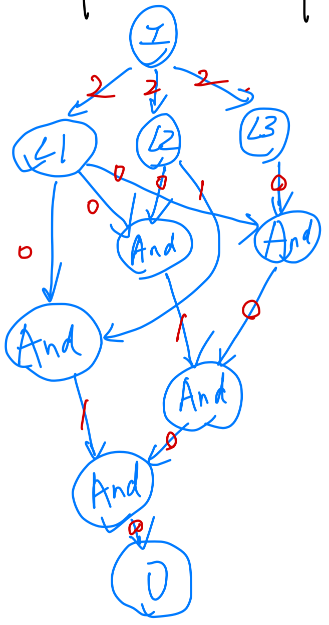
Type 1: And gate as vertices (node)

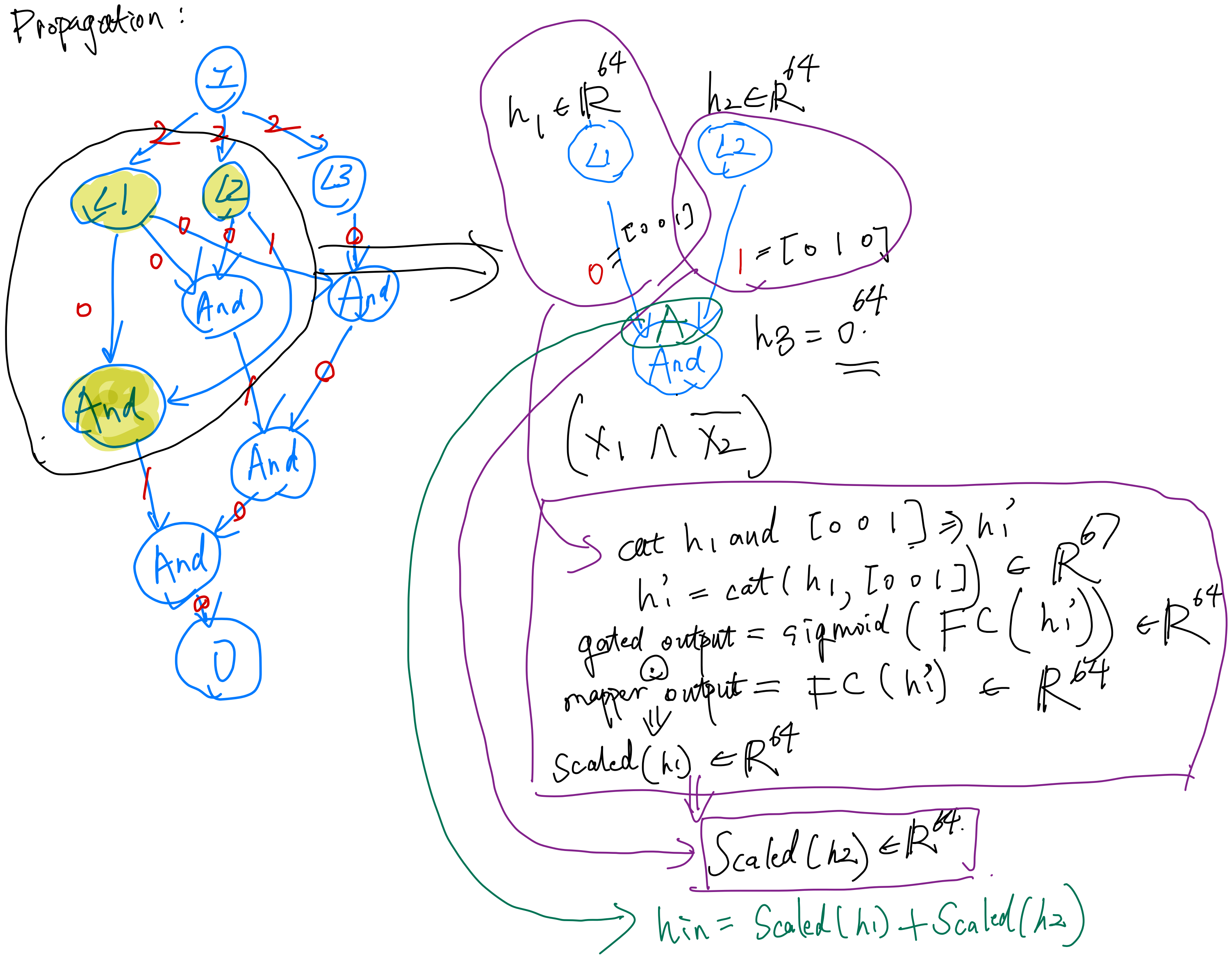
Invertor gate as Edge.

Example: one expression of SR3



		One-hot encoding
Node:	1. Input (Virtual single input)	[0 0 0 1]
	2. Literal (# used variables)	[0 0 1 0]
	3. And gate	[0 1 0 0]
	4. Output	[1 0 0 0]
Edge:	0: straight-through	[0 0 1]
	1: invertor	[0 1 0]
	2. virtual edge (from Input node to Literal node)	[1 0 0]

Propagation:


I
L1
L2
L3
And
O
0
1
2
$h_1 \in \mathbb{R}^{64}$
$h_2 \in \mathbb{R}^{64}$
$0 = [0\ 0\ 1]$
$1 = [0\ 1\ 0]$
$h_3 = 0^{64}$
$(x_1 \wedge \overline{x_2})$

cat h_1 and $[0\ 0\ 1] \Rightarrow h_1'$

$h_1' = \text{cat}(h_1, [0\ 0\ 1]) \in \mathbb{R}^{67}$

gated output $= \text{sigmoid}(FC(h_1')) \in \mathbb{R}^{64}$

$\odot$

mapper output $= FC(h_1') \in \mathbb{R}^{64}$

$\Downarrow$

$\text{Scaled}(h_1) \in \mathbb{R}^{64}$

$\text{Scaled}(h_2) \in \mathbb{R}^{64}$

$h_{in} = \text{Scaled}(h_1) + \text{Scaled}(h_2)$

$h_3 = GRU(x_3, h_{in})$

$[0\ 1\ 0\ 0]$ Obtained before.

$\Downarrow$

Updated the hidden state of AND Gate.

Reasons : Invertor :

$gated(h_i) = sigmoid(FC(h_1, [0\ 1\ 0]))$

Non-Invertor :

$gated(h_i) = sigmoid(FC(h_1, [0\ 0\ 1]))$

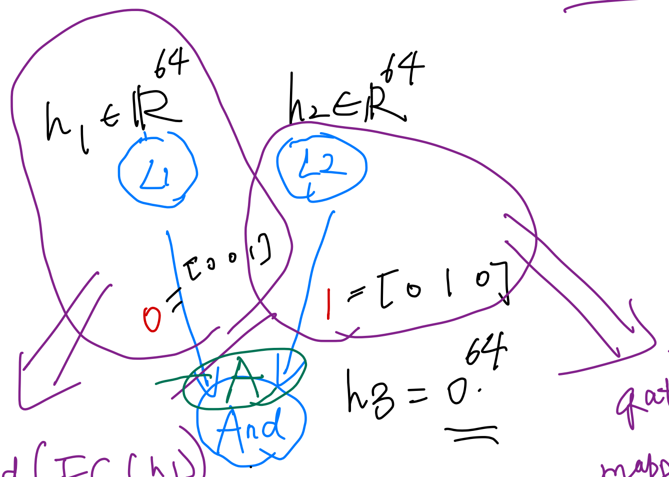
Such one-hot vector can act as Indicator.

Make the hidden state of previous nodes act differently.

Same idea : C-VAE. the class information is also encoded as one-hot vector and concatenated with hidden states.

Type 2 : And gate as vertices (node)

Invertor gate as Edge. just Negate the hidden state.



$h_3 = 0^{64}$.

$gated(h_1) = sigmoid(FC(h_1))$

$mapper(h_1) = FC(h_1)$

Invertor :

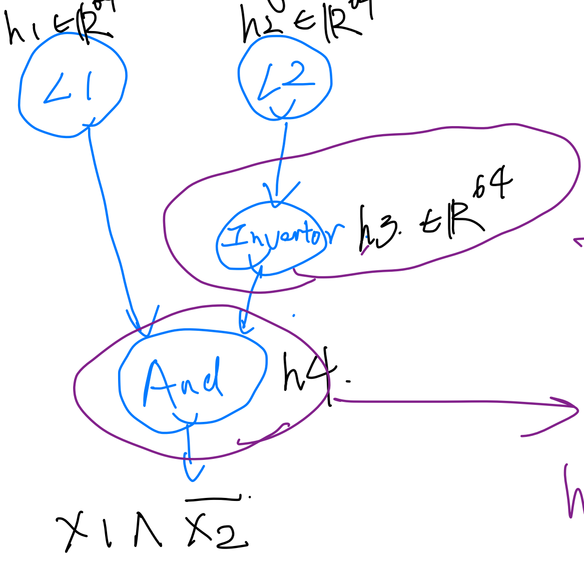
$gate(h_2) = sigmoid(FC(-h_2))$

$mapper(h_2) = FC(-h_2)$

Type 3 :

And gate

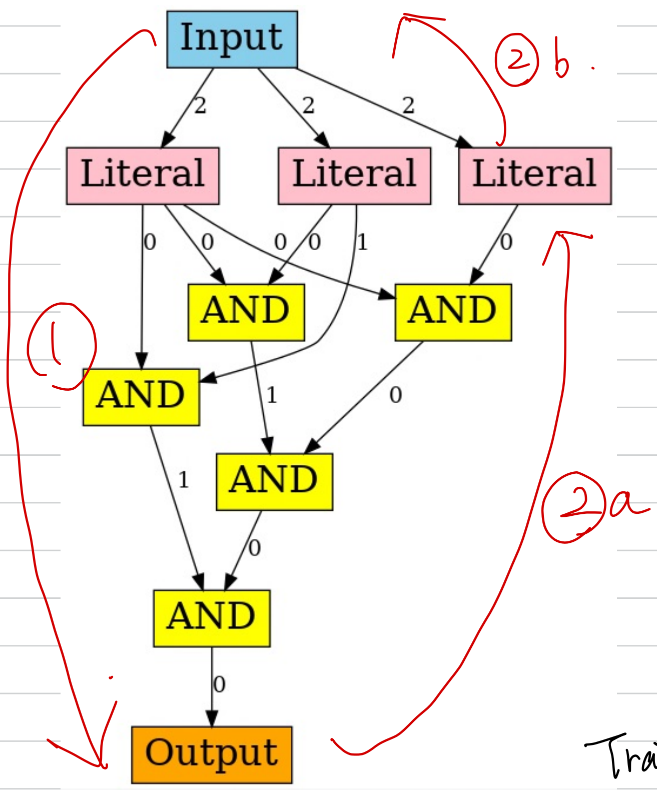
Inverter gate as vertices .


$h_1 \in \mathbb{R}^{64}$
$h_2 \in \mathbb{R}^{64}$
L1
L2
Inverter
$h_3 \in \mathbb{R}^{64}$
And
h4.
$x_1 \wedge \overline{x_2}$

$h_3 = \mathrm{GRU}(x_3, h_2)$

$h_4 = \mathrm{GRU}(x_4, h_4^{in})$

$h_4^{in} = \sum_{i=1,3} \mathrm{gated}(h_i) \odot \mathrm{mapper}(h_i)$



Training: (2 stages)

① : Forwarding Input → Output

Simulate logic computation.

Get the embedding of Output. $\vec{e}_{output}$.

② : Backwarding Output → Literal → Input.

Simulate Backtracking

②a. Get the embedding of Literal $\vec{e}_{Literal}$

②b. Get the embedding of Input $\vec{e}_{input}$

Training signals:

Graph level: SAT/UNSAT

Node Level:

If the graph is SAT.

The solution $\{0,1\}^N$. for Literal Nodes

How to train the network?

$$\vec{e}_{output} = GNN_{forwarding}(g)$$

$$(\vec{e}_{Input}, \vec{e}_{Literal}) = GNN\ backwarding(g, \vec{e}_{output})$$

Graph-Classification:

$$\ell_1 = BinaryCrossEntropyLoss[FC(cat(\vec{e}_{output}, \vec{e}_{input})), SAT/UNSAT]$$

$$\ell_2 = \begin{cases} BinaryCrossEntropyLoss[FC(\{\vec{e}_{Literal}\}), Solution], & \text{if } g \text{ is SAT.} \\ 0, & \text{otherwise.} \end{cases}$$