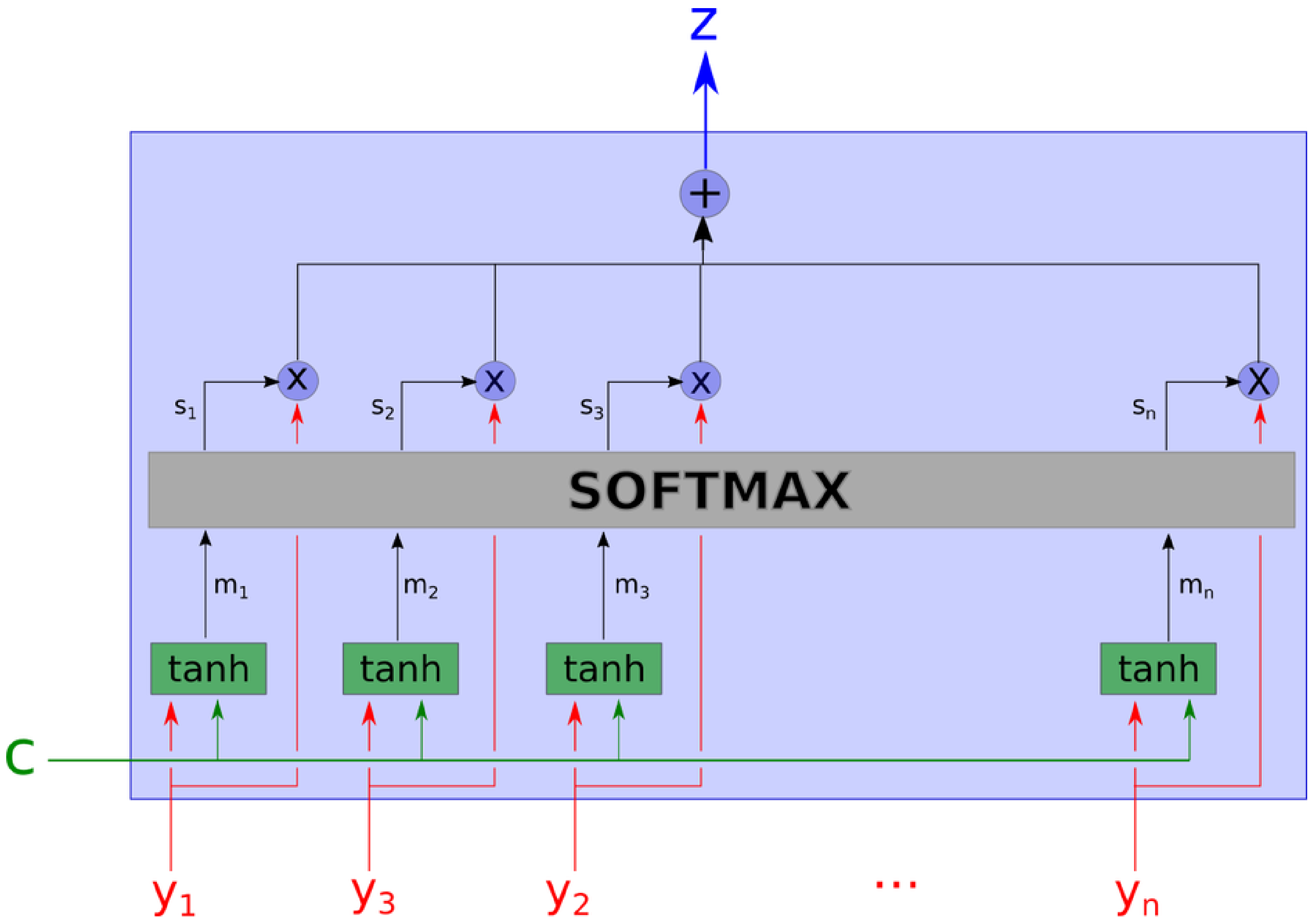
Attention: focusing on specific parts of the input - has been applied in Deep Learning, for speech recognition, translation, reasoning, and visual identification of objects.

Attention Module

An attention model is a method that takes n arguments $y_1, \ldots y_n$, and a context c. It return a vector z which is supposed to be the “summary” of the y_i, focusing on information linked to the context c.

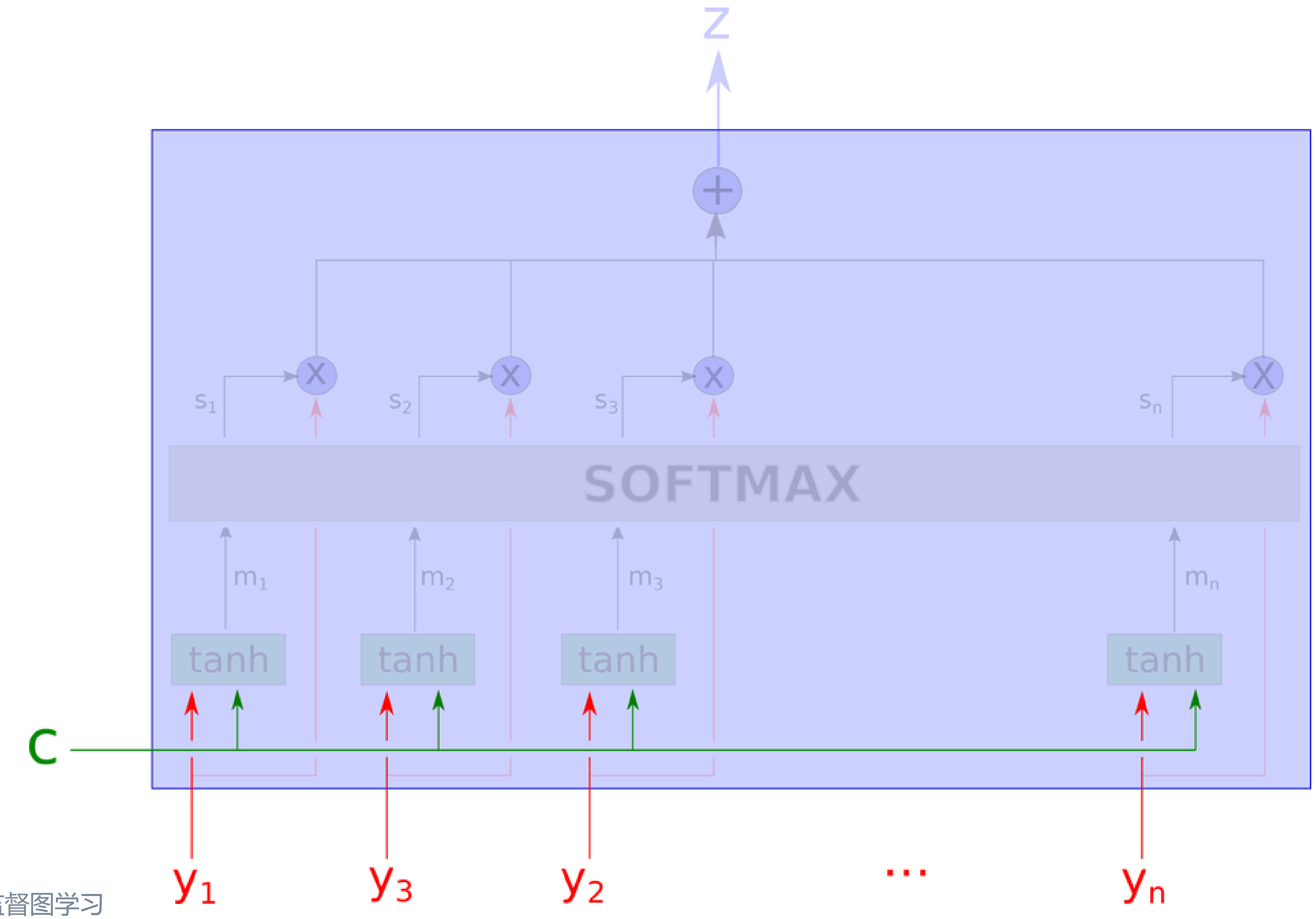


Attention Module



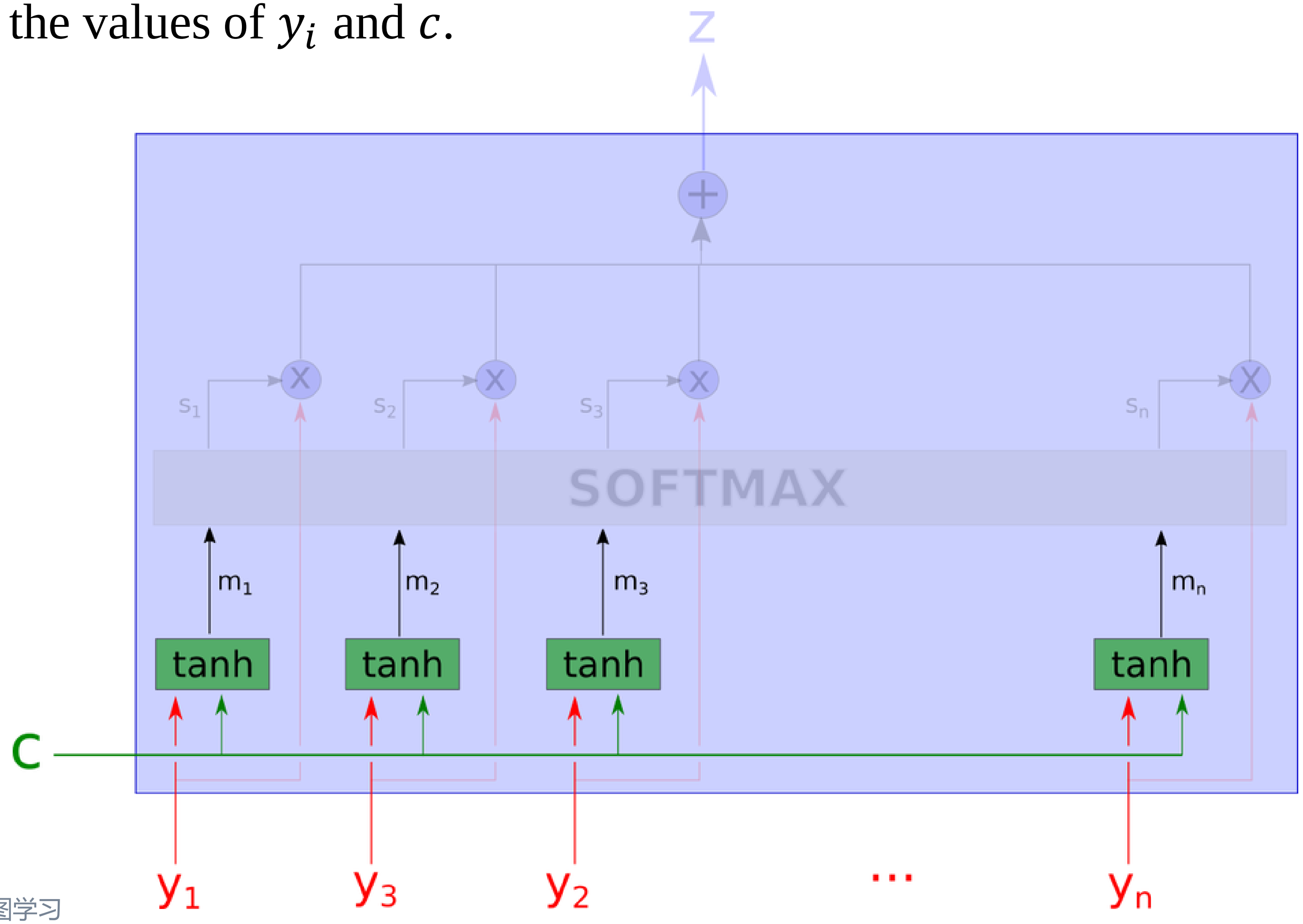
Attention Module

First, we recognize the input c is the context, and the y_i are the "part of the data" we are looking at.



Attention Module

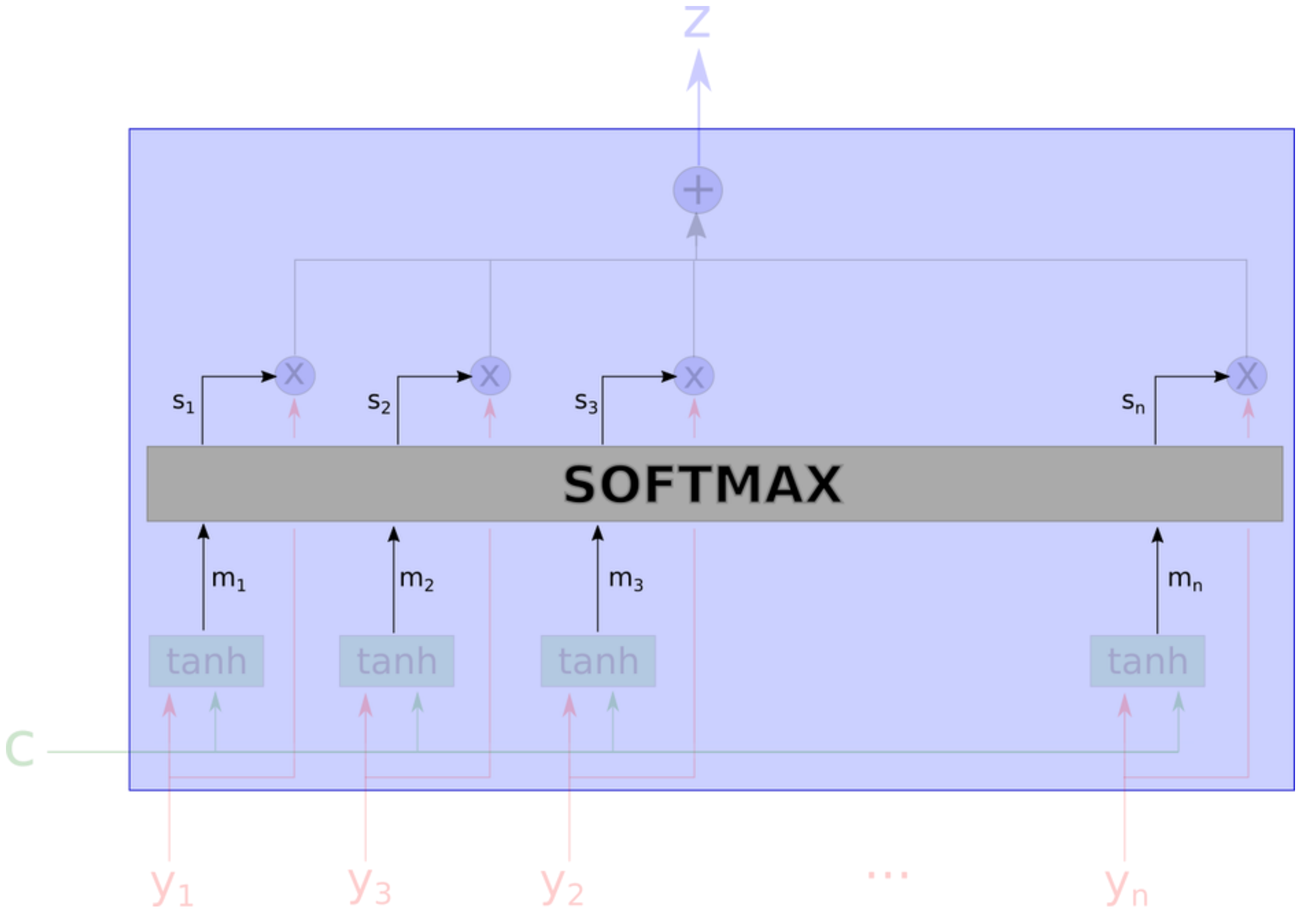
At the next step, the network computes $m_1, \ldots m_n$ with a $tanh$ layer. It means that we compute an "aggregation" of the values of y_i and c.

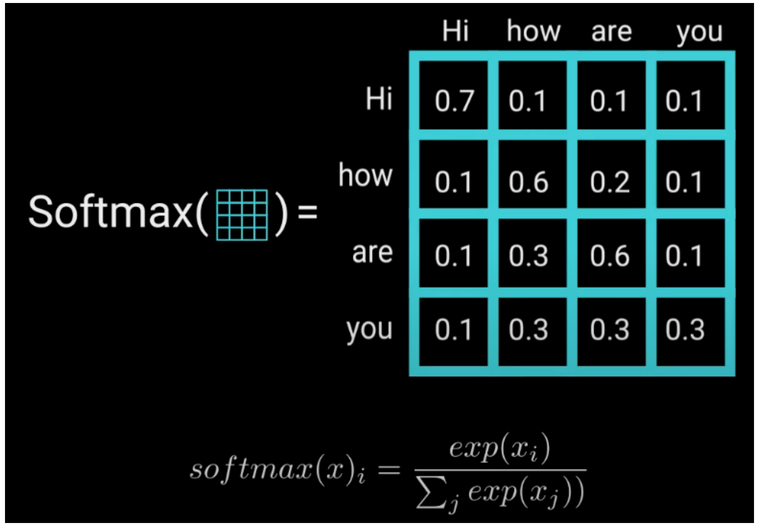


$$m_i = \tanh\left(W_{cm}c + W_{ym}y_i\right)$$

Attention Module

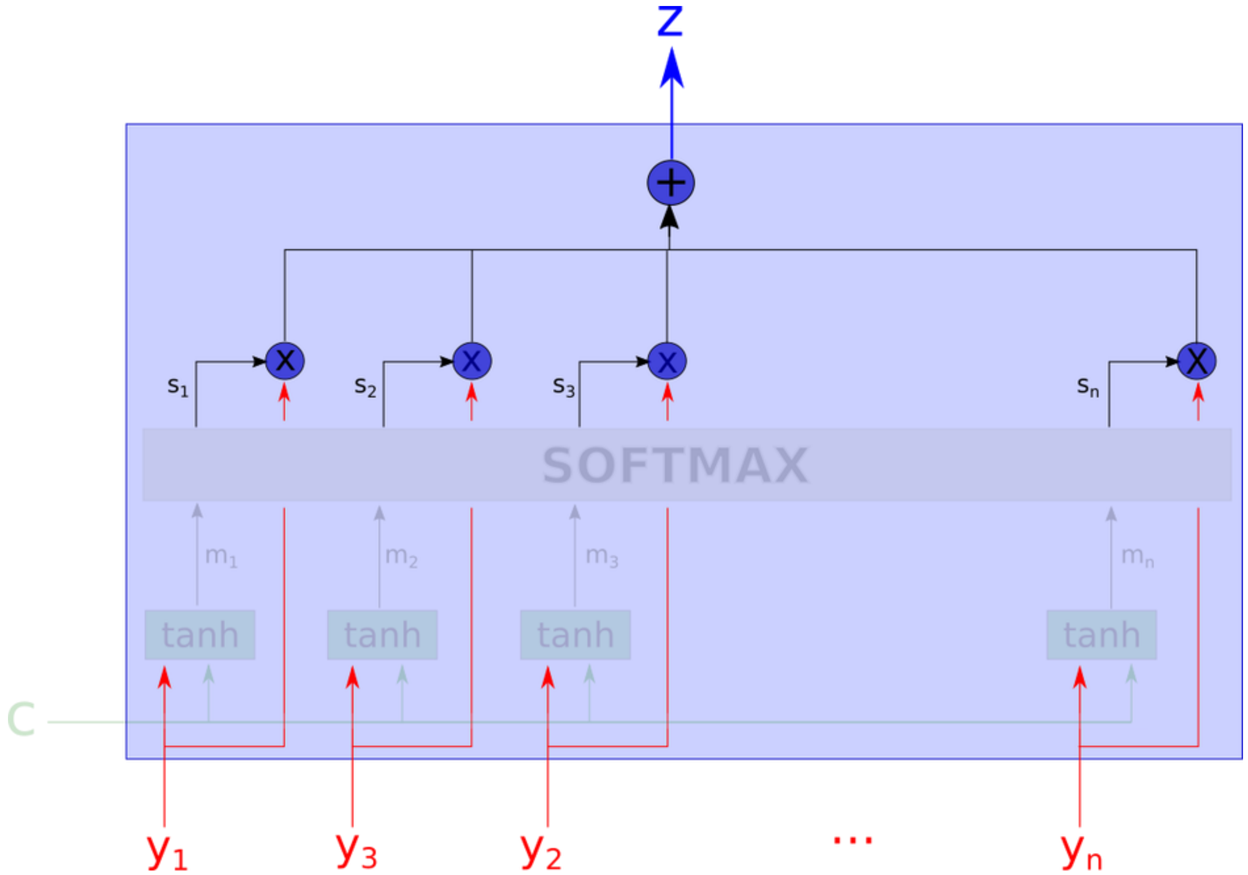
$$softmax(z_1, \ldots, z_n)$$

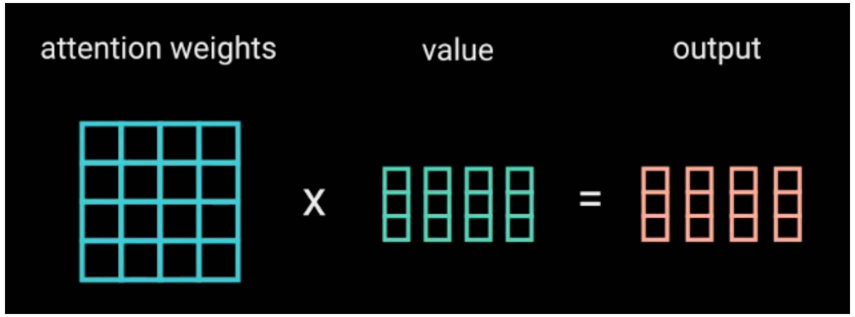




Attention Module

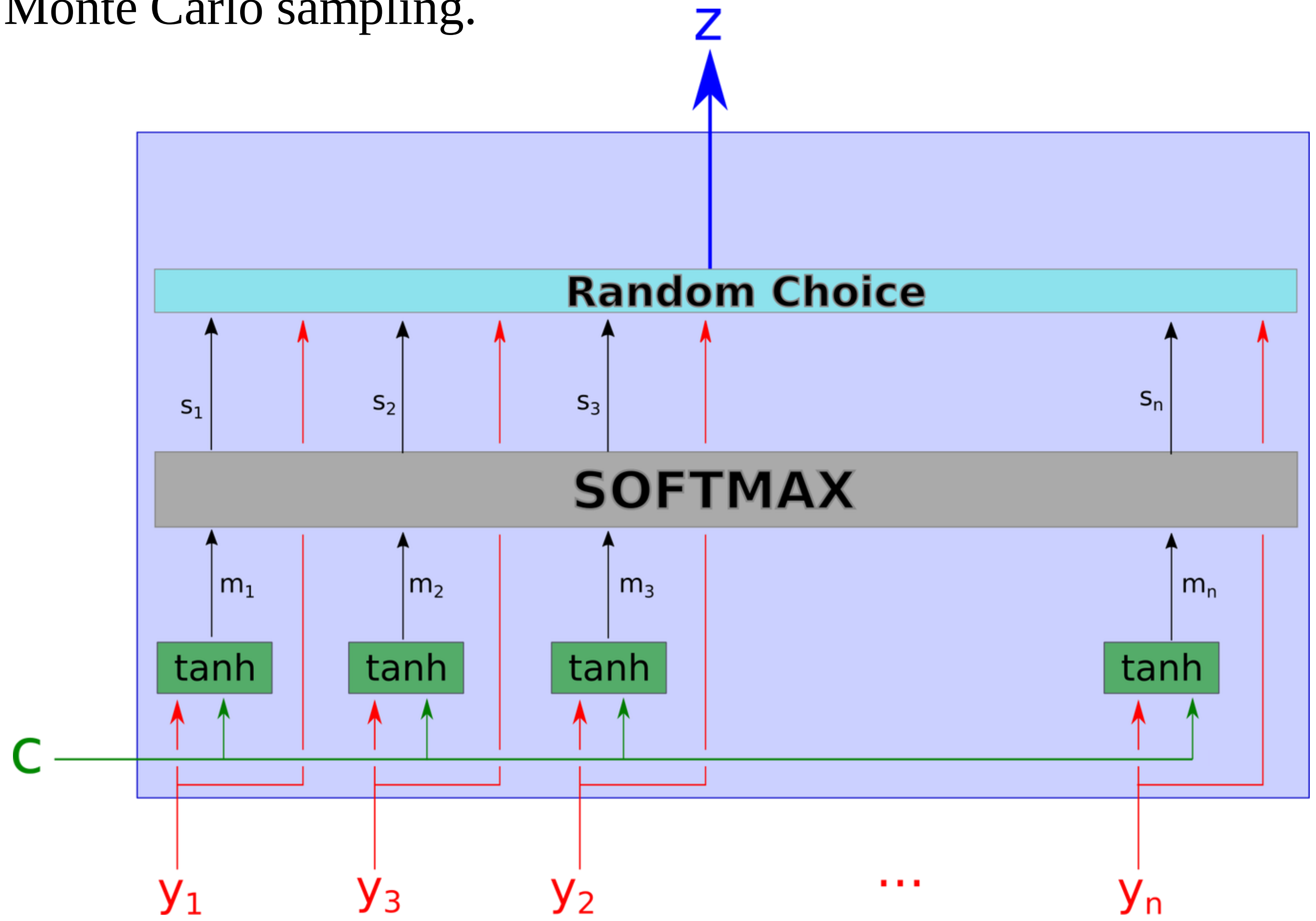
The output z is the weighted arithmetic mean of all the y_i, where the weight represents the relevance for each variable according to the context c.





Hard Attention Module

Stochastic Process: Instead of using all the hidden states as an input for the decoding, the system samples a hidden state y_i with the **probabilities** s_i. In order to propagate a gradient through this process, we estimate the gradient by Monte Carlo sampling.



/03 Graph Attention Network

图注意力网络

Graph Attention Network

The Graph Attention Network (GAT) is a non-spectral learning method which utilizes the spatial information of the node directly for learning.

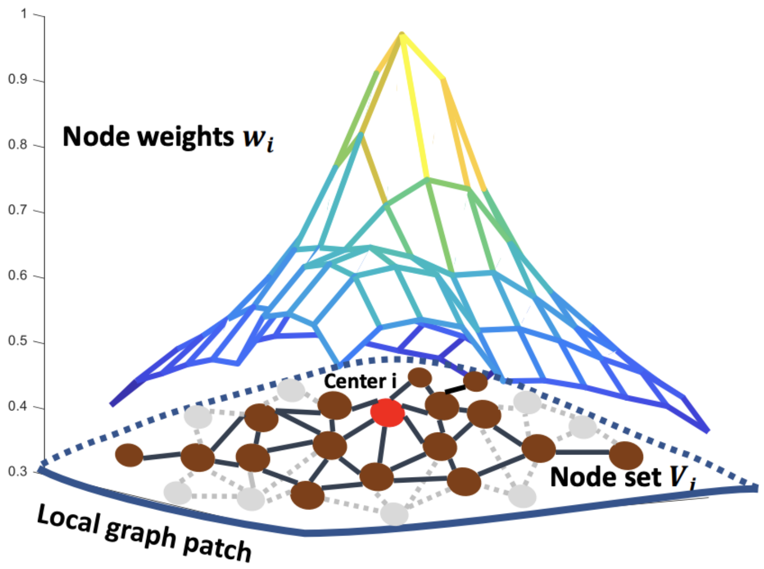
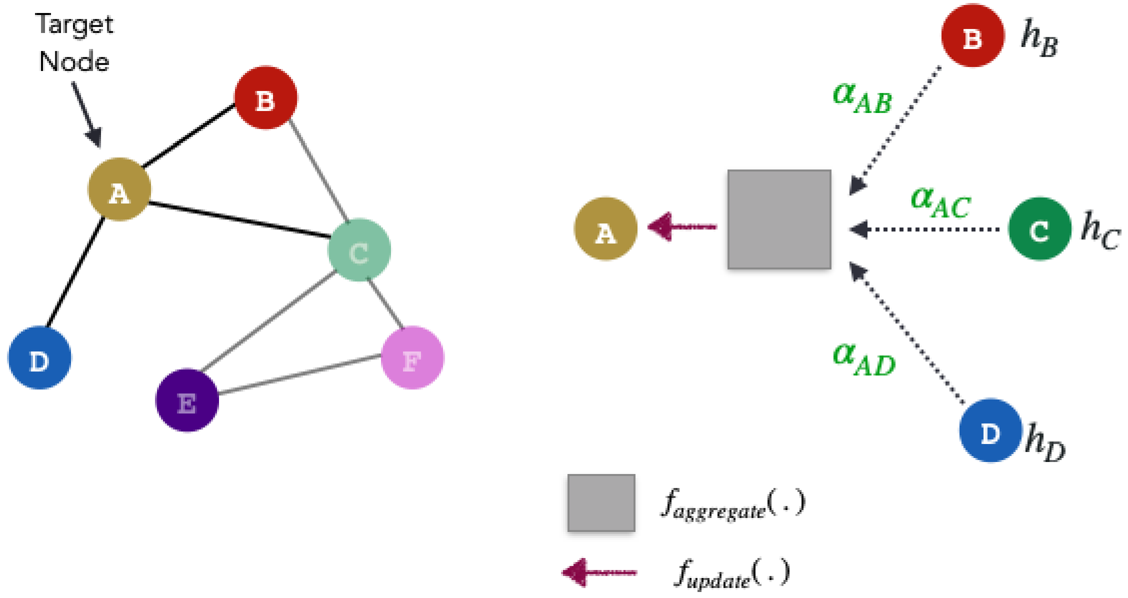


Figure 2: Structural fingerprint for a given node i, denoted by $F_i = (V_i, \mathbf{w}_i)$, where V_i is the set of nodes in this local receptive field and $\mathbf{w}_i$ is the contributing weights of the nodes.

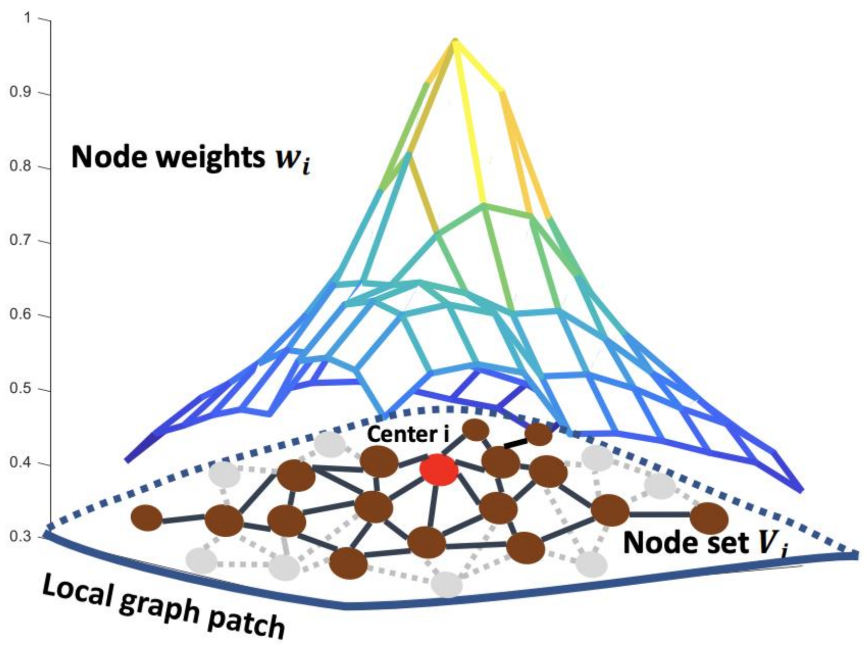
Graph Attention Network

GAT layer only focus on obtaining a node representation based on the immediate neighbors of the target node.



Graph Attention Layer

Step 1: Linear Transformation;
Step 2: Computation of Attention Coefficients;
Step 3: Normalization of Attention Coefficients;
Step 4: Computation of Final Output Features;
Step 5: Computation of Multi-Head Attention Mechanisms.



Step 1: Linear Transformation

The first step performed by the Graph Attention Layer is to apply a linear transformation：

- ❖ Weighted matrix W to the feature vectors of the nodes.

$$W h_i$$

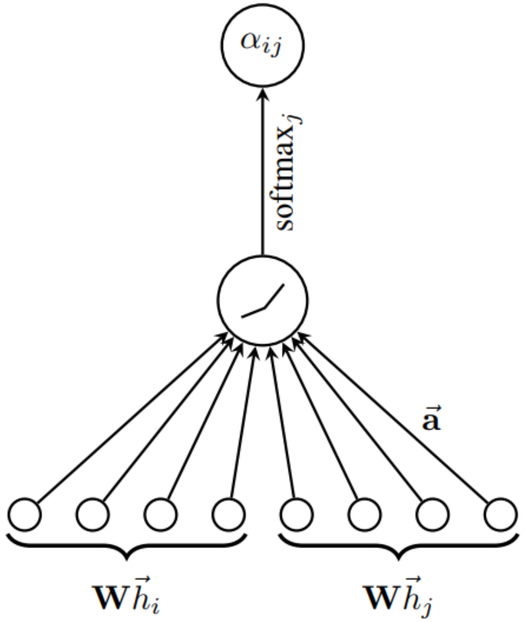
Step 2: Computation of Attention Coefficients

Let us perform self-attention on the nodes—a *shared attentional mechanism* $\boldsymbol{a}$ to compute attention coefficients.

Attention Coefficients determine the importance of node i's features to node j.

In its most general formulation, the model allows every node to attend on every other node, *dropping all structural information*.

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$



Step 3: Normalization of Attention Coefficients

Due to the varied structure of graphs, nodes can have a different number of neighbors.

To have a common scaling across all neighborhoods, the Attention coefficients are Normalized.

$$\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$$

$$\alpha_{ij} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T [\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T [\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

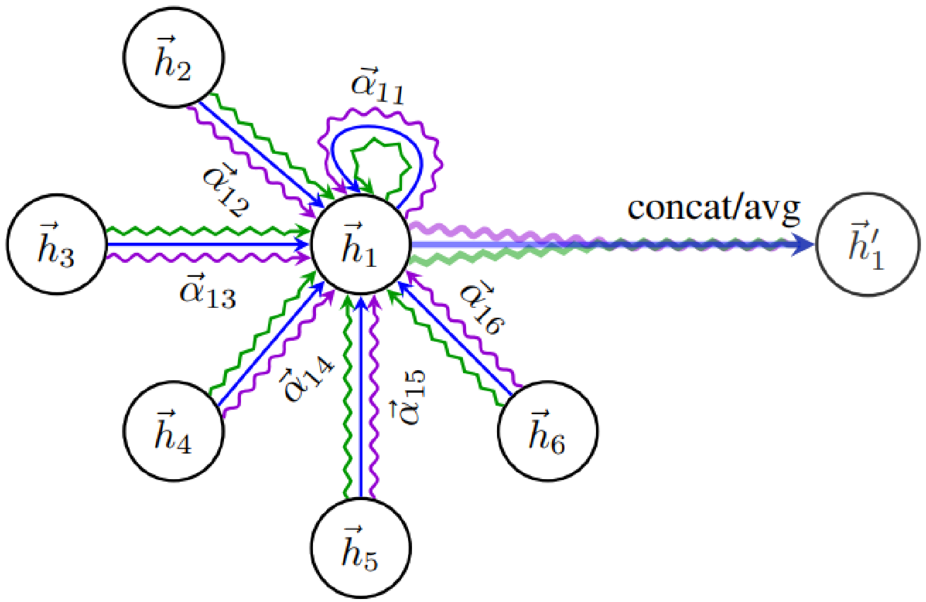
Step 4: Computation of Final Output Features

Now we compute the learned features of nodes. σ is a Non-Linear Transformation.

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}\mathbf{W}\vec{h}_j\right)$$

Step 5: Computation of Multi-Head Attention Mechanisms

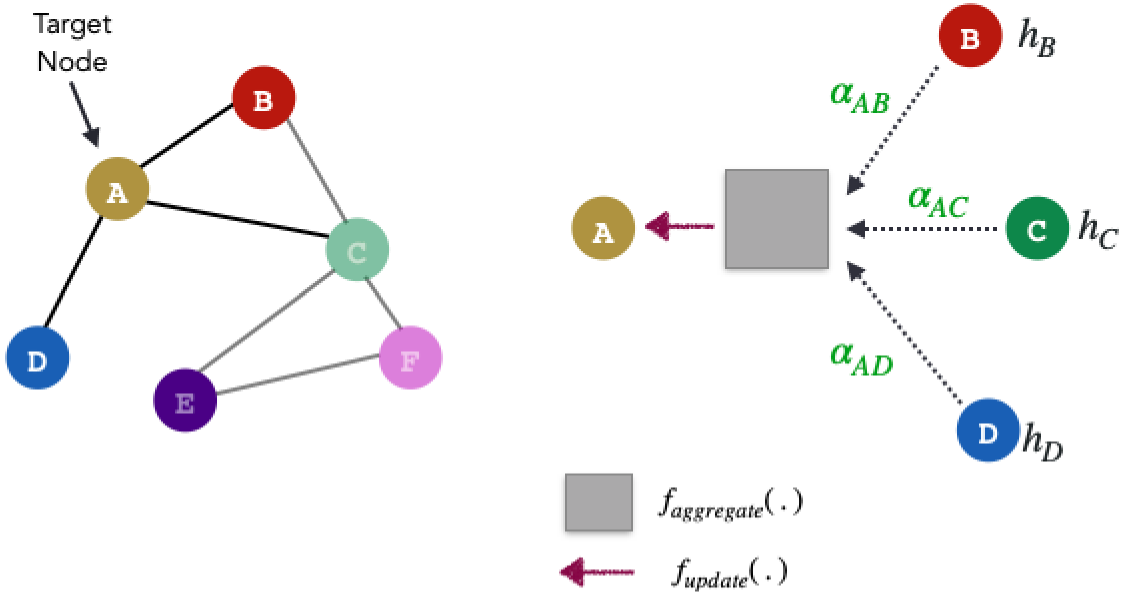
The multi-head attention is utilized to be beneficial to stabilize the learning process of self-attention.

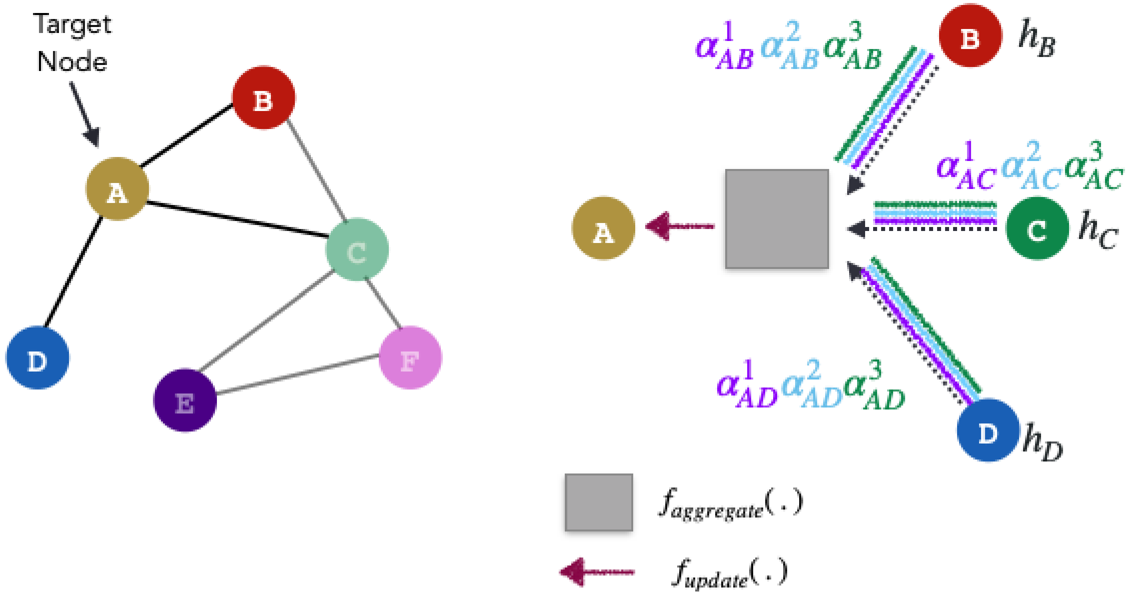


$$\vec{h}'_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$

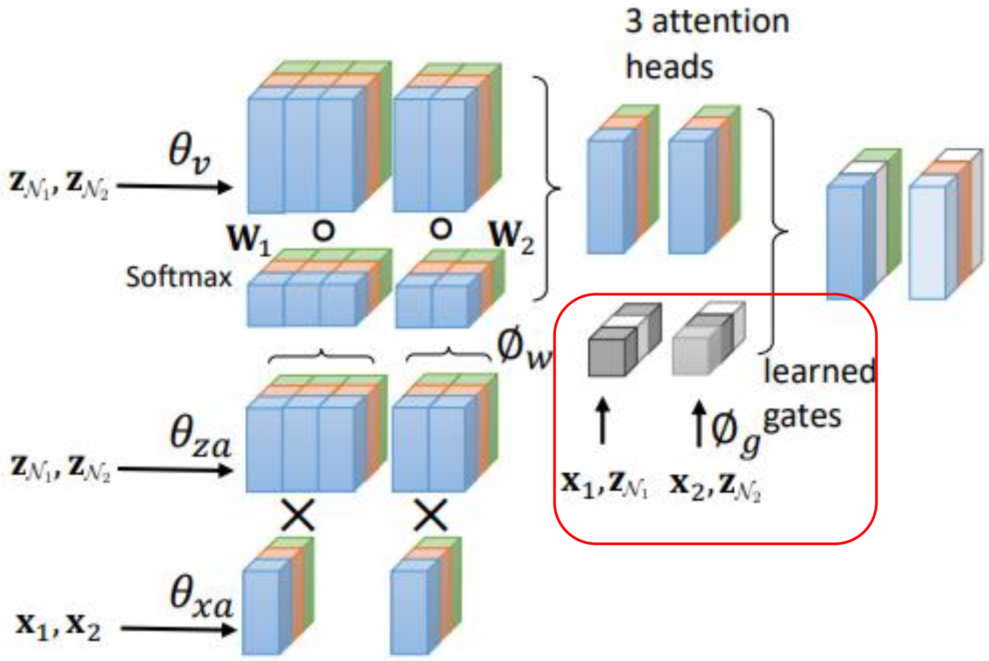
Graph Attention Network





Gated Graph Attention Network

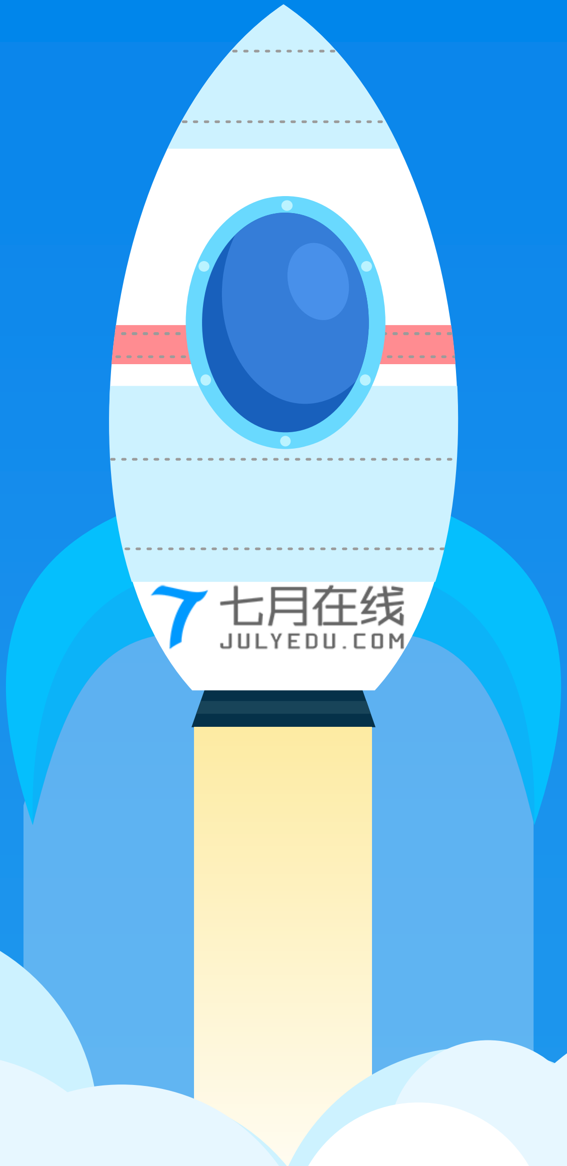
GaAN acts as a high-level controller that determines how to aggregate the features extracted by the attention heads.



CV就业小班

https://www.julyedu.com/Employment/cv12?from=pcbanner

微信扫一扫关注我们



THANKS

https://www.julyedu.com